Estd. 2000

L1

LECTURE HANDOUTS

ECE

II/IV

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : I – 8051MICROCONTROLLER

Date of Lecture:

Topic of Lecture: Architecture of 8051

Introduction :

A microcontroller is essentially a small computer on a chip. Like any computer, it has memory, and can be programmed to do calculations, receive input, and generate output. Unlike a PC, it incorporates memory, a CPU, peripherals and I/O interfaces into a single chip.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Digital circuits

Detailed content of the Lecture:

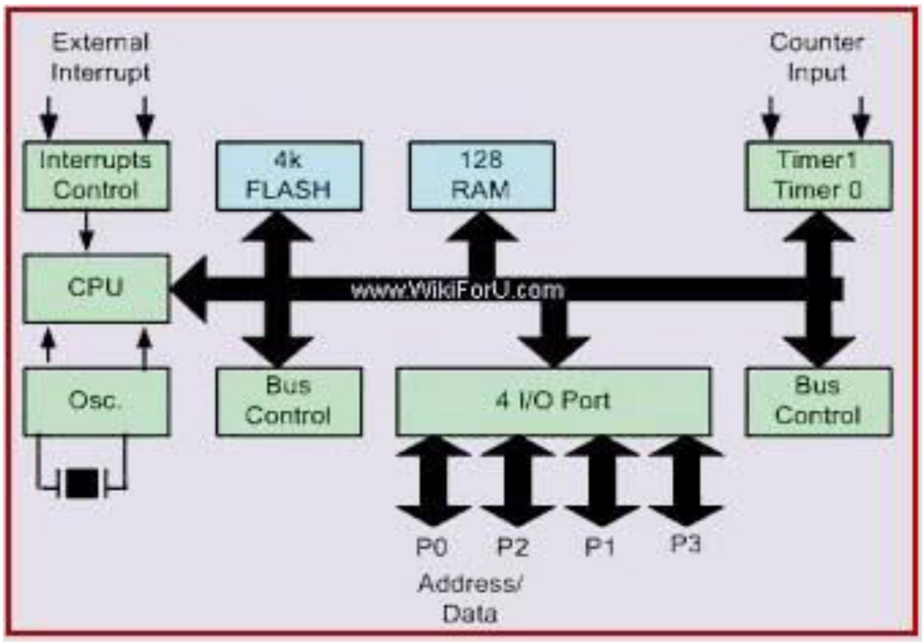
The 8051 Microcontroller is one of the basic type of microcontroller, designed by Intel in 1980's. This microcontroller was based on Harvard Architecture and developed primarily for use in embedded systems technology. Normally, this microcontroller was developed using NMOS technology, which requires more power to operate. Therefore, Intel redesigned Microcontroller 8051 using CMOS technology and their updated versions came with a letter C in their name, for instance an 80C51 it is an 8 bit microcontroller. These latest Microcontrollers requires less power to operate as compared to their previous versions. The 8051 Microcontroller has two buses and two memory spaces of 64K X 8 size for program and data units. It has an 8 bit processing unit and 8 bit accumulator units.

Central Processor Unit (CPU)

As we know that the CPU is the brain of any processing device of the microcontroller. It monitors and controls all operations that are performed on the Microcontroller units. The User has no control over the work of the CPU directly . It reads program written in ROM memory and executes them and do the expected task of that application.

Interrupts

As its name suggests, Interrupt is a subroutine call that interrupts of the microcontrollers main operations or work and causes it to execute any other program, which is more important at the time of operation. The feature of Interrupt is very useful as it helps in case of emergency operations. An Interrupts gives us a mechanism to put on hold the ongoing operations, execute a subroutine and then again resumes to another type of operations. The Microcontroller 8051 can be configured in such a way that it temporarily terminates or pause the main program at the occurrence of interrupts. When a subroutine is completed, Then the execution of main program starts. Generally five interrupt sources are there in 8051 Microcontroller.



***M*emory**

Microcontroller requires a program which is a collection of instructions. This program tells microcontroller to do specific tasks. These programs require a memory on which these can be saved and read by Microcontroller to perform specific operations of a particular task. The memory which is used to store the program_of_the microcontroller is known as code memory or Program memory of applications. It is known as ROM memory of microcontroller also requires a memory to store data or operands temporarily of the micro controller. The data memory of the 8051 is used to store data temporarily for operation is known RAM memory. 8051 microcontroller has 4K of code memory or program memory,That has 4KB ROM and also 128 bytes of data memory of RAM.

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=O-UALuVyyLY

Important Books/Journals for further learning including the page nos.:

Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (4-10)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L2

II/IV

ECE

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : I – 8051 MICROCONTROLLER **Date of Lecture:**

Topic of Lecture: Architecture of 8051 Microprocessor

Introduction :

A microcontroller is essentially a small computer on a chip. Like any computer, it has memory, and can be programmed to do calculations, receive input, and generate output. Unlike a PC, it incorporates memory, a CPU, peripherals and I/O interfaces into a single chip.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Processors and controllers
- Central processing unit
- BUS

Detailed content of the Lecture:

Basically Bus is a collection of wires which work as a communication channel or medium for transfer of Data. These buses consists of 8, 16 or more wires of the microcontroller. Thus, these can carry 8 bits,16 bits simultaneously. Hire two types of buses that are shown in below

- Address Bus
- Data Bus

Address Bus

Microcontroller 8051 has a 16 bit address bus for transferring the data. It is used to address memory locations and to transfer the address from CPU to Memory of the microcontroller.

Data Bus

Microcontroller 8051 has 8 bits of the data bus, which is used to carry data of particular applications.

Oscillator

Generally, we know that the microcontroller is a device, therefore it requires clock pulses for its operation of microcontroller applications. For this purpose, microcontroller 8051 has an on-chip oscillator which works as a clock source for Central Processing Unit of the microcontroller. The output pulses of oscillator are stable. Therefore, it enables synchronized work of all parts of the 8051 Microcontroller.

Input/Output Port

Normally microcontroller is used in embedded systems to control the operation of machines in the microcontroller. Therefore, to connect it to other machines, devices or peripherals we require I/O interfacing ports in the microcontroller interface. For this purpose microcontroller 8051 has 4 input, output ports to connect it to the other peripherals.

Timers/Counters

8051 microcontroller has two 16 bit timers and counters. These counters are again divided into a 8 bit register. The timers are used for measurement of intervals to determine the pulse width of pulses.

Applications of 8051 Microcontroller

Some of the applications of 8051 is mainly used in daily life & industrial applications also some of that applications are shown below

- Light sensing and controlling devices
- Temperature sensing and controlling devices
- Fire detections and safety devices
- Automobile applications
- Defense applications

Some industrial applications of micro controller and its applications

- Industrial instrumentation devices
- Process control devices

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=O-UALuVyyLY

Important Books/Journals for further learning including the page nos.:

- Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (24-30)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

L3

LECTURE HANDOUTS

ECE **II/IV**

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : I – 8051MICROCONTROLLER **Date of Lecture:**

Topic of Lecture: Special Function Registers(SFRs)

Introduction :

Each bit is wired across to one or other of the microcontroller peripherals. Each is then used either to set up the operating mode of the peripheral or to transfer data between the peripheral and the microcontroller core.

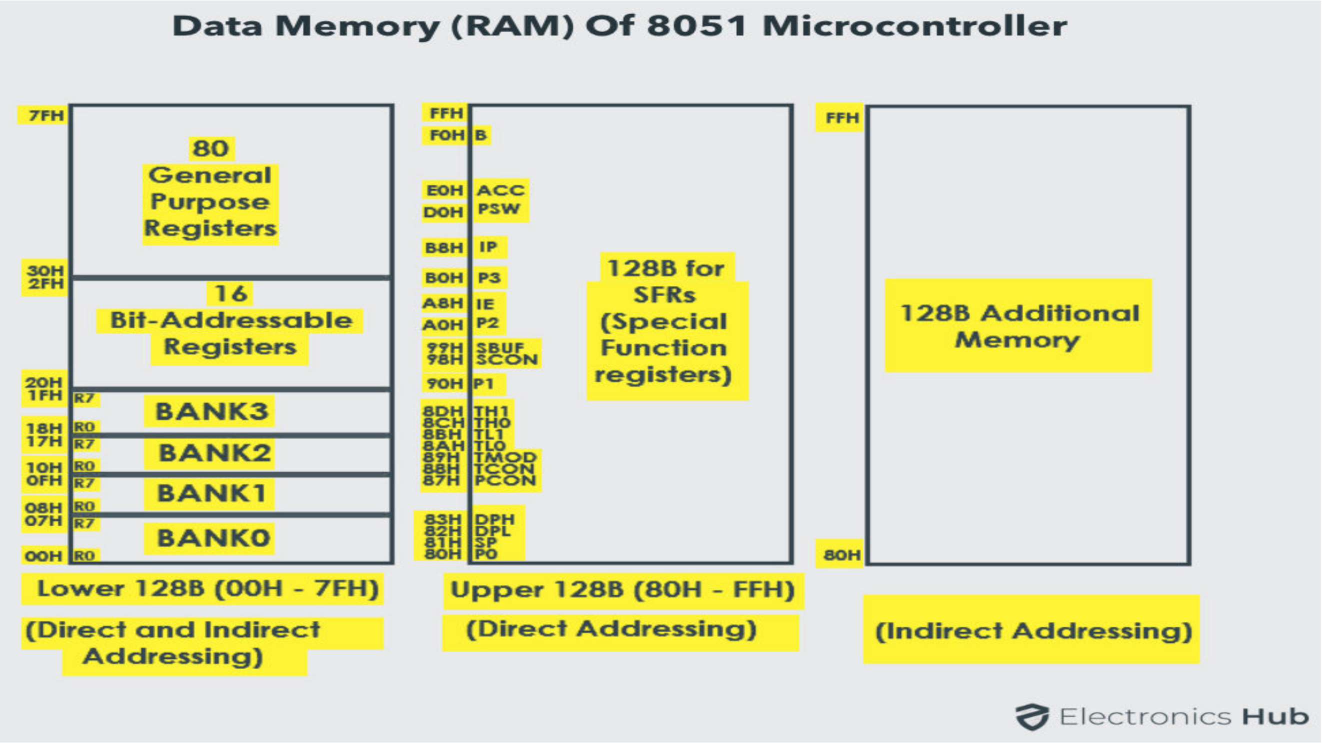
Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of digital electronics

Detailed content of the Lecture:

The 8051 Microcontroller Special Function Registers are used to program and control different hardware peripherals like Timers, Serial Port, I/O Ports etc. In fact, by manipulating the 8051 Microcontroller Special Function Registers (SFRs), you can assess or change the operating mode of the 8051 Microcontroller.

The following image shows you the basic structure of 8051 Microcontroller's Internal RAM.



8051 Microcontroller Special Function Registers (SFRs):

The 8051 Microcontroller Special Function Registers act as a control table that monitor and control the operation of the 8051 Microcontroller. If you observe in Internal RAM Structure, the Address Space from 80H to FFH is allocated to SFRs. Out of these 128 Memory Locations (80H to FFH), there are only 21 locations that are actually assigned to SFRs. Each SFR has one Byte Address and also a unique name which specifies its purpose. Since the SFRs are a part of the Internal RAM Structure, you can access SFRs as if you access the Internal RAM. The main difference is the address space: first 128 Bytes (00H to 7FH) is for regular Internal RAM and next 128 Bytes (80H to FFH) is for SFRS.

List of 8051 Microcontroller Special Function Registers

- A or ACC
- B
- DPL
- DPH
- IE
- IP
- P0
- P1
- P2
- P3
- PCON
- PSW
- SCON
- SBUF
- SP
- TMOD
- TCON
- TL0
- TH0
- TL1
- TH1

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=i8IT4Mu8afI

Important Books/Journals for further learning including the page nos.:

- Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (24-30)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

ECE **II/IV**

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : I – 8015 MICROCONTROLLER **Date of Lecture:**

Topic of Lecture: Special Function Registers(SFRs)

Introduction :

Each bit is wired across to one or other of the microcontroller peripherals. Each is then used either to set up the operating mode of the peripheral or to transfer data between the peripheral and the microcontroller core

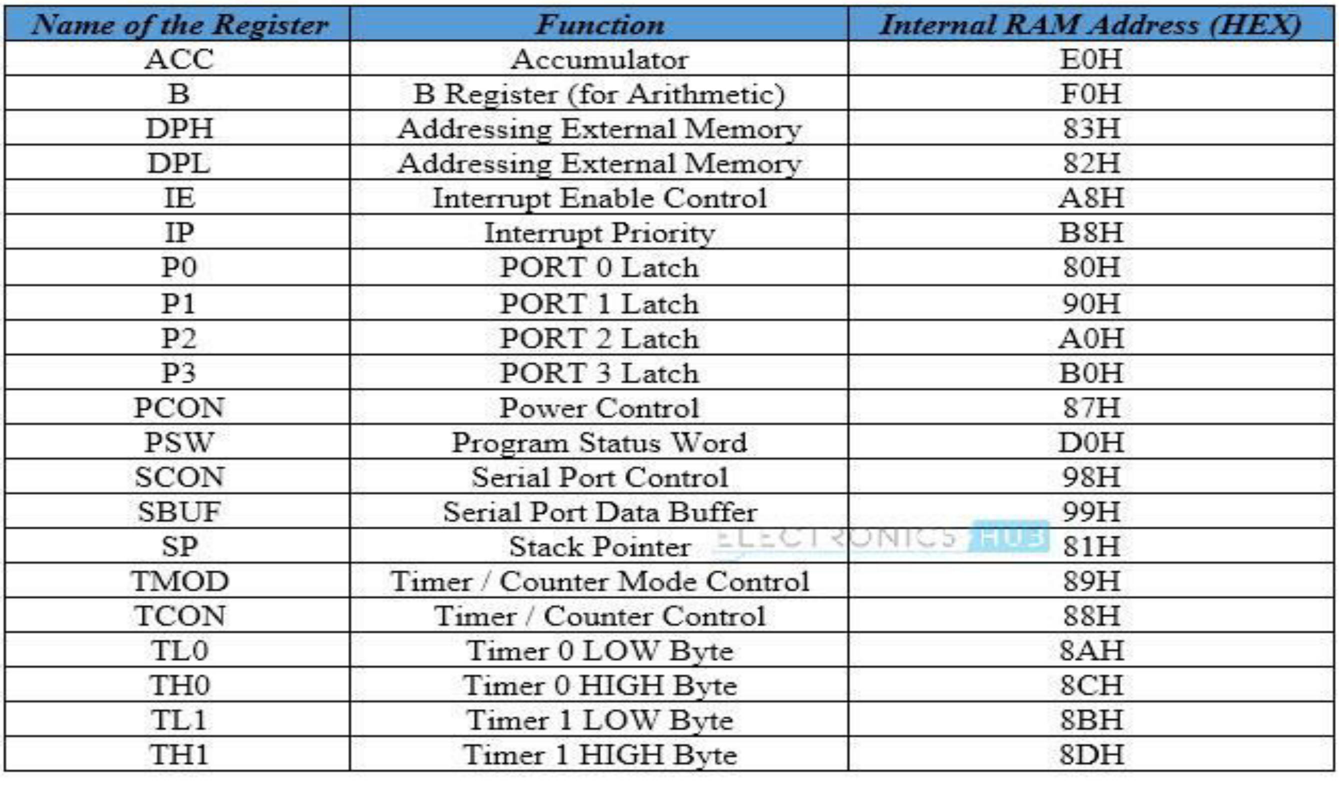
Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of digital electronics

Detailed content of the Lecture:

Categories of 8051 Microcontroller Special Function Registers:

All the 21 8051 Microcontroller Special Function Registers (SFRs) along with their functions and Internal RAM Address is given in the following table.



Name of the Register	*Function*	*Internal RAM Address (HEX)*
ACC	Accumulator	E0H
B	B Register (for Arithmetic)	F0H
DPH	Addressing External Memory	83H
DPL	Addressing External Memory	82H
IE	Interrupt Enable Control	A8H
IP	Interrupt Priority	B8H
P0	PORT 0 Latch	80H
P1	PORT 1 Latch	90H
P2	PORT 2 Latch	A0H
P3	PORT 3 Latch	B0H
PCON	Power Control	87H
PSW	Program Status Word	D0H
SCON	Serial Port Control	98H
SBUF	Serial Port Data Buffer	99H
SP	Stack Pointer	81H
TMOD	Timer / Counter Mode Control	89H
TCON	Timer / Counter Control	88H
TL0	Timer 0 LOW Byte	8AH
TH0	Timer 0 HIGH Byte	8CH
TL1	Timer 1 LOW Byte	8BH
TH1	Timer 1 HIGH Byte	8DH

There are many ways to categorize these 21 Special Function Registers but I find the following way as an appropriate one. The 21 Special Function Registers of 8051 Microcontroller are categorized in to seven groups. They are:

Math or CPU Registers: A and B

Status Register: PSW (Program Status Word)

Pointer Registers: DPTR (Data Pointer – DPL, DPH) and SP (Stack Pointer)

I/O Port Latches: P0 (Port 0), P1 (Port 1), P2 (Port 2) and P3 (Port 3)

Peripheral Control Registers: PCON, SCON, TCON, TMOD, IE and IP

Peripheral Data Registers: TL0, TH0, TL1, TH1 and SBUF

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=i8IT4Mu8afI

Important Books/Journals For Further Learning Including The Page Nos.:

- Gaonkar.R.S, Microprocessor Architecture Programming And Applications With 8085,Wiley Eastern Ltd, New Delhi 2013-Page No(12-16)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L5

ECE

II/IV

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : I – 8051 MICROCONTROLLER

Date of Lecture:

Topic of Lecture: I/O Pins Ports and Circuits

Introduction :

8051 microcontrollers have **4 I/O ports each of 8-bit**, which can be configured as input or output. Hence, total 32 input/output pins allow the microcontroller to be connected with the peripheral devices.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Instruction Sets

Detailed content of the Lecture:

8051 microcontrollers have 4 I/O ports each of 8-bit, which can be configured as input or output. Hence, total 32 input/output pins allow the microcontroller to be connected with the peripheral devices.

- **Pin configuration**, i.e. the pin can be configured as 1 for input and 0 for output as per the logic state.

- **Input/Output (I/O) pin** – All the circuits within the microcontroller must be connected to one of its pins except P0 port because it does not have pull-up resistors built-in.

- **Input pin** – Logic 1 is applied to a bit of the P register. The output FE transistor is turned off and the other pin remains connected to the power supply voltage over a pull-up resistor of high resistance.

Port 0 – The P0 (zero) port is characterized by two functions –

- When the external memory is used then the lower address byte (addresses A0A7) is applied on it, else all bits of this port are configured as input/output.

- When P0 port is configured as an output then other ports consisting of pins with built-in pull-up resistor connected by its end to 5V power supply, the pins of this port have this resistor left out.

Input Configuration

If any pin of this port is configured as an input, then it acts as if it “floats”, i.e. the input has unlimited input resistance and in-determined potential.

Output Configuration

When the pin is configured as an output, then it acts as an "open drain". By applying logic 0 to a port bit, the appropriate pin will be connected to ground (0V), and applying logic 1, the external output will keep on "floating".

In order to apply logic 1 (5V) on this output pin, it is necessary to build an external pullup resistor.

Port 1

P1 is a true I/O port as it doesn't have any alternative functions as in P0, but this port can be configured as general I/O only. It has a built-in pull-up resistor and is completely compatible with TTL circuits.

Port 2

P2 is similar to P0 when the external memory is used. Pins of this port occupy addresses intended for the external memory chip. This port can be used for higher address byte with addresses A8-A15. When no memory is added then this port can be used as a general input/output port similar to Port 1.

Port 3

In this port, functions are similar to other ports except that the logic 1 must be applied to appropriate bit of the P3 register.

Pins Current Limitations

- When pins are configured as an output (i.e. logic 0), then the single port pins can receive a current of 10mA.
- When these pins are configured as inputs (i.e. logic 1), then built-in pull-up resistors provide very weak current, but can activate up to 4 TTL inputs of LS series.
- If all 8 bits of a port are active, then the total current must be limited to 15mA (port P0: 26mA).
- If all ports (32 bits) are active, then the total maximum current must be limited to 71mA.

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=WU8uvapSIic

Important Books/Journals for further learning including the page nos.:

- Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (156-188)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L6

ECE

II/IV

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : I – 8051 MICROCONTROLLER

Date of Lecture:

Topic of Lecture: Instruction set

Introduction :

An instruction set is **a group of commands for a CPU in machine language**. The term can refer to all possible instructions for a CPU or a subset of instructions to enhance its performance in certain situations. ... Some instructions are simple read, write and move commands that direct data to different hardware.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Instruction set

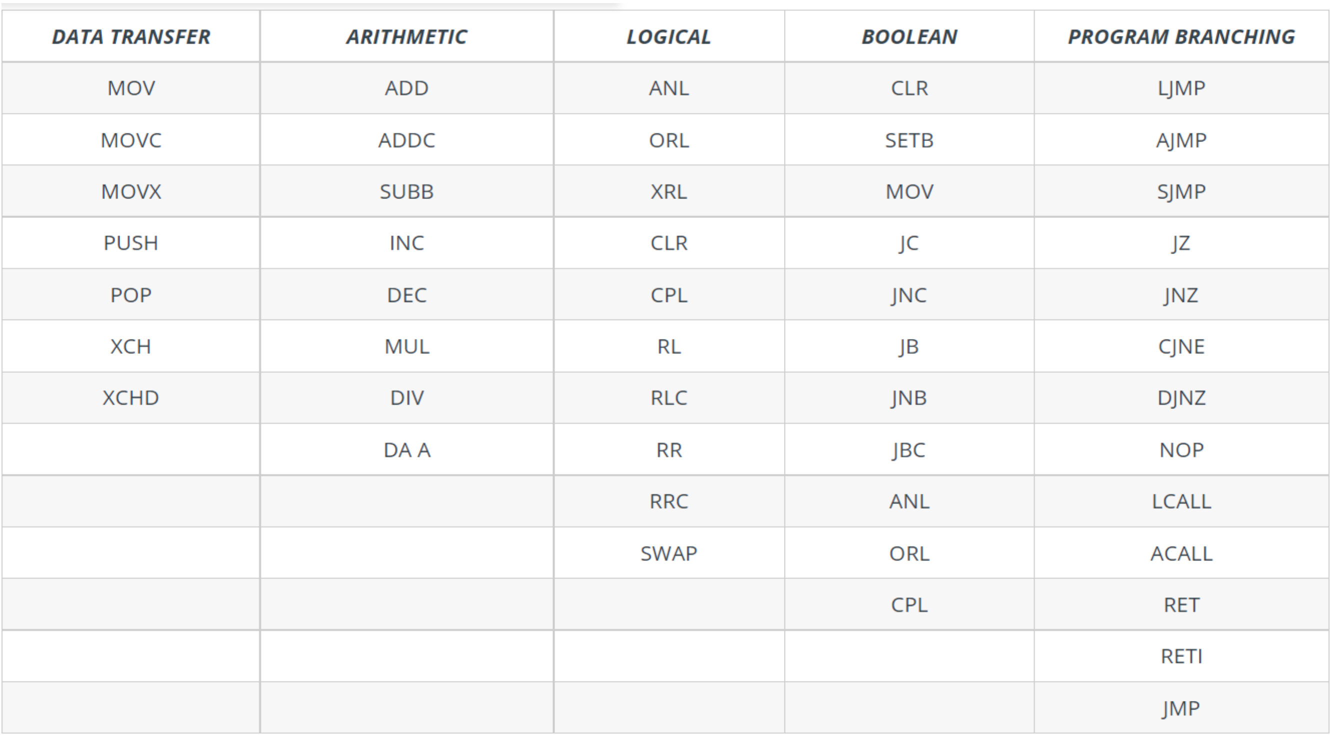
Detailed content of the Lecture:

An Instruction Set is unique to a family of computers. This tutorial introduces the 8051 Microcontroller Instruction Set also called as the MCS-51 Instruction Set. As the 8051 family of Microcontrollers are 8-bit processors, the 8051 Microcontroller Instruction Set is optimized for 8-bit control applications. As a typical 8-bit processor, the 8051 Microcontroller instructions have 8-bit Opcodes. As a result, the 8051 Microcontroller instruction set can have up to 2^8 = 256 Instructions.

A Brief Look at 8051 Microcontroller Instructions and Groups:

Before going into the details of the 8051 Microcontroller Instruction Set, Types of Instructions and the Addressing Mode, let us take a brief look at the instructions and the instruction groups of the 8051 Microcontroller Instruction Set (the MCS-51 Instruction Set).

The following table shows the 8051 Instruction Groups and Instructions in each group. There are 49 Instruction Mnemonics in the 8051 Microcontroller Instruction Set and these 49 Mnemonics are divided into five groups.



DATA TRANSFER	ARITHMETIC	LOGICAL	BOOLEAN	PROGRAM BRANCHING
MOV	ADD	ANL	CLR	LJMP
MOVC	ADDC	ORL	SETB	AJMP
MOVX	SUBB	XRL	MOV	SJMP
PUSH	INC	CLR	JC	JZ
POP	DEC	CPL	JNC	JNZ
XCH	MUL	RL	JB	CJNE
XCHD	DIV	RLC	JNB	DJNZ
	DA A	RR	JBC	NOP
		RRC	ANL	LCALL
		SWAP	ORL	ACALL
			CPL	RET
				RETI
				JMP

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=9VY6d6oJr7s

Important Books/Journals for further learning including the page nos.:

Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (352-370)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408,Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

ECE

II/IV

L/

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : I – 8051 MICROCONTROLLER **Date of Lecture:**

Topic of Lecture: : Instruction set

Introduction :

An instruction set is **a group of commands for a CPU in machine language**. The term can refer to all possible instructions for a CPU or a subset of instructions to enhance its performance in certain situations. ... Some instructions are simple read, write and move commands that direct data to different hardware

Prerequisite knowledge for Complete understanding and learning of Topic:

- Instruction set

Detailed content of the Lecture:

What is an Addressing Mode?

An Addressing Mode is a way to locate a target Data, which is also called as Operand. The 8051 Family of Microcontrollers allows five types of Addressing Modes for addressing the Operands. They are:

- Immediate Addressing
- Register Addressing
- Direct Addressing
- Register – Indirect Addressing
- Indexed Addressing

Immediate Addressing

In Immediate Addressing mode, the operand, which follows the Opcode, is a constant data of either 8 or 16 bits. The name Immediate Addressing came from the fact that the constant data to be stored in the memory immediately follows the Opcode. The constant value to be stored is specified in the instruction itself rather than taking from a register. The destination register to which the constant data must be copied should be the same size as the operand mentioned in the instruction.

Example: MOV A, #030H

Here, the Accumulator is loaded with 30 (hexadecimal). The # in the operand indicates that it is a data and not the address of a Register. Immediate Addressing is very fast as the data to be loaded is given in the instruction itself.

Register Addressing

In the 8051 Microcontroller Memory Organization Tutorial, we have seen the organization of RAM and four banks of Working Registers with eight Registers in each bank. In Register Addressing mode, one of the eight registers (R0 – R7) is specified as Operand in the Instruction. It is important to select the appropriate Bank with the help of PSW Register. Let us see a example of Register Addressing assuming that Bank0 is selected.

Example: MOV A, R5

Here, the 8-bit content of the Register R5 of Bank0 is moved to the Accumulator.

Direct Addressing

In Direct Addressing Mode, the address of the data is specified as the Operand in the instruction. Using Direct Addressing Mode, we can access any register or on-chip variable. This includes general purpose RAM, SFRs, I/O Ports, Control registers.

Example: MOV A, 47H

Here, the data in the RAM location 47H is moved to the Accumulator.

Register Indirect Addressing

In the Indirect Addressing Mode or Register Indirect Addressing Mode, the address of the Operand is specified as the content of a Register. This will be clearer with an example.

