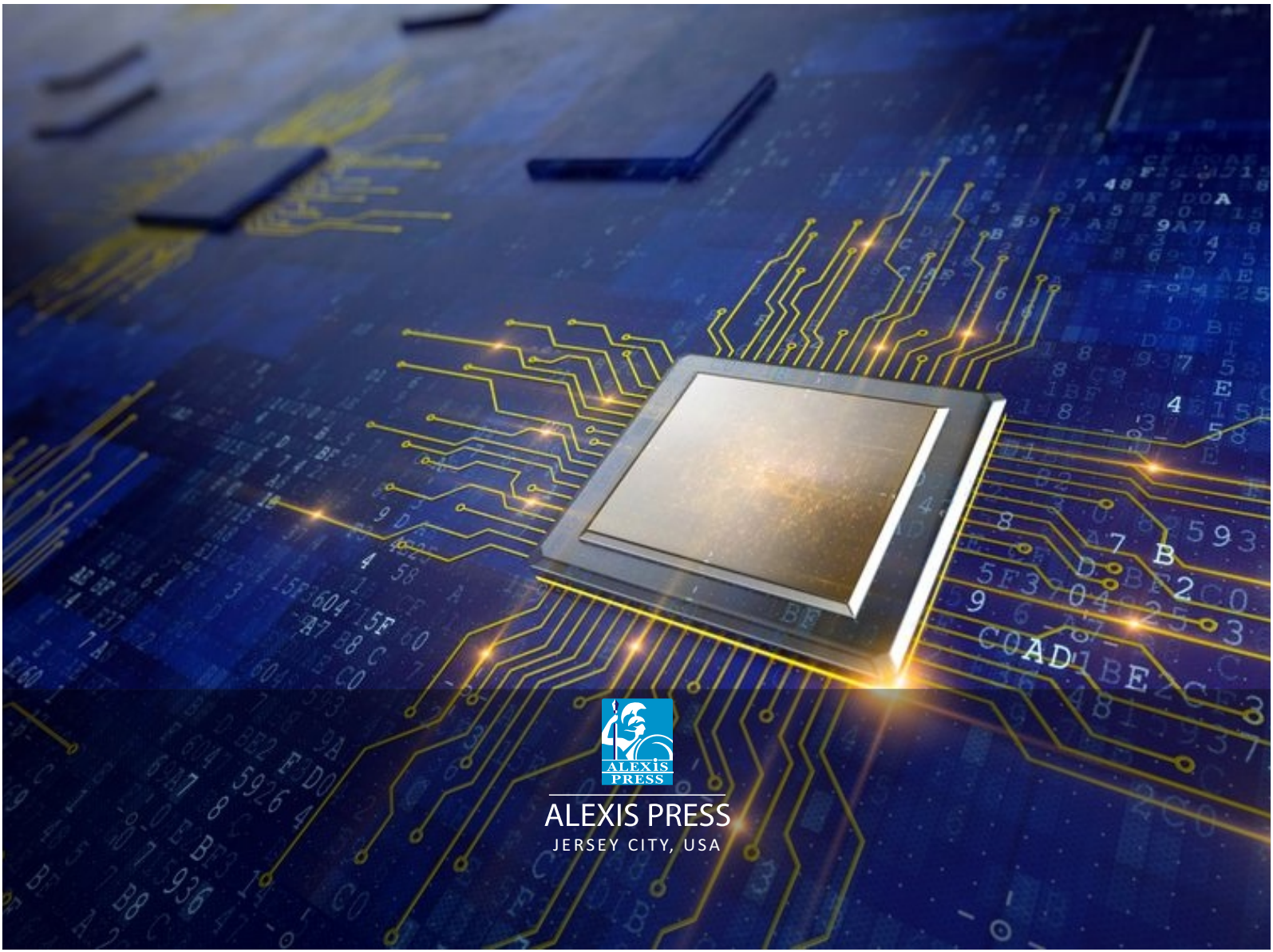
Dr. T.K. Thivakaran
Mohan Kumar A V

FUNDAMENTALS OF MICROCONTROLLER



ALEXIS PRESS
JERSEY CITY, USA

FUNDAMENTALS OF MICROCONTROLLER

FUNDAMENTALS OF MICROCONTROLLER

Dr. T.K. Thivakaran

Mohan Kumar A V

ALEXIS PRESS

ALEXIS PRESS

Published by: Alexis Press, LLC, Jersey City, USA
www.alexispress.us

First Published 2022

A catalogue record for this publication is available from the British Library

Library of Congress Cataloguing in Publication Data

Includes bibliographical references and index.

Fundamentals of Microcontroller by *Dr. T.K. Thivakaran, Mohan Kumar A V*

ISBN 978-1-64532-890-2

CONTENTS

CHAPTER 1

AN INTRODUCTION TO MICROPROCESSOR AND MICROCOMPUTERARCHITECTURE

Dr.T.K.Thivakaran, Professor,
Department of Computer Science and Engineering,
Presidency University, Bangalore, Karnataka, India
Email Id- thivakaran@presidencyuniversity.in

ABSTRACT:

A central processor unit (CPU), a memory unit, and an input/output (I/O) unit are the three fundamental building ingredients of a microcomputer. The CPU processes all of the directions and applies logic and arithmetic to the data. Several registers to store data, an arithmetic logic unit (ALU) to conduct arithmetic and logical operations, instruction decoders, counters, and control lines make up the microprocessor unit of a computer. The CPU executes the activities listed by reading instructions from memory.

KEYWORDS:

Central Processor Unit, Electronics, Microcontroller, Microprocessor, Software.

INTRODUCTION

A microprocessor is a programmable electrical chip with processing and decision-making abilities comparable to those of a computer's central processing unit. Microcomputers are any microprocessor-based systems with a constrained number of resources. The microprocessor is a component found in practically all modern electronics, including smartphones, printers, washing machines, etc. Radars, satellites, and aircraft are just a few of the sophisticated applications that employ microprocessors [1]. Microprocessors and their derivatives are used more frequently and at lower cost as a result of the rapid improvements in the electronic industry and large-scale device integration, as seen in Figure 1.

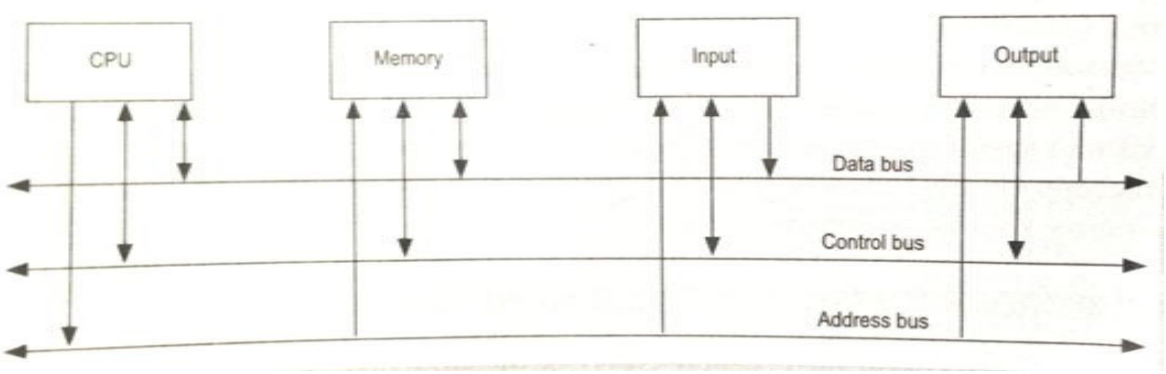


Figure 1: Represented that the Microprocessor based System [2]**.**

i. **Bit**: A bit is a single binary digit.

ii. **Word**: A word refers to the basic data size or bit size that can be processed by the arithmetic and logic unit of the processor. A 16-bit binary number is called a word ina 16-bit processor.

iii. **Bus**: A bus is a group of wires/lines that carry similar information.

iv. **System Bus**: The system bus is a group of wires/lines used for communication between the microprocessor and peripherals.

v. **Memory Word**: The number of bits that can be stored in a register or memory element is called a memory word.

vi. **Address Bus**: It carries the address, which is a unique binary pattern used to identifya memory location or an I/O port. For example, an eight bit address bus has eight lines and thus it can address $2^8 = 256$ different locations. The locations in hexadecimal format can be written as 00H – FFH.

vii. **Data Bus**: The data bus is used to transfer data between memory and processor or between I/O device and processor. For example, an 8-bit processor will generally have an 8-bit data bus and a 16-bit processor will have 16-bit data bus.

viii. **Control Bus**: The control bus carry control signals, which consists of signals for selection of memory or I/O device from the given address, direction of data transfer and synchronization of data transfer in case of slow devices.

Arithmetic and logic units (ALUs) work together with control units to process the instructions that are executed in a conventional microprocessor. The store-program concept is the basis for practically all microprocessors. Programming or instructions are sequentially preserved in memory locations where they will be executed in the store-program concept. A microprocessor must be programmable by the user in order to execute any task. So, the programmer needs to be aware of the resources, features, and supported commands inside the computer. The manufacturer of the microprocessors provides a list of the command for each microprocessor. A microprocessor's instruction set is available in binary machine code and mnemonics [3].

In binary numerals 0 and 1, the microprocessor communicates and performs operations. A machine language is a collection of binary pattern-based instructions that is challenging for us to comprehend. The assembly language is created by giving short names to the binary patterns, or mnemonics. An application called assembler is used to translate assembly-level language into binary machine-level language.

Technology Used

The semiconductor manufacturing technologies used for chips are:

i. Transistor-Transistor Logic (TTL),

ii. Emitter Coupled Logic (ECL),

iii. Complementary Metal-Oxide Semiconductor (CMOS)

Classification of Microprocessors

Based on their specification, application and architecture microprocessors are classified.

Based on Size of Data Bus

i. 4-bit microprocessor

ii. 8-bit microprocessor

iii. 16-bit microprocessor

iv. 32-bit microprocessor

Based on Application

i. General-purpose microprocessor- used in general computer system and can be used by programmer for any application. Examples, 8085 to Intel Pentium.

ii. Microcontroller- microprocessor with built-in memory and ports and can be programmed for any generic control application. Example, 8051.

iii. Special-purpose processors- designed to handle special functions required for an application. Examples, digital signal processors and application-specific integrated circuit (ASIC) chips [4].

Based on Architecture

i. Reduced Instruction Set Computer (RISC) processors,

ii. Complex Instruction Set Computer (CISC) processors

8085 Microprocessor Architecture

The 8085 microprocessor is an 8-bit device with a 40-pin IC package that runs on +5 V. It has a maximum operating frequency of 3 MHz. It can address 216 = 64 KB of memory because its data bus is 8 bits wide and address bus is 16 bits wide. Figure 2 displays the internal design of the 8085.

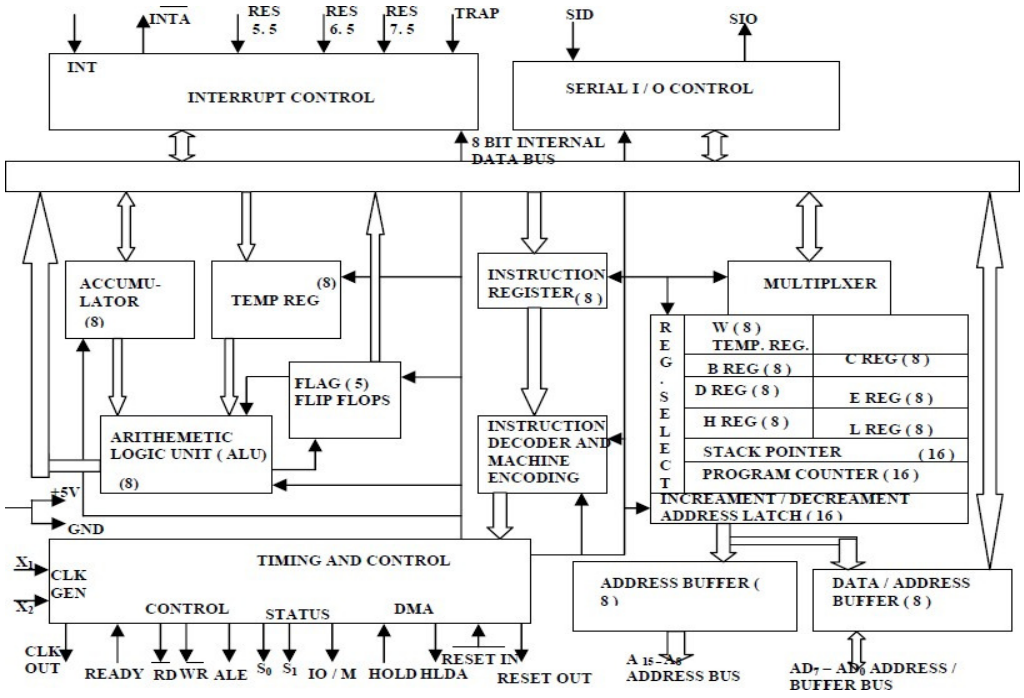


Figure 2: Represented that the Internal Architecture of 8085.

Arithmetic and Logic Unit

The ALU performs the actual numerical and logical operations such as Addition (ADD), Subtraction (SUB), AND, OR etc. It uses data from memory and from Accumulator to perform operations. The results of the arithmetic and logical operations are stored in the accumulator [5].

Registers

The 8085 includes six registers, one accumulator and one flag register, as shown in Figure 3.In addition, it has two 16-bit registers: stack pointer and program counter. They are briefly described as follows.

The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H and L. they can be combined as register pairs - BC, DE and HL to perform some 16-bit operations. The programmer can use these registers to store or copy data into theregister by using data copy instructions [6].

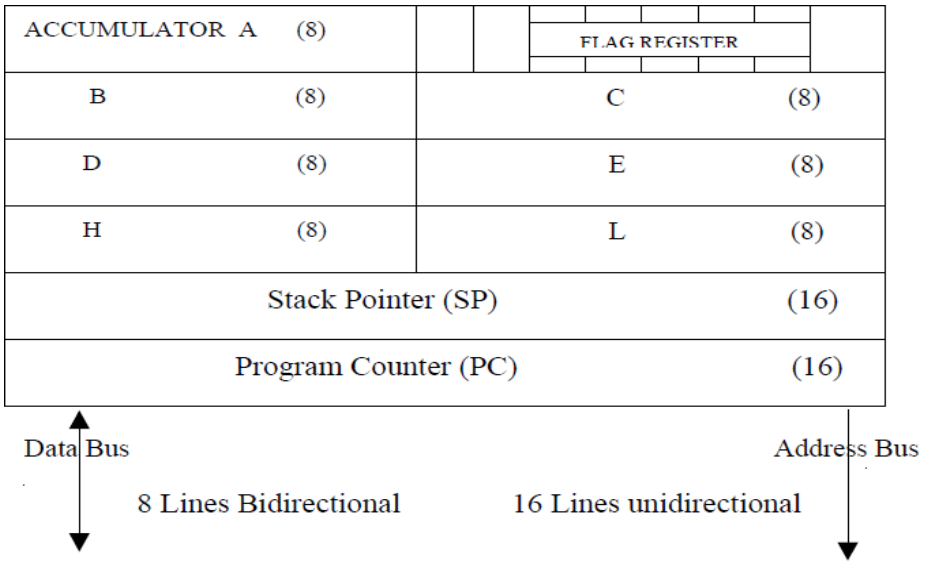


Figure 3: Represented that the Register Organization.

Accumulator

The accumulator is an 8-bit register that is a part of ALU. This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator [7]. The accumulator is also identified as register A.

Flag Register

The ALU includes five flip-flops, which are set or reset after an operation according to data condition of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary Carry (AC) flags. Their bit positions in the flag register are shown in Figure 4. The microprocessor uses these flags to test data conditions [8].

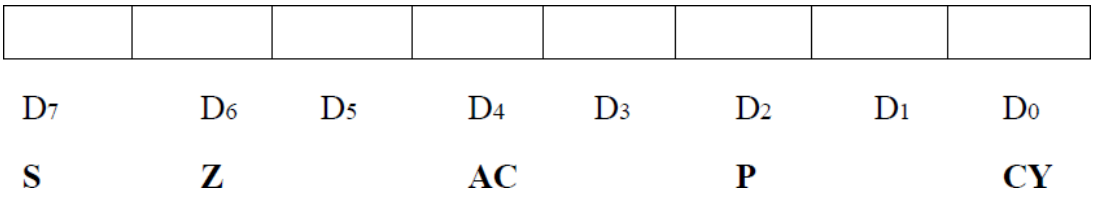


Figure 4: Represented that the Flag Register.

For example, after an addition of two numbers, if the result in the accumulator is larger than8-bit, the flip-flop uses to indicate a carry by setting CY flag to 1. When an arithmetic operation results in zero, Z flag is set to 1. The S flag is just a copy of the bit D7 of the accumulator. A negative number has a 1 in bit D7 and a positive number has a 0 in 2's complement representation. The AC flag is set to 1, when a carry result from bit D3 and passes to bit D4. The P flag is set to 1, when the result in accumulator contains even numberof 1s.

Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte is being fetched, the program counter is automatically incremented by one to point to the next memory location [9].

Stack Pointer (SP)

The stack pointer is also a 16-bit register, used as a memory pointer. It points to a memory location in R/W memory, called stack. The beginning of the stack is defined by loading 16- bit address in the stack pointer.

Instruction Register/Decoder

It is an 8-bit register that temporarily stores the current instruction of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and decodes or interprets the instruction. Decoded instruction then passed to next stage [10].

Control Unit

Generates signals on data bus, address bus and control bus within microprocessor to carry out the instruction, which has been decoded. Typical buses and their timing are described as follows

- **Data Bus**

Data bus carries data in binary form between microprocessor and other external units such as memory. It is used to transmit data i.e. information, results of arithmetic etc between memory and the microprocessor. Data bus is bidirectional in nature. The data bus width of 8085 microprocessor is 8-bit i.e. 2^8 combination of binary digits and are typically identified as D0 – D7. Thus size of the data bus determines what arithmetic can be done. If only 8-bit wide then largest number is 11111111 (255 in decimal). Therefore, larger numbers have to be broken down into chunks of 255. This slows microprocessor.

- **Address Bus**

The address bus carries addresses and is one way bus from microprocessor to the memory or other devices. 8085 microprocessor contain 16-bit address bus and are generally identified as A0 - A15. The higher order address lines (A8 – A15) are unidirectional and the lower order lines (A0 – A7) are multiplexed (time-shared) with the eight data bits (D0 – D7) and hence, they are bidirectional.

- **Control Bus**

Control bus are various lines which have specific functions for coordinating and controlling microprocessor operations. The control bus carriescontrol signals partly unidirectional and partly bidirectional. The following control and status signals are used by 8085 processor:

- **ALE (output)**

Address Latch Enable is a pulse that is provided when an address appears on the AD0 – AD7 lines, after which it becomes 0. RD (active low output): The Read signal indicates that data are being read from the selected I/O or memory device and that they are available on the data bus.

- **WR (active low output)**

The Write signal indicates that data on the data bus are to be written into selected memory or I/O location.

- **IO/M (output)**

It is a signal that distinguished between a memory operation and an I/O operation. When 1 it is an I/O operation. When, **O/M** = 0 it is a memory operation and IO/M = 1 it is an I/O operation.

- **S1 and S0 (output)**

These are status signals used to specify the type ofoperation being performed; they are listed in Table 1.

Table 1: Represented that the Status Signals and Associated Operations

S1	S0	States
0	0	Halt
0	1	Write
1	0	Read
1	1	Fetch

The schematic representation of the 8085-bus structure is as shown in Figure 5. The microprocessor performs primarily four operations:

1. Memory Read: Reads data (or instruction) from memory.
2. Memory Write: Writes data (or instruction) into memory.
3. I/O Read: Accepts data from input device.
4. I/O Write: Sends data to output device.

The 8085 processor performs these functions using address bus, data bus and control bus as shown in Figure 5.

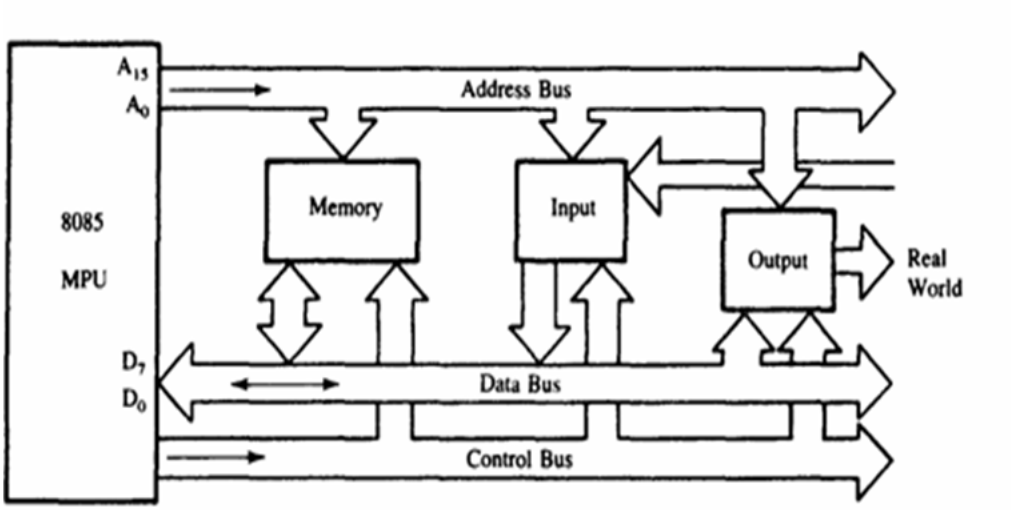


Figure 5: Illustrated that the 8085 Bus Structure.

8085PIN Description

Properties:

A. It is an 8-bit microprocessor.

B. Manufactured with N-MOS technology.

C. 40 pin IC package.

D. It has 16-bit address bus and thus has 2^{16} = 64 KB addressing capability.

E. Operate with 3 MHz single-phase clock.

F. +5 V single power supply

The logic pin layout and signal groups of the 8085nmicroprocessor are shown in Figure 6. All the signals are classified into six groups:

1. Address bus
2. Data bus
3. Control & status signals
4. Power supply and frequency signals
5. Externally initiated signals
6. Serial I/O signals

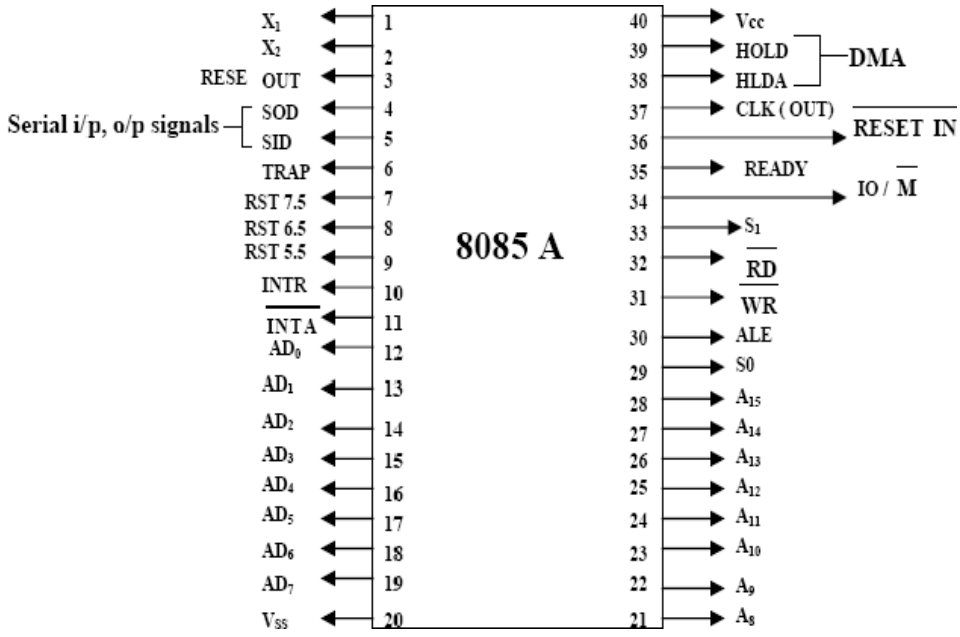


Figure 6: Represented that the 8085 Microprocessor PIN Layout and Signal Groups.

Address and Data Buses

1. A8 – A15 (output, 3-state): Most significant eight bits of memory addresses and the eight bits of the I/O addresses. These lines enter into tri-state high impedance state during HOLD and HALT modes.
2. AD0 – AD7 (input/output, 3-state): Lower significant bits of memory addresses and the eight bits of the I/O addresses during first clock cycle. Behaves as data bus during third and fourth clock cycle. These lines enter into tri-state high impedancestate during HOLD and HALT modes.

Control & Status Signals

1. **ALE:** Address latch enable
2. **RD:** Read control signal.
3. **WR:** Write control signal.
4. **IO/M:** S1 and S0: Status signals.

Power Supply & Clock Frequency

1. Vcc: +5 V power supply
2. Vss: Ground reference
3. X1, X2: A crystal having frequency of 6 MHz is connected at these two pins
4. CLK: Clock output

Externally Initiated and Interrupt Signals

1. **RESET IN:** When the signal on this pin is low, the PC is set to 0, the buses are tri-stated and the processor is reset.
2. **RESET OUT:** This signal indicates that the processor is being reset. The signal canbe used to reset other devices.
3. **READY:** When this signal is low, the processor waits for an integral number ofclock cycles until it goes high.
4. **HOLD:** This signal indicates that a peripheral like DMA (direct memory access) controller is requesting the use of address and data bus.
5. **HLDA:** This signal acknowledges the HOLD request.
6. **INTR:** Interrupt request is a general-purpose interrupt.
7. **INTA:** This is used to acknowledge an interrupt.
8. **RST 7.5, RST 6.5, RST 5,5 restart interrupt:** These are vectored interrupts andhave highest priority than INTR interrupt.
9. **TRAP:** This is a non-maskable interrupt and has the highest priority.Serial I/O Signals:
10. **SID:** Serial input signal. Bit on this line is loaded to D7 bit of register a using RIM instruction.
11. **SOD:** Serial output signal. Output SOD is set or reset by using SIM instruction.

Instruction Set and Execution In 8085

1. Based on the design of the ALU and decoding unit, the microprocessor manufacturer provides instruction set for every microprocessor. The instruction set consists of both machine code and mnemonics.
2. An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called instruction set. Microprocessor instructions can be classified based on the parameters such functionality, length and operand addressing.

Classification based on Functionality

1. Data transfer operations: This group of instructions copies data from source to destination. The content of the source is not altered.

2. Arithmetic operations: Instructions of this group perform operations like addition, subtraction, increment & decrement. One of the data used in arithmetic operation is stored in accumulator and the result is also stored in accumulator.

3. Logical operations: Logical operations include AND, OR, EXOR, NOT. The operations like AND, OR and EXOR uses two operands, one is stored in accumulator and other can be any register or memory location. The result is stored in accumulator. NOT operation requires single operand, which is stored in accumulator.

4. Branching operations: Instructions in this group can be used to transfer program sequence from one memory location to another either conditionally or unconditionally.

5. Machine control operations: Instruction in this group control execution of other instructions and control operations like interrupt, halt etc.

Classification based on Length

1. One-byte instructions: Instruction having one byte in machine code. Examples are depicted in Table 2.
2. Two-byte instructions: Instruction having two byte in machine code. Examples are depicted in Table 3.
3. Three-byte instructions: Instruction having three byte in machine code. Examplesare depicted in Table 4.

Table 2: Illustrated that the Examples of one byte Instructions.

Opcode	**Operand**	**Machine code/Hex code**
MOV	A, B	78
ADD	M	86

Table 3: Represented that the Examples of two byte Instructions

Opcode	Operand	Machine code/Hex code	Byte description
MVI	A, 7FH	3E	First byte
		7F	Second byte
ADI	0FH	C6	First byte
		0F	Second byte

Table 4: Represented that the Examples of three byte Instructions

Opcode	Operand	Machine code/Hex code	Byte description
JMP	9050H	C3	First byte
		50	Second byte
		90	Third byte
LDA	8850H	3A	First byte
		50	Second byte
		88	Third byte

DISCUSSION

MIPS (Microprocessor without Interlocked Pipe Stages) is a new general purpose microprocessor architecture designed to be implemented on a single VLSI chip. The main goal of the design is high performance in the execution of compiled code. The architecture is experimental since it is a radical break with the trend of modern computer architectures. The basic philosophy of MIPS is to present an instruction set that is a compiled-driven encoding of the micro engine. Thus, little or no decoding is needed and the instructions correspond closely to microeode instructions. The processor is pipelined but provides no pipeline interlock hardware; this function must be provided by software. The MIPS architecture presents the user with a fast machine with a simple instruction set. This approach has been used by the IBM 8071 project I and is currently being explored by the RISC project at Berkeley2; it is directly 'opposed to the approach taken by architectures such as the VAX. However, there are significant differences between the RISC approach and the approach used in MIPS which is The RISC architecture is simple both in the instruction set and the hardware needed to implement that instruction set. Although the MIPS instruction set has a simple hardware implementation (i.e. it requires a minimal amount of hardware control), the user level instruction set is not as straightforward, and the simplicity of the user level instruction set is secondary to the performance goals. The thrust of the RISC design is towards efficient implementation of a straightforward instruction set. In the M1PS design, high performance from the hardware engine is a primary goal, and the micro engine is presented to the end user with a minimal amount of interpretation. This makes most of the microcngine's parallelism available at the instruction set level.

CONCLUSION

The entire MIPS processor has been raided out and partitioned into a set of six test chips that cover all the data path and control functions on the chip. Four test chips have been sent out for fabrication as of August 1982; we expect send the remainder to fabrication during August 1982. In the software area code generators have been written for boll: C and Pascal these code generators produce simple instructions, relying on a pipeline reorganizer. A complete version of the pipeline reorganizer is running. An instruction level simulator is being used to obtain performance estimates. The dimensions of the chip are approximately 6.9 by 7.2 mm with a minimum feature size. The chip area is heavily dedicated to the data path as opposed to control structure, but not as radically as in RISC implementation. -Early estimates of performance seem to indicate that we should achieve approximately 2 MIPS using the Puzzle program as a benchmark compared to other architectures executing compiler generated code. We expect to have more accurate and complete benchmarks available in the near future.

REFERENCES

[1] V. G. Gaitan, N. C. Gaitan, and I. Ungurean, “CPU architecture based on a hardware scheduler and independent pipeline registers,” *IEEE Trans. Very Large Scale Integr. Syst.*, 2015, doi: 10.1109/TVLSI.2014.2346542.

[2] S. Mühlbach and S. Wallner, “Secure communication in microcomputer bus systems for embedded devices,” *J. Syst. Archit.*, 2008, doi: 10.1016/j.sysarc.2008.04.003.

[3] M. Woodward, “Microprocessors/microcomputers: Architecture, software and systems,” *J. Microcomput. Appl.*, 1982, doi: 10.1016/0745-7138(82)90008-2.

[4] W. Malyj, R. E. Smith, and J. M. Horowitz, “New directions in scientific computing: Impact

of advances in microprocessor architecture and system design," *Computer Programs in Biomedicine*. 1984. doi: 10.1016/0010-468X(84)90047-3.

[5] *Microcontroller Based Temperature Monitoring and Control*. 2002. doi: 10.1016/b978-0-7506-5556-9.x5000-2.

[6] F. Restrepo-Calle, S. Cuenca-Asensi, A. Martínez-Álvarez, E. Chielle, and F. L. Kastensmidt, "Application-Based Analysis of Register File Criticality for Reliability Assessment in Embedded Microprocessors," *J. Electron. Test. Theory Appl.*, 2015, doi: 10.1007/s10836-015-5513-9.

[7] M. Abheesh Kumar, A. Sudhakar, and J. Venkata Suman, "Design and implementation of compressor based 32-bit multipliers for mac architecture," *Int. J. Innov. Technol. Explor. Eng.*, 2019, doi: 10.35940/ijitee.i8517.078919.

[8] Q. Abu Al-Haija, H. Al-Amri, M. Al-Nashri, and S. Al-Muhaisen, "An engineering design of 4-bit special purpose microprogrammed processor," in *Procedia Computer Science*, 2013. doi: 10.1016/j.procs.2013.09.071.

[9] J. Kim, E. Teran, P. V. Gratz, D. A. Jiménez, S. H. Pugsley, and C. Wilkerson, "Kill the program counter: Reconstructing program behavior in the processor cache hierarchy," *ACM SIGPLAN Not.*, 2017, doi: 10.1145/3037697.3037701.

[10] S. M. T. Siddiquee, K. Kumar, B. Pandey, and A. Kumar, "Energy efficient instruction register for green communication," *Int. J. Eng. Adv. Technol.*, 2019.

CHAPTER 2

AN OVERVIEW OF THE ADDRESSING MODES IN INSTRUCTIONS

Dr.Pravinthraja, Associate Professor,
Department of Computer Science and Engineering,
Presidency University, Bangalore, Karnataka, India.
Email Id- pravinth.raja@presidencyuniversity.in

ABSTRACT:

Addressing modes are an aspect of the instruction set architecture in most central processing unit designs. The various addressing modes that are defined in a given instruction set architecture define how the machine language instructions in that architecture identify the operand of each instruction. Here are the addressing modes discussed: Immediate: The operand is included in the instruction. Direct: The effective address of the operand in memory is part of the instruction. Indirect: The instruction contains a memory address, which contains the effective address of the operand in memory.

KEYWORDS:

Communication, System, Electronics, Microprocessor, 8085 Interrupts.

INTRODUCTION

The Addressing mode refers to how the operand of an instruction is specified. It specifies a rule for interpreting/modifying the address field of the instruction before referencing the operand. Before jumping to different addressing modes, it is crucial that we first understand the basic concept of the operation cycle of a computer. The control unit is designed to go through an instruction cycle divided into three major phases:

i. **Fetch the Instruction from Memory:**

Program Counter (PC) is one of the registers present in the system that keeps track of the program's instructions stored in the memory. It holds the address of the next instruction to be executed and is incremented every time an instruction is fetched from memory.

ii. **Decode the Instruction:**

It determines the operation to be performed: the operands' location and the instruction's addressing mode. Further, the computer executes the instruction and returns to step 1 (fetching the instruction from memory) to fetch the next instruction in the sequence.

iii. **Execute the Instruction:**

The addressing mode of the instruction is specified with a unique binary code that designates both the operation and the mode of the instruction. Instructions can be defined with a variety of addressing modes. Combining two or more addressing modes in one instruction is also possible. The operands needed for the operation are located using the mode field. It is not necessary to have

an address field in the instruction. An address field can contain either a memory address or a processor register if there is an address field [1].

The process of specifying the data to be operated on by the instruction is called addressing. The various formats for specifying operands are called addressing modes. The 8085 has the following five types of addressing:

1. Immediate addressing
2. Memory direct addressing
3. Register direct addressing
4. Indirect addressing

Implicit Addressing Immediate Addressing:

In this mode, the operand given in the instruction - a byte or word transfers to the destination register or memory location. Ex: MVI A, 9AH.

A. The operand is a part of the instruction.

B. The operand is stored in the register mentioned in the instruction.Memory Direct Addressing:

C. Memory direct addressing moves a byte or word between a memory location and register.The memory location address is given in the instruction. Ex: LDA 850FHThis instruction is used to load the content of memory address 850FH in the accumulator.

Register Direct Addressing:

Register direct addressing transfer a copy of a byte or word from source register todestination register.

Ex: MOV B, C.

It copies the content of register C to register B.Indirect Addressing:

A. Indirect addressing transfers a byte or word between a register and a memory location.Ex: MOV A, M.

B. Here the data is in the memory location pointed to by the contents of HL pair. The data ismoved to the accumulator.

Implicit Addressing

A. In this addressing mode the data itself specifies the data to be operated upon.Ex: CMA.

B. The instruction complements the content of the accumulator. No specific data or operand ismentioned in the instruction [2].

Instruction SET OF 8085

The various techniques to specify data for instructions are:

- **i.** 8-bit or 16-bit data may be directly given in the instruction itself.
- **ii.** The address of the memory location, I/O port or I/O device, where data resides, may be given in the instruction itself.
- **iii.** In some instructions, only one register is specified. The content of the specified register is one of the operands.
- **iv.** Some instructions specify two registers. The contents of the registers are the required data.
- **v.** In some instructions, data is implied. The most instructions of this type operate on the content of the accumulator.

Due to different ways of specifying data for instructions, the machine codes of all instructions are not of the same length. It may 1-byte, 2-byte or 3-byte instruction.

Addressing Modes

Each instruction requires some data on which it has to operate. There are different techniques to specify data for instructions. These techniques are called **addressing modes**. Intel 8085 uses the following addressing modes:

Direct Addressing

In this addressing mode, the address of the operand (data) is given in the instruction itself.

Example

STA 2400H: It stores the content of the accumulator in the memory location 2400H.

32, 00, 24: The above instruction in the code form.

In this instruction, 2400H is the memory address where data is to be stored. It is given in the instruction itself. The 2nd and 3rd bytes of the instruction specify the address of the memory location. Here, it is understood that the source of the data is accumulator[3].

Register Addressing

In register addressing mode, the operand is in one of the general-purpose registers. The opcode specifies the address of the register(s) in addition to the operation to be performed.

Example:

MOV A, B: Move the content of B register to register A.

Instruction Execution and Timing Diagram

Each instruction in 8085 microprocessor consists of two part- operation code (opcode) and operand. The opcode is a command such as ADD and the operand is an object to be operated on,

such as a byte or the content of a register.

Instruction Cycle

The time taken by the processor to complete the execution of an instruction. An instruction cycle consists of one to six machine cycles [4].

Machine Cycle

1. The time required to complete one operation; accessing either the memory or I/O device. A machine cycle consists of three to six T-states.
2. T-State: Time corresponding to one clock period. It is the basic unit to calculate execution of instructions or programs in a processor. To execute a program, 8085 performs various operations as:
    i. Opcode fetch
    ii. Operand fetch
    iii. Memory read/write
    iv. I/O read/write

External communication functions are:

i. Memory read/write
ii. I/O read/write

Interrupt request acknowledgeOpcode Fetch Machine Cycle:

It is the first step in the execution of any instruction. The timing diagram of this cycle isgiven in Figure 1.

The following points explain the various operations that take place and the signals that are changed during the execution of opcode fetch machine cycle:

T1 Clock Cycle

1. The content of PC is placed in the address bus; AD0 - AD7 lines contains lower bit address and A8 A15 contains higher bit address.
2. IO/M signal is low indicating that a memory location is being accessed. S1 and S0also changed to the levels as indicated in Table 1.
3. ALE is high, indicates that multiplexed AD0 – AD7 act as lower order bus.

T2 Clock Cycle

1. Multiplexed address bus is now changed to data bus.
2. The RD signal is made low by the processor. This signal makes the memory deviceload

the data bus with the contents of the location addressed by the processor.

T3 Clock Cycle

1. The opcode available on the data bus is read by the processor and moved to the instruction register.
2. The RD signal is deactivated by making it logic 1.

T4 Clock Cycle

The processor decodes the instruction in the instruction register and generate the necessary control signals to execute the instruction. Based on the instruction further operations such as fetching [5], writing into memory etc. takes place.

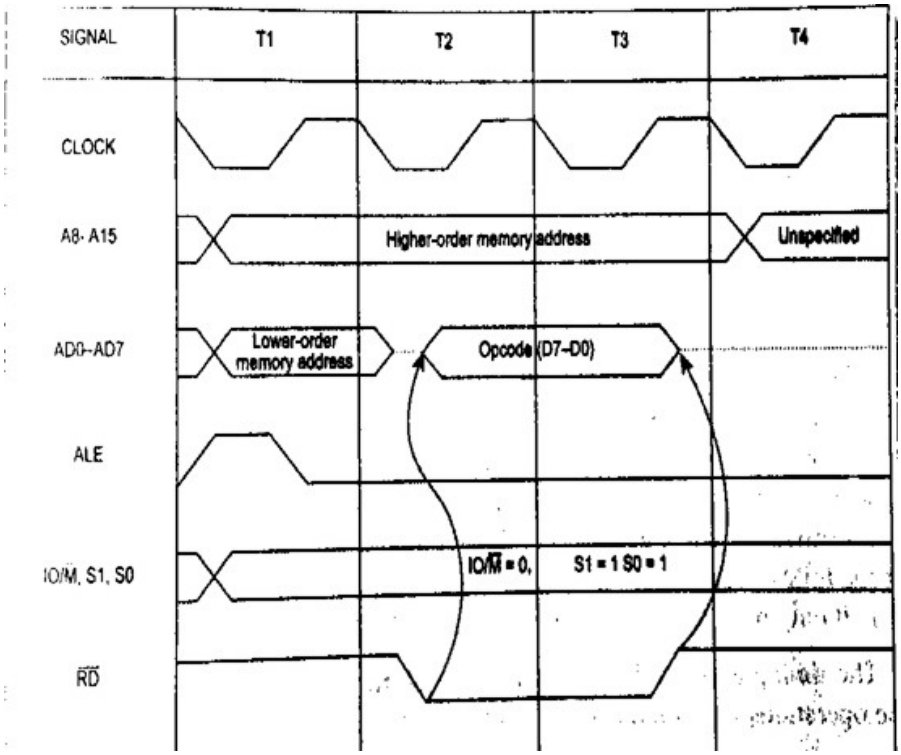


Figure 1: Represented that the Timing Diagram for Opcode Fetch Cycle.

Memory Read Machine Cycle:

The memory read cycle is executed by the processor to read a data byte from memory. The machine cycle is exactly same to opcode fetch except: a) It has three T-states b) The S0 signal is set to 0. The timing diagram of this cycle is given in Figure 2.

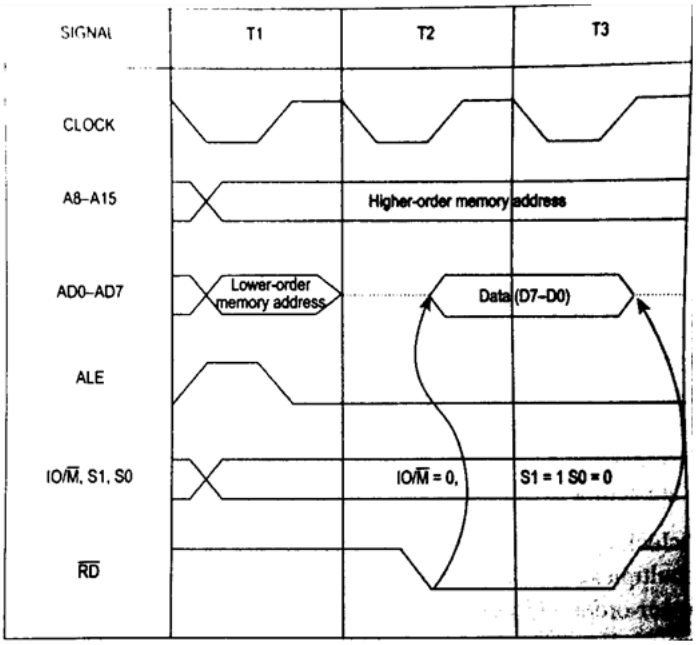


Figure 2: Represented that the Timing Diagram for Memory Read Machine Cycle.

Memory Write Machine Cycle

The memory write cycle is executed by the processor to write a data byte in a memory location. The processor takes three T-states and WR signal is made low. The timing diagram of this cycle is given in Figure 3.

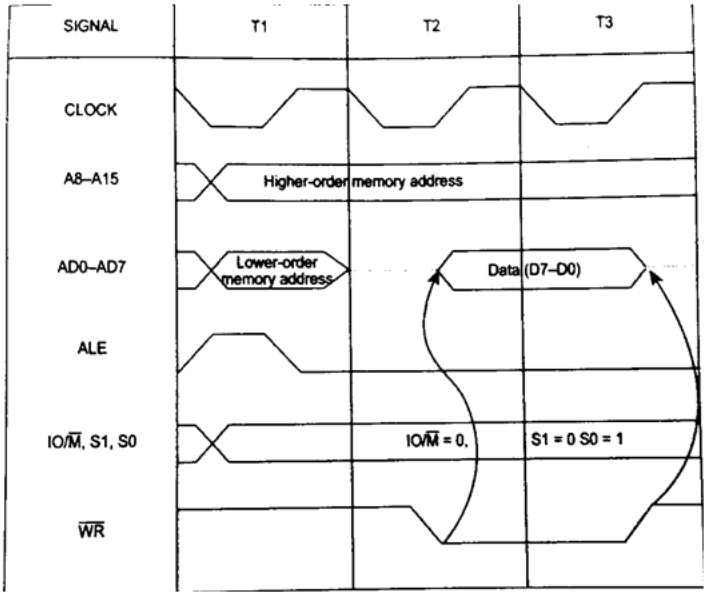


Figure 3: Represented Timing Diagram for Memory Write Machine Cycle

I/O Read Cycle:

The I/O read cycle is executed by the processor to read a data byte from I/O port or from peripheral, which is I/O mapped in the system. The 8-bit port address is placed both in the lower and higher order address bus. The processor takes three T-states to execute this machine cycle. The timing diagram of this cycle is given in Figure 4.

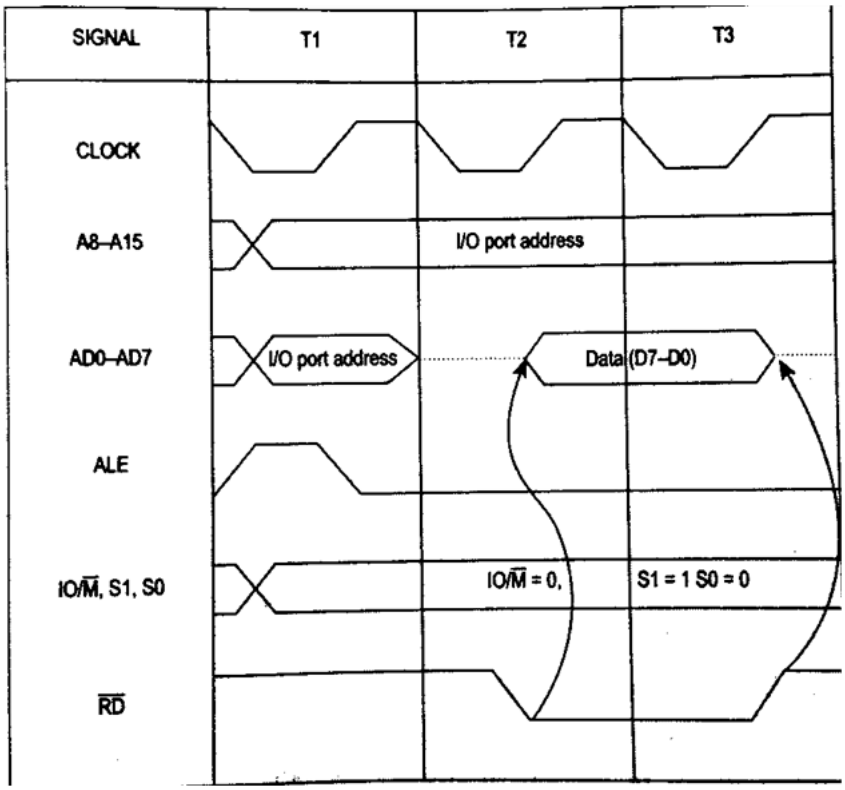


Figure 4: Represented that the Timing diagram I/O Read Machine Cycle.

I/O Write Cycle

The I/O write cycle is executed by the processor to write a data byte to I/O port or to a peripheral, which is I/O mapped in the system. The processor takes three T-states to executethis machine cycle. The timing diagram of this cycle is given in Figure 5.

Ex: Timing diagram for IN 80H.

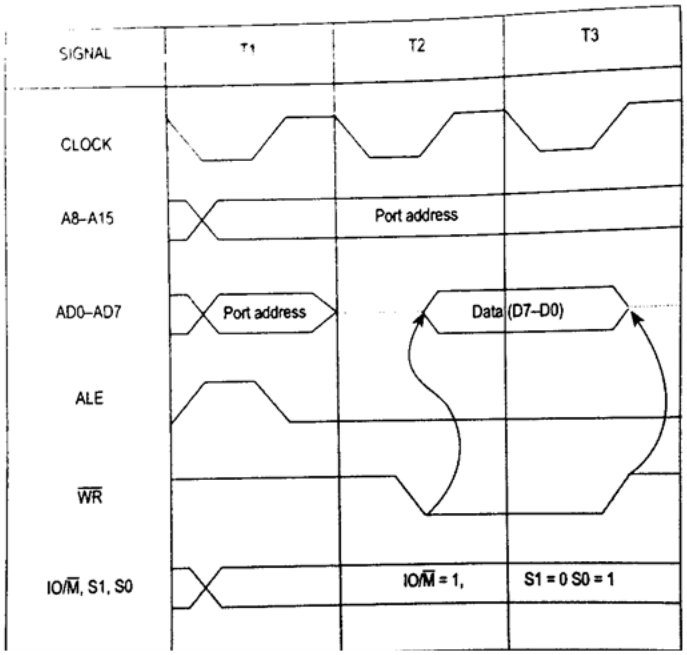


Figure 5: Represented that the Timing diagram I/O Write Machine Cycle

The instruction and the corresponding codes and memory locations are given in Table 1.

Table 1: Represented that the IN instruction

Address	Mnemonics	Opcode
800F	IN 80H	DB
8010		80

i. During the first machine cycle, the opcode DB is fetched from the memory, placedin the instruction register and decoded.

ii. During second machine cycle, the port address 80H is read from the next memory location.

iii. During the third machine cycle, the address 80H is placed in the address bus and the data read from that port address is placed in the accumulator. The timing diagram is shown in Figure 6**.**

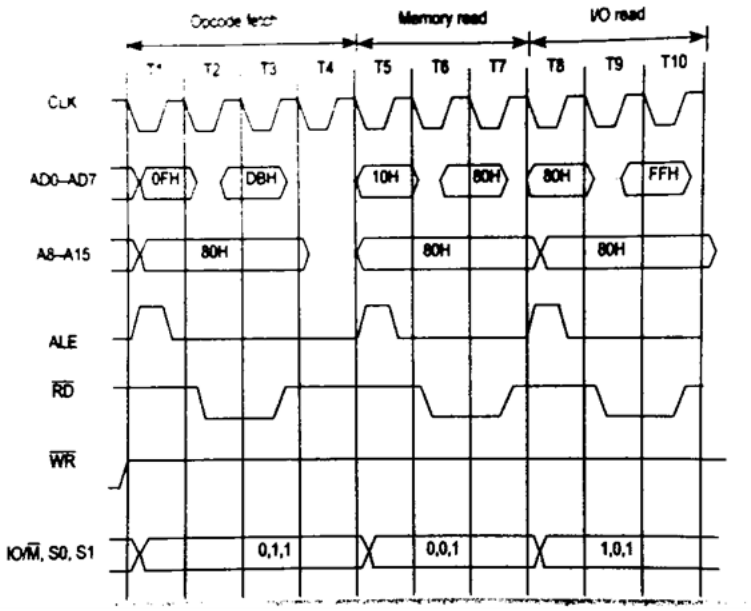


Figure 6: Represented that the Timing Diagram for the IN instruction

DISCUSSION

Energy conservation has been a driving factor in the design of new hardware and software for wireless sensor networks over the last several years. Many energy efficient systems have been proposed by the community, spanning architectures, operating systems, sensor usage, and algorithm and protocol design. [6] One technique not applied to sensor networks to date, which might deliver more energy efficiency savings across the board, is low power compilation. Researchers in the low power compilation community have developed a number of techniques that can optimize the compilation process for energy consumption when targeting specific systems design. These researchers observed that the standard optimization techniques and heuristics offered by today's compilers do not result in binaries that are optimized for energy efficiency. Rather, compilers are designed to optimize the binary for execution performance and to a lesser degree binary size [7], [8].

While there has been work on low power compilation, we are not aware of it being applied to sensor networks. Existing low power compilation techniques are based on an understanding of the relationship between the execution of computational workloads on a specific targeted processor and the associated energy consumption observed. The contribution of our paper only represents a first step at studying this relationship when considering the Tmote Sky sensor device. We conduct a number of experiments using the Tmote Sky in order to characterize this relationship. We conjecture that our results and findings can assist in the future designs of compilers that can specifically exploit sensor hardware design and application workload to reduce the energy consumption. For example, one very simple optimization, derived experimentally, results in a modest but measurable reduction in an existing sensor network application's energy consumption of 4.3%. We conjecture more work on low power compilation techniques can lead to more significant savings, but may require accepting tradeoffs. We discuss these tradeoffs and use the results from a set of experiments to argue for more work in the area of low power compilation for sensor networks [8]–[10].

CONCLUSION

In this chapter, it proposed applying techniques used in the area of low power compilation to sensor networks in order to potentially extend the lifetime of such networks. We studied the impact of instruction types, circuit state effects, instruction operand ordering, memory addressing modes and GCC compiler optimization flags on energy consumption. We reported a subset of our findings which we used to identify a simple instruction level optimization to an existing sensor application. The application of this optimization resulted in a modest 4.3% reduction in the power consumption of the application. This result in itself is inconclusive as to establishing the utility of this approach but it does suggest further examination is warranted. While this paper presents more questions than it answers our main objective is to raise the issue of developing new low power compiler optimization techniques for sensor networks.

REFERENCES

[1] R. A. Karim, N. F. Zakaria, M. A. Zulkifley, M. M. Mustafa, I. Sagap, and N. H. Md Latar, "Telepointer technology in telemedicine : A review," BioMedical Engineering Online. 2013. doi: 10.1186/1475-925X-12-21.

[2] H. Cheng, D. Dinu, and J. Großschädl, "Efficient implementation of the sha-512 hash function for 8-bit AVR microcontrollers," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2019. doi: 10.1007/978-3-030-12942-2_21.

[3] D. Whetzel, T. Sullivan, and R. McCloy, "Situational Judgment Tests: An Overview of Development Practices and Psychometric Characteristics," Pers. Assess. Decis., 2020, doi: 10.25035/pad.2020.01.001.

[4] C. Lund et al., "Sustainable energy education: addressing the needs of students and industry in Australia," Renew. Energy Environ. Sustain., 2017, doi: 10.1051/rees/2017046.

[5] S. Rajamoorthi, R. Mishori, L. Buchanan, and E. Morris, "Global is Local: Assessing Family Medicine Residency Programs' Training on the Care of Immigrants, Migrants, Torture Survivors, Asylees and Refugees (IMTARs)," Ann. Glob. Heal., 2017, doi: 10.1016/j.aogh.2017.03.297.

[6] S. P. Dandamudi, Guide to assembly language programming in linux. 2005. doi: 10.1007/0-387-26171-0.

[7] T. Daher and B. Meyer, "Using blended learning to address instructional challenges in a freshman engineering course," in ASEE Annual Conference and Exposition, Conference Proceedings, 2016. doi: 10.18260/p.27133.

[8] J. R. Smith, "External Memory," in Programming the PIC Microcontroller with MBASIC, 2005. doi: 10.1016/b978-075067946-6/50020-1.

[9] D. Kusswurm, "X86-32 Core Architecture," in Modern X86 Assembly Language Programming, 2014. doi: 10.1007/978-1-4842-0064-3_1.

[10] C. Kuehl et al., "Fast Fourier Transform Co-processor (FFTC) - Towards embedded GFlops," in European Space Agency, (Special Publication) ESA SP, 2012. doi: 10.1117/12.974825.

CHAPTER 3

AN ELABORATION OF THE 8085 INTERRUPTS

Dr. Jothish C, Assistant Professor,
Department of Computer Science and Engineering,
Presidency University, Bangalore, Karnataka, India.
Email Id- Jothish.c@presidencyuniversity.in

ABSTRACT:

As an instance for a large specification, an algebraic specification of the Intel 8085 microprocessor is given. The specification is based on the concepts of hierarchical abstract types and conditional equations. With the help of the specification interpreter RAP, the specification is validated against some of its informal requirements. In the design of large software systems, a number of informal specification properties have to be considered such as style, readability, and structuredness of a specification. These properties are talked about using a couple of small examples.

KEYWORDS:

Communication, System, Electronics, Microprocessor, 8085 Interrupts.

INTRODUCTION

As already discussed, in Interrupt Driven I/O data transfer methods the microprocessor gets interrupted by the I/O device when it is ready to transfer the data. The microprocessor suspends its job after executing the current instruction. It saves the contents of program counter to stack and jumps to the subroutine program. This subroutine program is called Interrupt Service Subroutine (ISS) program. The ISS saves the processor status into stack; and after executing the instruction for the data transfer, it restores the processor status and then returns to main program. The ISS executes the relevant set of instructions stored at a predetermined memory block. The Intel 8085 microprocessor has five hardware interrupt inputs on its chip. Besides these, the microprocessor has eight interrupt instruction in the instruction set. The microprocessor responds to both the hardware and software interrupts. In the following sections the details of both software and hardware interrupts will be discussed[1].

Software Interrupts

The 8085 microprocessor has eight software interrupts namely, RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6 and RST 7. The syntax for these interrupt instructions is given by:

RSTn

Where n varies from 0 to 7. These instructions are single byte instructions. When an interrupt occurs, a CALL instruction to a predetermined location of the memory is executed. These RST instructions are like vectors because they point to specific locations in the memory. The starting address of each instruction is called a Vector Location. The vector location for RST 0 is 0000 H and for RST 1 is 0008 H and so on. The vector locations of each instruction are 8 bytes apart. Therefore 8 bytes of instructions can be stored beginning at any vector location. Figure 1, shows

the hardware for the implementation of the software interrupt instruction. As display in Figure 1, this circuit is for the implementation of an instruction RST0. In response to the interrupt request signal, the 8085 microprocessor sends the I NTA (interrupt acknowledge) signal, which is used to enable the buffer. The hex code C7 H of RST 0 will be placed on the data bus.

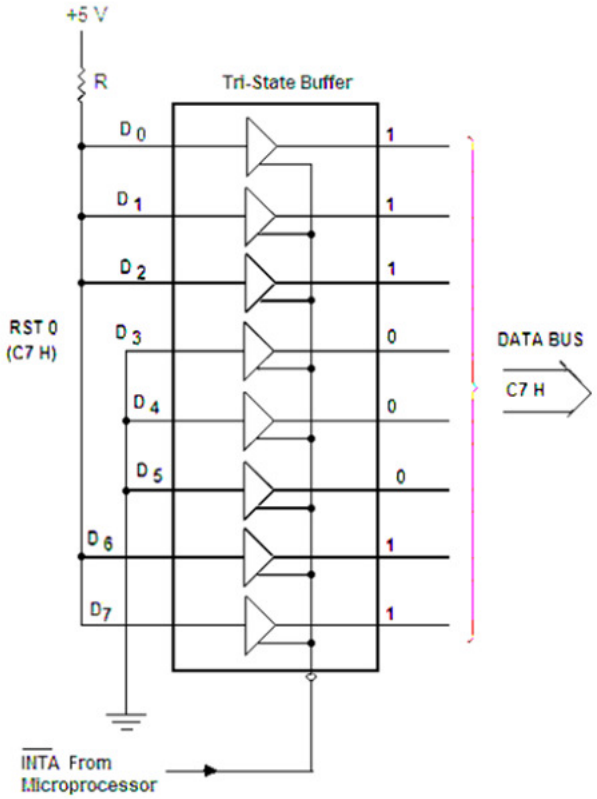


Figure 1: Represented that the Hardware for the Implementation of the Software Interrupt Instruction.

Figure 2, illustrates how software interrupt instructions are executed. Suppose in the main program RST2 instruction is given, and when this instruction is executed the contents of the program counter is pushed onto the stack. Then the program counter jumps to the address 0010 H. The subroutine located from 0010 H to 0017 H is executed, with the RET instruction as the last instruction of the subroutine program. So as the RET instruction is executed it returns to the main program. Similarly, if another restart instruction RST4 is encountered in the main program, then the contents of the program counter are again pushed onto the stack. The program jumps to the subroutine program whose starting address is given by 0020 H. After the execution of this subroutine program, it returns to the main program.

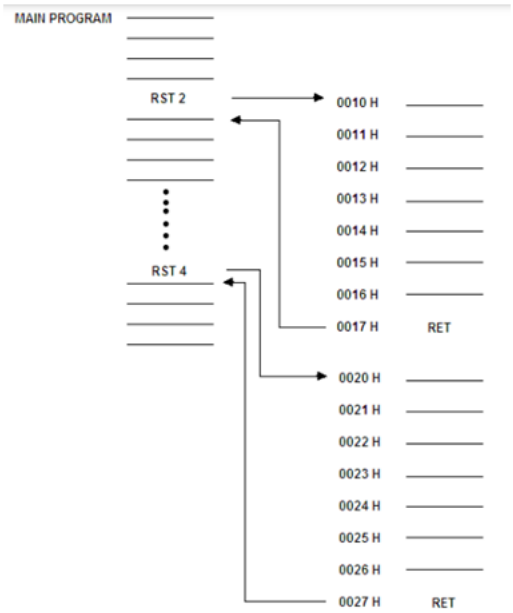


Figure 2: Represented That the Software Interrupt Instructions are executed.

From the above discussion it is clear that the software interrupt instructions are basically special kind of CALL instructions; when any of these instructions is executed it branches to the predetermined address. Notice that all these instructions are of one byte while the standard CALL instruction is of three bytes. The vector addresses of these 8 software interrupts are 8 bytes apart. So 8 bytes of instructions can be stored in the subroutine program associated with each of these instructions. But generally, the subroutine programs may require more than 8 bytes. For this reason the complete subroutine program are usually not stored in the vector locations, rather the vector locations are used to specify the starting address of the longer subroutine program i.e. in the vector location of these instructions JMP XXXX H may be stored. The XXXX H represents the starting address of the longer subroutine program. The last instruction of the program will be RET, so that after completing the subroutine program it jumps to the main program as shown in Figure 3 .