Example: MOV A, @R1

The @ symbol indicates that the addressing mode is indirect. If the contents of R1 is 56H, for example, then the operand is in the internal RAM location 56H. If the contents of the RAM location 56H is 24H, then 24H is moved into accumulator. Only R0 and R1 are allowed in Indirect Addressing Mode. These register in the indirect addressing mode are called as Pointer registers.

Indexed Addressing Mode

With Indexed Addressing Mode, the effective address of the Operand is the sum of a base register and an offset register. The Base Register can be either Data Pointer (DPTR) or Program Counter (PC) while the Offset register is the Accumulator (A). In Indexed Addressing Mode, only MOVC and JMP instructions can be used. Indexed Addressing Mode is useful when retrieving data from look-up tables.

Example: MOVC A, @A+DPTR

Here, the address for the operand is the sum of contents of DPTR and Accumulator

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=9VY6d6oJr7s

Important Books/Journals for further learning including the page nos.:

- Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (78-83)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

ECE **II/IV**

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : I –8051 **MICROCONTROLLER** **Date of Lecture:**

Topic of Lecture: Addressing modes

Introduction :

An Addressing Mode is a way to locate a target Data, which is also called as Operand. The 8051 Family of Microcontrollers allows five types of Addressing Modes for addressing the Operands. They are: Immediate Addressing. Register Addressing.

Prerequisite knowledge for Complete understanding and learning of Topic:

➢ Addressing modes

Detailed content of the Lecture:

In 8051 there are 1-byte, 2-byte instructions and very few 3-byte instructions are present. The opcodes are 8-bit long. As the opcodes are 8-bit data, there are 256 possibilities. Among 256, 255 opcodes are implemented.

The clock frequency is12MHz, so 64 instruction types are executed in just 1 μs, and rest are just 2 μs. The Multiplication and Division operations take 4 μsto to execute.

In 8051 There are six types of addressing modes.

- Immediate Addressing Mode
- Register Addressing Mode
- Direct Addressing Mode
- Register Indirect Addressing Mode
- Indexed Addressing Mode
- Implied Addressing Mode

Immediate addressing mode

In this Immediate Addressing Mode, the data is provided in the instruction itself. The data is provided immediately after the opcode. These are some examples of Immediate Addressing Mode.

```
MOVA, #0AFH;
MOVR3, #45H;
MOVDPTR, #FE00H;
```

In these instructions, the # symbol is used for immediate data. In the last instruction, there is DPTR. The DPTR stands for Data Pointer. Using this, it points the external data memory location. In the first instruction, the immediate data is AFH, but one 0 is added at the beginning. So when the data is starting with A to F, the data should be preceded by 0.

Register addressing mode

In the register addressing mode the source or destination data should be present in a register (R0 to R7). These are some examples of Register Addressing Mode.

```
MOVA, R5;
MOVR2, #45H;
MOVR0, A;
```

In 8051, there is no instruction like **MOVR5, R7**. But we can get the same result by using this instruction **MOV R5, 07H**, or by using **MOV 05H, R7**. But this two instruction will work when the selected register bank is **RB0**. To use another register bank and to get the same effect, we have to add the starting address of that register bank with the register number. For an example, if the RB2 is selected, and we want to access R5, then the address will be (10H + 05H = 15H), so the instruction will look like this **MOV 15H, R7**. Here 10H is the starting address of Register Bank 2.

Direct Addressing Mode

In the Direct Addressing Mode, the source or destination address is specified by using 8-bit data in the instruction. Only the internal data memory can be used in this mode. Here some of the examples of direct Addressing Mode.

```
MOV80H, R6;
MOVR2, 45H;
MOVR0, 05H;
```

The first instruction will send the content of registerR6 to port P0 (Address of Port 0 is 80H). The second one is forgetting content from 45H to R2. The third one is used to get data from Register R5 (When register bank RB0 is selected) to register R5.

Register indirect addressing Mode

In this mode, the source or destination address is given in the register. By using register indirect addressing mode, the internal or external addresses can be accessed. The R0 and R1 are used for 8-bit addresses, and DPTR is used for 16-bit addresses, no other registers can be used for addressing purposes. Let us see some examples of this mode.

```
MOV0E5H, @R0;
MOV@R1, 80H
```

In the instructions, the @ symbol is used for register indirect addressing. In the first instruction, it is showing that theR0 register is used. If the content of R0 is 40H, then that instruction will take the data which is located at location 40H of the internal RAM. In the second one, if the content of R1 is 30H, then it indicates that the content of port P0 will be stored at location 30H in the internal RAM.

```
MOVXA, @R1;
MOV@DPTR, A;
```

In these two instructions, the X in MOVX indicates the external data memory. The external data memory can only be accessed in register indirect mode. In the first instruction if the R0 is holding 40H, then A will get the content of external RAM location40H. And in the second one, the content of A is overwritten in the location pointed by DPTR.

Indexed addressing mode

In the indexed addressing mode, the source memory can only be accessed from program memory only. The destination operand is always the register A. These are some examples of Indexed addressing mode.

```
MOVCA, @A+PC;
MOVCA, @A+DPTR;
```

The C in MOVC instruction refers to code byte. For the first instruction, let us consider A holds 30H. And the PC value is1125H. The contents of program memory location 1155H (30H + 1125H) are moved to register A.

Implied Addressing Mode

In the implied addressing mode, there will be a single operand. These types of instruction can work on specific registers only. These types of instructions are also known as register specific instruction. Here are some examples of Implied Addressing Mode.

```
RLA;
SWAPA;
```

These are 1- byte instruction. The first one is used to rotate the A register content to the Left. The second one is used to swap the nibbles in A.

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=LUlq_KBhhXE

Important Books/Journals for further learning including the page nos.:

- Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (41-45)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

ECE **II/IV**

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : I – 8051 MICROCONTROLLER **Date of Lecture:**

Topic of Lecture- Assembly language programming

Introduction :

An assembly language is a low-level programming language for microprocessors and other programmable devices. It is not just a single language, but rather a group of languages. An assembly language implements a symbolic representation of the machine code needed to program a given CPU architecture

Prerequisite knowledge for Complete understanding and learning of Topic:

- Assembly language programming

Detailed content of the Lecture:

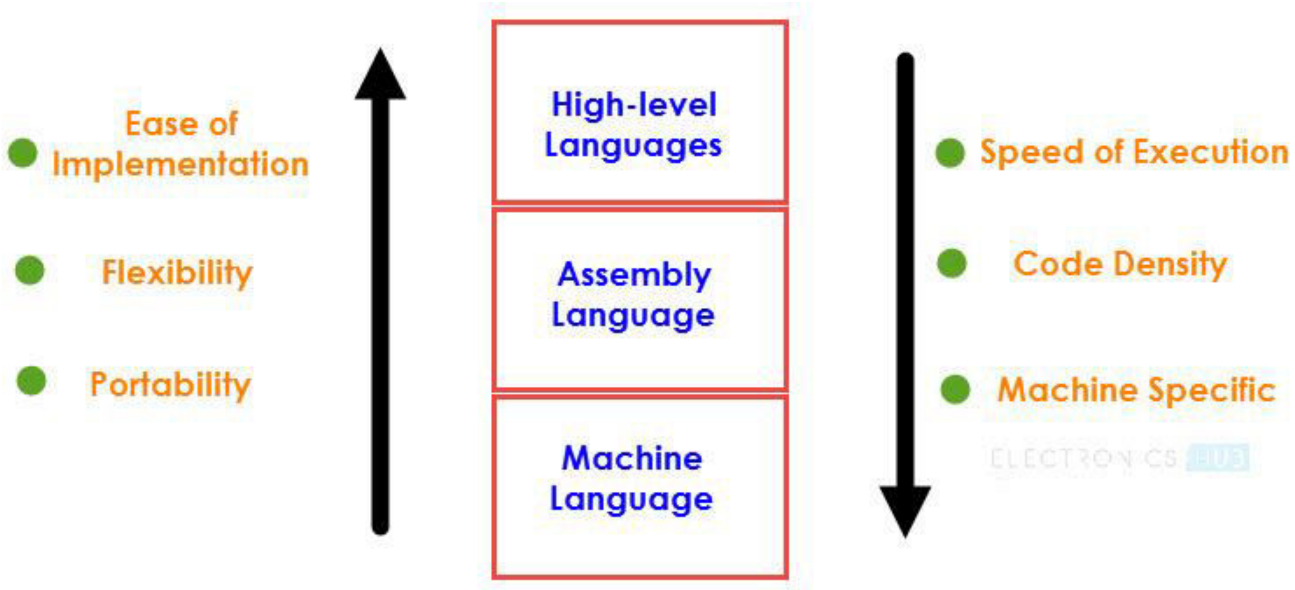
Programming in the sense of Microcontrollers (or any computer) means writing a sequence of instructions that are executed by the processor in a particular order to perform a predefined task. Programming also involves debugging and troubleshooting of instructions and instruction sequence to make sure that the desired task is performed.

Like any language, Programming Languages have certain words, grammar and rules. There are three types or levels of Programming Languages for 8051 Microcontroller. These levels are based on how closely the statements in the language resemble the operations or tasks performed by the Microcontroller.

The three levels of Programming Languages are:

- Machine Language

- Assembly Language
- High-level Language



Machine language

In Machine language or Machine Code, the instructions are written in binary bit patterns i.e. combination of binary digits 1 and 0, which are stored as HIGH and LOW Voltage Levels. This is the lowest level of programming languages and is the language that a Microcontroller or Microprocessor actually understands.

Assembly Language

The next level of Programming Language is the Assembly Language. Since Machine Language or Code involves all the instructions in 1's and 0's, it is very difficult for humans to program using it. Assembly Language is a pseudo-English representation of the Machine Language. The 8051 Microcontroller Assembly Language is a combination of English like words called Mnemonics and Hexadecimal codes.

It is also a low level language and requires extensive understanding of the architecture of the Microcontroller.

High-level Language

The name High-level language means that you need not worry about the architecture or other internal details of a microcontroller and they use words and statements that are easily understood by humans.

Few examples of High-level Languages are BASIC, C Pascal, C++ and Java. A program called Compiler will convert the Programs written in High-level languages to Machine Code.

Why Assembly Language?

Although High-level languages are easy to work with, the following reasons point out the advantage of Assembly Language

- The Programs written in Assembly gets executed faster and they occupy less memory.
- With the help of Assembly Language, you can directly exploit all the features of a Microcontroller.
- Using Assembly Language, you can have direct and accurate control of all the Microcontroller's resources like I/O Ports, RAM, SFRs, etc.
- Compared to High-level Languages, Assembly Language has less rules and restrictions.

Structure of the 8051 Microcontroller Assembly Language:

The Structure or Syntax of the 8051 Microcontroller Assembly Language is discussed here. Each line or statement of the assembly language program of 8051 Microcontroller consists of three fields: Label, Instruction and Comments.

The arrangement of these fields or the order in which they appear is shown below.

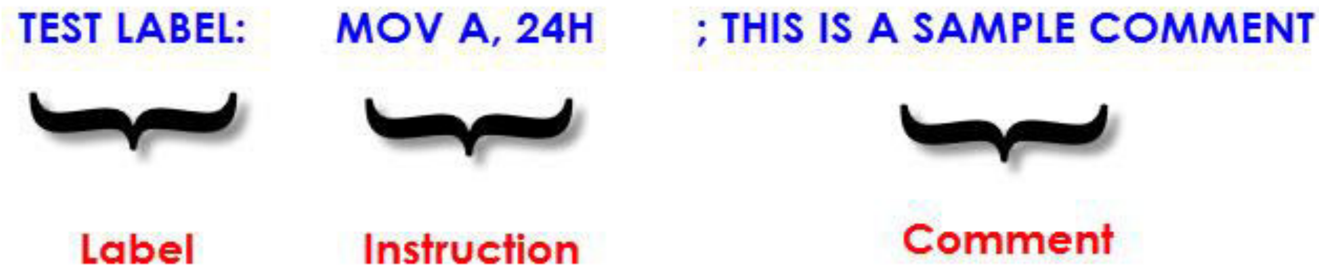
```
[Label:]          Instructions                [//Comments]
```

NOTE: The brackets for Label and Comments mean that these fields are optional and may not be used in all statements in a program.

Before seeing about these three fields, let us first see an example of how a typical statement or line in an 8051 Microcontroller Assembly Language looks like.

```
TESTLABEL:   MOV A, 24H   ; THIS IS A SAMPLE COMMENT
```

In the above statement, the "TESTLABEL" is the name of the Label, "MOV A, 24H" is the Instruction and the "THIS IS A SAMPLE COMMENT" is a Comment.

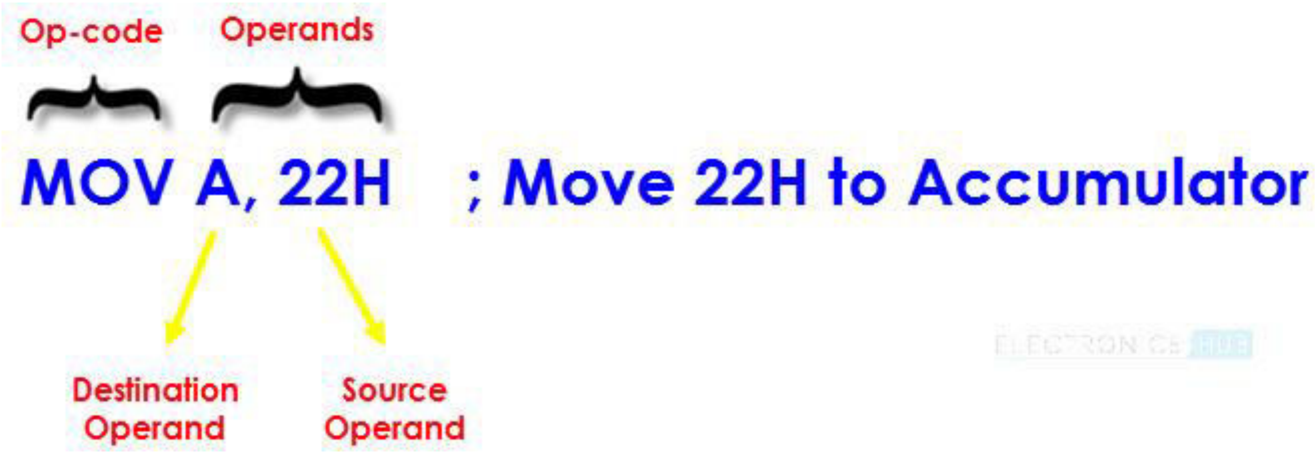


Label
The Label is programmer chosen name for a Memory Location or a statement in a program. The Label part of the statement is optional and if present, the Label must be terminated with a Colon (:).An important point to remember while selecting a name for the Label is that they

should reduce the need for documentation.

Instruction

The Instruction is the main part of the 8051 Microcontroller Assembly Language Programming as it is responsible for the task performed by the Microcontroller. Any Instruction in the Assembly Language consists of two parts: Op-code and Operand(s).



The first part of the Instruction is the Op-code, which is short for Operation Code, specifies the operation to be performed by the Microcontroller. Op-codes in Assembly Language are called as Mnemonics. Op-codes are in binary format (used in Machine Language) while the Mnemonic (which are equivalent to Op-codes) are English like statements.

The second part of the instruction is called the Operand(s) and it represents the Data on which the operation is performed. There are two types of Operands: the Source Operand and the Destination Operand. The Source Operand is the Input of the operation and the Destination Operand is where the result is stored.

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=g7Gypl9zNz8

Important Books/Journals for further learning including the page nos.:

- Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (30-41)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

L10

LECTURE HANDOUTS

L10

ECE

II/IV

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : II – 8051 MICROCONTROLLER INTERFACING

Date of Lecture:

Topic of Lecture: Programming 8051 Timers

Introduction :

- 8051 Microcontrollers have two timers/counters which work on the clock frequency. Timer/counter can be used for time delay generation, counting external events, etc..

Prerequisite knowledge for Complete understanding and learning of Topic:

- Processors and controllers

Detailed content of the Lecture:

The programming of 8051 Timers can be done by using either polling method or by using interrupt. In polling, the microcontroller keeps monitoring the status of Timer flag. While doing so, it does no other operation and consumes all its processing time in checking the Timer flag until it is **raised on a rollover**. In interrupt method controller responds to only when the Timer flag is raised. The interrupt method prevents the wastage of controller's processing time unlike polling method.
Polling is mostly used for time delay generation and interrupt method is more useful when waveforms are to be generated or some action has to be repeated in fixed delays.

(i) Polling Method

The polling method involves the following algorithm:
1. Configure the Timer mode by passing a hex value into the TMOD register. This will tell the controller about which Timer is be used; the mode of Timer; operation (to be used as timer or counter); and whether external interrupt is required to start Timer.
2. Load the initial values in the Timer low TLx and high THx byte. (x = 0/1)
3. Start the Timer by setting TRx bit.
4. Wait while the Timer flag TFx is raised.
5. Clear the Timer flag. The Timer flag is raised when Timer **rolls over** from FFFFH to 0000H. If

the Timer is not stopped, it will start updating from 0000H in case of modes 0 & 1 while with initial value in case of mode 2. If TFx is not cleared, controller will not be able to detect next rollover.

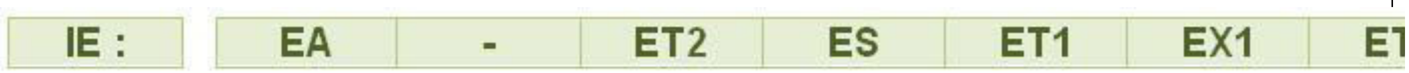
6. Stop the Timer by clearing TRx bit. If TRx bit is not cleared the Timer will restart updating from 0000H after the rollover in case of modes 0 and 1 while with initial value in case of mode 2.

(ii) Interrupt Method

The **interrupt** method makes use of a register called **Interrupt Enable** (IE) register. An 8051 microcontroller has 6 hardware interrupts. The interrupts refer to a notification, communicated to the controller, by a hardware device or software, on receipt of which controller skips temporarily whatsoever it was doing and responds to the interrupt.

The controller starts the execution of an **Interrupt Service Routine** (ISR) or Interrupt Handler which is a piece of code that tells the processor or controller **what to do on receipt of an interrupt**. After the execution of ISR, controller returns to whatever it was doing earlier (before the interrupt was received).

The Interrupt Enable register has the following bits which enable or disable the hardware interrupts of 8051 microcontroller.



IE :	EA	-	ET2	ES	ET1	EX1	ET

When EA (IE.7) is set (=1), interrupts are enabled. When clear (EA=0), interrupts are disabled and controller does not respond to any interrupts.

ET0, ET1 & ET2 (IE.3, IE.4 & IE.5) are Timer interrupt bits. When set (high) the timer are enabled and when cleared (low) they are disabled. (8052 controllers have three Timers, so ET2 is its Timer 2 interrupt bit.) The ISR corresponding to these interrupts are executed when the TFx flags of respective Timers are raised. For more details on other IE register bits, refer **Interrupts with 8051**.

Note that IE register is bit addressable.

Timer programming using Timer interrupts involves following algorithm.

Configure the Timer mode by passing a hex value to TMOD register.

Load the initial values in the Timer low TLx and high THx byte.

Enable the Timer interrupt by passing hex value to IE register or setting required bits of IE register. For example,

IE = 0x82; enables Timer 0 interrupt
IE = 0x88; enables Timer 1 interrupt

or

EA = 1;
ET0 = 1; enables Timer 0 interrupt
IE^7 = 1;
IE^3 = 1; enables Timer 1 interrupt

Start the Timer by setting TRx bit.

Write Interrupt Service Routine (ISR) for the Timer interrupt. For example,

ISR definition for Timer 0 :

```
void ISR_Timer0(void) interrupt 1

{

<Body of ISR>

}
```

ISR definition for Timer 1 :

```
void ISR_Timer1(void) interrupt 3
{
<Body of ISR>
}
```

6. If the Timer has to be stopped after once the interrupt has occurred, the ISR must contain the statement to stop the Timer.

For example,

```
void ISR_Timer1(void) interrupt 3
{
<Body of ISR>
TR1 =0;
}
```

If a routine written for Timer interrupt has to be repeated again and again, the Timer run bit need not be cleared. But it should be kept in mind that **Timer will start updating from 0000H** and **not the initial values in case of mode 0 and 1**. So the initial values must be reloaded in the interrupt service routine.

For example,

```
void ISR_Timer1(void) interrupt 3
{
<Body of ISR>
TH1 =0XFF;              //load with initial values if in mode 0 or 1
TL1 = 0xFC;
}
```

Video Content / Details of website for further learning (if any):

https://www.engineersgarage.com/timers-8051-timer-programming/

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 Page No: 24-28

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L11

ECE

II/IV

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : II – 8051 MICROCONTROLLER INTERFACING

Date of Lecture:

Topic of Lecture: Serial Port Programming

Introduction :

- One of the 8051's many powerful features -integrated UART, known as a serial port easily read and write values to the serial port instead of turning on and off one of the I/ lines in rapid succession to properly "clock out" each individual bit, including start bi stop bits and parity bits.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Processors and controllers
- Central processing unit
- Memory Devices

Detailed content of the Lecture:

Serial Port Programming: 8051 Serial Communication

One of the 8051's many powerful features -integrated *UART*, known as a serial port to easily re and write values to the serial port instead of turning on and off one of the I/O lines in ra succession to properly "clock out" each individual bit, including start bits, stop bits and parity bit

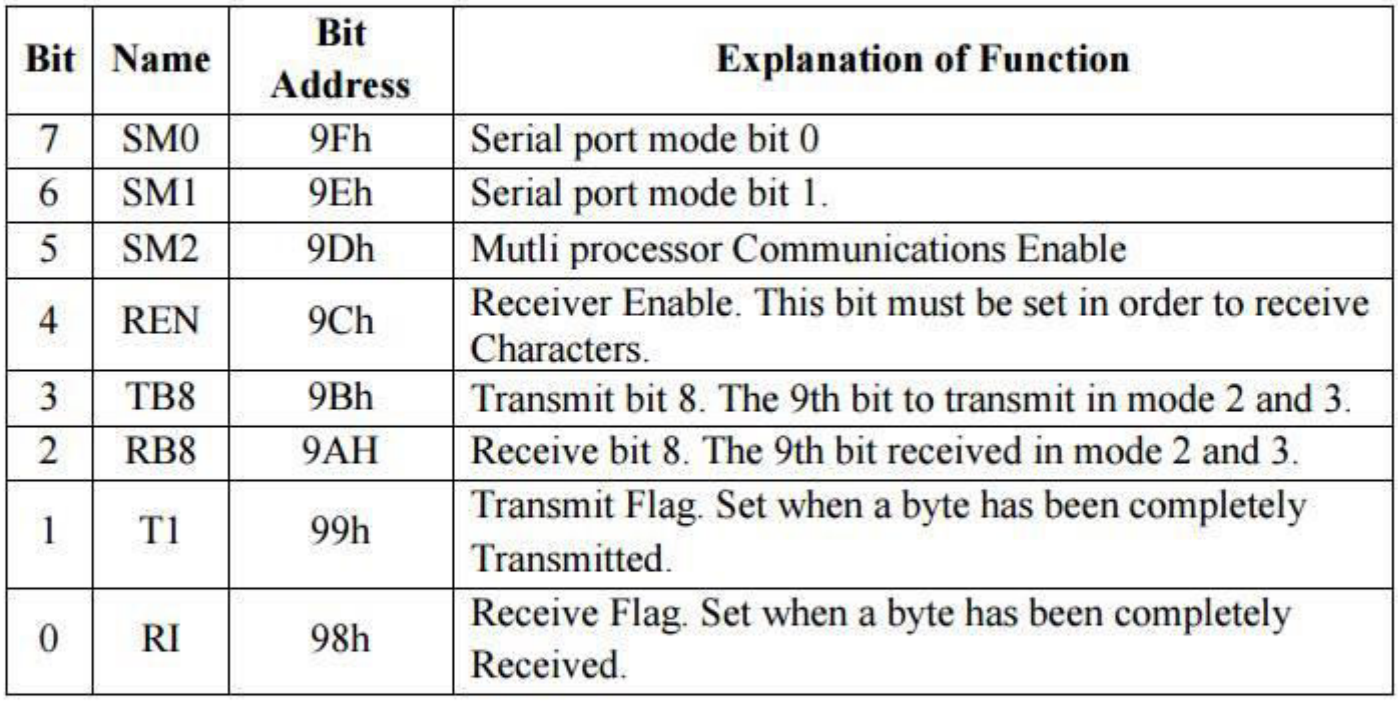
✓ **Setting the Serial Port Mode** configures it by specifying 8051 how many data bits

we want, the baud rate we will be using and how the baud rate will be determined. First, le present the "Serial Control" (SCON) SFR and define what each bit of the SFR

represents:

Table : Definition of SCON SFR

Table 5.3 Definition of SCON SFR

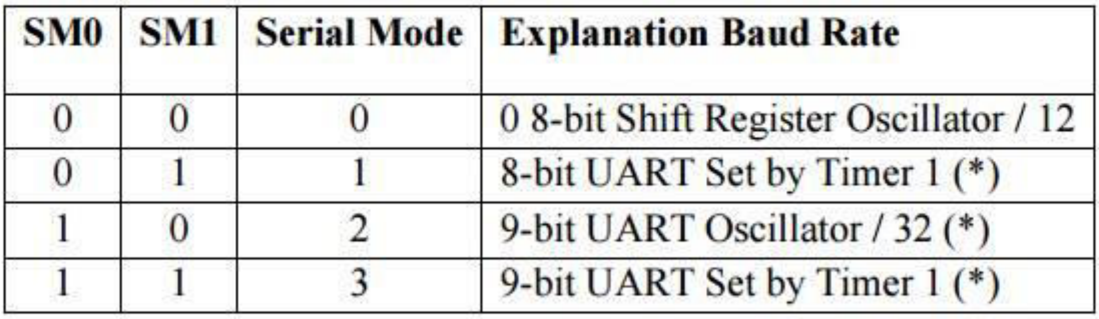


Bit	Name	Bit Address	Explanation of Function
7	SM0	9Fh	Serial port mode bit 0
6	SM1	9Eh	Serial port mode bit 1.
5	SM2	9Dh	Mutli processor Communications Enable
4	REN	9Ch	Receiver Enable. This bit must be set in order to receive Characters.
3	TB8	9Bh	Transmit bit 8. The 9th bit to transmit in mode 2 and 3.
2	RB8	9AH	Receive bit 8. The 9th bit received in mode 2 and 3.
1	T1	99h	Transmit Flag. Set when a byte has been completely Transmitted.
0	RI	98h	Receive Flag. Set when a byte has been completely Received.

Additionally, it is necessary to define the function of SM0 and SM1 by an additional table: Ta 5.4 SCON as serial Port

Table 5.4 Modes of SCON

Table 5.4 Modes of SCON



SM0	SM1	Serial Mode	Explanation Baud Rate
0	0	0	0 8-bit Shift Register Oscillator / 12
0	1	1	8-bit UART Set by Timer 1 (*)
1	0	2	9-bit UART Oscillator / 32 (*)
1	1	3	9-bit UART Set by Timer 1 (*)

The SCON SFR allows us to configure the Serial Port. The first four bits (bits 4 through are configuration bits:

Bits **SM0** and **SM1** is to set the *serial mode* to a value between 0 and 3, inclusive as in ta above selecting the Serial Mode selects the mode of operation (8-bit/9-bit, UART

or Shift Register) and also determines how the baud rate will be calculated. In modes 0 and 2 baud rate is fixed based on the oscillator’s frequency. In modes 1 and 3 the baud rate is

variable based on how often Timer 1 overflows.

The next bit, **SM2**, is a flag for " Multiprocessor communication whenever a byte has be received the 8051 will set the "RI" (Receive Interrupt) flag to let the program know that a byte l been received and that it needs to be processed.

However, when SM2 is set the "RI" flag will only be triggered if the 9th bit received wa "1". if SM2 is set and a byte is received whose 9th bit is clear, the RI flag will never be set .Y will almost always want to clear this bit so that the flag is set upon reception of *any* character.

The next bit, **REN**, is "Receiver Enable." is set indicate to data received via the serial port

The last four bits (bits 0 through 3) are operational bits. They are used when actually send and receiving data--they are not used to configure the serial port.

The **TB8** bit is used in modes 2 and 3. In modes 2 and 3, a total of nine data bits are

transmitted. The first 8 data bits are the 8 bits of the main value, and the ninth bit is taken fr TB8. If TB8 is set and a value is written to the serial port, the data’s bits will be

written to the serial line followed by a "set" ninth bit. If TB8 is clear the ninth bit will be "clear."

The **RB8** also operates in modes 2 and 3and functions essentially the same way as TB8, l on the reception side. When a byte is received in modes 2 or 3, a total of nine bits are received. this case, the first eight bits received are the data of the serial byte received and the value of nineth bit received will be placed in RB8.**TI** means "Transmit Interrupt."

When a program writes a value to the serial port, a certain amount of time will pass bef the individual bits of the byte are "clocked out" the serial port. If the program were to write anot byte to the serial port before the first byte was completely output, the data being sent would garbled. Thus, the8051 lets the program know that it has "clocked out" the last byte by setting TI bit.

When the TI bit is set, the program may assume that the serial port is "free" and ready send the next byte. Finally, the **RI** bit means "Receive Interrupt." It functions similarly to the "' bit, but it indicates that a byte has been received. Whenever the 8051 has received a complete b it will trigger the RI bit to let the program know that it needs to read the value quickly, bef another byte is read.

Video Content / Details of website for further learning (if any):

https://www.brainkart.com/article/Serial-Port-Programming--8051-Serial-Communication_7888/

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedd Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 28-32)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram – 637 408, Namakkal Dist., Tamil Nadu

DESIGNING YOUR FUTURE
Estd. 2000

L12

LECTURE HANDOUTS

L12

ECE

II/IV

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : II – 8051 MICROCONTROLLER INTERFACING Date of Lecture:

Topic of Lecture: Interrupts Programming

Introduction :

Interrupt is one of the most important and powerful concepts and features in **microcontroller**/processor applications. Almost all the real world and real time systems built around microcontrollers and microprocessors make use of interrupts.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Programming of 8051

Detailed content of the Lecture:

What is Interrupt

The interrupts refer to a notification, communicated to the controller, by a hardware device or software, on receipt of which controller momentarily stops and responds to the interrupt. Whenever an interrupt occurs the controller completes the execution of the current instruction and starts the execution of an **Interrupt Service Routine** (ISR) or Interrupt Handler. **ISR** is a piece of code that tells the processor or controller what to do when the interrupt occurs. After the execution of ISR, controller returns back to the instruction it has jumped from (before the interrupt was received). The interrupts can be either **hardware interrupts** or **software interrupts**.

Why need interrupts

An application built around microcontrollers generally has the following structure. It takes input from devices like keypad, ADC etc; processes the input using certain algorithm; and generates an output which is either displayed using devices like seven segment, LCD or used further to operate other devices like motors etc. In such designs, controllers interact with the inbuilt devices like **timers** and other interfaced peripherals like sensors, serial port etc. The programmer needs to monitor their status regularly like whether the sensor is giving output, whether a signal has been received or transmitted, whether timer has finished counting, or if an interfaced device needs service from the controller, and so on. This state of continuous monitoring is known as polling.

In polling, the microcontroller keeps checking the status of other devices; and while doing so it does no other operation and consumes all its processing time for monitoring. This problem can be addressed by using interrupts. In interrupt method, the controller responds to only when an interruption occurs. Thus in interrupt method, controller is not required to regularly monitor the status (flags, signals etc.) of interfaced and inbuilt devices.

To understand the difference better, consider the following. The polling method is very much similar to a salesperson. The salesman goes door-to-door requesting to buy its product or service. Like controller keeps monitoring the flags or signals one by one for all devices and caters to whichever needs its service. Interrupt, on the other hand, is very similar to a shopkeeper. Whosoever needs a service or product goes to him and apprises him of his/her needs. In our case, when the flags or signals are received, they notify the controller that they need its service.

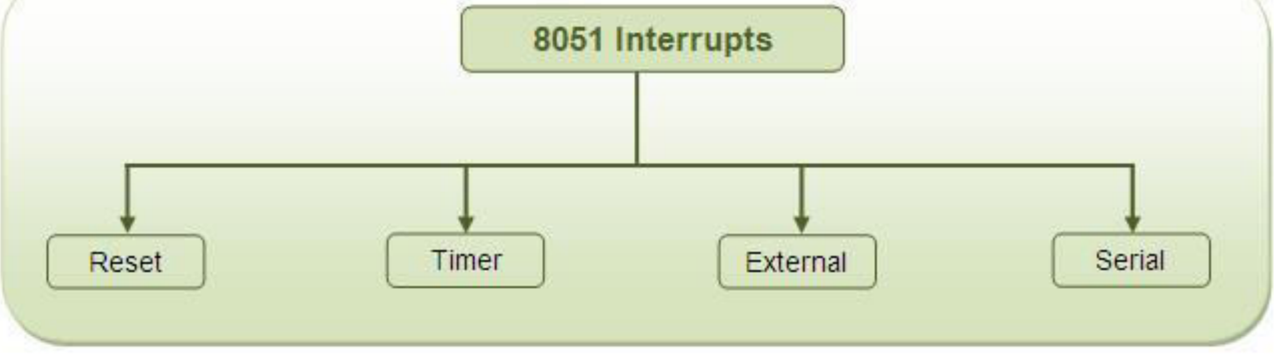
Hardware & Software Interrupt

Hardware and Software interrupt

The interrupts in a controller can be either hardware or software. If the interrupts are generated by the controller's inbuilt devices, like timer interrupts; or by the interfaced devices, they are called the hardware interrupts. If the interrupts are generated by a piece of code, they are termed as software interrupts.

Multiple interrupts

What would happen if multiple interrupts are received by a microcontroller at the same instant? In such a case, the controller assigns priorities to the interrupts. Thus the interrupt with the highest priority is served first. However the priority of interrupts can be changed configuring the appropriate registers in the code.



1. RESET interrupt – This is also known as Power on Reset (POR). When the RESET interrupt is received, the controller restarts executing code from 0000H location. This is an interrupt which is not available to or, better to say, need not be available to the programmer.

2. Timer interrupts – Each Timer is associated with a Timer interrupt. A timer interrupt notifies the microcontroller that the corresponding Timer has finished counting.

3. External interrupts – There are two external interrupts EX0 and EX1 to serve external devices. Both these interrupts are active low. In **AT89C51**, P3.2 (INT0) and P3.3 (INT1) pins are available for external interrupts 0 and 1 respectively. An external interrupt notifies the microcontroller that an external device needs its service.

4. Serial interrupt – This interrupt is used for **serial communication**. When enabled, it notifies the controller whether a byte has been received or transmitted.

Video Content / Details of website for further learning (if any):

https://www.engineersgarage.com/interrupts-programming-8051-hardware-interrupts/

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 Page No: 93-106

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

L13

LECTURE HANDOUTS

L13

ECE

II/IV

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : II –8051 MICROCONTROLLER INTERFACING Date of Lecture:

Topic of Lecture:

Introduction :

- LCDs can display numbers, characters, and graphics. To produce a proper display, the information has to be periodically refreshed. This can be done by the CPU or internally by the LCD device itself.
- Keys in a keyboard are arranged in a matrix of rows and columns. The controller access both rows and columns through ports. Using two ports, we can connect to an 8x8 or a 4x4 matrix keyboard.

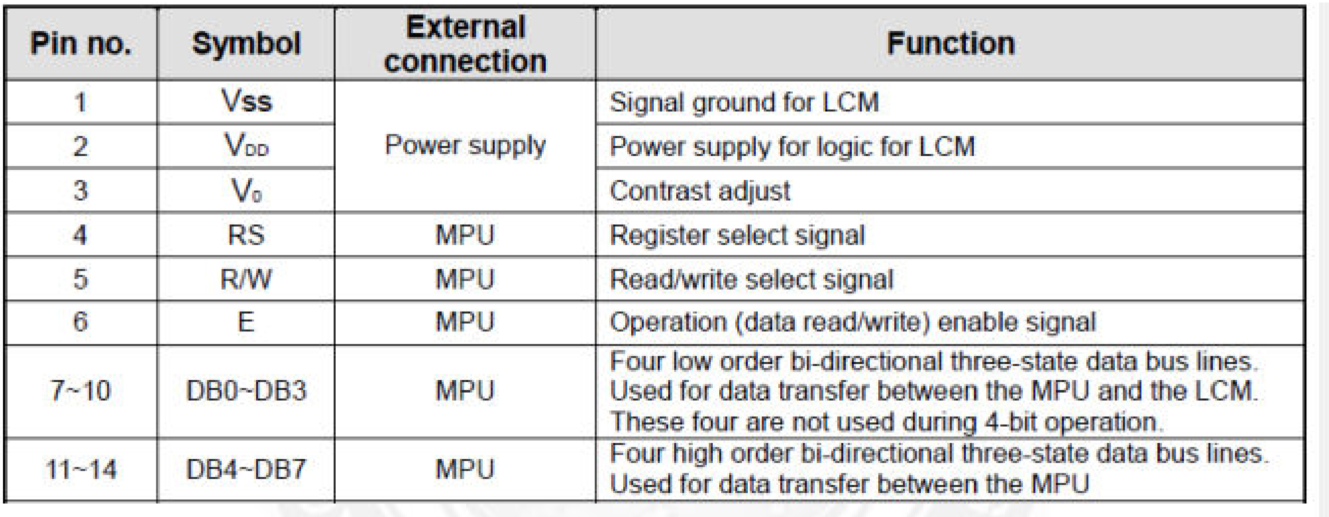
Prerequisite knowledge for Complete understanding and learning of Topic:

- LCD & Keyboard Interfacing 8051

Detailed content of the Lecture:

LCD (LIQUID CRYSTAL DISPLAY) INTERFACE

LCDs can display numbers, characters, and graphics. To produce a proper display, the information has to be periodically refreshed. This can be done by the CPU or internally by the LCD device itself. Incorporating a refreshing controller into the LCD, relieves the CPU of this task and hence many LCDs have built-in controllers. These controllers also facilitate flexible programming for characters and graphics. Table shows the pin description of an LCD. from Optrex.



Pin no.	Symbol	External connection	Function
1	Vss	Power supply	Signal ground for LCM
2	V_{DD}		Power supply for logic for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7~10	DB0~DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11~14	DB4~DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU

- Vss and VDD provide +5v and ground, V0 is used for controlling LCD contrast.
- If RS=0, the instruction command register is selected, allowing the user to send a command such as clear display, cursor at home, etc.
- If RS=1 the data register is selected, allowing the user to send data to be displayed on the LCD.
- R/W input allows the user to Read/ Write the information to the LCD.
- The enable pin is used by the LCD to latch information presented to its data pins.
- The 8-bit data pins are used to send information to LCD.
- Section 5.1 discusses about command codes for writing the instructions on the LCD register. Section 5.2 gives an example program for displaying a character on the LCD.

LCD COMMAND CODES

The LCD's internal controller can accept several commands and modify the display accordingly. These commands would be things like:

- Clear screen
- Return home
- Decrement/Increment cursor

After writing to the LCD, it takes some time for it to complete its internal operations. During this time, it will not accept any new commands or data. Figure 5.4.1 shows the command codes of LCD and Figure 5.4.2 shows the LCD interfacing. We need to insert a time delay between any two commands or data sent to LCD.

KEYBOARD INTERFACING WITH 8051

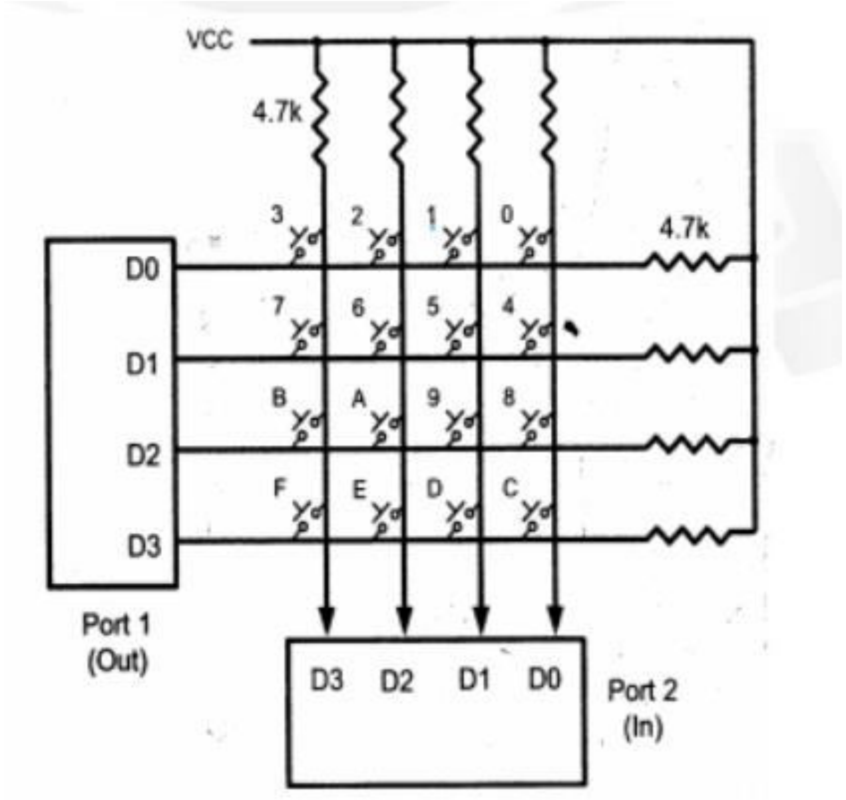
Keys in a keyboard are arranged in a matrix of rows and columns. The controller access both rows and columns through ports. Using two ports, we can connect to an 8x8 or a 4x4 matrix keyboard. When a key is pressed, a row and column make a contact, otherwise there is no contact. We will look at the details using a 4x4 keyboard.

4X 4 KEYBOARD

Figure 5.4.31 shows a 4 x4 matrix connected to two ports

- The rows are connected to an output port(Port 1) and the columns are connected to an input port. (Port 2)
- If no key has been pressed, reading the input port will yield 1s for all columns since they are all connected to high (Vcc).
- If all the rows are grounded and a key is pressed, one of the columns will have 0 since the key pressed provides the path to ground.
- It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed.

Matrix Keyboard Connections to Ports



KEY SCAN

To find out the key pressed , the controller grounds a row by sending a '0' on the corresponding line of the output port. It then reads the data at the columns using the input port. If data from columns is D3-D0=1111, then no key is pressed. If any bit of the column is '0', it indicates that a key is pressed in that column. In this example, the column is identified by the following values: 1110 – key pressed in column 0 1101 – key pressed in column 1 1011 – key pressed in column 2 0111 – key pressed in column 3

Video Content / Details of website for further learning (if any):

https://www.rcet.org.in/uploads/academics/rohini_10051412485.pdf

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 218-223)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408,Namakkal Dist., Tamil Nadu

DESIGNING YOUR FUTURE
Estd. 2000

IQAC

LECTURE HANDOUTS

ECE

II/IV

L14

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : II –8051 MICROCONTROLLER INTERFACING Date of Lecture:

Topic of Lecture:

ADC, DAC & Sensor Interfacing

Introduction :

- ADCs (analog-to-digital converters) are among the most widely used devices for data acquisition
- The digital-to-analog converter (DAC) is a device widely used to convert digital pulses to analog signals

Prerequisite knowledge for Complete understanding and learning of Topic:

- Architecture
- Pin diagram

Detailed content of the Lecture:

ADC Devices

- ADCs (analog-to-digital converters) are among the most widely used devices for data acquisition
- A physical quantity, like temperature, pressure, humidity, and velocity, etc., is converted to electrical (voltage, current) signals using a device called a transducer, or sensor
- We need an analog-to-digital converter to translate the analog signals to digital numbers, so microcontroller can read them

Digital to Analog Conversion

- The conversion process: Digital code is converted to a proportional voltage or current Reference voltage determines the max output DAC can output
- Analog (pseudo analog) output Finite number of output levels
- Input weights Each bit of the input has a different weight
- Resolution (step size) Difference between two consecutive output levels smallest amount of change realizable at the output Weight of the LSB of input

Digital to Analog Conversion

- Bipolar DACs Many DACs produce both positive and negative values 2's complement can be used to represent negative voltages
- We are going to focus on unipolar DACs
- ADCs (Analog-to-Digital Converters) are among the most widely used devices for data acquisition.
- A physical quantity, like temperature, pressure, humidity, and velocity, etc., is converted to electrical (voltage, current) signals using a device called a transducer or sensor
- We need an Analog-to-Digital Converter to translate the analog signals to digital numbers, so microcontroller can read and process them.
- An ADC has n-bit resolution where n can be 8, 10, 12, 16 or even 24 bits.
- The higher-resolution ADC provides a smaller step size, where step size is the smallest change that can be discerned by an ADC.

- In addition to resolution, conversion time is another major factor in judging an ADC.

- Conversion time is defined as the time it takes the ADC to convert the analog input to a digital (binary) number.

- In parallel ADC, we have 8 of more pins dedicated to bringing out the binary data, but in serial ADC we have only one pin for data out.

ADC804 CHIP:

ADC804 IC is an 8-bit parallel analog-to-digital converter.

- It works with +5 volts and has a resolution of 8bits.
- In ADC804 conversion time varies depending on the clocking signals applied to the CLK R and CLK IN pins, but it cannot be faster than 110μs.
- It is the Pin out of ADC0804 in free running mode.

STEPS TO BE FOLLOWED FOR DATA CONVERSION:

The following steps must be followed for data conversion by the ADC804 chip:

- Make CS= 0 and send a L-to-H pulse to pin WR to start conversion.
- Monitor the INTR pin, if high keep polling but if low, conversion is complete, go to next step.
- Make CS= 0 and send a H-to-L pulse to pin RD to get the data out.

Video Content / Details of website for further learning (if any):

https://www.rcet.org.in/uploads/academics/rohini_40959333711.pdf

Important Books/Journals for further learning including the page nos.:

- Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 Page No: (70-75)

Course Faculty
Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408,Namakkal Dist., Tamil Nadu

LECTURE HANDOUTS

ECE **II/IV**

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : II –8051 MICROCONTROLLER INTERFACING Date of Lecture:

Topic of Lecture: External Memory Interface

Introduction :

An **external memory interface** is a bus protocol for communication from an integrated circuit, such as a microprocessor, to an external memory device located on a circuit board..

Prerequisite knowledge for Complete understanding and learning of Topic:

- Architecture
- External Memory

Detailed content of the Lecture:

An **external memory interface** is a bus protocol for communication from an integrated circuit, such as a microprocessor, to an external memory device located on a circuit board. The memory is referred to as external because it is not contained within the internal circuitry of the integrated circuit and thus is externally located on the circuit board.

The external memory interface enables the processor to interface with third level caches, peripherals, and external memory.[1]

Some common external memory interfaces include:

- DDR
- DDR2
- GDDR

DDR SDRAM

Double Data Rate Synchronous Dynamic Random-Access Memory (DDR SDRAM)

It is a double data rate (DDR) synchronous dynamic random-access memory (SDRAM) class of memory integrated circuits used in computers. DDR SDRAM, also retroactively called DDR1 SDRAM, has been superseded by DDR2 SDRAM, DDR3 SDRAM, DDR4 SDRAM and DDR5 SDRAM. None of its successors are forward or backward compatible with DDR1 SDRAM, meaning DDR2, DDR3, DDR4 and DDR5 memory modules will not work in DDR1-equipped motherboards, and vice versa.

Compared to single data rate (SDR) SDRAM, the DDR SDRAM interface makes higher transfer rates possible by more strict control of the timing of the electrical data and clock signals. Implementations often have to use schemes such as phase-locked loops and self-calibration to reach the required timing accuracy.[4][5] The interface uses double pumping (transferring data on both the rising and falling edges of the clock signal) to double data bus bandwidth without a corresponding increase in clock frequency. One advantage of keeping the clock frequency down is that it reduces the signal integrity requirements on the circuit board connecting the memory to the controller. The name "double data rate" refers to the fact that a DDR SDRAM with a certain clock frequency achieves nearly twice the bandwidth of a SDR SDRAM running at the same clock frequency, due to this double pumping.

With data being transferred 64 bits at a time, DDR SDRAM gives a transfer rate (in bytes/s) of (memory bus clock rate) × 2 (for dual rate) × 64 (number of bits transferred) / 8 (number of bits/byte). Thus, with a bus frequency of 100 MHz, DDR SDRAM gives a maximum transfer rate of 1600 MB/s.

DDR2 SDRAM

Double Data Rate 2 Synchronous Dynamic Random-Access Memory (**DDR2 SDRAM**) is a double data rate (DDR) synchronous dynamic random-access memory (SDRAM) interface. It superseded the original DDR SDRAM specification, and was itself superseded by DDR3 SDRAM (launched in 2007). DDR2 DIMMs are neither forward compatible with DDR3 nor backward compatible with DDR.

In addition to double pumping the data bus as in DDR SDRAM (transferring data on the rising and falling edges of the bus clock signal), DDR2 allows higher bus speed and requires lower power by running the internal clock at half the speed of the data bus. The two factors combine to produce a total of four data transfers per internal clock cycle.

Since the DDR2 internal clock runs at half the DDR external clock rate, DDR2 memory operating at the same external data bus clock rate as DDR results in DDR2 being able to provide the same bandwidth but with better latency. Alternatively, DDR2 memory operating at twice the external data bus clock rate as DDR may provide twice the bandwidth with the same latency. The best-rated DDR2 memory modules are at least twice as fast as the best-rated DDR memory modules. The maximum capacity on commercially available DDR2 DIMMs is 8GB, but chipset support and availability for those DIMMs is sparse and more common 2GB per DIMM are used

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=0uKdlNT0EMU

Important Books/Journals for further learning including the page nos.:

- Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 Page No: 70-75

Course Faculty

Verified by HOD

LECTURE HANDOUTS

ECE **II/IV**

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : II –8051 MICROCONTROLLER INTERFACING Date of Lecture:

Topic of Lecture: Stepper Motor

Introduction :

A **stepper motor**, also known as **step motor** or **stepping motor**, is a brushless DC electric motor that divides a full rotation into a number of equal steps.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Stepper Motor

Detailed content of the Lecture:

Stepper motor

A **stepper motor**, also known as **step motor** or **stepping motor**, is a brushless DC electric motor that divides a full rotation into a number of equal steps. The motor's position can be commanded to move and hold at one of these steps without any position sensor for feedback (an open-loop controller), as long as the motor is correctly sized to the application in respect to torque and speed.

Switched reluctance motors are very large stepping motors with a reduced pole count, and generally are closed-loop commutated.

Mechanism:

Brushed DC motors rotate continuously when DC voltage is applied to their terminals. The stepper motor is known for its property of converting a train of input pulses (typically square waves) into a precisely defined increment in the shaft's rotational position. Each pulse rotates the shaft through a fixed angle.

Stepper motors effectively have multiple "toothed" electromagnets arranged as a stator around a central rotor, a gear-shaped piece of iron. The electromagnets are energized by an external driver circuit or a micro controller. To make the motor shaft turn, first, one electromagnet is given power, which magnetically attracts the gear's teeth. When the gear's teeth are aligned to the first electromagnet, they are slightly offset from the next electromagnet. This means that when the next electromagnet is turned on and the first is turned off, the gear rotates slightly to align with the next one. From there the process is repeated. Each of those rotations is called a "step", with an integer number of steps making a full rotation. In that way, the motor can be turned by a precise angle.

The circular arrangement of electromagnets is divided into groups, each group called a phase, and there is an equal number of electromagnets per group. The number of groups is chosen by the designer of the stepper motor. The electromagnets of each group are interleaved with the electromagnets of other groups to form a uniform pattern of arrangement. For example, if the stepper motor has two groups identified as A or B, and ten electromagnets in total, then the grouping pattern would be ABABABABAB.

Electromagnets within the same group are all energized together. Because of this, stepper motors with more phases typically have more wires (or leads) to control the motor.

Types:

There are three main types of stepper motors:

1. Permanent magnet stepper
2. Variable reluctance stepper
3. Hybrid synchronous stepper

Permanent magnet motors use a permanent magnet (PM) in the rotor and operate on the attraction or repulsion between the rotor PM and the stator electromagnets.

Pulses move the rotor in discrete steps, CW or CCW. If left powered at a final step a strong detent remains at that shaft location. This detent has a predictable spring rate and specified torque limit; slippage occurs if the limit is exceeded. If current is removed a lesser detent still remains, therefore holding shaft position against spring or other torque influences. Stepping can then be resumed while reliably being synchronized with control electronics.

Variable reluctance (VR) motors have a plain iron rotor and operate based on the principle that minimum reluctance occurs with minimum gap, hence the rotor points are attracted toward the stator magnet poles. Whereas hybrid synchronous are a combination of the permanent magnet and variable reluctance types, to maximize power in a small size.[2]

VR motors have power on detents but do not have power off detents.

Video Content / Details of website for further learning (if any):

https://en.wikipedia.org/wiki/Stepper_motor

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 Page No: 109-131

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

LECTURE HANDOUTS

L17

ECE

II/IV

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : II –8051 MICROCONTROLLER INTERFACING Date of Lecture:

Topic of Lecture: Waveform

Introduction :

In electronics, acoustics, and related fields, the **waveform** of a signal is the shape of its graph as a function of time, independent of its time and magnitude scales and of any displacement in time.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Waveform
- Stepper Motor

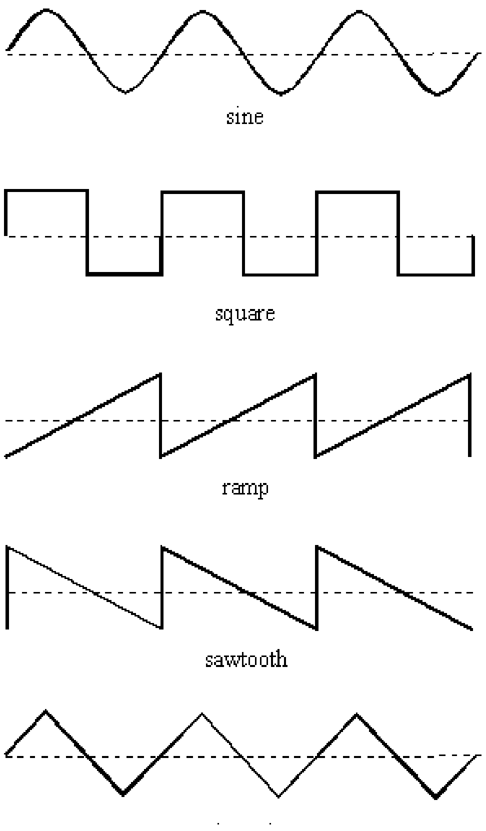
Detailed content of the Lecture:

In electronics, acoustics, and related fields, the **waveform** of a signal is the shape of its graph as a function of time, independent of its time and magnitude scales and of any displacement in time.

In electronics, the term is usually applied to periodically varying voltages, currents, or electromagnetic fields. In acoustics, it is usually applied to steady periodic sounds—variations of pressure in air or other media. In these cases, the waveform is an attribute that is independent of the frequency, amplitude, or phase shift of the signal. The term can also be used for non-periodic signals, like chirps and pulses.

The waveform of an electrical signal can be visualized in an oscilloscope or any other device that can capture and plot its value at various times, with a suitable scales in the time and value axes. The electrocardiograph is a medical device to record the waveform of the electric signals that are associated with the beating of the heart; that waveform has important diagnostic value. Waveform generators, that can output a periodic voltage or current with one of several waveforms, are a common tool in electronics laboratories and workshops.

The waveform of a steady periodic sound affects its timbre. Synthesizers and modern keyboards can generate sounds with many complicated waveforms.



Some AC waveforms are irregular or complicated. Square or sawtooth waves are produced by certain types of electronic oscillators, and by a low-end UPS (uninterruptible power supply) when it is operating from its battery. Irregular AC waves are produced by audio amplifiers that deal with analog voice signals and/or music.