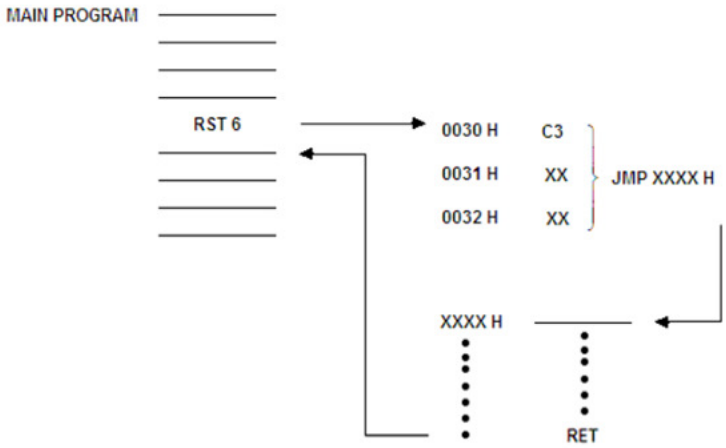


Figure 3: Illustrated that the Completing the Subroutine Program it Jumps.

The hardware solution to multi-interrupt problem is carried out by the circuit. The multi-interrupt problem means the possibility of two requests arriving simultaneously. This circuit will accept

eight interrupt request and work as per their priority and separate RST instruction for each request is generated. For the generation of all eight RST instructions, a 3-to-8 priority encoder (IC 74LS148) along with a tristate octal buffer (IC 74LS244) is used as shown in Figure 4.

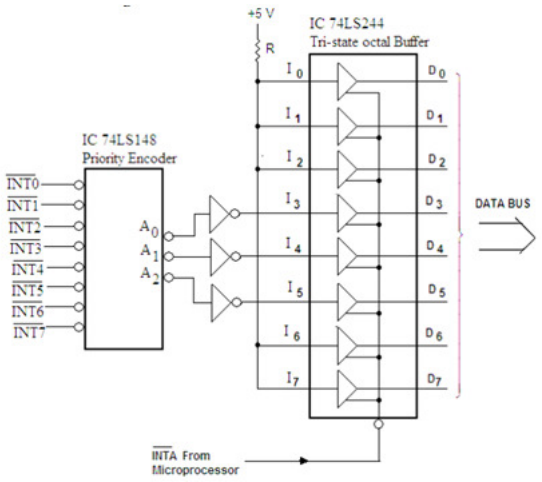


Figure 4: Represented that the Multi Interrupt Problem.

The 3-to-8 priority encoder generates a 3-bit binary code corresponding to active input. If two or more inputs are active simultaneously, the highest numbered input will be encoded. The three bits are inverted by the inverted buffer and then applied to an octal buffer (IC 74LS244). The op code corresponding to the input will be generated which is latched on to the data bus if INTA signal is active. The op codes for each RST instruction given in Table 1, are reproduced below:

Table 1: Represented that the Different RST Instruction.

Sr. No.	Instruction	Op Code	Binary equivalent
1.	RST 0	C7	11000111
2.	RST 1	CF	11001111
3.	RST 2	D7	11010111
4.	RST 3	DF	11011111
5.	RST 4	E7	11100111
6.	RST 5	EF	11101111
7.	RST 6	F7	11110111

8.	RST 7	FF	11111111

It may be noted from this op code table that D0 to D2 and D6 to D7 bits of the op codes are same for all the RST instructions (RST 0 to RST 7). The bits D3 to D5 are different and are in the sequence of 3 bit binary numbers for the RST 0 to RST 7. In general the code for RST instructions may be written as:[2]

11CCC111

Where CCC is,

000 for RST 0

001 for RST 1

010 for RST 2

011 for RST 3

100 for RST 4

101 for RST 5

110 for RST 6

111 for RST 7

This sequence permits to use the 3-to-8 priority encoder. It can be understood that a signal say INT3 is low then A0 A1 A2 will be100 which will be inverted by the inverter buffer as 011 (I3 I4 I5). Since the other lines I0, I1, I2, I6, I7 are all high, the octal buffer will provide the op code DF H (op code for RST 3) on the 8085 data bus as soon as the interrupt acknowledge signal (INTA) is made low in response to the interrupt request signal. Similarly by activating other inputs of the priority encoder, the vector CALL instructions can be generated on the 8085 data bus.

Hardware Interrupts

In addition of software interrupts discussed above the 8085 has five hardware interrupts also. For this, five hardware input pins are provided in the microprocessor. The hardware interrupts are initiated by an external device, by placing an appropriate signal at the interrupt pin of the microprocessor. The five interrupts RST 5.5, RST 6.5, RST 7.5, TRAP and INTR are shown in Table 2. Out of these interrupts RST 5.5, RST 6.5, RST 7.5 and TRAP are vector interrupts and INTR is the non-vectored interrupt. In vector interrupts the microprocessor automatically sends the specific address to program counter in response to an interrupt request signal. However, in non-vectored interrupt, the interrupt device has to give the address of the interrupt service subroutine [3].

Table 2: Represented that the Different Interrupts and Priority.

Interrupt	Priority	Vector Location
TRAP	1	0024 H
RST 7.5	2	003C H
RST 6.5	3	0034 H
RST 5.5	4	002C H
INTR	5	-----

It may be noted from the table that TRAP has the highest priority, RST 7.5 next highest and so on. If two or more hardware interrupts are served at the same time then the microprocessor executes them in order of their priority level; i.e. TRAP is served first, then RST 7.5 and so on. When highest priority interrupt is executed, the lower priority interrupts remain pending rather they are known as pending interrupts.

The RST 7.5, RST 6.5 and RST 5.5 are the mask able interrupts and TRAP is non-maskable interrupt. The mask able means the prevention of any interrupt. Sometimes it may be required to prevent one or more interrupts when a certain task is being carried out by the microprocessor; for this the masking of these interrupts is done. The masking of any interrupt is carried out by an instruction known as SIM (Set Interrupt Mask). The SIM is one byte instruction. To find the status of the interrupts i.e. to know which interrupt is masked or which interrupt is pending, another instruction known as RIM is used [4].

Interrupt Control Circuit

The interrupt control circuit with 8085 microprocessors. The TRAP is non-maskable interrupt i.e., it is neither be affected by any flag nor can be masked. It is used to handle very important functions. Since it is a highest priority interrupt, so it is used to take care of parity errors, power failure and other events that needs immediate attention. In the case of power failure, it may execute a routine to transfer the contents of the main memory to the backup memory (if any); and also, for parity errors, the data may be corrected before carrying on. It is edge and level trigger which means the input has to go high and stays high. The rising edge and level triggered TRAP signal triggers the D flip-flop. The logic '1' at the output of D flip-flop and logic '1' of the TRAP input enable the AND gate to trigger the TRAP interrupt i.e. it calls the vector location 0024 H. Once the TRAP is recognized, further inputs at the TRAP will not be considered unless the interrupt is reset. The TRAP interrupt will be reset if the RESET signal is active (low) or internal TRAP Acknowledge signal is high. After the 8085 microprocessor recognizes a TRAP interrupt, it will send a high TRAP acknowledge signal which will reset the D flip-flop[5].

RST 7.5 is a maskable interrupt which can be enabled or disabled by using SIM instruction. This is an edge triggered interrupt. When a leading edge signal appears at the RST 7.5 pin of 8085 microprocessor, D flip-flop 2 will be set. The output of this flip-flop is labeled as I 7.5 which is known as pending interrupt. This is one input of AND gate 2. The AND gate 2 will not be enabled until other two inputs of the gates are high. The RST flip-flop will be reset either by having a high R 7.5 bit or by having a high RST Acknowledge signal. The interrupt Acknowledge signal resets it for future RST 7.5 interrupt.

RST 6.5 and RST 5.5 are also maskable interrupts which may be enabled or disabled using SIM instruction. Both are level triggered interrupts. When a high signal (constant voltage of +5 V) appears at RST 6.5 pin of the microprocessor, it will enable one pin of AND gate 3. Till RST 6.5 interrupt pin is high and other two pins of AND gate 3 are low, this interrupt is known as pending interrupt (I 6.5). Similarly, when a high signal appears on RST 5.5 pin of the microprocessor, it will enable one terminal of AND gate 4. This interrupt will be pending interrupt (I 5.5) till RST 5.5 pin is high and other two inputs of AND gate 4 are low. It may be noted from the one input each of the AND gates 2, 3 and 4 is connected to IE signal (called interrupt enable flag). The second pin of these gates will be enabled if IE signal is high. The IE signal may be made to high by enabling the EI (Enable Interrupt) signal, which may be enabled by a software instruction EI. This instruction will be discussed in the succeeding section [6].

The mask bits M 7.5, M 6.5 and M 5.5 for the interrupts may be set using the SIM instruction. To mask an interrupt, the corresponding mask bit has to be set. So to enable a pending interrupt, the corresponding mask bit should be made low and interrupt enable flag should made high (by EI instruction). All the maskable interrupts may be disabled by either sending a low signal to RESETIN terminal or a high signal to 'Any Interrupt Acknowledge' terminal. The DI signal may also disable all the maskable interrupts. The DI (Disable Interrupts) is a software instruction. All these hardware interrupts discussed above, once enabled will execute the corresponding hardware CALL instruction specified by their vector locations, as per their priority levels. INTR is a lowest priority, maskable and level triggered interrupt. It uses handshaking. A high signal to INTR pin of the microprocessor will cause the current instruction to complete and will put the contents of the program counter into stack. The microprocessor will then generate a low INTA (interrupt acknowledge) signal. This signal is then used to enable a tristate buffer for the execution of hardware CALL instruction. It will execute the service subroutine program corresponding to any of the 8 software interrupts (RST 0 through RST 7).

Interrupt Instructions

Once the microprocessor recognizes any of the interrupts, it immediately disables all the interrupts except TRAP. This is done just to ensure that no further interrupts are recognized while the interrupt service subroutine (ISS) is being executed. Once the ISS is complete the program is required to enable the interrupts again as display in Figure 5. For this one byte instruction EI is introduced just before the RET instruction of ISS.

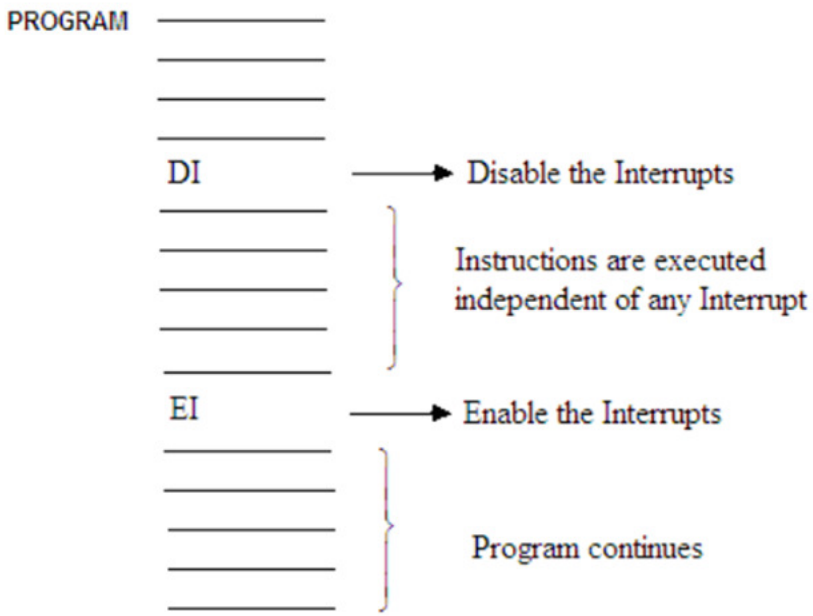


Figure 5: Represented that the Interrupt Instructions.

Need For Interrupts

Interrupt is a signal send by an external device to the processor, to the processor to perform a particular task or work. Mainly in the microprocessor-based system the interrupts are used for data transfer between the peripheral and the microprocessor. When a peripheral is ready for data transfer, it interrupts the processor by sending an appropriate signal to the interrupt pin of the processor. If the processor accepts the interrupt, then the processor suspends its current activity and executes an interrupt service subroutine to complete the data transfer between the peripheral and processor. After executing the interrupt service routine, the processor resumes its current activity. This type of data transfer scheme is called interrupt driven data transfer scheme [7].

Types of Interrupts

The interrupts are classified into software interrupts and hardware interrupts.

1. The software interrupts are program instructions. These instructions are inserted at desired locations in a program. While running a program, If a software interrupt instruction is encountered, then the processor executes an interrupt service routine (ISR)[8].
2. The hardware interrupts are initiated by an external device by placing an appropriate signal at the interrupt pin of the processor. If the interrupt is accepted, then the processor executes an interrupt service routine (ISR).

DISCUSSION

Interrupts are the signals that are generally produced by the devices externally connected to the microprocessor, requesting for the services. Whenever an interrupt request is generated in the system then it must not be neglected and be acknowledged as soon as possible. Basically whenever an interrupt is generated then the microprocessor suspends its current execution and switches to service the interrupt that is requested by the external device. In order to service the interrupt, the processor executes a routine which is called the interrupt service routine. So, after the execution

of interrupt service routine (ISR) the processor, resumes the original program that was under operation before the generation of the interrupt. Sometimes some special instructions inside the processor generate interrupts. Those are handled by the processor internally. Suppose the processor is executing an instruction and a keyboard key is pressed. Then at this time, the processor gets to know that the external device keyword is requesting its services. So, if the processor checks for the priority of the generated interrupt and if the generated interrupt holds high priority then the processor switches to execute the ISR by storing the address of the current program in the stack. This is done so that after the execution of ISR, the processor can switch back to the main program[9], [10].

CONCLUSION

Abstract data types can really not be called a hardware description language, Sint 80D, since the latter are designed and used for specific hardware description support. So it is a remarkable fact that algebraic specifications do their job rather nicely here. The hidden entities in a device like state and communication streams must however be made explicit. This looks pretty unfamiliar, but once the concepts are established, they can be used in a straightforward way. COLD is certainly more comfortable in this point of view since it supports imperative features like procedures and states, too. During the sessions, a lot of bugs in the specification could be eliminated. Most of them were quite trivial, they were syntax or typing errors. Some nontrivial bugs however came up at runtime, during the execution of small tasks for test purposes. The specification interpreter had usually no problems to solve equation systems on top of the microprocessor specification. This is a bit surprising, because of the large amount of data around. For instance, about 260 laws have to be considered, and the terms can easily grow over a couple of screens. All the same, many tasks terminate already after a few seconds with a most general solution. Sometimes however, even if nice properties hold for the specification, the interpreter is likely to fail. This is when the proof tree complexity becomes exponential. Even simple-looking goals can by exponential growth become non-manageable.

REFERENCES

[1] J. G. Hunt, “Interrupts,” Softw. Pract. Exp., 1980, doi: 10.1002/spe.4380100704.

[2] J. E. Smith and A. R. Pleszkun, “Implementing Precise Interrupts in Pipelined Processors,” IEEE Trans. Comput., 1988, doi: 10.1109/12.4607.

[3] H. Yao et al., “Methadone interrupts neural growth and function in human cortical organoids,” Stem Cell Res., 2020, doi: 10.1016/j.scr.2020.102065.

[4] A. Chehab, S. Hanna, K. Y. Kabalan, and A. El-Hajj, “8085 Microprocessor simulation tool ‘8085 SimuKit,’” Comput. Appl. Eng. Educ., 2004, doi: 10.1002/cae.20022.

[5] V. A. F. Lamme, K. Zipser, and H. Spekreijse, “Masking interrupts figure-ground signals in V1,” J. Cogn. Neurosci., 2002, doi: 10.1162/089892902320474490.

[6] S. H. Chaudhry et al., “The determinants of maternal homocysteine in pregnancy: Findings from the Ottawa and Kingston Birth Cohort,” Public Health Nutr., 2020, doi: 10.1017/S1368980019004002.

[7] L. Liang, T. Melham, D. Kroening, P. Schrammel, and M. Tautschnig, "Effective verification for low-level software with competing interrupts," ACM Trans. Embed. Comput. Syst., 2017, doi: 10.1145/3147432.

[8] T. Krumpen et al., "Arctic warming interrupts the Transpolar Drift and affects long-range transport of sea ice and ice-rafted matter," Sci. Rep., 2019, doi: 10.1038/s41598-019-41456-y.

[9] D. J. Hobson, W. Wei, L. M. Steinmetz, and J. Q. Svejstrup, "RNA Polymerase II Collision Interrupts Convergent Transcription," Mol. Cell, 2012, doi: 10.1016/j.molcel.2012.08.027.

[10] M. J. Player et al., "What interrupts suicide attempts in men: A qualitative study," PLoS One, 2015, doi: 10.1371/journal.pone.0128180.

CHAPTER 4
INTERFACING MEMORY AND I/O DEVICES WITH 8085

Dr. M.Chandra Sekhar, Professor and Hod,
Department of Computer Science and Engineering,
Presidency University, Bangalore, Karnataka, India.
Email Id- mchandrasekhar@presidencyuniversity.in

ABSTRACT:

The work presents here outline the development of 8085 microprocessor-based system, specifically designed for education purpose. 8085 is an excellent teaching material to teach fundamental of microprocessor-based system design. The 8085 is in the core of the entire system. In our system Demultiplexing is achieved with the help of D- type latch integrated circuit, which provides the separate data lines and address line. The interfacing of memory chip is done with the microprocessor to provide the necessary instruction set to be implement on the system. The output port is designed with the combinational logic and interfaced it with the entire system to gain the necessary output.

KEYWORDS:

Communication, Electronics, Microprocessor, I.O Devices, 8085 Interrupts.

INTRODUCTION

The programs and data that are executed by the microprocessor have to be stored inROM/EPROM and RAM, which are basically semiconductor memory chips. The programs and data that are stored in ROM/EPROM are not erased even when power supply to the chip is removed. Hence, they are called non-volatile memory. They can be used to store permanent programs. In a RAM, stored programs and data are erased when the power supply to the chip is removed. Hence, RAM is called volatile memory. RAM can be used to store programs and data that include, programs written during software development for a microprocessor-based system, program written when one is learning assembly language programming and data enter while testing these programs.

Input and output devices, which are interfaced with 8085, are essential in anymicroprocessor based system. They can be interfaced using two schemes: I/O mapped I/O and memory-mapped I/O. In the I/O mapped I/O scheme, the I/O devices are treateddifferently from memory. In the memory-mapped I/O scheme, each I/O device is assumedto be a memory location.

Interfacing Memory Chips With 8085

8085 has 16 address lines (A0 - A15), hence a maximum of 64 KB (= 2^{16} bytes) of memory locations can be interfaced with it. The memory address space of the 8085 takes values from0000H to FFFFH. The 8085 initiates set of signals such as IO/M, RD and WR when it wants to read from

and write into memory. Similarly, each memory chip has signals such as CE or CS (chip enable or chip select), OE or RD (output enable or read) and WE or WR (write enable or write) associated with it [1].

Generation of Control Signals for Memory

When the 8085 wants to read from and write into memory, it activates IO/M, RD and WRyy signals as shown in Table 1.

Table 1: Represented that the Status of IO/M, RD and WR Signals during Memory Read and Write Operations.

IO/M	RD	WR	Operation
0	0	1	8085 reads data from memory
0	1	0	8085 writes data from memory

Using IO/M, RD and Wr signals, two control signals MEMR (memory read) and MEMW (memory write) are generated. Figure 1 shows the circuit used to generate these signals.

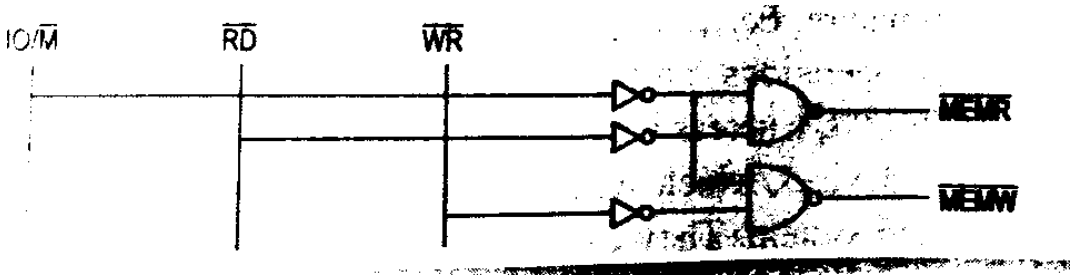


Figure 1: Represented that the Circuit used to Generate MEMR and MEMW Signals.

When is IO/M high, both memory control signals are deactivated irrespective of the status of RD and WR signals.

Ex: Interface an IC 2764 with 8085 using NAND gate address decoder such that the address range allocated to the chip is 0000H – 1FFFH.

Specification of IC 2764

a. 8 KB (8 x 2^{10} byte) EPROM chip

b. 13 address lines (2^{13} bytes = 8 KB)Interfacing:

Address lines of IC are connected to the corresponding address lines of 8085.

Remaining address lines of 8085 are connected to address decoder formed usinglogic gates, the output of which is connected to the CE pin of IC.

c. Chip is enabled whenever the 8085 places an address allocated to EPROM chip inthe address bus. This is shown in Figure 2.

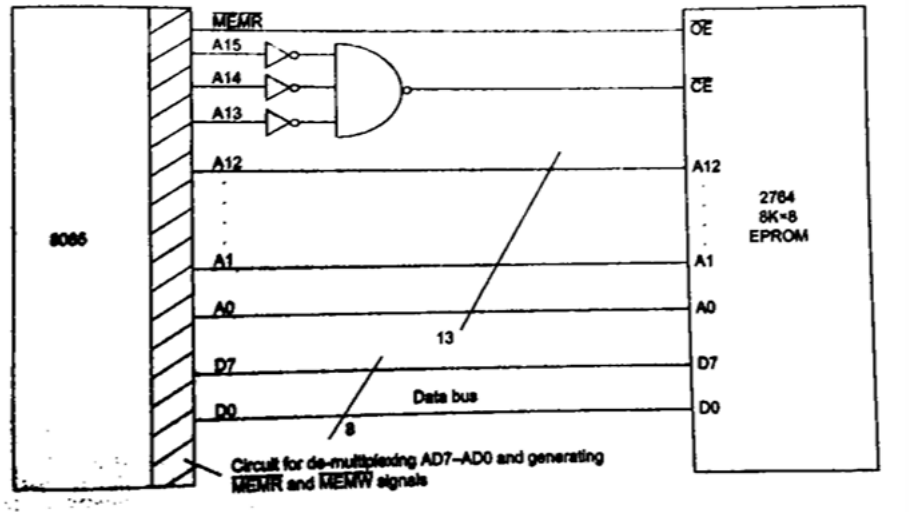


Figure 2: Represented that the Interfacing IC 2764 with the 8085.

d. Address range allocated to the chip is shown in Table 2.

Table 2: Represented that the Address allocated to IC 2764.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001H
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1FFEH
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH

Ex: Interface a 6264 IC (8K x 8 RAM) with the 8085 using NAND gate decoder such that the starting address assigned to the chip is 4000H.

Specification of ic 6264:

- **i.** 8K x 8 RAM
- **ii.** 8 KB = 2^{13} bytes
- **iii.** 13 address lines

The ending address of the chip is 5FFFH (since 4000H + 1FFFH = 5FFFH). When the address 4000H to 5FFFH are written in binary form, the values in the lines A15, A14, A13 are 0, 1 and 0 respectively. The NAND gate is designed such that when the lines A15 and A13 carry 0 and A14 carries 1, the output of the NAND gate is 0. The NAND gate output is in turn connected to the CE1 pin of the RAM chip. A NAND output of 0 selects the RAM chip for read or write operation, since CE2 is already 1 because of its connection to +5V. Figure 3, shows the interfacing of IC 6264 with the 8085 [2].

Ex: Interface two 6116 ICs with the 8085 using 74LS138 decoder such that the starting addresses assigned to them are 8000H and 9000H, respectively.

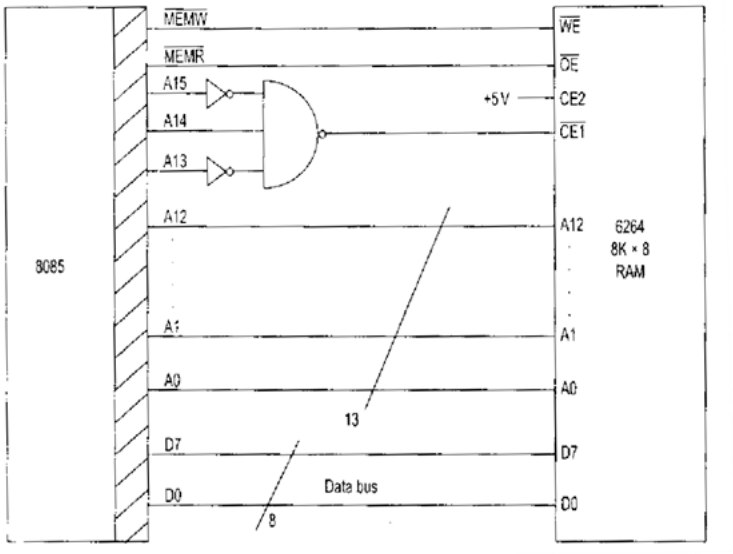


Figure 3: Represented that the Interfacing 6264 IC with the 8085.

Specification of IC 6116:

i. 2 K x 8 RAM

ii. 2 KB = 2^{11} bytes

iii. 11 address lines

iv. 6116 has 11 address lines and since 2 KB, therefore ending addresses of 6116 chip 1 is and chip 2 are 87FFH and 97FFH, respectively shown in Figure 4. Table 3 shows the address range of the two chips.

Table 3: Represented that the Address Range for IC 6116.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Address
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000H
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	87FFH (RAM chip 1)
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	9000H
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	97FFH (RAM chip 2)

Interfacing

i. A0 – A10 lines of 8085 are connected to 11 address lines of the RAM chips.

ii. Three address lines of 8085 having specific value for a particular RAM are connected to the three select inputs (C, B and A) of 74LS138 decoder.

iii. Shows that A13=A12=A11=0 for the address assigned to RAM 1 and A13=0, A12=1 and A11=0 for the address assigned to RAM 2.

iv. Remaining lines of 8085 which are constant for the address range assigned to the two RAM are connected to the enable inputs of decoder.

v. When 8085 places any address between 8000H and 87FFH in the address bus, the select inputs C, B and A of the decoder are all 0. The Y0 output of the decoder is also 0, selecting RAM 1.

vi. When 8085 places any address between 9000H and 97FFH in the address bus, the select inputs C, B and A of the decoder are 0, 1 and 0. The Y2 output of the decoder is also 0, selecting RAM 2 [3].

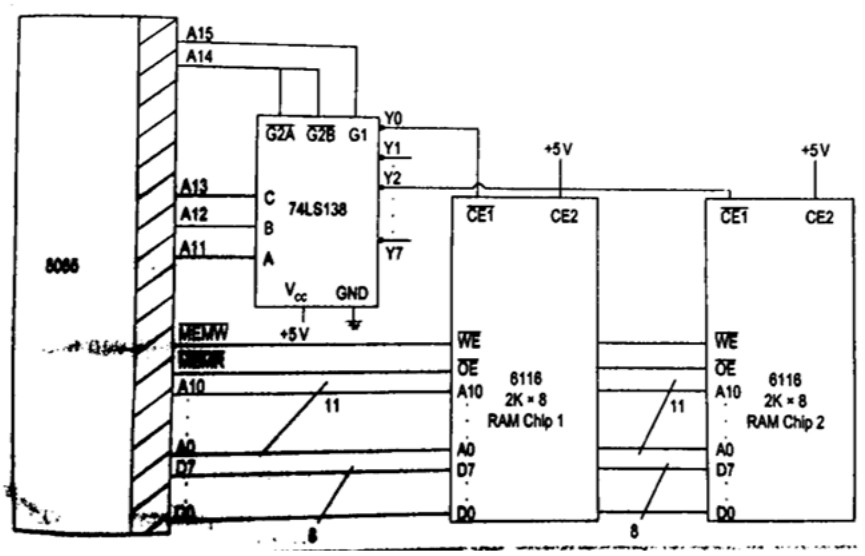


Figure 4: Represented that the Interfacing two 6116 RAM chips using 74LS138 decoder

Peripheral Mapped I/O Interfacing

In this method, the I/O devices are treated differently from memory chips. The control signals I/O read (IOR) and I/O write (IOW), [4] which are derived from the IO/M , RD and WR signals of the 8085, are used to activate input and output devices, respectively. Generation of these control signals is shown in Figure 5. Table 4 shows the status of IO/M, RD and WR signals during I/O read and I/O write operation.

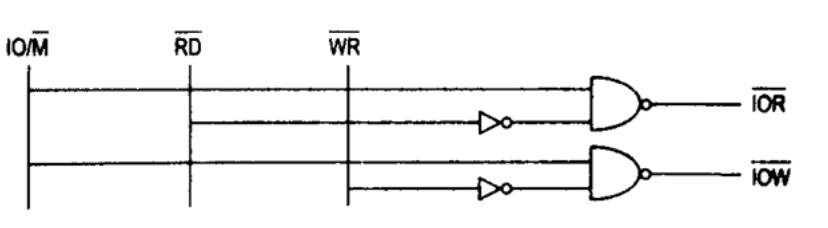


Figure 5: Represented that the Generation of IOR and IOW Signals.

IN instruction is used to access input device and OUT instruction is used to access output device. Each I/O device is identified by a unique 8-bit address assigned to it. Since the control signals used to access input and output devices are different, and all I/O device use 8-bit address, a maximum of 256 (2^8) input devices and 256 output devices can be interfaced with 8085 [5].

Table 4: Display that the Status of IOR and IOW signals in 8085.

IO/M	RD	WR	IOR	IOW	Operation
1	0	1	0	1	I/O read operation
1	1	0	1	0	I/O write operation
0	X	X	1	1	Memory read or write operation

Ex: Interface an 8-bit DIP switch with the 8085 such that the address assigned to the DIP switch if F0H.

IN instruction is used to get data from DIP switch and store it in accumulator. Steps involved in the execution of this instruction are:

i. Address F0H is placed in the lines A0 – A7 and a copy of it in lines A8 – A15.

ii. The IOR signal is activated (IOR = 0), which makes the selected input device to place its data in the data bus.

iii. The data in the data bus is read and store in the accumulator. Figure 6 shows the interfacing of DIP switch.

A7	A6	A5	A4	A3	A2	A1	A0	
1	1	1	1	0	0	0	0	= F0H

A0 – A7 lines are connected to a NAND gate decoder such that the output of NAND gate is 0. The output of NAND gate is ORed with the IOR signal and the output of OR gate is connected to 1G and 2G of the 74LS244. When 74LS244 is enabled, data from the DIP switch is placed on

the data bus of the 8085. The 8085 read data and store in the accumulator. Thus data from DIP switch is transferred to the accumulator [6].

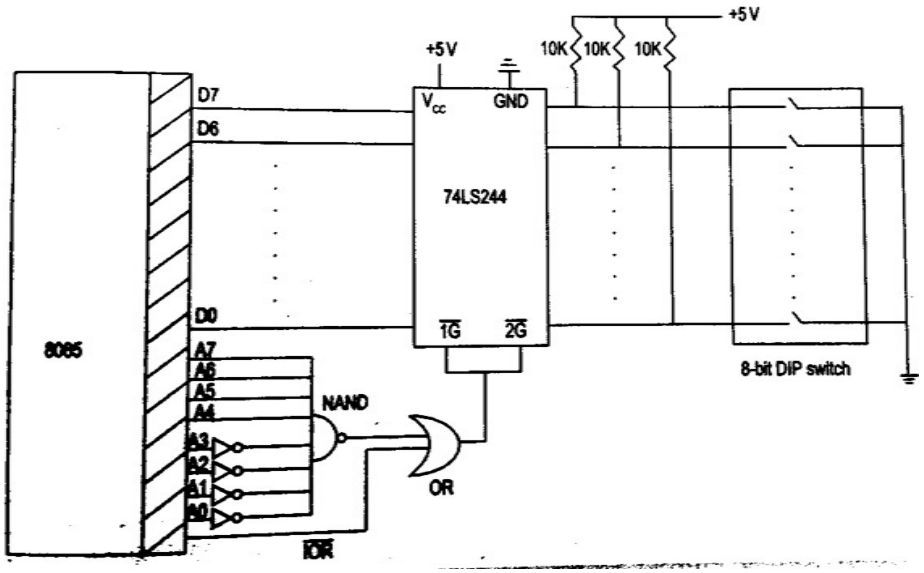


Figure 6: Represented that the Interfacing of 8-bit DIP Switch with 8085.

Memory Mapped I/O Interfacing

In memory-mapped I/O, each input or output device is treated as if it is a memory location. The MEMR and MEMW control signals are used to activate the devices. Each input or output device is identified by unique 16-bit address, similar to 16-bit address assigned to memory location. All memory related instruction like LDA 2000H, LDAX B, MOV A, M can be used. Since the I/O devices use some of the memory address space of 8085, the maximum memory capacity is lesser than 64 KB in this method[7].

Ex: Interface an 8-bit DIP switch with the 8085 using logic gates such that the address assigned to it is F0F0H.

Since a 16-bit address has to be assigned to a DIP switch, the memory-mapped I/O technique must be used. Using LDA F0F0H instruction, the data from the 8-bit DIP switch can be transferred to the accumulator. The steps involved are:

i. The address F0F0H is placed in the address bus A0 – A15.

ii. The MEMR signal is made low for some time.

iii. The data in the data bus is read and stored in the accumulator.Figure 7, shows the interfacing diagram.

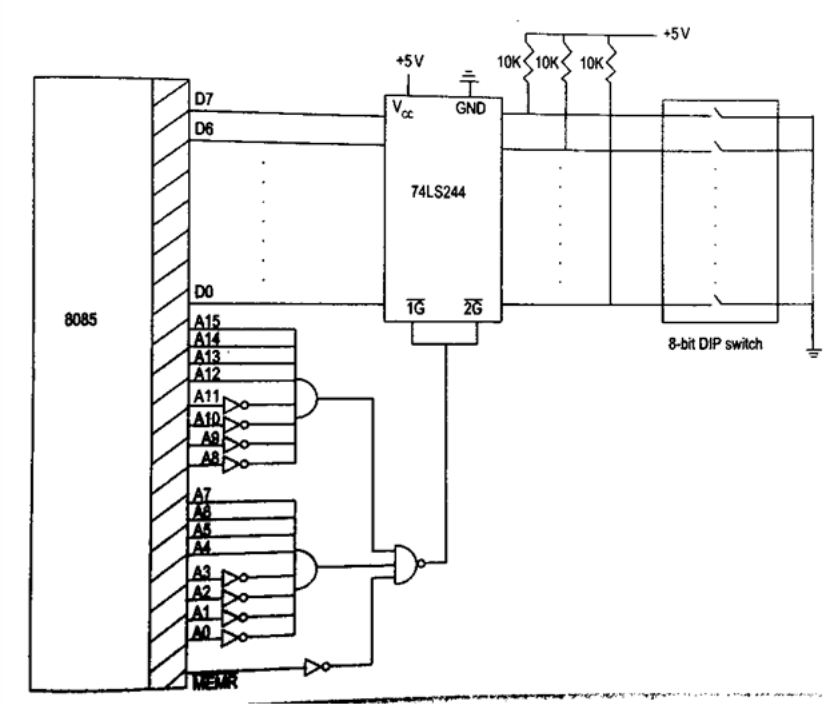


Figure 7: Represented that the 22 Interfacing 8-bit DIP switch with 8085.

When 8085 executes the instruction LDA F0F0H, it places the address F0F0H in the address lines A0 – A15 as:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	= F0F0H

The address lines are connected to AND gates. The output of these gates along with MEMR signal are connected to a NAND gate, so that when the address F0F0H is placed in the address bus and MEMR = 0 its output becomes 0, thereby enabling the buffer 74LS244.The data from the DIP switch is placed in the 8085 data bus. The 8085 reads the data from the data bus and stores it in the accumulator[8], [9].

DISCUSSION

The microprocessor 8085 was developed by Intel Corporation in 1976-77. It can able to operate on 3 MHz to 5 MHz clock frequency. Which is actually very less compared to today's advance high performance processors. But, then after 8085 is widely used in tiny embedded systems and it is excellent teaching material available. These is because of its very simple architecture and adequate instruction set. We have chosen the concept of implementation of 8085 system to improve the understanding of the students regarding the 8085 microprocessor based system design, interfacing concepts and the fundamental of programming knowledge. It has been observed that the designing of microprocessor based system is vital in improvement of practical knowledge regarding the microprocessor and its overall working. Our work focuses on the same. As described earlier the 8085 microprocessor is in the heart of the system. As the architecture of the 8085 suggest we need to separate out the address line as well as data line of our microprocessor. The system

must have a device to store a program, in this case we have used EPROM. We have followed the way to manually program the memory chip. On execution, the instructions set which is, placed in memory is able to send the data word from the accumulator register to the designed output port. In this paper we describe the development of the output port which is specifically designed to work with the 8085 microprocessor and implementation of the output port with the 8085[10].

CONCLUSION

The 8085-microprocessor based system has been developed to address the issues arise in the practical implementation of the memory interfacing with the processor. The developed system gives the complete idea about interfacing of memory and output port with the 8085 microprocessors. The system is capable to fetch the instruction from the memory and after execution of the instruction the specified task to send the data word on the output port FFH is performed. The developed system plays the major role in understanding of microprocessor-based systems. It can be an excellent teaching material to teach fundamentals of microprocessor-based system design.

REFERENCES

[1] M. Narayana Moorthi and R. Manjula, "A new integrated computer architecture simulation framework towards next generation approach," Int. J. Appl. Eng. Res., 2015.

[2] V. L. Patil, "Parallel I/O options for 8085-based systems," Microprocess. Microsyst., 1984, doi: 10.1016/0141-9331(84)90121-2.

[3] J. Shuja et al., "A Survey of Mobile Device Virtualization," ACM Comput. Surv., 2017, doi: 10.1145/2897164.

[4] G. W. Trott, A. G. Brenton, and J. H. Beynon, "OBTAINING A VOLTAGE FROM IEEE-488 BUS-EQUIPPED INSTRUMENTS.," IEEE Trans. Ind. Electron., 1984.

[5] J. Shuja et al., "Survey of mobile device virtualization: Taxonomy and state of the art," ACM Comput. Surv., 2016, doi: 10.1145/2897164.

[6] W. S. Kim, W. T. Corbett, R. K. Treece, A. E. Giddings, and J. L. Jorgensen, "Radiation-hard design principles utilized in cmos 8085 microprocessor family," IEEE Trans. Nucl. Sci., 1983, doi: 10.1109/TNS.1983.4333113.

[7] C. Clausen et al., "A source of polarization-entangled photon pairs interfacing quantum memories with telecom photons," New J. Phys., 2014, doi: 10.1088/1367-2630/16/9/093058.

[8] X. L. Pang et al., "A hybrid quantum memory–enabled network at room temperature," Sci. Adv., 2020, doi: 10.1126/sciadv.aax1425.

[9] C. Ko et al., "Ferroelectrically gated atomically thin transition-metal dichalcogenides as nonvolatile memory," Adv. Mater., 2016, doi: 10.1002/adma.201504779.

[10] D. Berco and D. Shenp Ang, "Recent Progress in Synaptic Devices Paving the Way toward an Artificial Cogni-Retina for Bionic and Machine Vision," Adv. Intell. Syst., 2019, doi: 10.1002/aisy.201900012.

CHAPTER 5
AN OVERVIEW OF INTEL 8255 PROGRAMMABLE PERIPHERAL INTERFACE

Dr. G. Shanmugarathinam, Professor and Hod,
Department of Computer Science and Engineering,
Presidency University, Bangalore, Karnataka, India.
Email Id- shanmugarathinam@presidencyuniversity.in

ABSTRACT:

The work presents here outline the development of 8085 microprocessor-based system, specifically designed for education purpose. 8085 is an excellent teaching material to teach fundamental of microprocessor-based system design. The 8085 is in the core of the entire system. In our system De-multiplexing is achieved with the help of D- type latch integrated circuit, which provides the separate data lines and address line. The interfacing of memory chip is done with the microprocessor to provide the necessary instruction set to be implement on the system. The output port is designed with the combinational logic and interfaced it with the entire system to gain the necessary output.

KEYWORDS:

Communication, Electronics, Microprocessor, I.O Devices, 8085 Interrupts.

INTRODUCTION

The 8255A is a general purpose programmable I/O device designed for use with Intel microprocessors. It consists of three 8-bit bidirectional I/O ports (24I/O lines) that can be configured to meet different system I/O needs. The three ports are PORT A, PORT B & PORT C. Port A contains one 8-bit output latch/buffer and one 8-bit input buffer. Port B is same as PORT A or PORT B. However, PORT C can be split into two parts PORT C lower (PC0-PC3) and PORT C upper (PC7-PC4) by the control word. The three ports are divided in two groups Group A (PORT A and upper PORT C) Group B (PORT B and lower PORT C). The two groups can be programmed in three different modes. In the first mode (mode 0), each group may be programmed in either input mode or output mode (PORT A, PORT B, PORT C lower, PORT C upper). In mode 1, the second's mode, each group may be programmed to have 8-lines of input or output (PORT A or PORT B) of the remaining 4-lines (PORT C lower or PORT C upper) 3-lines are used for hand shaking and interrupt control signals. The third mode of operation (mode 2) is a bidirectional bus mode which uses 8-line (PORT A only for a bidirectional bus and five lines (PORT C upper 4 lines and borrowing one from other group) for handshaking. The 8255 is contained in a 40-pin package, whose pin out is shown in Figure 1 below[1]:

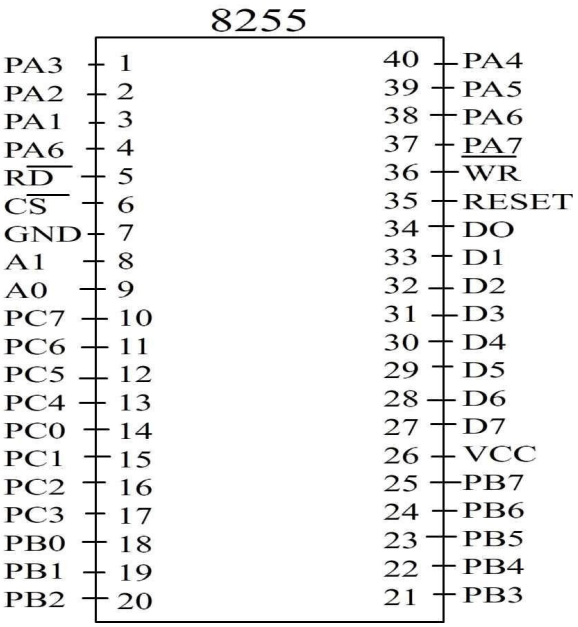


Figure 1: Represented that the 40-pin Diagram of MP-8255.

PIN Names

i. RESET: Reset input

ii. $\overline{CS}$: Chip selected

iii. $\overline{RD}$: Read input $\overline{WR}$ - Write input

iv. A_0 A_1 :Port Address$PA_7 – PA_0$ – PORT A$PB_7 – PB_0$ – PORT B$PC_7 – PC_0$ – PORT C
VCC - +5v

v. GND - Ground

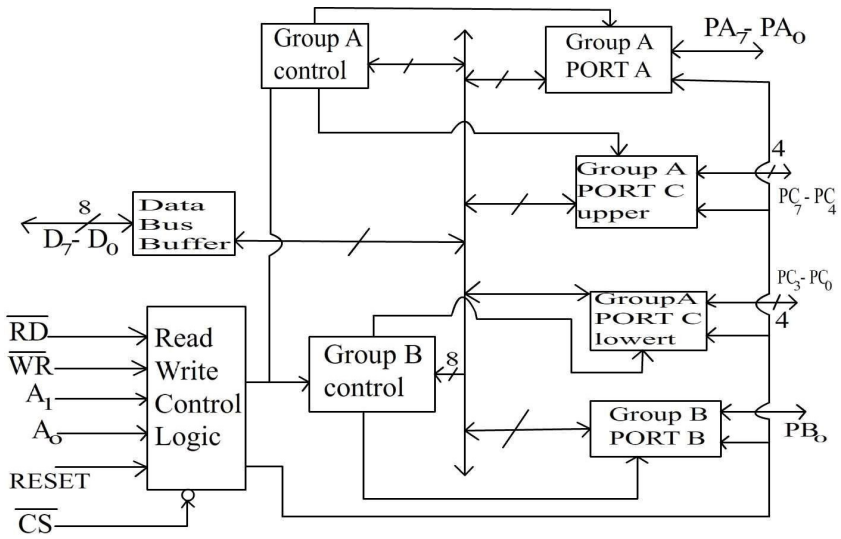


Figure 2: Represented that the Block Diagram of Microprocessor 8255.

Functional Description

This support chip is a general purpose I/O component to interface peripheral equipment to the microcomputer system bus which is mention in Figure 2. It is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

It is a tri-state 8-bit buffer used to interface the chip to the system data bus. Data is transmitted or received by the buffer upon executionof input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer. The data lines are connected to BDB of p.

Read/Write and Logic Control

The function of this block is to control the internal operation of thedevice and to control the transfer of data and control or status words.It accepts inputs from the CPU address and control buses and in turn issues command to both the control groups.

Chip Select

A low on this input selects the chip and enables the communication between the 8255 A & the CPU. It is connected to the output of address decode circuitry to select the device when it (Read). A low on this input enables the 8255 to send the data or status information to the CPU on the data bus.

Write

A low on this input pin enables the CPU to write data or control wordsinto the 8255 A.

A_1, A_0 Port Select

These input signals, in conjunction with the $\overline{RD}$ and $\overline{WR}$ inputs, control the selection of one of the three ports or the control word registers. They are normally connected to the least significant bits of the address bus (A_0 and A_1). Following Table 1 gives the basic operation, all other states put data bus into tri-state/illegal condition [2].

Table 1: Represented that the basic operation, all Other States put Data Bus into tri-state/illegal Condition

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Input operation
0	0	0	1	0	PORT A→ Data bus
0	1	0	1	0	PORT B→ Data bus
1	0	0	1	0	PORT C→ Data bus

0	0	1	0	0	Output operation Data bus→	PORT A
0	1	1	0	0	Data bus→	PORT B
1	0	1	0	0	Data bus→	PORT C
1	1	1	0	0	Data bus→	control

RESET

A high on this input pin clears the control register and all ports (A, B &C) are initialized to input mode. This is connected to RESET OUT of 8255. This is done to prevent destruction of circuitry connected toport lines. If port lines are initialized as output after a power up or reset, the port might try to output into the output of a device connected to same inputs might destroy one or both of them.

PORTs A, B and C

The 8255A contains three 8-bit ports (A, B and C). All can be configured in a variety of functional characteristic by the system software.

PORT 'A'

One 8-bit data output latch/buffer and one 8-bit data input latch.

PORT 'B'

One 8-bit data output latch/buffer and one 8-bit data input buffer.

PORT 'C'

One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signals inputs in conjunction with ports A and B.

Group A & Group B Control

The functional configuration of each port is programmed by the system software. The control words outputted by the CPU configure the associated ports of the each of the two groups. Each control block accepts command from Read/Write content logic receives control words from the internal data bus and issues proper commands to its associated ports.

Control Group A – Port A & Port C upper Control Group B – Port B & Port C lower

The control word register can only be written into No read operation if the control word register is allowed.

Operation Description

Mode Selection

There are three basic modes of operation that can be selected by the system software.

Mode 0: Basic Input/output

Mode 1: Strobes Input/output

Mode 2: Bi-direction bus.

When the reset input goes HIGH all poets are set to mode'0' as input which means all 24 lines are in high impedance state and can be used as normal input. After the reset is removed the 8255A remains in the input mode with no additional initialization. During the execution of the program any of the other modes may be selected using a single output instruction[3].

The modes for PORT 'A' & PORT 'B' can be separately defined, while PORT 'C' is divided into two portions as required by the PORT 'A' and PORT 'B' definitions. The ports are thus divided into two groups Group 'A' & Group 'B'. All the output register, including the status flip-flop will be reset whenever the mode is changed. Modes of the two group may be combined for any desired I/O operation e.g. Group 'A' in mode '1' and group 'B' in mode '0'. The basic mode definitions with bus interface and the mode definition format are given in Figure 3(a) & 3(b),

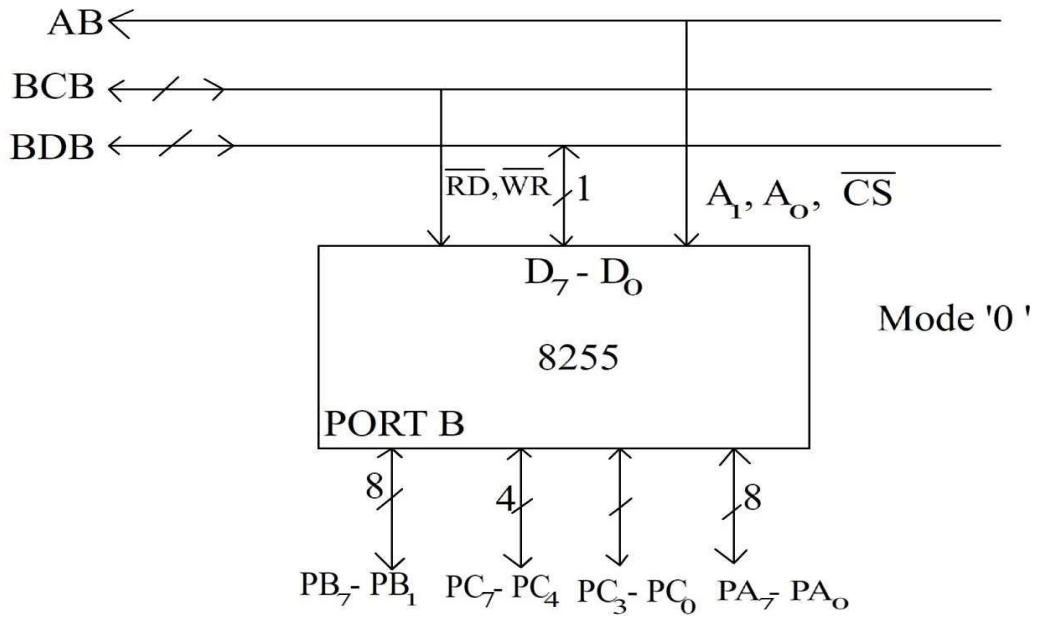


Figure 3(a)

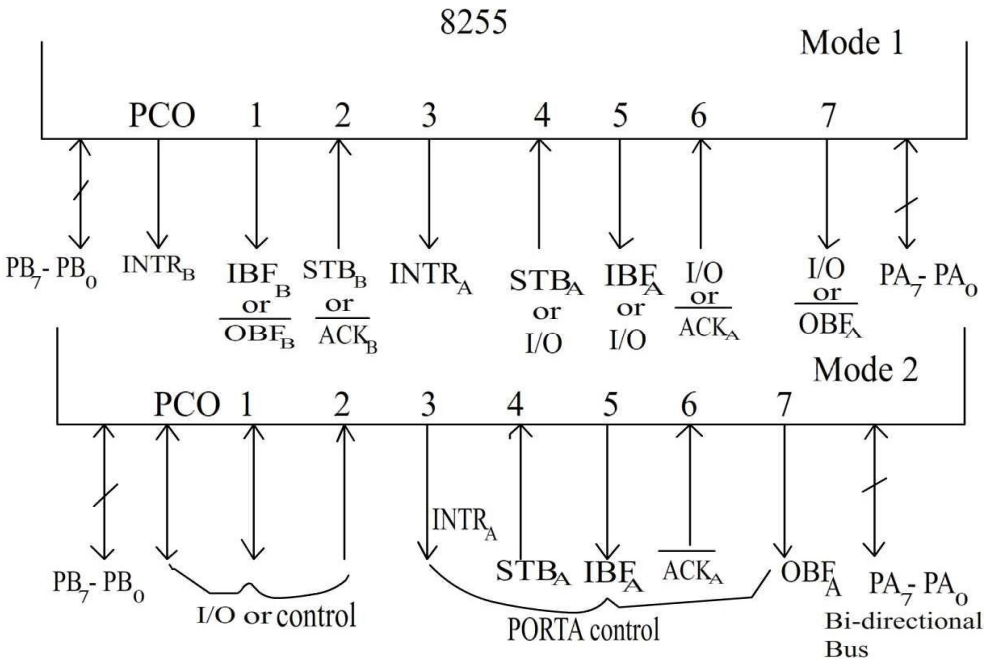


Figure 3(b)

INTEL 8259A Programmable Interrupt Controller

The 8259A is a programmable interrupt controller designed to work with Intel microprocessor 8080 A, 8085, 8086, 8088. The 8259 A interrupt controller can:

i. Handle eight interrupt inputs. This is equivalent to providing eight interrupt pins on the processor in place of one INTR/INT pin.

ii. Vector an interrupt request anywhere in the memory map. However, all the eight interrupt are spaced at the interval of either four or eight location. This eliminate the major drawback, 8085 interrupt, in which all interrupts are vectored to memory location on page 00H.

iii. Resolve eight levels of interrupt priorities in a variety of modes.

iv. Mask each interrupt request individually.

v. Read the status of pending interrupts, in service interrupts, and masked interrupts.

vi. Be set up to accept either the level triggered or edge triggered interrupt request.

vii. Mine 8259 as can be cascade in a master slave configuration to handle 64 interrupt inputs.

The 8259 A is contained in a 28-element in line package that requires only a compatible with 8259. The main difference between the two is that the 8259 A can be used with Intel 8086/8088 processor. It also induces additional features such as level triggered mode, buffered mode and automatic end of interrupt mode. [4] The pin diagram andinterval block diagram is shown in Figure 4:

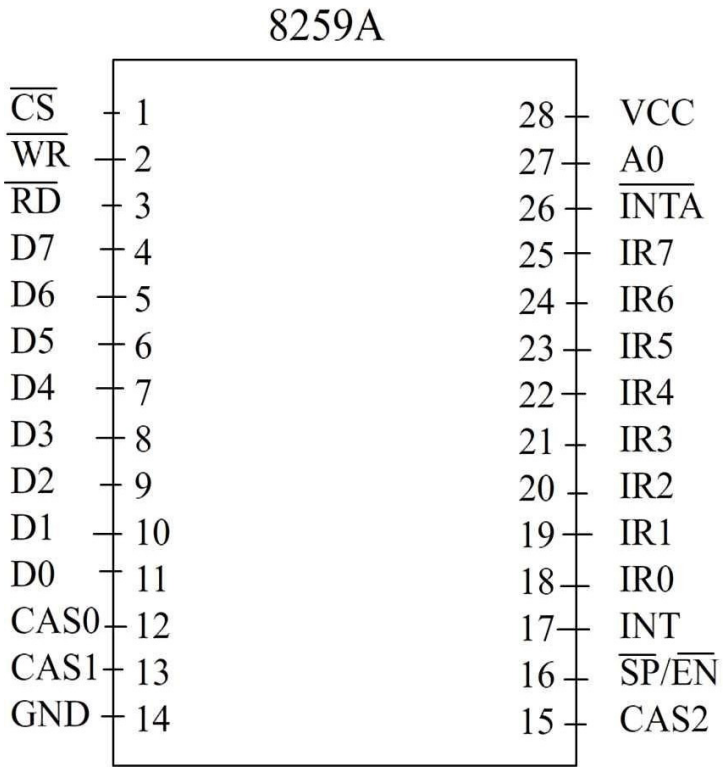


Figure 4: Represented that the Pin Diagram of 8259A.

i. Data Bus Buffer

This Block is used as a mediator between 8259 and 8085/8086 microprocessor by acting as a buffer. It takes the control word from the 8085 (let say) microprocessor and transfer it to the control logic of 8259 microprocessor. After selection of Interrupt by 8259 microprocessor (based on priority of the interrupt), it transfer the opcode of the selected Interrupt and address of the Interrupt service sub routine to the other connected microprocessor. The data bus buffer consists of 8 bits represented as D0-D7 in the block diagram. Thus, shows that a maximum of 8 bits data can be transferred at a time.

ii. Read/Write Logic

This block works only when the value of pin CS is low (as this pin is active low). This block is responsible for the flow of data depending upon the inputs of RD and WR. These two pins are active low pins used for read and write operations.

iii. **Control Logic**

It is the center of the PIC and controls the functioning of every block. It has pin INTR which is connected with other microprocessor for taking interrupt request and pin INT for giving the output. If 8259 is enabled, and the other microprocessor Interrupt flag is high then this causes the value of the output INT pin high and in this way 8259 responds to the request made by other microprocessor.

iv. Interrupt Request Register (IRR)

It stores all the interrupt level which are requesting for Interrupt services.

v. Interrupt Service Register (ISR)

It stores the interrupt level which are currently being executed.

vi. Interrupt Mask Register (IMR)

It stores the interrupt level which have to be masked by storing the masking bits of the interrupt level.

vii. Priority Resolver

It examines all the three registers and set the priority of interrupts and according to the priority of the interrupts, interrupt with highest priority is set in ISR register. Also, it reset the interrupt level which is already been serviced in IRR.

viii. Cascade Buffer

To increase the Interrupt handling capability, we can further cascade more number of pins by using cascade buffer. So, during increment of interrupt capability, CSA lines are used to control multiple interrupt structure.

SP/EN (Slave program/Enable buffer) pin is when set to high, works in master mode else in slave mode. In Non-Buffered mode, SP/EN pin is used to specify whether 8259 work as master or slave and in Buffered mode, SP/EN pin is used as an output to enable data bus [5], [6].

DISCUSSION

Microcomputer system design requires that I.O devices such as keyboards, displays, sensors and other components receive servicing in a an efficient manner so that large amounts of the total system tasks can be assumed by the microcomputer with little or no effect on throughput. The most common method of servicing such devices is the Polled approach. This is where the processor must test each device in sequence and in effect ‘‘ask’’ each one if it needs servicing. It is easy to see that a large portion of the main program is looping through this continuous polling cycle and that such a method would have a serious detrimental effect on system throughput, thus limiting the tasks that could be assumed by the microcomputer and reducing the cost effectiveness of using such devices [7]. A more desirable method would be one that would allow the microprocessor to be executing its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is complete, however, the processor would resume exactly where it left off. This method is called Interrupt. It is easy to see that system throughput would drastically increase, and thus more tasks could be assumed by the microcomputer to further enhance its cost effectiveness. The Programmable Interrupt Controller (PIC) functions as an overall manager in an Interrupt-Driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination. Each peripheral device or structure usually has a special program or

‘‘routine’’ that is associated with its specific functional or operational requirements; this is referred to as a ‘‘service routine’’. The PIC, after issuing an Interrupt to the CPU, must somehow input information into the CPU that can ‘‘point’’ the Program Counter to the service routine associated with the requesting device. This ‘‘pointer’’ is an address in a vectoring table and will often be referred to, in this document, as vectoring data[8]–[10].

CONCLUSION

The microprocessor 8085 was developed by Intel Corporation in 1976-77. It can able to operate on 3 MHz to 5 MHz clock frequency. Which is actually very less compared to today’s advance high performance processors. But, then after 8085 is widely used in tiny embedded systems and it is excellent teaching material available. These is because of its very simple architecture and adequate instruction set. We have chosen the concept of implementation of 8085 system to improve the understanding of the students regarding the 8085 microprocessor based system design, interfacing concepts and the fundamental of programming knowledge. It has been observed that the designing of microprocessor based system is vital in improvement of practical knowledge regarding the microprocessor and its overall working. Our work focuses on the same. As described earlier the 8085 microprocessor is in the heart of the system. As the architecture of the 8085 suggest we need to separate out the address line as well as data line of our microprocessor. The system must have a device to store a program, in this case we have used EPROM. We have followed the way to manually program the memory chip. On execution, the instructions set which is, placed in memory is able to send the data word from the accumulator register to the designed output port. In this paper we describe the development of the output port which is specifically designed to work with the 8085 microprocessor and implementation of the output port with the 8085. To achieve this we need, to demultiplex the lower order address and data bus, control signal generation and memory interfacing. In section II, the Demultiplexing of lower order address and data bus is described. In section III, we focus on necessary control signal generation. These control signals are necessary to control memory read operation and the write operation from the system to the output port. In section IV, we describe the design of the entire system along with the design of the output port and its selection logic.

REFERENCES

[1] Y. W. Bai, H. G. Wei, C. Y. Lien, and H. L. Tu, “A windows-based dual-channel arbitrary signal generator,” Conf. Rec. - IEEE Instrum. Meas. Technol. Conf., 2002, doi: 10.1109/IMTC.2002.1007167.

[2] A. Zargari and M. S. Combs, “Construction, interfacing, and application of an 8255 - Based programmable peripheral interface card,” in Proceedings of the Electrical/Electronics Insulation Conference, 2001. doi: 10.1109/eeic.2001.965598.

[3] M. C. Ndinechi, “Computer interfacing for data acquisition and systems control,” Model. Meas. Control A, 2002.

[4] P. X. Rong and Z. J. He, “Design of numerical control system for stepper motor,” Dianji yu Kongzhi Xuebao/Electric Mach. Control, 2009.

[5] Y. W. Bai and H. G. Wei, “Design and implementation of a client-server remote windows-based signal generator,” Conf. Rec. - IEEE Instrum. Meas. Technol. Conf., 2001, doi: 10.1109/IMTC.2001.928791.

[6] J. T. Huang, S. Y. Hou, C. C. Lin, Y. J. Lai, and W. S. Chang, "Universal platform for developing an integrated biochip or micro-TAS based on electrokinetics," in Tamkang Journal of Science and Engineering, 2004.

[7] R. Ramnarine, "INTERFACE FOR A MICROPROCESSOR CONTROLLED FREQUENCY RESPONSE ANALYSER (FRA)," West Indian J. Eng., 1987.

[8] A. Siegel, B. Dumery, and V. Van Toi, "Portable blink rate recorder," Investig. Ophthalmol. Vis. Sci., 1997.