The sine wave is unique in that it represents energy entirely concentrated at a single frequency. An ideal, unmodulated wireless signal has a sine waveform, with a frequency usually measured in megahertz (MHz) or gigahertz (Ghz). Household utility current has a sine waveform with a frequency of 60 Hz in most countries including the United States, although in some countries it is 50 Hz.

Video Content / Details of website for further learning (if any):

https://whatis.techtarget.com/definition/waveform

Important Books/Journals for further learning including the page nos.:

- Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 Page No: (239-240)

Course Faculty
Verified by HOD

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : II –8051 MICROCONTROLLER INTERFACING Date of Lecture:

Topic of Lecture: ADC Sensor Interfacing

Introduction :

Analog-to-digital converters are widely used in data acquisition. Basically, an ADC converts analog signal into digital signal. This is required because in the physical world we live every quantity we interact with is analog in nature.

Prerequisite knowledge for Complete understanding and learning of Topic:

- ADC
- ADC Sensor Interfacing

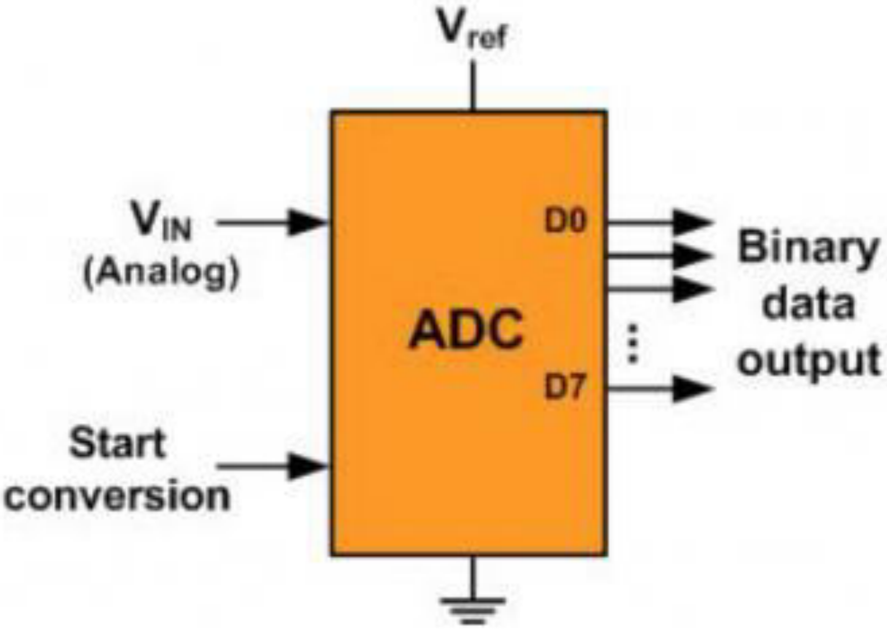
Detailed content of the Lecture:

Analog-to-digital converters are widely used in data acquisition. Basically, an ADC converts analog signal into digital signal. This is required because in the physical world we live every quantity we interact with is analog in nature. By physical quantities, i mean quantities like temperature, humidity, weight, velocity pressure etc. However, digital computers use discrete or binary values only, this brings us to transducers. A transducer is a device used to convert physical quantities to electrical signal i.e. voltage and current. When this transducer is used to generate output then it is known as sensor. Hence we have temperature sensor, pressure sensor, humidity sensor and so on.

Resolution of an ADC

The ADC resolution is defined as the smallest change the ADC can measure.
An ADC usually has n-bits resolution.
where n = 8,10,12,16 or 24

The higher the resolution, the smaller the change the ADC is able to measure. The resolution of the ADC is set during the design of the ADC meaning we as embedded system designers and developers do have direct control over the resolution . However, we do have an indirect means to control the resolution. We can control the smallest amount of voltage we want the ADC to increment on . This is know as is known as the step size . We can achieve this by manipulating the reference voltage, Vref. In the block diagram presented below we can see that the ADC has two voltage sources, one is labelled Vin and the other Vref. Vin, is the input voltage i.e. where we connect the pin of the sensor and Vref is the reference voltage we want to use to adjust the step size.



Video Content / Details of website for further learning (if any):

https://cortex-m.com/adc-analog-sensors/

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 Page No: (255-260)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L19

ECE

II/IV

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : III – PIC MICROCONTROLLER

Date of Lecture:

Topic of Lecture: Introduction to PIC Microcontrollers

Introduction :

PIC is a Peripheral Interface Microcontroller which was developed in the year 1993 by the General Instruments Microcontrollers. It is controlled by software and programmed in such a way that it performs different tasks and controls a generation line. PIC microcontrollers are used in different new applications such as smartphones, audio accessories, and advanced medical devices.

Prerequisite knowledge for Complete understanding and learning of Topic:

Detailed content of the Lecture:

PIC is a Peripheral Interface Microcontroller which was developed in the year 1993 by the General Instruments Microcontrollers. It is controlled by software and programmed in such a way that it performs different tasks and controls a generation line. PIC microcontrollers are used in different new applications such as smartphones, audio accessories, and advanced medical devices.

- PIC devices are popular with both industrial low cost developers and hobbyists due to their , wide availability, large user base, extensive collection of application notes, availability of low cost or free development tools, serial programming, and re-programmable flash-memory capability.
- programmer/debugger hardware under the MPLAB and Picklist series.
- There are many PICs available in the market ranging from PIC16F84 to PIC16C84. These types of PICs are affordable flash PICs. Microchip has recently introduced flash chips with different types, such as 16F628, 16F877, and 18F452. The 16F877 costs twice the price of the old 16F84, but it is eight times more than the code size, with more RAM and much more I/O pins, a UART, A/D converter and a lot more features.

Video Content / Details of website for further learning (if any):
https://www.youtube.com/watch?v=EY15NUjnM-A

Important Books/Journals for further learning including the page nos.:

R.S. Gaonkar, ‘Microprocessor Architecture Programming and Application’, with 8085, Wiley Eastern Ltd., New Delhi, 2013 (Page No: 135-154)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L20

ECE

II/IV

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : **Mrs.S.Punitha**

Unit : III - **PIC MICROCONTROLLER**

Date of Lecture:

Topic of Lecture: Architecture and pipelining

Introduction :
Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end. Pipelining increases the overall instruction throughput.

Prerequisite knowledge for Complete understanding and learning of Topic:

1. Architecture and pipelining processing

Detailed content of the Lecture:

- Pipeline Processor consists of a sequence of m data-processing circuits, called stages or segment.
- which collectively perform a single operation on a stream of data operands passing through them.
- Some processing takes place in each stage, but a final result is obtained only after an operand set has passed through the entire pipeline.
- As shown in figure, a stage $S_{(i)}$ contains a multiword input register or latch $R_{(i)}$, and a datapath circuit $C_{(i)}$, that is usually combinational.
- The $R_{(i)}$'s hold partially processed results as they move through the pipeline; they also serve as buffers that prevent neighbouring stages from interfering with one another.
- A common clock signal causes the $R_{(i)}$'s to change state synchronously. Each $R_{(i)}$'s to change state synchronously.

- Each R$_{(i)}$ receives a new set of input data D$_{(i-1)}$ from the preceding stage S$_{(i-1)}$ except for R$_{(1)}$ whose data is supplied from an external source.
- D$_{(i-1)}$ represents the results computed by C$_{(i-1)}$ during the preceding clock period. Once D$_{(i-1)}$ has been loaded into R$_{(i)}$, C$_{(i)}$ proceeds to D$_{(i-1)}$ to computer a new data set D$_{(i)}$.
- Thus in each clock period, every stage transfers its previous results to the next stage and computers a new set of results

ARCHITECTURE OF PIPELINING DIAGRAM :

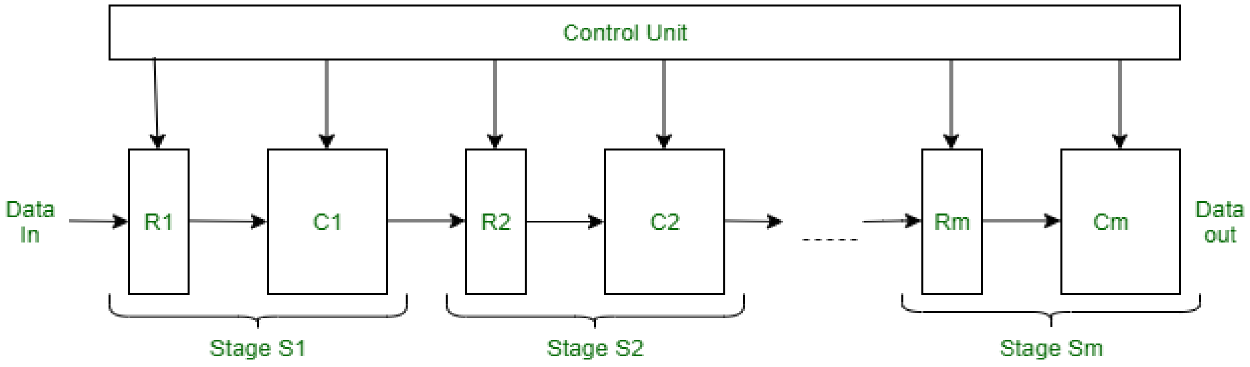


Figure - Structure of a Pipeline Processor

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=EY15NUjnM-A

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 Page no(37-55)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L21

ECE

II/IV

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : **Mrs.S.Punitha**

Unit : III - **PIC MICROCONTROLLER**

Date of Lecture:

Topic of Lecture: Program Memory Considerations

Introduction :

- PIC microcontroller has 13 bits of program memory address. Hence it can address up to 8k of program memory. The program counter is 13-bit. ... Similarly, while addressing 4k of memory, 12 bits are required.

Prerequisite knowledge for Complete understanding and learning of Topic:

1. Program Memory Organizations

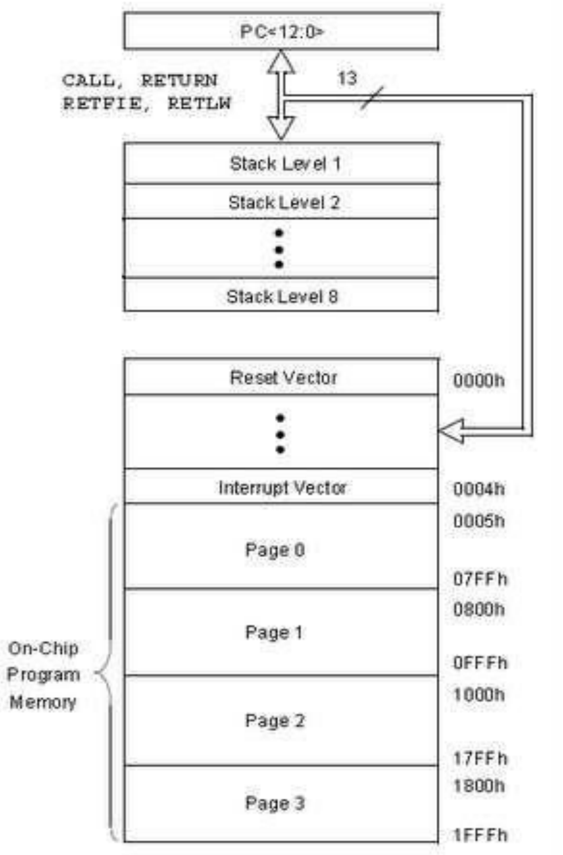
Detailed content of the Lecture:

- PIC microcontroller has 13 bits of program memory address.
- Hence it can address up to 8k of program memory.
- The program counter is 13-bit. PIC 16C6X or 16C7X program memory is 2k or 4k.
- While addressing 2k of program memory, only 11- bits are required.
- Hence two most significant bits of the program counter are ignored.
- Similarly, while addressing 4k of memory, 12 bits are required.
- Hence the MSB of the program counter is ignored.
- Program memory is a non-volatile memory. All modern PIC processors use a Flash memory technology that allows the program memory to be reprogrammed using a simple hardware interface.
- It is common to include some kind of programming connector on even a production product to allow for firmware updates if needed.
- Some older PIC parts that have a “C” after the first number are either one time programmable or may have a window is installed on the part to be UV light erasable.
- Normally a device programmer cannot only write to program memory but it can read the memory as well.
- All the PIC chips have a configuration bit that can be used to read-protect the program memory so the device programmer can no longer read the memory.
- This may be done for production products so code cannot be stolen. Once the protect is enabled the only way a device programmer can access the chip is to erase the whole chip.
- This is called code protection. Some device programmers also have an option to verify

the program is secure.

- This does not prevent the chip from being erased and reprogrammed but it does prevent it from being read. It should also be pointed out that for high volume projects, Microchip can preprogram the chips at the factory or you can have a parts distributer load the chips before delivery to you.

PROGRAM MEMORY MAP :



Video Content / Details of website for further learning (if any):

https://microchipdeveloper.com/8avr:memory

Important Books/Journals for further learning including the page nos.:

- Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (206-239)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L22

ECE

II/IV

Course Name with Code : 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : **Mrs.S.Punitha**

Unit : III - **PIC MICROCONTROLLER**

Date of Lecture:

Topic of Lecture: ADDRESSING MODES

Introduction :

- To know the working principal and data handling, we need to have clear knowledge on addressing modes of pic microcontroller. Now we can see that how we can categories different addressing modes of pic microcontroller. In PIC micro controller, it having mainly five addressing modes.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Immediate addressing mode
- Register operand addressing mode
- Memory operand addressing mode
- Direct addressing
- Indirect addressing.

Detailed content of the Lecture:

Immediate addressing mode:

In this addressing mode, the operand is a number or constant not an address as **MOVLW 43h**, the operand here is data not address. So in this addressing mode of pic microcontroller data is directly transfer. And data is immediate after the opcode. That is why this type of addressing is called immediate addressing. This way is fast in execution.

Register operand addressing mode:

In this addressing mode, the operand is a Register which holds the data to be execute. Register operand addressing mode deals with the registers like: CLR W

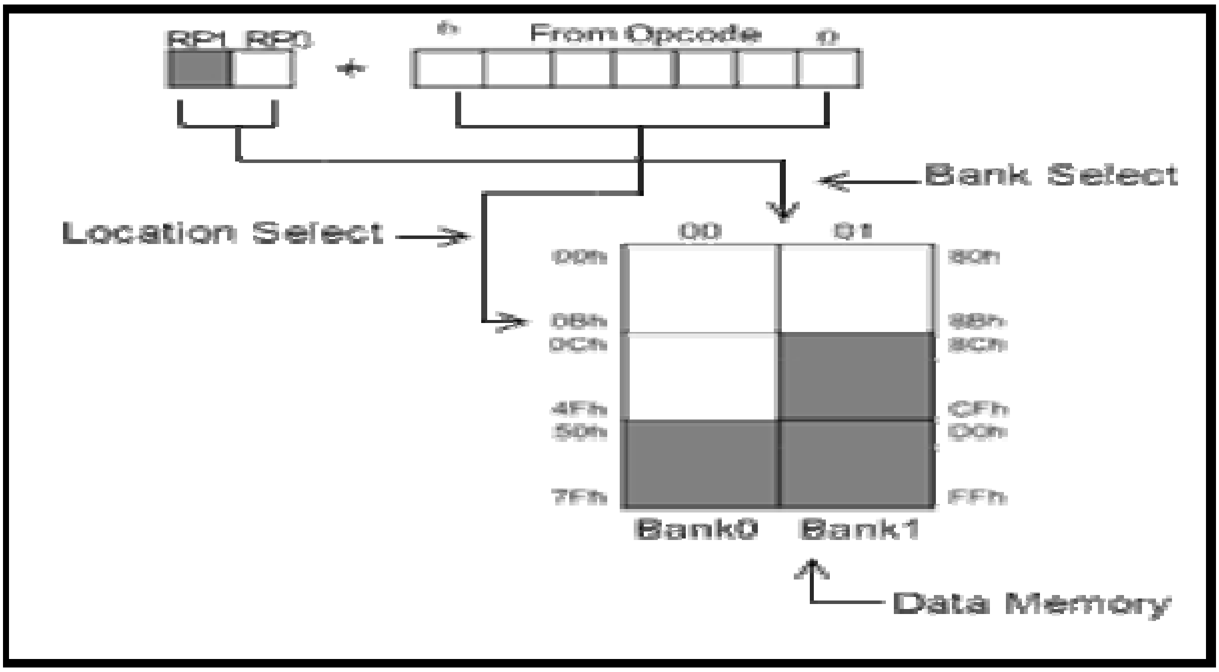
Memory operand addressing mode :

In this addressing mode, the operand is an address of Memory location which holds the data to be execute. Again memory operand addressing mode is under two category

1) Direct addressing like CLRF 13h. We deal with the address or the memory location.
2) Indirect addressing. we use in it INDF and FSR registers.

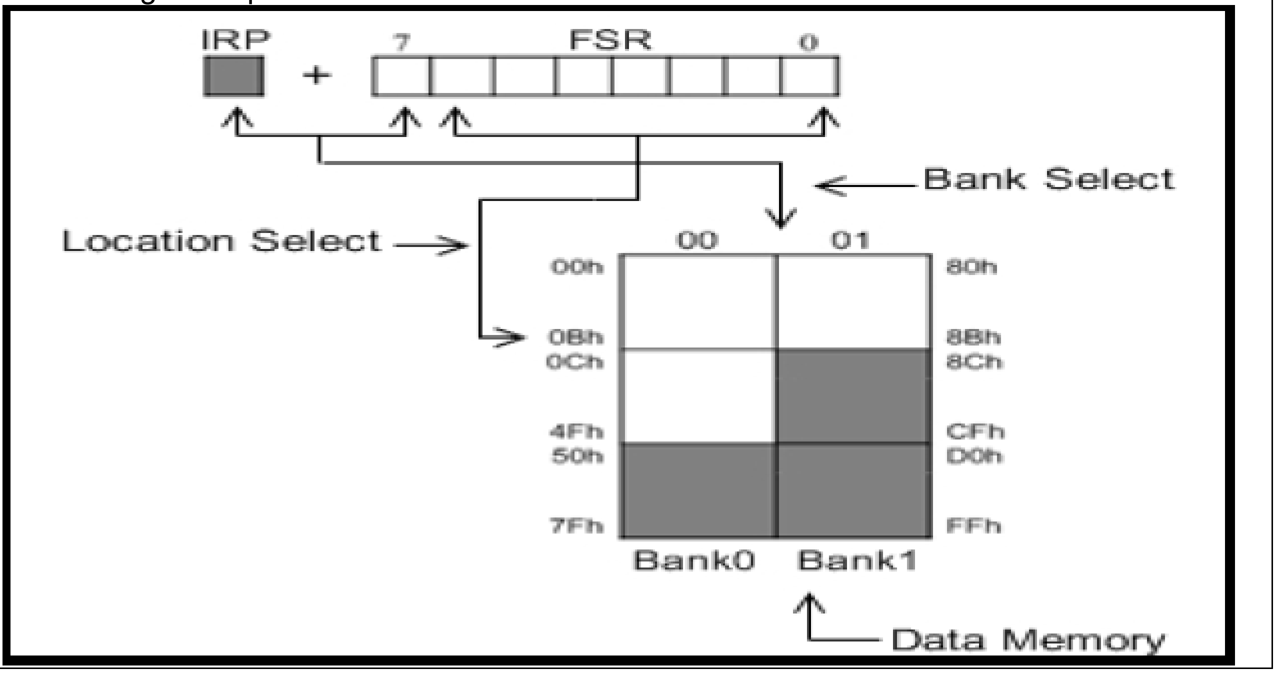
Direct addressing:

Direct Addressing is done through a 9-bit address. This address is obtained by connecting 7th bit of direct address. By using an instruction with two bits (RP1, RP0) from STATUS register. this is shown on bellow Figure . Any access to SFR registers can be an example of direct addressing.



Indirect addressing:

It does not take an address from an instruction. But it derives from IRP bit of STATUS and FSR registers. Addressed location is accessed through INDF register. And INDF register in fact holds the address indicated by the FSR. Indirect addressing is very convenient for manipulating data arrays located in GPR registers. In this case, it is necessary to initialise FSR register with a starting address of the array, and the rest of the data can be accessed by increment the FSR register. Figure shows the indirect addressing concept.



Video Content / Details of website for further learning (if any):
https://www.youtube.com/watch?v=_CH4cm5PhK8

Important Books/Journals for further learning including the page nos.:

- Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (252-260)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L23

ECE

II/IV

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : **Mrs.S.Punitha**

Unit : III - **PIC MICROCONTROLLER**

Date of Lecture:

Topic of Lecture: CPU Registers

Introduction :

- A register is a place inside the PIC which used to read or write the data/program. The memory of the PIC is divided into a series of registers. Each of the registers has its own address and memory locations. These addresses are normally denoted by using hexadecimal numbers

Prerequisite knowledge for Complete understanding and learning of Topic:

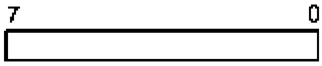
1. CPU Registers of pic microcontroller

Detailed content of the Lecture:

The following section will explain the registers commonly used by the PIC CPU.

Working Register:

W, the working register, is used by many instructions as the source of an operand.
This is similar to accumulator in 8051.
It may also serve as the destination for the result of the instruction execution.
It is an 8 - bit register.

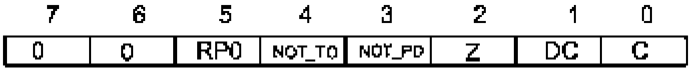


W, Working register

STATUS Register:

The STATUS register is an 8-bit register that stores the status of the processor. This also stores carry, zero and digit carry bits.

STATUS - address 03H, 83H.



7	6	5	4	3	2	1	0
0	0	RP0	NOT_TO	NOT_PD	Z	DC	C

C = Carry bit

DC = Digit carry (same as auxiliary carry)

Z = Zero bit

NOT_TO and NOT_PD - Used in conjunction with PIC's sleep mode

RP0- Register bank select bit used in conjunction with direct addressing mode.

FSR Register

(File Selection Register, address = 04H, 84H)

FSR is an 8-bit register used as data memory address pointer. This is used in indirect addressing mode.

INDF Register

(INDirect through FSR, address = 00H, 80H)

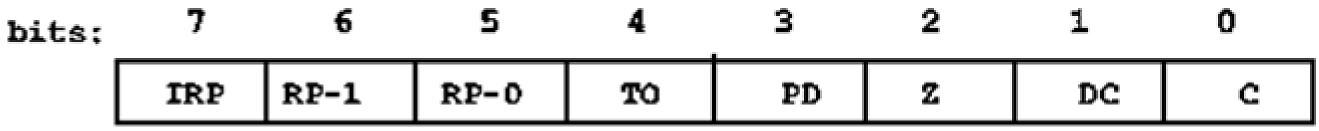
INDF is not a physical register. Accessing INDF access is the location pointed to by FSR in indirect addressing mode.

PCL Register

(Program Counter Low Byte, address = 02H, 82H)

PCL is actually the lower 8-bits of the 13-bit program counter. This is a both readable and writable register. **STATUS REGISTER:**

Following Figure is a bitmap of the STATUS register.



bits:	7	6	5	4	3	2	1	0
	IRP	RP-1	RP-0	TO	PD	Z	DC	C

bit 7 **IRP**: Register Bank Select bit (used for indirect addressing)

1 = Bank 2, 3 (0x100 - 0x1ff)

0 = Bank 0, 1 (0x000 - 0xff)

For devices with only Bank0 and Bank1 the IRP bit is reserved, always maintain this bit clear.

Bit 6:5 **RP1:RP0:**

Register Bank Select bits (used for directaddressing)

11 = Bank 3 (0x180 - 0x1ff)

10 = Bank 2 (0x100 - 0xx17f)

01 = Bank 1 (0x80 - 0xff)

00 = Bank 0 (0x00 - 0x7f)

Each bank is 128 bytes. For devices with only Bank0 and Bank1 the IRP bit is reserved, always maintain this bit clear.

Bit 4 **TO:** Time-out bit

1 = After power-up, CLRWDT instruction, or SLEEP instruction

0 = A WDT time-out occurred

Bit 3 **PD** Power-down bit

1 = After power-up or by the CLRWDT instruction

0 = By execution of the SLEEP instruction

Bit2 **Z:** Zero bit

1 = The result of an operation is zero

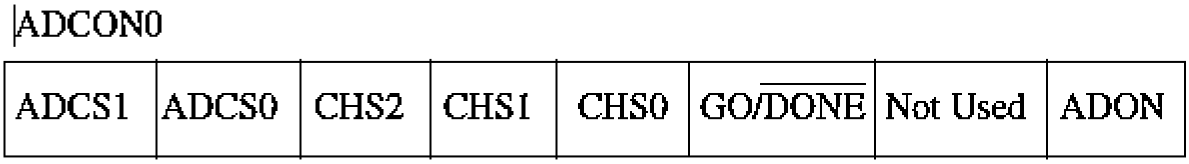
0 = The result of an operation is not zero

A/D Registers:

Analog-to-digital registers are used specifically to control the A/D ports. On most A/D equipped PICs, the A/D ports will be included on Port A only. Some of the larger PICs also add more A/D ports by using Port E.

A/D structure:

The A/D structure in the PIC uses three registers for access and control: the “A/D control register 0” (ADCON0), the “A/D control register 1” (ADCON1), and the “A/D result register” (ADRES). The ADCON0 register is really more of a control register while ADCON1 is a setup register.



ADCON0

ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	Not Used	ADON

ADCS1-0—A/D Conversion Clock Select

These two bits allow you to pick from four different clock sources. The clock signal is used in the sample and hold A/D circuitry inside the PIC. The best choice is the internal RC oscillator since it runs independent of the external crystal/ Resonator.

The other choices are for more precise measurements and require a lot of specific calculations—more calculations than the average Pic Basic user will want to deal with.

The bit selections are:

00: External Oscillator / 2

01: External Oscillator / 8

10: External Oscillator / 32

11: Internal RC Oscillator

CHS2-0—Analog Channel Select

These bits choose which A/D port you want to read within your program. You will have to select this at the beginning of your PBC A/D routine. PBPro automatically selects this when you use the ADCIN command.

CHS2-0 select as follows:

000: Channel 0 (A0 pin)

001: Channel 1 (A1 pin)

010: Channel 2 (A2 pin)

011: Channel 3 (A3 pin)

100: Channel 4 (A5 pin)

101: Channel 5 (E0 pin)

110: Channel 6 (E1 pin)

111: Channel 7 (E2 pin)

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=OJ_LPCVYkKM

Important Books/Journals for further learning including the page nos.:

- Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (260-262)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L24

ECE

II/IV

Course Name with Code : 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : III - PIC MICROCONTROLLER

Date of Lecture:

Topic of Lecture: Instruction Sets & simple operations

Introduction :

- Instruction sets are the source codes that are written by the programmer for performing the desired operations in a PIC chip. These codes can be usually written in any of the programming languages such as C, C++, assembly languages, and so on. The instruction set commands are pre-determined for each and every function with its own command syntaxes and are executed by the PIC chip.

Prerequisite knowledge for Complete understanding and learning of Topic:

1. Instruction Sets & simple operations of pic microcontroller

Detailed content of the Lecture:

The instruction sets in PIC is developed by the basis of RISC structure. The instruction sets can be classified into 5 separate categories (depends on the basis of type of operation). They are

.Data Transfer Group Instructions in PIC:

Data transfer group instructions are one of the major instructions widely used in PIC programming for data transfer purposes. By using these types of instructions, we can move or change the value (bits) from one location to another.

Data Transfer Instructions with Syntaxes

1. MOVLW

"MOVLW" instruction is used to write constant in W register (move the value from literal to W register).

- Syntax:

Label MOVLW k

- Description:

8 bit constant is written in W reg.

- Operation:

k to (W)

- Operand:

0 < k < 255

- No. of words: 1
- No. cycles: 1
- Flags: Nil

2. MOVWF
MOVWF instruction is used to move the data (bits) from W register to flag register F (copy/move the value from W register to F register).

- Syntax:

Label MOVWF f

- Description:

Content of W is copied into f register (flag register).

- Operation:

W to (f) (W register to Flag register)

- Operand:

0 < f < 127

- No. of words: 1
- No. cycles: 1
- Flags: Nil

3. MOVF
MOVF command is used for copy/move the contents (bits) in the flag register to D register (Copy f to d register).

D register (destination register) is a special register which can be switchable in its destination according to its status. If the status of D register is '0', the destination is W register and if the status is '1', the destination of D register became F register (flag)

- Syntax:
- Label MOVF f,d
- escription:

Content of f is copied into destination.

If d = 0, the destination is W register

If d = 1, the destination is f register

- Operation:
- Operand:

0 < k < 127

- No. of words: 1
- No. cycles: 1
- Flags: Z

4. CLRW

CLRW is a clearing instruction that helps to reset the values of W register to ‘0’ (write ‘0’ in W register).

- Syntax:

Label CLRW

- Description:

Zero is copied into W register
Z flag in status register is set to one

- Operation:

0 to (W)

- Operand: nil
- No. of words: 1
- No. cycles: 1
- Flags: Z

5. CLRF f

CLRF f Write ‘0’ in F register that helps to reset the current status to ‘0’

- Syntax:

  Label CLRF f

- Description:

Zero is copied into f register

Z flag in status register is set to one

- Operation:

0 to (f)

- Operand: nil
- No. of words: 1
- No. cycles: 1
- Flags: Z

6. SWAPF

SWAPF used for swap (interchanging functions) functions which Swap the nibbles (4bits).The destination of this function depends on the destination register status.

- Syntax:

Label SWAPF f, d

- Description:

Upper, Lower nibbles are exchanged

If d = 0, the destination is W register

If d = 1, the destination is f register

- Operation:

f (0:3) to d(4:7) and f(4:7) to d(0:3)

- Operand:

0 < f < 127

- No. of words: 1
- No. cycles: 1
- Flags: —

ARITHMETICAL AND LOGICAL OPERATIONS:

Arithmetic and logic operation group instructions are used for performing all arithmetic operations and logic operations. By using these types of instructions, the PIC chip can easily perform all arithmetic and logic operations inside the micro controller (PIC). The arithmetic operations are addition (ADD), subtraction (SUB), multiplication (MUL), division (DIV) and logical operations are AND, OR, NOT, XOR, and so on. The basic Arithmetical and Logical operations that are performed by a PIC is given below.

1) ADDLW

“ADDLW” instruction is used for performing addition operation (adding a constant with W register). By using this instruction, we can add two bits easily and the result value can be stored in another register or memory location.

- **Syntax:** Label ADDLW k
- **Description:** Given constant is added with W reg.
- **Operation:** (w) + k to w
- **Operand:** 0 < k < 255
- **No. of words:** 1
- **No. cycles:** 1
- **Flags:** C, DC, Z

2) ADDWF

“ADDWF” is also used for performing the addition operation. This ADDWF instruction adds the constant with W register.

- **Syntax:** Label ADDWF f, d
- **Description:** Add W reg. content with f register
- **Operation:** (w) + (f) to w if d = 0 and (w) + (f) to f if d = 1
- **Operand:** 0 < f < 127
- **No. of words:** 1
- **No. cycles:** 1
- **Flags:** C, DC, Z

3) SUBLW

“SUBLW” used for performing subtraction function which can be subtracting two values and can be stored to another memory location. This instruction helps to Subtract W content from given constant.

- **Syntax**: Label SUBLW k
- **Description:** W reg. content is subtracted from k
- **Operation:** k – (w) to w
- **Operand:** 0 < k < 255
- **No. of words:** 1
- **No. cycles:** 1
- **Flags:** C, DC, Z

4) SUBWF

SUBWF is used for performing subtraction operation. In SUBLW, this instruction Subtracts W content from f register.

- **Syntax:** Label SUBWF f
- **Description:** W reg. content is subtracted from f
- **Operation:** f – (w) to w if d = 0 and f – (w) to f if d = 1
- **Operand:** 0 < f < 127
- **No. of words:** 1
- **No. cycles:** 1
- **Flags:** C, DC, Z

5) ANDLW

ANDLW is a logical instruction which used for performing Logic AND. By using this instruction helps AND the constant with W.

- **Syntax:** Label ANDLW k
- **Description:** Given constant is .and. with W reg.
- **Operation:** (w) .and. k to w
- **Operand:** 0 < k < 255
- **No. of words:** 1
- **No. cycles:** 1
- **Flags:** Z

Bit Operation Group Instructions in PIC:

Program Flow Control Group in PIC:

1) BTFSC

BTFSC is a special type program flow instruction which control the current program flow. Normally BTFSC Test the bit in f, skip if it is zero.

- **Syntax:** Label BTFSC f, b
- **Description:** Test the specified bit of register f, skip the next instruction if it is zero
- **Operation:** Skip the next instruction if f (b) = 0
- **Operand:** 0 < f < 127 and 0 < b < 7
- **No. of words**: 1
- **No. cycles:** 1 or 2 depends on bit value
- **Flags:** NIL

2) BTFSS

BTFSS also program flow control instruction. This instruction Test the bit in f, skip if it is one

- **Syntax:** Label BTFSS f, b
- **Description:** Test the specified bit of register f, skip the next instruction if it is one
- **Operation:** Skip the next instruction if f (b) = 1
- **Operand:** 0 < f < 127 and 0 < b < 7
- **No. of words:** 1
- **No. cycles:** 1 or 2 depends on bit value
- **Flags:** nil

3) INCFSZ

INCFSZ is a content increment command which Increment f content, skip if it is zero

- **Syntax:** Label INCFSZ f, d
- **Description:** Increment the f content, skip the next instruction if f is zero
- **Operation:** Skip the next instruction if f = zero
- **Operand:** (f) + 1 = w if d = 0 and (f) + 1 = f if d = 1
- **No. of words:** 1
- **No. cycles:** 1 or 2 depends on bit value
- **Flags:** nil

4) DECFSZ

DECFSZ command Decrement f content, skip if it is zero

- **Syntax:** Label DECFSZ f, d
- **Description:** Decrement the f content, skip the next instruction if f is zero
- **Operation:** Skip the next instruction if f = zero
- **Operand:** (f) – 1 = w if d = 0 and (f) – 1 = f if d = 1
- **No. of words:** 1
- **No. cycles:** 1 or 2 depends on bit value

- **Flags:** nil

5) GOTO

GOTO instruction is used for Jump to specified address locations.

- **Syntax:** Label GOTO Label
- **Description:** Unconditional jump to specified label
- **Operation:** k to PC(10:0), PCLATH(4:3) to PC(12:11)
- **Operand:** 0 < k < 2048
- **No. of words:** 1
- **No. cycles:** 2
- **Flags:** nil

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=2aCR5l26jJY

Important Books/Journals for further learning including the page nos.:

Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (291-293)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

L25

LECTURE HANDOUTS

ECE

II/IV

Course Name with Code: 19ECC07 / MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : III - PIC MICROCONTROLLER

Date of Lecture:

Topic of Lecture: Pic Microcontroller Architecture

Introduction :
The PIC microcontroller is based on RISC architecture. Its memory architecture follows the Harvard pattern of separate memories for program and data, with separate buses.

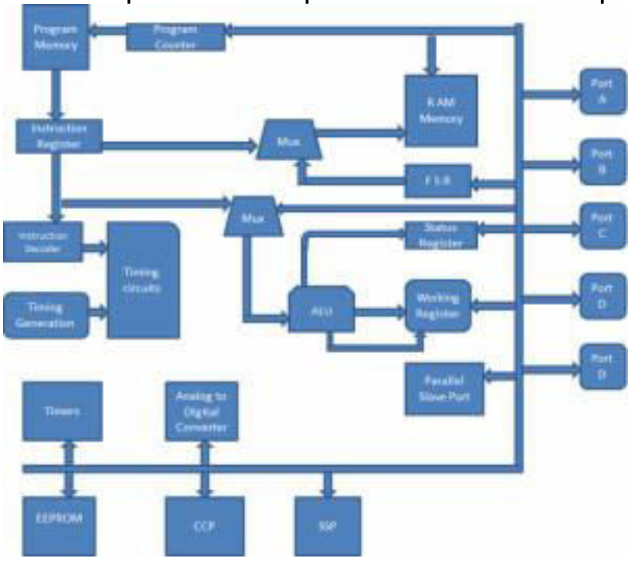
Prerequisite knowledge for Complete understanding and learning of Topic:

1. Pic Microcontroller Architecture

Detailed content of the Lecture:

Architecture of PIC Microcontroller

The PIC microcontroller is based on RISC architecture. Its memory architecture follows the Harvard pattern of separate memories for program and data, with separate buses.



1. Memory Structure

The PIC architecture consists of two memories: Program memory and Data memory.

Program Memory: This is a 4K*14 memory space. It is used to store 13-bit instructions or the program code. The program memory data is accessed by the program counter register that holds the address of the program memory. The address 0000H is used as reset memory space and 0004H is used as interrupt memory space.

Data Memory: The data memory consists of the 368 bytes of RAM and 256 bytes of EEPROM. The 368 bytes of RAM consists of multiple banks. Each bank consists of general-purpose registers and special function registers.

The special function registers consist of control registers to control different operations of the chip resources like Timers, Analog to Digital Converters, Serial ports, I/O ports, etc. For example, the TRISA register whose bits can be changed to alter the input or output operations of the port A. The general-purpose registers consist of registers that are used to store temporary data and processing results of the data. These general-purpose registers are each 8-bit registers.

Working Register: It consists of a memory space that stores the operands for each instruction. It also stores the results of each execution.

Status Register: The bits of the status register denotes the status of the ALU (arithmetic logic unit) after every execution of the instruction. It is also used to select any one of the 4 banks of the RAM.

File Selection Register: It acts as a pointer to any other general-purpose register. It consists of a register file address, and it is used in indirect addressing.

Another general-purpose register is the program counter register, which is a 13-bit register. The 5 upper bits are used as PCLATH (Program Counter Latch) to independently function as any other register, and the lower 8-bits are used as the program counter bits. The program counter acts as a pointer to the instructions stored in the program memory.

EEPROM: It consists of 256 bytes of memory space. It is a permanent memory like ROM, but its contents can be erased and changed during the operation of the microcontroller. The contents into EEPROM can be read from or written to, using special function registers like EECON1, EECON, etc.

2. I/O Ports

PIC16 series consists of five ports, such as Port A, Port B, Port C, Port D, and Port E.

Port A: It is a 16-bit port, which can be used as an input or output port based on the status of the TRISA register.

Port B: It is an 8-bit port, which can be used as both an input and output port. 4 of its bits, when used as input, can be changed upon interrupt signals.

Port C: It is an 8-bit port whose operation (input or output) is determined by the status of the TRISC register.

Port D: It is an 8-bit port, which apart from being an I/O port, acts as a slave port for connection to the microprocessor bus.

Port E: It is a 3-bit port that serves the additional function of the control signals to the A/D converter.

3. Timers

PIC microcontrollers consist of 3 timers, out of which the Timer 0 and Timer 2 are 8-bit timers and the Time-1 is a 16-bit timer, which can also be used as a counter.

4. A/D Converter

The PIC Microcontroller consists of 8-channels, 10-bit Analog to Digital

Converter. The operation of the A/D converter is controlled by these special function registers: ADCON0 and ADCON1. The lower bits of the converter are stored in ADRESL (8 bits), and the upper bits are stored in the ADRESH register. It requires an analog reference voltage of 5V for its operation.

5. Oscillators

Oscillators are used for timing generation. PIC microcontrollers consist of external oscillators like crystals or RC oscillators. In the case of crystal oscillators, the crystal is connected between two oscillator pins, and the value of the capacitor connected to each pin determines the mode of operation of the oscillator. The different modes are low-power mode, crystal mode, and the high- speed mode. In the case of RC oscillators, the value of the Resistor and Capacitor determines the clock frequency. The clock frequency ranges from 30 kHz to 4 MHz.

6. CCP module:

A CCP module works in the following three modes:

Capture Mode: This mode captures the time of arrival of a signal, or in other words, captures the value of the Timer1 when the CCP pin goes high.

Compare Mode: It acts as an analog comparator that generates an output when the timer1 value reaches a certain reference value.

PWM Mode: It provides pulse width modulated output with a 10-bit resolution and programmable duty cycle.

Other special peripherals include a Watchdog timer that resets the microcontroller in case of any software malfunction and a Brownout reset that resets the microcontroller in case of any power fluctuation and others. For a better understanding of this PIC microcontroller, we are giving one practical project which uses this controller for its operation.

Street Light that Glows on Detecting Vehicle Movement

This LED street light control project is designed to detect the vehicle movement on the highway to switch on a block of street lights ahead of it, and to switch off the trailing lights to save energy. In this project, a PIC microcontroller programming is done by using embedded C or assembly language.

Video Content / Details of website for further learning (if any):

https://youtu.be/fozc9OPF_Nc

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 112-117)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L26

ECE

II/IV

Course Name with Code : 19ECC07 /MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : III - PIC MICROCONTROLLER

Date of Lecture:

Topic of Lecture: Pic family microcontroller

Introduction :

Device families. PIC microchips are designed with a Harvard architecture, and are offered in various device families. The baseline and mid-range families use 8-bit wide data memory, and the high-end families use 16-bit data memory.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Pic family microcontroller

Detailed content of the Lecture:

12Cxxx family

- 12/14 bit internal operations
- 33/35 instructions
- 0.4μs time instruction cycle time (minimum time for instruction execution)

PIC12F675

- high performance with flash memory
- • 12F675 has 1k of code space (program memory), 64 bytes of RAM and 128 bytes of EEPROM and runs up to 20 MHz clock speed 2011 Microcontrollers-... 2nd Ed. Raj Kamal Pearson Education
- High performance
- Flash memory
- 1k of code space (program memory)
- 64 bytes of RAM
- 128 bytes of EEPROM

- Runs up to 20 MHz clock speed 2011 Microcontrollers-... 2nd Ed. Raj Kamal Pearson Education 9

16C5xx family

- 12 bit internal operations
- 33 instructions
- 0.2µs (200 ns) instruction cycle time 2011 Microcontrollers-... 2nd Ed. Raj Kamal Pearson Education 10

16CFxx family

- 14 bit internal operations
- 35 instructions
- 0.2µs (200 ns) instruction cycle time 2011 Microcontrollers-... 2nd Ed. Raj Kamal Pearson Education 11

PIC 16F877A

- High performance
- Flash memory
- 8 k × 14 code space
- 368 bytes RAM
- 256 bytes of EEPROM
- Single-cycle (0.2µs) instructions for all except branch instruction [20 MHz clock]
- Branch takes two cycles 2011 Microcontrollers-... 2nd Ed. Raj Kamal Pearson Education 12

17C5xx family

- 16 bit internal operations
- 58 instructions
- 0.12µs time for instruction cycle time

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=A4gO2tZ7ek0

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 118-124)

Course Faculty

Verified by HOD

L27

LECTURE HANDOUTS

ECE

II/IV

Course Name with Code: 19ECC07 / MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : **Mrs.S.Punitha**

Unit : III - **PIC MICROCONTROLLER**

Date of Lecture:

Topic of Lecture: pipelining in pic microcontroller

Introduction :

Pipelining works best if fetch and execute cycles are always of the same duration, such as a RISC structure gives. This fairly simple design upgrade gives a doubling in execution speed! All PIC microcontrollers implement pipelining, which is one of the reasons for their comparatively high speed of operation.

Prerequisite knowledge for Complete understanding and learning of Topic:

1. pipelining in pic microcontroller

Detailed content of the Lecture:

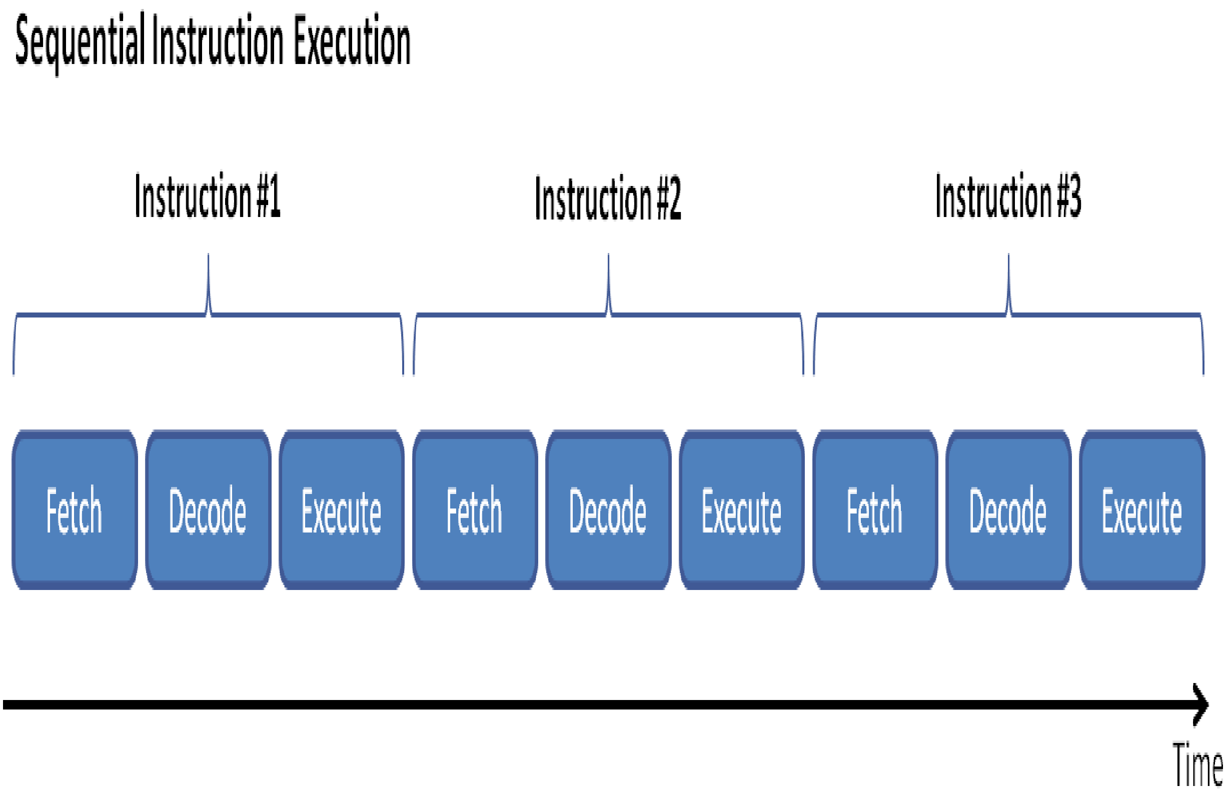
The control unit in all Central Processing Units (CPUs) follows the same basic instruction processing sequence:

1. **fetch** the instruction
2. **decode** the instruction
3. **execute** the instruction

On traditional CPUs, these phases are typically executed **sequentially** as shown:

Modern, high-performance CPUs (like MIPS®) use a technique called **Pipelining**, whereby these phases of instruction processing are executed in independent overlapping stages, as shown:

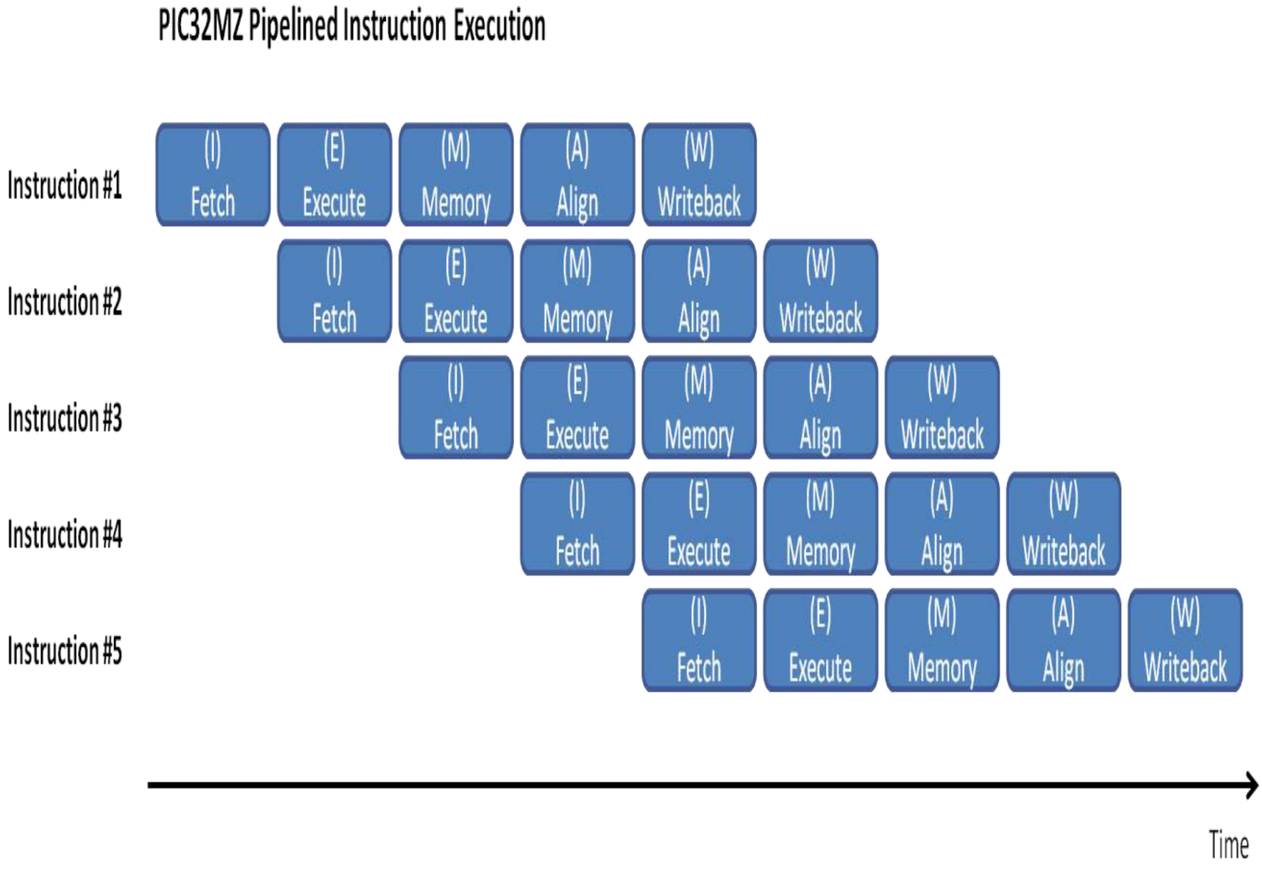
[[include :xsi:imagesrc:ras; margin: 0 auto;"]]



N-stage pipelines therefore have n-instructions at different stages of execution moving through the pipeline, similar to an automotive assembly line.

PIC32MZ Pipeline

The MIPS microAptive® MPU Core implements a five-stage pipeline as shown here:



- **I-STAGE: INSTRUCTION FETCH**
  - Fetch the instruction from cache/instruction SRAM. Increment the PC by four (4).
- **E-STAGE: EXECUTION**
  - Fetch operands from the register file. Begin ALU Operation. Calculate branch target address. Begin MDU operation.
- **M-STAGE: MEMORY FETCH**
  - ALU Op Completes. Data cache/SRAM access is performed for load/store operations. MDU operations proceed.
- **A-STAGE: ALIGN**
  - Align loaded data with its word boundary. Load data or MDU result to E-STAGE for bypassing.
- **W-STAGE: WRITEBACK**
  - For register-to-register or load instructions, the result is written back to register file..

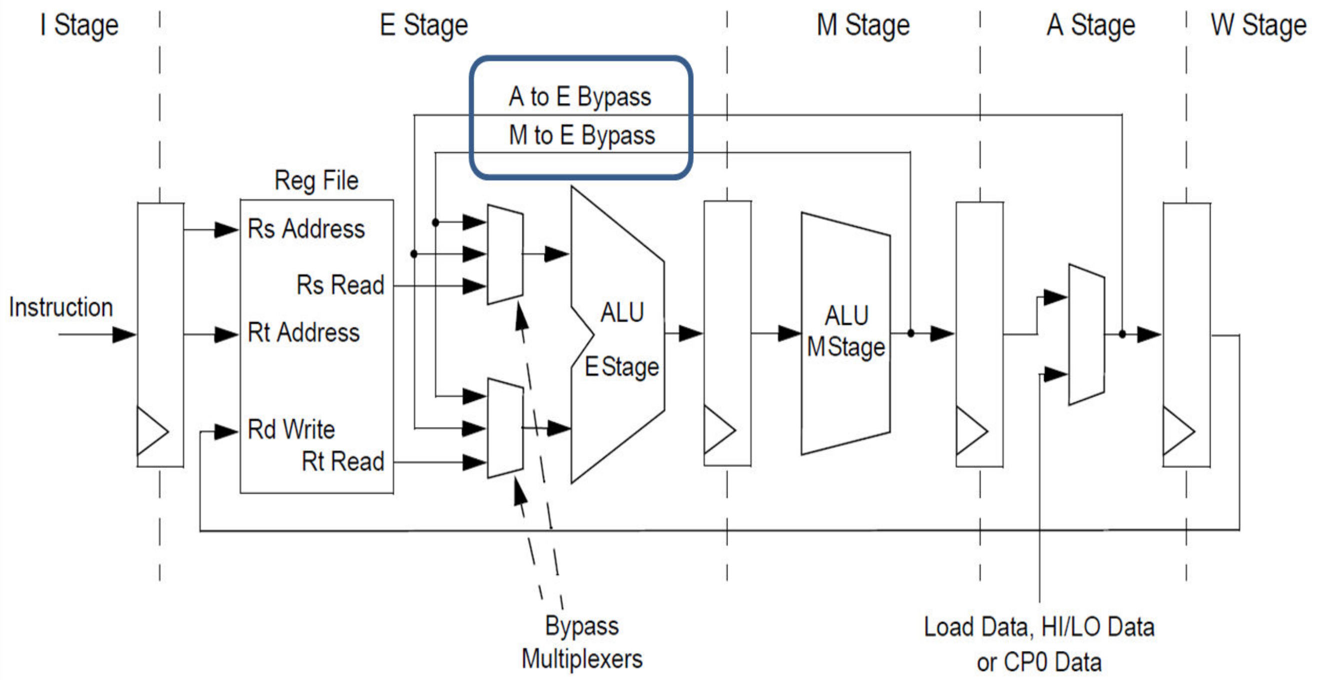
Data Hazards

The term **data hazard** refers to the situation where results from prior ALU operations are required before they have been written back to the register file.

In the following example, the result of the first operation (in register t3) is required as input for execution of the second operation:

The microAptiv® Microprocessor cores implement a “bypass” mechanism that allows the result of an operation to be sent directly to the instruction that needs it without having to write the result to the register, and then read it back.

The following diagram depicts the PIC32MZ Datapath, showing the **A » E** and **M » E** stage bypass connections:



The following pipeline diagram depicts how the result (t3) can be made available after the M-STAGE to the following instruction without stalling the pipeline:

Control Hazards

Control hazards arise in pipelined CPU architectures whereby instructions that follow branch instructions are fetched by the control hardware, and (traditionally) are flushed if the branch is taken, reducing instruction throughput.

microAptiv® MCU cores implement a **delay slot** in the pipeline and include **branch target address calculation hardware in the E-stage of the pipeline**.