[9] Z. Sibin, Y. Zhengmao, and L. Guixian, "Modelling and simulation of high fidelity CDU for flight simulator based on object-oriented and hardware-in-loop technique," in Advanced Materials Research, 2010. doi: 10.4028/www.scientific.net/AMR.108-111.530.

[10] B. G. Thompson and A. F. Kuckes, "8255 Programmable Peripheral Interface data sheets," in IBM-PC in the Laboratory, 2010. doi: 10.1017/cbo9780511622885.014.

CHAPTER 6
AN INTRODUCTION OF 8085 MICROCONTROLLER

Zafar Ali Khan N, Associate Professor,
Department of Computer Science and Engineering,
Presidency University, Bangalore, Karnataka, India
Email Id- zafaralikhan@presidencyuniversity.in

ABSTRACT:

In the Electronics Technology course, students were directed to construct, troubleshoot, and interface an 8085-based microcontroller in order to control electromechanical circuits. This microprocessor, using Intel 8085A processor, was assembled from components andwas interfaced to take full control of a stepper motor's motion. In addition, the microprocessor'sapplication to generate and measure waveforms was examined. This manuscript attempts to describe the basic architecture of a microcontroller system, and examine its interfacing techniques as well as its applications in providing communication to the outside world.

KEYWORDS:

Computer System, Data Indicator, Electronics Device, Microprocessor, Microcontroller.

INTRODUCTION

The knowledge of computers and their applications has become a fundamental technical skill required of electronics engineering technology students. In order to provide students with an opportunity to develop microprocessor application skills, this course is focused on teaching programming and application of Motorola 68HC11-A8 microprocessor. Although students who have learned one type of microcontrollers in class can adapt without too much difficulty to other types, this usually takes a considerable time and effort and they may not afford the practice and learning of the function of other types of processors during their academic years of university study. On the other hand, the competitive labor market may require them to demonstrate skills on major types of microcontrollers such as Intel-based processors immediately after their graduation. In order to help students to develop a crosslinking skill so that they are able to work on at least two types of microcontrollers, i.e., Motorola and Intel, a senior project was integrated in the Computer Electronics course – EET 445. In this course, while a student is provided with the knowledge of Motorola microcontroller instruction set and interfacing in classroom and laboratory setting, he or she builds an 8085A-based microcomputer system as a major project from components [1].

8085a-Based Computer System

The 8085A microprocessor is a 40-pin semiconductor integrated circuit (IC) device. In the 8085A microprocessor, in addition to execution of the instruction set, the functions of clock generation, system bus control and interrupt priority selection are contained. The 8085A transfersdata on an 8-bit bi-directional 3-state bus (AD0-AD7) which is time multiplexed so as to also transmit the eight low-order address bits. The 8085A CPU generates control signals that can be used to select

appropriate external devices and functions to perform READ and WRITE operations and also to select memory or I/O ports. The 8085A can address up to 256 different I/Olocations. These addresses have the same numerical value (00-FF) as the first 256 memory addresses. They are distinguished by means of the IO/M output from the CPU.

The 8085A is provided with internal 8-bit and 16-bit registers. General-purpose registers B, C, D, E, H, L may be used as six 8-bit registers or as three 16-bit registers, interchangeably, depending on the instruction being performed. H-L pair functions as a data pointer to reference memory addresses that are either the sources or the destinations in a number of instructions [2]. A smaller number of instructions can use B-C or D-E for indirect addressing. The flag register contains five one-bit flags. These are carried, auxiliary carry, sign, zero and parity.

A simplified functional block diagram of the 8085A-based computer system. The 8085A functions as the central processor unit (CPU). The memory is in two parts: the read-only memory (ROM) and the random-access memory (RAM). The ROM device is the Intel 2816 integrated circuit which has a capacity of 2K bytes of memory, this memory is electronically erasable and data may be written to it at a slow rate by the CPU or manually by data and controlswitches. The RAM device is the Intel 8156 integrated circuit, which contains 256 bytes of memory. The 8156 also contains the input/output (I/O) section. This section consists of three ports: A, B and C. Ports A and B are 8-bit in length and port C is 6-bit long. The computer's built-in keyboard consisting of 16 data and 8 function keys can be used to input data to the I/O section. Outputs from the I/O section are directed to two 7-segment displays. In order to providethe ability for the computer to interface with an external device, the output of one of the two 8- bit I/O ports (Port B) is also directed to the 8 pins of a 16-pin integrated circuit connector [3].

Attached to the address-data bus are also the address-data bus logic indicators and the data/control switches. According ti Figure 2, the address-data bus logic indicators consist of 8 drive circuits each to drive a light-emitting diode (LED) to indicate jointly the logic state of the address-data bus. Thedata/control switches are used to write to or read from the ROM, RAM or I/O ports.

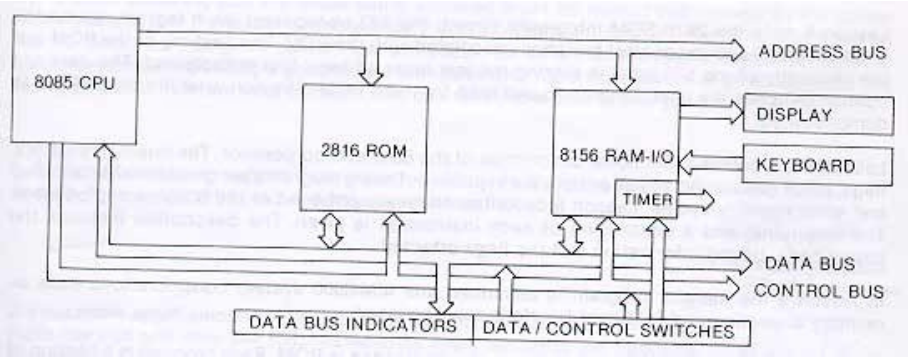


Figure 1: Represented that the Block Diagram 8085 Microcontroller.

Construction of the Power Distribution System

Student built the 8085A-based computer system on a pre-designed circuit board. The first step is to implement the power distribution system on to the circuit board. This was done by soldering the necessary capacitors and a voltage regulator on to the circuit board. Figure 2, is a simplified schematic of the computer system's power distribution system. Unregulated DC voltage in the

range of 12-13V is brought in through connector S3 to the power ON-OFF switch SW16. Switch SW16 applies this voltage to filter capacitors C3and C14 and to voltage regulator IC7. The 5V output of the regulator is filtered by capacitors C4 through C12. Capacitor C4 is a 100 micro farad capacitor to filter out low frequency noise such as 60 Hz and capacitors C5 through C12 filter out any high frequency noise which may be generated by the rapid switch operation. After completion of this part of the circuit, students are directed to identify the correct voltage at various voltage-feeding points on the circuit board [4].

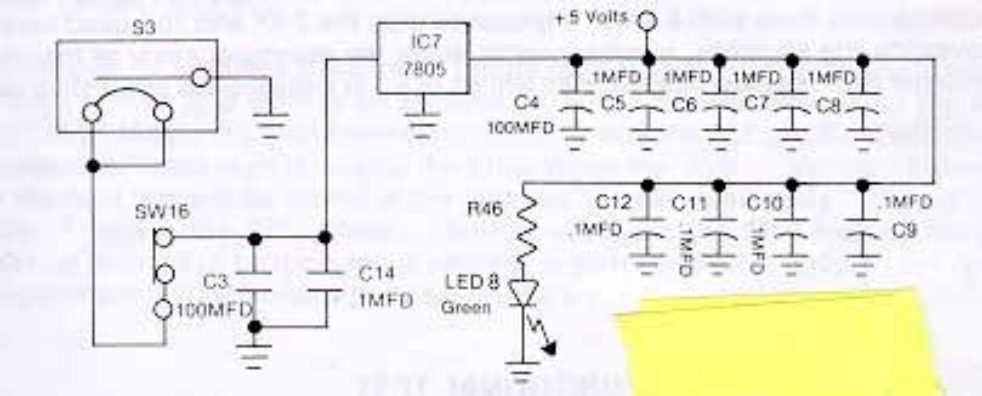


Figure 2: Represented that the Power Distribution System.

IMPLEMENTING OF THE ADDRESS-DATA INDICATOR CIRCUIT

To construct this part of the circuit, 8 SPDT switches and 8 LED have to be installed onto the circuit board. The 74HC04 Hex inverter consisting of 6 independent inverter circuits each driving one LED logic level indicator.

When a logic 1 is present on the data bus, a voltage of 3.5V is applied to the input of the inverter. This causes the output of the inverter to be a logic 0 corresponding to a voltage level of 0.2V from the ground. This causes a current of 2.5mA to flow through the corresponding LED. The condition of the LED to be lit is used to indicate a logic 1 level on the data line. When a 0 is present on the data bus, the input to the inverter circuit is 0.2V. The output of the inverter is 5V and no current can flow through the LED connected to this output. The LED remains dark indicating a 0 is present on the data line. The hex inverter thus provides a method of indicating the logic states on the address-data lines AD0-AD5. Since a NAND gate will be used later to make a de-bounce circuit, Two of the four NAND gates in a 74HC00 is used to drive the remaining 2 logic level indicators which are LED6 and LED7[5].

Implementing RAM

The Intel 8156 programmable interface adapter (PIA) consists both RAM and I/O on a single chip. Introducing this device onto the circuit board at this stage is to implement RAM. A 40-pin socket is first soldered onto the circuit board and the 8156 is inserted into the socket. Figure 3 shows the switch networks necessary for manual operation of the 8156 memory circuit. The IO/M switch is used to select the RAM or the input-output ports. For selecting RAM it should be in M position (switch down). A brief description of the functions of the lines each operated by a button (dimple switch) is given as follows in Figure 3:

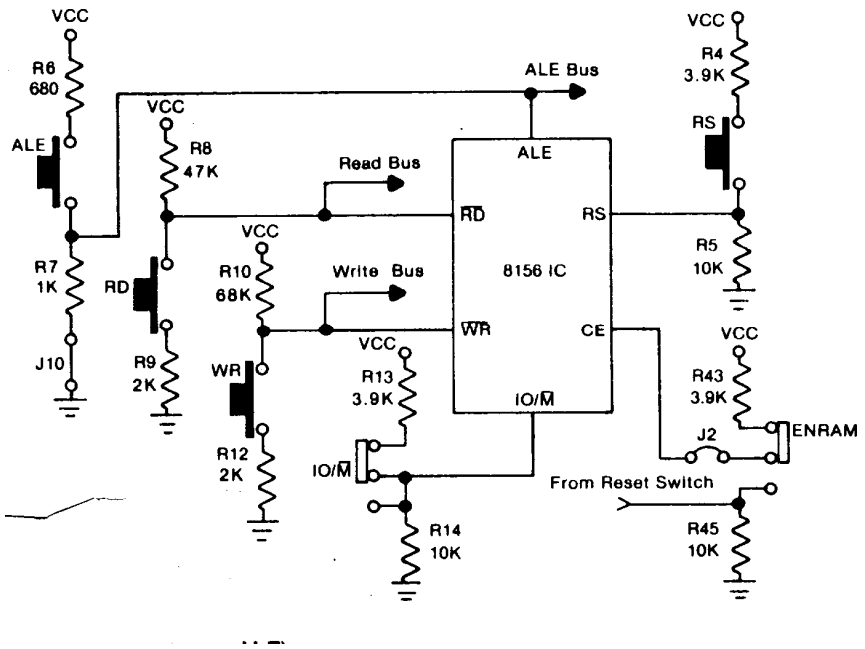


Figure 3: Represented that the 8156 Switch Network

Address Latch Enable (ALE)

The ALE line is normally held near ground potential (0V). When the ALE button is pressed, the line is brought to a high voltage by the voltage divider R6 and R7. The logic 1 condition of the ALE pin of the 8156 enable the RAM to store the data on the address-data bus as an address.

Read from Memory (RD)

The RD pin on the 8156 is held at a high voltage by resistor R8. When the RD button is pressed, the logic level on the RD pin becomes 0. This enables a read operation from the RAM. The data in the memory location at the latched address is reproduced on the 8 data lines. The IO/M. Switch must be in the M position in order to read from the RAM.

Write to Memory (WR)

When WR key is pressed, data on the data bus are written into the latched memory address. The IO/M Switch also must be in the M position in order to write to the memory [6].

Reset (RS)

This key is used to reset the 8156.

Implementing Input/Output and Display

When the IO/M switch is in the IO position, the 8156 passes the data on the data bus to input/output ports. Before data can be transmitted or received through the ports, the 8156 must be configured in terms of its input/output ports. Instructions are given to the 8156 by setting bit patterns in the Command Status Register. This register is located at address 0000 0000. Figure 4, shows how the Command Status Register operates.

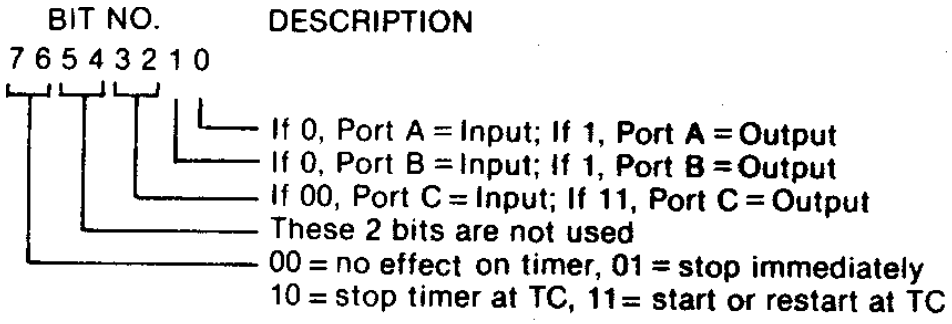


Figure 4: represented that the. Command Status Register

For example, if the binary pattern 1000 1110 is written to address location 0000 0000 when the IO/M switch is in the IO position, the 8156 will function as follows:

- The timer will stop when terminal count is reached
- Port A will be input port
- Both Port B and C will be output ports

After the ports are set, data are transmitted by first going to the address of the desired port. Next, data are output from the port pins and placed on the data bus if the port is configured as an outputport. If the port is configured as an input port, data are input into the port pins. [7] The address of each port is listed in Figure 5.

ADDRESS	NAME
7654 3210	
XXXX X000	Command Register
XXXX X001	Port A
XXXX X010	Port B
XXXX X011	Port C

X-Indicates don't care, can be 1 or 0

Figure 5: Represented that the Ports Addresses

In our construction, Both Display1 and Display2 are connected to Port B as illustrated in Figure 4C. However, they are time multiplexed and according to Figure 6, the multiplexing is controlled by bit 0 of Port C. To light any segment of the display, a logic 0 must be placed in the corresponding bit of Port B.

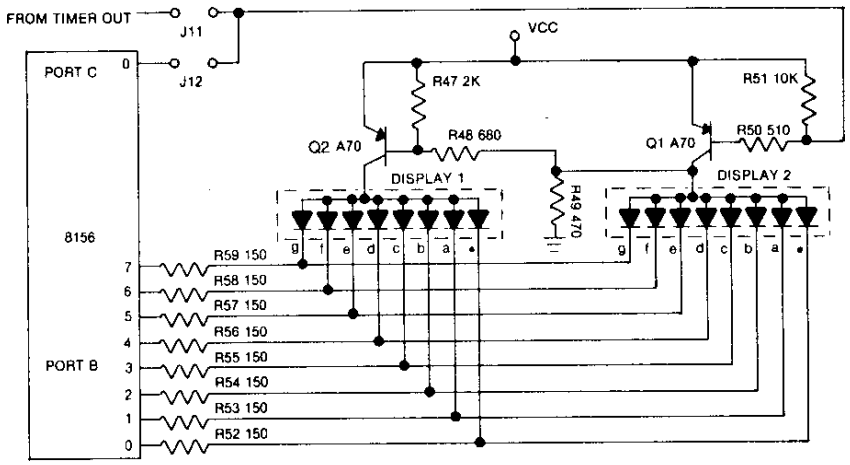


Figure 6. Display the Combination of 1 & 2.

IMPLEMENTING THE TIMER

The 8156 timer circuit is a 14-bit programmable counter. The counter is programmed by setting the two bytes of the Count Length Register as shown in Figure 7A. The Count Length Register bits T0 through T13 are counted down by TIMER IN pulses. When the terminal count is reached, the cycle is complete and a pulse or square wave is present at the TIMER OUT pin of the 8156 integrated circuit. Bits M1 and M2 of the Count Length Register specify the timer mode as shown in Figure 7B. The Count Length Register may be set to any value from 002H to 3FFFH (H stands for hexadecimal number).

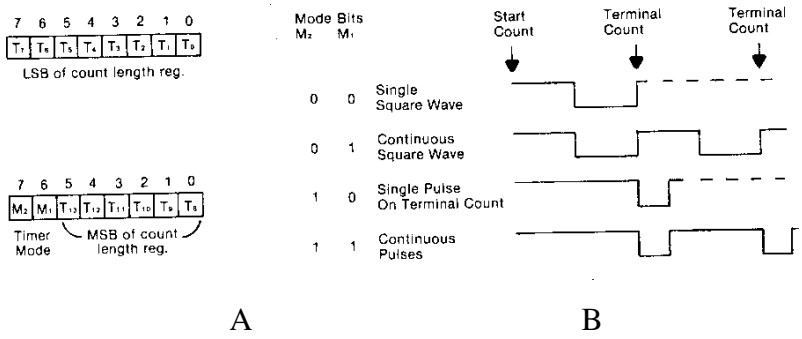


Figure 7: Represented that the Count Length Register.

The LSB (least significant Byte) of the Count Length Register is at I/O address 04H. The MSB (most significant byte) of the Count Length Register is at I/O address 05H. The start and stop of the counter is controlled by bits 6 and 7 (TM1 and TM2) of the CommandStatus Register. If the timer is not running when the Count length Register is loaded, it will remain in not running condition until a new start command is issued. If the timer is running when a new mode or count length is loaded into the Count Length Register, the timer will continue to run with the old mode or count length until a new start command is issued through the Command Status Register[8].

To clock the timer a low-to-high transition must be applied to the TIMER IN pin of the 8156. De-bounce is accomplished by tying the TIM switch to a flip-flop circuit as shown in Figure 5C. Both pins 12 and 13 of integrated circuit 74HC00 must be high to make pin 11 go low. With theTIM switch in the down position, a low is applied to pin 13causing pin 11 hence pin 10 to be high. Pin 9 is biased high by resister 39. With both pin 9 and 10 high, pin 8 will be low. This applies a low to pin 12 and the TIMER IN through jumper J3. The low on pin 12 keeps pin 11 high even if the low is removed from pin 13. The flip-flop is now in the logic low state.

Write Enable (WE)

This line serves the same purpose as the **WR** line of the 8156 to allow data to be write into 2816. Figure 6 is the schematic diagram of ROM This line serves the same purpose as the **RD** line of 8156 allowing data to be read from 2816. To clock the timer, the TIM switch is moved to the up position. This breaks the ground connection to pin 13 causing it to be biased high by resistor R40. Next, pin 9 is connected to ground causing pins 8 and 12 to go high. Since pin 13 is high, pin 11 hence pin 10 goes low which forces pin 8 to stay high even if pin 9 goes high and low several times due to switch de- bounce. The flip-fop is now in the logic high state and the low-to-high transition has been sent tothe TIMER IN pin of the 8156 to clock the TIMER.

Implementing ROM

As mentioned earlier, a 2816 EEPRM is used as the ROM. This is a 24-pin device. The write operation for this device requires 10 ms. The 2816 contains 2K memory locations, each capable of storing 1 byte data. The total memory capacity of 2816 is 16K.

Address Lines (A0-A10)

These 11 address lines tell the 2816 which of the 2K memory locations is to be written to or read from.

Input/Uutput Lines (I/O0- I/O7)

During a write operation, data on these lines are stored in the memory location specified by the address on the address lines A0-A10. During a read operation, the data specified by the address lines are placed on the I/O lines.

Output Enable (OE)

This line serves the same purpose as the **RD** line of 8156 allowing data to be read from 2816.

Write Enable (WE)

This line serves the same purpose as the **WR** line of the 8156 to allow data to be written into2816.

The schematic diagram of ROM circuit. The 74HCT573 is an Octal-D type transparentlatch which consist of 8 independent latches controlled by a control input at pin C. When C is high, the latches are transparent. A high on any of the inputs, D1-D8, will result in a high on the corresponding Q output. A low on a D input will likewise result in a low on the corresponding Q output. When C goes low, the data that was present in the latch is retained on the Q outputs regardless of any of the D input. The ALE bus is connected to the C input of the 74HCT573 integrated circuit. When the ALE key is pressed, the ALE bus goes high and the LSB of the address on the address-data bus is passed through the transparent latches to the address inputs

A0-A7 of the 2816. When the ALE button is released, the address is retained at the A0-A7 inputs. The address-data bus is then free to pass data to or from the 2816 I/O pins. Address lines A8, A9 and A10 are connected to the appropriate pins of the 2816 [9].

The Read bus is connected to the OE pin and is used to gate data from the 2816 onto the data bus. The WE input to the 2816 is tied to the top of the WEN switch. With the WEN switch in the up position, The WE input is connected to the write bus as shown in Figure. The WR button can then be used to write data into the 2816 memory. Pushing the WR button ties the WE input to a low voltage (logic 0) through the resistor network R10, R11 and R12. Releasing the WR button ties the WE input to a high voltage (logic 1) through resistors R10 and R11. If the WEN switch is in the down position, The WR button is disconnected from the WE input. The WE input is tied to high voltage (logic 1) through resistor R11 and writing is inhibited. To prevent accidental erasure of the contents in 2816, the WEN switch should always be in down position when power is turned ON or OFF.

The center of the Enable ROM (ENROM) switch is connected to the CE input of the 2816. With the switch in the up position, the CE is tied to a low voltage (logic 0). The 2816 will then respond to orders on the WE and OE lines. With the switch in the down position, the CE input istied to a high voltage (logic level 1) through resistor R44 and the 2816 is inhibited from responding to orders on the WE and OE lines.

DISCUSSION

The monitor program resides in ROM locations 0000-0003 and 0400-012D. RAM locations80DC-80FF are also used. All codes of the monitor program are entered manually via data/control switches with the address-data bus logic indicators as a visual aid. This a long and tedious process. However, students gain efficiency by practice. Further, they build up better understanding of the machine language programming. Once all codes are entered without error,the monitor program is ready for running. When the RESET switch is down, the 8085A comes up running the monitor program whenever the power is on. With the monitor program running, the keyboard is capable of entering data or new programs into memory. Any byte in memory may also be sent to the display. By using the GO button on the keyboard, control can be transferred to an application program [10].

There are three modes of operation, Data (DA), Address Low (AL), and Address High (AH). The active mode is determined by the code in the MODE L byte. The MODE L byte is set to the low order byte of the address of DDA which is FC for DA mode, to the low order byte of the address of DAL which is FD for AL mode, and to the low order byte of the address of DAH which is FE for AH mode. The MODE H byte always contains 80 which is the high order address byte of DDA, DAL and DAH. In data (DA) mode, the MODE H and MODE L bytes contain the address of the DDA byte which is at 80FC. The content of DDA is displayed in Display1 and Display2. The display operation is performed in the following way: When one of the keyboard data keys (0-F) is pressed, the data representing the key is entered into the low order digit of DDA anddisplayed on Display2. When a new key is pressed, the low order digit of DDA is shifted intohigh order and displayed on Display1, the new data becomes the low order digit displayed onDisplay2. No decimal point is lit indicating DDA byte is displayed[11], [12].

CONCLUSION

This project enables students to integrate classroom and laboratory knowledge with project-based learning. It has allowed students to develop advanced technical skills by cross-linking two sets of contemporary microprocessor technologies, i.e., Motorola and Intel, in a compact and efficient way. The development of such competencies is essential in order to functionsuccessfully in today's competitive electronic job market. Our preliminary assessment indicates that although the project is challenging, it is not overwhelming or beyond electronics technology students capabilities. It has helped students todevelop the troubleshooting and problem solving skills that are required in real life situations. This project has helped to demystify the inner workings of microcontroller technology and hasgenerated and enhanced the students' interest in further exploration of microprocessor technology. Upon completion of the project, the students demonstrate a sense of accomplishment, confidence, and self-satisfaction.

REFERENCES

[1] P. Kundu and A. Das, "Microprocessor-based system for identification of phase sequence and detection of phase unbalance of three-phase ac supply," J. Sci. Ind. Res. (India)., 2009.

[2] K. V. G. S. Kethan, K. Bhargavi, K. N. V. Satyanarayana, and G. Rambabu, "DTMF based Automatic Electricity Billing," Int. J. Eng. Res., 2016, doi: 10.17577/ijertv5is020606.

[3] C. S. Jawalkar, P. Kumar, and A. K. Sharma, "Parametric study while microchanneling on optical glass using microcontroller driven ECDM process," in Advanced Materials Research, 2012. doi: 10.4028/www.scientific.net/AMR.585.417.

[4] J. R. Li and A. Zagari, "Interfacing an 8085-based microcontroller: A practical approach to developing computer application skills," in ASEE Annual Conference Proceedings, 2001. doi: 10.18260/1-2--9454.

[5] E. Ali and W. Pora, "VHDL Implementation of ARM Cortex-M0 Laboratory for Graduate Engineering Students," in 2020 5th International STEM Education Conference, iSTEM-Ed 2020, 2020. doi: 10.1109/iSTEM-Ed50324.2020.9332721.

[6] D. D. Harter, "New microcontroller systems course at the University of Pittsburgh at Johnstown," Comput. Educ. J., 2000.

[7] M. Kimbrough, R. Chrysler, and S. Sukittanon, "Increase student project outcome in embedded system course through design competition," in ASEE Annual Conference and Exposition, Conference Proceedings, 2010. doi: 10.18260/1-2--16053.

[8] Z. Fedra and T. Fryza, "Different poles of three decades development in microcontrollers' domain," in IEEE History of Telecommunications Conference, HISTELCON 2008, 2008. doi: 10.1109/HISTELCON.2008.4668731.

[9] J. Gope et al., "JBWM-μP learner - An ICT based initiative for μP learning," in 2016 IEEE 7th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2016, 2016. doi: 10.1109/UEMCON.2016.7777927.

[10] Walk Free Foundation, “The Global Slavery Index 2018,” Int. J. Mach. Tools Manuf., 2018.

[11] R. K. S. C. Putri, “rahayu,” Int. J. Mach. Tools Manuf., 2018.

[12] M. A. Hidayat, “COST-EFFECTIVENESS ANALYSIS PENGGUNAAN ANTIBIOTIK UNTUK PASIEN RAWAT INAP DEMAM TIFOID DI RSUD BANGIL TAHUN 2016,” Int. J. Mach. Tools Manuf., 2018.

CHAPTER 7

AN OVERVIEW OF THE ARCHITECTURE OF INTEL 8085 MICROPROCESSOR

Afroz Pasha, Assistant Professor,
Department of Computer Science and Engineering,
Presidency University, Bangalore, Karnataka, India
Email Id- afrozpasha@presidencyuniversity.in

ABSTRACT:

The architecture of the 8085 microprocessors mainly includes the timing & control unit, Arithmetic and logic unit, decoder, instruction register, interrupt control, a register array, serial input/output control. The most important part of the microprocessor is the central processing unit. The microprocessor is a single IC package in which several useful functions are integrated and fabricated on a single silicon semiconductor chip. Its architecture consists of a central processing unit, memory modules, a system bus, and an input/output unit.

KEYWORDS:

Central Processor Unit, Electronics, Microcontroller, Microprocessor, Software.

INTRODUCTION

The first invention of the integrated circuit was in the year 1959 and this commemorated the history of microprocessors. And the first microprocessor that was invented was Intel 4004 in the year 1971. It is even termed as a central processing unit (CPU) where multiple computer peripheral components are integrated on one chip. This includes registers, a control bus, clock, ALU, a control section, and a memory unit. Passing many generations, the current generation of the microprocessor was able to perform high computational tasks that use 64-bit processors also. This is a brief evaluation of microprocessors and the one type which we are going to discuss today is the 8085 microprocessor Architecture. Generally, the 8085 is an 8-bit microprocessor, and it was launched by the Intel team in the year of 1976 with the help of NMOS technology. This processor is the updated version of the microprocessor. The configurations of 8085 microprocessor mainly include data bus-8-bit, address bus-16 bit, program counter-16-bit, stack pointer-16 bit, registers 8-bit, +5V voltage supply, and operates at 3.2 MHz single segment CLK. The applications of 8085 microprocessor are involved in microwave ovens, washing machines, gadgets, etc. The architecture of INTEL 8085 microprocessor is as shown in Figure 1: The features of the 8085 microprocessor are as below[1]:

i. This microprocessor is an 8-bit device that receives, operates, or outputs 8-bit information in a simultaneous approach.

ii. The processor consists of 16-bit and 8-bit address and data lines and so the capacity of the device is 2^{16} which is 64KB of memory.

iii. This is constructed of a single NMOS chip device and has 6200 transistors.

iv. A total of 246 operational codes and 80 instructions are present

v. As the 8085 microprocessor has 8-bit input/output address lines, it has the ability to address 28 = 256 input and output ports.

vi. This microprocessor is available in a DIP package of 40 pins.

vii. In order to transfer huge information from I/O to memory and from memory to I/O, the processor shares its bus with the DMA controller.

viii. It has an approach where it can enhance the interrupt handling mechanism.

ix. An 8085 processor can even be operated as a three-chip microcomputer using the support of IC 8355 and IC 8155 circuits.

x. It has an internal clock generator

xi. It functions on a clock cycle having a duty cycle of 50%.

The Arithmetic Logic Unit

i. In addition to the arithmetic & logic circuits, the ALU includes the accumulator, which is part of every arithmetic & logic operation.

ii. Also, the ALU includes a temporary register used for holding data temporarily during the execution of the operation. This temporary register is not accessible by the programmer[2].

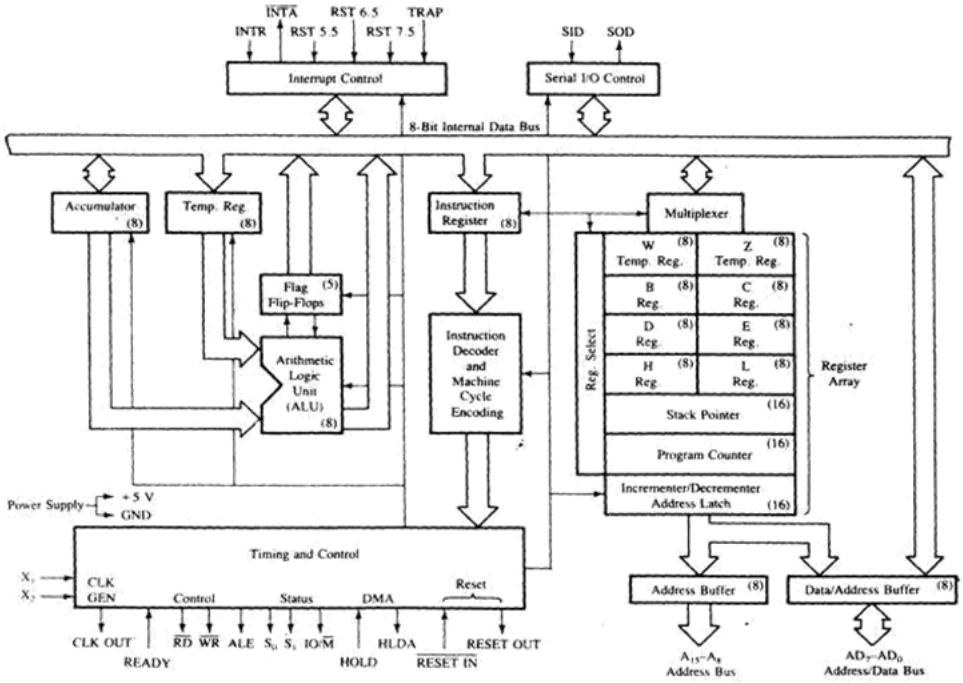


Figure 1: represented that the Architecture of Intel 8085 Microprocessor.

Registers

General Purpose Registers

i. B, C, D, E, H & L (8 bit registers)

ii. Can be used singly

iii. Or can be used as 16 bit register pairs BC, DE& HL

iv. HL used as a data pointer (holds memory address)

Accumulator (8 BIT REGISTER)

i. Store 8 bit data

ii. Store the result of an operation

iii. Store 8 bit data during I/O transfer Address

Flag Register

i. According to the Table 1, 8 bit register shows the status of the microprocessor before/after an operation.S (sign flag), Z (zero flag), AC (auxiliary carry flag), P (parity flag) & CY (carry flag).

Table 1: Represented that the 8-Bit Register.

D7	D6	D5	D4	D3	D2	D1	D0
S	Z	X	AC	X	P	X	CY

Sign Flag

i. Used for indicating the sign of the data in the accumulator.

ii. The sign flag is set if negative (1 – negative).

iii. The sign flag is reset if positive (0 –positive)

Zero Flag

i. Is set if result obtained after an operation is 0.

ii. Is set following an increment or decrement operation of that register.

Carry Flag

i. Is set if there is a carry or borrow from arithmetic operation.

Auxiliary Carry Flag

i. Is set if there is a carry out of bit 3

Parity Flag

i. Is set if parity is even.

ii. Is cleared if parity is odd.

The Program Counter (PC)

i. This is a register that is used to control the sequencing of the execution of instructions.

ii. This register always holds the address of the next instruction.

iii. Since it holds an address, it must be 16 bits wide.

The Stack Pointer

i. The stack pointer is also a 16-bit register that is used to point into memory.

ii. The memory this register points to be a special area called the stack. The stack is an area of memory used to hold data that will be retrieved soon.

iii. The stack is usually accessed in a Last in First out (LIFO) fashion.

Non-Programmable Registers

i. Instruction Register & Decoder

ii. Instruction is stored in IR after fetched by processor

iii. Decoder decodes instruction in IR

Internal Clock Generator

i. 3.125 MHz internally

ii. 6.25 MHz externally

The Address and Data Busses

i. The address bus has 8 signal lines A8 – A15 which are unidirectional.

ii. The other 8 address bits are multiplexed (time shared) with the 8 data bits.

iii. So, the bits AD0 – AD7 are bi-directional and serve as A0 – A7 and D0 – D7 at the same time.

iv. During the execution of the instruction, these lines carry the address bits during the early part, then during the late parts of the execution, they carry the 8 data bits.

v. In order to separate the address from the data, we can use a latch to save the value before the function of the bits changes[3].

Demultiplexing AD7-AD0

i. From the above description, it becomes obvious that the AD7– AD0 lines are serving a dual purpose and that they need to be demultiplexed to get all the information.

ii. The high order bits of the address remain on the bus for three clock periods. However, the low order bits remain for only one clock period and they would be lost if they are not saved externally. Also, notice that the low order bits of the address disappear when they are needed most.

iii. To make sure we have the entire address for the full three clock cycles, we will use an external latch to save the value of AD7– AD0 when it is carrying the address bits. We use the ALE signal to enable this latch.

Demultiplexing AD7-AD0

Given that ALE operates as a pulse during T1, we will be able to latch the address. Then when ALE goes low, the address is saved and the AD7– AD0 lines can be used for their purpose as the bi-directional data lines[4].

Demultiplexing the Bus AD7 – AD0

i. The high order address is placed on the address bus and hold for 3 clk periods.

ii. The low order address is lost after the first clk period, this address needs to be hold however we need to use latch.

iii. The address AD7 – AD0 is connected as inputs to the latch 74LS373.

iv. The ALE signal is connected to the enable (G) pin of the latch and the OC – Output control – of the latch is grounded

Addressing Modes

The microprocessor has different ways of specifying the data for the instruction. These are called addressing modes.

The 8085 has four addressing modes:

i. Implied CMA

ii. Immediate MVI B, 45

iii. Direct LDA 4000

iv. Indirect LDAX B Mode

Load the accumulator with the contents of the memory location whose address is stored in the register pair BC). Many instructions require two operands for execution. For example transfer of data between two registers. The method of identifying the operands position by the instruction format is known as the addressing mode. When two operands are involved in an instruction, the first operand is assumed to be in a register Mp itself. Types of Addressing Modes

i. Register addressing
ii. Direct addressing mode
iii. Register indirect addressing
iv. Immediate Addressing mode
v. Implied addressing

Register Addressing

This type of addressing mode specifies register or register pair that contains data.ie only the register need be specified as the address of the operands. Example MOV B, A (the content of A is copied into the register B).

Direct Addressing Mode

Data is directly copied from the given address to the register.

Example LDA 3000H (The content at the location 3000H is copied to the register A).

Register Indirect Addressing

In this mode, the address of operand is specified by a register pair. Example MOV A, M (Move data from memory location specified by H-L pair to accumulator)

Immediate Addressing Mode

In this mode, the operand is specified within the instruction itself. Example MVI A, 05 H Move 05 H in accumulator.

Implied Addressing Mode

This mode doesn't require any operand. The data is specified by opcode itself. Example RAL, CMP.

Instruction Set OF 8085

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called Instruction Set. Since the 8085 is an 8-bit device it can have up to 28 (256) instructions. However, the 8085 only uses 246 combinations that represent a total of 74 instructions. Each instruction has two parts. The first part is the task or operation to be performed. This part is called the opcode (operation code). The second part is the data to be operated on. This part is called the operand[5].

Instruction Size

i. Depending on the operand type, the instruction may have different sizes. It will occupy a different number of memory bytes.

ii. Typically, all instructions occupy one byte only.

iii. The exception is any instruction that contains immediate data or a memory address.

iv. Instructions that include immediate data use two bytes.

v. One for the opcode and the other for the 8-bit data.

vi. Instructions that include a memory address occupy three bytes.

vii. One for the opcode, and the other two for the 16-bit address.

Classification of Instruction Set

i. Data Transfer Instruction

ii. Arithmetic Instructions

iii. Logical Instructions

iv. Branching Instructions

v. Machine Control Instructions

Arithmetic Instructions

These instructions perform arithmetic operations such as addition, subtraction, increment, and decrement.

Addition

Any 8-bit number, or the contents of register, or the contents of memory location can be added to the contents of accumulator. The result (sum) is stored in the accumulator. No two other 8-bit registers can be added directly. Example: The contents of register B cannot be added directly to the contents of register C.

Subtraction

Any 8-bit number, or the contents of register, or the contents of memory location can be subtracted from the contents of accumulator. The result is stored in the accumulator. Subtraction is performed in 2's complement form. If the result is negative, it is stored in 2's complement form. No two other 8-bit registers can be subtracted directly.

Increment/Decrement

The 8-bit contents of a register or a memory location can be incremented or decremented by 1.The 16-bit contents of a register pair can be incremented or decremented by 1. Increment or decrement can be performed on any register or a memory location.

Logical Instructions

These instructions perform logical operations on data stored in registers, memory and status flags. The logical operations are:

i. AND

ii. OR

iii. XOR

iv. Rotate

v. Compare

vi. Complement AND, OR, XOR

Any 8-bit data, or the contents of register, or memory location can logically have with the contents of accumulator. The result is stored in accumulator.

i. AND operation

ii. OR operation

iii. XOR operation

Branching Instructions

The branching instruction alters the normal sequential flow. These instructions alter either unconditionally or conditionally.

Branch operations are of two types:

Unconditional branch: Go to a new location no matter what.

Conditional branch: Go to a new location if the condition is true.

Timing Diagram

Timing diagram is the display of initiation of read/write and transfer of data operations under the control of 3-status signals IO / M, S1, and S0. All actions in the microprocessor are controlled by either leading or trailing edge of the clock.

Machine Cycle

It is the time required by the microprocessor to complete the operation of accessing the memory devices or I/O devices. In machine cycle various operations like opcode fetch, memory read, memory write, I/O read, I/O write are performed.

T-STATE

Each clock cycle is called as T-states. Each machine cycle is composed of many clock cycles. Since, the data and instructions, both are stored in the memory, the µP performs fetch operation to read the instruction or data and then execute the instruction. The 3-status signals: IO / M, S1, and S0 are generated at the beginning of each machine cycle. The unique combination of these 3-status signals identify read or write operation and remain valid for the duration of the cycle[6].

Table 1: Represented that the Machine Cycle Status and Control Signals

	Status			*Controls*		
Machine cycle	$IO/\overline{M}$	S_1	S_0	$\overline{RD}$	$\overline{WR}$	$\overline{INTA}$
Opcode Fetch (OF)	0	1	1	0	1	1
Memory Read	0	1	0	0	1	1
Memory Write	0	0	1	1	0	1
I/O Read (I/OR)	1	1	0	0	1	1
I/O Write (I/OW)	1	0	1	1	0	1
Acknowledge of INTR (INTA)	1	1	1	1	1	0
BUS Idle (BI) : DAD	0	1	0	1	1	1
ACK of RST, TRAP	1	1	1	1	1	1
HALT	Z	0	0	Z	Z	1
HOLD	Z	X	X	Z	Z	1

X ⇒ Unspecified, and Z ⇒ High impedance state

Table 1 shows details of the unique combination of these status signals to identify different machine cycles. Thus, time taken by any µP to execute one instruction is calculated in terms of the clock period. The execution of instruction always requires read and writes operations to transfer data to or from the µP and memory or I/O devices. Each read/ write operation constitutes one machine cycle (MC1) as indicated in Figure 2. Each machine cycle consists of many clock periods/ cycles, called T-states.

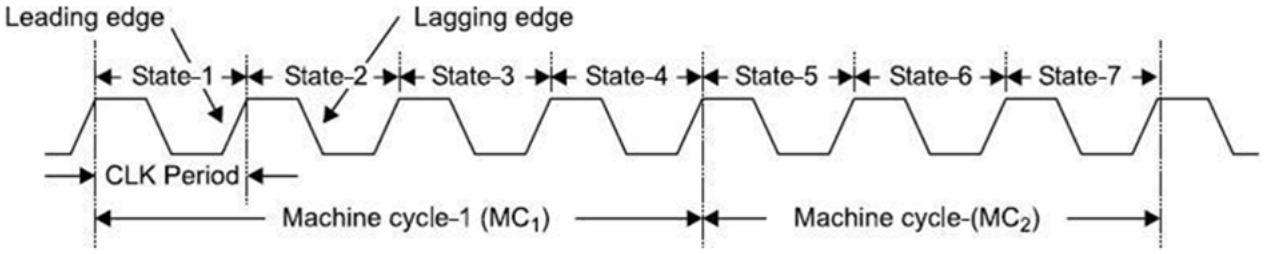


Figure 2: Represented that the Machine Cycle Showing Clock Periods.

Processor Cycle

The functions of the microprocessor are divided into fetch and execute cycle of any instruction of a program. The program is nothing but number of instructions stored in the memory in sequence. In the normal process of operation, the microprocessor fetches (receives or reads) and executes one instruction at a time in the sequence until it executes the halt (HLT) instruction.

Instruction Cycle

An instruction cycle is defined as the time required to fetch and execute an instruction. For executing any program, basically 2-steps are followed sequentially with the help of clocks

i. Fetch

ii. Execute.

The time taken by the μP in performing the fetch and execute operations are called fetch and execute cycle. Thus, sum of the fetch and execute cycle is called the instruction cycle as indicated in Fig. 8. Each read or writes operation constitutes a machine cycle. The instructions of 8085 require 1–5 machine cycles containing 3–6 states (clocks). The 1st machine cycle of any instruction is always an Op Code fetch cycle in which the processor decides the nature of instruction? It is of at least 4-states. It may go up to 6-states.

DISCUSSION

With the development of microprocessor devices, there was a huge transition and changeover in the lives of many people across multiple industries and domains. As because of the device's cost-effectiveness, minimal weight, and usage of minima power, these microprocessors are in huge usage these days. Today, let us consider the applications of the 8085 microprocessor architecture. As the 8085 microprocessor architecture is included with the instructional set which has multiple basic instructions like Jump, Add, Sub, Move, and others. With this instructional set, instructions are composed in a programming language that is understandable by the operational device and performs numerous functionalities like addition, division, multiplication, moving to carry, and many. Even more complicated can also be done through these microprocessors[7].

i. Engineering Applications

The applications those use microprocessor are in traffic management device, system servers, medical equipment, processing systems, lifts, huge machinery, protection systems, and investigation domain and in few lock systems those have automatic entry and exits.

ii. Medical Domain

The foremost usage of microprocessors in the medical industry is in the insulin pump where the microprocessor regulates this device. It operates multiple functionalities like storage of calculations, processing of information that is received from biosensors, and examining the outcomes.

iii. Communication

a. In the communication domain, the telephonic industry is the most crucial and enhancing too. Here, microprocessors come into usage in digital telephonic systems, modems, data cables, and in telephone exchanges, and many others.

b. The application of the microprocessor in the satellite system, TV has allowed for the possibility of teleconferencing also[8].

c. Even in airline and railway registration systems, microprocessors are used. LAN's and WAN's for establishing communication of vertical data across the computer systems.

iv. Electronics

The brain of the computer is the technology of microprocessors. These are implemented in the various types of systems like in microcomputers to the range of supercomputers. In the gaming industry, many numbers of gaming instructions are developed by using a microprocessor. Televisions, Ipad, virtual controls even comprise these microprocessors to perform complicated instructions and functionalities[9], [10].

CONCLUSION

A hardware realization scheme of a CMOVE Architecture has been proposed and presented in this paper. The proposed system is constructed out of bit-sliced AMD 2900 equipment (at least most parts of the system). The advantage of the proposed system in high speed, multi-precision (multiple-byte) multiplication, compared to existing microprocessors (such as Intel 8085) has been demonstrated. The proposed system has a flexibility for distributed processing expansion, which can be used in various application areas (chemical process control, traffic control) involving distributed process control, monitoring and data acquisition. Thousands of devices can be connected to a CMOVE processor with just a 16-bit word size. Further studies should be conducted in a more detailed development of this type of distributed multi- terminal systems. The experimental installation of the system described in this paper is currently under construction. It is expected to be completed by the end of 1980. Subsequently, laboratory experimental studies will be conducted.

REFERENCES

[1] T. D. Stanley et al., "From archi torture to architecture: Undergraduate students design and implement computers using the Multimedia Logic emulator," Comput. Sci. Educ., 2007, doi: 10.1080/08993400601165735.

[2] E. Ali and W. Pora, "VHDL Implementation of ARM Cortex-M0 Laboratory for Graduate Engineering Students," in 2020 5th International STEM Education Conference, iSTEM-Ed 2020, 2020. doi: 10.1109/iSTEM-Ed50324.2020.9332721.

[3] G. Mostafa, "Development of a 16-bit microprocessor learning system using Intel 8086 architecture," in Proceedings of 2013 2nd International Conference on Advances in Electrical Engineering, ICAEE 2013, 2013. doi: 10.1109/ICAEE.2013.6750323.

[4] A. J. van de Goor and O. Jansen, "Self test for the Intel 8085," Microprocess. Microprogramming, 1990, doi: 10.1016/0165-6074(90)90003-R.

[5] A. Lane, "SIMULATING A MICROPROCESSOR.," Byte, 1987.

[6] E. G. Codding, "Addressing, accessing and interfacing peripheral devices to microprocessors," Talanta, 1981, doi: 10.1016/0039-9140(81)80083-5.

[7] H. Azaria and D. Tabak, "BIT-SLICED REALIZATION OF A CMOVE ARCHITECTURE MICROCOMPUTER.," EUROMICRO J. (European Assoc. Microprocess. Microprogramming), 1980, doi: 10.1016/0303-1268(80)90082-6.

[8] J. R. Li and A. Zagari, "Interfacing an 8085-based microcontroller: A practical approach to developing computer application skills," in ASEE Annual Conference Proceedings, 2001. doi: 10.18260/1-2--9454.

[9] A. Hsu, I. Chow, and S. Yang, "FPGA implementation in nuclear digital I&C applications," in 9th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies, NPIC and HMIT 2015, 2015.

[10] A. Elahi, "TELEPHONE SYSTEM FOR THE HEARING IMPAIRED.," in Bioengineering, Proceedings of the Northeast Conference, 1987.

CHAPTER 8

AN ELABORATION BETWEEN INTEL 8085 AND INTEL 8086

Zafar Ali Khan N, Associate Professor,
Department of Computer Science and Engineering,
Presidency University, Bangalore, Karnataka, India
Email Id- zafaralikhan@presidencyuniversity.in

ABSTRACT:

In the following paper, we shall see features of the 8085 microprocessor and 8086 microprocessor and we will compare them briefly and will see which is better suited for the overall task. To ease this learning process, we have developed a software simulation of the popular Model ET3400A microprocessor trainer. All hardware experiments are carried out on the physical trainers, but students can do software development at home without physically possessing a trainer. Anecdotal evidence to date suggests that the simulator is of significant benefit to the students in quickly mastering the 6800 architectures.

KEYWORDS:

Communication, System, Electronics, Microprocessor, 8085 Interrupts.

INTRODUCTION

A microprocessor is a controlling unit of a micro-computer, fabricated on a small chip capable of performing ALU (Arithmetic Logical Unit) operations and communicating with the other devices connected to it that are used by a computer to do its work. It is a central processing unit on a single integrated chip containing millions of very small components including transistors (The transistors are mostly MOSFETs), registers, and diodes that work together. Some microprocessors in the 20th century required several chips. Microprocessors help to do everything from executing the most basic daily tasks up to the very complex ones. Everything a computer does is described by a set of instructions, and microprocessors carry out these instructions many millions of times a second. Microprocessors were invented in the 1970s for use in embedded systems. The majority are still used that way, in such things as phones, automation, home appliances [1].

8085 Microprocessor

085 microprocessor was designed by Intel in March 1976. It is an 8-bit software, binary compatible microprocessor. According to the Figure 1, there are three types of buses in 8085 microprocessor Address bus, Databus, and Control bus. Address bus recognizes the data in memory, find or tells the location of data in 8085 microprocessor it is of 16 bits, Address bus has 16 bits that means 216 bytes can be handled by 8085 microprocessors. Data bus carries data from memory, once the address or location is known, Data is being transferred. The control bus has two tasks either to read or to write on the location .8085 is an 8-bit microprocessor as its ALU is of 8-bit size.

PC (Program Counter), is used for sequencing of programs, that is which instruction has to be executed after which one, that is controlled by PC, in simple words address of the next instruction

is stored by PC [2]. The increment and decrement of PC are done by INC/DEC (Increment/Decrement) register. The size of INC/DEC registers is 16 bits, as they work on PC, which stores an address of 16 bits.

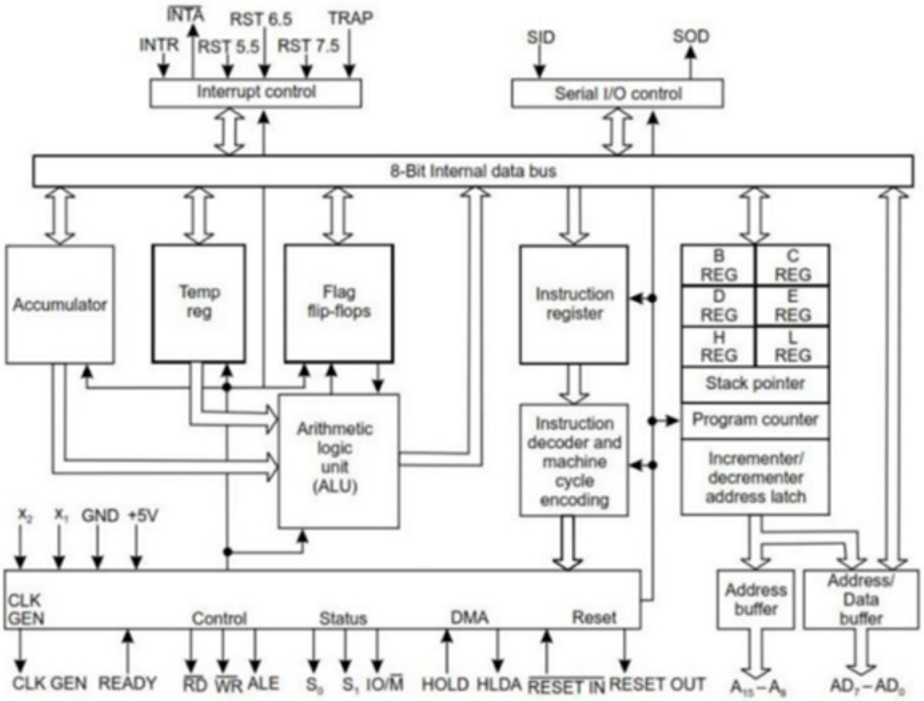


Figure 1: Represented that the 8085 Architecture

Address buffer is a storage technique used by PC to store instructions address here there are two address buffers, each of size 8 bit. One of the two buffers is used to store just address lines from 8 to 15 are A15 to A8, while the other line 0 to 7 are AD0 to AD7, these lines are multiplexed address and data bus. In this paper we showed how to build a HADES simulation object that models the behavior of the Intel 8085 processor architecture. We used an existing open source 8085 simulator written in Java, and adjusted its interface to the HADES simulation model. Instead of letting the GUI maintain the interrupts, memory and I/O ports, the model uses the HADES component pins to communicate with the rest of the HADES design, thus providing a powerful and realistic simulation environment for testing and analyzes of microprocessor systems. The HADES framework and its Simulation API provide almost unlimited possibilities when it comes to component design, and having in mind their open-source availability, their use should be increased in both academic as well as professional system development areas. This discrete simulation model is sufficient to model and simulate any kind of digital logic on the gate level, or above. The only limit may be seen in the continuous-space problems, although custom-made components that work with continuous signals may also be easily written in Java [3].

Data stored in the address buffer goes to the internal data Bus by a channel to IR (Instruction Register). Data fetched from memory is to be stored by IR. It only holds the data that is being fetched from the memory After IR comes ID which information decoder, it converts Data into 0s and 1s after it decodes comes the Timing and Control unit. The timing and Control unit is like a brain to the microprocessor 8085 it provides information and control signal to all the registers of the microprocessor. After T and C unit comes to ALU (Arithmetic Logical Unit), It is an 8-bit unit.

B, C, A, E, H, and L are all 8-bit registers known as General purpose registers these registers are programmer friendly because they can be programmed easily. Register A or B the Accumulator is not a programmer-friendly register, it always contains the first operand. The second Operand is contained by the temporary (TEMP) register which stores the operand‘s value temporarily by taking the actual value from general-purpose registers which are given by the programmer[4].

The accumulator not only stores the first operand but also the output of the processor. ALU receives two inputs from Accumulator and Temporary register and gives the result back to the accumulator through an 8-bit internal Data Bus. Flag Flip Flop does not contain output but the status of the output, they are known as status registers, SP is stack pointer it denotes the last elements stored in the stack that will be executed (because of the LIFO-Last In First Out concept). It also holds the address of the data (not the data but its address which is of 16 bit, that ‘s why stack pointer has the size of 16 bit). During this whole process if any interrupt arises that is handled by the Interrupt controller, it works on two pins INTA (Interrupt acknowledged) and INTR (Interrupt Request). Serial I/O controller also works on two pins SID (Serial Input Data) and SOD (Serial Output Data). Programmers sometimes combine general-purpose registers to store 16 bits of data [5].

Storage and Interface Unit of 8085

B, C, D, E, H, and L registers, are 8 bits registers that can be combined to form 16 bit registers, and that are BC, DE, and HL. The program counter is utilized to point next instruction to get executed at a particular location and it leads to address line and data line that is address latch-A8 to A15, address and data latch-AD0 to AD7.

Instruction Unit of 8085 Microprocessor

IR has data of the next instruction to get executed.ID decodes the instruction and after decoding it is given to the timing and control unit, where there are control signals.X1and X2 are interfaced crystal clock synchronized and calibrated with an 8085 microprocessor as display in Figure 2.

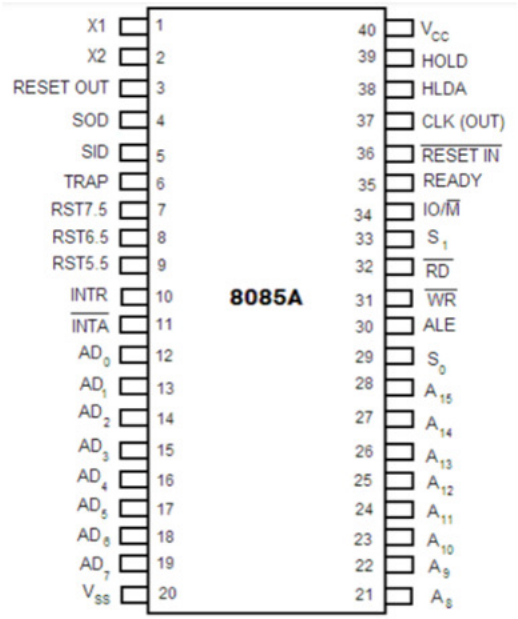


Figure 2: Illustrated that the 8085 Pin Diagram.

Pin 1 and 2orX1 and X2 respectively are crystal clocks or frequency pins they maintain the internal frequency of 8085 microprocessor. Pin-3 or RESET OUT is reset output pin which resets the output as well as memory and I/O devices. Pin 4 and 5 are SID.SID is Serial Input Data and I/O devices transfer data, bit by bit serially through this pin. SOD is Serial Output Devices pin microprocessor gives data, bit by bit serially through this pin. Pin 6 to Pin 11 is Interrupt pins they manage Interrupts. There are two types of interrupts maskable interrupts and unmaskable interrupts and the pins in priority order are TRAP, RST 7.5, RST 6.5, RST 5.5, INTR. Pin 12 to Pin19 These are Address-Data multiplexed Bus, address bus are unidirectional and address-data bus are bidirectional, contains data and is used for transfer of data. Pin-20 or GND is the ground pin used for grounding as display in Table 1. Pin 21 to Pin 28are Address line pins (A8 to A15).Pin 30 is ALE address latch enabled it latches the address. Pin 29 to Pin 34 (except pin 30) are as following S0 and S1 are Staus pins, IO/M are Input data memory bar, RD and WR are read and write pins respectively.

Table 1: Illustrated that the Truth Table of IO/M, S0 and S1.

IO/M	S0	S1	Status
0	0	0	HALT
0	0	1	M-WRITE
0	1	0	M-READ
1	0	1	I/O-WRITE
1	1	0	I/O-READ
0	1	1	Opcode Fetch
1	1	1	INT-A

Pin-35 -READY -works when every pin of 8085 is in its position and then it is executed. Pin-36 REST-1N (resets input and by that it means I/O devices and the memory).Pin-37 CLK(OUT) keeps the record of the data in terms of time. Pin 38 and Pin 39 are HOLD and HLDA respectively they hold request and they acknowledge the requests respectively.Pin40is Vcc pin it provides +5V power supply[1].

Features OF 8086

8086 is a microchip designed by Intel between early 1976 and June 8, 1978. The structure are display in Figure 3, Intel 8086 Microprocessor has two units BIU (Bus Interface Unit) and EU (Excitation Unit), It was the world's first 16-bit microprocessor, It supports pipelining, All microprocessors have three basic tasks to do Fetching, Decoding, and Execution. BIU is connected directly to memory with I/O (Input-Output) devices, there is no connection of EU and memory directly, the communication of EU with memory and I/O devices happens through BIU. First data comes from memory or I/O devices to BIU and then from BIU it goes to EU, where it is executed and the result will go back to BIU and then to I/O devices or memory. The segment is a part of memory, for instance, 8086 has two Bus lines Data line and an Address line, Data line is 16 bits and the Address line is 20 bits. Since the size of the area line is 20 bits, so it can handle up to 2^20 bytes of memory (2^20 bytes is almost equal to 1 MB(Megabyte)).CS (Code Segment) register, DS (Data Segment) register, SS (Stack Segment) register, ES (Extra Segment) register, IP (Input

Pointer) register are not memory segments but are segment registers. Segment registers are registers that contain the address of the segment, That is they store their respective segment's address location, as they store address so they should have the same size as the ALU(Arithmetic Logical Unit) of 8086 microprocessor which is 16 bit. There are two types of addresses Virtual address and Physical address. The virtual address is the address that is user or programmer-friendly, while the physical address is machine-friendly it is in binary form.

There is another type of address that is calculated with the physical address and that is the Offset address. A physical address can be easily calculated by the formula:

PA=SA* 10H +OA.

Where, PA is the physical address, SA is the segment address, 10 H is 10 in hexadecimal system, and OA is the offset address. The physical address is given by segment registers and then the information moves to IQ (Instruction Queue) which has the size of 6 bytes (6*8 bits), IQ is responsible for the fast execution of 8086. After IQ information is transferred to the Control section of the architecture, which is referred to as the brain of the EU. In 8086 microprocessors, the general-purpose registers are divided into separate 8 bits parts so that if we want to store data that is 8 bits or less there is no point in wasting a 16-bit register. SP is the Stack Pointer, it points at the Top of memory, BP is the Base pointer it points at the base of the memory.SI and DI are Source Index and Destination Index respectively. SI acts as the offset to DS and DI acts as the offset of ES. After this comes ALU, it takes two inputs and gives output based on the instruction given to it and the way it is programmed. The output of ALU is connected to the Flag register and the Temp (temporary) register. Flag register shows or calculates the status of the output, Flag has many registers in it but some common ones are zero registers, parity register, sign register, and size register. Ax (AH + AL), Bx (BH+BL), Cx(CH+CL), Dx (DH+DL), SP, BP, SI, DI all these are general-purpose registers, all of these registers are programmer friendly.

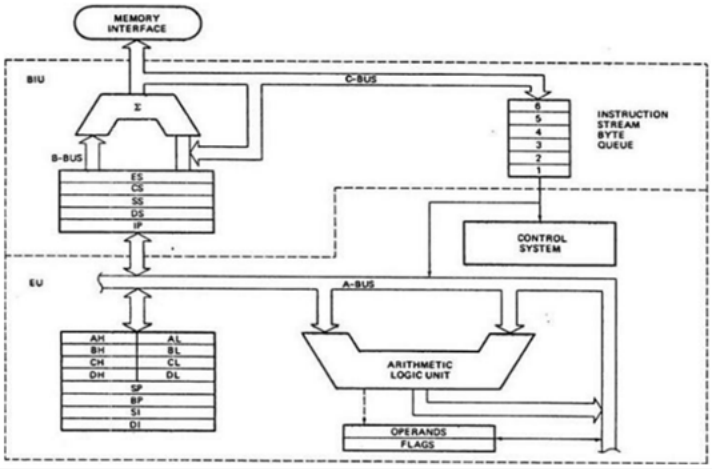


Figure 3: Represented that the 8086 Structure.

Common Mode Pins

Pin-1 is the GND pin which is the ground pin, it moves excessive electricity into the ground.Pin-2 to Pin-16 are AD14 to AD0, these pins are multiplexed addresses and data lines.Pin-17 is the NMI pin it is the non-maskable interrupt pin, it handles the interrupt that is not under the programmer's control.Pin-18 is an INTR pin it is the Interrupt Request pin, which requests the

maskable interrupts and can be programmed. Pin 19 is the CLK pin it is the Clock pin, all internal parts of the 8086 microprocessor are synchronized by this pin. Pin 20 is another GND pin. Pin 21 is the RESET pin, and as the name suggests it resets the whole architecture. Pin 22 is the READY pin, it tells whether external devices are ready to fetch data or not. Pin 23 is the $T\ E\ S\ T$ it works on WAIT instruction, if the processor is in WAIT state, then this pin works. Pin 32 is the $R\ D$ pin, it is an active low pin. Pin 33 is MN/$M\ X$ pin, If VCC is connected with this pin then MIN mode is enabled. If the GND pin is connected with this pin then MAX mode is enabled. Pin 34 is the BHE/S7 it is the Bus high enabled pin.

Table 2: Represented that the Truth Table of BHE and S7.

BHE	S7	Operation
0	0	Whole (W)
0	1	Upper (ODD)
1	0	Lower (EVEN)
1	1	Idle state

Pin 35 is the A19/S6, this is the only pin that is reserved for future purposes, it has been set to default logic-0. Pin 36 is A18/S5 pin, it contains the value of the interrupt flag, which tells the status of the interrupt. Pin 37 and Pin 48 are A16/S3 and A17/S4, these pins tell the status, also tells which segment is being used.

Table 3: Represented that the Truth table of A16/S3 and A17/S4.

A16/S3	A17/S4	Segment
0	0	ES (Extra Segment)
0	1	SS (Stack Segment)
1	0	CS (Code Segment)
1	1	DS (Data Segment)

Un-Common pins (MIN mode)

Pin 24 is the INTA pin, which is Interrupt acknowledgment pin. Pin 25 is the ALE pin which is an address latch enabled pin, this pin decides whether the Address line will be used or the data line will be used in multiplexed address and data line. If ALE is logic-0 then the address line is enabled, If ALE is logic-1 then the Data line is enabled. Pin 26 is the DEN pin, which is the Data-Enabled pin, if a data buffer is present in any data bus, then DEN is enabled or disabled to access that data buffer. Pin 27 is the DTR pin, it is Data transmit request pin. If DTR is logic 0 then the microprocessor transmits the data, If DTR is logic-1 then the microprocessor receives the data. Pin

28 is the M/ *I*O, Microprocessor can either take data from memory or I/O devices, this pin tells whether data is being taken from Memory or I/O devices. Pin 29 is the WR pin which is the write pin. Pin 30 and Pin 31 are HOLD pin and HLDA pin respectively, these are Hold and Hold acknowledge pins. HOLD holds memory while accessing and HLDA acknowledges it [6].

Need for a Simulator

The disadvantage of switching processors is that part of the second course must now be consumed with learning the new architecture and assembly language. The students have the programming skills necessary, and there are no major conceptual hurdles in the new architecture. Nevertheless, practice is necessary to become skilled in the new assembly language. Some programming is required to test the hardware in the various hardware experiments. A weakness in programming is a hindrance to performing the hardware experiments within the time constraints of the class.

DISCUSSION

Historically, a two-semester sequence of courses in microprocessor software and interfacing was taught using a processor such as the Motorola 6800. However, advances in the field have necessitated teaching more advanced microprocessors (such as the Intel 80x86 family) in the software course. For several reasons we continue to teach the interfacing course using the Motorola 6800. Reasons include legacy laboratory hardware, tractability of the 8 bit hardware bus, and the popularity of the 6800 family derivatives as embedded controllers. Students are now required to master the 6800 architecture and assembly language in a short time at the beginning of this course. [7] To ease this learning process, we have developed a software simulation of the popular Heath kit Model ET3400A microprocessor trainer. All hardware experiments are carried out on the physical trainers, but students can do software development at home without physically possessing a trainer. Anecdotal evidence to date suggests that the simulator is of significant benefit to the students in quickly mastering the 6800 architecture.

One of the most time-consuming aspects of testing programs on the real trainer is the time required to enter the program in hex by hand. Although an assembler may be used to translate the program into opcodes, these opcodes must still be entered one by one. If a program overwrites itself in memory due to a logic error or a typing mistake, the entire program may need to be re-entered repeatedly [8]. A large amount of time is consumed entering, checking, and re-entering programs. The simulator significantly improves this situation by adding the capability to load programs directly into the simulator memory. Most 6800 assemblers including the freeware assembler available from Motorola can produce output in the Motorola standard S19 format. The simulator loads these S19 files directly into memory. We do require students to perform some hand assembly of programs. In their first exposure to the trainers, they do enter the programs by hand, in order to become familiar with the interface. After that, they quickly begin using an assembler. Note that this feature is not currently available on our trainers. When the students perform the hardware experiments, they must still enter the programs by hand[9], [10].

CONCLUSION

The 8085 is an 8-bit microprocessor. It was produced by Intel and first introduced in 1976. The 8086 is an enhanced version of the 8085 microprocessor. It is a 16-bit processor. Even though both are made at different times and intended for different purposes. The biggest advantage is address bus in 8086 is 20 bits and the data bus is 16 bits in a single clock cycle can operate 2 bytes word.

So it gives much faster and better performance than 8085.8086 data pins are of 16 bits, so execution time is unaffected for the 16-bit opcode to be read or write into memory locations, 8085 has 8-bit data pins thus it requires additional four clock cycles to fetch the 16-bit opcode. The 8086 has a 20-bit address bus, so it can address 2^20 addresses. Each address represents a stored byte. To make it possible to read or write a word with one machine cycle, the memory for an 8086 is set up in 2 banks, i.e Lower Bank and Upper Bank. The Upper Memory bank contains all bytes which have odd addresses while the Lower contains bytes of even addresses, if we read a byte from or write a byte to an even address the A0 will be low BHE will be high enabling the lower bank and disabling the upper bank. BHE stands for Bus High Enable, BHE is absent in 8085 and there is no such concept of banking. A0 and BHE signals prevent the writing of an unwanted signal. While entering HALT in minimum mode ALE is not delayed in 8086 but the ALE gets delayed by one clock cycle in 8085. So, yes there is enough difference in programming 8085 and 8086.

REFERENCES

[1] S. Mazor, "Intel's 8086," IEEE Ann. Hist. Comput., 2010, doi: 10.1109/MAHC.2010.22.

[2] A. K. Vaidya, "DESIGN OF A PROGRAMMABLE PROTOCOL FOR IEEE-488 INTERFACE BUS.," in IECI Annual Conference Proceedings (Industrial Electronics and Control Instrumentation Group of IEEE), 1980.

[3] S. K. Sen and A. K. Datta, "Microprocessor based fibre optic environmental monitoring system for underground mine," in 1991 IEEE Industry Application Society Annual Meeting, 1991. doi: 10.1109/ias.1991.178025.

[4] S. Mustawa, "PERANGKAT LUNAK PEMBELAJARAN OPERASI ARITMATIKA PADA MIKROPROSESOR INTEL 8088/8086 DENGAN METODE CBT (Computer Based Training)," J. Inform., 2019, doi: 10.36987/informatika.v1i2.106.

[5] P. Ambegaonkar and A. Ellis, "MICROPROCESSOR BASED DIGITAL AUTOPILOT.," in Proceedings of the National Electronics Conference, 1980.

[6] S. P. Morse, "The Intel 8086 Chip and the Future of Microprocessor Design," Computer. 2017. doi: 10.1109/MC.2017.120.

[7] R. Finkel, "Operating systems," in Computers, Software Engineering, and Digital Devices, 2005. doi: 10.1145/1041478.1041480.

[8] S. P. Morse, B. W. Ravenel, S. Mazor, and W. B. Pohlman, "Intel Microprocessors — 8008 to 8086," Computer (Long. Beach. Calif)., 1980, doi: 10.1109/MC.1980.1653375.

[9] J. Carrillo-Mondejar, J. M. Castelo Gomez, C. Núñez-Gómez, J. Roldán Gómez, and J. L. Martínez, "Automatic Analysis Architecture of IoT Malware Samples," Secur. Commun. Networks, 2020, doi: 10.1155/2020/8810708.

[10] Arpaci-Dusseau, "Operating Systems: Three easy pieces," Comput. Softw. Eng. Digit. Devices, 2005.

CHAPTER 9

AN OVERVIEW OF ACCESS CONTROLLER IN MICROCONTROLLER 8257

Ayesha Taranum, Assistant Professor,
Department of Computer Science and Engineering,
Presidency University, Bangalore, Karnataka, India.
Email Id- ayesha.taranum@presidencyuniversity.in

ABSTRACT:

DMA, or Direct Memory Access Controller, is an external device that manages data transmission between I/O devices and memory without involving the processor. It may directly access the main memory for reading or writing operations. Using a DMA controller, the device requests the CPU to hold its data, address and control bus, so the device is free to transfer data directly to/from the memory. The DMA data transfer is initiated only after receiving HLDA signal from the CPU.

KEYWORD:

Addressing Mode, External Accessing, Data Memory, Microcontroller, Microprocessor.

INTRODUCTION

An embedded system's microcontroller is a small integrated circuit that controls a single process. On a semiconductor technology, a typical microcontroller has a CPU, memory, and input/output (I/O) peripherals. Microcontrollers, also known as embedded processors or microcontroller units (MCU), may be found in a variety of devices, including vending machines, robotics, office equipment, medical devices, and office equipment [1]. They are basically straightforward mini-personal computers (PCs) without a complicated front-end operating system that are used to operate minor aspects of bigger components (OS). All controllers in a family have the same CPU core, making their code compatible, although they vary in the number of timers and memory they have. Anyone may simply verify that there are many microcontrollers mostly on market nowadays by visiting the websites of a few electronics retailers and looking through their microcontroller inventories. There are several distinct controller families, including 8051, PIC, HC, and ARM to mention a few, and even inside a single controller family, you may again have a selection of numerous different controllers [2].

Working of Microcontrollers

To control a single device function, a microcontroller is integrated into a system. It does this by using its core CPU to evaluate data that it receives from its own I/O peripherals. The microcontroller receives momentary data that is stored through its data memory, where another processor accesses it and employs programme memory instructions to interpret and apply the incoming data. It then communicates and takes the necessary action via its I/O peripherals. Numerous gadgets and systems make use of microcontrollers. Devices often use a number of microcontrollers, which cooperate to carry out the device's many functions. An automobile, for instance, could have a large number of microcontrollers that manage a variety of internal systems,

including the anti-lock brake system, traction control, fuel injectors, and suspension control. To inform the appropriate actions, all the microcontrollers interact with one another. Some may just interface with other microcontrollers, while others may connect to a more sophisticated central computer only within vehicle. Using their I/O peripherals, they transmit and receive data, which they then analyses to do the tasks they have been given [3].

Elements of a Microcontroller

The core elements of a microcontroller are:

i. **CPU:** The device's "brain" is the processor (sometimes known as the "CPU"). It interprets and reacts to several commands that regulate how the microcontroller operates. This calls for doing basic logic, I/O, and arithmetic operations. Additionally, it carries out data transmission activities that send orders to other embedded system parts.

ii. **Memory:** A microcontroller's memory is where data is stored so that the processor may utilise it to perform commands that have been programmed into it. There are two major memory types in a microcontroller:

iii. Long-term information about just the instructions that now the CPU executes is kept in programme memory. Since programme memory is non-volatile, it can store data indefinitely without a power supply. Data memory is needed to store temporary data while instructions are being carried out. Data memory typically volatile, which means the information it stores is only kept current if the device is plugged into a power source.

iv. **I/O Peripherals:** The CPU communicates with the outside world via its input and output devices. Information is received through the input ports and sent as binary data to the CPU. After receiving the data, the processor transmits the appropriate instructions appropriate output devices that carry out activities not controlled by the microcontroller. While the microprocessor's core components are its processor, memory, and I/O peripherals, additional components are routinely added as well. Simply said, I/O peripherals are supporting devices that interact with the CPU and memory. Peripherals are a broad category that includes various supporting parts. Given that they are the means through which the CPU is used, I/O peripherals in some forms are essential to microprocessors [4].

Microcontroller Features

Application determines the processor used by a microcontroller. Simple 4-bit, 8-bit, or 16-bit processors are available, as well as more sophisticated 32- or 64-bit processors. Flash memory, erasable programmable read-only memory (EPROM), and electrically erasable programmable read-only memory are non-volatile memory types that microcontrollers may employ. Volatile memory types include RAM (EEPROM). Microcontrollers often have enough onboard memory and provide pins for common I/O operations so they may directly connect with sensors as well as other components. As a result, they are intended to be easily useable without extra computational components.

The Harvard architecture and von Neumann architecture, which both provide various ways of transmitting data between both the processor and memory, may be used as the foundation for microcontroller design. A Harvard design allows for simultaneous transfers since the data bus and instruction were independent. One bus is used in a Von Neumann design for both data and

instructions. Complex instruction set computing (CISC) or reduced instruction set computing are two possible bases for microcontroller processors (RISC). RISC typically includes 30 instructions opposed to 80 for CISC, and CISC also has more addressing modes (12–24 vs. 3-5) than RISC. CISC may be simpler to design and utilise memory more effectively, but since it requires more clock cycles to instruction set, performance may suffer. Due to its streamlined instruction set and thus more straightforward architecture, RISC, which prioritises software over hardware, often outperforms CISC processors in terms of speed. However, since it prioritises software over hardware, the programme it runs may be more sophisticated. Various ISCs are used depending on the application [5].

Microcontrollers were the first devices to employ assembly language. The C programming language is a common choice nowadays. Python and JavaScript are two other popular microprocessor technologies. Input and output pins are available on MCUs to implement peripheral functionalities. Asynchronous to digital converters, LCD processors, real-time clocks, universal synchronous/asynchronous receiver transmitters (USART), timers, universal asynchronous receiver transmitters (UART), and USB connection are a few examples of these features. Microcontrollers often have sensors connected to them that collect data on things like humidity and temperature.

8257 Microcontroller Architecture

Intel created a microcontroller similar to the8257in 1981. The microcontroller is a type of integrated circuit having 40 pins, the dual inline package (DIP), 128 bytes of RAM, 4 KB of ROM, and two 16-bit timers. It has four parallel 8-bit ports that are addressable and programmable based on the need. The system bus is crucial throughout the8257microcontroller design because it links all the components to the CPU. This bus consists of an 8-bit data bus, a 16-bit address bus, and bus control signals. Ports, memory, interrupt control, synchronous serial, main CPU, and timers are just a few examples of additional devices that may be connected through the system bus. The8257Microcontroller has two buses: one for the program and one for the data. It thus contains two 64K by 8 storage chambers for both programmes and data. The microcontroller has an 8-bit processor unit and an 8-bit accumulator [6].

Additionally, it has an 8-bit B register, which is one of its main functional building pieces. The 8257 microcontroller is programmed with embedded C that used the Keil software. Several additional 8-bit and 16-bit registers are also included. Intel created the8257Microcontroller in the 1980s. It was built using Harvard Architecture as a basis and was primarily designed to implement Embedded Systems. It was originally designed using NMOS technology, but as NMOS requires more power to operate, Intel redesigned Microcontroller8257using CMOS technology. As a result, a new edition with a letter "C" inside the title name, for example: 80C51, was produced. These most recent Microcontrollers use less electricity to operate than their predecessors did. An8257microcontroller may be used in several applications. Therefore, 8257 Microcontroller Projects are quite important in the last year of engineering. The microcontroller 8257 has an integrated built-in RAM for internal processing. It is used to store temporary data and therefore is prime memory. Because it is an unexpected memory, whenever the power towards the microcontroller is turned off, its data may be lost. This microcontroller's design and instruction set are straightforward, it costs less, and it is easy to operate.

Features

The main features of the8257microcontroller architecture include the following Figure 1.

8-bit CPU with two Registers A and B, internal ROM of 8K bytes, plus flash memory that facilitates system programming. Internal RAM with 256 Bytes, where the initial RAM's 128 Bytes between 00H to 7FH are once again divided into four banks with 8 registers each, addressable registers with such a 16-bit resolution, and general-purpose registrations with an 80-bit resolution. Special Function Registers are included in the last 128 bytes of the RAM, from 80H to FFH (SFRs). Those registers manage a variety of peripherals, including all I/O ports, serial ports, and timers. Interruptions such as Internal-3 and External-2. The oscillator and CLK circuit. Such as PCON, SCON, TMOD, TCON, IE, and IP are control registers. 16-bit counters or timers, such as T0 and T1. 16-bit programme counter with DPRT (Data Pointer). There are 32 I/O pins, which are organised into four ports (P0, P1, P2, and P3). PSW (Processor Status Word) with Serial Data Tx myself and Rx for Full-Duplex Operation with Stack Pointer (SP) - 8bit.

Architecture

The Block Diagram of8257Microcontroller in Figure 1.

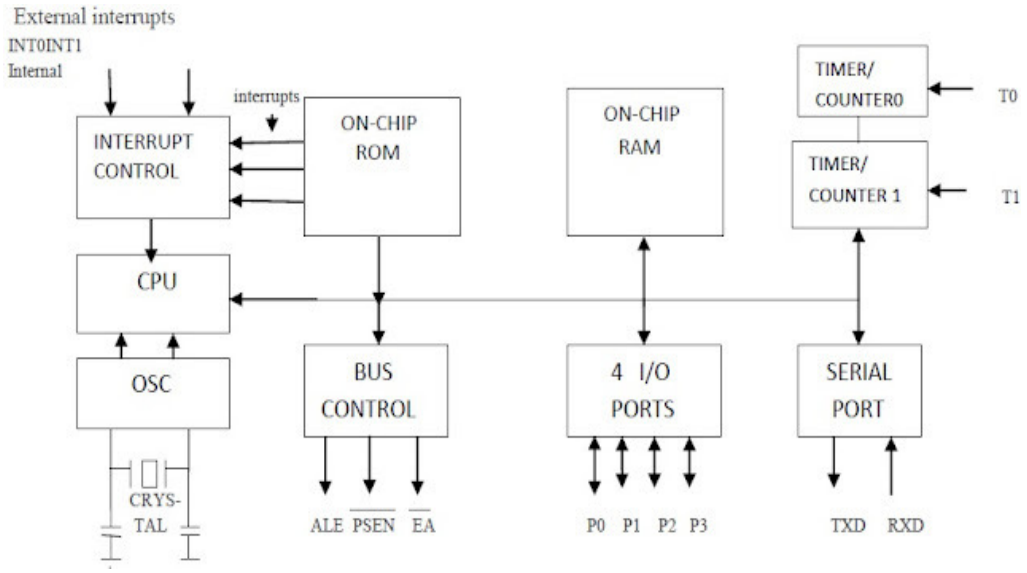


Figure 1: Illustrates the Block Diagram of 8257Microcontroller [7]**.**

CPU (Central Processor Unit)

The Central Processor Unit, or CPU, is the brain of any processing system, as you may be aware. It examines and controls every operation made in the microcontroller. The operation of the CPU is completely independent of the user. It understands the software written in storage space (ROM), executes every command, and completes the intended task. The CPU controls many register types within8257microcontrollers.

Interrupts

As stated in the headline, an interrupt is a subroutine call which reads the microcontroller's primary function or duty and enables it to execute a different programme that is more crucial at that time.

The8257Interrupt feature is quite helpful since it helps in emergency situations. We may pause or postpone the ongoing process with the help of interruptions, do a subroutine task, and then continue the main programme execution. The Micro-controller8257may be built in such a way that when an interrupt occurs, it briefly breaks or pauses the main programme. When the sub-routine work is complete, the main program's implementation begins automatically, as is customary. The8257microcontroller has five interrupt sources; two of them are peripheral interrupts, four are timer interrupts, and one is the serial port interrupt.

Memory

A programme was necessary for the microcontroller's functionality. The microcontroller is directed to carry out the specified tasks using this software. The microcontroller's installed programme needed some on-chip memory to store the programme. Memory was also needed by the microcontroller for temporary data and operand preservation. Microcontroller8257contains a data memory (RAM) of 128 bytes in addition to 4 KB of code or programme memory, and 4 KB of ROM.

Bus

A bus is a collection of cables that may be used for data transport or as a communication channel. Depending on the bus arrangement, there may be 8, 16, or more cables. In light of this, a bus may carry 8 bits in total, or 16 bits.

Types of Interrupts

The sources of an interrupts on8257microcontrollers are as follows:

TF0 (Timer 0 Overflow Interrupt)

TF1 (Timer 1 Overflow Interrupt)

INT0 (External Hardware Interrupt)

INT1 (External Hardware Interrupt)

RI/TI (Serial Communication Interrupt)

Memory

A programme memory and a data memory are part of the8257microcontroller technology's memories. The Program Memory houses the CPU's instructions. The programme stored there will be preserved even if the power is turned off or the system is reset, and it is often implemented as read-only programming or ROM. A microcontroller's storage device is in charge of keeping track of variable values, temporary data, interim results, and other information necessary for the programme to run smoothly [8].

Timer and Control Unit

A timer's primary purpose is to create a delay or a time gap between two occurrences. This microcontroller has two 16-bit timers, each of which may create two delays simultaneously to provide the desired delay. A timer is a physical device that may be utilized by the processor to create the specific delay that is employed by every microcontroller in general. The timer may produce the delay depending on the needs of the processor and sends a signal to both the processor

each time the specific delay is generated. We may also create a delay who have used this processor depending on the needs of the system. Unfortunately, so because CPU won't complete any other tasks at that time, this will advise keeping it active always. Because the microcontroller has a timer, this allows the CPU to be free to carry out other tasks. A programme counter, data pointer, stack and stack pointer, instruction registers with latches, temporary register, and buffers for the I/O ports are also included in the microcontroller.

Registers

In microcontrollers, registers are primarily used to store memory and short-term instructions that are utilized to process addresses and retrieve data. The 8-bit registers on just this microcontroller have a start value of D0 to D7. Here, the LSB (least significant byte) ranges from D0 to D7, with D7 being the most important bit (MSB). The data must be divided into eight distinct bit pieces in order to handle it better than 8-bit. It has a number of registers, although programmers typically have access to general-purpose registers. There are two categories: general purpose and special purpose. Thus, the majority of general-purpose registers. The basic purposes of an accumulator are to carry out mathematical and logical operations. Instruction addresses and data are stored in registers including B, R0 through R7.

Data Pointers

To enable and process data in various addressing modes, DPTR is employed. This register has two bytes, DPH (high) and DPL (low), which are mostly used to store a 16-bit address. As a result, it could be used as a base register for lookup table operations, external data transfer, and indirect jumps. Program counter, sometimes known as PC, is a 16-bit register was using to hold the address of the next instruction. Save for the programme counter and data pointer registers, remaining registers are 8-bits.

Data Types

This microcontroller only has one 8-bit data type, and each register is 8 bits in size. The programmer must divide data into 8-bit chunks before processing it if it is more complex than 8 bits. The DB directive throughout assembly language is the most often used data command for assemblers.

PSW Register

Program status word (PSW), a kind of register throughout the microcontroller, is what the phrase refers to. It is also known as a flag register and is used to show where arithmetic logic instructions like the zero-carry bit, carry bit, etc. are located. An 8-bit register known as the PSW or flag register uses six bits. This register has eight flags, which are referred to as conditional flags. If indeed the condition is met, these flags will execute the instruction without further action. Overflow, parity, auxiliary carry, and carry are all these conditional flags. While bits 1 and 5 are not utilized but might be used by the programmer to carry out a particular operation, bits 3 and 4 in the Program Status Word Registers are employed in order change the bank registers.

Register Banks

Ram with 32 bytes is utilized for stacks and register banks, which are divided into four different kinds of banks. Therefore, each back has eight registers, numbered R0 through R7. Here, R0 and R7 stand for the zero and seventh RAM locations, respectively. The second bank record starts at

address 8 and concludes at time 05H. The third bank registration starts at 10 a.m. and ends at 17 a.m. The last bank might be positioned between the 18H and 1FH.

Stack

The CPU mostly uses the Stack portion of RAM for temporary data storage or address. It plays a crucial role in a microprocessor since there are only a relatively limited number of registers available to store addresses and data. The stack in8257microcontrollers is 8 bits wide and can contain data in the range of 00 to FFH. The CPU may utilize the stack pointer to enable the stack. Since this microcontroller has an 8-bit stack pointer, this could accept values between 00H and FFH. After activation, the stack pointer has the value 07.

Organization of Memory

The microcontroller's memory was organized in a complicated way, and programme memory, peripheral RAM, and data memory each have their own address bus. It is dependent on the Harvard architecture created by Harvard in 1944.

Addressing Modes

The application programmer must've been able to define the explicit operands of an arithmetic instruction before utilizing it. Constants, register contents, or memory location components may all be used as operands. Therefore, the processor must have a way for the operand type to be specified. While the aforementioned kinds may be specified on any processor, there are several methods to access memory address depending on the situation. Therefore, another crucial aspect of any processor is the quantity and variety of addressing modes offered. A microprocessor may get data in a variety of ways. In general, the information kept in a memory or register may be utilized right away. Thus, addressing modalities refer to these various data access techniques. Depending on the manufacturer's intention, different microcontroller types have several addressing modes. The following are some examples of this microcontroller's addressing modes.

Immediate Addressing

The operand, which comes after the opcode using immediate addressing mode, is a constant information of either 8 or 16 bits. The constant data that needs to be kept in memory after the Opcode is what gave Immediate Addressing its name. Instead than using a register, the instruction directly specifies the linear relation to be saved. Constant data must always be copied to a destination register that has the same size as even the operand specified in the instruction.

Register Addressing

In the8257Microcontroller Memory Organization Tutorial, they looked at how RAM was organized as well as how four banks with eight Working Registers were organized. A single of the eight registers (R0 - R7) is designated as the Operand in the Instruction when Register Addressing mode is active. It is crucial to choose the right Bank with the aid of PSW Register.

Direct Addressing

The address of a data is supplied as the Operand throughout the instruction when using Direct Addressing Mode. Users are able to access whatever register or on-chip variables using direct addressing method. This contains control registers, I/O ports, SFRs, and general-purpose RAM.

Register Indirect Addressing

The address of a Operand is defined as the contents of a Register in the Indirect Addressing Mode or Register Indirect Addressing Mode.

Indexed Addressing Mode

The effective address of an Input signal in the Indexed Addressing Mode is the product of the base register as well as the offset register. Whereas the Offset register is indeed the Accumulator, the Base Register may either be a Data Pointer (DPTR) or a Program Counter (PC) (A). Only MOVC as well as JMP instructions may be utilized in index-addressing mode. When obtaining information from look-up databases, Indexed Addressing Mode can helpful. Example: If R0 is set to the value 20H and there is the data 2F H stored under address 20H, however after performing this instructions in Figure 2, the value 2F H will be transferred towards the accumulator.

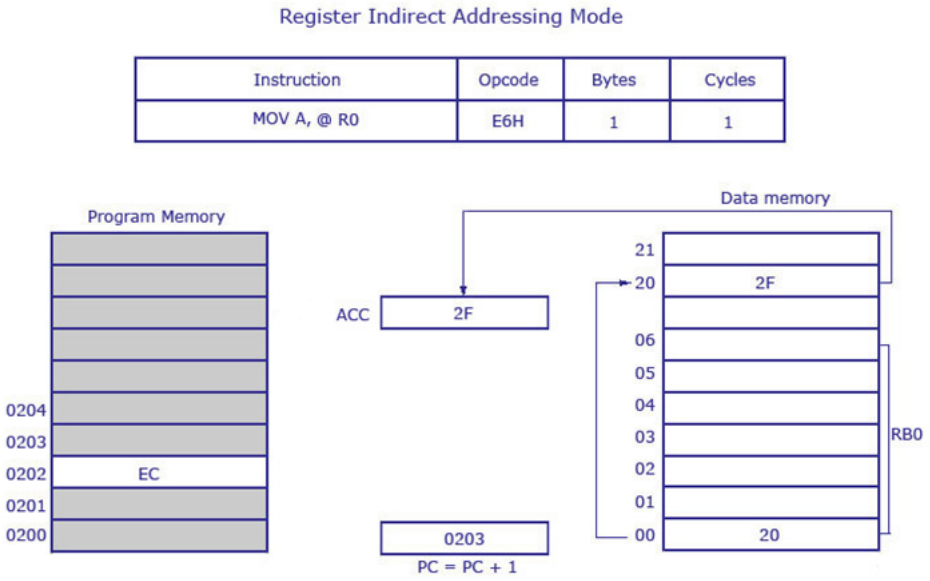


Figure 2: Illustrates an example of indexed addressing mode [9]**.**

Memory

Of course, the register file is merely a little piece of internal CPU memory. Additionally, we briefly touched on the exchange of data between registers as well as the data memory as well as the retrieval of instructions from the instruction memory. As a result, a clear differentiation between memory types may be drawn based on their purpose:

Register File

A (often) little memory that is integrated into the CPU. You may think of it as the CPU's short-term memory since it serves as a scratchpad for values the CPU is working with.

Data Memory

Generic CPUs often use an external memory that is substantially bigger than the register file for longer-term storage. It's possible that there's temporary data kept, but there is also a chance that it

will remain accurate while the CPU is active. It goes without saying that adding external memory to a CPU involves some hardware work and costs money. Because of this, microcontrollers often have on-chip data memory.

Instruction Memory

The instruction memory is often a large external memory, similar to the data memory (at least with general CPUs). In fact, with von-Neumann architectures, both physical memory and the data memory could even be the same. The instruction memory is often also built into microcontrollers themselves. These represent the most prevalent CPU-related memory usage. A CPU has more memory than is readily apparent, however. There may be pipeline registries, caches, different buffers, and so on according on the kind of CPU.

Instruction Set of 8051

Writing a program for a microcontroller mostly entails providing instructions (commands) in the precise sequence that they should be carried out to complete a certain job. Electronics cannot "understand" instructions like "if the power button is pressed- turn just on light," thus instead, a fixed number of clearer, more concise commands that a decoder can comprehend must be utilized. All orders are referred to as INSTRUCTION SET. There are 255 total instructions, or 255 distinct words, available for program authoring on all 8051-compatible microcontrollers. At first glance, there are a lot of strange indicators that must be memorized. It is not, however, as difficult as it seems. There are only 111 actually separate commands since many instructions are regarded as "different" even if they carry out the identical task. For instance, the instructions ADD A, R0, ADD A, R1, and ADD A, R7 all carry out the same operation (addition of the accumulator and register). Each instruction was counted individually due to the fact that there are 8 of these registers. There are really 20-30 acronyms to learn, which is fair given that all commands only carry out 53 operations (addition, subtraction, copy, etc.), and that the majority of them are seldom ever utilized in reality [10].

Types of Instructions

Depending on operation they perform, all instructions are divided in several groups in Figure 3:

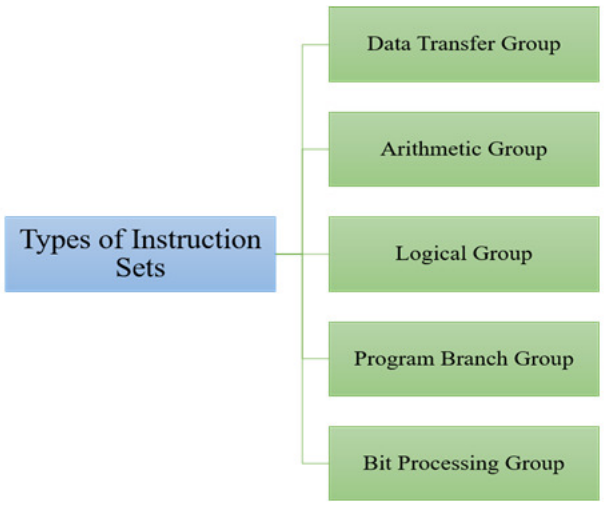


Figure 3: Illustrates the types of instruction sets in 8257 microcontrollers.

Arithmetic Instructions

A number of fundamental operations, including addition, subtraction, division, and multiplication, are carried out using arithmetic instructions. The outcome is kept in the first component after execution. For instance, ADD A, R1 - The accumulator will hold the result of addition (A+R1).

Data Transfer Instructions

Instructions for data transfer transferring the information from one register to another. The relocated content of the register does not change. If they end in "X" (MOVX), the information is transferred to an external memory.

Logic Instructions

Logic instructions operate on the corresponding bits from two registers using logic. Following execution, the outcome is kept in the first operand.

Bit-oriented Instructions

Bit-oriented instructions carry out logic operations in a manner similar to logic instructions. The distinction is that these operations are carried out on single bits.

8051 Memory Organization

Program Memory and Data Memory are the two sections of the 8257 microcontroller's memory. Although data memory (RAM) is utilized to temporarily store and maintain intermediate results as well as variables, programme memory (ROM) is used to permanently save programmes that are being run.

Program Memory (ROM)

Program Memory (ROM) is employed to store programmes (CODE) permanently while they are being run. The memory can only be read. A constant variable could also be stored in programme memory, dependent on the compiler settings. The 8257 only runs programmes that are stored in programme memory. Program memory was referred to via the code memory type specifier. External programme memory may be added thanks to the memory structure of the 8051. The logical state of pin EA determines how the microcontroller manages external memory.

External Data Memory

External memory access takes longer than internal data memory access. The amount of external data memory is up to 64K Bytes. There is on-chip XRAM space available in a number of8257devices, and it may be accessed using the same commands as the conventional external data space. This XRAM area, which overlays the external memory space, is normally activated by properly setting the SFR register. Before another access for external memory or XRAM storage is performed, that register must first be explicitly set in code.

Code Memory

The memory that contains the actual 8257 software that has to be executed is known as code memory. This memory can only hold 64K and comes in a variety of forms: On-chip code memory might well be found as ROM or EPROM that has been burnt onto the microcontroller. Additionally, code may be entirely off-chip stored in an external ROM and, more often, an external

EPROM. Another well-liked approach to programme storage is flash RAM. Additionally, other combinations among those memory types may be employed; for example, an EPROM may include 4K of code memory on again and 64K of code memory off-chip.

DISCUSSION

To evaluate the performance gains of the DMA module we developed a data intensive test program with a high frequency of memory accesses. The test program is basically a small Kernel that computes the dot product of two vectors A and B. Both vectors are stored in the disk and the processor first has to retrieve them and put them into the RAM before it can start performing the computation. Since both vectors are very large, the processor brings them into the RAM one part at a time. Each part is referred to as a page. This is basically to mimic the working of a complete operating system with the Virtual memory translation enabled. The pages are brought into the RAM on-demand. This demand driven process allows the processor to work in the foreground while the DMA can transfer the page in the background. We first run this program without the DMA and count the total number of cycles the program takes to execute. We then repeat the experiment with the DMA. In this version the processor instructs the DMA to load the subsequent pages of the vectors while it works on the current page. The processor keeps track of which pages have been moved into memory using a score boarding technique to ensure that it only computes valid data. That is, it makes sure the page has been transferred before it tries to use its' values. Each time the DMA interrupts the processor signaling that a transfer has been complete; the processor requests it to fetch the following page. This way the processor works on the data that has already been transferred into the memory while the DMA fetches the rest of the data. We performed this experiment several times while varying the disk latency. We wanted to see how the disk latency affects the performance boost of the DMA. Finally, we measured the number of cycles the processor sits in an idle state waiting for the data. This analysis was done for a latency of zero cycles.

CONCLUSION

In conclusion we successfully implemented a DMA module in System C. We successfully demonstrated the gains in both processor utilization and total number cycles with the use of a DMA controller. It is important to note that these gains can only be realized if the application is data intensive and has very high frequency of memory accesses. An important result that we found was that the DMA's effectiveness diminishes as the Disk Latency increases. If we had more time we would have liked to do an analysis of the bus utilization with and without a DMA. It will be interesting to see how much of a bottleneck the DMA was and to measure the performance losses due to bus congestion. The next step after this project is to design and implement the system with a 2-port memory for an extremely high-performance system. The results can then be used to see if the more expensive 2 port memory is worth the performance tradeoff.

REFERENCES

[1] M. A. Ahmed, A. Aljumah, and M. G. Ahmad, "Design and implementation of a Direct Memory Access Controller for embedded applications," Int. J. Technol., 2019, doi: 10.14716/ijtech.v10i2.795P.

[2] K. N. Muralidhara Professor, "International Journal on Recent and Innovation Trends in Computing and Communication Secured Smart Healthcare Monitoring System Based on Iot," Int. J. Recent Innov. Trends Comput. Commun., 2015.

[3] C. Iwendi, M. A. Alqarni, J. H. Anajemba, A. S. Alfakeeh, Z. Zhang, and A. K. Bashir, "Robust Navigational Control of a Two-Wheeled Self-Balancing Robot in a Sensed Environment," IEEE Access, 2019, doi: 10.1109/ACCESS.2019.2923916.

[4] A. R. Al-Ali, R. Gupta, and A. Al Nabulsi, "Cyber physical systems role in manufacturing technologies," in AIP Conference Proceedings, 2018. doi: 10.1063/1.5034337.

[5] J. Jung, J. Kwon, J. Mfitumukiza, S. Jung, M. Lee, and J. Cha, "IoT Enabled Smart Emergency LED Exit Sign controller Design using Arduino," Int. J. Adv. smart Converg., 2017, doi: 10.7236/ijasc.2017.6.1.76.

[6] J. A. J. Alsayaydeh et al., "Development of vehicle door security using smart tag and fingerprint system," Int. J. Eng. Adv. Technol., 2019, doi: 10.35940/ijeat.E7468.109119.

[7] S. Hrishikesan, "Block Diagram of Microcontroller 8051," 2019.

[8] I. J. Chung, "The efficient implementation of the multi-channel active noise controller using a low-cost microcontroller unit," J. Acoust. Soc. Korea, 2019, doi: 10.7776/ASK.2019.38.1.009.

[9] C. Today, "8051 Addressing modes," 2018.

[10] A. A. Fuadzi, I. Santosa, and G. R. Wilis, "Instrumen Kendali Mesin Cnc Portable Berbasis Microcontroller Arduino Dan Modul Cnc Shield," 1st Mech. Eng. Natl. Converence, 2018, 2018.

CHAPTER 10

AN OVERVIEW OF RANDOM ACCESS MEMORY IN MICROPROCESSOR

Mohan Kumar A V, Assistant Professor,
Department of Computer Science and Engineering,
Presidency University, Bangalore, Karnataka, India
Email Id- mohankumar.av@presidencyuniversity.in

ABSTRACT:

There are two basic kinds of memory used in microprocessor systems - commonly called Read Only Memory and Read / Write Memory, but more usually called ROM and RAM - "Read Only Memory" and "Random Access Memory". What is RAM used. RAM is a common computing acronym that stands for random-access memory. Sometimes it's called PC memory or just memory. In essence, RAM is your computer or laptop's short-term memory. It's where the data is stored that your computer processor needs to run your applications and open your files.

KEYWORDS:

Memory Hierarchy, Magnetic Random Access Memory, Non-volatile Memory, Semiconductors, Embedded Systems,

INTRODUCTION

Only while the system is switched on can volatile memory keep its data. Since non-volatile memory is easily accessible, do you utilize volatile memory at all, Non-volatile memory is often slower, harder to use, and more costly, which is the issue here. While computer computer's volatile memory may be accessed in as little as a nanosecond, certain non-volatile memory types won't be accessible for milliseconds after receiving only one subpar byte [1].

Static RAM

Static Random Access Memory (SRAM), which predates the development of integrated circuits within computers, was the first kind of volatile memory to become extensively used. Each cell on to an SRAM chip has the capacity to store one piece of information. A so-called flip-flop, consisting essentially consists of six transistors, is employed to store a little amount of information. Let's simply think of the cell as just a black box for the time being since understanding its internal structure is not within the purview of our course.

Dynamic RAM

Contrary to popular belief, no one will ever need more that 640 kilobytes of RAM, there is always a shortage of memory. Naturally, they want to squeeze as much storage as possible can from a memory chip that has a given size. Students already know that six transistors are typically required in SRAM to store a single piece of information. Naturally, the silicon area will increase as the number of transistors per cell increases. One could increase the storage capacity by roughly twofold if we could cut the amount of components required, for example, by using just half as

many transistors. That is what Dynamic Random Access Memory, or DRAM, has accomplished: One transistor was all that was required for each piece of information. Of course, this decreased the silicon area for a specific cell count. A DRAM hence has a much higher storage capacity than an SRAM with the same chip size. It should be clear by now how a DRAM functions: Activating the transistor addresses the memory cell you wish to access in order to maintain a logical one. After that, simply supply a voltage towards the capacitor to charge it. Then choose the cell then discharge the capacitor to store a logical zero. Well, all have to do is make sure the capacitor is charged. Simple. Of course, that would be too easy; there is however a catch. There are really a few drawbacks, the most irksome of which is that the chip doesn't have complete insulation. The capacitor should maintain the charge after it is charged, in theory. The capacitor still loses its charge while not being accessible because very small currents referred to as leakage currents pass through the chip's imperfect insulators. Additionally, the capacity of these capacitors is low due to their relatively tiny size. This indicates that the charge will inevitably diminish once the capacitor has been charged.

Non-volatile Memory

Non-volatile memories, in contrast to SRAMs and DRAMs, maintain their content despite the fact that power is interrupted. However, as was previously said, this benefit has a cost: Non-volatile memory type writing is often significantly slower and more difficult, and frequently just plain frustrating.

Programmable Read Only Memory (PROM)

According to the size of the MROM, employing ROM is obviously only a possibility for mass manufacturing - maybe hundreds of thousands of units. The setup cost for this kind of production run is too high for prototypes. PROM, or programmable read-only memory, is an option. Essentially, they are matrix of memory cells, each of which includes a silicon fuse. Each fuse is complete at first, and each cell seems to be logical. A logical 0 may be programmed into a cell by choosing it and delivering a brief but intense current pulse that blows the cell's fuse. Visitors may come across so-called One Time Programmable (OTP) [2] microcontrollers from time to time. These have PROM as its on-chip instruction memory. Of course, PROMs and OTP micro - controllers really aren't appropriate for development, as the memory's contents may need to be modified. However, once development is complete, they are ideally suited for medium range mass manufacturing, provided that the quantities are low enough to make the creation of MROMs unprofitable[3].

Erasable Programmable Read Only Memory (EPROM)

Changes are often required even after the original development is complete and the items are in use. With ROMs or OTP microcontrollers, on the other hand, the memory content cannot be changed; the IC itself must be replaced. EPROM gets around this problem. Programming is not detrimental in this case. The so-called field effect transistor (FET) or, more specifically, the gate pin on a FET is where memory is kept. Given that it is fully isolated from the rest of the circuit, it is appropriately called a floating gate. However, it is feasible to energize the floating gate using a physical procedure known as avalanche injection by using a sufficiently high voltage. Therefore, electrons are introduced into the floating gate to close the transistor switch rather than burning fuses. The electrons should always stay in the floating gate after a cell has been programmed. Nevertheless, the imperfect insulators only experience a small amount of leakage currents, similar

to DRAMs. The floating gate gradually loses sufficient electrons to become programmed. The maker of the EPROM indicates how much the memory contents will be intact in the datasheet; typically, this is a duration of roughly 10 years. However, this short lifespan of EPROMs really works to their advantage: The process may be quickened by shining UV light on the silicon chip. The floating gates will have been discharged by the UV light and the EPROM will have been wiped after around 30 minutes. Because of this, EPROMs feature a tiny glass panel in their packaging that allows users to see the chip. This window is often protected from light by a seal. Remove the seal and exposed the chip to powerful UV light to delete the EPROM (The EPROM is often erased using an EPROM eraser, which involves placing the EPROM in a light-proof box and exposing it to UV radiation since ultraviolet light is potent enough just to permanently harm human vision).

Electrically Erasable and Programmable (EEPROM)

Programming EPROMs, and specifically the erasing procedure, is a labor-intensive operation. A unique programming voltage that is often greater than the operational voltage is utilized to programme them. An UV light source is required to remove them. Undoubtedly, technology needed to advance. All the benefits of an EPROM are available with the EEPROM (Electrically Erasable and Programmable ROM) without the inconvenience. No specific voltage is needed any more for reprogramming, and, as the name suggests, no UV light source is necessary any longer for erasing. With the exception of the ability to remove the electrons from of the floating gate by employing a high voltage, EEPROM functions very similarly to EPROM. When we said that no particular voltage was required, we got a bit carried away: Although a larger voltage is still required, it may be produced on-chip by so-called charge pumps, which can provide the chip with higher voltages than are delivered outside [4].

Flash

EEPROM now seems to be the best option for non-volatile memory. One disadvantage is that it is rather pricey. Flash EEPROM is an option as a workaround. Flash is a kind of EEPROM wherein erasing is only feasible for bigger blocks or even for the full memory (erased "in a flash," so to speak), rather than for each address. In this manner, the underlying logic is streamlined, resulting in a significant price reduction. Additionally, Flash EEPROM is often utilized for program memory rather than data memory because to the inability to delete individual bytes. As a result, lesser durability is acceptable; although a data EEPROM may be updated often, a microcontroller's program Flash is typically not updated 100,000 times. As a result, compared to EEPROMs, Flash-EEPROMs frequently have lower guaranteed write/erase cycle endurance between 1,000 and 10,000 cycles. This also lowers the price of Flash-EEPROMs.

Non-Volatile (NVRAM)

Non-Volatile RAM is a kind of memory that combines the benefits of volatile and non-volatile storage (NVRAM). This may be done in a variety of ways. One is to simply include a tiny internal battery inside an SRAM device, ensuring that the SRAM keeps its data even when the external power source is turned off. Another option is to package an SRAM and an EEPROM together. Data is transferred from the EEPROM towards the SRAM during startup. Data is read from and transferred to the SRAM when it is in use. The data is transferred to the EEPROM before power is lost.

Accessing Memory

A lot of microcontrollers have on-chip data and programme memory. The data memory is often made up of some SRAM as well as some EEPROM, while the programme memory is typically of the Flash-EEPROM variety. Essentially, there are two approaches:

Each Memory is Addressed Separately

Figure 1 shows that the three distinct memory types may share the same address ranges. Different connectivity techniques are used by the programmer to specify which memory is going to be accessed. For instance, a particular EEPROM-index register is utilized to access EEPROM.

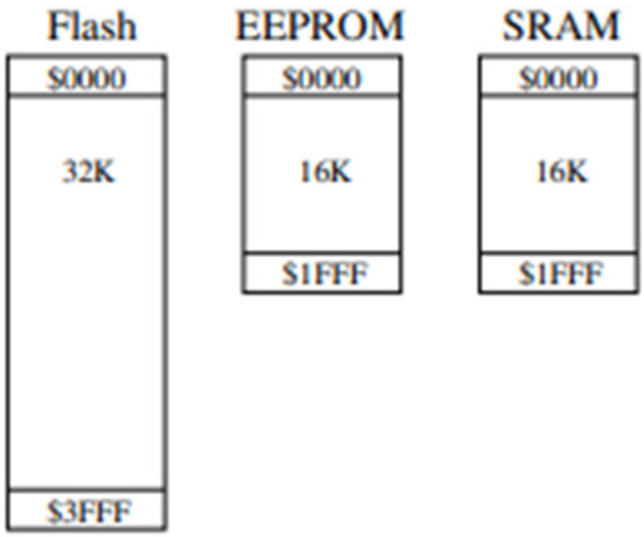


Figure 1: Illustrates the Separate Memory Addressing Ranges.

Memory types Share a Common Address Range

Similar to how SRAM is accessed by the programmer, EEPROM is also. The address is used by the microcontroller to choose which recollection to access. For instance, SRAM may appear in the address range 0x2000 - 0x3000, whereas EEPROM may appear in the range 0x1000 - 0x2000. Now that the microcontroller recognizes address 0x1800 as being in the EEPROM range, it will access the EEPROM when the programmer reads this address. Although this procedure is incredibly simple, it is additionally inherently riskier: The improper kind of memory may be accessed by using the incorrect address. This would be particularly risky if you accidentally accessed the EEPROM rather than the SRAM since, with regular access, the EEPROM may quickly become unusable. On the other hand, separate memory addressing has an inherent safeguard against access to the incorrect kind of memory see Figure 2.

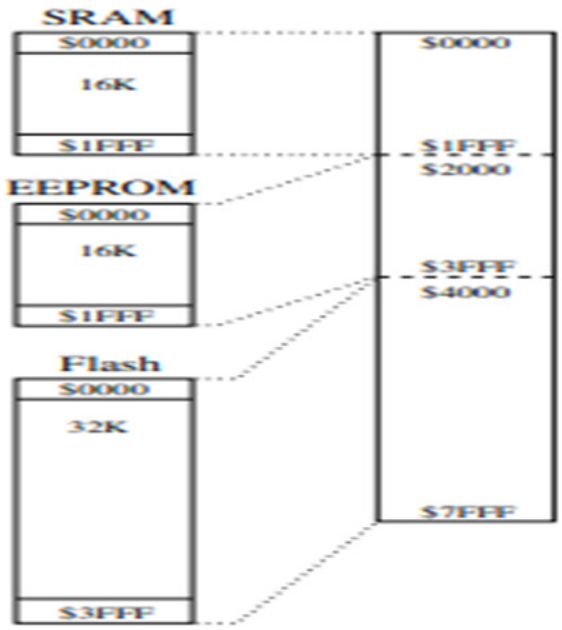


Figure 2: Illustrates the Different memory types mapped into one address range.

Digital I/O

The primary attribute of microcontrollers is digital I/O, or more generally, the capacity to directly control and monitor equipment. Because of this, almost all microcontrollers feature at least a few digital I/O pins that can be attached to hardware description language (within the electrical limits of the controller). On average, most controllers contain 8 to 32 pins, although some have much more (such as Motorola's HCS12, which has over 90 I/O pins). I/O pins are often arranged in ports of eight pins, each of which may be accessed with such a single byte. Pins may either be input-only, output-only, or—most frequently—bidirectional, which means they can both accept and send signals. Most pins contain one or more additional purposes than digital I/O in order to save pins & keep the chip compact. The analogue module and the timer are only two examples of different controller modules that employ other uses for digital I/O pins. By turning on the relevant module's capabilities, the application programmer may decide which function the pin should be utilized for. The hardware designer must carefully pick which pins to utilize for specific purposes since, of course, if a pin is utilized for the analogue module, it is wasted for digital I/O & vice versa. We'll focus on pins' capabilities for digital I/O in this section. The alternative functions will be covered in later sections. Let us first define what they mean by "digital": When using a voltmeter to measure the voltage levels of a pin (with respect to GND)[5].

Registers Control

As far as digital I/O is concerned, three registers control the behavior of the pins:

Data Direction Register (DDR)

With one bit for every pin of the port, also every bidirectional port contains its own DDR. Clearing or changing a pin's DDR bit determines whether it is an input or an output pin. It is acceptable to have three pins of a port set as outputs and utilize the remaining five pins as inputs since various

ports might be configured differently. The DDR values are typically initialized to input after a reset. The value is returned by reading the register.

Port Register (PORT)

This register is employed to regulate the output pins' voltage levels. If a pin has been programmed to output, the value will be high if its bit inside the PORT register is set and low if the bit is cleared. It is often advisable to utilize the controller's bit procedures to prevent changing a specific bit from overwriting some other bits inside the port. In the absence of this, you must utilize a read-modify-write access thus that must make sure it is not stopped. Reading the register with output pins yields the value originally entered. The functioning of input pins is determined by the controller. Through the port register, certain controllers let users view the status of the input pins. You will read back whatever value previously sent to the register because certain controllers, like the ATmega16, utilize the port bits for different functions if the relevant pins were set to input.

Port Input Register (PIN)

The current status (high or low) of every pin, regardless of whether they are set up as outputs or inputs, is stored in the PIN register, which is typically read-only. It is employed to read input pin states, but it may also be used this to read output pin states to make sure the output was switched over properly. Typically, a write towards this register has little impact.

Digital Input

When the observed signal should indeed be interpreted electronically, that is, when it merely alternates between the two states "high" (corresponding to logic 1) and "low," the digital input capability is employed (corresponding to 0). The controller's requirements, which are influenced by the operating voltage of a controller, determine whether a particular signal should be regarded as high or low.

Digital Sampling

The topic of how a voltage value is converted into a binary number inside a register arises that's because a digital signal merely represents a voltage value. The first approach would be to simply latch the pin's current state into the PIN register using latches. If the system clock activates the latch, it will save the current state just at start of each cycle. This naturally implies that we can only detect a state change after it has already happened because we are able only sample with both the granularity of a system clock. If an impulse is less than a clock cycle per Figure 3, it may even completely ignore it [6].

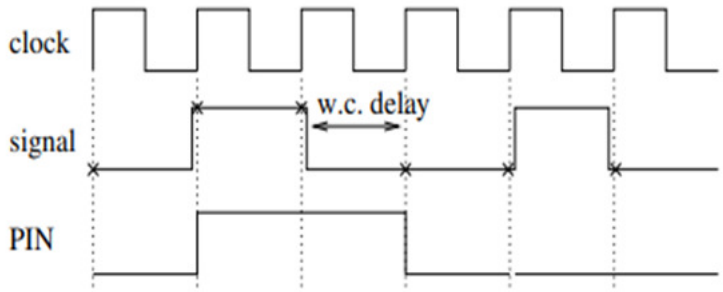


Figure 3: Illustrates the Sampling an input signal once every clock cycle [7]**.**

The sampling granularity causes a delay of [0,1] clock cycles (dlatch). Because it is unknown what will happen whenever a signal changes during the same time as that of the sampling clock edge, zero is left out of this equation. It may be tasted or it could not be. Consequently, it is wise to exclude 0 from the interval. Using the same logic, impulses need to be longer than what a clock cycle to also be positively identified. In the remainder text, the input delay interval will be denoted as din = (d min in, d max in), within which d min in represents the lower limit of the input delay and d max in represents its upper bound.

Schmitt-trigger

Schmitt-triggers are devices that enable "digitalization" of analogue input signals. The Schmitt-trigger does this by having two threshold voltages, Vlo and Vhi, which are opposite to one another, and only changing its output from logical 0 to logical 1 if indeed the input signal exceeds Vhi. Therefore, the signal must be below Vlo for the Schmitt-trigger to switch from 1 to 0. As a consequence, regardless of an input signal shown Figure 4, the Schmitt-trigger somehow doesn't forward tiny voltage changes as well as its output has always had brief and distinct rising and falling durations?

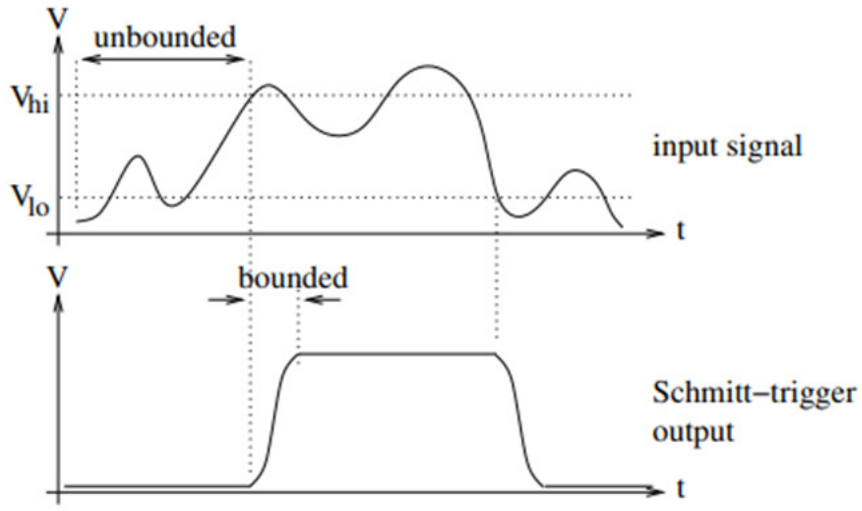


Figure 4: Illustrates the Input and Output of a Schmitt-trigger.

Noise Cancellation

Although the controller's PIN register should typically track the input pin's condition as accurately as possible, this is very undesirable if the signal is noisy. Sometimes, electromagnetic interference from either the surroundings cause brief voltage spikes upon that line. Those voltage fluctuations shouldn't typically be controlled by the controller since they could result in incorrect responses, particularly when combined with interrupts. Some controllers thus have noise cancellation. If enabled, the pin is sampled by the controller more than once say, k times and it only accepts a new value when all k samples were equal. Clearly, this increases the input delay by another constant decant = k 1 cycles, making the delay bounds clock cycle.

$$d_{in} = d_{latch} + d_{sync} + d_{ncanc}$$

Pull Resistors

Pull resistors are often used into the input circuitry of controllers. They typically supply pull-up resistors, although some controllers also include pull-down resistors (e.g., the HCS12). In the event that the input pin also isn't driven by that of the external devices, the pull resistor's job is to link it to a certain voltage. Pull resistors are managed via a register, which allows them to be separately enabled or deactivated for every pin. For example, the ATmega16 controls the pull resistors on input pins by using the PORT register bits. For this use, several controllers provide specific registers. When employing basic mechanical switches, such as DIP switches or buttons, it might happen very commonly that such an input pin is not always driven by hardware. The input pin gets connected to a particular value whenever the switch is closed, whereas when the switch is activated, it is left floating (that is, unattached and at an unspecified voltage level). In Figure 5, a pull resistor is required to set the pin to a certain level when the switch is open because floating pins are Indeed a Bad Thing (they are highly susceptible to noise!).

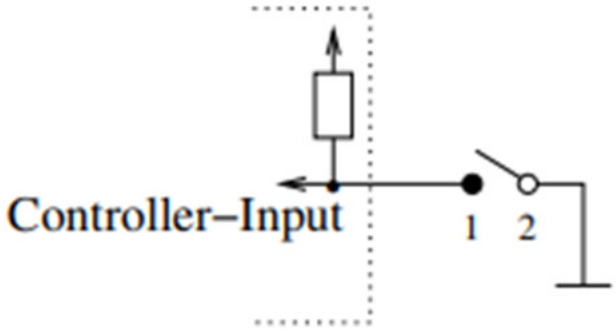


Figure 5: Illustrates the Attaching a Switch to an Input Pin with Activated Pull-up Resistor [7]**.**

It attached the switch to an open-drain input, also known as an input with active pull-up. The input pin is attached to VCC when the switch is open, allowing the controller to read a. When the switch is closed, the pin is connected to ground, as well as the controller displays 0. There is yet another intriguing development coming when the switch is closed, the pull-up resistor causes current to flow from the controller through the use of the input pin to the exterior ground. Because the controller shouldn't be able to affect the external circuit only by reading it, there wouldn't be any significant current flowing to or from the controller pin without the need of a pull resistor. However, when the pull-up is turned on, the controller actively decides the state of the line, and as a result, current flows here between input pin and the external electronics. The input is referred to as a source input because that supplies current if current4 passes from the controller towards the external circuit. A sink input is one where current flow from of the device into the controller. The amount of electricity that controllers can source and drain is quite specific, and going beyond the limits listed in the datasheet risked damaging the pin or perhaps the controller itself. The typical range of current handling for controllers is between 4 and 20 mA, and if they distinguish between sources and sinking at all, transistors can typically sink more current than they are able source.

Digital Output

The output pins are tuned to specific voltage levels using the digital output capabilities. Once again, the controller determines the levels that correspond to high and low and these levels are based on the operational voltage of the controller. The highest output low voltage and lowest output high voltage for the ATmega16 at VCC = 5V are 0.7 V and 4.2 V, respectively. The controller drives a pin in accordance with the value stored in the PORT register however when the DDR of the pin is set toward output. We may again differentiate between a sink output as well as a source output since an output pin often needs to drain or source current. They are referring to the maximum current ratings that were covered in the previous section, which is 4-20 mA. Since output pins mainly rely on external current protection, there are more crucial than input pins. After all, you might set an output pin to 1 and connect it straight to GND, causing a short-circuit. Even though such short-circuits are often tolerated by controllers for a small period of time (usually less than a second), a short will ultimately cause the controller to malfunction. In order to prevent short circuits, the hardware designer should make sure that now the external hardware never cause them. The controller at least provides the option to read back the pin's current state through the use of the PIN register if it can or when the application programmer wishes to be cautious. A short-circuit won't be apparent in the PIN register until after the output pin has indeed been set since the PIN register naturally experiences input delay. As a result, the programme must wait this long before checking the PIN. The application software should quickly set the pin to input and alert the user if a mismatch is found. Be aware that certain microcontrollers just drive the 0 and generate the 1 via an open-drain input[8].

Finally, they want to bring up the issue of whether register PORT or DDR should be set first for output ports. The pin is often set to input after a reset. The solution is obvious: simply first set the PORT followed by the DDR, guaranteeing that the right value is placed on the line from the beginning. This is true if your controller does not utilize the PORT bits of input pins for other reasons (like the ATmega16, who utilizes them to regulate the pull-ups). Unfortunately, the situation is more complicated with Atmel's AVR controllers, such as the ATmega16. Because the pull-up resistors are controlled by the PORT in this case, you must momentarily enable them before setting the port to output if you wish to output 1 and first set the PORT before setting the DDR. Although most of the time this won't matter, one should still research the hardware to ensure that allowing the pull-ups seems to have no unfavorable impacts. A wise hardware designer will ensure that you can set the two registers in either direction by designing the external circuit so that the equipment's default state, whenever it is not powered by that of the controller (this happens, for example, during a controller reset, so the hardware designer must account for it), is identical to when the controller pin outputs its own PORT reset value. In this approach, if users set the pin to output initially, its value won't change, and you may set the PORT pin whenever they want. In conclusion, put PORT first of all and DDR second if the controller permits and the hardware is unconcerned. However, always make sure that this has no negative side effects.