For branch operations, these result in:

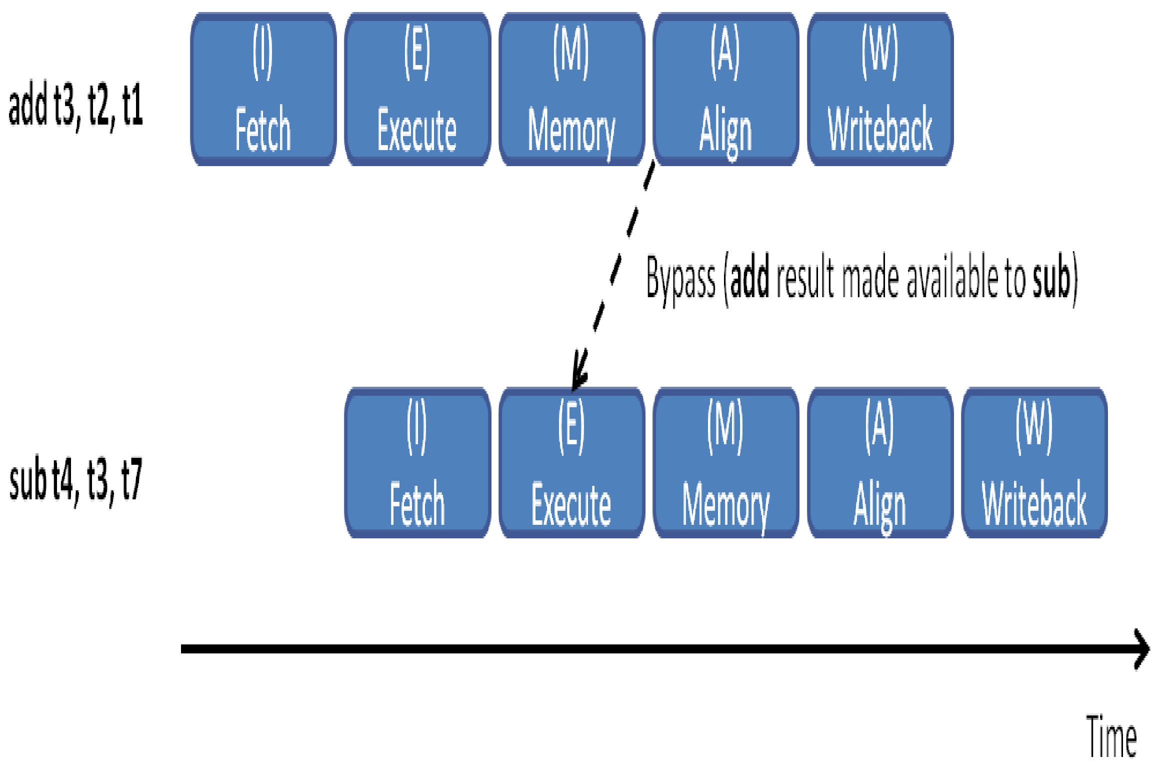
- Having an instruction slot available after all branch instructions to do useful work.
- Guaranteeing that the pipeline fetches either the 'branch taken' instruction or 'fall-through' instruction in the cycle immediately following the delay slot.

Since the branch cannot take effect until the second stage of the pipeline, we say that it is a **delayed branch** (i.e. the branch/jump will take place after the instruction following the branch is in the pipeline).

The following pipeline diagram demonstrates how either the *branch taken* instruction (label **L2**) or the *fall-through* instruction (label **L1**) is executed immediately after the delay slot instruction (**sw t0, 0(sp)**):

[[include :xsi:imagesrc:ras; margin: 0 auto;"]]

Active A >> E Stage Bypass on Data Hazard



add t3, t2, t1

(I) Fetch | (E) Execute | (M) Memory | (A) Align | (W) Writeback

Bypass (**add** result made available to **sub**)

sub t4, t3, t7

(I) Fetch | (E) Execute | (M) Memory | (A) Align | (W) Writeback

Time

Video Content / Details of website for further learning (if any):
https://www.youtube.com/watch?v=SWSTjQosOD0

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 132-135)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L28

ECE

II/IV

Course Name with Code: 19ECC07 / MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : IV – ARM 32 BIT MICROCONTROLLER

Date of Lecture:

Topic of Lecture: Thumb-2 technology and applications of ARM,

Introduction :

Thumb-2 technology is a major enhancement to the Thumb instruction set. It adds 32-bit instructions that can be freely intermixed with 16-bit instructions in a program. The additional 32-bit encoded Thumb instructions enable Thumb to cover most of the functionality of the ARM instruction set.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basics of Processor and Controller

Detailed content of the Lecture:

Thumb-2 technology

Thumb-2 technology is available in the ARMv6T2 and later architectures. However, the ARMv6-M architecture only uses a limited set of 32-bit Thumb instructions.

The additional 32-bit encoded Thumb instructions enable Thumb to cover most of the functionality of the ARM instruction set. The availability of 16-bit and 32-bit instructions enables Thumb-2 technology to combine the code density of earlier versions of Thumb with the performance of the ARM instruction set.

An important difference between the Thumb and ARM instruction sets is that most Thumb instructions are unconditional, whereas most ARM instructions can be conditional. Thumb-2 technology introduces a conditional execution instruction, `IT`, that is a logical if-then-else operation that you can apply to subsequent instructions to make them conditional.

ARM Thumb-2 technology features enhanced performance

ARM has introduced the Thumb-2 core technology, which is an instruction set that provides enhanced levels of performance, energy efficiency, and code density for a wide range of embedded applications.

The core technology is built on the foundation of ARM's Thumb code compression technology, retaining all of the compact code qualities and code compatibility with existing ARM solutions. It is a blended instruction set combining both 16-bit and 32-bit instructions designed to deliver the best balance of density and performance enabling new embedded and mobile devices that support feature-rich applications with longer battery life.

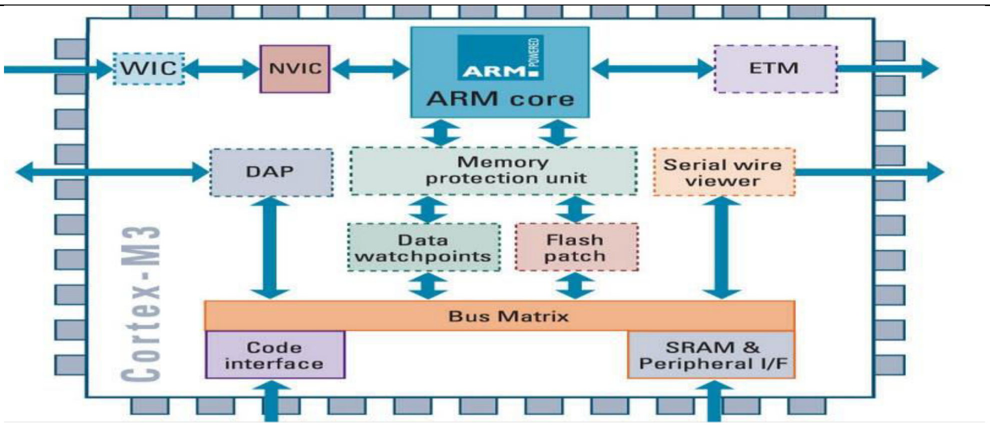
Richard Phelan, Embedded CPU Manager, said, "Thumb-2 core technology uses 26 percent less memory than pure 32-bit code to reduce system cost, and at the same time, Thumb-2 core technology delivers 25 percent better performance than 16-bit code alone enabling designers to save power by reducing clock speeds."

Applications of ARM[1]

The ARM microcontroller stands for Advance RISC Machine; it is one of the extensive and most licensed processor cores in the world. The first ARM processor was developed in the year 1978 by Cambridge University, and the first ARM RISC processor was produced by the Acorn Group of Computers in the year 1985. These processors are specifically used in portable devices like digital cameras, mobile phones, home networking modules and wireless communication technologies and other embedded systems due to the benefits, such as low power consumption, reasonable performance, etc.

The ARM architecture processor is an advanced reduced instruction set computing [RISC] machine and it’s a 32bit reduced instruction set computer (RISC) microcontroller. It was introduced by the Acron computer organization in 1987. This ARM is a family of microcontroller developed by makers like ST Microelectronics, Motorola, and so on. The ARM architecture comes with totally different versions like ARMv1, ARMv2, etc., and, each one has its own advantage and disadvantages.

ARM Architecture



Video Content / Details of website for further learning :

https://www.youtube.com/watch?v=cP6NxivTY94

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 449-452)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

L29

LECTURE HANDOUTS

ECE

II/IV

Course Name with Code: 19ECC07 /MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : IV - ARM 32 BIT MICROCONTROLLER

Date of Lecture:

Topic of Lecture: architecture of ARM Cortex-M3

Introduction :

- The Cortex-M3 processor is specifically developed for high-performance, low-cost platforms for a broad range of devices including microcontrollers, automotive body systems, industrial control systems and wireless networking and sensors.

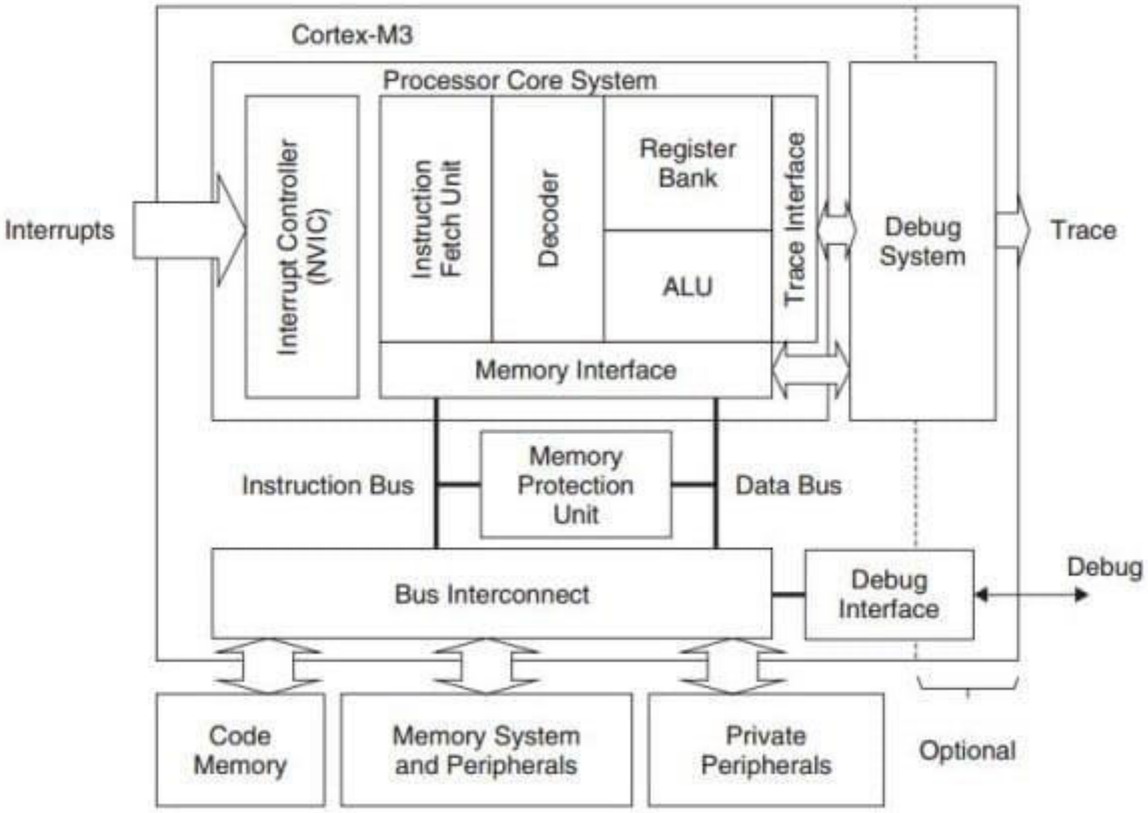
Prerequisite knowledge for Complete understanding and learning of Topic:

- Interfacing Concepts

Detailed content of the Lecture:

architecture of ARM Cortex-M3:

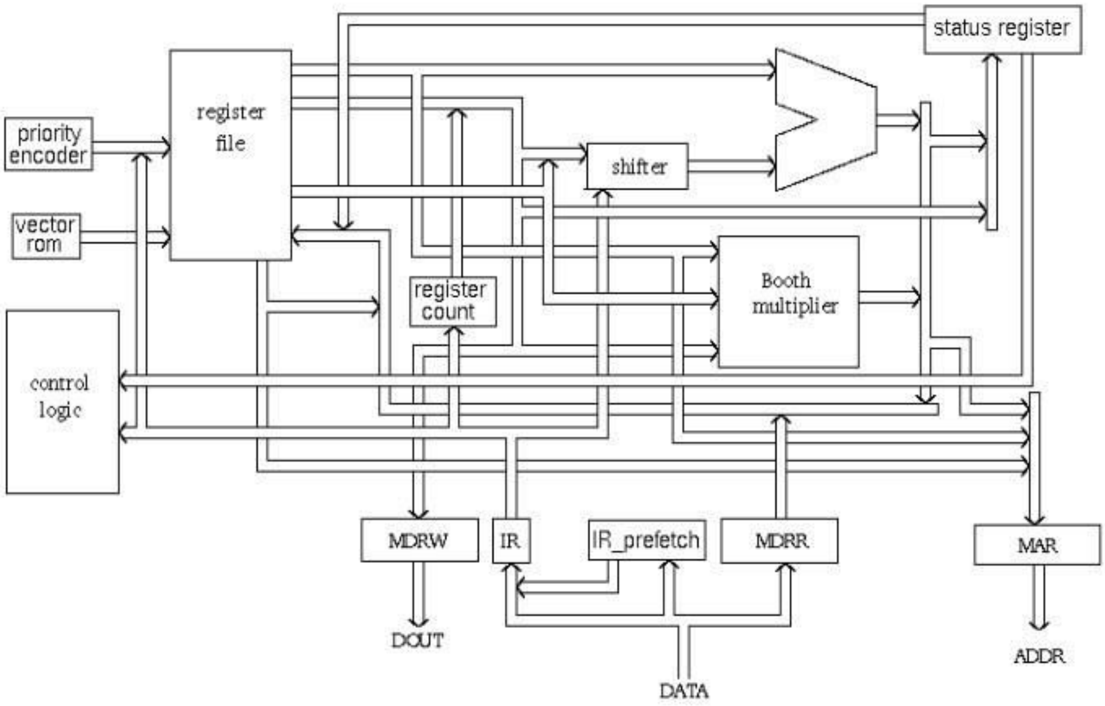
The Contex-M3 is 32-bit Microprocessor. It has 32-bit data path, 32-bit register bank and 32-bit memory interfaces. The Cortex-M3 offers many new features including Thumb-2 Instruction Set and very low power consumption, low interrupt latency etc.The LPC1768 is microcontroller belongs to Cortex-M3 core. The processor consists of Harvard Architecture, which means it has separate instruction bus and data bus. This allows instruction and data access take place at the same time which results in higher performance. However instruction and data buses share same memory for complex application where more memory is primary need



ARM Architecture

The ARM architecture processor is an advanced reduced instruction set computing [RISC] machine and it's a 32bit reduced instruction set computer (RISC) microcontroller. It was introduced by the Acron computer organization in 1987. This ARM is a family of microcontroller developed by makers like ST Microelectronics,Motorola, and so on. The ARM architecture comes with totally different versions like ARMv1, ARMv2, etc., and, each one has its own advantage and disadvantages.

ARM Block Diagram



Video Content / Details of website for further learning :

https://youtu.be/JPfG0UQd3x4

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 452-458)

Course Faculty

Verified by HOD

L30

LECTURE HANDOUTS

ECE

II/IV

Course Name with Code: 19ECC07 / MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : IV - ARM 32 BIT MICROCONTROLLER

Date of Lecture:

Topic of Lecture: Various units in the architecture

Introduction :

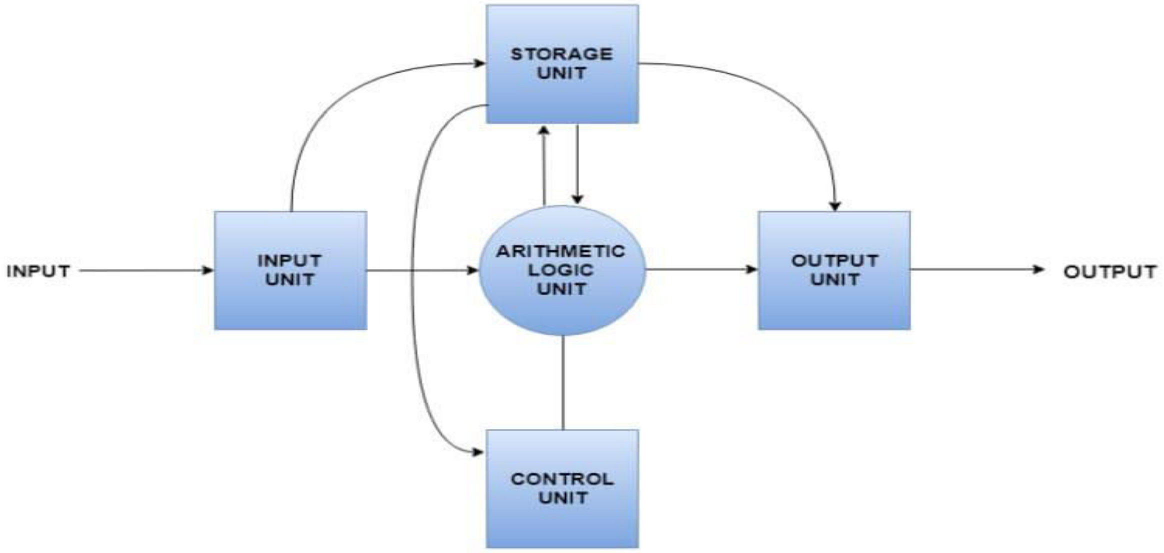
A computer system is basically a machine that simplifies complicated tasks. It should maximize performance and reduce costs as well as power consumption. The different components in the Computer System Architecture are Input Unit, Output Unit, Storage Unit, Arithmetic Logic Unit, Control Unit etc.

Prerequisite knowledge for Complete understanding and learning of Topic:

Interfacing Concepts

Detailed content of the Lecture:

A diagram that shows the flow of data between these units is as follows



The input data travels from input unit to ALU. Similarly, the computed data travels from ALU to output unit. The data constantly moves from storage unit to ALU and back again. This is because stored data is computed on before being stored again. The control unit controls all the other units as well as their data.

Details about all the computer units are

Input Unit:

The input unit provides data to the computer system from the outside. So, basically it links the external environment with the computer. It takes data from the input devices, converts it into machine language and then loads it into the computer system. Keyboard, mouse etc. are the most commonly used input devices.

Output Unit:

The output unit provides the results of computer process to the users i.e it links the computer with the external environment. Most of the output data is the form of audio or video. The different output devices are monitors, printers, speakers, headphones etc.

Storage Unit:

Storage unit contains many computer components that are used to store data. It is traditionally divided into primary storage and secondary storage.Primary storage is also known as the main memory and is the memory directly accessible by the CPU. Secondary or external storage is not directly accessible by the CPU. The data from secondary storage needs to be brought into the primary storage before the CPU can use it. Secondary storage contains a large amount of data permanently.

Arithmetic Logic Unit:

All the calculations related to the computer system are performed by the arithmetic logic unit. It can perform operations like addition, subtraction, multiplication, division etc. The control unit transfers data from storage unit to arithmetic logic unit when calculations need to be performed. The arithmetic logic unit and the control unit together form the central processing unit.

Control Unit:

This unit controls all the other units of the computer system and so is known as its central nervous system. It transfers data throughout the computer as required including from storage unit to central processing unit and vice versa. The control unit also dictates how the memory, input output devices, arithmetic logic unit etc. should behave.

Computer system has five basic units that help the computer to perform operations, which are given below:

- Input Unit
- Output Unit
- Storage Unit

- Arithmetic Logic Unit
- Control Unit

Input Unit

Input unit connects the external environment with internal computer system. It provides data and instructions to the computer system. Commonly used input devices are keyboard, mouse, magnetic tape etc.

Video Content / Details of website for further learning (if any):

https://youtu.be/Xc7qfdhASwY

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 458-462)

Course Faculty

Verified by HOD

L31

LECTURE HANDOUTS

ECE

II/IV

L31

Course Name with Code: 19ECC07/ MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : **IV -** ARM-32 BIT MICROCONTROLLER

Date of Lecture:

Topic of Lecture: Debugging support

Introduction :

Before the introduction of on-chip debugging, most software developers were using expensive **In-Circuit Emulators (ICE)** for application testing on microcontrollers. High-end emulators were connected via complex adapters, and offered also extensive instruction and data trace capabilities with complex triggers.

These emulators were based on special bond-out devices that were different from the standard production devices and therefore very expensive compared to the actual MCU device.

Prerequisite knowledge for Complete understanding and learning of Topic:

Debugging with Cortex-M3 Microcontrollers

Detailed content of the Lecture:

Embedded ICE is a low-cost hardware block that provides complete run-control with two hardware break registers that can trigger either on program execution or memory access.

An additional Debug Communication Channel (DCC) allows data exchange with the user application during program execution. The Embedded ICE is the standard debug unit for all ARM7 and ARM9 processor-based microcontrollers that are available today from many silicon vendors. Itis widely supported by the tools industry with standard low-cost JTAGinterfaces that eliminate the need for costly hardware adaptations.

Since the Embedded ICE on-chip debug hardware does not provide any data or instruction trace, some ARM processor-based microcontrollers also integrate the **Embedded TraceMacrocell (ETM).**

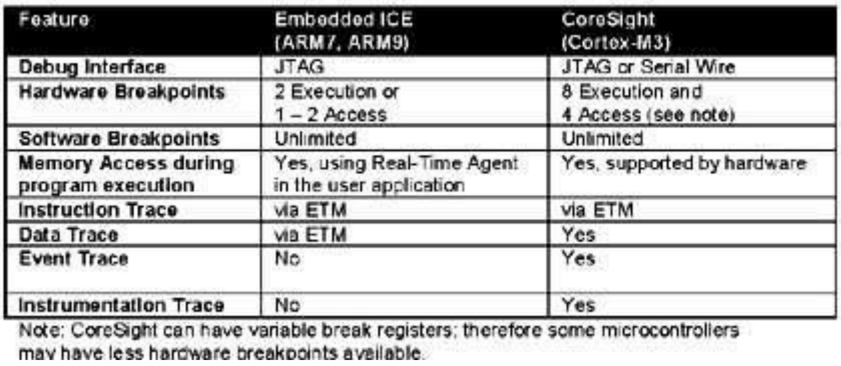
However, the high data bandwidth used by an instruction trace requires data output lines, in addition to those of the standard JTAGpins. A special ETM emulator connects to these ETM data output lines and interprets the trace information.

In microcontrollers, ETM often shares useful I/O lines that are required by the user application and therefore engineers frequently cannot use the ETM unit. To minimize the I/O pins required for debugging, the new Core Sight solution provides additional operating modes via a standard JTAG connector:

1) A standard JTAG mode using five I/O pins for the connection to a JTAG chain or legacy JTAG adapters.

2) A Serial Wire (SW) mode that requires only two I/O pins for run-control debugging. The SW mode is a different mode of the JTAG port that requires only the pins TCLK and TDI for communication.

3) When working with the SW mode, an additional Serial Wire Viewer (SWV) output on the TDO line can provide data trace, event trace, and instrumentation trace information.



Feature	Embedded ICE (ARM7, ARM9)	CoreSight (Cortex-M3)
Debug Interface	JTAG	JTAG or Serial Wire
Hardware Breakpoints	2 Execution or 1 – 2 Access	8 Execution and 4 Access (see note)
Software Breakpoints	Unlimited	Unlimited
Memory Access during program execution	Yes, using Real-Time Agent in the user application	Yes, supported by hardware
Instruction Trace	via ETM	via ETM
Data Trace	via ETM	Yes
Event Trace	No	Yes
Instrumentation Trace	No	Yes

Note: CoreSight can have variable break registers; therefore some microcontrollers may have less hardware breakpoints available.

Table1: Comparison of Debug Features

core Sight is the debug technology used in Cortex-M3 processor-based microcontrollers. A low-cost JTAG adapter (for example the Keil ULINK2)is all that is required to interface to the core Sight on-chip debug unit.

In addition to the trace features, the core Sight unitimplementsadditional break registers and provides on-the-fly memory access during program execution without additional software overhead.

Development Tools

ARM processor-based microcontrollers are widely supported by the development tool industry. For example, the ARM Real View Microcontroller Development Kit (**Figure 1, below**) available from ARM/Keil provides device-specific support for more than 260 standards

Video Content / Details of website for further learning (if any):

https://www.coursera.org/lecture/embedded-software-hardware/7-debugging-a-microcontroller-program-part-1-7rK4H

Important Books/Journals for further learning including the page nos.:

Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (449-462)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

ECE

II/IV

Course Name with Code : 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : IV -ARM-32 BIT MICROCONTROLLER **Date of Lecture:**

Topic of Lecture: general purpose registers

Introduction :

General purpose registers are used to store temporary data within the microprocessor. ... It is of 16 bits and is divided into two 8-bit registers BH and BL to also perform 8-bit instructions. It is used to store the value of the offset.

Prerequisite knowledge for Complete understanding and learning of Topic:

general purpose registers

Detailed content of the Lecture:

In computer architecture, a processor register is a quickly accessible location available to a computer's central processing unit (CPU). Registers usually consist of a small amount of fast storage, although some registers have specific hardware functions, and may be read-only or write-only. Registers are normally measured by the number of bits they can hold, for example, an “8-bit register”, “32-bit register” or a “64-bit register” (or even with more bits).

General Registers or General Purpose Registers are a kind of registers which can store both data and addresses. All general registers of the intel 8086 microprocessor can be used for arithmetic and logic operations.

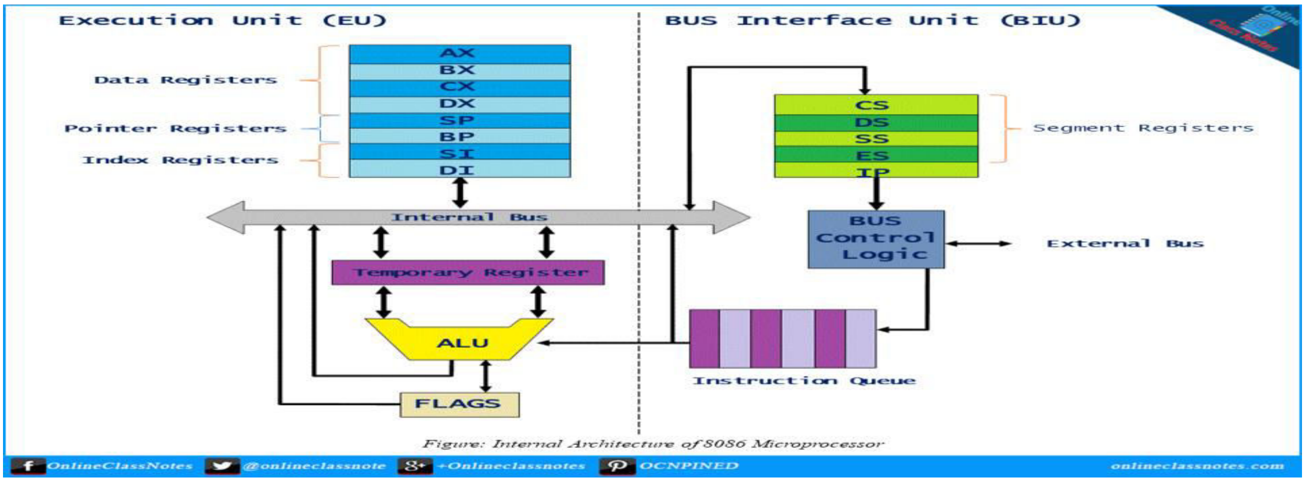


Figure: Internal Architecture of 8086 Microprocessor

AX (Accumulator)

This is accumulator register. It gets used in arithmetic, logic and data transfer instructions. In manipulation and division, one of the numbers involved must be in AX or AL

BX (Base Register)

This is base register. BX register is an address register. It usually contain a data pointer used for based, based indexed or register indirect addressing.

CX (Count register)

This is Count register. This serves as a loop counter. Program loop constructions are facilitated by it. Count register can also be used as a counter in string manipulation and shift/rotate instruction.