DISCUSSION

Microcontrollers are often used in systems that must respond to events. Events represent changes throughout the controlled system's state and typically call for a response from the microcontroller. Simple reactions include increasing a counter every time a workpiece passes across a photoelectric barriers on such a conveyor belt, while time-sensitive actions include shutting down the machine if someone approaches into a machine's working area. The dilemma about how the controller

should watch the input line remains even if it is assumed that the controller could observe the occurrence, that is, whether there is an input line that modifies its state to signify the occurrence. Of course, it is feasible to simply poll that input signal, or monitor for state changes on a regular basis. But this polling has certain limitations: If the event only happens sometimes, it not only wastes unnecessary processing time but is also challenging to alter or expand[9], [10].

The event is polled frequently so that the remainder of the programme may be run as well since a microcontroller often has a lot more to accomplish than just wait for a single event. On the other extreme, the polling function may need to be run several times during the main programme if the signal has to be polled with a specific maximum duration to prevent missing occurrences. Determining the places in the code where the signal should indeed be polled in the initial place takes time, and these locations must always be reevaluated anytime the main changes made. As a result, polling quickly loses its appeal, and the software developer begins exploring for other methods to deal with these rare occurrences. Fortunately, the interrupt feature of the microcontroller itself provides a practical solution. In this case, the microcontroller polled the signal and only interrupts the main programme upon the detection of a condition change. The main programme just runs without thinking about the event as long that there is no condition change. The microcontroller invokes an interrupt service routine (ISR) to handle the event as soon as it happens. The application programming is required to deliver the ISR.

Interrupt Control

The primary interface towards the interrupt logic of a microcontroller is formed by two bits: The application programmer sets the interrupt enable (IE) bit to signal that the controller should invoke an ISR in response to the event. The interrupt flag (IF) bit is activated by that of the microcontroller however when the event takes place, and it is either explicitly cleared by the programmer or automatically removed when exiting the ISR. In essence, the IE bit permits the interrupt itself to occur whereas the IF bit indicates that the interrupt condition has happened. For each interrupt sources the controller has, the IE and IF bits are typically given. However, many identical interrupt sources may sometimes be mapped to a single IE bit in order to conserve bits. Each individual input of a digital I/O port, for instance, on the Motorola HCS12 microcontroller, has the potential to cause an interrupt. However, the whole port only has one ISR since there is only single IE bit. The real reason of the interrupt may be found there since each pin here on port has a distinct IF bit. The controller will probably provide extra control bits (interrupt mode) for a number of its interrupt resources in addition to the IE and IF bits.

They are used to determine which specific signal alterations should result in an interruption (e.g., only a falling edge, any edge,). It is occasionally even feasible to respond to an input signal that hasn't changed. Having a level interrupt is what this is. A microcontroller additionally has one global interrupt disable bit that enables/disables all presently enabled interrupts since it would be cumbersome and time-consuming to deactivate all already enabled interruptions however when the programme function should not be stopped by an ISR (atomic action).

The global IE flag as well as the IE bit again for interrupt source must both be enabled in order for an ISR to be called. Always verify whether the global IE bit should be reset or cleared to allow interrupts since "enabled" does not always equate to "set" in this scenario. Disabling interrupts doesn't really guarantee that events will be missed. Irrespective of whether the IE bit is set or not, the appearance of an event is kept in its IF (this refers to both the global and local IE). Therefore, the relevant ISR will be invoked, although somewhat belatedly, if an event happens while its

interrupt is disabled and is subsequently enabled again. Only when a second event happens before the first has been handled will anyone miss events. In this scenario, the second event (or the most recent event) will be dealt with, but the first event (or all earlier events) would be lost. However, if the interval between events has a lower limit, it is assured that no event will be missed if all atomic portions are maintained within the smallest lower constraint. Some controllers include non-mask able interrupts (NMI), which are in addition to the standard interrupts that may be deactivated, but which are not affected by the global IE bit. These interruptions are beneficial when there is a need to respond immediately to an urgent incident, whether or not it has an impact on the programme. In the Texas Instruments MSP430 series, the NMI may have a distinct control bit to activate or disable it. Interrupts are typically turned off globally and at the source after a reset. The application programmer should understand, nevertheless, that the startup code produced by the comp.

Interrupt Priorities

The challenge of how to handle instances when two or more interrupt events happen at once arises because a controller has several interrupt sources and may in reality support a variety of interrupts. This is more common than you may imagine, particularly if the software sometimes disables interrupts. Therefore, it is necessary to provide a predictable and sane method for choosing which interrupt to handle next. The interrupt vector is used by the majority of controllers with a vector table and several interrupts as a sign of priority. For instance, the ATmega16 statically gives the interrupt with the lowest interrupt vector the greatest priority. The likelihood is that NMIs will be given first priority if the controller provides them. Priorities, however, may be utilized to decide if an interrupt will stop an ongoing ISR in addition to deciding who wins the race: An interrupt with a higher priority in these systems will, if it is enabled, interrupt the ISR of an interruption with a lower priority (nested interrupt). As longer as the interrupt enable bit on another controller, such the ATmega16, is set, any interrupt may interrupt an ISR. Since an interruption isn't always wanted, many controllers deactivate the global IE bit when starting the ISR, or they provide the ISR some other method to decide whether or not it should be interrupted. Of fact, a fixed prioritization may not always correspond to the needs of the application programme. As a result, some controllers let the user adjust precedence to at least certain interrupts on the fly. Others let the user choose which interruptions inside the ISR should be permitted to interrupt the ISR.

Interrupt Handling

Naturally, a controller must provide the ability to manage interrupts if it supports interrupts. This requires hardware to first recognize the event and a method to contact the ISR. Condition The controller samples the input signal there at start of each cycle as described in order to be capable to detect an external event, and it then compares the sampled result to its previous value. The interrupt flag is activated if an interrupt situation is found. If the IE bit is set while no other interrupt with such a higher priority is waiting, the interrupt logic subsequently determines whether to raise it, which calls the ISR. It should be noted that the sampling circuit delays the event's detection, as well as the signal must be constant for more than one clock cycles in order for the controller to always notice the event. Longer signals may or might not be seen. The undesirable aspect of external events is that they are produced by external hardware, which is frequently wired directly to the controller via unshielded circuits. Therefore, small spikes on the line may cause edges even if the related hardware did not produce them if the interrupt state is an edge here on input line. As a result, this noise generates spurious interrupts, which are famously annoying sources of mistakes

and odd behavior in programmes due to their rarity. Because of this, it is very hard to trace down the source of these interrupts (they just never happen when you are searching). Some microcontrollers include noise cancellation for various external interrupt sources to stop this noise from influencing your software. When enabled, the controller sampled the line 2k times and only detects an edge, for instance, if the first k samples all return a value of 0, and the following k samples all return a value of 1. Obviously, this eliminates small spikes on the wire but delays edge detection about k 1 cycles. The appropriate component provides the equipment to set the IF for internal events, such as timer events or notifications that a byte has indeed been received through the serial port. The regular interrupt logic may then take over from there. It goes without saying that internal events are unaffected by noise and don't need any mitigation in this respect [7].

Calling the ISR

Calling the ISR requires more than merely navigating to the right address, despite what they may have implied up to this point. As with any other function, the controller must first store the return address here on stack. Some controllers also preserve registers, as well. The controller can clear the interrupt flag perhaps if single interrupt source is obviously assigned to the vector. The controller often turns off interrupts by turning off the global IE, which is crucial. This offers the ISR the opportunity to run continuously if the programmer so chooses. The global IE bit may be enabled again in the ISR to serve any further interrupts that need to be handled. One should only use such nested interrupts, however, if absolutely required since they may lead to some severe and difficult-to-find issues. The microcontroller performs the first instruction of a ISR after it has accomplished all of its housekeeping tasks. Except that you might not be able to perform certain blocking system calls if you are executing under an operating system, there aren't many differences between a conventional subroutine and an ISR inside the ISR. The key distinction between the ISR and the subroutine the fact that the ISR must be terminated using a unique "return from interrupt" (RETI) instruction that reverses whatever actions the controller took prior to executing the ISR: The return address is loaded into the PC, the global IE bit is enabled, and any saved registers may be restored. Some controllers, such as the ATmega16, ensure that at least one main programme instruction is carried out after an ISR returns before calling the subsequent ISR. As a result, even if there are many interruptions, the main application won't starve even if its execution would be sluggish.

To sum up, interrupt management is carried out in the following ways starting with the event's detection: Interrupt flag set: The interrupt condition's occurrence is recorded by the controller in the IF. Finished with the lesson? It is often simpler to merely finish the current instruction before responding to the event since aborting half-completed instructions confuses the hardware. Of course, this adds one or more phases to the delay before a response to the event. It won't be necessary for the controller to complete an instruction if it was in sleep mode whenever the event happened, but it will take some time for it to awaken. If the controller has to await for its oscillator to stabilize, this period of time might extend to several milliseconds. Determine ISR: An ISR should not always be called simply because an event occurred. The client does not want an interrupt if the matching IE bit also isn't set. Additionally, several IF flags may be set because the controller has many interrupt sources that may generate events concurrently. Therefore, among all sources having set IF and IE bits, the controller must identify the interrupt source with both the greatest priority. Call ISR: Following the selection of the beginning address, the controller stores the PC and other data before executing the ISR.

The whole series of events from the moment they occur until the first instruction in the ISR is executed results in a delay in the response to the event, which is included into the interrupt latency. Depending on the particular actions taken by the controller prior to executing the ISR, the delay typically ranges between 2 and 20 cycles. It is important to keep in mind that if instructions may be stopped, the latency will vary depending on which instruction were interrupted and where. The interrupt latency is a crucial component of the microcontroller especially time-critical applications; thus its upper limit is often given in the documentation (underneath the assumption that the interrupt is activated and that there aren't any other interruptions that might delay the call to the ISR). Since at minimum the PC must be stored, a minimum delay is essential, but storing registers here on stack may cause the latency to increase needlessly, therefore comparing various controllers may be helpful. If latency is a problem, seek for controllers that offer rapid instructions (only requiring one or a few oscillator cycles), can be clocked quickly (high oscillator frequency), and that do not keep registers on the stack. And besides, the application programmer need not waste time saving unnecessary registers since they may store the appropriate registers in the ISR.

Interrupt Service Routine

The code required to respond to the interrupt is included in the interrupt service procedure. If the interrupt flag hasn't previously been removed, this could include clearing it. If indeed the interrupt is no longer needed, it might also entail deactivating it. The programme that responds to the incident that caused the interrupt may also be found in the ISR. A decent design, however, needs a lot of effort and expertise since deciding what to accomplish in the ISR and what to perform in the main programme is sometimes difficult and dependent on the scenario. Although they can't make anyone possess any of these qualities, but can at least get you started by highlighting a few factors that should take into account. They sometimes refer to ideas that are presented in subsequent parts in the examples that follow. So, if you don't understand anything, simply go on; you can always go back and read it after you've finished the rest of the material.

Polling

Consider of a button that is pressed by a person as an illustration. A dc motor should be running for the duration that the button is depressed. This seems to be a polling solution at first sight because we are concerned in the state. This is not a fair argument, however, since if we first verify the state, then may deduce the present state from its state changes. The decision really largely hinges on other factors in their application. You may think about using interruptions and putting that controller into sleep mode in main if there is nothing better to do. The user won't notice if indeed the dc motor is shut off once or twice in the initial few milliseconds of operation, thus button bouncing is not really an issue here. Polling is preferable due to its simplicity if the main programme has a manageable number of tasks to complete. You don't need to be concerned about how inaccurate your solution is. The time is not crucial since neither the human nor the dc motor are very exact instruments. One can easily get by with checking every few milliseconds and fitting a lot of additional code within a period of 1–10 ms.

CONCLUSION

In this chapter, we have demonstrated that STT-MRAM can replace Static Random Access Memory (SRAM) in the memory hierarchy of a microprocessor. We achieve this conclusion based on our memory hierarchy evaluation methodology flow. Using this flow, we have investigated the performance and power characteristics of STT-MRAM, when used as a replacement for SRAM.

Although STT-MRAM has higher latencies, the lower leakage power of STT-MRAM, as compared to SRAM, makes it an attractive candidate replacement technology. Current results indicate that it could be a solution to address the rising power consumption of CMOS circuits. The use of STT-MRAM enables the possibility of new techniques for the implementation of power-saving mechanisms. The non-volatility could be explored to power-off the devices whenever they are idle. Furthermore, non-volatile memory arrays do not need refreshing, reducing the dynamic power and leakage. Independently of our results, the physical properties used in our evaluation of STT-MRAM produce two possible pathways for integrated circuit design. In one, we can conserve the total silicon die area and increase the amount of memory at least four-fold, or we could maintain the same amount of memory, but increase the production yield of the circuit four-fold by switching from SRAM to STT-MRAM. Given our evaluation and the benefits STT-MRAM brings to integrated circuit design, we conclude that STT-MRAM is a strong candidate to replace SRAM in the memory hierarchy of microprocessors.

REFERENCES

[1] X. Dong, X. Wu, Y. Xie, Y. Chen, and H. Li, "Stacking magnetic random access memory atop microprocessors: An architecture-level evaluation," *IET Comput. Digit. Tech.*, 2011, doi: 10.1049/iet-cdt.2009.0091.

[2] A. Afuah, "Strategies to turn adversity into profits," *IEEE Eng. Manag. Rev.*, 1999.

[3] R. Sato *et al.*, "High-Speed Operation of Random-Access-Memory-Embedded Microprocessor with Minimal Instruction Set Architecture Based on Rapid Single-Flux-Quantum Logic," *IEEE Trans. Appl. Supercond.*, 2017, doi: 10.1109/TASC.2016.2642049.

[4] S. M. Yakout, "Spintronics: Future Technology for New Data Storage and Communication Devices," *Journal of Superconductivity and Novel Magnetism*. 2020. doi: 10.1007/s10948-020-05545-8.

[5] D. Armstrong, "The social life of data points: Antecedents of digital technologies," *Soc. Stud. Sci.*, 2019, doi: 10.1177/0306312718821726.

[6] C. Weulersse *et al.*, "Contribution of Thermal Neutrons to Soft Error Rate," *IEEE Trans. Nucl. Sci.*, 2018, doi: 10.1109/TNS.2018.2813367.

[7] R. Bannatyne and G. Viot, "Introduction to Microcontrollers," in *Microcontrollers*, CRC Press, 2009, pp. 1–14. doi: 10.1201/9781420077681.ch1.

[8] J. Crenne, R. Vaslin, G. Gogniat, J. P. Diguet, R. Tessier, and D. Unnikrishnan, "Configurable memory security in embedded systems," *Trans. Embed. Comput. Syst.*, 2013, doi: 10.1145/2442116.2442121.

[9] M. Qiu, Z. Ming, J. Li, K. Gai, and Z. Zong, "Phase-Change Memory Optimization for Green Cloud with Genetic Algorithm," *IEEE Trans. Comput.*, 2015, doi: 10.1109/TC.2015.2409857.

[10] B. Zhang, Y. Wang, S. Tu, and Z. Jin, "FPGA-based real-time digital solver for electro-mechanical transient simulation," *Energies*, 2018, doi: 10.3390/en11102650.

CHAPTER 11
AN ANALYSIS OF PARALLEL MICROPROCESSOR SYSTEM

Shilpa C N, Assistant Professor,
Department of Computer Science and Engineering,
Presidency University, Bangalore, Karnataka, India.
Email Id- shilpacn@presidencyuniversity.in

ABSTRACT:

In computing, parallel processing pertains to the use of two or more processors (CPUs) to handle individual elements of a larger operation. The amount of time it takes to execute arbitrary code may be decreased by splitting up a task's many sections across several processors. In a parallel system known as a multiprocessor, each processing has direct access to shared memory, which together comprise a single address space. A single computer with many processing, a number of computers linked by something like a network to create a parallel processing cluster, or a composite of both may all be used continuously by parallel systems.

KEYWORDS:

Control Processor, Global Memory, Local Memory, Parallel Processing, System Control.

INTRODUCTION

The objective of a research project on parallel processing at the Los Alamos National Laboratory is to investigate if it has the capacity to rapidly perform complex calculations in fields includes hydrodynamics, Monte Carlo methods, reactor safety studies, and nuclear waste disposal. The percentage of speedup in parallel processing (the difference between the time it takes to complete an operation using one processor vs p processors) dependent on both the parallelism of the code and the number of processors. While software companies could be able to provide such systems by 1990, there are currently no systems with 16 or more processors. Moreover, no systems with floating-point calculation, big memory, or FORTRAN programmability are now available in the United States. In order to evaluate the parallel processing of big codes on diverse multiprocessing architectures, we are establishing an experimental Parallel Micro Processing System (PMPS) as part of the Los Alamos study on parallel processing [1].

A strongly connected shared memory (16 Mbyte) machine is created by an orthogonal array of 20 processor components and 32 memory elements in the revolutionary computer architecture known as PMPS. To allow direct comparisons of algorithm movie scenes for a range of parallel architectures, the system will contain reconfigurable processor-to-memory and processor-to-processor interconnections. Architectures with numerous processors sharing common memory and networks of processors using only local memory, such as rings, trees, and stars, are made possible by the reconfigurable design. The hardware design replaces the conventional medium-scale integration (MSI) or small-scale integration (SSI) circuitry with programmable array logic, programmable logic sequencers, and programmable logic arrays, as well as very-large-scale integration (VLSI) components like 16-bit microprocessors, floating-point co-processors, and random-access dynamic memories. By using these arrays, the system's various kinds of logic

devices are drastically reduced, which in turn lowers the engineering work required to buy the necessary components and to maintain them once the computer is operational. The following specifications are part of the design goals for the experimental PMPS: a system with a lot of processing power (more than eight processors), a lot of memory, components that are readily available from vendors, floating-point hardware, FORTRAN compilation, and minimal software development. The Computation Division of the Los Alamos National Laboratory's parallel processing research includes this experimental PMPS as one of its components.

Parallel Microprocessor System Architecture

The PMPS is an intimately connected shared memory multiple-instruction multiple-data machine that permits rearrangement between processor and memory nodes to facilitate for testing of common memory architectures as well as different microprocessor network configurations includes rings, trees, and stars. On the basis of the parallel processing goals, the appropriate memory and processing unit pairings may be chosen. The system is made up of a large number of memory and processing nodes that are directly coupled together via a number of processor-to-memory buses and multipored global memory nodes. A complete cross-bar switch between the processor and memory nodes is functionally accomplished by the multiple-bus multipored memory architecture. The concurrent operations of processor-to-processor communication and execution from either local or global memory is made possible by this multiple-bus architecture. Each microprocessor node has a memory mapping function that allows for reconfiguration between the microprocessor and global memory nodes. Restructuring will only be possible at startup; but, when hardware and software are used more often, dynamic reconfiguration may become possible. Figure 1 depicts the architecture of the system. Each processor element has a local memory that combines random access memory (RAM) and programmable read-only memory (PROM) (RAM). A control computer starts execution via the system control bus having loading the implementation code into the global memory. Processing components and global recollections may be arranged in systems with either a big shared memory [2], [3].

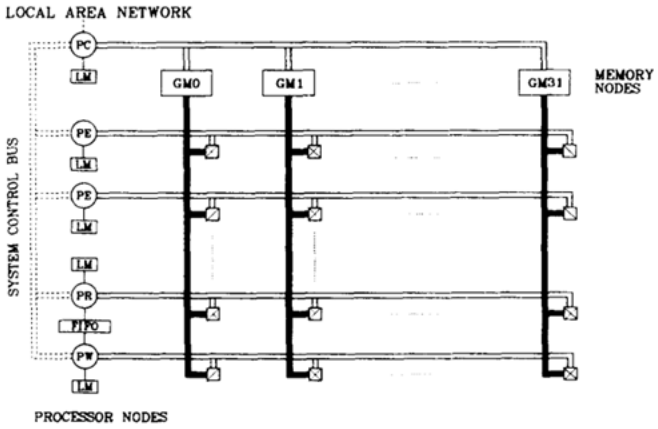


Figure 1: Represented that the PMPS architecture: PE processing element, GM global memory, LM Local Memory, CP Control Processor, PR Read Processor; PW Write Processor; FIFO First-In First-Out.

Systems with only local memory can be configured with the general processing elements, global memories and data transfer processors. A data transfer processor consists of a read processor, which uses one bus to read from memory and to store in a FIFO memory, and a write processor,

which uses one bus to read the FIFO memory and to write to memory. This architecture is enabled by the orthogonal packaging scheme shown in Figure 2. The processing element cards plug horizontally into the processor section of the back plane. The global memory element cards plug vertically into the left and right global memory sections low and high memory addresses of the back plane. The left and right sections provide contiguous logical addressing. However, the left and right sections are electrically isolated, limiting each back-plane section to 16 global memory cards and one bus termination card. Therefore, the back-plane sections provide 21 microcomputer busses: one control processor bus and 20 processing element busses. The control processor bus is a Multi-bus-oriented bus. Each processing element bus back-plane bus consists of 32 lines. 24 lines provide 24 address signals, 16 of which are multiplexed address-data signals. Seven lines provide bus control and memory error identification. The remaining line is a reserved line, possibly for a diagnostic bus. Figure 2 details the devices used in the back-plane bus design and address-data flow.

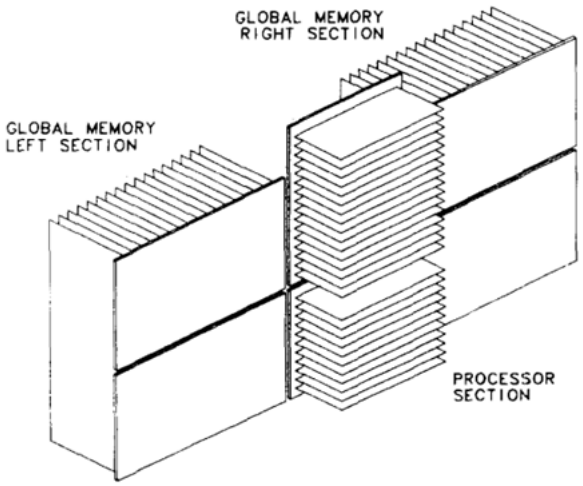


Figure 2: Represented that the Orthogonal Packaging Diagram.

Processing Element Implementation

Selection of the floating-point hardware for the processing element was required early in the design cycle. The design goal of using commercially available components led to the selection of the only available floating-point co-processors at that time: the Intel iAPX 86/20, which consists of the Intel 8087 and 8086 co-processors. The Intel 8087 is the high performance numeric data co-processor for the processing element or execution processor. This co-processor provides the floating-point hardware required to create a processing element with significant scientific computational capabilities. The Intel 8086 16-bit high performance metal-oxide-semiconductor (HMOS) microprocessor allows the processing element to address a megabyte of global memory directly. The memory mapping elements 74LS610 used in the design extend addressing to 16 Mbytes[4].

Three types of processor nodes are included within the system: the system control processor, the processing elements and the data transfer processors. The system control processor performs system initialization (downloading of global memory, configuration control etc.), initiation of parallel processing applications code, performance measurements and memory error processing. This processor also incorporates an interprocessor interruption facility. In addition, because the

multiprocessor is strictly an execution environment, the system control processor provides communication with an external local area network that includes development work stations. Each processing element includes the Intel 8086 and 8087 co-processors, 8 - 32 kbytes of local dedicated PROM and 4 -16 kbytes of RAM, real-time interrupt facility and memory mapping logic that allows 61 16-kbyte segments to be permanently and/or dynamically allocated within the system global memory. Each data transfer processor is a high speed controller specifically designed for implementing processor-to-processor communications by performing data movement between global memory segments. Two Intel 8089 HMOS input-output (I/O) processors are used to implement each data transfer processor.

Global Memory Element Implementation

The system global memory is made up of a number of memory nodes, each of which has a multiported memory controller and a Microbar DBR50-256 256–512 kbyte RAM array that may be accessed by the system control processor. The system control processor's port offers capability for downloading and reporting memory errors. A high speed memory access controller, memory arbitration logic that uses the last-granted lowest priority mechanism, and interface logic for 20 ports are all included in the multiported memory controller. Each memory node may be assigned as either private or public memory for each processor node thanks to memory mapping logic within each processor node. The 32 global memory cards have the same design. To permit memory access on the particular card, the left and/or right bus controllers check the top five address bits less the most important bit with the 4-bit back-plane geographic address. The memory mapping logic at each processor node, the multiported memory controller at each global memory node, and the multiple-bus interconnection back plane that enables the orthogonal arrangement of processor and global memory boards all work together to achieve the processor-to-memory interconnection. In order to provide full physical connections between the CPU and global memory nodes, the packaging approach makes advantage of short bus distances. When many processors are concurrently accessing the same global memory node, the processor-to-memory interconnection resolves access arbitration, offers completely reconfigurable processor-to-memory connections, and facilitates mutual exclusion to shared memory. An expansion of the lock mechanism found on the Intel 8086, 8087, and 8089 co-processors is used to control shared memory[5], [6].

Back Plane Bus Control

A memory cycle (non-local memory) using the system back plane is controlled by five hardware controllers. An Intel 8288 bus controller provides the microprocessor bus control. A left or a right back-plane bus controller located on the execution processor board provides the control signals for the processor interface to the back plane. A port interface controller one for each bus; 20 per global memory card provides the control signals for the memory port interface to the back plane and initiates a request to the last granted lowest priority bus arbitration controller. This controller issues a grant to only one of the 20 port interface controllers. The final controller, the high speed bus controller, multiplexes the address and data bus connection to the Microbar Multiuse memory board. The principal hardware components of these controllers are Monolithic Memories programmable array logic devices (PAL16L8) and programmable logic sequencers (PAL16R8).

Hardware and Software Parallel Microprocessor System Development

The equipment involved in the development of the PMPS is shown in Figure 3. The microcomputer development system (MDS) is an Intel Series 3 MDS-800 with in-circuit-emulation capability

ICE-86A for the iAPX 86/20 co-processors. The MDS system includes a Data I/O System 19 universal programmer for the local memory PROMs and for the programmable logic devices. Memory storage for the system is provided by a 40-Mbyte hard disk. System files and source file entry and/or back-up is provided by a single double-density floppy disk drive.

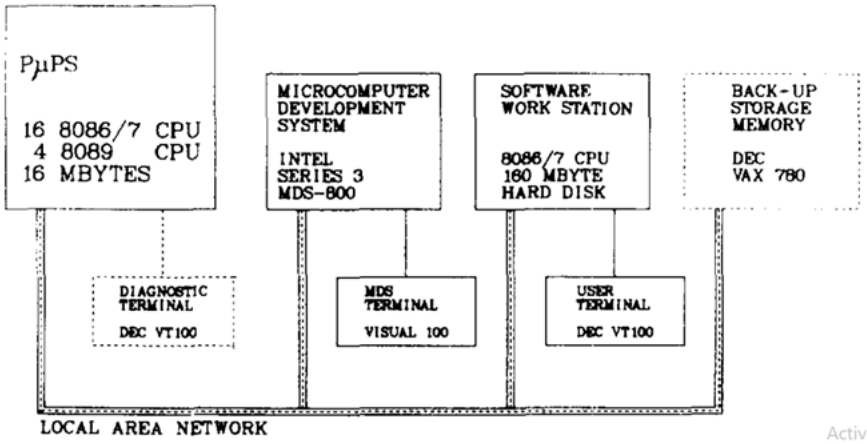


Figure 3: Represented that the PMPS Development Support Block Diagram.

A local area network links the parallel microprocessor, the MDS, backup storage (a VAX 780) and a software work station as display in Figure 3. The software work station will be used to convert existing FORTRAN production code to parallelized, compiled, linked and located code for the execution processor. The operating system for the software work station and possibly the control processor will be Intel's iRMX 86 operating system, which provides a multi-terminal and multiuser interface capability.

General System Construction

The PMPS enclosure is approximately 1 m deep, 2 m high and 3 m wide. The MDS is also shown in the front view before installation of the hard disk unit. The processor card cage section for housing 27 processor cards is shown in the center with circuit breakers to the left. Seven card slots are reserved for the control processor, leaving 20 card slots for the processing elements. The global memory card cage section to the right has global memory cards installed with one of its circuit breaker panels above the cage. Global memory cards have been removed from the left global memory card cage section to show the system back-plane cards. Two 200 A power supplies provide 5 V to one global memory section. One 200 A power supply provides 5 V to the processor section. Voltage (and return) is applied to each card at the front edge so that no supply voltage is in the back-plane wiring. Each card voltage is also switched by its own circuit breaker.

The PMIPS is an experimental computer architecture design consisting of an orthogonal array of 20 processing elements by 32 memory elements establishing a tightly coupled shared memory (16-Mbyte) machine. The principal objective is to develop an experimental P/xPS to serve as a research tool for evaluating the parallel processing of production codes on various multiprocessor architectures. The hardware design incorporates VLSI components, such as 16-bit microprocessors, floating-point co-processors and dynamic random access memories, and replaces conventional MSI or SSI circuitry with programmable array logic, programmable logic sequencers and programmable logic arrays. This experimental PMPS is one element of the parallel processing research within the Computing Division at the Los Alamos National Laboratory.

DISCUSSION

Several processors or CPUs are used in parallel processing to manage different aspects of a single task. By distributing a task's many components over multiple processors, systems may reduce the amount of time a software takes to run. Parallel processing is possible on systems with more than one CPU, which is common in contemporary computers and multi-core processors. Multi-core processors are IC chips containing two or more CPUs for faster processing, less power consumption, and better handling of many tasks. Most computers have between two and four cores, but some have as many as twelve. Parallel processing is routinely used to finish complicated procedures and calculations. The usage of registers differentiates between parallel and serial processes at the most basic level. Whereas registers with parallel loading handle each bit of the word concurrently, shift registers work serially, processing each bit one at a time. By using a multitude of functional units that carry out the same or different tasks concurrently, parallel processing may be managed at a greater degree of complexity[7], [8].

The late 1950s saw the emergence of interest in parallel computing, and the 1960s and 1970s saw the emergence of supercomputer advancements. These multiprocessors operated concurrently on a single data set while sharing memory. A new kind of parallel computing was developed in the middle of the 1980s when the Caltech Concurrent Computing project built a supercomputer for scientific purposes utilizing 64 Intel 8086/8087 processors. This system proved that one might reach great performance using microprocessors available off the shelf in the general market. These massively parallel processors (MPPs) came into being to take control of the high-end of computing after the ASCI Red supercomputer computer surpassed the barrier of one trillion floating point operations per second in 1997. Since then, the number and power of MPPs have increased. Late in the 1980s, clusters made their market debut and quickly took MPPs' position in many applications. Many commercial computers connected by a commercial network make up a cluster, which is a parallel computer. Clusters now rule the data centers that power the contemporary information age and are the workhorses of scientific computing. Parallel computing, which is based on multi-core computers, is gaining popularity[9], [10].

CONCLUSION

A job is typically divided amongst at least two microprocessors when processing is done in parallel. The principle is pretty straightforward: a computer scientist utilizes specialized software built for the job to break down a complicated issue into its component pieces. They then assign a unique processor to each component. Each processor completes its piece of the computation task. The data is reassembled by the programme to address the challenging original problem. A large project is divided into multiple smaller jobs that are more suited to the quantity, variety, and type of available processing units when processing is done in parallel. Each processor begins working on their portion of the job without consulting the others after it has been split. Instead, they communicate with one another and monitor the progress of their assignments using software. When all the programme sections have been processed, the outcome is a completely processed programme segment. This holds true regardless of whether there were an equal number of tasks, processors, and tasks, and whether they completed all at once or sequentially. Fine-grained and coarse-grained parallel processes are the two different categories. Fine-grained parallelism allows tasks to interact with one another dozens of times per second and provide results in real time or very near to real time. The lack of regular contact between coarse-grained parallel processes causes their decreased performance. To finish jobs more rapidly, a parallel processing system may process

data at the same time. For instance, while the CPU's arithmetic-logic unit is processing the present instruction, the system could get the following instruction from memory (ALU). The primary objective of parallel processing is to enhance throughput, or the amount of work that can be completed in a given amount of time, as well as the processing capacity of a computer. By doing related or unrelated tasks simultaneously, a system of parallel processing may be built using a number of functional units.

REFERENCES

[1] L. S. Maganti, P. Dhar, T. Sundararajan, and S. K. Das, "Mitigating non-uniform heat generation induced hot spot(s) in multicore processors using nanofluids in parallel microchannels," Int. J. Therm. Sci., 2018, doi: 10.1016/j.ijthermalsci.2017.11.015.

[2] T. Gokmen and Y. Vlasov, "Acceleration of deep neural network training with resistive cross-point devices: Design considerations," Front. Neurosci., 2016, doi: 10.3389/fnins.2016.00333.

[3] G. Velez, A. Cortés, M. Nieto, I. Vélez, and O. Otaegui, "A reconfigurable embedded vision system for advanced driver assistance," J. Real-Time Image Process., 2015, doi: 10.1007/s11554-014-0412-3.

[4] R. Lumbiarres-Lopez, M. Lopez-Garcia, and E. Canto-Navarro, "A new countermeasure against side-channel attacks based on hardware-software co-design," Microprocess. Microsyst., 2016, doi: 10.1016/j.micpro.2016.06.009.

[5] R. Kuhn and D. Padua, "Parallel Processing, 1980 to 2020," in Synthesis Lectures on Computer Architecture, 2020. doi: 10.2200/S01049ED1V01Y202009CAC054.

[6] H. Zhu, G. F. Harris, J. J. Wertsch, W. J. Tompkins, and J. G. Webster, "A Microprocessor-Based Data-Acquisition System for Measuring Plantar Pressures from Ambulatory Subjects," IEEE Trans. Biomed. Eng., 1991, doi: 10.1109/10.83573.

[7] C. M. A. da Luz, E. M. Vicente, and F. L. Tofoli, "Experimental evaluation of global maximum power point techniques under partial shading conditions," Sol. Energy, 2020, doi: 10.1016/j.solener.2019.11.099.

[8] M. C. Mitchell, V. Spikmans, A. Manz, and A. J. De Mello, "Microchip-based synthesis and total analysis systems (μSYNTAS): Chemical microprocessing for generation and analysis of compound libraries," J. Chem. Soc. Perkin 1, 2001, doi: 10.1039/b009037i.

[9] M. A. Kesterson, J. D. Luck, and M. P. Sama, "Development and preliminary evaluation of a spray deposition sensing system for improving pesticide application," Sensors (Switzerland), 2015, doi: 10.3390/s151229898.

[10] O. Lorenzo, T. Pena, J. Cabaleiro Domínguez, J. Pichel Campos, and F. Fernández Rivera, "Using an extended Roofline Model to understand data and thread affinities on NUMA systems," Ann. Multicore GPU Program. AMGP, 2014.

CHAPTER 12

AN ELABORATION OF THE 8086 MICROPROCESSOR OPERATION

Ashendra Kumar Saxena, Professor & Vice Principal
College of Computing Sciences and IT, Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India.
Email Id- ashendrasaxena@gmail.com

ABSTRACT:

It is a 16 bit µp. 8086 has a 20 bit address bus can access up to 220 memory locations (1 MB). It can support up to 64K I/O ports. It provides 14, 16-bit registers. It has multiplexedaddress and data bus AD0- AD15 and A16 – A19. It requires single phase clock with 33% duty cycle to provide internal timing. 8086 is designed to operate in two modes, Minimum and Maximum. It can prefaces upto 6 instruction bytes from memory and queues them in order to speed up instruction execution. It requires +5V power supply. A 40 pin dual in linepackage.

KEYWORDS:

Data Bus, Data Cycle, Microprocessor, Memory Location, Register Instruction.

INTRODUCTION

The minimum mode is selected by applying logic 1 to the MN/MX# input pin. This is a single microprocessor configuration. The maximum mode is selected by applying logic 0 to the MN / MX# input pin. This is a multi-microprocessors configuration. 8086 has two blocks BIU and EU. The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue. EU executes instructions from the instruction system byte queue. Both units operateasynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance. BIU contains Instruction queue, Segment registers, Instruction pointer, and Address adder. EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.

Bus Interface Unit

It provides a full 16 bit bidirectional data bus and 20 bit address bus. The bus interface unit is responsible for performing all external bus operations.

Specifically, it has the Following Functions

Instruction fetch, Instruction queuing, Operand fetch and storage, Address relocation and Bus control. The BIU uses a mechanism known as an instruction stream queue to implement a pipeline architecture. This queue permits prefect of up to six bytes of instruction code. Whenever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU isnot requesting it to read or write operands from memory, the BIU is free to look ahead inthe program by prefetching the next sequential instruction. These prefetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle. After a byte is loaded at the input end of the queue, it automatically shifts up through the FIFO to the empty location nearest the output [1].

The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory. These intervals of no bus activity, which may occur between bus cyclesare known as idle state. If the BIU is already in the process of fetching an instruction when the EU request it to read or write operands from memory or I/O, the BIU first completes theinstruction fetch bus cycle before initiating the operand read / write cycle. The BIU also contains a dedicated adder which is used to generate the 20 bit physical address that is output on the address bus. This address is formed by adding an appended 16 bit segment address and a 16 bit offset address. For example, the physical address of the next instructionto be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register. The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write.

Execution Unit

The Execution unit is responsible for decoding and executing all instructions. The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform theread or write bys cycles to memory or I/O and perform the operation specified by the instruction on the operands. During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction as display in Table 1. If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top ofthe queue. When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions. Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue [2].

Table 1: Represented that the Minimal Mode Signals.

Minimum Mode Signals ($\overline{MN}/MX$ = Vcc)		
Name	**Function**	**Type**
HOLD	Hold Request	Input
HLDA	Hold Acknowledge	Output
$\overline{WR}$	Write Control	Output 3- state
M/$\overline{IO}$	Memory or IO Control	Output 3-State
DT/$\overline{R}$	Data Transmit / Receiver	Output 3-State
$\overline{DEN}$	Date Enable	Output 3-State
ALE	Address Latch Enable	Output
$\overline{INTA}$	Interrupt Acknowledge	Output

Table 2: Represented that the Maximum Mode Signals.

Maximum mode signals (MN / $\overline{MX}$ = GND)		
Name	**Function**	**Type**
RQ / $\overline{GT1, 0}$	Request / Grant Bus Access Control	Bidirectional
$\overline{LOCK}$	Bus Priority Lock Control	Output, 3- State
$\overline{S_2} - \overline{S_0}$	Bus Cycle Status	Output, 3- State
QS1, QS0	Instruction Queue Status	Output

Internal Registers of 8086

The 8086 has four groups of the user accessible internal registers. They are the instruction pointer, four data registers, four pointer and index register, four segment registers. The 8086 has a total of fourteen 16-bit registers including a 16 bit register called the status register, with 9 of bits implemented for status and control flags. Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers:

i. **Code segment** (CS) is a 16-bit register containing address of 64 KB segment with processorinstructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

ii. **Stack segment** (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

iii. **Data segment** (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.

iv. **Extra segment** (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It is possible to change default segments used by general and index registers by prefixing instructions with a CS, SS, DS or ES prefix.

All general registers of the 8086 microprocessor can be used for arithmetic and logic operations. The general registers are:

Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations andstring manipulation.

Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing[1], [3].

Count register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low-order byteof the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation,

Data register consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low-order byte of the word, and DH contains the high- order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number [4].

The following registers are both general and index registers:

i. **Stack Pointer** (SP) is a 16-bit register pointing to program stack.

ii. **Base Pointer** (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

iii. **Source Index** (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.

iv. **Destination Index** (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Other registers:

i. **Instruction Pointer** (IP) is a 16-bit register.

ii. **Flags** is a 16-bit register containing 9 one-bit flags.

iii. **Overflow Flag** (OF): set if the result is too large positive number, or is too small negative number to fit into destination operand.

Direction Flag (DF): if set then string manipulation instructions will auto-decrement index registers. If cleared then the index registers will be auto-incremented.

i. **Interrupt-enable Flag** (IF): setting this bit enables mask able interrupts.

ii. **Single-step Flag** (TF): if set then single-step interrupt will occur after the next instruction.

iii. **Sign Flag** (SF): set if the most significant bit of the result is set.

iv. **Zero Flag** (ZF): set if the result is zero

v. **Auxiliary carry Flag** (AF): set if there was a carry from or borrow to bits 0-3 in the ALregister

vi. **Parity Flag** (PF): set if parity (the number of "1" bits) in the low-order byte of the result iseven.

vii. **Carry Flag** (CF): set if there was a carry from or borrow to the most significant bit duringlast result calculation.

Addressing Modes

i. **Implied**: The data value/data address is implicitly associated with the instruction.

ii. **Register:** references the data in a register or in a register pair.

iii. **Immediate:** the data is provided in the instruction.

iv. **Direct** - the instruction operand specifies the memory address where data is located.

v. **Register Indirect:** instruction specifies a register containing an address, where data islocated. This addressing mode works with SI, DI, BX and BP registers.

vi. **Based:** 8-bit or 16-bit instruction operand is added to the contents of a base register (BXor BP), the resulting value is a pointer to location where data resides.

vii. **Indexed:** 8-bit or 16-bit instruction operand is added to the contents of an index register(SI or DI), the resulting value is a pointer to location where data resides.

viii. **Based Indexed:** the contents of a base register (BX or BP) is added to the contents of anindex register (SI or DI), the resulting value is a pointer to location where data resides.

ix. **Based Indexed with displacement:** 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), the resulting value is a pointer to location where data resides [5].

Interrupts

The processor has the following interrupts:

i. **INTR** is a mask able hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction. When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 * <interrupt type>. Interrupt processing routine should return with the IRET instruction.

ii. **NMI** is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority than the mask able interrupt.

Software interrupts can be caused by:

INT instruction - breakpoint interrupt. This is a type 3 interrupt.

i. INT <interrupt number> instruction - any one interrupt from available 256 interrupts. INTO instruction interrupt on overflow.

ii. Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When theCPU processes this interrupt it clears TF flag before calling the interrupt processing routine.

Processor exceptions:

Divide Error (Type 0), UnusedOpcode (type 6) and Escape opcode (type 7).

i. Software interrupt processing is the same as for the hardware interrupts.

ii. The Figure 1 shows the 256 interrupt vectors arranged in the interrupt vector table in the memory.

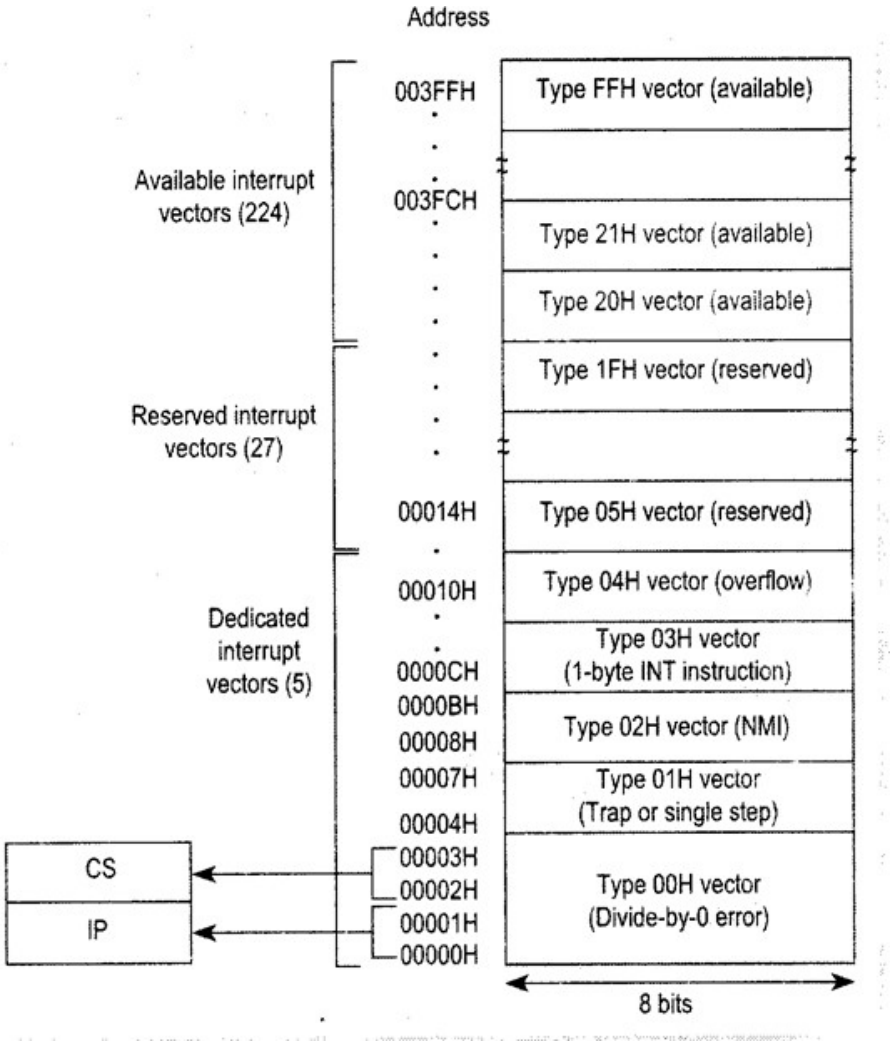


Figure 1: Represented that the Interrupt Vector Table in the 8086.

Minimum Mode Interface

When the Minimum mode operation is selected, the 8086 provides all control signalsneeded to implement the memory and I/O interface. The minimum mode signal can be divided into the following basic groups: address/data bus, status, control, interrupt and DMA.

Address/Data Bus

These lines serve two functions. As an address bus is 20 bits long and consists of signal lines A0 through A19. A19 represents the MSB and A0 LSB. A 20bit address gives the 8086 a 1Mbyte memory address space. Moreover it has an independent I/O address space which is 64K bytes in length.

According to the Figure 3, the 16 data bus lines D0 through D15 are actually multiplexed with address lines A0 through A15 respectively. By multiplexed we mean that the bus work as an address bus during first machine cycle and as a data bus during next machine cycles. D15 is the

MSB and D0 LSB. When acting as a data bus, they carry read/write data for memory, input/output data for I/O devices, and interrupt type codes from an interrupt controller.

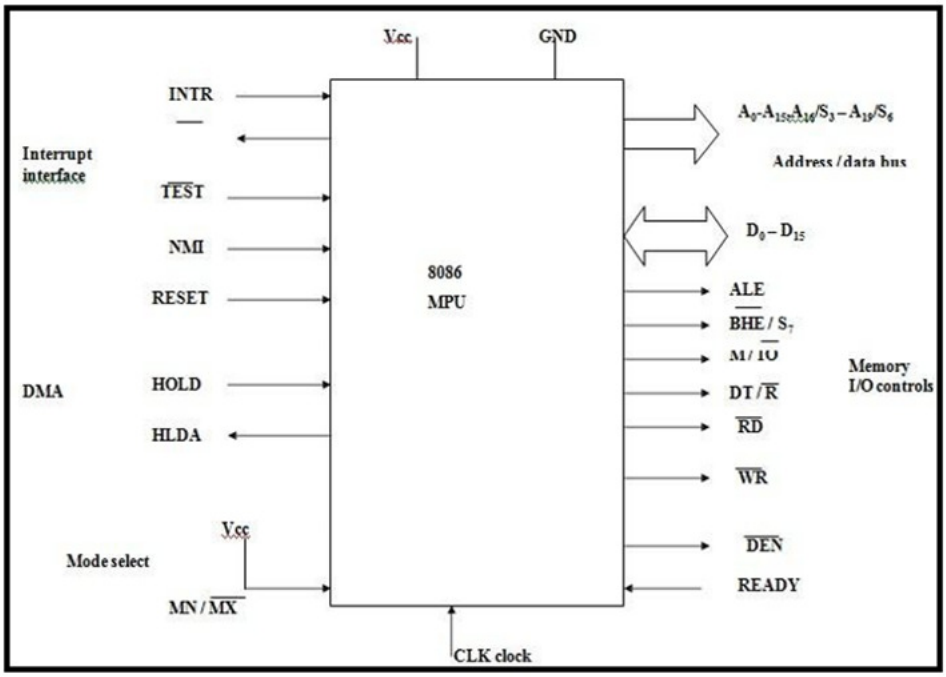


Figure 2: Represented that the Block Diagram of Minimum Mode 8086 MPU.

Status Signal

The four most significant address lines A19 through A16 are also multiplexed but in this case with status signals S6 through S3. These status bits are output on the bus at the same time that data are transferred over the other bus lines. Bit S4 and S3 together from a 2 bit binary code that identifies which of the 8086 internal segment registersare used to generate the physical address that was output on the address bus during the current bus cycle. Code S4S3 = 00 identifies a register known as extra segment register as the source of the segment address as mentioned in Table 3.

Table 3: Represented that the Status of All Signals.

S_4	S_3	Segment Register
0	0	Extra
0	1	Stack
1	0	Code / none
1	1	Data

Memory Segment Status Code

Status line S5 reflects the status of another internal characteristic of the 8086. It is the logic level of the internal enable flag. The last status bit S6 is always at the logic 0 level.

Control Signals

i. The control signals are provided to support the 8086 memory I/Ointerfaces. They control functions such as when the bus is to carry a valid address in which direction data are to be transferred over the bus, when valid write data are on the bus and when to put read data on the system bus.

ii. ALE is a pulse to logic 1 that signals external circuitry when a valid address word is on the bus. This address must be latched in external circuitry on the 1-to-0 edge of the pulse at ALE.

iii. Another control signal that is produced during the bus cycle is BHE bank high enable. Logic 0 on this used as a memory enable signal for the most significant byte half of the databus D8 through D1. These lines also serves a second function, which is as the S7 status line.

iv. Using the M/IO and DT/R lines, the 8086 signals which type of bus cycle is in progress and in which direction data are to be transferred over the bus.

v. The logic level of M/IO tells external circuitry whether a memory or I/O transfer is taking place over the bus. Logic 1 at this output signals a memory operation and logic 0 an

I/O Operation

i. The direction of data transfer over the bus is 74ignallin by the logic level output at DT/R. When this line is logic 1 during the data transfer part of a bus cycle, the bus is in thetransmit mode. Therefore, data are either written into memory or output to an I/O device[6].

ii. On the other hand, logic 0 at DT/R signals that the bus is in the receive mode. This corresponds to reading data from memory or input of data from an input port.

iii. The signal read RD and write WR indicates that a read bus cycle or a write bus cycle is in progress. The 8086 switches WR to logic 0 to signal external device that valid write or output data are on the bus. On the other hand, RD indicates that the 8086 is performing a read of data of the bus.During read operations, one other control signal is also supplied. This is DEN (data enable) and it signals external devices when they should put data on the bus.

iv. There is one other control signal that is involved with the memory and I/O interface. This isthe READY signal.

v. READY signal is used to insert wait states into the bus cycle such that it is extended by a number of clock periods. This signal is provided by an external clock generator device and can be supplied by the memory or I/O sub- system to signal the 8086 when they are ready topermit the data transfer to be completed.

Maximum Mode Interface

When the 8086 is set for the maximum-mode configuration, it provides signals for implementing a multiprocessor coprocessor system environment. By multiprocessorenvironment we mean that one microprocessor exists in the system and that each processor is executing its own program. Usually in this type of system environment, there are some system resources that are common to all processors. They are called as global resources. There are also other resources that are assigned to specific processors. These are known as local or private resources. Coprocessor also means that there is a second processor in the system. In this two processor does not access the bus at the same time. One passes the control of the system bus to the other and then may suspend its operation. In the maximum-mode 8086 system, facilities are provided for implementing allocation of global resources and passing bus control to other microprocessor or coprocessor [7].

8288 Bus Controller – Bus Command and Control Signals

8086 does not directly provide all the signals that are required to control the memory, I/O and interrupt interfaces. Specially the WR, M/IO, DT/R, DEN, ALE and INTA, signals are no longer produced by the 8086. Instead it outputs three status signals S0, S1, S2 prior to the initiation of each bus cycle. This 3-bit bus status code identifies which type of bus cycle is to follow. S2S1S0 are input to the external bus controller device, the bus controller generates the appropriately timed command and control signals.

Table 4: Represented that the Bus Status Code

Status Inputs			CPU Cycles	8288 Command
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$		
0	0	0	Interrupt Acknowledge	$\overline{INTA}$
0	0	1	Read I/O Port	$\overline{IORC}$
0	1	0	Write I/O Port	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	Halt	None
1	0	0	Instruction Fetch	$\overline{MRDC}$
1	0	1	Read Memory	$\overline{MRDC}$
1	1	0	Write Memory	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	Passive	None

The 8288 produces one or two of these eight command signals for each bus cycles. For instance, when the 8086 outputs the code S2S1S0 equals 001, it indicates that an I/O read cycle is to be

performed. In the code 111 is output by the 8086, it is 75ignalling that no bus activity is to take place. The control outputs produced by the 8288 are DEN, DT/R and ALE. These 3 signals provide the same functions as those described for the minimum system mode. This set ofbus commands and control signals is compatible with the Multibus and industry standardfor interfacing microprocessor systems.

8289 Bus Arbiter Bus Arbitration and Lock Signals

This device permits processors to reside on the system bus. It does this by implementing the Multibus arbitration protocol in an 8086-based system. Addition of the 8288 bus controller and 8289 bus arbiter frees a number of the 8086 pins for use to produce control signals that are needed to support multiple processors. Bus priority lock (LOCK) is one of these signals.It is input to the bus arbiter together with status signals S0 through S2[8].

Queue Status Signals

Two new signals that are produced by the 8086 in the maximum- mode system are queue status outputs QS0 and QS1. Together they form a 2-bit queuestatus code, QS1QS0. Following Table 5, shows the four different queue status.

Table 5: Represented that the Queue Status Codes.

QS_1	QS_0	Queue Status
0 (low)	0	No Operation. During the last clock cycle, nothing was taken from the queue.
0	1	First Byte. The byte taken from the queue was the first byte of the instruction.
1 (high)	0	Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction.
1	1	Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction.

DISCUSSION

The 8086 microprocessor has been characterized for total dose response. Commercial and military standard devices implemented in HMOS II technology as well as commercial devices implemented in HMOS technology were tested. Functional failures were observed between 6.8 krad (Si) and 25.9 krad (Si). The effect of fabrication technology and of variation in clock frequency is discussed. Functional failure modes as determined by test software are listed. There is a great deal of interest in NMOS devices due to their functional capability, which is not generally available in other technologies. However, the sensitivity of NMOS technology to total dose limits its application to systems with low to moderate gamma radiation requirements. Specific devices need to be characterized for radiation response. This paper presents the total dose test results for the Intel 8086 (iAPX 86/10) microprocessor. The 8086 is a 16-bit microprocessor which is capable of operating in both single and multiple processor configurations, and is implemented in n-channel, depletion load, silicon gate technology [9], [10].

CONCLUSION

Most microprocessor implementations of speech recognition hardware preclude the precision afforded by digital processing of speech by selecting analog preprocessing of the speech waveform for feature extraction. This is because digital-based speech recognition presents a heavy computational load. Feature extraction by digital means requires on the order of 200,000 multiply-add operations per second of speech, and usually requires complicated hardware to perform in real time. Hardware for analog feature extraction, on the other hand, requires much less speed to operate in real time. For example, one common analog feature extractor uses 16 parallel channels of bandpass filters followed by rectifiers and low pass filters with the channel output digitized and recorded every 10 msec. This requires only 1600 read operations per second of speech. The recognizer described here uses a microprocessor and peripheral speech processing circuitry consisting of large-scale integrated circuits to achieve near real time response in a compact, all digital module. Since then, the Acoustics Research Department at Bell Laboratories has carried out tests on a version of the recognizer which uses a Data General Eclipse minicomputer and CSP MAP-200 array processor. These tests used experienced and inexperienced talkers speaking over dialed-up telephone lines to examine the performance of most aspects of the recognition algorithm. Other tests have imbedded the recognizer in systems which used vocabulary partitioning, directory searches, and syntactic analysis to perform such voice-activated tasks as repertory dialing of telephone numbers, retrieving telephone directory information, and making airline reservations.

REFERENCES

[1] A. V. Pultyakov, R. V. Likhota, and V. A. Alekseenko, “Incident management in system of technical operation of microprocessor devices of railway automation and telemechanics,” Transp. Ural., 2020, doi: 10.20291/1815-9400-2020-1-43-47.

[2] S. Jamali and H. Borhani-Bahabadi, “Non-communication protection method for meshed and radial distribution networks with synchronous-based DG,” Int. J. Electr. Power Energy Syst., 2017, doi: 10.1016/j.ijepes.2017.06.019.

[3] D. A. Connors and W. M. W. Hwu, “Architecture,” in The Mechatronics Handbook, 2002. doi: 10.1177/1076217515613385.

[4] F. Romanyuk, V. Rumiantsev, A. Dziaruhina, V. Kachenya, and K. Kierczynski, “Increase of operation speed of digital measuring elements of microprocessor protection of electrical installations,” Prz. Elektrotechniczny, 2020, doi: 10.15199/48.2020.03.33.

[5] G. Gautam, G. Sumanth, K. C. Karthikeyan, S. Sundar, and D. Venkataraman, “Eye Movement Based Electronic Wheel Chair For Physically Challenged Persons,” Eye Mov. based Electron. Wheel chair Phys. challenged Pers., 2014.

[6] L. Shao et al., “Figure-of-merit for phase-change materials used in thermal management,” Int. J. Heat Mass Transf., 2016, doi: 10.1016/j.ijheatmasstransfer.2016.05.040.

[7] A. A. Gryzlov and M. A. Grigor’ev, “Improving the Reliability of Relay-Protection and Automatic Systems of Electric-Power Stations and Substations,” Russ. Electr. Eng., 2018, doi: 10.3103/S1068371218040077.

[8] M. Rhee and M. A. Burns, "Microfluidic pneumatic logic circuits and digital pneumatic microprocessors for integrated microfluidic systems," Lab Chip, 2009, doi: 10.1039/b904354c.

[9] R. Lavrijsen, D. C. M. C. Petit, A. Fernández-Pacheco, J. Lee, M. Mansell, and R. P. Cowburn, "Multi-bit operations in vertical spintronic shift registers," Nanotechnology, 2014, doi: 10.1088/0957-4484/25/10/105201.

[10] A. Pedroza De La Crúz, J. R. Reyes Barón, S. Ortega Cisneros, J. J. Raygoza Panduro, M. Á. Carrazco Díaz, and J. R. Loo Yau, "Characterization and synthesis of a 32-bit asynchronous microprocessor in synchronous reconfigurable devices," J. Appl. Res. Technol., 2015, doi: 10.1016/j.jart.2015.10.004.

CHAPTER 13

AN OVERVIEW OF THE MEMORY STRUCTURE OF MICROCONTROLLER

Rajendra P. Pandey, Assistant Professor
College of Computing Sciences and IT, Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India.
Email Id- panday_004@yahoo.co.uk

ABSTRACT:

Microcontrollers have a very limited internal memory, a total of 256 positions for the data, employment records and special records. In the 8052 microcontroller family has been expanded through the addition of 128 new positions between memory locations 80H to FFH that overlap with the current. To access a group or the other must be done by addressing, in this way, the main memory is accessed by direct addressing and high school through indirect or covert.

KEYWORDS:

ECC, Embedded System, Microcontroller, Microprocessor, Memory Management, Parallelized Execution.

INTRODUCTION

A Microcontroller is a programmable digital processor with necessary peripherals. Both microcontrollers and microprocessors are complex sequential digital circuitsmeant to carry out job according to the program or instructions. Sometimes analog input/output interface makes a part of a microcontroller circuit of mixed mode both analog and digital nature. A microprocessor requires an external memory for program or data storage. Instruction execution requires movement of data from the external memory tothe microprocessor or vice versa. Usually, microprocessors have good computing power and they have higher clock speed to facilitate faster computation. A microcontroller has required on-chip memory with associated peripherals. Amicrocontroller can be thought of a microprocessor with inbuilt peripherals. A microcontroller does not require much additional interfacing ICs foroperation and it functions as a standalone system. The operation of a microcontroller is multipurpose, just like a Swiss knife. Microcontrollers are also called embedded controllers. A microcontroller clockspeed is limited only to a few tens of MHz. Microcontrollers are numerous andmany of them are application specific [1].

Development of Microcontrollers:

Microcontrollers have gone through a silent evolution (invisible). The evolution can be rightly termed as silent as the impact or application of a microcontroller is not well known to a common user, although microcontroller technology has undergone significant change since early 1970's. Development of some popular microcontrollers is given in Table 1.

Table 1: Represented that the Developments of Microcontrollers.

Intel 4004	4 bit (2300 PMOS trans, 108 kHz)	1971
Intel 8048	8 bit	1976
Intel 8031	8 bit (ROM-less)	.
Intel 8051	8 bit (Mask ROM)	1980
Microchip PIC16C64	8 bit	1985
Motorola 68HC11	8 bit (on chip ADC)	.
Intel 80C196	16 bit	1982
Atmel AT89C51	8 bit (Flash memory)	.
Microchip PIC16F877	8 bit (Flash memory + ADC)	.

Development of Microprocessors (Visible)

Microprocessors have undergone significant evolution over the past four decades. This development is clearly perceptible to a common user, especially, in terms of phenomenal growth in capabilities of personal computers. Development of some of themicroprocessors can be given as Table 2.

Table 2: Represented that the Development of Microprocessors (Visible).