DX (Data Register)

This is data register. Data register can be used as a port number in I/O operations. It is also used in multiplication and division.

SP (Stack Pointer)

This is stack pointer register pointing to program stack. It is used in conjunction with SS for accessing the stack segment.

BP (Base Pointer)

This is base pointer register pointing to data in stack segment. Unlike SP, we can use BP to access data in the other segments.

SI (Source Index)

This is source index register which is used to point to memory locations in the data segment addressed by DS. Thus when we increment the contents of SI, we can easily access consecutive memory locations.

DI (Destination Index)

This is destination index register performs the same function as SI. There is a class of instructions called string operations, that use DI to access the memory locations addressed by ES.

ALU (Arithmetic & Logic Unit)

This unit can perform various arithmetic and logical operation, if required, based on the instruction to be executed. It can perform arithmetical operations, such as add, subtract, increment, decrements, convert byte/word and compare etc and logical operations.

Video Content / Details of website for further learning:

https://www.youtube.com/watch?v=8VPYmN5DXo0

Important Books/Journals for further learning including the page nos.:

Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (490-497)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

L33

LECTURE HANDOUTS

ECE

II/IV

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : IV - ARM 32 BIT MICROCONTROLLER

Date of Lecture:

Topic of Lecture: Special registers

Introduction :

- A Special Function Register (or Special Purpose Register, or simply Special Register) is a register within a microprocessor, which controls or monitors various aspects of the microprocessor's function.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Architecture
- Working of DMA Controller

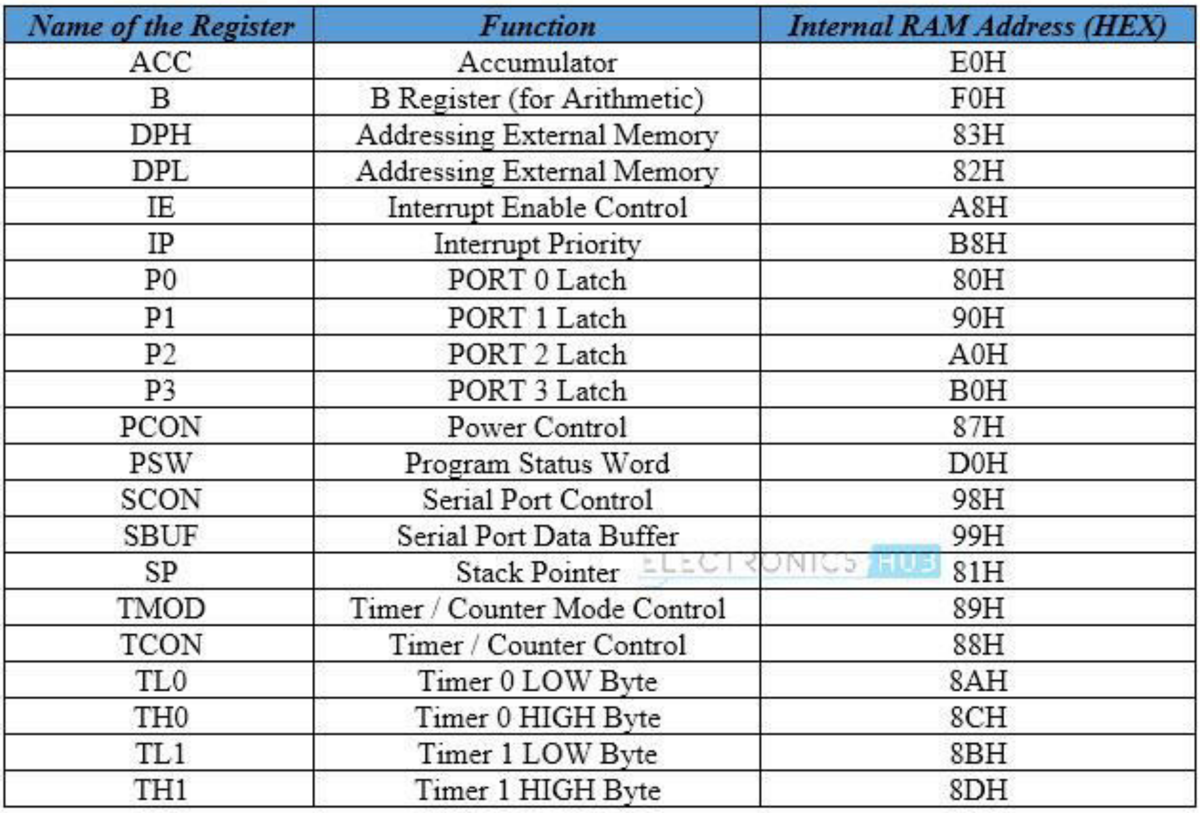
Detailed content of the Lecture:

Special Registers:

- Special registers are reserved words that name storage areas generated by the compiler. Their primary use is to store information produced through specific COBOL features. Each such storage area has a fixed name, and must not be further defined within the program.
- In the general formats of this specification, a special register can be used, unless otherwise restricted, wherever a data-name or identifier is specified provided that the special register is the same category as the data-name or identifier. If qualification is allowed, special registers can be qualified as necessary to provide uniqueness.
- When control of a program is transferred for the first time from one program to another within the run unit by the CALL statement, the compiler initializes the special register fields to their initial values. The RETURN-CODE and SORT-RETURN special

registers are reset to their initial values in the following instances:

- Whenever the CANCEL statement is invoked to initialize a referenced subprogram
- For programs that possess the INITIAL attribute
- For programs that possess the RECURSIVE attribute
- In all other cases, the special registers are not reset to their initial values. Instead, they remain unchanged from the value retained the previous time program control was transferred via the CALL statement.
- You can specify an alphanumeric register in a function wherever an alphanumeric argument is allowed, unless specifically prohibited.
- You can specify a numeric special register in a function wherever a numeric argument is allowed, unless specifically prohibited.
- Each special register is discussed in the section beginning on the indicated page.
- The special registers are:
- ADDRESS OF
- DB-FORMAT-NAME
- DEBUG-ITEM
- FORMAT OF
- LENGTH OF
- LINAGE-COUNTER
- LOCALE OF
- RETURN-CODE
- SORT-RETURN
- WHEN-COMPILED
- XML-CODE
- XML-EVENT
- XML-NTEXT
- XML-TEXT



Name of the Register	*Function*	*Internal RAM Address (HEX)*
ACC	Accumulator	E0H
B	B Register (for Arithmetic)	F0H
DPH	Addressing External Memory	83H
DPL	Addressing External Memory	82H
IE	Interrupt Enable Control	A8H
IP	Interrupt Priority	B8H
P0	PORT 0 Latch	80H
P1	PORT 1 Latch	90H
P2	PORT 2 Latch	A0H
P3	PORT 3 Latch	B0H
PCON	Power Control	87H
PSW	Program Status Word	D0H
SCON	Serial Port Control	98H
SBUF	Serial Port Data Buffer	99H
SP	Stack Pointer	81H
TMOD	Timer / Counter Mode Control	89H
TCON	Timer / Counter Control	88H
TL0	Timer 0 LOW Byte	8AH
TH0	Timer 0 HIGH Byte	8CH
TL1	Timer 1 LOW Byte	8BH
TH1	Timer 1 HIGH Byte	8DH

Video Content / Details of website for further learning:

https://youtu.be/zzr0zLLeCQw

Important Books/Journals for further learning including the page nos.:

Gaonkar.R.S, Microprocessor architecture programming and applications with 8085,wiley eastern ltd, New Delhi 2013-Page no (490-497)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

LECTURE HANDOUTS

ECE

L34

II/IV

Course Name with Code : 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : IV - ARM 32 BIT MICROCONTROLLER

Date of Lecture:

Topic of Lecture: Exceptions

Introduction :

Exceptions, are situations where the processor needs to stop executing the current code because of an error. ... For instance, if the ALU attempts to divide by zero, or if an addition cause's overflow, an exception might be triggered. The processor needs to stop operation and fix the error before the program can be resumed.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Communication Modes

Detailed content of the Lecture:

Exceptions:

- Exceptions are events that disrupt the normal execution flow of the program. When an exception occurs the processor handles it by usually executing dedicated piece of code called exception handler. Each type of exception can have its own exception handler.
- In some literature sources exceptions and interrupts are analyzed as two different things. Expeptions being generated internally by the microprocessors itself and interrupts being generated by external source (e.g hardware peripherals SPI, ADC, GPIO etc). For the sake of simplifying the categorization we are looking at the interrupts as a specific type of exception.

Types of exceptions:

There are many types of exceptions and they are usually dependent on the architecture of the processor. However there are exceptions that are found in almost every processor type. Below are listed some of the most popular exception types.

What happens when an exception occurs:

- The program counter (PC) value is put into the stack
- The reason for the exception can be stored in dedicated status registers
- Exceptions can be disabled while the current one is being handled
- Execute an exception handler routine for the current exception

Exception Conditions:

The following sections describe each of the possible exception conditions in detail. Each description classifies the exception as a fault, trap, or abort. This classification provides information needed by systems programmers for restarting the procedure in which the exception occurred:

Faults :

The CS and EIP values saved when a fault is reported point to the instruction causing the fault.

Traps:

The CS and EIP values stored when the trap is reported point to the instruction dynamically after the instruction causing the trap. If a trap is detected during an instruction that alters program flow, the reported values of CS and EIP reflect the alteration of program flow. For example, if a trap is detected in a JMP instruction, the CS and EIP values pushed onto the stack point to the target of the JMP, not to the instruction after the JMP.

Aborts:

An abort is an exception that permits neither precise location of the instruction causing the exception nor restart of the program that caused the exception. Aborts are used to report severe errors, such as hardware errors and inconsistent or illegal values in system tables.

The bit configuration of mode instruction is shown in Figure. In the case of synchronous mode, it is necessary to write one-or two byte sync characters. If sync characters were written, a function will be set because the writing of sync characters constitutes part of mode instruction.

Video Content / Details of website for further learning (if any):

https://youtu.be/FrFRUDeK6Ws

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 202-208)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

ECE

II/IV

L 35

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : IV- ARM-32 BIT MICROCONTROLLER **Date of Lecture:**

Topic of Lecture: Interrupts, stack operation,

Introduction :

The ARM processor has two levels of external interrupt, **FIQ and IRQ**, both of which are level-sensitive active LOW signals into the processor. For an interrupt to be taken, the appropriate disable bit in the CPSR must be clear. ... FIQs are handled first when multiple interrupts occur.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basic operation

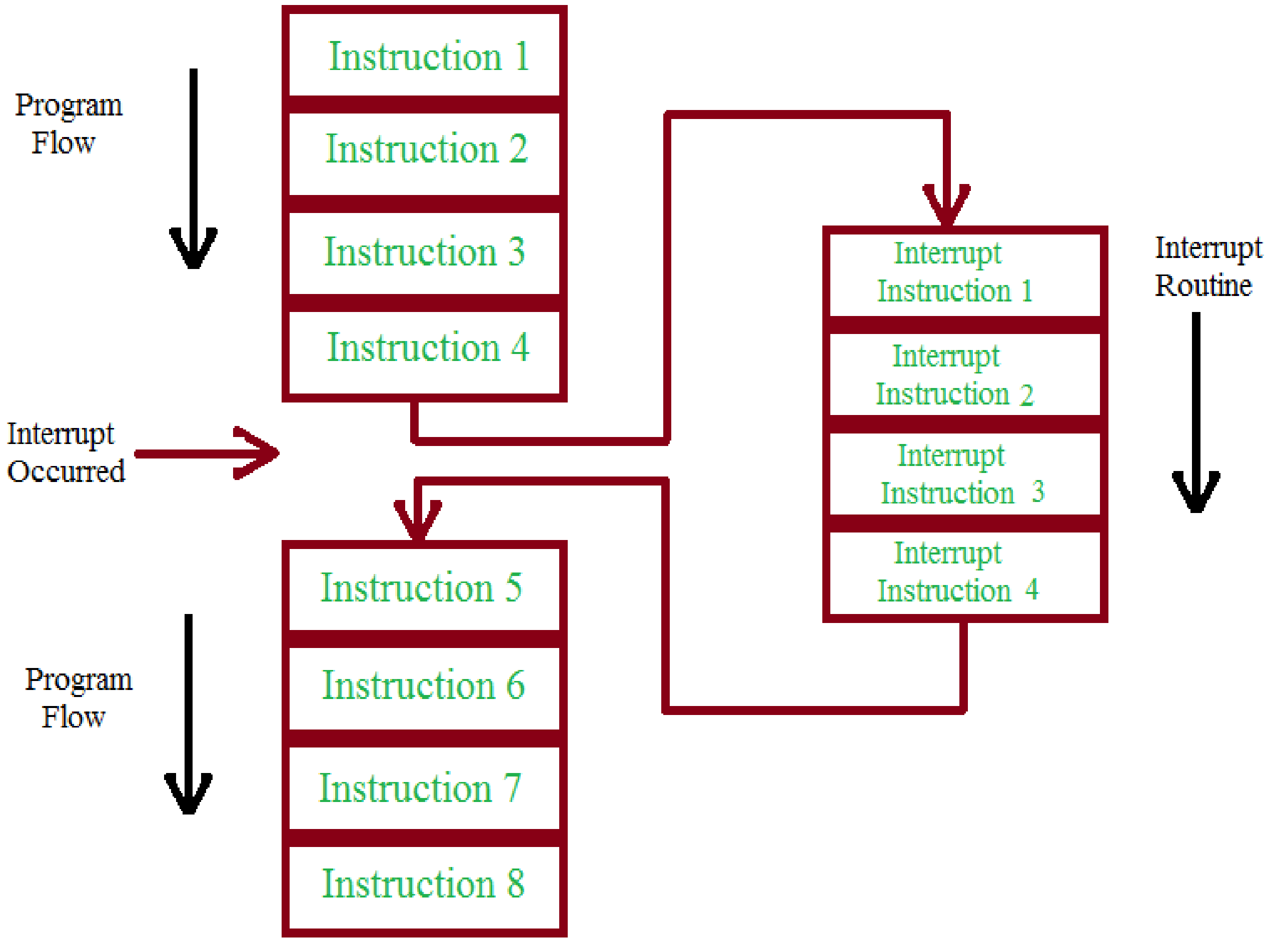
Detailed content of the Lecture:

The main purpose of any microcontroller is to accept input from input devices and accordingly drive the output. Hence, there will be several devices connected to a microcontroller at a time. Also, there are many internal components in a microcontroller like timers, counters etc. that require attention of the processor.

Since all the devices can't obtain the attention of the processor at all times, the concept of "Interrupts" comes in to picture. An Interrupt, as the name suggests, interrupts the microcontroller from whatever it is doing and draws its attention to perform a special task. The following image depicts the procedure involved in Interrupts.

In the event of an interrupt, the source of the interrupt (like a Timer, Counter etc.) sends a special request to the processor called Interrupt Request (IRQ) in order to run a special piece of code. The special code or function is called as Interrupt Service Routine (ISR).

From the above figure, the CPU executes its normal set of codes until an IRQ occurs. When the IRQ signal is received, the CPU stops executing the regular code and starts executing the ISR. Once the execution of the ISR is completed by the CPU, it returns back to execution of the normal code.



Vectored Interrupt Controller (VIC) handles the interrupts in LPC214x series of MCUs. It can take up to 32 Interrupt Requests. The interrupts in LPC2148 microcontroller are categorized as Fast Interrupt Request (FIQ), Vectored Interrupt Request (IRQ) and Non – Vectored Interrupt Request. All the interrupts in LPC214x have a programmable settings i.e. the priorities of the Interrupts can be dynamically set.

Of the three categories, the FIQ requests have the highest priority, Vectored IRQ requests have the medium priority and non – vectored IRQ requests have the least priority.

When we are talking about "Vectored" and "Non – Vectored" IRQ requests, we are actually talking about the address of the ISR. In case of Vectored IRQ requests, the CPU has a knowledge of the ISR. A special table called Interrupt Vector Table (IVT) contains all the information about the Vectored IRQ. This information can be about the source of the interrupts, ISR address of the IRQ requests etc.

So, each Vectored IRQ has its own unique ISR address. Out of the possible 32 interrupt requests, 16 interrupt requests can be defined as Vectored IRQ. In this 16 slots, any of the 22 interrupts that are available in LPC2148 can be assigned. In the 16 Vectored IRQ slots, slot 0 has the highest priority while slot 16 has the least priority.

In case of Non – Vectored IRQ, as the name itself indicates, the CPU isn't aware of either the source of the Interrupt or the ISR address of the Interrupts. In this case, the CPU must be provided with a default ISR address. For handling Non – Vectored IRQ requests, a special register called "VICDefVectAddr" is available in LPC2148. The address of the default ISR must be given in this register by the user in order to handle the Non – Vectored IRQ requests.

What is Stack Pointer?

Definition: The stack is a storage device, used for storing information or data in a manner of LIFO (Last In First Out). Whenever we enter the data in the form of LIFO manner, the element that has to be deleted first is the last inserter element, so the last inserted element is taken out first. It is the memory unit within an address register called stack pointer (SP). The stack pointer always indicates the top element in the stack that means which location the data has to be inserted.

Types of Stack

There are two types of stacks they are register stack and the memory stack.

Register Stack

The register stack is also a memory device present in the memory unit, but it handles only a small amount of data. The stack depth is always limited in the register stack because the size of the register stack s very small compared to the memory.

Push Operation in Register Stack

Step1: The stack pointer increments by 1.
SP←SP+1

Step2: Enter the data into the stack.

M[SP]←DR

Where DR is the Data Register

Step3: Check whether the stack is full or not
if (sp=0) then (full←1)

Step4: Mark not empty

empty←0

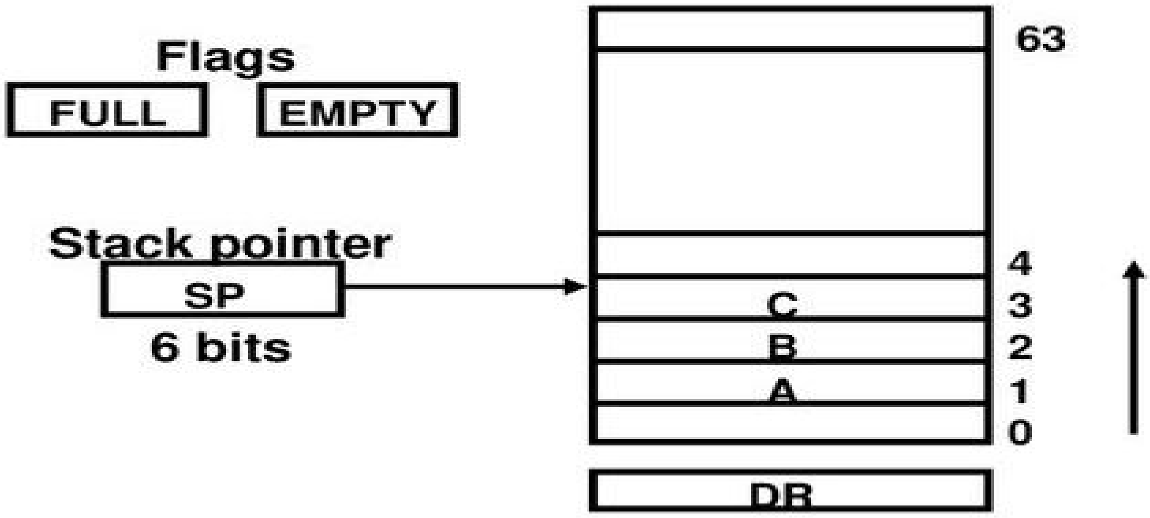
Pop Operation in Register Stack

Step1: Read data from the stack.
DR←M[SP]

Step2: Decrement stack point.
SP←SP-1

Step3: Check whether the stack is empty or not
if sp=0 then empty←1

The stack organization of the 64-bit register stack is shown in the below fi



Video Content / Details of website for further learning:

https://www.youtube.com/watch?v=8Umh-E7bi_
I**https://www.youtube.com/watch?v=CV7lI5a3Zdc**

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 431-438)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 36

ECE

II/IV

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : IV- ARM-32 BIT MICROCONTROLLER

Date of Lecture:

Topic of Lecture: Reset sequence.

Introduction :

When you press the reset button, it will copy the Stack pointer to MSP (Main Stack Pointer) from the location of 0x00000000. Then it moves to Reset-Handler. In the Reset-handler, it will initialize the hardware (system), then copy the initialized data (Initialized global variable and static variable) to SRAM..

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basic operation

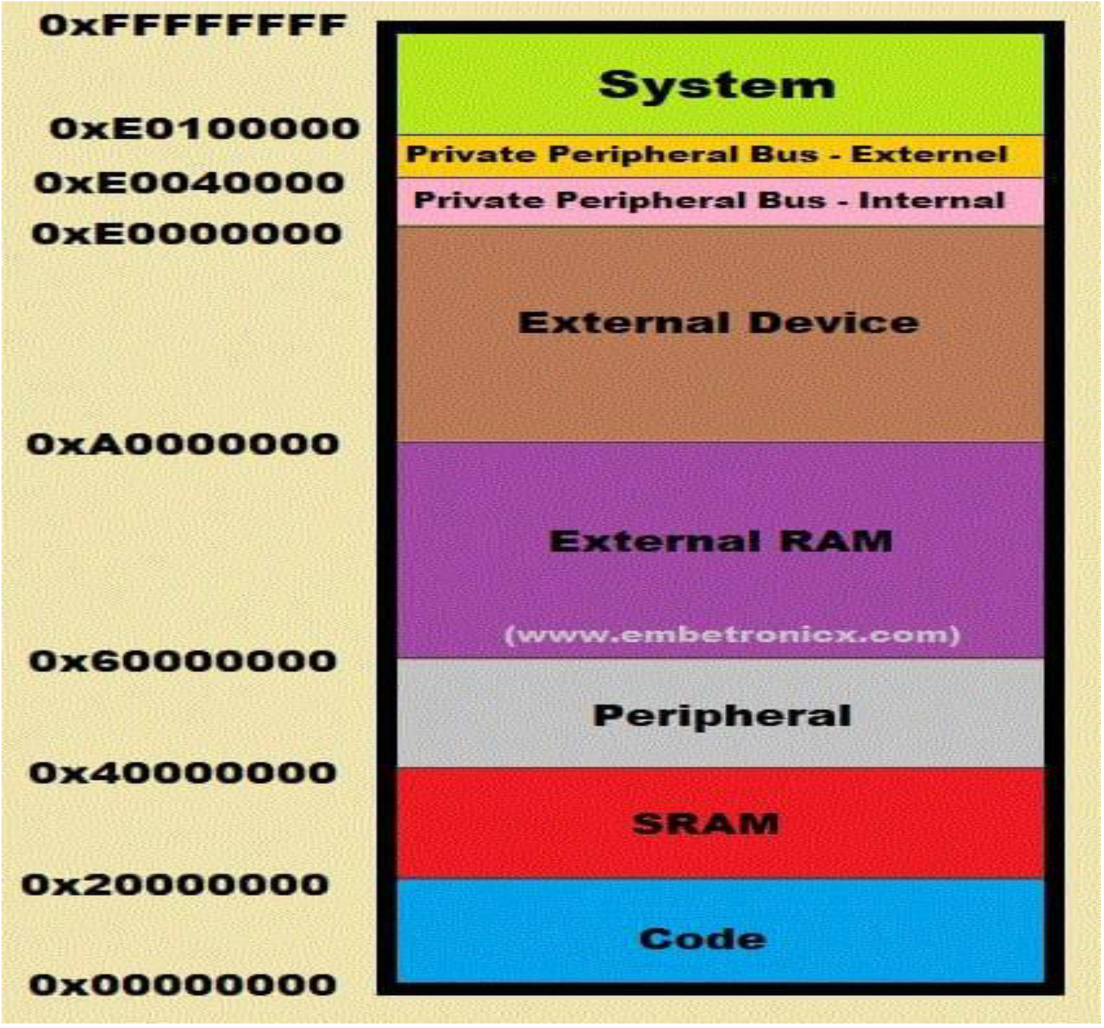
Detailed content of the Lecture:

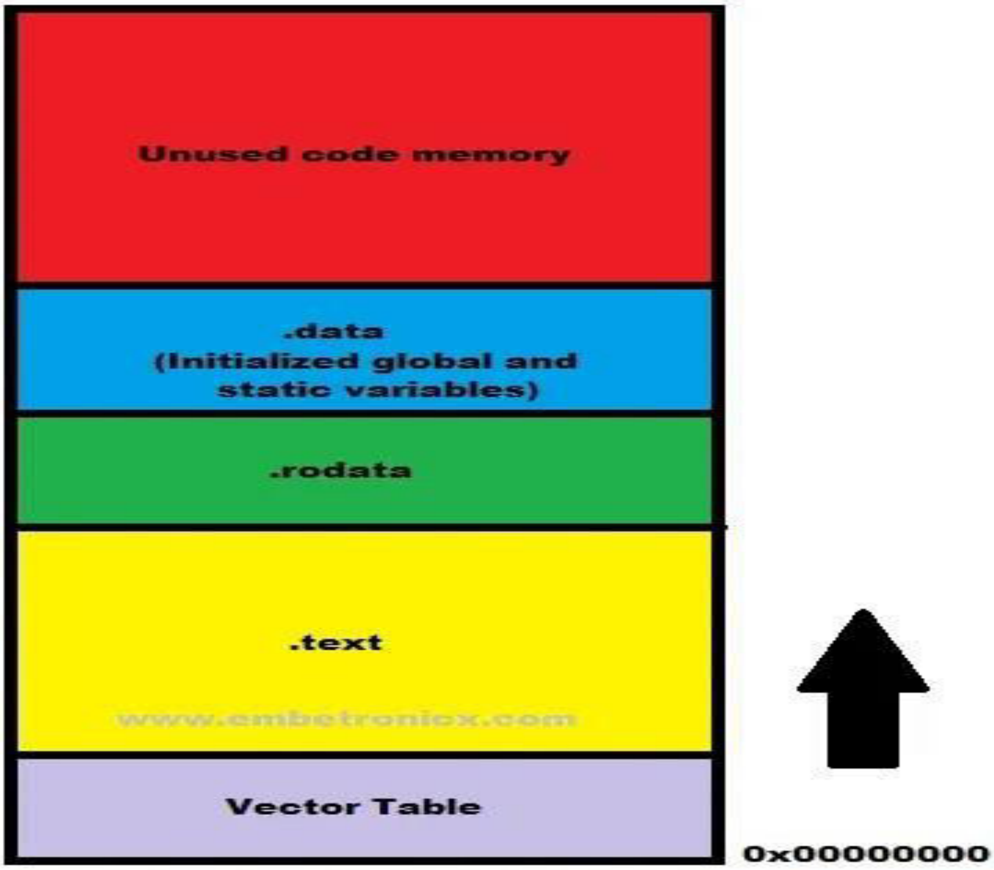
Reset Sequence in ARM Cortex-M4 Reset Sequence in ARM Cortex-M4 by SLRAs you are an embedded engineer, have you think anytime, what happens when you press the reset button (Reset Sequence in ARM Cortex-M4)? How it is calling the main() function in your application automatically? And also, If you are attending any interviews, the interviewer might have asked you this question. Okay, Let's discuss the reset sequence in the microcontroller. Before starting this we must know about the microcontroller memory architecture. Please find the below image to know about the Memory map of ARM Cortex-M4.

Here, we have a code region, where we are going to write our final binary. That memory is starting from 0x00000000 to 0x1FFFFFFF. Now we will forget about other regions. We will take only the SRAM and Code region. We know that the code region has the final output of our program (.hex or .bin or etc). SRAM will be having stack, heap, global RW variables, and Static variables, etc.

Memory layout of the program

Now we have to know how the final output of our program is stored in the code region. The below image will explain how our program is stored in the code region.