Intel 4004	4 bit (2300 PMOS transistors)	1971
Intel 8080 8085	8 bit (NMOS) 8 bit	1974
Intel 8088 8086	16 bit 16 bit	1978
Intel 80186 80286	16 bit 16 bit	1982
Intel 80386	32 bit (275000 transistors)	1985

Intel 80486 SX DX	32 bit 32 bit (built in floating point unit)	1989
)586 I MMX Celeron II IIIIV	64 bit	1993 1997 1999 2000
Z-80 (Zilog)	8 bit	1976
Motorola Power PC601 602 603	32-bit	1993 1995

We use more number of microcontrollers compared to microprocessors. Microprocessors are primarily used for computational purpose, whereas microcontrollers find wide application in devices needing real time processing or control. Application of microcontrollers are numerous. Starting from domestic applications such as in washing machines, TVs, air conditioners, microcontrollers are used in automobiles, process control industries, cell phones, electrical drives, and robotics and in space applications[2].

Microcontroller Chips

Broad Classification of different microcontroller chips could be as follows:

Embedded (Self -Contained) 8 - bit Microcontroller

16 to 32 Microcontrollers

Digital Signal Processors

Features of Modern Microcontrollers

Built-in Monitor Program

Built-in Program Memory

Interrupts

Analog I/O

Serial I/O

Facility to Interface External Memory

Timers

Internal Structure of a Microcontroller

At times, a microcontroller can have external memory also (if there is no internal memoryor extra memory interface is required). Early microcontrollers were manufactured using bipolar or NMOS technologies. Most modern microcontrollers are manufactured with CMOS technology, which leads to reduction in size and power loss. Current drawn by theIC is also reduced considerably from 10mA to a few micro-Amperes in sleep mode for a microcontroller running typically at a clock speed of 20MHz. The block diagram are mention in the Figure 1 [3].

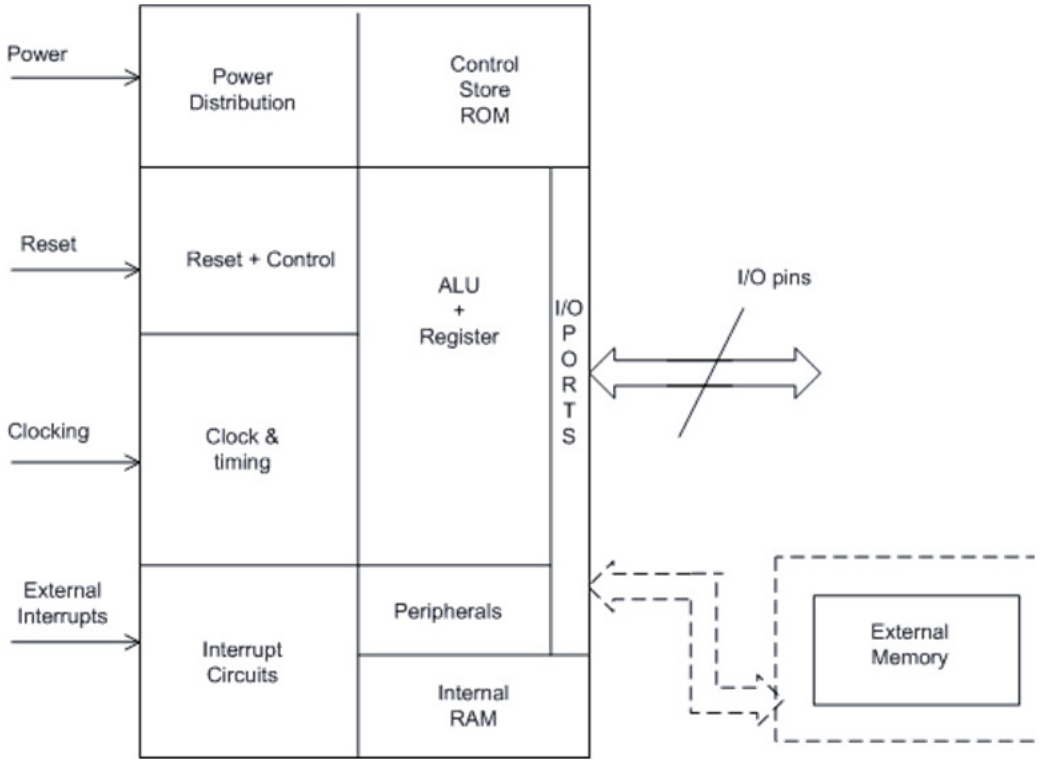


Figure 1: Represented that the Harvard Architecture (Separate Program and Data Memory interfaces)

Memory Organization

In the 8051, the memory is organized logically into program memory and data memory separately. The program memory is read-only type; the data memory is organized as read-write memory. Again, both program and data memories can be within the chip or outside.

128 bytes of Internal RAM Structure (lower address space)

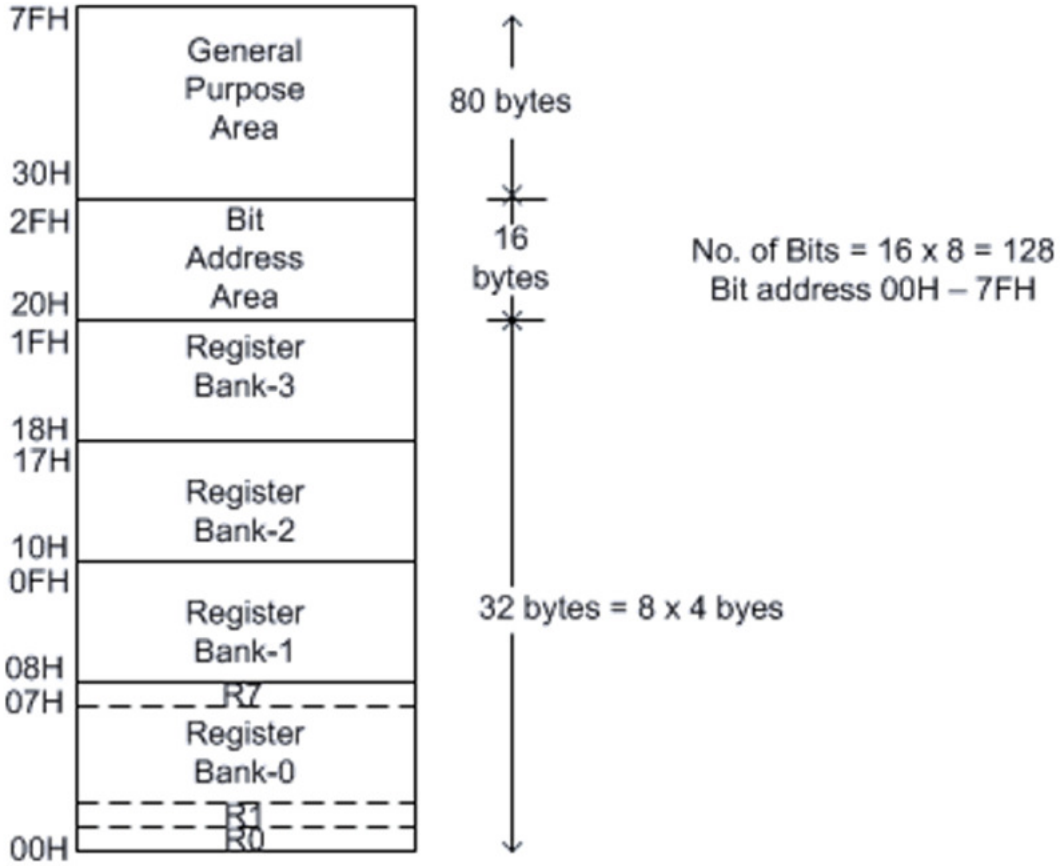


Figure 2: Represented that the Internal RAM Structure.

The lower 32 bytes are divided into 4 separate banks. Each register bank has 8 registers of one byte each. A register bank is selected depending upon two bank select bits in the PSW register. Next 16bytes are bit addressable. In total, 128bits (16X8) are available in bit addressable area. Each bit can be accessed and modified by suitable instructions. The bitaddresses are from 00H (LSB of the first byte in 20H) to 7FH (MSB of the last byte in 2FH). Remaining 80bytes of RAM are available for general purpose as display in Figure 2 [4].

Internal Data Memory and Special Function Register (SFR) Map:

The special function registers (SFRs) are mapped in the upper 128 bytes of internal data memory address. Hence there is an address overlap between the upper 128 bytes of data RAM and SFRs. Please note that the upper 128 bytes of data RAM are present only in the 8052 family. The lower128 bytes of RAM (00H - 7FH) can be accessed both by direct or indirect addressing while the upper 128 bytes of RAM (80H - FFH) are accessed by indirectaddressing. The SFRs (80H - FFH) are accessed by direct addressing only. This feature distinguishes the upper 128 bytes of memory from the SFRs, as shown in Figure 3.

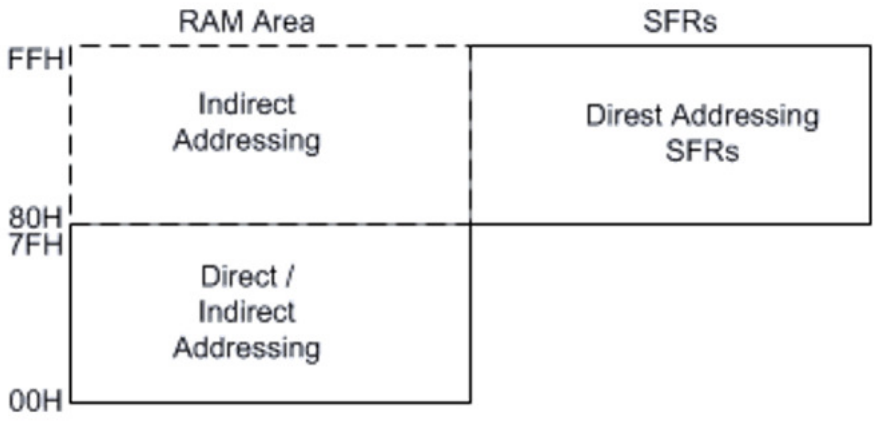


Figure 3: Represented that the Internal Data Memory Map.

Special Function Registers (SFRs) Map

It should be noted that all registers appearing in the first column are bit addressable. The bit address of a bit in the register is calculated as follows in Table 3[5].

i. Bit address of 'b' bit of register 'R' is Address of register 'R' + bwhere $0 \leq b \leq 7$.

F8H								
F0H	B*							
E8H								
E0H	ACC*							
D8H								
D0H	PSW*							
C8H	(T2CON)*		(RCAP2L)	(RCAP2H)	(TL2)	(TH2)		
C0H								
B8H	IP*							
B0H	P3*							
A8H	IE*							
A0H	P2*							
98H	SCON*	SBUF						
90H	P1*							
88H	TCON*	TMOD	TL0	TL1	TH0	TH1		
80H	P0*	SP	DPL	DPH				PCON

Processor Status Word (PSW)

PSW register stores the important status conditions of the microcontroller as display ion Figure 4. It also stores thebank select bits (RS1 & RS0) for register bank selection [6].

Address=D0H

CY	AC	F0	RS1	RS0	OV	-	P

Figure 4: Represented that the Processor Status Word.

8051 Addressing Modes

8051 has four addressing modes.

i. Immediate Addressing

Data is immediately available in the instruction.For example -

a. ADD A, #77; Adds 77 (decimal) to A and stores in A

b. ADD A, #4DH; Adds 4D (hexadecimal) to A and stores in A MOV DPTR, #1000H; Moves 1000 (hexadecimal) to data pointer

ii. Bank Addressing or Register Addressing

This way of addressing accesses the bytes in the current register bank. Data is available inthe register specified in the instruction. The register bank is decided by 2 bits of ProcessorStatus Word (PSW). For example:

ADD A, R0; Adds content of R0 to A and stores in A

iii. Direct Addressing

The address of the data is available in the instruction.For example: MOV A, 088H; Moves content of SFR TCON (address 088H) to A

iv. Register Indirect Addressing

The address of data is available in the R0 or R1 registers as specified in the instruction.For example: MOV A, @R0 moves content of address pointed by R0 to A.

v. External Data Addressing

Pointer used for external data addressing can be either R0/R1 (256 byte access) or DPTR(64kbyte access). For example: MOVX A, @R0; Moves content of 8-bit address pointed by R0 to A MOVX A, @DPTR; Moves content of 16-bit address pointed by DPTR to A.

vi. External Code Addressing

Sometimes we may want to store non-volatile data into the ROM e.g. look-up tables. Suchdata may require reading the code memory. This may be done as follows:

MOVC A, @A+DPTR; Moves content of address pointed by A+DPTR to AMOVC A, @A+PC; Moves content of address pointed by A+PC to A.

I/O Port Configuration

Each port of 8051 has bidirectional capability. Port 0 is called 'true bidirectional port' as itfloats (tristate) when configured as input. Port-1, 2, 3 are called 'quasi bidirectional port'.

a. Port-0 Pin Structure

b. Port -0 has 8 pins (P0.0-P0.7).

The structure of a Port-0 pin is shown in Figure 4.

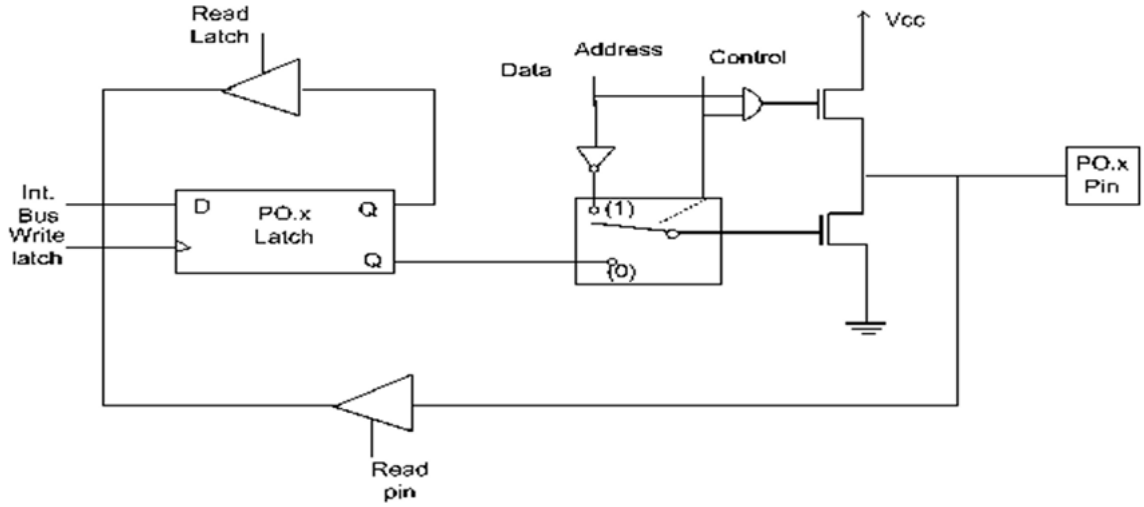


Figure 4: Represented that the Port-0 Structure.

Port-0 can be configured as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory. When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a normalbidirectional I/O port. Let us assume that control is '0'. When the port is used as an input port, '1' is written to thelatch. In this situation both the output MOSFETs are 'off'. Hence the output pin floats.

This high impedance pin can be pulled up or low by an external source. When the port is used as an output port, a '1' written to the latch again turns 'off' both the output MOSFETsand causes the output pin to float. An external pull-up is required to output a '1'. But when'0' is written to the latch, the pin is pulled down by the lower MOSFET. Hence the output becomes zero.

When the control is '1', address/data bus controls the output driver MOSFETs. If the address/data bus (internal) is '0', the upper MOSFET is 'off' and the lower MOSFET is 'on'. The output becomes '0'. If the address/data bus is '1', the upper transistor is 'on' andthe lower transistor is 'off'. Hence the output is '1'. Hence for normal address/data interfacing (for external memory access) no pull-up resistors are required.

a. Port-0 latch is written to with 1's when used for external memory access.

b. Port 1 Pin Structure

Port-1 has 8 pins (P1.1-P1.7) .The structure of a port-1 pin is shown in Figure 5.

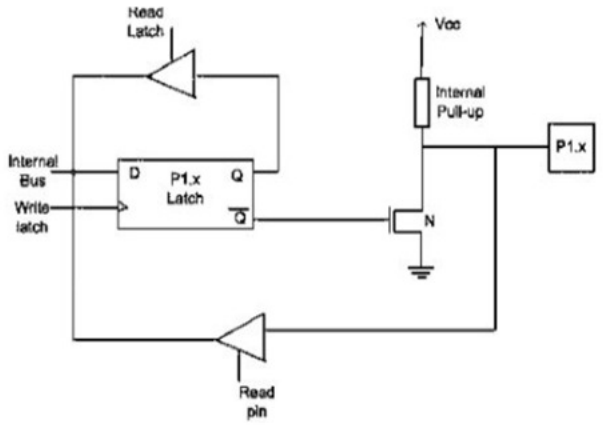


Figure 5: Represented that the Port-1 Structure.

Port-1 does not have any alternate function i.e. it is dedicated solely for I/O interfacing. When used as output port, the pin is pulled up or down through internal pull-up. To use port-1 as input port, '1' has to be written to the latch. In this input mode when '1' is writtento the pin by the external device then it read fine. But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading[7].

PORT 2 Pin Structure

Port-2 has 8-pins (P2.0-P2.7). The structure of a port-2 pin is shown in Figure 6.

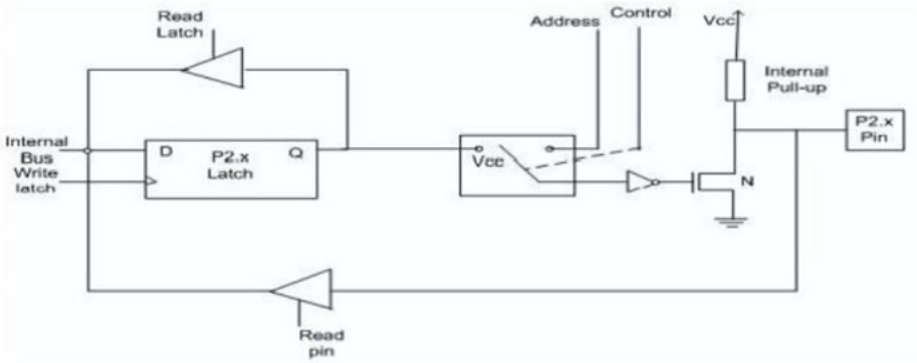


Figure 6: Represented that the PORT 2 Pin Structure.

Port-2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1. Port-2 latch remains stable when Port-2 pin are used for external memory access. Here again due to internal pull-up there is limited current drivingcapability.

PORT 3 Pin Structure

Port-3 has 8 pin (P3.0-P3.7). Port-3 pins have alternate functions. The structure of a port-3 pin is shown in Figure 7.

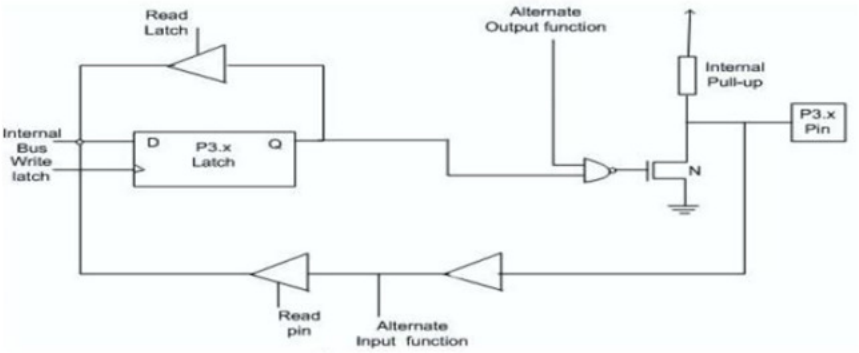


Figure 7: Represented that the PORT 3-Pin Structure.

Each pin of Port-3 can be individually programmed for I/O operation or for alternate function. The alternate function can be activated only if the corresponding latch has been written to '1'. To use the port as input port, '1' should be written to the latch. This port alsohas internal pull-up and limited current driving capability.

Alternate functions of Port-3 pins are:

a. Port 1, 2, 3 each can drive 4 LS TTL inputs.

b. Port-0 can drive 8 LS TTL inputs in address /data mode. For digital output port, itneeds external pull-up resistors.

c. Ports-1,2and 3 pins can also be driven by open-collector or open-drain outputs.

d. Each Port 3 bit can be configured either as a normal I/O or as a special functionbit.

Reading a Port (port-pins) versus Reading a Latch

There is a subtle difference between reading a latch and reading the output port pin. The status of the output port pin is sometimes dependent on the connected load. For instance if a port is configured as an output port and a '1' is written to the latch, the output pin should also show '1'. If the output is used to drive the base of a transistor, the transistor turns 'on'. If the port pin is read, the value will be '0' which is corresponding to the base-emitter voltage of the transistor[8]. Reading a latch: Usually the instructions that read the latch, read a value, possibly change it, and then rewrite it to the latch. These are called "read-modify-write" instructions.

Examples of a few instructions are- ORL P2, A; P2 <-- P2 or A

MOV P2.1, C; Move carry bit to PX.Y bit.

In this the latch value of P2 is read, is modified such that P2.1 is the same as Carry and is then written back to P2 latch.

Reading a Pin: Examples of a few instructions that read port pin, are- MOV A, P0 ; Move port-0 pin values to MOV A, P1; Move port-1 pin values to A Accessing external memory display in Figure 8.

P3.0	RxD
P3.1	TxD
P3.2	$\overline{INT0}$
P3.3	$\overline{INT1}$
P3.4	T0
P3.5	T1
P3.6	$\overline{WR}$
P3.7	$\overline{RD}$

Figure 8: Represented that the Functions 0f Port 3 pins.

DISCUSSION

The proposed model and the existing models have been examined in terms of speed, memory usage, reliability, and cell area. The proposed model achieved sufficient speed and reliability improvement. The proposed model achieved 7% speed improvement through parallel processing of ECC compared to the model using the existing ECC. In addition, the use of several parity bits and syndromes also increased reliability. This is related to improved performance and robust execution, which are important in embedded systems. In terms of circuit size and power consumption, it is inevitable to increase the size of the circuit because three decoders and encoders are used. In the decoder and encoder parts, there is a certain amount of increase in size, but this seems to result in a slight change in the overall area. Power consumption needs to be considered when using an embedded system. The change in power consumption is related to the change in speed. Since the proposed decoding model has improved speed compared to the existing decoding model, the overall execution time and energy consumption are reduced. Though overall memory usage increases, it may be necessary to increase the instruction memory. In addition, if the transmission error rate is high when longer instructions with ECC are loaded, additional errors may occur accordingly. To overcome these limitations, an additional ECC memory storing ECC parity bits is required. It needs to solve the memory usage problem by using more stable ECC memory and to guarantee the integrity between data transmission and reception[9], [10].

CONCLUSION

In this study, we proposed a decoder and encoder block parallelization structure to minimize the bottleneck that occurs when applying the Hamming code ECC in a TPU having both von Neumann and Harvard structures and to improve the reliability of the TPU. When the proposed structure was applied to the TPU, the use of additional memory improved speed and reliability. Experimental results confirmed that a negligible increase in size and power is achieved by using the proposed model. In contrast to the model without ECC, an additional ECC of 10 bits per instruction is required for the model, and an additional parity bit of 5 bits per instruction is required compared

to the existing ECC model. It is expected that the ECC encoder and decoder of the model proposed in this paper can contribute a lot to the above chips. In order for the concept of this paper to be applied to an actual chip, it is necessary to understand the bottleneck of the actual chip. Because the chip often has a lot of bottlenecks due to memory speed, in this case, this study can be very helpful. In addition, the parallel decoding method presented in this study is expected to be used in many processors with a speed limit in the instruction patch stage with a von Neumann structure.

REFERENCES

[1] O. M. Guillen, D. Schmidt, and G. Sigl, "Practical evaluation of code injection in encrypted firmware updates," in Proceedings of the 2016 Design, Automation and Test in Europe Conference and Exhibition, DATE 2016, 2016. doi: 10.3850/9783981537079_0089.

[2] M. Muttillo et al., "Structural health monitoring: An iot sensor system for structural damage indicator evaluation," Sensors (Switzerland), 2020, doi: 10.3390/s20174908.

[3] M. S. Kalairaj, T. Z. Feng, and H. Ren, "Soft-bodied flexible bending mechanism with silent shape memory alloys aiming for robotic endoscopy," in Flexible Robotics in Medicine: A Design Journey of Motion Generation Mechanisms and Biorobotic System Development, 2020. doi: 10.1016/B978-0-12-817595-8.00010-9.

[4] M. Muttillo, L. Di Battista, T. De Rubeis, and I. Nardi, "Structural health continuous monitoring of buildings-A modal parameters identification system," in 2019 4th International Conference on Smart and Sustainable Technologies, SpliTech 2019, 2019. doi: 10.23919/SpliTech.2019.8783051.

[5] D. Chen et al., "Single-event effect performance of a commercial embedded reram," IEEE Trans. Nucl. Sci., 2014, doi: 10.1109/TNS.2014.2361488.

[6] F. Sánchez-Texis, M. N. Ibarra-Bonilla, and I. Reyes-Castillo, "Pico satélite educativo CanSat desarrollado en las plataformas ARM-CortexM4 y FPGA," Rev. Apl. la Ing., 2019, doi: 10.35429/jea.2019.20.6.9.17.

[7] K. Rahul, H. Raheman, and V. Paradkar, "Design of a 4 DOF parallel robot arm and the firmware implementation on embedded system to transplant pot seedlings," Artif. Intell. Agric., 2020, doi: 10.1016/j.aiia.2020.09.003.

[8] Z. Hajduk, "An FPGA embedded microcontroller," Microprocess. Microsyst., 2014, doi: 10.1016/j.micpro.2013.10.004.

[9] S. Lee and S. Nirjon, "Fast and scalable in-memory deep multitask learning via neural weight virtualization," in MobiSys 2020 - Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services, 2020. doi: 10.1145/3386901.3388947.

[10] A. Villar-Martinez, L. Rodriguez-Gil, I. Angulo, P. Orduna, J. Garcia-Zubia, and D. Lopez-De-Ipina, "Improving the Scalability and Replicability of Embedded Systems Remote Laboratories through a Cost-Effective Architecture," IEEE Access, 2019, doi: 10.1109/ACCESS.2019.2952321.

CHAPTER 14

AN OVERVIEW OF THE CONNECTION BETWEEN MICROPROCESSOR ON INTERNET OF THINGS

Vineet Saxena, Assistant Professor
College of Computing Sciences and IT, Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India
Email Id- tmmit_cool@yahoo.co.in

ABSTRACT:

Gateways are generally used in Internet of Things (IoT) systems to enable endpoint device communication. Sadly, the gateways are unable to manage data logic and instead focus primarily on networking connections. Microcontrollers' built-in intelligence may be utilized as an illustration of what is available to make connections between endpoint devices easier. The applications highlight the microcontroller's critical role in Internet of Things (IoT) systems for providing users with intelligent services. A cloud-based simulation is also planned to evaluate the recommended strategy. The experiment's findings show how effective the recommended design.

KEYWORDS:

Cloud Model, Embedded Intelligence, Internet of Things, Microprocessor, Smart Microcontroller.

INTRODUCTION

The Internet of Things (IoT) idea has been extensively used in a variety of industries, including smart homes, health care, and items traceability, thanks to developments in microelectronics, communications, and information technology. The IoT system architecture, communication protocols, sensor networks, security and privacy protection, and applications are now the key areas of study in academia and industry [1].

The Internet of Things (IoT) is a general term for physical objects that may interact with one another via the use of communication protocols. It can be thought of as a network of inexpensive electronic devices that enable autonomous data collection and transmission. M2M communication, also known as "Machine to Machine" (M2M) communication, is a crucial component of IoT systems for managing uncertainty. Nevertheless, the absence of defined communication protocols makes it challenging for M2M devices to interact with one another in actual deployments. Typically, a gateway is used as an intermediary device between M2M devices. These gateways don't have sophisticated control capabilities, however. Since M2M devices have limited processing and memory capacity, these skills are crucial in IoT systems. Alternatively, the endpoint devices might be directly controlled by a microcontroller. In order to provide end customers intelligent services, it was also capable of communicating with the Internet. Figure 1 depicts a high-level perspective of an IoT system with the following essential elements:

Sensors: They are used to keep an eye on the surroundings in order to provide sensing data. This data might be handled locally or sent to a cloud storage facility for safekeeping.

Communication: Both internal and external connections between devices and the Internet are possible. This part might make use of communication technologies including Zigbee, Bluetooth, and WiFi.

Microcontrollers: To provide the intelligent services, they make all the gadgets function as a system. There are hardware and software components in a microcontroller. The microcontroller processes the sensor data and produces signals for the actuators. The goal condition of the whole system might be achieved in this fashion.

Actuators: They are used in environment manipulation. An example of a transducer device is an actuator. They take an electronic signal as an input and turn it into an actual mechanical action.

User Interfaces: The Internet of Things system must provide intelligent services to its users, and interactions between them must be suitable.

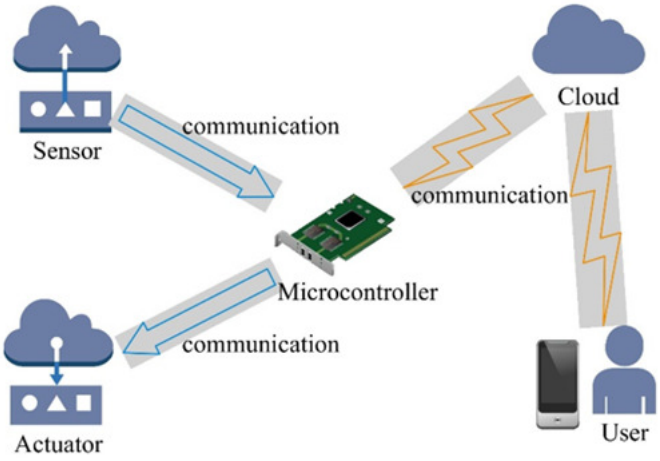


Figure 1: Represented that the Connection between Microcontroller and IoT.

Microelectromechanical systems (MEMS), which may build a system on a chip with improved performance and cheaper prices, have been identified in current research as a significant technology for the development of the IoT. As a result, it has been used extensively in sensors. Sensors have recently included embedded intelligence for more effective teamwork [2].

This is what drives our argument that, in order to increase the effectiveness of cooperation, the whole IoT system should integrate embedded intelligence. The microcontroller should be outfitted with sophisticated methods for effectively coordinating all of the components. The design and implementation of a smart microcontroller for Internet of Things systems are the main topics of this study. In our perspective, the microcontroller may be compared to a "brain" since it can create the final control orders, gather data from sensors, and evaluate it using computational intelligence methods like genetic algorithms and neural networks. The microcontroller design should also be expandable to accommodate various real-world applications. In this research, we provide a scalable and intelligent microcontroller architecture for Internet of Things devices. Intelligent controls are carried out by the microcontroller's logical circuit, which is connected to other microcontrollers through a system bus [3].

The following are the key contributions: The introduction of a smart microcontroller architecture comes first. To demonstrate the uses of the suggested architecture, two industrial IoT systems are

created as well. Implementations of both hardware and software are used in both systems. A single microcontroller is used in one system to offer a smart lock service. Another device uses many microcontrollers to demonstrate how they may work together to complete complicated tasks. The simulation experiment is then setup. Several microcontroller uncertainties are measured using the cloud model, which is often used in artificial intelligence. Moreover, the time cost measure is used to assess the system.

The rest of this essay is structured as follows: The materials and methods are covered in the second part, the results are covered in the third, and discussion of the suggested technique is covered in the fourth. The last portion of the essay wraps everything up.

The Smart Microcontroller Architecture

With the advancement of artificial intelligence technology, a new smart device is required to gather and process data in Internet of Things systems. This device's microcontroller could be thought of as the system's "brain," cooperating with sensors, actuators, and even people to provide services that are much more efficiently. Consequently, while building a smart microcontroller, the following three guidelines should be taken into account. The microcontroller needs first be modified. In certain applications, both the hardware and the software might be modified. Second, the microcontroller must possess intelligence. It needs to be able to do logical control and data mining. The microcontroller needs to be scalable, too. They might work together to complete a single difficult assignment [4].

Figure 2 depicts the suggested smart microcontroller design. A microcontroller typically consists of the following parts: a main board, a flash, a timer, a communication module, DC motors, and radio frequency (RF) modules. All components are connected to and managed by the main board. The flash component could be able to store the sensor and configuration data. The system's realtime clock might be provided via the timer component. The smart microcontroller may connect with distant systems, such cloud data centers, using the communication module. Two parts, DC motors and radio frequency modules, are created in order to receive data from the environment and communicate with it.

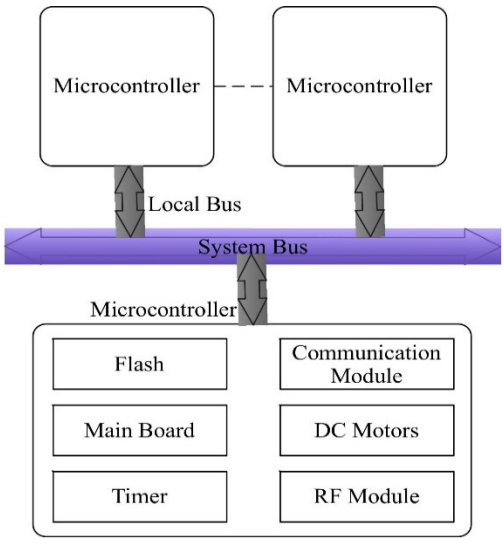


Figure 2: Represented that the Smart Microcontroller Architecture.

A microcontroller should also collaborate with other microcontrollers to complete the challenging tasks. In this study, it is referred to as the distributed architecture. The system bus connects the microcontrollers in a distributed design. A local bus connects each microcontroller to the system bus. A difficult job should first be broken down into smaller tasks in an IoT context before being handled. Thus, a single microcontroller may manage each sub-task. The system bus technique allowed every microcontroller to communicate with every other microcontroller. The first challenging job might be completed with the help of all microcontrollers working together cooperatively.

The Microcontroller Software Framework

A smart microcontroller's ultimate objective is to provide consumers intelligent services. Thus, it is important to understand what services a microcontroller offers in addition to the intricate design of its physical components. In reality, the manners in which a microcontroller interacts with other parts namely, the way it interacts with sensors, the way it interacts with actuators, and the way it interacts with users determine the ways in which it offers intelligent services to users. Figure 3 depicts the software architecture for the proposed microcontroller. The sensor layer, the communication layer, the control layer, and the application layer are the four functional levels that make up the microcontroller software.

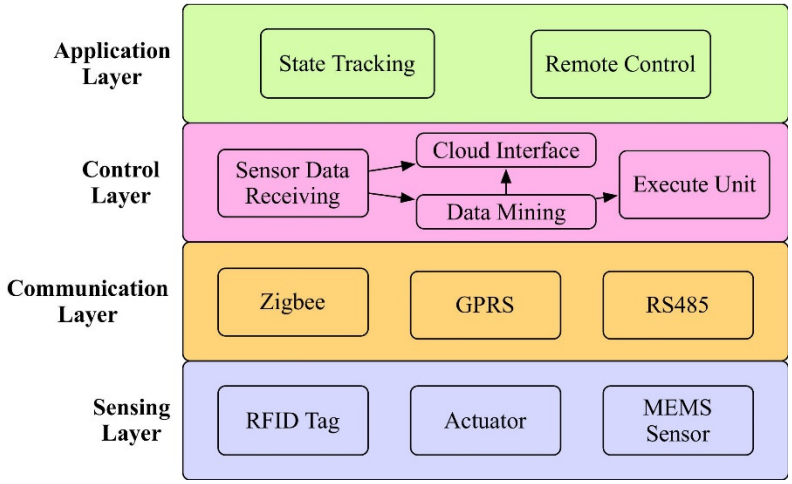


Figure 3: Illustrated that the Microcontroller Software Framework.

The microcontroller interacts with endpoint devices like sensors and actuators in the sensing layer. According to the various job categories, this layer involves the selection of sensor types, such as RFID tags and MEMS sensors. The microcontroller's input might be thought of as the sensor data. The actuator receives the microcontroller's output signal and carries out the required operations. The microcontroller interacts with other components, including as sensors, actuators, and cloud centres, via communication protocols, which are part of the communication layer. To interact with Zigbee nodes, the Zigbee protocol is often utilised. The distant cloud centres are contacted via the GPRS protocol. To locally connect with other microcontrollers, one uses the RS485 protocol. The main part of a microcontroller is the control layer. The microcontroller should store and process the received data in this layer. As a result, the control layer has the following functionalities: execution unit, data mining, receiving sensor data, and cloud centre interface. The capability for receiving sensor data replies by gathering and storing the sensor data. To analyse the data using machine learning techniques like classification and clustering, the data mining capability is

employed. The original data or the results of the analysis might be sent to the distant cloud centre through the cloud center interface [5].

The execute unit gathers the data mining findings and chooses the instructions for the output execution. There are two fundamental features of the application layer. State tracking is one, while remote control is another. The status of the sensors, actuators, and microcontroller itself might all be monitored via state tracking. A method of remotely controlling the sensors or actuators is offered by the remote control.

The Microcontroller Implementations in IoT Systems

Before implementing the suggested design, it is important to thoroughly examine the client needs. Initially, the hardware interfaces need to be created so that the microcontroller may access the sensor data. For instance, an RFID reader should be created and attached to the microcontroller if radio frequency identification (RFID) is the preferred mode of communication for the sensors. Second, a microcontroller should be used to accomplish the logical operations. The processing of the sensor data and generation of the control command would be possible in this fashion. Next, it is important to determine how the microcontroller communicates. The following two elements are part of this. The first is communication between microcontrollers, and the second is communication between the microcontroller and management software. The employment of communication protocols like RS485 is conceivable.

A RFID-Based Microcomputer Lock System Using One Microcontroller

In China's Jiangsu province, the first implementation was successfully used in an electrical firm. In actuality, due to the unique properties of electricity, equipment maintenance is challenging. When maintenance personnel run the machinery incorrectly, accidents usually happen. In the meanwhile, as the power grid has developed, the number of electrical equipment has grown quickly, which makes operation more challenging. In general, a duplicate naming scheme might be employed to aid maintenance personnel in distinguishing between various pieces of equipment. Unfortunately, when doing their everyday maintenance tasks, they often enter the wrong equipment room or use the incorrect electrical equipment. A lock mechanism is often used to effectively prevent the occurrence of these incidents. One such method to successfully prevent mishaps involving incorrect operations is the five anti microcomputer lock.

Nevertheless, the address space for encoding identifying numbers on the five anti-microcomputer lock is just 10 bits. That is, it only has 1024 permutations, which has the following two drawbacks: first, the identifying numbers provided to the locks would be duplicated if the number of pieces of equipment exceeded 1024. The danger of incorrect operations would result from this. Second, only one substation was permitted to utilize the identifying number. Also, there's a potential that it will be repeated at other substations. The digital keys need to be frequently setup on site to ensure the uniqueness of the lock identification number if the maintenance personnel attempt to service the substations in a consecutive manner. The time and effort of maintenance personnel will be wasted in this. The development of microchips for wireless data transfer is made possible by RFID technology, a significant advancement in the embedded communication paradigm. They serve as an electronic barcode to aid in the automated identification of anything they are connected. In order to successfully address the aforementioned issue, RFID technology with the suggested microcontroller architecture is used. This makes it much easier to run and maintain the electrical equipment.

The original five anti-microcomputer lock is enhanced in the following two phases to facilitate the product's deployment: An RFID tag with a special identification number is first affixed to the lock, and then the digital key with the suggested microcontroller design is put into practise. The digital key incorporates an RFID reader to decode the RFID tag's identifying number. Consequently, in addition to its original mechanism, the lock might be controlled by an RFID tag. To decide whether or not to open the lock, logical principles are used. This approach might guarantee the uniqueness of the equipment identifying number. In most cases, a 64-bit or 96-bit identifying number suffices. Also, it was possible to conduct remote authentication of the digital key to any machine. It is also possible to record the lock operator and the moment the lock was unlocked. Hence, the solution might significantly increase the efficiency of maintenance personnel and guarantee the equipment's regular operation. Figure 4 depicts the organization of the system. Two interfaces are left for the digital key, and an RFID tag is linked to the lock.

There are various prerequisites for creating a reliable system. The system must first operate correctly in a high electromagnetic environment. The RFID tag should be securely fastened to the surface of the lock in the interim. Second, an RFID tag situated more than two meters distant from the RFID tag reader might still connect with it, which is mention in Figure 4. Finally, a low power battery should be used to power the digital key. Moreover, it might save information like the unlock time. Fourth, the digital key might use USB or RS232 to connect to distant computers. The digital key should also automatically switch off if no operations are made for a certain period of time, such as 10 minutes.

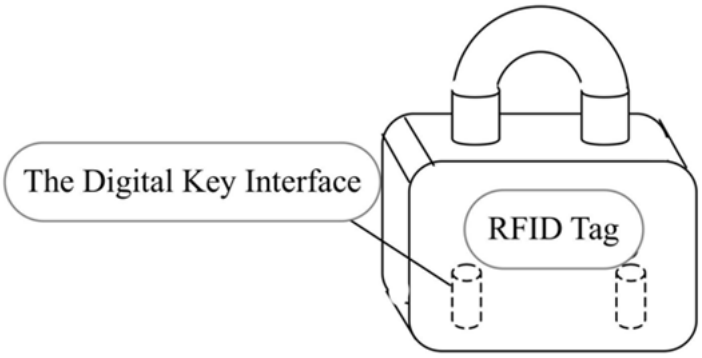


Figure 4: Represented that the RFID Lock System.

RFID Tag Component

Tag, reader, and the application software make up an RFID system. The tag may also be referred to as a transponder. The process goes as follows: first, the reader delivers a radio wave at a certain frequency to the transponder; next, the transponder sends out the data it has in memory; and last, the reader reads the data sequentially and sends them to the application program.

Two different kinds of RFID tags exist. The passive tag is the first, while the active tag is the second. The RFID tag for the passive tag picks up the radio frequency signal transmitted by the reader as it enters the magnetic field. With the energy produced by the induced current, the data contained in the tag chip is sent. It could broadcast a signal with a specific frequency for the active tag. The reader will transmit the data to the central information system for further processing after reading and decoding it.

In this system, a passive tag with a small size is used to save maintenance costs. The operating frequency of the RFID tag is also a crucial component of the whole hardware system that has been presented since it affects not only how the radio frequency identification system functions but also how difficult and expensive it will be to construct an RFID reader. An ultra-high frequency RFID tag with a frequency of 915 MHz is employed in the proposed system. The RFID tag reader is created based on this frequency.

RFID Tag Reader Component

There are two different types of RFID tag readers in this system. The first one is used to keep track of the RFID tag's lock-attached lock's unique identification number. It has the ability to both accurately read and write an RFID tag's encoded identification number. The digital key contains the other one as well. It may identify whether the target lock should be opened or not by reading the encoded identification number from an RFID tag and comparing it to the number stored in the digital lock flash. In the same equipment room, there are several nearby locks. In order to avoid misreads, the working distance should be appropriately calibrated.

The RMU900 module serves as the primary element of the RFID reader in our approach. The RMU900 has integrated radio frequency coupling, emission, receiver, and MCU components. Also, it provides a number of benefits. To easily control the module, an application programming interface (API) is first made available. Moreover, it operates at a low voltage, making it appropriate for portable devices. We created the following two features based on this module: the first is adjusting the module's operating frequency and reading range, and the second is reading data from an RFID tag and transmitting it to another circuit for processing and storage [6].

Central Control Module

The central control module is the microcontroller's main part. It can carry out logical commands and read data from flash. The Acorn RISC Machine (ARM7) processor, which has benefits of low power consumption and fast operating speed, is used by the central control module in the digital key.

Battery Component

A battery powers the digital key. Thus, a strong battery is required to guarantee that it operates normally. The benefits of lithium batteries are their high energy density and extended lifespan (more than 6 years). Moreover, a single battery's operational voltage is 3.7 V or 3.2 V, indicating that it has a strong power endurance. Moreover, it has a very low self-discharge rate. Because to these benefits, lithium batteries are among the top battery options for the suggested digital key.

Flash Component

Two different sorts of data are stored in the digital key's flash memory. The target lock's identifying number should be saved first. Second, additional crucial information like unlock worker and unlock time should be saved for further monitoring.

Our constructed digital key makes use of the SST39VF1601 NOR flash memory chip. Given that it has a 2M data storage capacity, it could satisfy the storage needs. Not all of the storage contents are included in this table. The address space is indicated by three main contents: working ID, unlock start time, and unlock end time.

Management System

Two software systems form part of the management system. The lock management system is one, while the remote authentication system is the other. All of the electric equipment's identifying numbers are kept up to date using the lock management system. In other words, it may provide electrical equipment a special identifying number. The management system might also record the unlock time and unlock operator.

The administrator may remotely authenticate the digital keys using the remote authentication mechanism. As a result, one or more locks might be opened in various places. The maintenance staff gathers all the tools required for unlocking, then submits a request to the administrator, who will review it. The identifying number will be given to the digital key if this is a legitimate request. The system will record each operation in the interim.

Figure 5 depicts the operation of the RFID-based microcomputer lock system. Initialization is the first action. The RFID tag that will be connected to the lock is given a unique identification number produced by the management system. "Authorization" is the next action. According to his working material, the maintenance worker makes an unlock request to the administrator. The digital key receives the audited identifying number from the administrator. "Unlock" is the third action. The technician opens the lock with the digital key. Connecting to the lock system, the digital lock will read the identification number. "Match" completes the process. The lock will unlock if the identifying number read and the number recorded in the digital key match. Otherwise, it is impossible to open the lock.

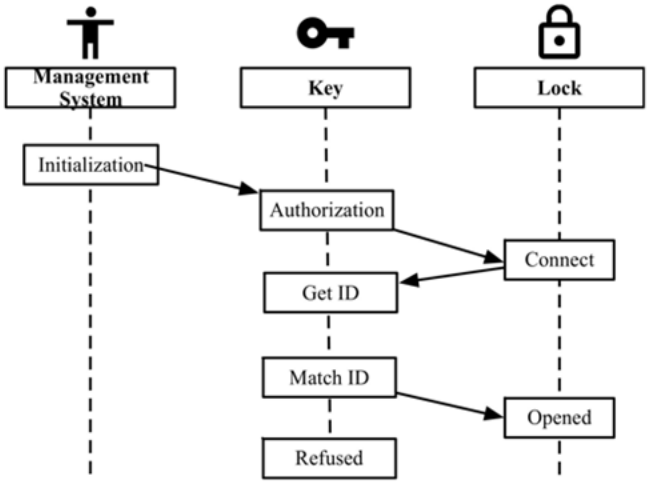


Figure 5: Represented that the Working Flow of RFID Lock System.

A Distributed Game System Using Multiple Microcontrollers

It is challenging to solve complicated issues with a single microcontroller. Thus, this objective might be accomplished using the distributed microcontroller architecture. This part will demonstrate a distributed microcontroller IoT gaming system. A computer game served as the inspiration for the actual room escape game. By finding the answers to the questions and the clues, players may exit the chamber.

This game has a wide variety of topics and situations. This necessitates the system's scalability. The initial system design shouldn't be changed while introducing a new scenario. The

microcontroller must to be reused as well. As a result, although the software needs be adjusted for each case, the hardware might be utilised in all of them. Lastly, for easy administration, all microcontrollers should be consistently managed. The distributed architecture is created with the aforementioned needs in mind, as shown in Figure 6. There are two themes in the game, and each game has a unique scenario. A separate room is designated for each scenario. The local controller is a microcontroller that is present in every situation. The system bus links every local controller to a central control unit.

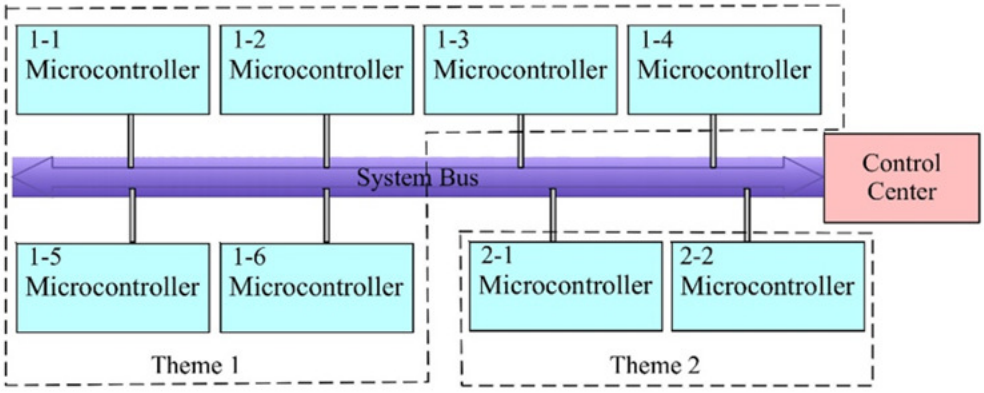


Figure 6. Represented that the Distributed Game System.

The local controller gathers information from a variety of sensors, including RFID tags, magnetic induction sensors, light sensors, steering sensors, and more. Moreover, the microcontroller's built-in logical rules are used to create the control signal. The electromagnetic lock, motor, alarm, and audio equipment will all be driven by the control signal to complete the control procedure. The local controller in our solution is powered by a 5 V battery. Moreover, it uses an RS485 system bus to connect with the control center. By adding a new microprocessor to the RS485 system bus, the system features might be easily expanded using this design [7].

The control center, which uses the same platform as the local controller, is likewise a microcontroller. As all local controllers are connected through the control center, each scenario's microcontroller may be controlled remotely. In the gaming rooms, for instance, every local controller might be reset by the control center. Also, the administration software in the control center enables the administrator to remotely operate the gaming room's equipment.

Though our constructed distributed gaming system uses several local controllers, just one local controller is chosen to show how it works. In Figure 7, the local controller system is shown. The left four parts of the diagram are sensors. The touching key is used to record players' touching behaviors. RFID tags connected to players' bodies provide signals to RFID readers, which are used to read such signals. The microcontroller performs the logical calculation and produces the control signals as a result. There are three control signals in this example. One is used to activate an electromagnetic lock, one to manage LED lighting, and one to manage the audio equipment.

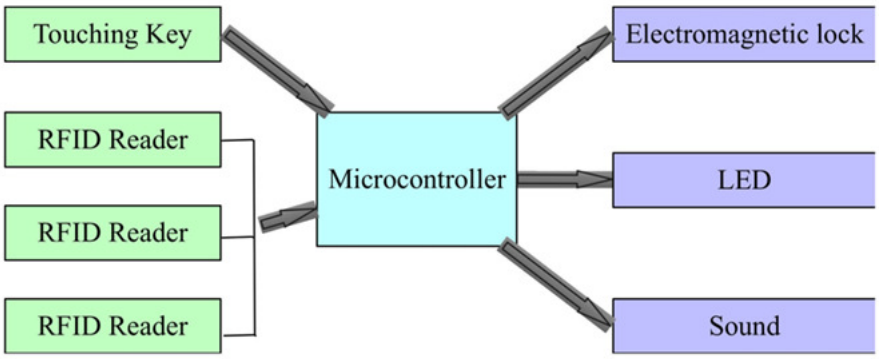


Figure 7. Represented that the Local Controller System.

The microcontroller's settings are reset during the startup phase. The microcontroller starts to check the key that is in contact with it for status. Data from one RFID reader is first read if the touching key is activated. If not, it will wait till players press the touching key. If the read data and the setting value are equal, the players have successfully completed this task. The program is over after all of the tasks have been completed.

DISCUSSION

In homes, businesses, hospitals, industries, and dozens of other locations connected to the Internet of Things, new gadgets are appearing every day (IoT). It is obvious that they must have internet access and that a significant quantity of unprocessed data must be gathered, kept, and processed on the cloud. Several of the microcontrollers utilized here were either modified or repurposed rather than being designed specifically for IoT applications. This unavoidably results in the absence of necessary qualities needed for IoT devices. An IoT device's microcontroller, which determines the features and specifications of the final IoT product, is its brain. Older microcontroller families are only partly appropriate for IoT applications due to the range of requirements. They are made for gadgets that have either been high performance or low power optimized, but not both. Older microcontrollers have less security features as well. Thus, designers of IoT devices must plan distinct security components like authentication ICs and cryptography processors in addition to the microcontroller. The interfaces also often need additional discrete components. As a result, designers must accept additional performance and energy consumption trade-offs in addition to the drawbacks of a high number of components and a big printed circuit board. Instead, utilize the particularly created IoT microcontrollers to get the greatest qualities for the corresponding design [8]–[10].

CONCLUSION

IoT innovations have altered how we live and users will get an increasing number of intelligent services from IoT devices. In this work, we argue that the Internet of Things (IoT) should be seen as a system in which both sensor and microcontroller technologies play crucial roles. Moreover, as artificial intelligence technologies become more prevalent, all components have the potential to be intelligently built to collaborate much more effectively so that consumers may get superior services. This article explored smart microcontroller technology for Internet of Things systems against this context. The two elements listed below are the key contributions. A microcontroller must be clever first. As the "brain" of the IoT system, it ought to function. Scalability in microcontroller architecture is the second need. For the complicated duties, the initial system should be readily expanded with a new microcontroller. To demonstrate the suggested design, two

commercial implementations are shown. Both versions have been used regularly in businesses with greater performance. Also, the cloud model, which is highly well-liked in the field of artificial intelligence with uncertainty, is used to simulate the second implementation. The simulation results show how well the suggested design works.

REFERENCES

[1] T. Adegbija, A. Rogacs, C. Patel, and A. Gordon-Ross, "Microprocessor optimizations for the internet of things: A survey," IEEE Trans. Comput. Des. Integr. Circuits Syst., 2018, doi: 10.1109/TCAD.2017.2717782.

[2] B. L. Risteska Stojkoska and K. V. Trivodaliev, "A review of Internet of Things for smart home: Challenges and solutions," Journal of Cleaner Production. 2017. doi: 10.1016/j.jclepro.2016.10.006.

[3] M. Swan, "Sensor mania! the internet of things, wearable computing, objective metrics, and the quantified self 2.0," Journal of Sensor and Actuator Networks. 2012. doi: 10.3390/jsan1030217.

[4] T. Saarikko, U. H. Westergren, and T. Blomquist, "The Internet of Things: Are you ready for what's coming?," Bus. Horiz., 2017, doi: 10.1016/j.bushor.2017.05.010.

[5] R. Senthilkumar, P. Venkatakrishnan, and N. Balaji, "Intelligent based novel embedded system based IoT enabled air pollution monitoring system," Microprocess. Microsyst., 2020, doi: 10.1016/j.micpro.2020.103172.

[6] S. Willner-Giwerc, C. Rogers, and K. B. Wendell, "The SymbIOTics System: Designing an Internet of Things Platform for Elementary School Students," Int. J. Des. Learn., 2020, doi: 10.14434/ijdl.v11i2.26719.

[7] S. Wachter, D. K. Polyushkin, O. Bethge, and T. Mueller, "A microprocessor based on a two-dimensional semiconductor," Nat. Commun., 2017, doi: 10.1038/ncomms14948.

[8] S. Mansfield-Devine, "Securing the Internet of Things," Comput. Fraud Secur., 2016, doi: 10.1016/S1361-3723(16)30038-0.

[9] S. Kadry, Y. D. Zhang, and S. Li, "Advanced microprocessor optimization methods for the Internet of Things," Transactions on Emerging Telecommunications Technologies. 2020. doi: 10.1002/ett.4187.

[10] Y. A. Rivas-Sánchez, M. F. Moreno-Pérez, and J. Roldán-Cañas, "Environment control with low-cost microcontrollers and microprocessors: Application for green walls," Sustain., 2019, doi: 10.3390/su11030782.

CHAPTER 15

AN OVERVIEW OF THE MICROCONTROLLER USE IN ENVIRONMENT SCIENCE

Amit Kumar Bishnoi, Assistant Professor
College of Computing Sciences and IT, Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India
Email Id- amit.vishnoi08@gmail.com

ABSTRACT:

Automation has been a long-term trend in technology. The fundamental idea of technology is to simplify life by expressly giving a guy less to do. Automation, without a question, is the way of the future since it immediately affects the environment. In every way, it makes life simpler and easier. An LCD display, a temperature sensor, and Arduino are used to construct the gadget. The gadget has a temperature sensor and a liquid crystal display for information. In this project, the Arduino Uno microcontroller board was utilized instead of any of the numerous clones that were available.

KEYWORD:

Automation, Environmental Temperature, LCD Display, Microcontroller, Measurement, Microprocessor.

INTRODUCTION

The present discussion around climate change has promoted the usage of plants to help reduce some of the issues connected to a lack of green space in urban areas, such as temperature rise. Due to the absence of greenery, there is less evapotranspiration and regular water penetration into the soil, which causes a heat island effect in big cities. Storm water management has been improved by green walls thanks to evapotranspiration and water retention. There are, however, other potential benefits, such as improved air quality, psychological benefits, and aesthetics. Green walls, also known as green façades or vertical gardens, provide a means to expand the amount of greenery in urban environments. Although though green wall irrigation systems are a relatively new technology that is still in development, the systems that we use on a daily basis have grown more automated as a result of technological advancement. Several irrigation methods that have been tried out and used are available on the market. These systems often have financial disadvantages and also need the user to regulate the water pump on-site [1].

Soil water content sensors are the most popular piece of equipment for keeping an eye on water levels in the substrate. These sensors, which are placed in several places and at various depths, provide helpful information regarding the water content of the green wall. Applications aimed at the efficient use of water in green walls heavily rely on the monitoring of substrate water content together with information from other sensors. To provide meaningful estimates of the water availability in the substrate of the green wall, the number of sensors and their spacing are essential. High-quality industrial devices are now on the market, but their use in green-wall technology has

been constrained by their cost and complexity. Alternative inexpensive devices have progressively been created in response.

Green walls, air temperature, and humidity are the environmental characteristics that are monitored regularly in each field experiment, independent of its specific purpose in our instance, since they reflect circumstances impacting plant phenology and development. The development of low-cost sensors, which are increasingly employed to provide high geographic coverage, was prompted by the need to obtain spatially dispersed meteorological observations, particularly in challenging settings like green walls. The development of smart irrigation systems that can link to transmission devices through the Internet and provide high space-time data coverage of substrate water and other various sensors put in the green walls is now possible thanks to recent advancements in open-source hardware components.

Together with the appropriate technology for storing, retrieving, and remotely transferring enormous amounts of data, multiplier devices are becoming more necessary. The wireless transmission technologies examined include the Bluetooth and Zigbee communication standards as well as modem-based connections like GPRS and GSM, which are often used in conjunction with Internet applications. The quantity of sensors and the frequency of measurements are limited by the cost of data loggers and sensors. The development of measuring stations that can link to wireless transmission devices to obtain high space-time data coverage with many sensors is now possible thanks to recent advancements in the area of open hardware components. Similar to open-source software, the idea behind open-source hardware is to provide users unrestricted, open access to projects, code, and hardware designs so they can quickly share, modify, and upgrade their systems.

The electro valve relay is switched on when the soil moisture content is below or above the pre-set humidity threshold based on plant requirements and off in all other circumstances or when it is raining. A UNO microcontroller is used and is programmed to detect the soil moisture levels or the presence of rain. A microcontroller is an electrical device housed in a pod that has multiple linked parts, such as resistors and condensers, so that it may autonomously carry out a set of instructions that have been predetermined by a programmer. The processing and control circuit's main part is the microcontroller.

It is possible to utilize the Arduino microcontroller, a physical open-source IT platform for creating interactive autonomous things or items that link to computers. Both the hardware and the software for Arduino are open source. Also, a Raspberry Pi CPU was utilized to save the data from the three-month trial period for the examination of the growth of the green wall that followed. The project offers instructions for replicating and modernizing an affordable, adaptable smart watering system for green walls. It is based on the open-source philosophy, which enables knowledge to be exchanged regarding hardware and software design, promoting the adoption and continued development of already accessible technology [2].

All the electrical parts and programming software for the microcontroller are available for free with the Arduino UNO chip. The user-friendliness and straightforward programming are in accordance with the low cost of the electrical components, contributing to the idea of a "democratized" technical practice. The automation system will examine the sensors and transmit the data through the Internet, where it can be quickly and readily seen in real-time in a visual format using the ThingSpeak.com.

The platform includes inexpensive dielectric sensors that are presently on the market, which lowers the cost of data collecting even further. As a result, measurements from several locations are more reasonably priced, allowing for an increase in the time-space data density needed for applications trying to save water in green walls. In order to examine the data in real time, it was decided to employ Internet of Things (IoT) technology on the irrigation systems and green walls. The green wall will be linked to the Internet in order to make it available to anybody who wants to view it from anywhere. Precision watering using the Internet of Things may increase the effectiveness of green walls while saving money.

The main goal of this work is to build an automated irrigation system that can be remotely supervised and controlled from any smartphone, computer, or tablet with an internet connection using inexpensive, simple-to-install, and easy-to-get materials like Arduino UNO and Raspberry Pi systems, which are in charge of automating irrigation and gathering environmental data for green wall prototypes. Our idea was that low-cost sensor technologies could be integrated into green wall systems.

Green Wall Prototype Structure

The green wall prototype consists of eight open-backed boxes with an electro welded mesh made of aluminum sheets, each with a rear bracket for mounting to an existing structure. The boxes for the prototypes measure 1 m2 by 10 cm. South-facing green walls resembling commercially available goods have been totally filled with two distinct kinds of substrates, four with sphagnum moss and four with coconut coir and rice husk. A 5-cm space between the green wall prototype and the building wall was left open to enable air to flow freely behind it.

The drip irrigation technology is used in the green wall prototype. Each module has irrigation pipes at the top and center, spaced 20 cm apart and with 4l/h adjustable drippers. To guarantee that all of the plants get enough moisture, the pipes in each row will irrigate from the top down. As can be seen in Figure 1, the pipes are linked to an irrigation pump that is controlled by a relay, allowing it to provide water to each of the green wall prototypes placed at the highest point above ground level [3].

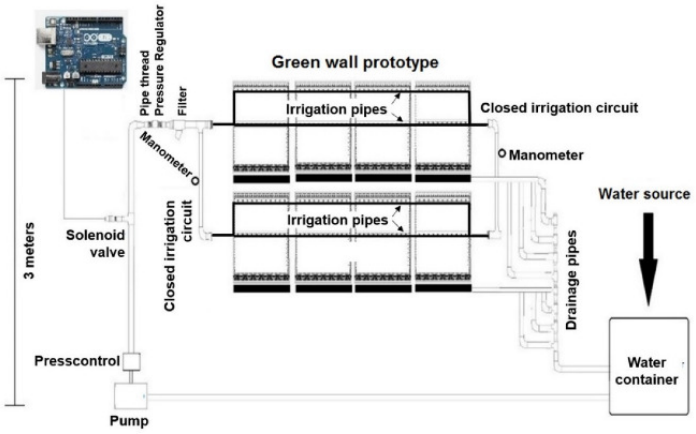


Figure 1: Represented that the Layout of the green walls.

Environmental Control System

As can be seen in Figure 2, the environmental control system of the variables that control irrigation are the following:

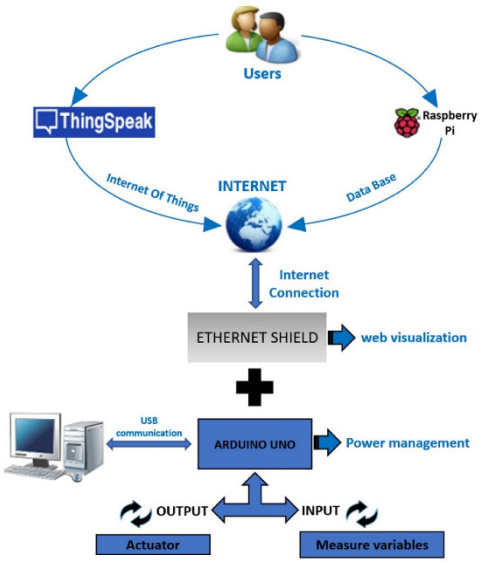


Figure 2: Represented that the Environmental Control System

The ATmega328-based Arduino UNO is a tiny microcontroller developed by Arduino that comes with a development environment for programming the board's software. Arduino may be used to control motors and other physical outputs by interacting with objects and/or a range of switches and sensors. The experiments performed using Arduino may be standalone or interact with computer-based software. The boards may be manually attached or bought already assembled.

The phrase "open-source Integrated Development Environment" (IDE) refers to a collection of software tools that may be used with Windows, Mac OS X, and Linux and that can be downloaded for free. These tools make it simple and easy for programmers to create their own applications. A certain set of instructions are included with the software, and they are all appropriately arranged and organized to produce a particular result.

The boards are programmable, allowing users to continuously add data to memory up until a new program the one we want the microcontroller to run is recorded. It is based on the avr-gcc process, which is also free source, and operates in a Java context. Since it can connect to a cabled TCP/IP network, the Ethernet Shield board is an extension Shield for the Arduino and is used to make online displays of the automating. It just requires connecting the module to the Arduino UNO Board. Shield's W5100 chip is responsible for doing this. The Ethernet programme library, which is by default a component of the recognized Arduino language, was used to configure the code for our application.

Due to the ease of its programming and popularity with API (Application Programming Interface) and apps for the receiving and storing of data from the sensors via HTTP through the Internet or via a LAN, Thing Talk was the website used for the Internet of Things. Applications were created using Thing Talk to capture data from the sensors during the trial. Up to eight fields, including relative humidity, temperature, soil moisture, water flow, and experiment location, may be entered into the Thing Talk channel 179863. For easier data analysis and visualization, the MATLAB App for numerical computing was also included.

A microprocessor manufactured in the UK for the Raspberry Pi Foundation, the Raspberry Pi model B comes with a Broadcom BCM2835 chip that has a GPU Video Core IV, 512 MB Memory, and an ARM CPU that can run at up to 1 GHz. Raspbian, a Debian-based Linux distribution system, was used for the project. The Raspberry Pi B+ was the model that was selected. It has HDMI and RCA video output connectors, a tiny audio jack, and a USB 2.0 connection that we add a little Wi-Fi antenna to in order to receive data provided from an Arduino board. The Raspberry Pi was equipped with a 16GB class 10 SD card to hold data.

Given that it shares a Wi-Fi network with Arduino and is not needed to be placed next to a green wall or be exposed to the weather, the Raspberry Pi is situated at university facilities. With an IP address issued by the university server, users may access the database from any terminal. LAMP, an abbreviation for "Linux, Apache, MySQL, and PHP," was used to develop the data gathering system of the green wall prototype. LAMP is a standard server setup for many online applications. Using Linux as its operating system, Apache as its web server, MySQL as its RDBMS, and PHP as its object-oriented scripting language, LAMP is an open-source web development platform.

The Raspberry Pi may be configured to run the web server programme Apache in order to serve web pages. With the addition of extra modules, Apache can serve dynamic web pages created using programming languages like the hypertext preprocessor as well as HTML files through HTTP. When a web page request is made to the server, PHP is the code that is run. When PHP has finished running, it decides what needs to be shown on the page and then sends it to the browser. PHP, as contrast to static HTML, may display various contents depending on the situation. The two PHP files config.php and iot.php were produced. The data necessary to establish a connection with the database is included in the config.php file. The data obtained from the sensors database is uploaded under the control of the iot.php file. A GET request is used to send data [4].

Based on Structured Query Language, MySQL is an open-source relational database management system (RDBMS) (SQL). Many operating systems, including Linux, UNIX, and Windows, support MySQL. MySQL is more often connected with web-based applications and online publication, despite the fact that it may be utilized in a broad variety of applications.

Environmental Sensors

In each green wall module, two FC-28 soil moisture sensors (Arduino Accessory) were used to detect the soil's moisture content. The FC-28 moisture sensor measures soil moisture and produces a value depending on that level, as shown in Figure 3. The LM393 analogue voltage comparator chip, which functions as a miniature voltmeter with inbuilt switches to offer both digital and analogue readings if a higher degree of accuracy is needed, is included inside the sensor, which runs between 3.3 and 5 volts. A pair of electrodes on the sensor are pressed into the substrate. The sensor's integrated LED displays the soil moisture level in real time in relation to a predetermined threshold. The basic idea is to make it easier to capture voltage values in this example, analogue values that are used to gauge the soil moisture content [5].

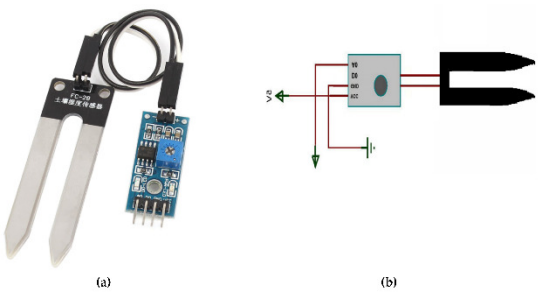


Figure 3: Shows the (a) FC-28 Soil Moisture Sensor; (b) Schematic Sensor Design.

In order to record moisture levels in both portions, each green wall module has two FC-28 soil moisture sensors, one in the top part and the other in the bottom half. With a total of eight FC-28 sensors in four distinct modules of the experimental green wall, the bottom half of the 1-m2 prototype typically has more water than the top part of the module. By monitoring the soil's dielectric constant, the FC-28 soil moisture sensors determine the volumetric substrate water content. While this fluctuates depending on the amount of water present, not all substrates have the same electrical characteristics because of differences in soil salinity, mineralogy, texture, and density. For the majority of mineral substrates, the factory calibration of the soil moisture sensors has an accuracy of 3–4%, while for growing substrates like sphagnum, coconut coir, etc., it has an accuracy of around 5%. Nonetheless, accuracy for all substrates with a particular soil-type calibration rises to 1-2% [6].

To get the most precise volumetric measurement for the water content, it is advised that FC-28 users perform a particular soil-type calibration. The sensors' particular soil-type calibration produces performance results that are equivalent to those of commercial sensors, but at a far more reasonable cost. Standard sensor calibration techniques were used in the laboratory to calibrate the soil moisture sensor. Four separate value ranges were established from the calibration process according to the soil moisture in the range of 0 to 1000 mV: from 0 to 370 (very wet substrate), from 370 to 600 (moist substrate), from 600 to 800 (dry substrate), and from 800 to 1000 (totally dry substrate).

While constructing an outdoor green wall, the air temperature and relative air moisture characteristics should be taken into account. A simple, inexpensive DHT22 digital sensor with digital output is used to assess relative humidity and temperature. It measures the air around it using a thermistor and a capacitive moisture sensor, providing a digital signal to the Arduino data port. The connections are straightforward: a first 3-5V pin, a second pin for data input, and the Ground pin, which is the one that is furthest to the right. These characteristics include monitoring temperature between -40 and 125 °C with an accuracy of 0.5 °C, measuring moisture between 0 and 100% with an accuracy of 2-5%, and sampling at a rate of two samples per second (0.5 Hz)[7].

DISCUSSION

Large computers, especially those used by financial institutions and the government, need a lot of energy to run and maintain the environment that the computer maker specifies. The power consumption of the environmental control devices may exceed that of the computer. Both startup and ongoing expenses are substantial. Some cooling plants use around a megawatt of electricity. Lately, highly costly systems have been created to recover computer waste heat for use elsewhere in buildings, such as for space heating or hot water supply. As the majority of computers will be

placed in existing buildings in the future, this strategy can only be readily used to newly constructed, tailored installations. When demand is lowest, in the summer, most heat may be produced by these systems [8], [9] they are, at best, a Band-Aid for an energy design that is conceptually flawed. This chapter promotes an integrated design strategy for the whole computer system, where all variables including energy consumption are taken into account from the start and their interactions are taken into account. On the basis of these ideas, a practical example that makes use of a fresh air ventilation system controlled by a microprocessor is given. Automatic equipment condition monitoring, enhanced security, and fire prevention are further benefits. The capital and operating expenses of the microprocessor-based environment controller are orders of magnitude lower than those of traditional solutions. The computers in question exhibit greater than average dependability, thus there is no data to imply that it is less effective [10].

CONCLUSION

The efficacy of watering green walls is currently being improved via the use of precise irrigation techniques. The real-time determination of the watering needs of the green wall and the definition of a location form the basis of precision irrigation scheduling. These data are used to activate the irrigation system when a certain substrate water content is reached. The number of sensors and the distance between them must be tuned for precise estimations of the amount of water that is accessible in the substrate at wall level. Important information about the rates and strategies employed by the plants to absorb water may be learned by observing changes in the water content of the substrate in green walls at different depths. As a result of the variety of the plants and the variability of the substrate, the soil water content changes regionally. As a result, strategies to increase water efficiency need information on water content that is accurate in both time and space.

REFERENCES

[1] J. M. Wang, M. T. Yang, and P. L. Chen, “Design and implementation of an intelligent windowsill system using smart handheld device and fuzzy microcontroller,” *Sensors (Switzerland)*, 2017, doi: 10.3390/s17040830.

[2] S. L. Brown, C. S. Goulsbra, M. G. Evans, T. Heath, and E. Shuttleworth, “Low cost CO2 sensing: A simple microcontroller approach with calibration and field use,” *HardwareX*, 2020, doi: 10.1016/j.ohx.2020.e00136.

[3] P. Teikari *et al.*, “An inexpensive Arduino-based LED stimulator system for vision research,” *J. Neurosci. Methods*, 2012, doi: 10.1016/j.jneumeth.2012.09.012.

[4] F. John Dian, R. Vahidnia, and A. Rahmati, “Wearables and the Internet of Things (IoT), Applications, Opportunities, and Challenges: A Survey,” *IEEE Access*. 2020. doi: 10.1109/ACCESS.2020.2986329.

[5] R. A. Firmansyah and D. Junianto, “Rancang Bangun Farming Box Dengan Pengaturan Suhu Menggunakan Fuzzy Logic Controller,” *ELKHA*, 2020, doi: 10.26418/elkha.v12i2.41196.

[6] A. Soia, O. Konnikova, and E. Konnikov, “The internet of things,” in *Proceedings of the 33rd International Business Information Management Association Conference, IBIMA 2019: Education Excellence and Innovation Management through Vision 2020*, 2019. doi: 10.4018/ijhiot.2018010101.

[7] B. B. L. Heyasa and V. R. K. R. Galarpe, "Preliminary Development and Testing of Microcontroller-MQ2 Gas Sensorfor University Air Quality Monitoring," *IOSR J. Electr. Electron. Eng.*, 2017, doi: 10.9790/1676-1203024753.

[8] A. J. Harvie, T. W. Phillips, and J. C. deMello, "A high-resolution polarimeter formed from inexpensive optical parts," *Sci. Rep.*, 2020, doi: 10.1038/s41598-020-61715-7.

[9] G. Lucenko, A. Kuzminskyi, and S. Burchak, "Organising the process of physics students' cognitive activity," *Univers. J. Educ. Res.*, 2020, doi: 10.13189/ujer.2020.080819.

[10] V. Nian, S. K. Chou, B. Su, and J. Bauly, "Life cycle analysis on carbon emissions from power generation - The nuclear energy example," *Appl. Energy*, 2014, doi: 10.1016/j.apenergy.2013.12.015.

CHAPTER 16
AN OVERVIEW OF WATER SENSING USE WITH MICROPROCESSOR

Shambhu Bhardwaj, Associate Professor
College of Computing Sciences and IT, Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India
Email Id- shambhu.bharadwaj@gmail.com

ABSTRACT:

A water sensor is a tool used in a variety of applications to gauge the water level. Water sensors may be found in a variety of forms, such as float sensors, pressure transducers, bubblers, and ultrasonic sensors. In this essay, we discuss the concept of managing and monitoring water levels in relation to water's electrical conductivity. More precisely, we look at wired and wireless microcontroller-based water level sensor and control systems. A water level control strategy would aid in lowering both household electricity use and water overflow. It is necessary to utilize such a priceless resource in a mobile application, at least in the consumers' eyes. Last but not least, we suggested a web- and cellular-based monitoring service protocol that would calculate and feel the worldwide water level.

KEYWORD:

Control Signal, Data Bus, Sensors, Microcontroller, Microprocessor.

INTRODUCTION

The sustainability of the world's water resources is now a major concern. This issue is subtly connected to inadequate water management practices, wasteful water consumption, and inadequate and disjointed water distribution. Agriculture, industry, and residential usage all often utilize water. A house or office's water management system may thus be limited by efficient usage and water monitoring. Many monitoring systems with water level sensors have gained acceptance during the last several decades. From a governmental and residential standpoint, measuring water level is a crucial responsibility. With the integration of multiple regulating actions, it would be able to monitor the actual execution of such projects. Implementing a water control system is thus potentially significant for domestic applications. The automatic level detection approach that is currently in use and that may be utilized to turn devices on and off is discussed. In addition, the most typical way to manage the level of a household appliance is to simply start the feed pump at a low level and let it run until the water level in the water tank rises.

The controlling system is not effectively supported by this. In addition, liquid level control systems are often used to monitor the levels of liquids in reservoirs, silos, dams, and other structures. These systems often provide visible multi-level and continuous level indicator. This management system may feature audiovisual alerts at desired levels and automated pump control depending on user needs. To verify that water sustainability is truly being attained, proper monitoring is required, with payment tied to automation and sensing. Automated water level detection and control are part of this programmed technique, which is based on microcontrollers. The structure of this essay is as follows. The second chapter focused on the fundamental principles of system design. We

covered the specific concept of the PIC16F84A in chapter three. The chapter four description covers the design and implementation portion. This Chapter explains the monitoring and control network we suggest. Sixth chapter discusses the conclusion and next projects. In our suggested way, the technology of water level monitoring and regulating system is condensed with a few fundamental components that are gently aggregated together. These are brief explanations of several parts:[1]

Water Level Indicator

We can utilize some LED lights that will function for water level indication in the water level indicator equipment. Water level sensors may be touched to show various water levels, and when the LED turns on, the sensor has detected water.

Water Level Sensor

We would like to present a few practical materials such as iron rod, nozzles, resistance, rubber, etc. to construct a specific water level sensor. There should be at least four nozzles linked to the +5v through a 1k resistance and a connecting rod composed of iron and steel that should be attached to ground. At the point where they link, we need to bond them together and place a rubber that will serve as an insulator for each nozzle. When the sensor comes into contact with water, water conductivity allows the nozzles and connecting rod to connect electrically.

Water Pump Controlling System

By connecting the water pump to a microcontroller's output pin using a motor driver circuit, we can control it. The water pump turns on or off depending on whether the microcontroller provides a positive signal (+5v) or a ground signal (0v) to the motor driver circuit. On the motor driver circuit, which is intended to be used for manual control, we would also want to employ a manual switch. It improves the usability of this system.

Microcontroller

Almost every control, sequencing, monitoring, and display function may be performed by a microcontroller, which is a computer on a chip. It becomes the designer's obvious option due to its comparatively modest price. The microcontroller is meant to be everything in one. Its major benefit is that all required peripherals are already included, hence no further external components are required for its operation. As a result, we can save the time, money, and resources required to build inexpensive gadgets.

Others

We require interface devices between the microcontroller pins and the high power devices in order to control various high power devices, such as lights, heaters, solenoids, and motors, using a microcontroller. There are mechanical relays, also known as contactors that can switch currents ranging from a few mili amperes to thousands of amperes. In order to adapt with high voltage ac current in this system, we need employ a relay circuit with the water pump. The negative cable side of the motor should be linked to the output of the relay circuit. A 220 volt ac current connection should be made to the cable's positive side. The electromagnetic relay may thus serve as an electrical amplifier [2].

PIC 16F84A Microcontroller

The PIC1650, which was first created by General Instrument's Microelectronics Division, is the ancestor of the PIC series of RISC microcontrollers, which is manufactured by Microchip Technologies. The PIC integrated circuit was often created to manage peripheral devices and reduce the workload on the primary CPU. The brain serves as the primary CPU in humans, whereas the PIC represents the autonomic nervous system. As a result, we suggest an affordable 8-bit PIC 16F84A microcontroller serve as the system's main controller.

PIC 16F84A Block Description

The PIC 16F84A is a member of the PIC-micro family of mid-range microcontrollers. As each 14-bit program memory word is the same width as a device instruction, the program memory has 1K words, or 1024 instructions. The RAM's data memory has a size of 68 bytes. EEPROM data is 64 bytes long. Moreover, 13 I/O pins may be individually set by the user. Some device functionalities are multiplexed with certain pins.

Memory Organization

The PIC 16F84A has two memory blocks, referred to as program memory and data memory. The software is kept in flash memory. Program memory has 1024 places and a width of 14 bits. Flash memory has a high rate of rewrite ability for updating purposes. Since the flash memory has an EEPROM, the contents will remain intact even if the power is turned off. In general, data registers are used to store numerical values like integer and floating-point numbers. It might serve as the memory's accumulator. The division of data memory into general purpose registers and special purpose registers, which are used to keep program state and store data address and other information, respectively.

Design and Implementation

An 8-bit microprocessor, an inverter, a reserve tank (res. tank), a water tank, and a water pump have all been used in this design experiment. Water level sensor has been used to operate the water pump. The water level is determined using four handmade water level sensors. Microcontroller input used to be made up of inverted sensor data. We programmed the PIC 16F84A memory using MPLAB software.

System Architecture

A water level sensor is created at the initial design stage to precisely sense the water level. The use of a microcontroller to automatically operate the whole system lowers the complexity of the design and control. The sensor unit, which monitors the water level using an inverter, provides input to the microcontroller. After analyzing the input variables, the output determines whether to turn the water pump on or off based on the tank's current water level. In Figure 1, the whole design flow chart is shown.

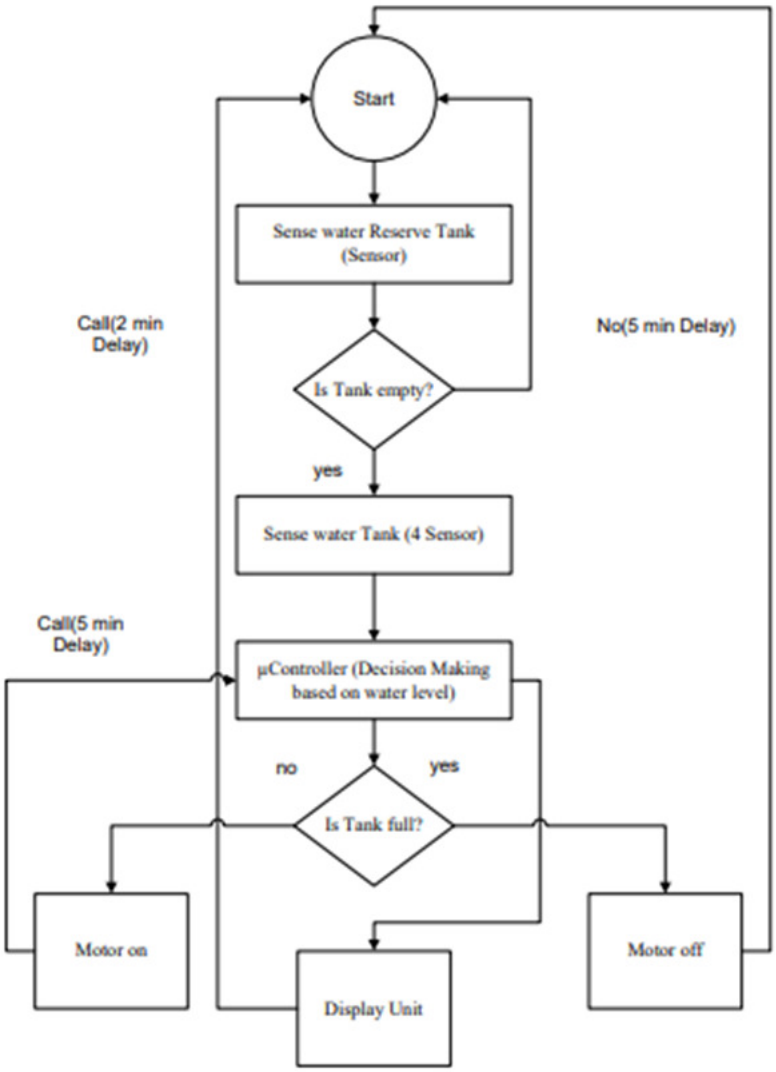


Figure 1: Represented the Flow chart of System Design.

Sensor Unit

One sensor is utilized in a reserve tank, while the other four sensors are installed within water tanks. A water level sensor unit is composed of two pieces. Moreover, sensors include rod, nozzles, inducting rubber, etc. Iron and steel are combined to create a rod that is attached to the ground. Nozzles are wired with +5 volts. Rubber is used to bond the iron rod and the nozzles together. Iron rod and nozzles' electrical connections are severed using rubber. Water conductivity has led to the utilization of a 22k resistance. The sensor's primary working principle is that when one of its nozzles is submerged in water, the conductivity of the water causes the nozzle and rod to join. The nozzle then receives a ground signal (0 v), which is linked to the inverter's input. The microcontroller, which turns on and off LEDs, receives its input from the output of the inverter and serves as the user display unit in Figure 2.

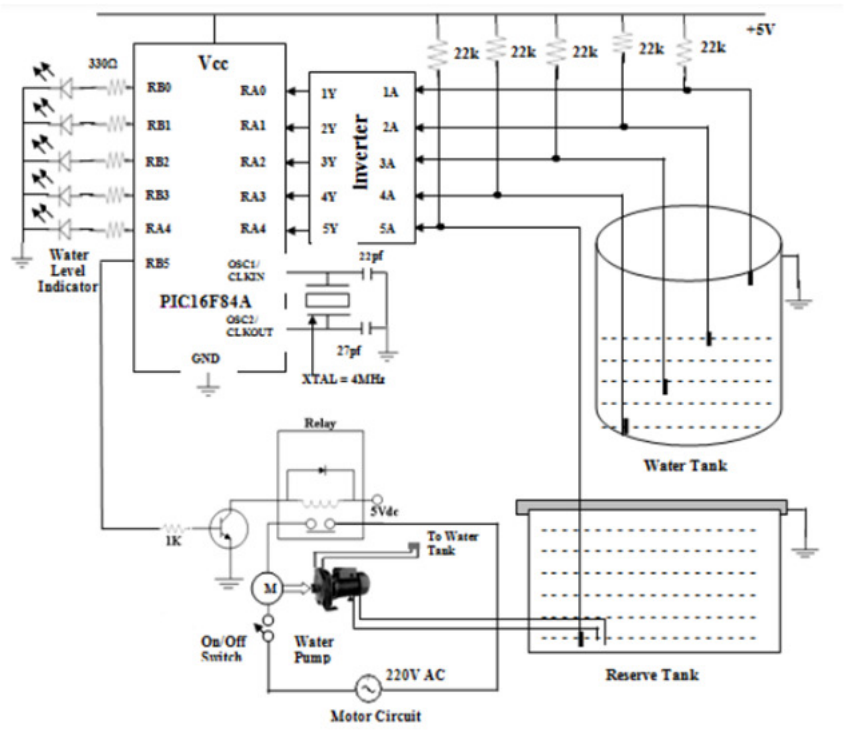


Figure 2: represented the Complete Circuit Diagram.

Control Unit

The fundamental function of the control unit is the microcontroller's control of the water pump, which is established by a specific program. A relay circuit that is linked with a transistor connects a water pump with a microcontroller's output pin. This transistor's emitter is grounded, and its collector is connected to the relay circuit. One diode is used in the relay circuit to deliver signals in one way, and one inductor is utilized to counteract a change in the direction of current flow. The motor pump's cable is linked to the relay circuit's output as a negative. Positive AC 220V electricity is connected to the opposite side of the motor cable [3].

Control unit performs following actions:

i. Off operation: The transistor turns off and its emitter and collector become open when the microcontroller applies 0 volts to the transistor's base. The relay circuit thus collects no ground signal (0 v). Hence, the negative side of the motor pump wire is receiving a positive signal (+5v). As a result of receiving a positive signal (+5v) on one side and 220v ac on the other, the motor pump will be turned off.

ii. On operation: When the microcontroller provides a positive signal (+5v), the transistor's emitter and collector become short, turning it on. As the motor pump will get a ground signal (0 volts) from the relay circuit and 220 volts of ac from the other side, the motor pump will turn on as a result.

Moreover, the inductor can withstand some resistance if the current is altered up to a positive signal (+5v) to ground or it’s opposite. We need to use a diode as a result. To manually regulate the motor driver circuit, a switch is utilized. Figure 2 depicts the control unit's schematic using a PIC16F84A microcontroller

Operation Description and Complete Circuit Diagram

The PIC 16F84A microcontroller, Crystal Oscillator, two capacitors with capacitances of 22 pF and 27 pF, an inverter, LED, a water tank, a water level sensor, a water pump, a transistor, an inductor, and a few capacitors are required to create the system. Fig. 2 depicts the visual representation of the whole circuit diagram. The microcontroller's RA4 pin is used to check whether there is water in the reserve tank. If there is no water present, it transmits a signal that controls the whole circuit and turns it OFF for a certain period of time. And it detects the reserve tank once again when the timer starts. It is possible to get inverted output from the water supply via the RA0, RA1, RA2, and RA3 pins. Crystal oscillator is attached to microcontroller pins 15 and 16. Two capacitors with corresponding capacitances of 22 pF and 27 pF link the opposite side of the crystal oscillator to the ground, acting as an external clock generator to carry out the program's instructions [4].

There is no water in the water tank if the ground signal (0 volts) is received on pins RA0, RA1, RA2, and RA3. Hence, all LEDs need to be off. We may also solve this problem in a cleverer manner; if pin RA3 detects a ground signal, we can be certain that the tank is dry. And if we discover that pin number RA0 detects a positive signal (+5v), we may infer that the water tank is filled with water. So, the water pump should turn on and the LED lights should turn out when the water tank is empty. If pin number RA3 receives a positive signal (+5v) and the other three get a ground signal (0v), the water level in the tank is 1/4 full. Due to this, the water pump is still running and the first LED should be on at this point. The other three Lights are still off. [5] The presence of a positive signal (+5v) on the four RA pins indicates that the water tank is presently filled with water. The water pump should now be turned off and all of the Lights should be on because of this. You'll see that we employed these Lights to monitor the user's house visually. The display LEDs should begin to turn off one by one when the water level in the tank decreases due to domestic usage. The water pump should turn back on automatically right after the final Light goes out if all the LEDs go out, which signifies the tank is once again empty. These processes ought to run in a cycle automatically[6].

Programming Description

Assembly language was utilized to create the software that managed the whole procedure on the PIC16F84A microcontroller. With the aid of MPLAB software, which is offered by MICROCHIP, all the codes have been tested and simulated. Our system uses a Crystal Oscillator (XTAL) 4MHZ as the external timer. When the system first turned on, the microcontroller used the inverter to receive information from the water sensor. Inverted inputs from the microcontroller's RA0, RA1, RA2, and RA3 are fed into a register, and its combination is verified. The following procedure was used to check the combination [7], [8].

i. After receiving the first pin signal, the microcontroller inserts the signal into its register. The next pin signal is then examined before being loaded into the register. The similar method is used for the functioning of other pin signals. Finally, it fills the register with the signals from all necessary pins. It decides on an output and transmits that signal to the output pin utilizing these four signal combinations.

ii. ii. With regard to the input signals, the whole process cycles or repeats itself.

DISCUSSION

The created system has been empirically shown to be a robust and trustworthy instrument for gauging water level in a variety of applications. It is affordable and versatile to use, and it has successfully achieved the goal of transferring data across small distances utilizing wireless communication networks. Moreover, the multipoint short-range wireless data collection and transmission network's integrated wireless receiver module offers a low-power, high-performance wireless data communication system that collects data from sensors using RF modules. A low-cost, adaptable, and portable WiDAS has been created to satisfy all of these needs. Moreover, a microcontroller-based Tx-Rx ultrasonic sensor system with the ability to measure water level has been created. The procedure was created in order to collect data from ultrasonic sensors, process it on a microcontroller, and then broadcast it to another microcontroller via a transmitter. Data that has been received by a receiver may be shown on an LCD screen. A buzzer will activate in the event of an overflow or empty level and the LED will display the various water levels [9], [10].

CONCLUSION

One of the most essential fundamental necessities for all living things is water. Yet regrettably, unchecked usage wastes a significant quantity of water. Several automatic water level monitoring systems have been proposed in the past, but most of them fall short in actuality. We made an effort to solve these issues by putting in place a productive automated water level monitoring and regulating system. We wanted to develop a system that was adaptable, affordable, and simple to configure in order to address our water loss issue. In order to keep costs down, we utilized a PIC 16F84A microcontroller, which is inexpensive. We successfully tested the system in the lab and have since developed a web-based network for monitoring and managing water levels. This network's versatility will allow us to operate the system from any location using the internet and a variety of devices. This study effort may have a significant impact on effective water management.

REFERENCES

[1] D. Z. Wan and C. S. Chin, “Simulation and prototype testing of a low-cost ultrasonic distance measurement device in underwater,” J. Mar. Sci. Technol., 2015, doi: 10.1007/s00773-014-0270-5.

[2] A. Big-Alabo and C. Isaac, “Automatic water level control system using discretized components,” J. Appl. Sci. Environ. Manag., 2020, doi: 10.4314/jasem.v24i10.11.

[3] H. F. Atlam, A. Alenezi, M. O. Alassafi, and G. B. Wills, “Blockchain with Internet of Things: Benefits, challenges, and future directions,” Int. J. Intell. Syst. Appl., 2018, doi: 10.5815/ijisa.2018.06.05.

[4] T. A. Howell, S. R. Evett, S. A. O’Shaughnessy, P. D. Colaizzi, and P. H. Gowda, “Advanced irrigation engineering: precision and precise.,” J. Agric. Sci. Technol. A, 2012.

[5] S. Suganya, “IoT based Standard Water Measuring System using GSM,” Int. J. Res. Appl. Sci. Eng. Technol., 2018, doi: 10.22214/ijraset.2018.4016.

[6] A. Ghosh, H. Ray, T. K. Ghosh, A. Das, and N. Bhattacharyya, “Generic handheld E-Nose platform for quality Assessment of agricultural produces and biomedical applications,” Chem. Eng. Trans., 2014, doi: 10.3303/CET1440044.

[7] K. York and S. Lewallen, "Expanded role for portable emissions analyzers," Diesel Gas Turbine Worldw., 1995.

[8] V. Bennett, T. Abdoun, K. O'Meara, M. Barendse, and T. Zimmie, "Wireless MEMS-Based In-Place Inclinometer-Accelerometer Array for Real-Time Geotechnical Instrumentation," in Sustainable Civil Infrastructures, 2018. doi: 10.1007/978-3-319-61648-3_6.

[9] Reinaldo L. Gomide, Ricardo Y. Inamasu, Daniel M. Queiroz, Evandro C. Mantovani, and Werner F. Santos, "An Automatic Data Acquisition and Control Mobile Laboratory Network for Crop Production Systems Data Management and Spatial Variability Studies in the Brazilian Center-West Region," 2013. doi: 10.13031/2013.7332.

[10] J. Cai, Y. Liu, L. Bai, H. Chen, and X. Li, "Low-cost and low-power dissipation system to monitor soil water status in real time for areal irrigation management," Nongye Gongcheng Xuebao/Transactions Chinese Soc. Agric. Eng., 2015, doi: 10.11975/j.issn.1002-6819.2015.20.013.

CHAPTER 17
AN OVERVIEW OF AUTOMOTIVE-GRADE MICROCONTROLLERS

Ajay Rastogi, Assistant Professor
College of Computing Sciences and IT, Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India
Email Id- ajayrahi@gmail.com

ABSTRACT:

Modern Deep Learning algorithms, which are able to predict the behaviors and operations of many physical systems, rely heavily on neural networks to be realized. Particularly when compared to the volume of data that neural networks normally need to generalize, a model's computational resource requirements are high. In this paper, we present a framework for implementing neural network-based models on a family of automotive microcontrollers, demonstrating the effectiveness of the framework through two case studies implemented to vehicles: intrusion detection on the Controller Area Network bus, and residual capacity assessment in Lithium-Ion batteries, which are widely used in Electric Vehicles.

KEYWORDS:

Datasets, Gaze Detection, Microprocessor, Microcontroller, Neural Networks, Text Tagging.

INTRODUCTION

One of the most innovative technologies powering the fourth industrial revolution is machine learning (ML), which makes it possible to extract useful information from vast volumes of data that the human brain is unable to do on its own. Deep Learning (DL), Deep Neural Networks (DNNs), and Deep Generative Models are just a few of the ML subfields that have seen significant advancements recently as a result of the rapid evolution of ML research (DGMs). Social media platforms, industrial control systems, and Internet of Things sensors are among the primary sources of data generation (IoT). Such a massive volume of data calls for incredibly sophisticated processing resources in order to develop relevant and trustworthy models. Nevertheless, there has been an increase in interest recently in embedding machine learning (ML) models inside microcontroller units (MCUs), which are tiny computers with inbuilt memory that are often at the heart of embedded technologies. Embedded systems are often needed to analyses information and make real-time choices depending on resource-constrained hardware in the healthcare, energy, consumer electronics, and automotive industries. Pre-trained models can now be incorporated on embedded systems thanks to the most recent TinyML technologies, opening up the possibility of making edge computing more efficient, affordable, and secure [1].

While TinyML technologies have their roots in the consumer and industrial worlds, numerous industries, including the automobile sector, stand to gain significantly from their use. In actuality, the computing and data storage capabilities of vehicle control systems are often constrained. The architectures built into automotive MCUs are designed to assess data as it comes in, with the training stage taking place offline on more capable computers [2]. Thus, it is crucial to use effective models in the toolchain, whose offline training phase takes into account the fact that the on-board

control units must carry out the main tasks for which they were created, ranging from the administration of on-board devices to engine control. In this paper, we present a framework for implementing NNbased models on a family of automotive MCUs, demonstrating their effectiveness in two case studies applied to vehicles: intrusion detection on the CAN bus and residual capacity estimation in Lithium-Ion (Li-Ion) batteries, which are widely used in Electric Vehicles (EVs). Two vehicle-related TinyML case studies are presented in the following sections. Electronic Control Units (ECUs), embedded systems that govern several vehicle operations and whose functional needs have become more complicated, have been added to contemporary automobiles in recent years [3].

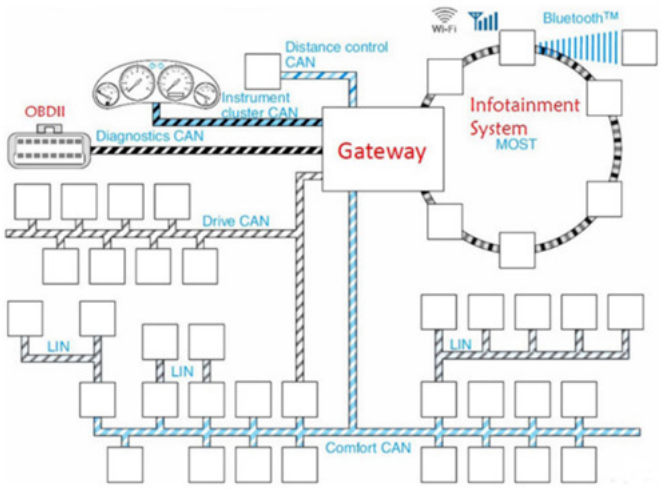


Figure 1: Represented the Vehicle Communication.

Figure 1 depicts an illustration of an in-vehicle communication system that consists of many sub-networks linked by a gateway and including various features. It becomes more crucial in such a system to mix conventional control methods with ML and Artificial Intelligence (AI) capabilities. TinyML technologies, which are already extensively utilized in other consumer applications, are gaining attention since MCUs in automobile embedded electronics are resource-constrained. Every year, both the industry and the literature produce new ML solutions for the automobile sector [4]. In this research, we provide a technique for embedding machine learning (ML) models into automotive grade MCUs, and we use it in two real-world applications, which are discussed below.

Intrusion Detection Systems

A multi-master messaging protocol called CAN bus is used in automotive systems to link ECUs (e.g., modern vehicles). Real-time communication and minimal implementation costs are met by this protocol of communication. The de facto standard for connecting ECUs is the CAN bus, which has a maximum transfer rate of 1 Mbit/sec. In order to avoid assaults on ECUs, such as those that take advantage of holes in the infotainment system, vehicle makers must carefully design connections between essential and noncritical sub-networks. Researchers proved that the braking system may be disabled by taking use of a feature included in the CAN standard. Any car that uses a CAN bus is vulnerable to attack, despite the fact that the CAN protocol was initially intended to be safe. The following is a list of CAN bus vulnerabilities that researchers have previously identified:

i. It is a multicast messaging protocol without an inherent addressing and authentication system. Thus, a hacked ECU has access to every communication in its sub-network and is capable of forging the sender's identity to transmit messages.

ii. Since modern automobiles use a protocol with limited capacity, adding message encryption is difficult.

iii. The majority of nodes are automotive-grade MCUs, which have constrained memory and processing power and make it difficult to implement intricate security mechanisms.

A good defense against CAN bus vulnerabilities is the addition of an IDS. The anomaly-based approach is one of the intrusion detection techniques. An easy way to explain this technique is to imagine a monitoring system an ECU that observes typical behavior by listening to CAN bus data. The trained IDS is informed by the anomalous traffic caused by the intruder actions [5].

Battery Releasable Capacity

Due to its many benefits, including high specific energy and power, Li-Ion batteries have attracted a lot of attention in recent years. Several systems, including mobile devices like smartphones and automotive systems like Hybrid Electric Vehicles (HEV) and Electric Vehicles, are powered by rechargeable battery stacks based on Li-Ion cells (EV). The Battery Management System (BMS) characteristics must be improved if a battery is to be made safer, more dependable, and more economical. Estimating battery capacity is crucial in this aspect since it enables the State of Health computation (SoH). A key indicator for the Battery Health Monitoring system, SoH measures how well a battery performs in energy storage and delivery (BHM). Even if a battery isn't being utilized, inherent ageing processes cause capacity to degrade over time, lowering performance. Usually, a 20% decrease in rated capacity is regarded as the component's maximum safe operating range i.e., Cmax 0.8Crated, beyond which the battery's performance may not be dependable. SoH diagnosis and precise releasable capacity estimate are crucial for lowering safety risks, preventing major failures, and choosing the right battery replacement. For estimating battery capacity, data-driven algorithms based on ML techniques are often utilized. Starting with quantifiable characteristics, like as voltage and current, which are readily collected from a car through CAN bus, they calculate the releasable capacity. DL algorithms are shown to be more adaptable and effective than conventional techniques since SoH is profoundly non-linear and not immediately observable.

SPC5-STUDIO-AI:

The SPC5-STUDIO development environment, which supports the Chorus SPC58 MCU family, has a plug-in component called Studio-AI. The block diagram are display in Figure 2. Using automotive-grade MCUs, this tool offers the ability to automatically develop, run, and evaluate pre-trained NN models. It automatically assesses the models' complexity and translates them into ANSI C code that is optimal for SPC58 MCUs. Such DL frameworks as Keras, TensorFlow Lite, ONNX, Lasagne, Caffe, and ConvNetJS are supported. The ANSI C library may be imported into projects that are application-specific since it has a limited number of well-defined public APIs [6].

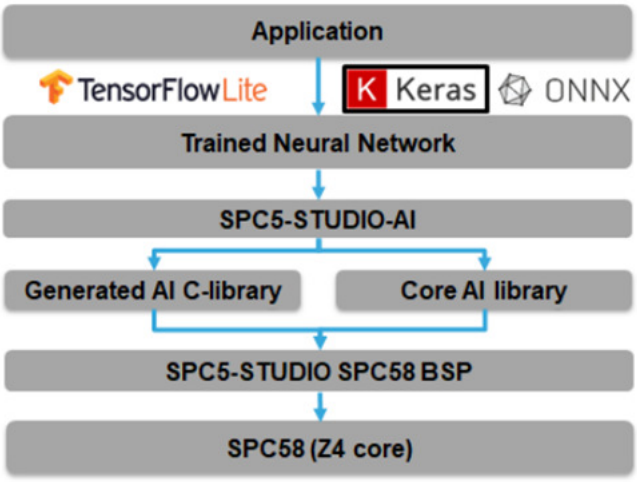


Figure 2: Represented the SPC5-Studio.AI Block Diagram.

Moreover, SPC5-Studio-AI offers NN validation and complexity profiling directly on MCUs, evaluating important parameters like validation error, memory needs (including Flash and RAM), and execution time. The SPC5-STUDIO development environment includes this plugin.

SPC5-Studio

Three automotive-grade MCUs, the SPC584B, SPC58EC, and SPC58NH, which are suited for applications requiring low-power, connection, and security, were utilised to embed and test the generated NNs using AI. Table 1 displays the salient characteristics of the selected MCUs. Each MCU's power use was calculated with its highest frequency and all cores active.

Table 1: Main features of the Automotive-grade MCUs used for the Complexity Analysis.

Device	Flash [Mb]	RAM [Mb]	Clock [MHz]	I/D Cache	FPU	Power Consumption [mA]
SPC584B	2	192	120	Yes	Yes	102.0
SPC58EC	4	512	180	Yes	Yes	132.6
SPC58NH	10	1024	200	Yes	Yes	239.6

Up to now, the CubeSat platform has only been used for brief technological demonstrations and educational purposes. The first CubeSats are now in space, and many businesses want to use them to provide commercial Earth observation services. The use of commercial CubeSats highlights the broad business potential of nano and microsatellites [7]. More dependability is needed for commercial initiatives than for isolated technical demos. The conventional space industry creates dependable parts, but nanosatellites, which have smaller sizes and shorter lifespan, cannot use technologies made for giant satellites with potentially long lives. On the other hand, dependable functioning throughout the course of their lifespan would be advantageous for commercially used nanosatellites in order to increase their duty cycles. Hence, reliable nanosatellite missions must be smaller, cheaper, and more reliable.

Space electronics face difficulties throughout the pre-launch, launch, and operating periods. The chosen and acquired components may age many years during design, assembly, storage, and transportation in the pre-launch phase and may not be at the cutting edge of technology at launch. Nevertheless, as the mission and system designs are built on existing components, this seldom results in issues if the components have been acquired before they become outdated. The electronics are exposed to noise, vibrations, and shocks during the launch phase. Unless there are weak solder connections in the electronics, this should not be a concern since it mostly affects mechanical structures [8], [9].

Equipment in space is subjected to operational and environmental pressures during the operating phase. Vacuum, heat fluctuation, and radiation are the three basic environmental risks. Even commercial nanosatellite initiatives seldom aim for more than a few years of operation since the major issue created by vacuum is tin whiskers growing in lead-free tin solders over a number of years. Thermal changes shouldn't affect the satellite's electronics if the passive thermal design is sound. The radiation environment is the most significant long-term environmental problem for electronics; both single-event effects (SEE) and total ionizing dose (TID) impacts must be taken into consideration. Single-event upsets (bit flips) and single-event latches are the two basic categories into which single-event effects may be separated (short circuits). Bit flips may happen in registers, volatile and non-volatile memory, and anywhere else where electric charge is utilised to hold bits. Consequences might range from relatively innocuous data noise to catastrophic program execution problems that could result in mission failure. If current is not restricted and a device is not power cycled when issues are found, short circuits may permanently harm the gadget. Even though so-called micro-latchups may only result in little increases in current, they halt operating devices. Error correction, current limiting, and power cycling unresponsive devices may all reduce the majority of single-event impacts.

Due to cumulative ionizing radiation effects brought on by accumulated ionizing exposure over time, equipment deteriorates with time. Impacts often become apparent initially as modifications to the device's electrical specifications, such as an increase in power usage. Device failure ultimately results from this deterioration. Even with the little shielding offered by CubeSat frames in LEO, most COTS components tolerate at least 5 krad (Si) of total exposure, and it sometimes takes years to collect this dosage. Operational requirements include the need for fault-tolerant functioning under particularly challenging diagnostic and repair scenarios. A radiation-related mistake or a ground control mistake might lead the spacecraft to take on a dangerous configuration. This must be reduced by a system that enables the spacecraft to quickly revert to a safe condition and continue regular operations. Single points of failure should be avoided in system design, and any damaged component that might endanger ongoing operations has to have a solution. Redundancy is generally used to accomplish this.

On-board Computer Requirements

The Aalto satellite, a nanosatellite project for Earth remote sensing being developed at Aalto University, serves as the example case for the discussion in this article. The proposal calls for a spaceship with a two-year minimum lifespan. The equipment is ideally over built for extended lives since the project focuses on genuine remote sensing rather than technical demonstration. The Aalto satellite OBC's primary duties are scheduling and allocating instructions, gathering and producing housekeeping telemetry, and offering a CCSDS-compliant TTC interface. The CAN

bus, which is reliable and has strong COTS component availability, must allow the OBC to control the satellite. Hence, requirements comprise:

a. Telemetry and telecommand interface that complies with CCSDS
b. the delivery and storage of telecommands at predetermined times
c. telemetry and housekeeping
d. the CAN spacecraft data bus master
e. a trustworthy mission life of at least two years
f. Form factor suitable with CubeSat

In this instance, the on-board computer just serves as a high-level controller of the satellite and does not handle data from the payload. While the OBC's use of the fault-tolerant Hercules microprocessor falls short of the standards and costs of the conventional space industry, it offers dependability that is higher than that of prior CubeSats. The OBC and associated controllers need less RAM and mass storage since orbit and payload control are handled by separate Hercules processors, and payload data processing is done using FPGAs.

Hercules Tms570 Family

Texas Instruments Hercules TMS570 is a microcontroller family designed for automotive-grade safety-critical applications, with aerospace and avionics as possible secondary applications. Vaughan states the TMS570 family has been developed in part to simplify the development of fault tolerant systems that meet the strict safety standards in the automotive and aerospace industries. TMS570 has been designed with internal diagnostics, lockstep processors and error correction features that increase fault tolerance. Texas Instruments also provides an integrated development environment with development tools, such as a Misra-C checker, that can be used to aid development of reliable software. The highest-end Hercules model, TMS570LS3137, operates at clock speeds of up to 180 MHz and has 3MB program memory, 256 kB RAM and 64 kB emulated EEPROM. Relevant safety features include two processors running in lockstep, error correction codes (ECC) in all internal memories and internal voltage and clock monitors.

TMS570LS3137 has two Cortex-R4F CPUs running in lockstep. The operation of the primary CPU is checked against a secondary CPU that has a physical separation inside the package and a temporal separation of two clock cycles. Mismatch in the CPU outputs causes an interrupt and an error signal. TMS570 also has ECC in all internal memories. Voltage and clock monitors respond to gross deviations from nominal parameters, but external voltage monitoring is recommended as an additional safeguard. Startup diagnostics are performed at boot, and errors again cause a signal to be output to the error pin. Some errors detected by the diagnostics can be corrected transparently at run-time, and uncorrectable errors cause an error signal to be output. TMS570LS3137 also has three internal CAN bus controllers. CAN bus is a reasonable choice for the data bus of a small satellite because of its robustness, extensive COTS component availability and the ongoing ECSS standardization effort.

Suitability for Space Applications

The primary environmental concern with microcontrollers is the radiation tolerance. Texas Instruments provides radiation test data only under a non-disclosure agreement. However,

TMS570 already has limited flight heritage aboard two successful CubeSats, CubeBug-1 and -2, and both Surrey Space Centre and NASA plan to use it in upcoming missions. This constitutes an experimental demonstration of the basic viability of TMS570 for space applications. Taylor et al. found in that the high performance and internal fault tolerant features of the TMS570 allow it to carry out many tasks performed earlier by other devices in the space industry. The TMS570 will be used in the TechDemoSat-1 mission as an intelligent supervisor in the Micro Radiation Environment Monitor (MuREM) experiment. NASA selected TMS570 for the motor driver electronics of the Robonaut 2 Climbing Legs. TMS570 was selected to achieve acceptable performance for the system in the EVA radiation environment. NASA has performed radiation testing on all Climbing Leg electrical components, including TMS570, and found that all radiation-induced events are detectable, non-hazardous and non-permanent. Additionally, NASA has used thermal vacuum testing on the Climbing Leg actuators and motor driver electronics, including TMS570, to verify that all materials in the Climbing Legs are EVA compatible.

On-Board Computer Concept

The designed Aalto satellite on-board computer uses the Hercules TMS570LS3137 microcontroller, as this highest-end model has internal 3 MB Flash memory and 256 kB RAM. Chips with quad flat packages (QFP) have been selected to facilitate reliable manual soldering. No external RAM is needed since the most intensive operations performed by the OBC are telemetry and telecommand processing. Telemetry and telecommands are formatted according to CCSDS protocols for compatibility with commercial ground station providers. The external memory interface of the TMS570 could be used to include additional RAM, if such a need should arise. In that case, error correction would need to be separately implemented for the external RAM. Payload data processing is intensive, but performed by stand-alone FPGAs in the reference mission. The OBC operates the payload, but as the payload data is processed and downlinked by a separate system, no extra RAM is needed in the OBC. Instead, the external memory interface is used to interface with a redundant mass memory system to store command schedules, telemetry and configuration information. Texas Instruments provides drivers for NOR flash and SDRAM support. At the time of writing NAND flash was not directly supported and required custom drivers. This could be a limitation if fast, highdensity memory is required. SPI can be used to interface with SD cards, providing large capacity with limited speed. SD cards are used for prototyping, but because of radiation reliability issues with SD cards, the flight model instead uses redundant NOR or NAND flash memories for the mass storage system.

Fault Tolerant Architecture

Failures can be divided into two main categories: systematic failures and random failures. Systematic failures, such as programming errors, result from failure in design, manufacturing or from failing to follow best practices. Random failures are random defects that result from the working environment of the device, such as radiation-induced bit flips and short circuits. As random failures cannot be completely prevented, focus must be on mitigating and handling them. Fault tolerance is always based on redundancy which can be spatial, temporal or informational. The fault tolerant architecture of the Aalto satellite OBC aims to be tolerant of failure of any single component. Two cold redundant Hercules microcontrollers with their own clock systems and power supplies are used. The circuit allows only one Hercules to be powered at the same time. MSP430, a CubeSat-proven 16-bit microcontroller, is used as an arbiter to switch between the Hercules systems. The circuit is designed in a way that one of the redundant TMS570

microcontrollers is enabled by default by a pull-up resistor in case the MSP430 arbiter fails. A common mass storage system is available for both Hercules microcontrollers through the Hercules External Memory Interface pins. The mass storage system consists of physically redundant memory circuits that store identical data. Both power systems can be used to power the mass storage system. A bus switch is used to select which Hercules system is connected to the spacecraft data bus and to the mass storage system. The fault tolerant architecture is depicted in Figure 2.

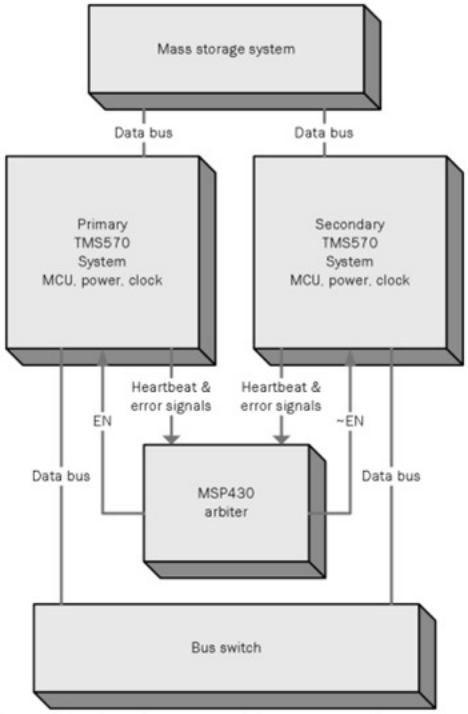


Figure 2: Represented that The TMS570-based Fault Tolerant Architecture.

The error signals and heartbeats provided by the TMS570 MCUs are used as inputs for the MSP430. Certain errors, such as compare errors between the two processors inside the TMS570 and uncorrectable memory errors, cause an error signal to be output from the error signaling module pin of the Hercules. This signal can be used to reset the system or perform other actions to respond to the error. In the Aalto satellite OBC implementation, any error severe enough to cause an error signal to be output causes a system reset. The design provides fail-silent operation: if a Hercules determines it cannot function, it goes silent, and tries to reboot; if this doesn't help, the MSP430 arbiter switches to the other Hercules based on the lack of a heartbeat. The internal fault protection mechanisms in the TMS570--lockstep CPUs, ECC and voltage and clock monitoring--allow correction of some error conditions, such as bit flips, at run-time; if the corrections fail, the error signal from the error signaling module (ESM) can be used to reboot the system or give control to the backup Hercules. Single-bit errors in a single RAM block are corrected transparently in runtime; two-bit errors in a single memory block because an error signal to be triggered.

In addition to this cold-redundant architecture, a hot-redundant architecture could also be used in hard real-time systems. Instead of a conventional two out of three-voting, two Hercules processors could be run in parallel, performing the same operations with same inputs but with only the primary one being allowed to control the system. If the primary Hercules detects a serious fault, it can use

its error pin to signal a secondary Hercules to take control while it restarts. Control can be returned to the primary controller when it signals "ready" to the secondary Hercules. However, this hot redundant, fail-operational architecture is not considered necessary for the on-board computer of the Aalto satellite.

Mass Storage System

SD cards, interfaced through SPI, could be used for mass storage, and they are a popular option in existing CubeSat on-board computers. Industrial-grade SD cards by Delkin Devices have been radiation tested for total ionizing dose, surviving doses up to 24 krad(Si). The cards are offered in industrial temperature ranges and in capacities between 128 MB and 8 GB. Delkin SD cards are unlikely to succumb to total dose even after several years in space, but single-event effects demand more consideration. Single-event upsets can be mitigated by the internal ECC of the SD cards, but latchups or other single-events affecting the SD card controller could cause component failure. Three redundant SD cards could be used on the same SPI bus, and all data could be redundantly maintained on all of them. ECC within the cards can be used for verifying data integrity, and in case of corruption in a single card, the data can be corrected using a backup stored on another card. The system could tolerate the failure of at least one and even two SD cards. The OBC would need to maintain identical file systems on all three cards. SD cards also facilitate easier laboratory prototyping. However, even multiple redundant SD cards may not achieve high enough reliability for several years. The SD card in the RAX-2 CubeSat failed after a few months. Kimura et al. tested various SD cards in and found that while SD cards can survive high total ionizing doses in Co-60 gamma ray testing, relatively small high-energy proton exposure can irrecoverably damage them.

The mass storage system in the Aalto satellite is therefore based on either NOR or NAND flash memories with available radiation data, as Hercules has an external memory interface (EMIF) supporting NOR flash, and custom NAND drivers can be developed. With NOR flash, the storage size is more limited than with SD cards, but this does not pose a problem since the on-board computer does not process payload data, but only handles telecommands and collects telemetry. The bus switch is used to select which Hercules has access to the mass storage system. The file system drivers are implemented in a way that transparently maintains an identical file system on all three memories, and the low-level flash operations are masked in the file system software; for the software developer, it does not matter whether the hardware implementation is SD card, NAND flash or NOR flash. In case of a total failure of the mass storage system, the 64 kB internal EEPROM of the Hercules could still be used to store vital configuration settings and command schedules[10].

Software Layer

The software layers of the Aalto satellite OBC are depicted in Figure 3. The application software implements the CCSDS-compliant telemetry and telecommand interface, command scheduling, housekeeping, and other required features. Free RTOS is used for scheduling and FatFS for file system abstraction. Texas Instruments provides a hardware abstraction layer code generator (HalCoGen) tool for generating drivers for the Hercules. The drivers generated by HalCoGen are evaluated against the Motor Industry Software Reliability Association's C coding standard (MisraC). TI also advocates using Free RTOS as the real-time operating system (RTOS) layer. FreeRTOS is almost completely Misra-C compliant. Using Misra-C increases software reliability in the Texas Instruments driver and Free RTOS layers by avoiding the pitfalls of C.

Driver and Free RTOS code can be auto-generated with HalCoGen and only require configuring some interface parameters for the specific application. Software development can then immediately concentrate on the high-level flight software. An open source File Allocation Table (FAT) file system implementation, FatFS, is used to mask the low-level implementation of the mass storage. FatFS only requires a few hardware-specific drivers to be written.

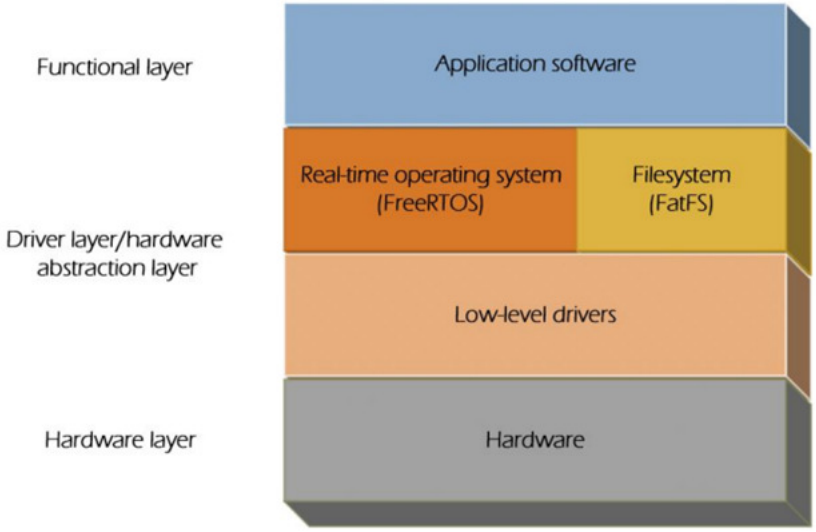


Figure 3: Represented the Software Layers when using Free RTOS for Scheduling and FatFS for Mass Storage.

DISCUSSION

According to Eickhoff, on-board computers should provide failure robustness using internal redundancy, electromagnetic compatibility with the space environment including the radiation environment, and a variety of bus interfaces to interface between other spacecraft subsystems. Onboard software should be a real-time control software that allows remote and automated control of the spacecraft. The largest issues for on-board computers in space are radiation hazards and operational fault tolerance. Like the space industry, automotive and aviation industries place emphasis on reliable operation of equipment. Fault tolerant electronic systems have become more important in the automotive and aviation industries due to the trend of moving from mechanical systems to electronic x-by-wire systems. Using components and technologies developed for automotive and aviation industries can leverage the huge research and development resources of these industries. For example, Texas Instruments has brought the automotive-grade TMS570 lockstep microcontroller family, with many internal fault tolerant features, to the market. The internal fault-tolerant features in TMS570 allow run-time correction of several error conditions [11].

CONCLUSION

The traditional space industry has concentrated on building large satellites with long lifetimes. The small satellite revolution has created a new industry alongside the traditional space industry, and the new industry produces smaller satellites with shorter lifetimes. In this paper, the design for a Hercules TMS570 -based cold-redundant on-board computer was presented to meet the requirements of a nanosatellite mission, Aalto satellite, with 2 years of operational lifetime. The existing CubeSat solutions are designed mainly for one-off technology demonstrations, while the

traditional space industry components are designed for nearly decades of lifetime. A Hercules-based on-board computer provides an option somewhere in between, and such an on-board computer should provide the required 2-year mission lifetime with low cost. The versatility and robustness of the Hercules allows it to be used in other tasks as well, such as attitude, propulsion and instrument control. This is also the case in the Aalto satellite mission discussed in this paper. While the fault tolerant architectures considered in the paper are not novel, the TMS570 provides a new, simpler way of implementing them. For nanosatellite missions, Hercules-based on-board computers provide better reliability to price ratio than other, previously available options.

REFERENCES

[1] N. A. Mutalib et al., "Development of shoe dryer system using microcontroller with GSM module," Int. J. Integr. Eng., 2019, doi: 10.30880/ijie.2019.11.03.029.

[2] P. Cappelletti, R. Annunziata, F. Arnaud, F. Disegni, A. Maurelli, and P. Zuliani, "Phase change memory for automotive grade embedded NVM applications," Journal of Physics D: Applied Physics. 2020. doi: 10.1088/1361-6463/ab71aa.

[3] F. Disegni et al., "Embedded PCM macro for automotive-grade microcontroller in 28nm FD-SOI," in IEEE Symposium on VLSI Circuits, Digest of Technical Papers, 2019. doi: 10.23919/VLSIC.2019.8778129.

[4] H. Kim, K. Kim, H. Kwon, and H. Seo, "ASIC-resistant proof of work based on power analysis of low-end microcontrollers," Mathematics, 2020, doi: 10.3390/MATH8081343.

[5] C. H. Choi, "Teaching Electromagnetic Interference on Microcontrollers through Lab Experiments," Int. J. Emerg. Technol. Learn., 2020, doi: 10.3991/ijep.v3i2.2252.

[6] L. Popa, B. Groza, and P. S. Murvay, "Performance evaluation of elliptic curve libraries on automotive-grade microcontrollers," in ACM International Conference Proceeding Series, 2019. doi: 10.1145/3339252.3341480.

[7] T. Andreica, B. Groza, and P. S. Murvay, "Applications of pairing-based cryptography on automotive-grade microcontrollers," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2018. doi: 10.1007/978-3-319-99229-7_28.

[8] B. Groza, L. Popa, and P. S. Murvay, "Highly Efficient Authentication for CAN by Identifier Reallocation with Ordered CMACs," IEEE Trans. Veh. Technol., 2020, doi: 10.1109/TVT.2020.2990954.

[9] V. Vasquez Lopez, J. M. Echeverry Mejia, and D. E. Contreras Dominguez, "Design and Statistical Validation of Spark Ignition Engine Electronic Control Unit for Hardware-in-the-Loop Testing," IEEE Lat. Am. Trans., 2017, doi: 10.1109/TLA.2017.7994782.

[10] M. Carfì, ; M Perroni, ; M Caruso, G. Piazza, ; O Weber, and ; P Zuliani, C17-2 Embedded PCM Macrocell for Automotive-Grade Microcontroller in 28nm FD-SOI Technology. 2013.

[11] F. Arnaud et al., "High density embedded PCM cell in 28nm FDSOI technology for automotive micro-controller applications," in Technical Digest - International Electron Devices Meeting, IEDM, 2020. doi: 10.1109/IEDM13553.2020.9371934.

CHAPTER 18

AN ELABORATION OF ASSISTANCE OF THE MICROPROCESSOR IN MEDICAL FIELD

Rohaila Naaz, Assistant Professor
College of Computing Sciences and IT, Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India
Email Id- rohailanaaz2@gmail.com

ABSTRACT:

The development of medical instrumentation systems is made possible by developments in computational hardware technology. The design of such a system is adversely affected by other technical advancements such as VLSI (Very Large Scale Integration) technology, microprocessor, digital signal processing (DSP), and discrete signal controller (DSC). The architectural progression of embedded cores has an impact on the platform's performance in terms of processing speed and power efficiency. The design of embedded cores therefore heavily relies on the programmable system. The system's effectiveness is greatly enhanced when one emphasizes on choosing the best controller for a certain application from the different possibilities on the market.

KEYWORDS:

Application Specific Processor (ASP), CISC, General Purpose Processor (GPP), Microprocessor, Microcontroller, RISC.

INTRODUCTION

Intelligent system requires smart controller for performance improvement of the system. Medical instrumentation also requires intelligent controller for more sophisticated facilities to improve the performance. Behind the intelligent system there is an intelligent controller plays an important role. In the block diagram shown in Figure 1, the medical instrumentation application requires sensor for sensing physiological signal from the patient body. After getting electrical signal from the sensor output the signal is amplified and conditioning for appropriate signal strength. Since the intelligent digital controller requires digital data to further processing there is Analog to Digital Convertor (ADC) chip. The controller here is responsible for overall handing of the signal like recording, displaying and sending to the remote places. Above mansion functionality of the system requires different architecture of the controller to handle different activity like speed, data storage programmability and power consumption of the system.

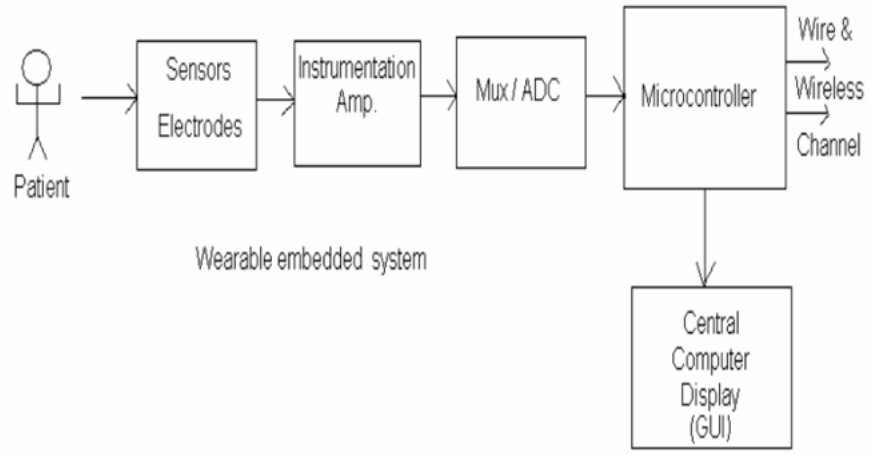


Figure 1: Represented the Wearable Embedded System.

The performance of the system depends on the processing power and the computational power of the central controller unit. Depending on the central controller selection one can improve the system performance in terms of power, speed and cost of the system. There are varieties of controller unit available like microcontrollers, Programmable system on chip, Digital Signal Processor and soft core which can be utilized on specialized programmable hardware like FPGAs and CPLDs. In this paper we surveyed different verities of controller available in the market with its different capability to handle the system requirements. In the following subsequent section we focus briefly on the specific category of the controller which enhancing the performance of the systems [1].

Microcontrollers

The advancement in the field of microprocessor offers the systems with lot of embedded peripherals in a single chip. The microprocessor with on chip internal peripherals results it to a single chip microcontroller. The microcontroller now days available with memory incorporated on chip. The power requirement is also low without affecting the performance of the controller. The on-chip resources in a single chip reduce the cost of the embedded systems. The microcontrollers now days available with lot of peripheral on chip like Analog to Digital Convertor (ADC) and DAC into the same chip. By using those peripherals one can reduce the interfacing task on the PCB, and reduce the cost of the system[2].

The programming of such varieties of microcontrollers can be done by the Electronic Design Automation Tools (EDA) which is provided by the controller manufacturer. The EDA tools used to generate specific hex code is known as cross-compilers. Today cross-compilers are available and due to that the microcontrollers can be programmed in the higher-level languages. The cross-compilers offer the programming in the higher-level language like C, C++, and JAVA. So, virtually user can work with any microcontrollers, however the user has to learn about the internal architecture and peripherals detail about that particular microcontroller. Here the assembly language mnemonics is not required to know. Even some of the cross-compilers also support simulator for the respective devices for testing various software module without investing any hardware.

Programmable System on Chip (PSoC)

The on-chip peripheral can also be incorporated with controller into the chip itself by programmable capability by software tools provided by the chip manufacturer. One such device from cypress semiconductor is PSoC. In PSoC we can configure the programmable peripherals like ADC, DAC, UART etc. The advantage of such device is that the peripherals incorporated into device that much power consume by the chip, so one can save the overall power consumption of the system. The PSoC device can be configured by software tools called PSoC designer from the cypress semiconductor. The PSoC block incorporated with the software tools doesn't required external interfacing of the peripheral with the controller and one can reduce the PCB designing task which is required. The other advantage of using PSoC device into the system is one can lead the time to market of the designing of the system.

DSP Controller

The other choice of the microcomputer for the controller of the systems is the digital Signal Processor (DSP) processors. The architecture of microcontroller and DSP is differed in the sense that some digital processing algorithm implemented in the hardware architecture. The algorithm which is implemented in the hardware improves the lot in the speed of operation. The DSP The limitation of the microcontrollers for the real time processing like speech and image processing due to sequential execution of the instructions so, it can be overcome by the DSP processor. The various real-time algorithm is incorporated into the hardware of the DSP chip one can go for embedded core where systems require such faculties. Again, here also manufacturer offers cross compilers for the programming the DSP processor. Even some of the manufacturers offer the integrated solution of the microcontrollers and DSP processor results into digital signal controllers (DSC) [3].

FPGA/CPLDs

The advancement in VLSI technology also improves a lot in the design of the embedded systems. Earlier in this field the programmable logic arrays (PLAs) and programmable logic devices (PLDs) were available. Those programmable devices had limited capabilities to develop the digital logic. Now days VLSI technology offers complex programmable logic devices (CPLD) and field programmable logic devices (FPGA). These devices offer total system requirements into a single chip by configuring by hardware descriptive languages (HDLs). The program can be erased and reprogram again as per user requirements. Here, the programmable logic devices can be programmed by the electronic design automation (EDA) tool which offers the programming in the various HDL options. The CPLD/FPGA based embedded core offers dynamic change of algorithm and reprogrammed it many times. So, depending upon the requirement of wearable biomedical system one can select appropriate microcomputer in the system design. The improvement in the software EDA tools also support the configuration of such programmable devices in Graphical User Interface (GUI), so one can configured without knowing HDL language.

ARM Controllers

In the recent trends of architectural design of the controllers Advanced RISC Machines (ARM) plays an important role in speed of operation and power consumption. The concurrent execution of the instructions improves the speed of operation of the system. The improvement in the speed of execution of the instructions improves the real time processing of the signal. The real time processing required for speech processing and image processing application for medical

instrumentation. Texas instruments offer also the combination of ARM and DSP together in a single chip. By using such combination of the central controller one can optimized the performance of the embedded medical instrumentation.

Software Elements

Deriving from the processor independence of the MMT System a universal assembler had to be developed, this way - in syntactic sense - processor independence exists on the assembly level as well. Users' programs must be partitioned into tasks (task is the name of a program segment realizing a basic function of an instrument, e.g. measuring N data, regression calculation, processing data sent from the keyborad, etc). The one-processor real-time monitor assures the software link among taks and between tasks and handlers, i.e. the monitor supervises the run of tasks according to the desired measuring and data processing algorithms. A program library is an important part of MMT System software. The library contains both the handlers of the different hardware elements (D/A, A/D, timer, interface cards, etc), and typical algorithms for data processing.

System Design with the Help of Standard Element

The First step in the system design was to elaborate the users' guide of the instrument. There were two main tasks to be done: complete the list of parameters to be measured along with their computational algorithms and the accuracy needed by means of continuous consultations with medical experts, and secondly on the basis of the above-mentioned lists and after defining the other tasks of the microprocessor test, self-test, front panel servicing, etc. the probable exploitation of the processor had to be estimated. The most important goal was to determine which tasks can run serially and which must run parallel. While working on this, we took into consideration the simplest possible syntax for operating the instrument. In this phase of development, we had the greatest difficulties when discussing with non-technical experts i.e. medical experts, doctors. To find a common language - as usually - the technical experts had to gain knowledge from the anatomy of lungs to a certain extent, by our opinion it would support the development of microprocessor-based non-technical aimed devices if the experts of the given professional field had a minimum technical knowledge, at least enough to draw up their problems. Having completed the operating specifications of the instrument the hardware and software system design started. As the sampling frequency is relatively low (50Hz) the Z-80 processor card was chosen. The accuracy needed did not exclude the use of the A/D card. Only a few additional hardware (non-standardized) was needed, e.g. a double function tone-generator giving an error tone after incorrect operator action and giving an alarm signal in case of danger in a guarding monitor system [4], [5].

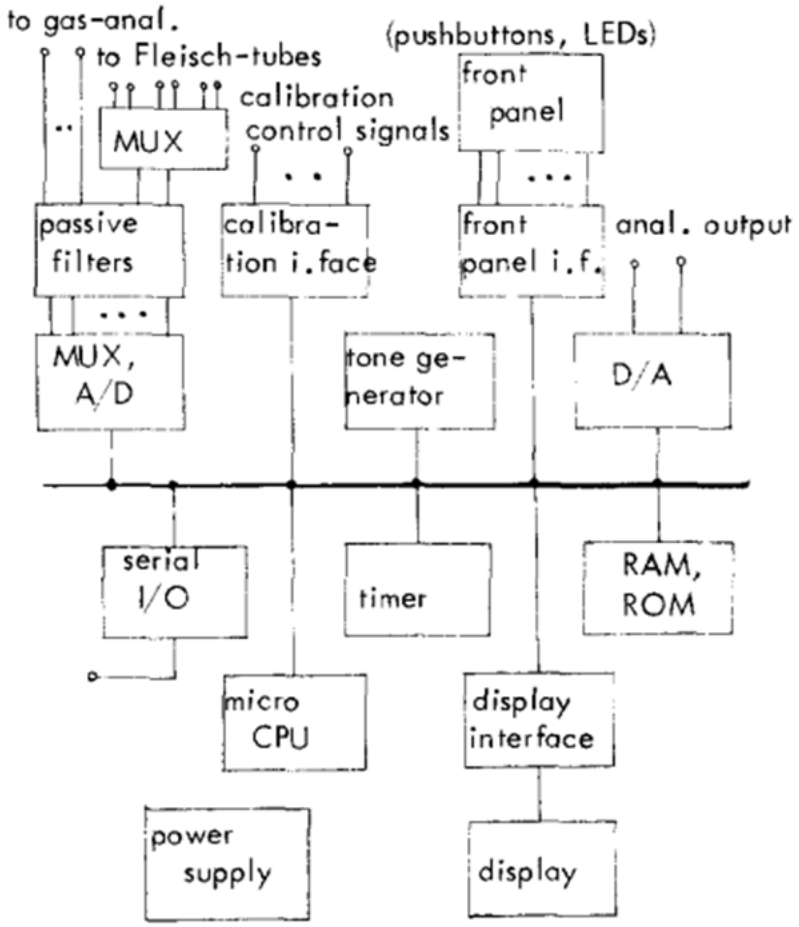


Figure 2: Represented the Microcontroller Hardware uses in Medical Field.

The hardware block diagram is given in Figure 2. The micro-CPU RAM, ROM, A/D, D/A, serial interface and the timer, display interface and front panel interface are parts of our laboratory system. The block diagram clearly shows that the whole hardware configuration is slightly specific to this instrument, the characteristic features can be found in its software. As for the software system design the starting points were partly the calculating algorithms of the parameters partly the classification of actions according to their relatively parallel or serial operation. On the basis of these findings tasks could have been defined. Action to be done were divided into two main groups: actions that must run parallel with sampling and actions that are independent in time from sampling. In the former case we minimized the number of tasks practically one long and sophisticated task serves for this purpose while in the latter case the main aspect was to partition the actions into the smallest reasonable units.

Instrument as a Product

As the instrument serves for medical purposes it must be very easily manageable. It can be operated via push-buttons. While defining the operator interface we made a one function/button distribution. The necessary data input patient data, BTPS correction data, measurement made selection and the handling of the instrument is helped by messages and instructions sent to the display. The built-in display simplifies the visualization of results as well as curves. There is a possibility to connect a slightly modified commercial TV set to the instrument for demonstration. For example parallel

with the examination the content of the built-in display can be seen on a remote TV set in a separate room. It can be used for educational purposes or for helping the doctor control the assistant's work without disturbing the patient. Naturally X-Y plotter and/or recorder and printer can be connected to the instrument for making hard-copy. Coming from the fact that on this field diagnosing of lungs no instrument has existed up to now with such specification and efficiency a certain feedback is to be expected: after the users will have gained enough experiences with this instrument they will be able to draw up their demands and recommendations. For the sake of easy modifications segmented programs must have been elaborated [6].

This need was taken into consideration when defining that above mentioned tasks. For the time being there are no generally widespread algorithms to calculate the "expected" values for the Hungarian population. It is a consequence of the lack of screening tests which could present the necessary data base for elaborating the algorithms needed. With this pulmonary testing instrument screening tests can be done easily and very fast compared to earlier devices, so on the basis of data gained by this new instrument the calculating algorithms of "expected" values can be elaborated that apply to the Hungarian population. It means that the algorithms presently built in (these are recommendations at the moment) must be easily changeable - it can be done by means of changing ROM chips. The pulmonary testing instrument serves for aeromechanical examination of lungs. Another important field for diagnosing lungs is body pletismography. The microprocessor-based version of the body-pletismograph is under development, the link between these instruments will be through the serial interface [7], [8].

DISCUSSION

In the current era, our lives have started to become more and more dependent on '**embedded systems**', digital information technology that is embedded in our environment. What are embedded systems anyway? Embedded systems are actually a combination of computer hardware and software which together form a component of an electrical device that we use in our daily life. In this article, we are going to discuss embedded systems medical applications. They are limited to a particular task and serves a particular function and they are a great help in real time system because they are very quick in performing their operations. Embedded system consists of a **microcontroller**, which contains a microprocessor, memory for storing data and programs, **AD converters,** DA converters which convert analog signals to digital signals and vice versa; sensors and actuators. **Embedded systems include a variety of applications** which not only safety-critical applications such as automotive devices and controls, railways, aircraft, aerospace and medical devices but also communications, 'mobile worlds' and 'e-worlds', the 'smart' home, clothes, factories etc. These applications have an enormous impact on our society, including security, privacy, and modes of working, living and health. More than 98% of processors applied today are in embedded systems, and are no longer visible to the customer as computers in the ordinary sense. New processors and methods of processing, sensors, actuators, communications, and infrastructures are 'enablers' for this very persistent computing. They are in a sense everywhere, that is, almost invisible to the user and almost omnipresent. As such, they form the basis for a significant economic push [9], [10].

CONCLUSION

In this paper we have discussed about various aspects of the designing a medical instrumentation system for health monitoring by using controller based intelligent systems. The evolution in the field of embedded technology improves the system performance day by day. In future we expect

a tremendous improvement in the wearable biomedical systems. The development in the technology improves the quality of human life day by day; however, the designed system must be safe to the human being and should be environment friendly. This care must be taken into consideration by the designer of the system.

REFERENCES

[1] C. C. Chen et al., "Design of bidirectional security system for intravenous drip infusion with hybrid communication," Sensors Mater., 2020, doi: 10.18494/SAM.2020.2684.

[2] G. M. Hobusch, K. Döring, R. Brånemark, and R. Windhager, "Advanced techniques in amputation surgery and prosthetic technology in the lower extremity," EFORT Open Rev., 2020, doi: 10.1302/2058-5241.5.190070.

[3] D. Strobel, D. Oswald, B. Richter, F. Schellenberg, and C. Paar, "Microcontrollers as (In)Security Devices for Pervasive Computing Applications," Proc. IEEE, 2014, doi: 10.1109/JPROC.2014.2325397.

[4] I. M. Shehabat and N. Al-Hussein, "Deploying internet of things in healthcare: Benefits, requirements, challenges and applications," J. Commun., 2018, doi: 10.12720/jcm.13.10.574-580.

[5] M. Santonico et al., "CO2 and O2 detection by electric field sensors," Sensors (Switzerland), 2020, doi: 10.3390/s20030668.

[6] B. Jenita Amali Rani and A. Umamakeswari, "Electroencephalogram-based Brain Controlled Robotic Wheelchair," Indian J. Sci. Technol., 2015, doi: 10.17485/ijst/2015/v8is9/65580.

[7] J. P. Killeen, T. C. Chan, C. Buono, W. G. Griswold, and L. A. Lenert, "A wireless first responder handheld device for rapid triage, patient assessment and documentation during mass casualty incidents.," AMIA Annu. Symp. Proc., 2006.

[8] L. A. Lenert, D. A. Palmer, T. C. Chan, and R. Rao, "An Intelligent 802.11 Triage Tag for medical response to disasters.," AMIA Annu. Symp. Proc., 2005.

[9] E. Zarepour, M. Hassan, C. T. Chou, and A. A. Adesina, "Energy-Harvesting Nanosensor Networks: Efficient event detection.," IEEE Nanotechnol. Mag., 2016, doi: 10.1109/MNANO.2016.2606682.

[10] D. W. Repperger, C. C. Ho, P. Aukuthota, C. A. Phillips, D. C. Johnson, and S. R. Collins, "Microprocessor based spatial tens (transcutaneous electric nerve stimulator) designed with waveform optimality for clinical evaluation in a pain study," Comput. Biol. Med., 1997, doi: 10.1016/S0010-4825(97)00022-X.

CHAPTER 19
AN OVERVIEW OF THE MICROPROCESSOR APPLICATIONS

Ramesh Chandra Tripathi, Professor
College of Computing Sciences and IT, Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India
Email Id- rctripathig@gmail.com

ABSTRACT:

In general computing to real-time monitoring systems, microprocessors may be used for a variety of information processing activities. The microprocessor makes it easier to utilize the large amount of information accessible online and offline, at home and at work, as well as new methods of communication. The majority of electronic products include a built-in microprocessor, such as computers, remote controls, washing machines, microwaves, mobile phones, iPods, and more. Personal computers, laptops, mobile phones, and sophisticated military and space systems all depend on microprocessors to function. This work illustrates how microprocessors are used in everyday life.

KEYWORD:

Communication, System, Electronics, Microprocessor, Medical Science.

INTRODUCTION

A microprocessor is usually a silicon chip that contains millions of transistors and other components that process millions of instructions per second integrated with memory chips and other special purpose chips, and directed by software. It is a multipurpose, programmable microchip that uses digital data as input and provides results as an output once it processes the input according to instructions stored in its memory. Microprocessors use sequential digital logic as they have internal memory and operate on numbers and symbols represented in the binary numeral system. They are designed to perform arithmetic and logic operations that make use of data on the chip. General purpose microprocessors in PCs are used for multimedia display, computation, text editing and communication. Several microprocessors are part of embedded systems. These embedded microprocessors provide digital control to several objects including appliances, automobiles, mobile phones and industrial process control. A microprocessor is also known as a central processing unit (CPU), which is a complete computing engine assembled on a single chip. It performs all the computational tasks, calculations and data processing of the computer. The most popular type of microprocessor is the Intel Pentium chip. A typical example is shown in Figure 1[1].

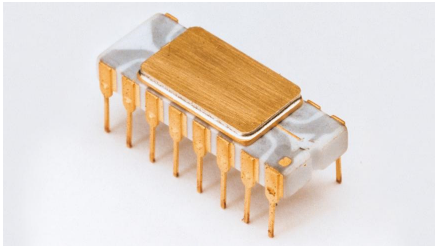


Figure 1: Represented the INTEL 4004 Microprocessor

The development of the first microprocessor began in 1969, when Intel engineer Marcian Edward "Ted" Hoff proposed to use a single-chip, general-purpose CPU to perform most computer programming functions. The result was the first microprocessor, the 4004, which was announced by Intel in 1971. This microprocessor evolved into a series of increasingly powerful Intel chips the 286, 386, 486, and in 1993, the Pentium--for the International Business Machines (IBM) Corp. personal computer (PC) and IBM-compatible PCs. Meanwhile, Motorola Corp. developed the 68000 series of chips for the Macintosh personal computer made by Apple Computer. Further advancements have led to robust microprocessors, such as the Intel core i7 microprocessor that is capable of rendering 3D images.

The function of the microprocessor is best described in a three step process fetching, processing, and decoding. In the fetching step, it gets an instruction from the computer's memory. In the decoding step, it decides what the instruction means. The last step is the processing itself, which involves the microprocessor's carrying out or performing the decoded set of instructions. A modern microprocessor can complete this three-step process millions of times in one second. Microprocessors may be classified by their hardware architecture. The two basic types of hardware are complex instruction set computer (CISC), and reduced instruction set computer (RISC). CISC processors can perform complex functions with one instruction while RISC chips usually need multiple instructions. The Intel Pentium and Atom chips are based on the CISC architecture, while PowerPC and ARM's Cortex chips are RISC systems [2].

The following are examples of microprocessor: AMD, ARM DEC, Elbrus, Fairchild Semiconductor, Freescale Semiconductor (Motorola), Hewlett-Packard, IBM, Intel, MIPS Technologies, National Semiconductors, NEC, NXP (Phillips), SPARC, Texas, VIA, Western Electronic, Zilog. Each of these microprocessors has their versions and kinds. There are two primary manufacturers of computer microprocessors. Intel and Advanced Micro Devices (AMD) lead the market in terms of speed and quality. Intel's desktop CPUs include Celeron, Pentium and Core. AMD's desktop processors include Sempron, Athlon and Phenom. Intel makes Celeron M, Pentium M and Core mobile processors for notebooks. AMD makes mobile versions of its Sempron and Athlon, as well as the Turion mobile processor which comes in Ultra and Dual-Core versions. Both companies make both single-core and multicore processors.

Each processor has a clock speed which is measured in gigahertz (GHz). It also has a front side bus which connects it with the system's random access memory (RAM.) CPUs typically have two or three levels of cache. Cache is a type of fast memory which serves as a buffer between RAM and the processor. The processor's socket type determines the motherboard type where it can be installed. The basic things to consider when choosing processors are size, front side bus (FSB) and cache. Whether one is buying a new computer or upgrading old one, one must get the fastest affordable processor. This is because the processor becomes obsolete very quickly.

The impact of microprocessor in different fields is significant. The availability of low cost, low power and small weight, computing capability makes it useful in different applications. Nowadays, microprocessor-based systems are used in instructions, automatic testing product, speed control of motors, traffic light control, light control of furnaces etc. This work presents the general application of microprocessor to different fields.

Microcontrollers

Microcontrollers are designed for industrial control applications, where ease of use and versatility rather than speed is the main requirements. They interface with sensors and other devices in applications ranging from on-board computers in cars to lighting systems and renewable energy control systems.

Input/output and memory functions are often embedded along with the core processing functions on one chip. In addition to Intel, Freescale, Micron and Texas Instruments are major manufacturers of microcontrollers. When the most modern technical engineering is applied to a microcontroller it allows the device to be extremely compact, making microcontrollers popular within mobile devices such as cell phones and PDAs [3].

To the layperson, microcontrollers and microprocessors may seem like very different devices; however, it is important to note that all microcontrollers contain microprocessors. The major difference between a microcontroller and a multifunctional PC microprocessor is the overall level of complexity. Microcontroller processors are designed to fill a smaller, more focused variety of roles while making use of less expensive and less complex circuitry. The main advantage of a microcontroller is that it allows electronic automation in situations where a full-sized computer is not needed. Microcontrollers integrate a microprocessor with peripheral devices for control of embedded system (computer system designed for specific control functions within a larger system, often with real-time computing constraints). Embedded systems range from portable devices such as digital watches, and MP3 players, to large stationary installations like traffic lights, and factory controllers.

Microcontrollers shine in situations where limited computing functions are required within an easily definable set of parameters. Microcontrollers excel at the low grade computational functions required to run devices such as electronic parking meters, vending machines, simple sensors and even home security equipment. Microcontrollers surround most Americans in their homes and offices, being present in devices such as televisions, remote controlled stereos and even the digital computer components of a timer on a newer stove [4].

In real-time computing systems, microprocessors are embedded in security devices such as the anti-lock braking system (ABS) that are widely used in modern automobiles. The microprocessor detects motions and changes, that are relative to the surrounding or environment of the security device and sends signals that correspond to the changes that it detected. Microcontrollers have innumerable applications. Some examples of their simple applications are in:

i. Biomedical instruments like an ECG LCD display cum recorder, blood cell recorder cum analyzer, patient monitor system,

ii. Communication systems like numeric pagers, cellular phones, cable TV terminals, FAX and transceivers with or without an accelerator, video game and so on,

iii. Peripheral controllers of a computer such as the keyboard controller, printer controller, laser printer controller, LAN controller and disk drive controller.

iv. Instruments such as an industrial process controller, and electronic smart weight display system,

v. A target tracker,

vi. An automatic signal tracker,

vii. Accurate control of the speed and position of a DC motor,

viii. A robotics system,

ix. A CNC machine controller

x. Automotive applications like a close loop engine control, a dynamic ride control, an anti-lock braking system monitor and so on,

xi. Electronic data acquisition and supervisory control system, the industrial moisture recorder cum controller, CRT display controller, digital storage system and spectrum analyzer.

The microcontrollers are classified in terms of internal bus width, embedded microcontroller, instruction set, memory architecture, IC chip or VLSI core (VHDL or Verilog) file and family. There are 8-bit, 16-bit and 32-bit microcontrollers.

General Purpose Processors (GPP)

Microprocessors commonly used in general computing tasks include those embedded in laptops or desktop computers. These microprocessors are responsible for the core computing processes, such as calculation and data transfer. General purpose processors are designed for personal computers, laptops, mobile devices and large central servers. Several companies make general purpose processors, including Intel, IBM and Motorola. Companies generally come out with faster and more complex chips every two to three years. Intel is the recognized industry leader in this space. From the earlier Pentium and Centrino microprocessors to the Core 2 and Atom chips for desktop and mobile computers and the high-end Itanium and Xeon processors for server applications, Intel is generally regarded as the company that sets the benchmark for others to follow. The PowerPC microprocessors were co-developed by Motorola, Apple and IBM originally for Apple's Macintosh computers, but Apple switched to Intel chips in 2006. Apple uses other processors as well. For example, the iPad 2, introduced in March 2011, uses the Cortex-A5 processor designed for mobile computing by computer manufacturer ARM.

Application-Specific Processors (ASP)

ASPs are specialized to perform one function well. There are three types of ASP. These are the digital-signal processor (DSP), the application-specific integrated circuit (ASIC), and the application-specific instruction set processor (ASIP). DSPs are used for fast numerical computation. ASICs have a particular algorithm implemented directly in their hardware. ASIPs are a halfway house between a GPP and an ASIC. ASIPs have some programmability [5].

Application of Microprocessors

Several items such as DVD players, cellular telephones, household appliances, car equipment, toys, light switches and dimmers, electrical circuit breakers, smoke alarms, battery packs, car keys, power tool and test instruments use microprocessors. Pollution control standards require automobile manufacturers to use microprocessor engine management systems for an optimal control of emissions over varying operating conditions. A typical microprocessor makes daily life easier because of its vast application in every field. The applications of microprocessors to life include but not limited to the following:

i. Household Devices

A complex home security system or the programmable thermostat neatly attached to the wall contains microprocessor technology. Technology-based home security system microprocessors assist with monitoring large or small properties. The simplest programmable thermostat allows the control of temperature in homes. Entering the preferred degree and achieving it on a consistent basis requires some intelligence on the part of the thermostat. A microprocessor in the system works with the temperature sensor to determine and adjust the temperature accordingly. Dishwashers, washing machines, high-end coffee makers and radio clocks contain microprocessor technology. Some home items that contain microprocessors include televisions, VCRs, DVD players, microwaves, toasters, ovens, stoves, clothes washers, stereo systems, home computers, hand-held game devices, thermostats, video game systems, alarm clocks, bread machines, dishwashers, home lighting systems and even some refrigerators with digital temperature control.

ii. Industrial Applications of Microprocessors

Some industrial items with microprocessors include: cars, boats, planes, trucks, heavy machinery, gasoline pumps, credit-card processing units, traffic control devices, elevators, computer servers, most high-tech medical devices, digital kiosks, security systems, surveillance systems and even some doors with automatic entry.

iii. Transportation Industry

Automobiles, trains and planes utilize microprocessor technology. The 1978 Cadillac Seville was the first consumer vehicle to use a microprocessor embedded in its digital display. The "Trip Computer" provided mileage and additional information on the current trip. Microprocessors work behind the scenes to keep commuters safe and timely. Consumer vehicles-cars, trucks, RVs-integrate microprocessors to communicate important information throughout the vehicle. For example, navigation systems provide information using microprocessors and global positioning system (GPS) technology. In addition, mass transportation systems like flights and trains rely on microprocessors for important information. Public transportation fare cards, or smartcards, contain processors to calculate fares, deduct the appropriate amount and retain information on how much funding remains. The aviation system relies heavily on microprocessors from calculating weather conditions to controlling the complex functions of an airplane [6].

iv. Computers and Electronics

The brain of the computer is microprocessor-drives technology. They are used in computer ranging from microcomputers to supercomputers. In addition, many electronic devices have central processing units (CPU) embedded. The CPU performs computer processing tasks by executing

software instructions relative to the data it contains. For example, a cell phone or mobile device executes game instructions by way of the microprocessor. While playing chess, the microprocessor holds data about the last action and executes software instructions for the next computer move. VCRs, televisions and gaming platforms also contain microprocessors for executing complex tasks and instructions.

v. Low-Power and Battery Management

The need for a processor in battery powered devices has spurred development of microcontrollers that draw little power, yet deliver the processing speed needed in small consumer devices. In some instances, the microcontrollers serve as battery management devices for monitoring the charging and discharging of batteries, such as lithium-ion cells, in portable electronics devices. Other low-power microcontrollers are designed to always be powered on and typically include an active mode for processing and a sleep mode for monitoring a signal while drawing a miniscule amount of current.

vi. In Medicals

Many medical devices, such as an insulin pump, are typically controlled by a microprocessor. The microprocessors perform various functions, such as processing data from bio-sensors, storing measurements, and analyzing results for these applications, designers often select a microprocessor with a rich instruction set and proven track record, ensuring reliable operation and maximizing the investment in code generation can be leveraged in the next generation medical products. The increasing use of microprocessors and associated software in both implanted and external medical devices poses special analytical challenges. Programmable pacemakers, long-term portable ECG recorders, and ECG arrhythmia detection monitors, for example, contain cardiac arrhythmia detection software. Examination of such devices is difficult, especially because failures are rare and transient and Figure 2 presents the state diagram of an Emergency Medical Services (EMS).

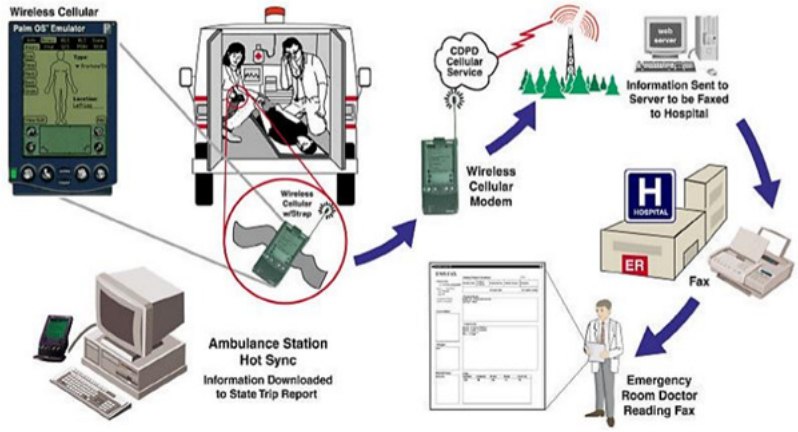


Figure 2: Represented the Diagram Showing Ems State Diagram

vii. Imaging Applications

Though some medical imaging applications remain tethered to the wall, ultrasound machines have benefited the most from the market's trends toward miniaturization and portability. Historically, ultrasound systems weighed hundreds of pounds and were large and expensive. In the past it was more practical to bring the bed-ridden patient, bed and all, to the ultrasound machine rather than

vice versa. Only in the case of the critically ill patients who could not be moved was the ultrasound system maneuvered, with difficulty, to the patient. Over time, portable ultrasound technologies emerged, but achieving image quality on par with the larger devices proved to be a challenge, as was achieving the battery life, high-power computing, and efficient memory access that these applications require. High-quality handheld systems enable routine bedside scanning. This has not only improved patient access to safe, noninvasive diagnostic medicine, but has reduced the time and costs associated with such diagnostics.

Flash-based, low-power and mixed-signal FPGAs can be utilized in data acquisition cards for filtering and data alignment, control cards, data consolidation cards for data buffering/FIFO and alignment as well as many systems management and control functions.

viii. Communication

Among the many types of peripheral, circuits that can be built into microcontrollers are communication interfaces, and in particular wireless interface circuits. Microcontrollers designed for communication applications include sections for handling communication protocols such as Wi-Fi, Bluetooth, ZigBee, CAN bus, infrared, USB and Ethernet. Communication microcontrollers can be found in wireless devices and in wired network devices such as those in automotive applications. In telephone industry, microprocessors are used in digital telephone sets, telephone exchanges and modem etc. The use of microprocessor in television, satellite communication has made teleconferencing possible. Railway reservation and air reservation system also uses this technology. LAN and WAN for communication of vertical information through computer network [7].

ix. Deep Cover Security Systems

The Deep Cover Secure Microcontroller (MAX32590) provides an interoperable, secure, and cost-effective solution to build new generations of trusted devices such as multimedia-enabled portable EFT-POS terminals. The MAX32590 integrates a Memory Management Unit (MMU), 32KB of instruction cache, 16KB of data cache, 4KB instruction TCM, 4KB data TCM, 384KB of system SRAM, 3KB of One-Time-Programmable (OTP) memory, 128KB of Boot ROM, and 24KB of battery-backed SRAM. The MAX32590 maximizes on-chip bandwidth when dealing with high-speed communication such as 100Mbps Ethernet, large color LCD displays, and gigabit-sized mass storage devices. DeepCover embedded security solutions cloak sensitive data under multiple layers of advanced physical security to provide the most secure key storage possible. In addition to hardware crypto functions, the MAX32590 provides a true random number generator, battery-backed RTC, nonvolatile SRAM and real-time environmental and tamper detection circuitry to facilitate system-level security for the application.

x. Automatic Process Control

A process is a series of actions or tasks that are carried out according to the instructions or program or plan so that a desired result is obtained. For example, raw material is treated by a series of processes so as to get the final product.

The solenoid valves open or close to control flow, and the heating elements go on and off to reach a final temperature in a sequence; the motors control and timers are also used in controlling the process. Process control means a control system in which the states and operations are defined as a function of time and there is a preset sequence in the program. A microcontroller has input ports

to receive the bits for the physical parameters, the timer to interrupt at set intervals and inputs. The MCU instructions check the needed actions. The output ports send the control bits to various points for the actions. Microprocessor-based controllers are available in appliances, such as microwave oven, washing machine etc microprocessors are being used in controlling various parameters like speed, pressure, temperature etc. These are used with the help of suitable transduction.

xi. Embedded systems at home

A vast number of modern devices in the home are microprocessor controlled. Examples include: washing machines; camera; calculators; hi-fi systems; telephones; microwave ovens; burglar alarms etc. The input are usually sensors, buttons or simple numeric keyboards while the output include simple LCD screens displays, motors and relays, LEDs, lights, buzzers etc.

xii. Office Automation and Publication

Microprocessor-based systems are used extensively in the workplace, often to automate and monitor production in some way. Examples include: electronic tills, automatic car washes, security systems, FAX and telephone systems, automated production lines, automated warehousing, manufacturing robots etc. Microprocessor based microcomputer with software packages has changed the office environment. Microprocessor-based systems are being used for word processing, spread sheet operations, storage etc. The microprocessor has revolutionized the publication technology.

xiii. Instrumentation

Microprocessor is very useful in the field of instrumentation. Frequency counters, function generators, frequency synthesizers, spectrum analyses and many other instruments are available, when microprocessors are used as controller. It is also used in medical instrumentation.

xiv. Entertainment

The use of microprocessor in toys, entertainment equipment and home applications is making them more entertaining and full of features. The use of microprocessors is more widespread and popular.

DISCUSSION

Most of the diseases and medical complications happen due to our lifestyles and embedding systems can go a long way in helping us to shape our daily habits and overhaul our lifestyle since nowadays people don't really have time to go to doctors for their regular checkups. So smart embedded technology based medical devices, which are mostly in the form of wearables, were introduced to provide aid to people who now can be more aware of their personal health as these devices would allow users to monitor their heart rate, blood pressure, glucose, weight and numerous other parameters. One such product which has taken the fitness world by storm is Fitbit. It is one of the most advanced examples of embedded systems. It can be worn on hand as a bracelet and it tracks all our health measures such as blood pressure, weight, sleep etc., and help us reach our health and fitness goals. Such users are less likely to develop health problems in the future. With the decrement in size and increment in processing power, small devices with the embedded system are capable of collecting patient data and making control decisions that may help in providing patients with better treatments and medications. In short, this system is replacing the need for having doctor to visit the patient and examine his symptoms. This has been a great advancement in the medical field in which not only the quality of the healthcare is augmented but

the cost of medical management is also diminished with these programmed automatic technologies introduced. Nevertheless, embedded systems have innumerable applications in every field of life including health and medicine. With the boom of Artificial Intelligence and **IoT(Internet of things)** around the corner, the eventual fate of embedded systems couldn't be brighter and we can only expect more applications of embedded systems in effect making healthcare more reachable and easier than ever in human history [8]–[10].

CONCLUSION

Microprocessor has and will continue to revolutionize our world. Thousands of items that were traditionally not computer-related now include microprocessors. These include large and small household appliances, cars and their accessories, car keys, tools and test instruments, toys, light switches/dimmers and electrical circuit breakers, smoke alarms, battery packs, and hi-fi audio/visual components. Such products as cellular telephones, DVD video system and HDTV broadcast systems fundamentally require consumer devices with powerful, low-cost, microprocessors. Increasingly stringent pollution control standards effectively require automobile manufacturers to use microprocessor engine management systems, to allow optimal control of emissions over widely varying operating conditions of an automobile. Non-programmable controls would require complex, bulky, or costly implementation to achieve the results possible with a microprocessor. A microprocessor control program can be easily tailored to different needs of a product line, allowing upgrades in performance with minimal redesign of the product. Different features can be implemented in different models of a product line at negligible production cost. Microprocessor control of a system can provide control strategies that would be impractical to implement using electromechanical controls or purpose-built electronic controls.

REFERENCES

[1] A. J. Nichols, "An Overview of Microprocessor Applications," Proc. IEEE, 1976, doi: 10.1109/PROC.1976.10247.

[2] A. J. Welsh, C. Delgado, C. Lee-Trimble, A. Kaboudian, and F. H. Fenton, "Simulating waves, chaos and synchronization with a microcontroller," Chaos, 2019, doi: 10.1063/1.5094351.

[3] R. Freer and A. V. Powell, "Realising the potential of thermoelectric technology: A Roadmap," Journal of Materials Chemistry C. 2020. doi: 10.1039/c9tc05710b.

[4] M. Specialties, "Piezo Film Sensors Technical Manual," Measurement, 2006.

[5] W. Kunikowski, E. Czerwiński, P. Olejnik, and J. Awrejcewicz, "An Overview of ATmega AVR Microcontrollers Used in Scientific Research and Industrial Applications," Pomiary Autom. Robot., 2015, doi: 10.14313/par_215/15.

[6] S. Samsugi, A. I. Yusuf, and F. Trisnawati, "SISTEM PENGAMAN PINTU OTOMATIS DENGAN MIKROKONTROLER ARDUINO DAN MODULE RF REMOTE," J. Ilm. Mhs. Kendali dan List., 2020, doi: 10.33365/jimel.v1i1.188.

[7] K. D. Thoben, S. A. Wiesner, and T. Wuest, "'Industrie 4.0' and smart manufacturing-a review of research issues and application examples," International Journal of Automation Technology. 2017. doi: 10.20965/ijat.2017.p0004.

[8] S. Dutta, "An overview on the evolution and adoption of deep learning applications used in the industry," Wiley Interdiscip. Rev. Data Min. Knowl. Discov., 2018, doi: 10.1002/widm.1257.

[9] A. Moshovos and G. S. Sohi, "Microarchitectural innovations: Boosting microprocessor performance beyond semiconductor technology scaling," Proc. IEEE, 2001, doi: 10.1109/5.964438.

[10] J. Watada, A. Roy, R. Kadikar, H. Pham, and B. Xu, "Emerging Trends, Techniques and Open Issues of Containerization: A Review," IEEE Access. 2019. doi: 10.1109/ACCESS.2019.2945930.

CHAPTER 20

AN ANALYSIS OF AUTOMATED LIFE SAVING DEVICE BASED ON A MICROCONTROLLER

Ranjana Sharma, Associate Professor
College of Computing Sciences and IT, Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, India
Email Id- sharmaranjana04@gmail.com

ABSTRACT:

With the course of progress in the field of medicine, most of the patients' lives can be saved. Only thing required is the proper attention in proper time. Our wearable solution tries to solve this issue by taking the patients vitals and transmitting them to the server for live monitoring using mobile app along with patient's current location. In case of an emergency, that is if any vitals show any abnormalities, a SMS is sent to the caregiver of the patient with the patient's location so that he can reach there on time.

KEYWORDS:

Communication, Electronics, Microprocessor, I.O Devices, 8085 Interrupts.

INTRODUCTION

The 21st century has witnessed a major paradigm shift in diagnosis technology in the field of medicine. Embedded technology has been effectively used for successful diagnosis, leading to more accurate treatments. High speed calculations, more complex testing methodologies, more accurate and reliable results have forever enhanced diagnosis process. Today's world is becoming increasingly dependent on heuristic data; medical treatment is no exception. Large volume of data is generated about patients, symptoms and diagnosis. In the previous decade, doctors would have recorded certain basic information about the patients in a prescription. In contrast, these days he will have a long stream of digital data, from minute details about the patient to high-definition medical images. Since the world is going green, this data is needs to be made available in digital format. This in turn has broadened the application of informatics to improve health care and has also contributed in medical research.

Today, informatics is being applied at every stage of health care from basic research to care delivery and includes many specializations such as bioinformatics, medical informatics, and biomedical informatics, and is often referred to as "in silico" research. Informatics specializes on introducing better means of using technology to process information. The medical field has developed sufficiently to ensure critical patients in majority of the cases, can survive cardiac arrest or similar issues. The person can return to normalcy, with proper diagnosis and medication. But the treatment has to be given at the right time, which is during the initial stage of an attack. This becomes difficult to achieve when the person is alone, or when the patient is a senior citizen. It tries solves the problem by being an automated companion who watches the patient's body vitals and calls for help when in danger. The gadget monitors the body vitals and updates the data in a

database. Caretaker of the patient can live monitor this information. In case, if the vitals are in the danger zone, an alert message is sent to the caretaker. So, the patient receives proper treatment on time [1].

The electronic gadget under consideration consists of a wearable device and a receiving kit. The wearable device is supposed to detect and get the body vitals of the person wearing it, post which it sends the same to the receiving kit. The wearable band has sensors to serve this purpose and an emergency button in case of emergency. The receiving kit has three tier functionalities. The first to track the current location of the person wearing it, second, it sends the collected information to a centralized server for live monitoring and also for future reference and lastly, to check whether the received vitals are within the normal range or not. If not, then it sends a SMS to the pre-configured number with the current location along with the current vitals for immediate assistance. Another scenario could be, if the patient feels any discomfiture, feels dizzy that can never be detected by any sensor. In that case, the patient presses the emergency button, then also a similar SMS will be sent to the pre fed mobile number stating the person requires attention. This being an embedded system, it has related software modules. There is a central server, where the current vitals & location are sent from the receiving kit. Server also stores daily data for future reference, from which a caregiver can get an idea about how the patient's health was in past 6 months. Again, there is a Multiplatform mobile App. The app fetches the latest vitals and location from the server and shows it on screen, so that caregiver can check anytime how his loved one is doing and where he is currently.

ATmega328

ATmega328 is an 8-bit AVR RISC-based microcontroller by Atmel which is used in the wearable gadget, to collect sensor information, then encrypt it and send it to the receiving kit.

Arduino Uno

Arduino Uno is a versatile microcontroller board based on the ATmega328 which is used in the receiving kit to receive sensor data and location from the wearable gadget and then it will send the data to the server. If required, this will also help in sending SMS. This can be replaced with just another ATmega328 microcontroller instead of this complete board.

LM35 Temperature Sensor

LM35 series are precision integrated-circuit temperature devices which gives an output voltage linearly proportional to the Centigrade temperature it receives. This LM35 Sensor is used to detect body temperature in the prototype.

2.4 M212 Pulse Sensor

Pulse Sensor which is used in this implementation is a photo plethysmograph, which is a well-known medical device used for non-invasive heart rate monitoring. Sometimes, it measures blood oxygen levels (SpO2), sometimes they don't. The heart pulse signal that comes out of a photoplethys-mograph is an analog fluctuation in voltage, and it has a predictable wave shape which is shown in Figure 1.

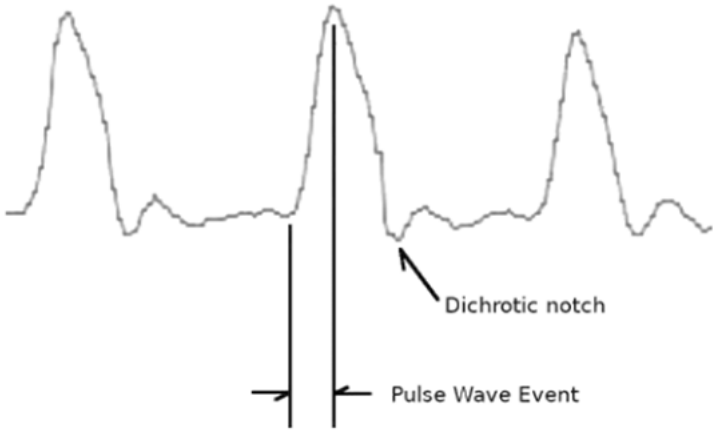


Figure 1: Represented that the Pulse Wave.

The depiction of the pulse wave is called a photoplethys-mogram, or PPG. The goal is to find successive moments of instantaneous heart beat and measure the time between them, called the Inter Beat Interval (IBI), which can be archived by following the predictable shape and pattern of the PPG wave. In this implementation, it is used to measure the pulse in the wearable gadget and send it to the AT mega328 for further processing.

434MHz Radio Frequency Receiver (Rx) & Transmitter (Tx)

RF transmitter and receiver module is used to transmit and receive the radio frequency for the wireless communication which is used to send sensor information from the wearable gadget to the receiving kit.

NEO6MV2 GPS Module

The NEO-6 module series is a family of GPS receivers which consist of high-performance u-blox 6 positioning engine, giving it receiver's excellent navigation performance even in the most challenging environments. This is used to triangulate location of the user and send it to the Arduino for further processing as required.

SIM300 GSM Module

A GSM Modem is a device that modulates and demodulates the GSM signals and in this particular case 2G signals. The modem which is used in this implementation is a SIMCOM SIM300 which is a Tri-band GSM/GPRS Modem. The TTL interface present in it allows to be directly interfaced with the Arduino present in the receiving kit, using which, the device is connecting to the designated web service and sending sensor plus location information. Also this GSM Module is used to send SMS when required.

AES 256bit Encryption

The Advanced Encryption Standard (AES), also known as Rijndael, is a specification for the encryption of electronic data. In this implementation AES with 256 bits Key Size is used for encrypting data while sending from wearable gadget to the receiving kit and from receiving kit to the server.

Microsoft Azure Virtual Machine

Microsoft Azure is a cloud computing platform and infrastructure created by Microsoft for building, deploying, and managing applications and services through a global network of Microsoft-managed data centers. This is used to host the Web Service and al- so to host Microsoft SQL Server in which the data received from receiving kit is stored for further use. Mobile App can access this to show the live information and also can be used to track patient history [2].

Working Mechanism

Wearable Gadget

First module is a wearable gadget in the form of a wrist band. This wearable gadget consists of many bio sensors. For this implementation, a LM-35 is used as the temperature sensor. For sensing the pulse, M212 heart rate sensor is used. Both the sensors are placed on the inner side of the band, placed strategically so that it directly touches the skin. Pulse sensor should be placed in such a way that it can detect the pulse. Sensors are connected to a circuit board, where ATmega328 Microcontroller is placed. Data collected by the sensors are periodically sent to the microcontroller. A string is constructed by concatenating the sensor values. If the user feels unwell, that can't be detected by any sensor. For that, there is an Emergency Button. If the button is pressed then the value of the switch is one, or else zero. This value is also concatenated with the sensor values. For example, if the switched is pressed along with that, current body temperature is 98OF and the current pulse rate is 77 bpm, then the string will be constructed as 1,98,77 This string is now transmitted from the wrist band to the second module, which is the receiving kit, explained later. In this implementation, this transmission is done using 434 MHz RF-Transmitter. Here a problem may creep up, if two or more radio device with the same frequency is used in close range, it may interfere with each other.

Receiving Kit

Second module of this implementation is the receiving kit. In the receiving kit a 434Mhz Receiver is connected to the Arduino, which receives the encrypted string from the wrist band and sends it to the Arduino Uno microcontroller board, which first decrypts it using the same AES scheme with the device ID as the key. This device ID is unique for each Wrist Band & Receiving Kit combo. So, the receiving kit has the same key which the wrist band used to encrypt the string. So, in this prototype MS001 is used for decryption and the plain text SXCMS: 1, 98, 77 is obtained. First, the string is validated to check whether it is coming from the correct device. If the string is starting with SXCMS then it is understood that it is coming from the correct device. After this, the string is broken to extract the parameter values of switch, hear rate and temperature. In this implementation, NEO6MV2 GPS module is used for triangulating the location of the device. Utmost priority is given to the switch. If the switch value is 1, irrespective of the sensor values, a SMS is sent using the SIM300 module present, to the care giver stating that person wearing this device needs his attention along with the exact location and sensor information. Apart from this, using the same SIM300 module, a web service is called, which is hosted in the Microsoft Azure server, by sending a simple HTTP Get Request. While sending the request, the device ID is also sent to identify the patient uniquely. So, the plain text that is going to be sent via the get request is MS001,98,77, Park Street where MS001 is the device ID, 98OF is the temperature, 77 is the pulse rate and Park Street is the triangulated location. This is sent to the server at a regular interval, say every minute. Besides sending this information to the server, it is also checked if there is any

anomalies in the pulse rate or temperature. Say, if the temperature goes beyond 99OF or pulse rate goes below 40 bpm, then an SOS message is sent to the care giver with the current sensor values plus the triangulated location stating that the person wearing this device may require medical help immediately [3].

Software Modules

An embedded system is incomplete without software modules First soft-module is the web service which receives the cipher text from each of the Medicare device, and decrypts it using the global key MedS. From the decrypted text, the device is first identified from the device ID. Consequently, the individual sensor data are extracted, and then lastly the location. Furthermore, this information is updated in the database, in the current Patient Data table. In every six hours, this information is stored in patient History table as well. Both web service and database are stored in the Microsoft Azure virtual machine, so as to access it centrally[4].

Other Live Application as Life Savior

This chapter covers three aspects of safety protection against drowning, fires, and navigational error. The common thread amongst all this equipment is that these are primarily tools to be used in case of emergencies, where we talked about the safety equipment necessary for everyday use on deck as well. So here's a glance at 11 personal safety devices that are required on ships to safeguard mariners, adventurists or even individuals at sea.

Lifeboats

Lifeboats are the most basic and mandated equipment on board. They are further classified into free-fall boats, partially-covered lifeboats and totally-covered lifeboats. Abiding by the SOLAS convention, each vessel must have enough lifeboats to secure 1.5 times the number of people on board.

Totally covered lifeboat is used most commonly. They are watertight, with access through hatches that can be opened from both sides. The advantage is that they safeguard occupants from extreme temperatures while allowing people to navigate from within.

Davit Systems

Davit systems are used for hoisting, lifting, and storing lifeboats so that they stay secure at all times but can be easily removed when necessary. There are various types of davits, including slide-on davits, sling davits, winch-on davits, lift-up davits, etc. along with assorted accessories such as hoists, cranes, hooks, and more.

All boats and ships should have well-maintained davit systems, inspected regularly from time to time, to ensure that the lifeboat and liferaft stowage/release systems are operational at times of emergencies.

Rescue Boats

Rescue boats are used to rescue people from drowning, near the shore, or in the deep sea. Rescue boats are rigid, inflatable or hybrid structure with a minimum length of 3.8 m. They can capacitate 6 people: 5 seated and 1 lying down.

The design of rescue boats allows high power capacity and hence ensures a faster pace than other conventional boats. They are also equipped with all the material required to provide first-aid to the person in distress.

Technically, rescue boats are expected to have a consistent speed of up to 6 knots for 4 hours straight and be able to tow life rafts and lifeboats of a ship, even when filled to complete capacity.

Line-throwing Devices

Line-throwing appliances are used to project from the boat to the person overboard or from one boat to another and pull the object in distress to safety. It is propelled by an internal striker mechanism and a rocket and needs to be capable of projecting the line with reasonable accuracy. Per the SOLAS regulations, every ship has to carry at least 4 line-throwing devices at all times [5].

GMDSS equipment

The Global Maritime Distress and Safety System (GMDSS) is an internationally established set of safety procedures, types of equipment, and communication protocols used to increase safety and make it easier to rescue distressed ships, boats and aircraft. Adding to that list, some other handy tools are Automatic Identification Systems and Personal Locators that ensure personal safety [6]. Their GPS-based functioning allows the devices to convey the precise location of the wearer to the control console, which helps locate the person that much more quickly.

Intelligent Fire Alarm Systems

Intelligent fire alarm systems include appliances such as fire detectors, smoke detectors, etc. They are high-performing devices that allow for fast detection and management of fires. The mechanism uses a series of control and relay modules and probes and provides an advanced warning to protect lives. The low cabling cost is an added advantage.

Portable Fire Extinguishers

Portable fire extinguishers are invaluable tools to protect seafarers in case a fire breaks out on the ocean. They are further classified based on the type of extinguishing material used in the fire extinguisher as well as the different causes of fires. As a result, it is imperative to choose the right type of fire extinguisher, as per the aforementioned conditions [7].

Thermal Suits

Thermal suits are waterproof suits that protect the wearer from hypothermia from immersion in cold water, after abandoning a sinking or capsized vessel, especially in the open ocean. They are designed to have a conductivity of less than 0.25 W/mK and are used to conserve body heat in extreme temperatures up to -30 degrees Celsius. Thermal protective aids generally cover the entire body of the wearer, except for the face hence providing maximum guarding against the external temperatures [3].

Gas Detectors and Spares

Gas detectors are highly useful in finding the presence of toxic and combustible gases on ships. These are especially popular in mining, oil and gas, chemical, and industrial sectors, where there is a high possibility of noxious leaks. These atmosphere testing instruments are placed at the entry to enclosed spaces and map the concentration levels of the various gases in the room.

Ladders

Ladders are a means of getting on or off ships safely, along the ships' side whenever necessary. Various types of ladders can be used to get on or off a ship, depending on the urgency and location. Of all various types, pilot ladders are the most frequently used, though they have a high rate of associated injuries. Embarkation ladders are to be used when the ship/boat has to be abandoned during adverse conditions and need to be SOLAS compliant to ensure guarded safety [8].

DISCUSSION

The Authors have implemented this concept by creating a minimalistic prototype consisting most of the components except the GPS Module, the outputs observed were close to the desired ones, but can be fine-tuned further. It was detecting heart-beat accurately, for body temperature, other sensors would give better results. This prototype is quite bulky, and not very easy to carry. But this can be made quite compact by using latest industry-based fabrication techniques. For more accurate results, some of the components can be replaced with some superior ones. Like custom made body temperature sensor based on DS1624, DS18B20 could be better for precision while sensing the body temperature. For the transmission of sensor values from the wrist band to the receiving kit, other transmission modules such as NRF24L01 or Zigbee can also be used to improve the transmission. For the GPRS Based communication, it would be better to use SIM900 over SIM300. Security is a major concern these days. While transmitting data from the receiving kit to the server, in this implementation, AES is used with a global key which is not that secure, as if someone gets hold of it, he can access such vital confidential information including locations. For that, any other means of security can be provided such as RSA. This implementation is very scalable as more sensors can be added to this for collecting other vitals from the patient [9], [10].

CONCLUSION

This device is not a medical device to help in treatment but it provides the caregiver to monitor the health condition and also to get alert when in need. Location tracking gives this solution that very required edge so that when a person is in trouble, caregiver can also know where that person is.

REFERENCES

[1] S. U. Bhandari, S. Subbaraman, and S. Pujari, "Power reduction in embedded system on FPGA using on the fly partial reconfiguration," in Proceedings - 2010 International Symposium on Electronic System Design, ISED 2010, 2010. doi: 10.1109/ISED.2010.23.

[2] S. Gao, M. Jia, and F. Cao, "The design of small scaled solar battery charge and discharge control system," in 2011 International Conference on Electric Information and Control Engineering, ICEICE 2011 - Proceedings, 2011. doi: 10.1109/ICEICE.2011.5777394.

[3] S. George Fernandez, R. Palanisamy, and K. Vijayakumar, "GPS & GSM based accident detection and auto intimation," Indones. J. Electr. Eng. Comput. Sci., 2018, doi: 10.11591/ijeecs.v11.i1.pp356-361.

[4] M. Zeyad, P. Biswas, M. Z. Iqbal, S. Ghosh, and P. Biswas, "Designing of Microcontroller Based Home Appliances Governor Circuits," Int. J. Comput. Electr. Eng., 2018, doi: 10.17706/ijcee.2018.10.2.94-105.

[5] J. Gaudet and G. Handrigan, "Assessing the validity and reliability of a low-cost microcontroller-based load cell amplifier for measuring lower limb and upper limb muscular force," Sensors (Switzerland), 2020, doi: 10.3390/s20174999.

[6] M. Al-Janabi, Z. Faris, and A. Taqi, "Smart Farm Management System Based on Sensors Network," Cienc. e Tec., 2018.

[7] Texas instruments, "MSP430F552x , MSP430F551x Mixed-Signal Microcontrollers," Man. usuario, 2020.

[8] H. Ghasemzadeh and R. Jafari, "Ultra low-power hardware-assisted signal screening in wearable systems," in Smart Sensors, Measurement and Instrumentation, 2015. doi: 10.1007/978-3-319-18191-2_4.

[9] S. El Ferik, C. B. Ahmed, L. Ben Amor, and S. A. Hussain, "A microcontroller-based soft-starter for residential air-conditioners: Harmonic analysis," COMPEL - Int. J. Comput. Math. Electr. Electron. Eng., 2008, doi: 10.1108/03321640810890780.

[10] F. S. Perilla, G. R. Villanueva, N. M. Cacanindin, and T. D. Palaoag, "Fire safety and alert system using arduino sensors with IoT integration," in ACM International Conference Proceeding Series, 2018. doi: 10.1145/3185089.3185121.