0xFFFFFFFF
System
0xE0100000
Private Peripheral Bus - Externel
0xE0040000
Private Peripheral Bus - Internal
0xE0000000
External Device
0xA0000000
External RAM
(www.embetronicx.com)
0x60000000
Peripheral
0x40000000
SRAM
0x20000000
Code
0x00000000


Unused code memory
.data
(Initialized global and
static variables)
.rodata
.text
www.embetronicx.com
Vector Table
0x00000000

What happens When you press the reset button in ARM Cortex-M4?
PC (Program Counter) will be loaded with 0x00000000. So will start from the address 0x00000000.
Since the address has an Initial Stack Pointer value, It will be fetched to the MSP (Main Stack Pointer). So, that value will be starting of the stack.
Then PC will be loaded with the Reset handler's address.
Note: These above 3 steps are done by the hardware (This is architecture-specific).

After that, the reset handler will perform the below operations.

Initialize the system.
Copy the Initialized global variable, static variable (.data) to SRAM.
Copy the Un-initialized data (.bss) to SRAM and initialize it to 0.
It Calls main().
This is how your main() is getting called while power-up or press the reset button. You may understand clearly if you see the execution below.

Video Content / Details of website for further learning:

http://fastbitlab.com/arm-cortex-m-processor/

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 431-438)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

ECE **II/IV**

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : V - ARM CORTEX M3 PROGRAMMING

Date of Lecture:

Topic of Lecture: Assembly basics

Introduction :

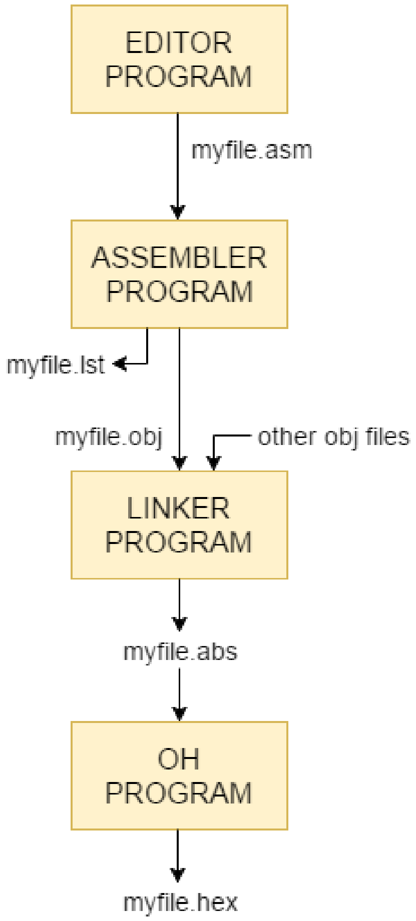
Assembly language is a low-level programming language for a computer or other programmable device specific to a particular computer architecture in contrast to most high-level programming languages, which are generally portable across multiple systems. Assembly language is converted into executable machine code by a utility program referred to as an assembler like NASM, MASM, etc

Prerequisite knowledge for Complete understanding and learning of Topic:

- Interfacing functions

Detailed content of the Lecture: Assembly Basics

- Assembly language is also called as low-level language because it directly works with the internal structure of CPU. For programming in assembly language, a programmer must have the knowledge of all the registers in a CPU.
- Different programming languages like C, C++, Java and various other languages are called as high-level languages because they are not dealing with the internal details of CPU.



- **Editor Program** : At first, we use an editor for type in a program. Editors like MS-DOS program that comes with all Microsoft operating systems can be used for creating or edit a program. The editor produces an ASCII file. The ?asm? extension for a source file is used by an assembler during next step.
- **Assembler Program**: The "asm" source file contain the code created in Step 1. It is transferred to an 8051 assembler. The assembler is used for converting the assembly language instructions into machine code instructions and it produced the , It is also called as source file because some assembler requires that this file must have "src" extension.
- **Linker Program**: The linker program is used for generating one or more object files and produces an absolute object file with an extension "abs".
- **OH Program**: The OH program fetches the "abs" file and fed it to a program called "OH". OH is called as object to hex converter it creates a file with an extension "hex" that is ready for burn in to the ROM.

Labels in assembly Language

All labels used in assembly language follow the certain rules as given below:

- Each label name should be unique. The name used as label in assembly language programming consist of alphabetic letters in both lowercase and uppercase, numbers from 0 to 9, and special characters such as at the rate (@), question mark (?), underscore(_), and dollar ($) etc.

- Reserved words are not allowed to be used as a label in the program. For example, MOV and ADD words are reserved words.
- The first character must be an alphabetical character, it cannot be a number.

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=0nZauIwn-xo

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (R1-363-370)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

L38

LECTURE HANDOUTS

ECE

II/IV

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : **V –** ARM CORTEX M3 PROGRAMMING **Date of Lecture:**

Topic of Lecture: INSTRUCTION LISTS AND DESCRIPTION

Introduction :

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

Prerequisite knowledge for Complete understanding and learning of Topic:

- 8085 & 8051 Features

Detailed content of the Lecture:

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.

- **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.

78: The instruction in the code form.

In the above example, MOV A, B is 78H. Besides the operation to be performed the opcode also specifies source and destination registers.

The opcode 78H can be written in binary form as 01111000. The first two bits, i.e. 0 1 are for MOV operation, the next three bits 1 1 1 are the binary code for register A, and the last three bits 000 are the binary code for register B.

- **Register Indirect Addressing**

In Register Indirect mode of addressing, the address of the operand is specified by a register pair.

Example

- **LXI H, 2500 H** - Load H-L pair with 2500H.
- **MOV A, M** - Move the content of the memory location, whose address is in H-L pair (i.e. 2500 H) to the accumulator.
- **HLT** - Halt.

In the above program the instruction MOV A, M is an example of register indirect addressing. For this instruction, the operand is in the memory. The address of the memory is not directly given in the instruction. The address of the memory resides in H-L pair and this has already been specified by an earlier instruction in the program, i.e. LXI H, 2500 H.

- **Immediate Addressing**

In this addressing mode, the operand is specified within the instruction itself.

Example

LXI H, 2500 is an example of immediate addressing. 2500 is 16-bit data which is given in the instruction itself. It is to be loaded into H-L pair.

- **Implicit Addressing**

There are certain instructions which operate on the content of the accumulator. Such instructions do not require the address of the operand.

Example

CMA, RAL, RAR, etc.

Status Flags

There is a set of five flip-flops which indicate status (condition) arising after the execution of arithmetic and logic instructions. These are:

- Carry Flag (CS)
- Parity Flag (P)
- Auxiliary Carry Flags (AC)
- Zero Flags (Z)
- Sign Flags (S)

Video Content / Details of website for further learning:

https://www.youtube.com/watch?v=G3iUO96XhC4

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (R1-352-362)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

ECE

II/IV

L39

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIG

Course Faculty : Mrs.S.Punitha

Unit : **V -** ARM CORTEX M3 PROGRAMMING **Date of Lecture:**

Topic of Lecture: DESCRIPTION OF INSTRUCTON LISTS

Introduction :

An instruction is a binary command used to execute an operation inside the microprocessor over a given data. A group of instructions that are supported by the 8085 microprocessor is known to be the instruction set of 8085 microprocessors.

Prerequisite knowledge for Complete understanding and learning of Topic:

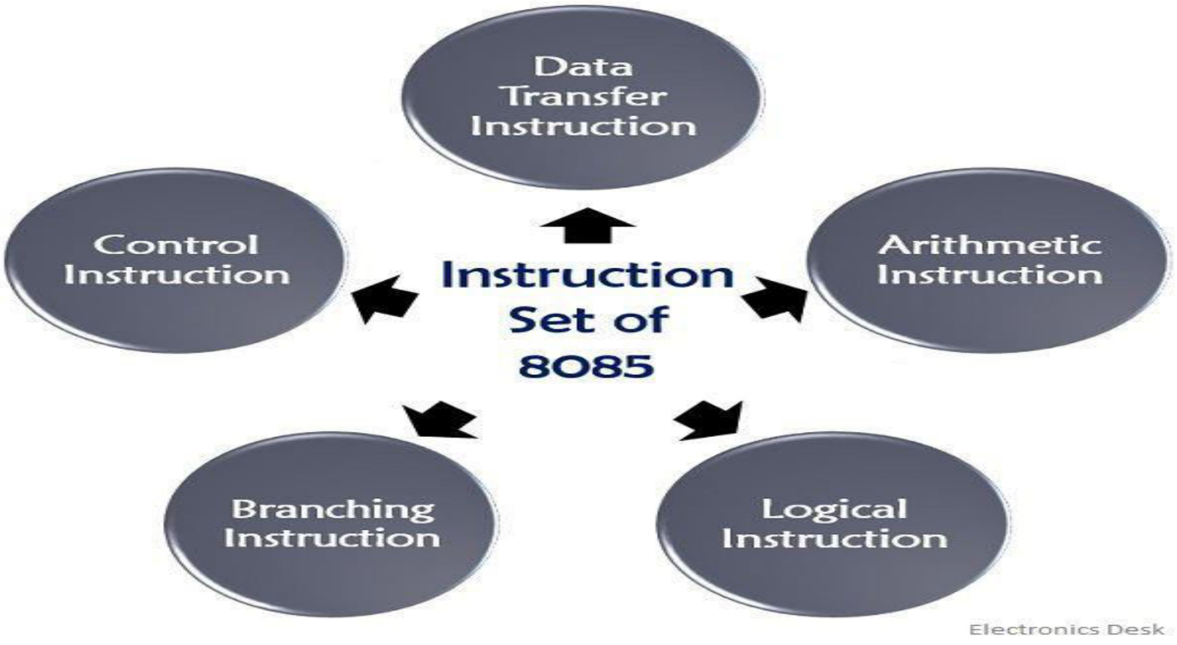
- 8085 & 8084

Detailed content of the Lecture: instructions list using 8085 and 8051

Instruction set of 8085 microprocessors

We know that an instruction set is a group of instruction that executes the desired operation on the data specifically described by the microprocessor 8085.

The instruction set of 8085 microprocessor is classified as follows:



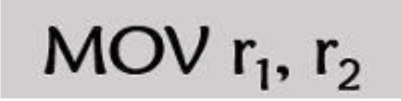
Let us now move further to understand each instruction set separately.

Data Transfer Instruction

These instructions are used by the microprocessor in order to transfer the data from one location to another. More specifically, it helps in transferring the data from source operand to destination operand, without making any changes in the source data.

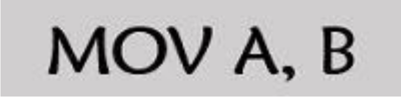
As the data present in the source is not modified. Thus it is more convenient to call it a data copy instruction. As while transferring the data from source to destination, the data present in the source is just copied to the destination also. Due to this no change in source data is noticed.

Basically, the instructions that come under this instruction set, are as follows:



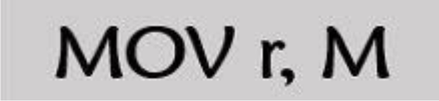
MOV r_1, r_2

This instruction helps to transfer the contents of register r_2 into register r_1. While the data in register r_2 will remain the same.



MOV A, B

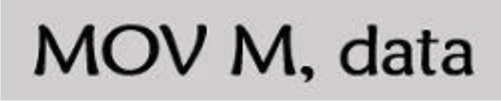
This instruction transfers the data in register B to an accumulator.



MOV r, M

This instruction specifies the transfer of data present in the memory to the register. However, the address of the memory location must be present in the HL register.

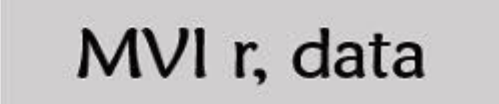
Example: MOV r, 1020H



MOV M, data

This instruction specifies the immediate transfer of data to a memory location. And the address of this memory location is specified at the H-L registers.

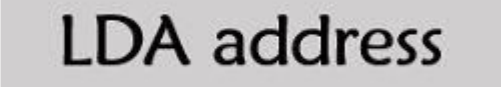
Example: MOV M, 32H



MVI r, data

In this instruction, the data is immediately transferred to the specified register.

Example: MVI r, 16H



LDA address

It is load accumulator instruction. This instruction is used to copy the data present in the memory address specified as the operand of the instruction to the accumulator. More specifically, in this case, the data present in the 16-bit memory address is transferred to the accumulator.

Example: LDA 1000H

LDAX register.pair

It is load accumulator indirect instruction. In this instruction, the register that acts as operand holds a memory location, where the actual data is present which is to be loaded at the accumulator.

Example: LDAX C/D

LHLD address

It stands for load H-L registers direct instruction. For this particular instruction, the data present in the address specified in the operand is copied to the L register. Also, the data present in the next memory location is loaded to the H register.

Example: LHLD 4500H

STA address

STA is stored accumulator direct instruction. Whenever this instruction is passed, then the data present in the accumulator is transferred to the memory address specified in the operand.

Example: STA 2032H

STAX register

It is stored accumulator indirect instruction. The register present as the operand holds a memory address. So, the data of the accumulator is copied to that particular memory location.

Example: STAX D

XCHG

This instruction is used to exchange the data present in two registers.

Example: We have XCHG H-L and D-E

So the contents of H and D while L and E are exchanged.

SPHL

This instruction transfers the data of HL pair into the stack pointer.

PCHL

Like the SPHL, this instruction copies the data of H-L register into the stack pointer by placing the higher order bytes at H and lower order bytes at L.

PUSH reg.pair

In this instruction, the stack is loaded with the data present in the register given in the operand.

Firstly, the SP gets decremented and higher order bytes are copied to the stack. Further SP gets decremented in order to load the lower order register bytes.

Example: PUSH C

POP reg.pair

This instruction specifies the transfer of the data present at the top of the stack to the register given as the operand.

Example: POP D

OUT address

In case of OUT instruction, the data at the accumulator is copied to the I/O port. At the operand an 8-bit port address is present.

Example: OUT 56 H

IN address

This instruction specifies the loading of data present at the I/O port to the accumulator. The operand holds the address of the port from where the data is to be copied.

Example: IN, 5B H

Arithmetic Instruction

This instruction set allows arithmetic operations to be performed over the data in memory and register inside the 8085 microprocessors.

Video Content / Details of website for further learning (if any):
https://www.youtube.com/watch?v=G3iUO96XhC4

Important Books/Journals for further learning including the page nos.:

R.S. Gaonkar, ‘Microprocessor Architecture Programming and Application’, with 8085, Wiley Eastern Ltd., New Delhi, 2013 (R3 498-507)

Course Faculty

Verified by HOD

EEE	LECTURE HANDOUTS	III / V
		L40

Course Name with Code: 16EEC09/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : **Mrs.S.Punitha**

Unit : **V -** ARM CORTEX M3 PROGRAMMING **Date of Lecture:**

Topic of Lecture: Useful instructions in mpmc

Introduction :

The 8086 microprocessor supports 8 types of instructions −

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- String Instructions
- Program Execution Transfer Instructions (Branch & Loop Instructions)
- Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions
-

Prerequisite knowledge for Complete understanding and learning of Topic:

- 8086 microprocessors

Detailed content of the Lecture: Useful instructions in mpmc

Data Transfer Instructions

These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group −

Instruction to transfer a word

- **MOV** − Used to copy the byte or word from the provided source to the provided destination.
- **PPUSH** − Used to put a word at the top of the stack.
- **POP** − Used to get a word from the top of the stack to the provided location.
- **PUSHA** − Used to put all the registers into the stack.
- **POPA** − Used to get words from the stack to all registers.
- **XCHG** − Used to exchange the data from two locations.
- **XLAT** − Used to translate a byte in AL using a table in the memory.

Instructions for input and output port transfer

- **IN** − Used to read a byte or word from the provided port to the accumulator.

- **OUT** − Used to send out a byte or word from the accumulator to the provided port.

Instructions to transfer the address

- **LEA** − Used to load the address of operand into the provided register.
- **LDS** − Used to load DS register and other provided register from the memory
- **LES** − Used to load ES register and other provided register from the memory.

Instructions to transfer flag registers

- **LAHF** − Used to load AH with the low byte of the flag register.
- **SAHF** − Used to store AH register to low byte of the flag register.
- **PUSHF** − Used to copy the flag register at the top of the stack.
- **POPF** − Used to copy a word at the top of the stack to the flag register.

Arithmetic Instructions

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.

Following is the list of instructions under this group −

Instructions to perform addition

- **ADD** − Used to add the provided byte to byte/word to word.
- **ADC** − Used to add with carry.
- **INC** − Used to increment the provided byte/word by 1.
- **AAA** − Used to adjust ASCII after addition.
- **DAA** − Used to adjust the decimal after the addition/subtraction operation.

Instructions to perform subtraction

- **SUB** − Used to subtract the byte from byte/word from word.
- **SBB** − Used to perform subtraction with borrow.
- **DEC** − Used to decrement the provided byte/word by 1.
- **NPG** − Used to negate each bit of the provided byte/word and add 1/2's complement.
- **CMP** − Used to compare 2 provided byte/word.
- **AAS** − Used to adjust ASCII codes after subtraction.
- **DAS** − Used to adjust decimal after subtraction.

Instruction to perform multiplication

- **MUL** − Used to multiply unsigned byte by byte/word by word.
- **IMUL** − Used to multiply signed byte by byte/word by word.
- **AAM** − Used to adjust ASCII codes after multiplication.

Instructions to perform division

- **DIV** − Used to divide the unsigned word by byte or unsigned double word by word.
- **IDIV** − Used to divide the signed word by byte or signed double word by word.
- **AAD** − Used to adjust ASCII codes after division.
- **CBW** − Used to fill the upper byte of the word with the copies of sign bit of the lower byte.
- **CWD** − Used to fill the upper word of the double word with the sign bit of the lower word.

Bit Manipulation Instructions

These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc.

Following is the list of instructions under this group −

Instructions to perform logical operation

- **NOT** − Used to invert each bit of a byte or word.
- **AND** − Used for adding each bit in a byte/word with the corresponding bit in another byte/word.
- **OR** − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.
- **XOR** − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.
- **TEST** − Used to add operands to update flags, without affecting operands.

Instructions to perform shift operations

- **SHL/SAL** − Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
- **SHR** − Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
- **SAR** − Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

Instructions to perform rotate operations

- **ROL** − Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
- **ROR** − Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
- **RCR** − Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
- **RCL** − Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

String Instructions

String is a group of bytes/words and their memory is always allocated in a sequential order.

Following is the list of instructions under this group −

- **REP** − Used to repeat the given instruction till CX ≠ 0.
- **REPE/REPZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
- **REPNE/REPNZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
- **MOVS/MOVSB/MOVSW** − Used to move the byte/word from one string to another.
- **COMS/COMPSB/COMPSW** − Used to compare two string bytes/words.
- **INS/INSB/INSW** − Used as an input string/byte/word from the I/O port to the provided memory location.
- **OUTS/OUTSB/OUTSW** − Used as an output string/byte/word from the provided memory location to the I/O port.
- **SCAS/SCASB/SCASW** − Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
- **LODS/LODSB/LODSW** − Used to store the string byte into AL or string word into AX.

Program Execution Transfer Instructions (Branch and Loop Instructions)

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

Instructions to transfer the instruction during an execution without any condition −

- **CALL** − Used to call a procedure and save their return address to the stack.

- **RET** − Used to return from the procedure to the main program.
- **JMP** − Used to jump to the provided address to proceed to the next instruction.

Instructions to transfer the instruction during an execution with some conditions −

- **JA/JNBE** − Used to jump if above/not below/equal instruction satisfies.
- **JAE/JNB** − Used to jump if above/not below instruction satisfies.
- **JBE/JNA** − Used to jump if below/equal/ not above instruction satisfies.
- **JC** − Used to jump if carry flag CF = 1
- **JE/JZ** − Used to jump if equal/zero flag ZF = 1
- **JG/JNLE** − Used to jump if greater/not less than/equal instruction satisfies.
- **JGE/JNL** − Used to jump if greater than/equal/not less than instruction satisfies.
- **JL/JNGE** − Used to jump if less than/not greater than/equal instruction satisfies.
- **JLE/JNG** − Used to jump if less than/equal/if not greater than instruction satisfies.
- **JNC** − Used to jump if no carry flag (CF = 0)
- **JNE/JNZ** − Used to jump if not equal/zero flag ZF = 0
- **JNO** − Used to jump if no overflow flag OF = 0
- **JNP/JPO** − Used to jump if not parity/parity odd PF = 0
- **JNS** − Used to jump if not sign SF = 0
- **JO** − Used to jump if overflow flag OF = 1
- **JP/JPE** − Used to jump if parity/parity even PF = 1
- **JS** − Used to jump if sign flag SF = 1

Processor Control Instructions

These instructions are used to control the processor action by setting/resetting the flag values.

Following are the instructions under this group −

- **STC** − Used to set carry flag CF to 1
- **CLC** − Used to clear/reset carry flag CF to 0
- **CMC** − Used to put complement at the state of carry flag CF.
- **STD** − Used to set the direction flag DF to 1
- **CLD** − Used to clear/reset the direction flag DF to 0
- **STI** − Used to set the interrupt enable flag to 1, i.e., enable INTR input.
- **CLI** − Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

Iteration Control Instructions

These instructions are used to execute the given instructions for number of times. Following is the list of instructions under this group −

- **LOOP** − Used to loop a group of instructions until the condition satisfies, i.e., CX = 0
- **LOOPE/LOOPZ** − Used to loop a group of instructions till it satisfies ZF = 1 & CX = 0
- **LOOPNE/LOOPNZ** − Used to loop a group of instructions till it satisfies ZF = 0 & CX = 0
- **JCXZ** − Used to jump to the provided address if CX = 0

Interrupt Instructions

These instructions are used to call the interrupt during program execution.

- **INT** − Used to interrupt the program during execution and calling service specified.

- **INTO** − Used to interrupt the program during execution if OF = 1
- **IRET** − Used to return from interrupt service to the main program

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=Ef3gbGIfG-c

Important Books/Journals for further learning including the page nos.:

R.S. Gaonkar, ‘Microprocessor Architecture Programming and Application’, with 8085, Wiley Eastern Ltd., New Delhi, 2013 (R3 498-507)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to
Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

ECE **LECTURE HANDOUTS** **II/IV**

L41

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : **Mrs.S.Punitha**

Unit : **V -** ARM CORTEX M3 **Date of Lecture:**

Topic of Lecture: MEMORY MAPPING

Introduction:

Memory-mapping is a mechanism that maps a portion of a file, or an entire file, on disk to a range of addresses within an application's address space. The application can then access files on disk in the same way it accesses dynamic memory.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Basic operations of 8085 & 8051

Detailed content of the Lecture: Memory Mapping

Benefits of Memory-Mapping

The principal benefits of memory-mapping are efficiency, faster file access, the ability to share memory between applications, and more efficient coding.

Faster File Access

Accessing files via memory map is faster than using I/O functions such as fread and fwrite. Data are read and written using the virtual memory capabilities that are built in to the operating system rather than having to allocate, copy into, and then deallocate data buffers owned by the process.

MATLAB® does not access data from the disk when the map is first constructed. It only reads or writes the file on disk when a specified part of the memory map is accessed, and then it only reads that specific part. This provides faster random access to the mapped data.

Efficiency

Mapping a file into memory allows access to data in the file as if that data had been read into an array in the application's address space. Initially, MATLAB only allocates address space for the array; it does not actually read data from the file until you access the mapped region. As a result, memory-mapped files provide a mechanism by which applications can access data segments in an extremely large file without having to read the entire file into memory first.

Efficient Coding Style

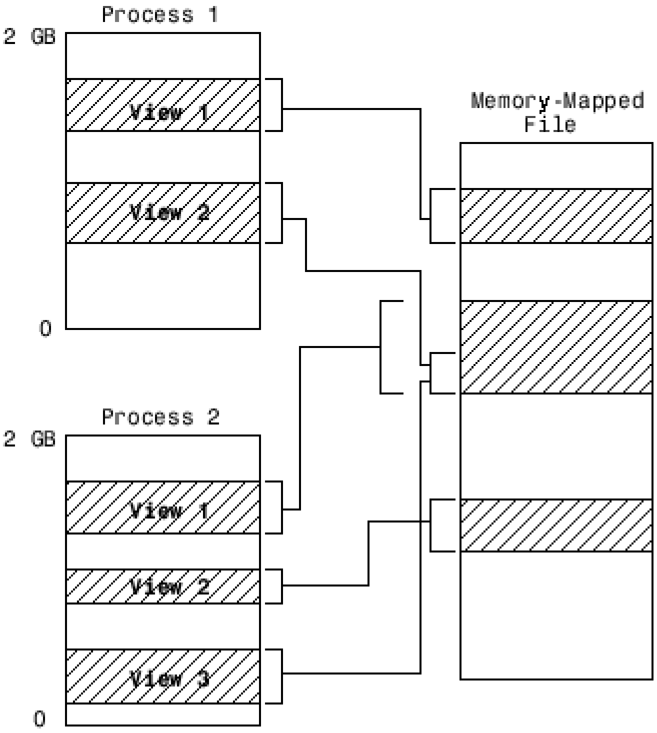
Memory-mapping in your MATLAB application enables you to access file data using standard MATLAB indexing operations. Once you have mapped a file to memory, you can read the contents of that file using the same type of MATLAB statements used to read variables from the MATLAB workspace. The contents of the mapped file appear as if they were an array in the currently active workspace. You simply index into this array to read or write the desired data from the file. Therefore, you do not need explicit calls to the fread and fwrite functions.

In MATLAB, if x is a memory-mapped variable, and y is the data to be written to a file, then writing to the file is as simple as

```
data = y;
```

Sharing Memory Between Applications

Memory-mapped files also provide a mechanism for sharing data between applications, as shown in the figure below. This is achieved by having each application map sections of the same file. You can use this feature to transfer large data sets between MATLAB and other applications.



Also, within a single application, you can map the same segment of a file more than once.

When to Use Memory-Mapping

Just how much advantage you get from mapping a file to memory depends mostly on the size and format of the file, the way in which data in the file is used, and the computer platform you are using.

When Memory-Mapping Is Most Useful

Memory-mapping works best with binary files, and in the following scenarios:

- For large files that you want to access randomly one or more times
- For small files that you want to read into memory once and access frequently
- For data that you want to share between applications
- When you want to work with data in a file as if it were a MATLAB array

When the Advantage Is Less Significant

The following types of files do not fully use the benefits of memory-mapping:

- Formatted binary files like HDF or TIFF that require customized readers are not good for memory-mapping. Describing the data contained in these files can be a very complex task. Also, you cannot access data directly from the mapped segment, but must instead create arrays to hold the data.
- Text or ASCII files require that you convert the text in the mapped region to an appropriate type for the data to be meaningful. This takes up additional address space.
- Files that are larger than several hundred megabytes in size consume a significant amount of the virtual address space needed by MATLAB to process your program. Mapping files of this size may result in MATLAB reporting out-of-memory errors more often. This is more likely if MATLAB has been running for some time, or if the memory used by MATLAB becomes fragmented.

Maximum Size of a Memory Map

Due to limits set by the operating system and MATLAB, the maximum amount of data you can map with a single instance of a memory map is 2 gigabytes on 32-bit systems, and 256 terabytes on 64-bit systems. If you need to map more than this limit, you can either create separate maps for different regions of the file, or you can move the window of one map to different locations in the file.

Byte Ordering

Memory-mapping works only with data that have the same byte ordering scheme as the native byte ordering of your operating system. For example, because both Linus Torvalds' Linux® and Microsoft® Windows® systems use little-endian byte ordering, data created on a Linux system can be read on Windows systems. You can use the computer function to determine the native byte ordering of your current system.

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=jkT9Bgz8PAg

Important Books/Journals for further learning including the page nos.:
Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 465-472)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

ECE

LECTURE HANDOUTS

II/IV

L42

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : V - ARM CORTEX M3 PROGRAMMING **Date of Lecture:**

Topic of Lecture: BIT-BAND OPERATION

Introduction :

The bit-band region operation is like that of the ARM Cortex-M3 and ARM Cortex-M4 processors. It maps a complete word of memory onto a single bit in the bit-band region. For example, writing to one of the alias words sets or clears the corresponding bit in the bit-band region.

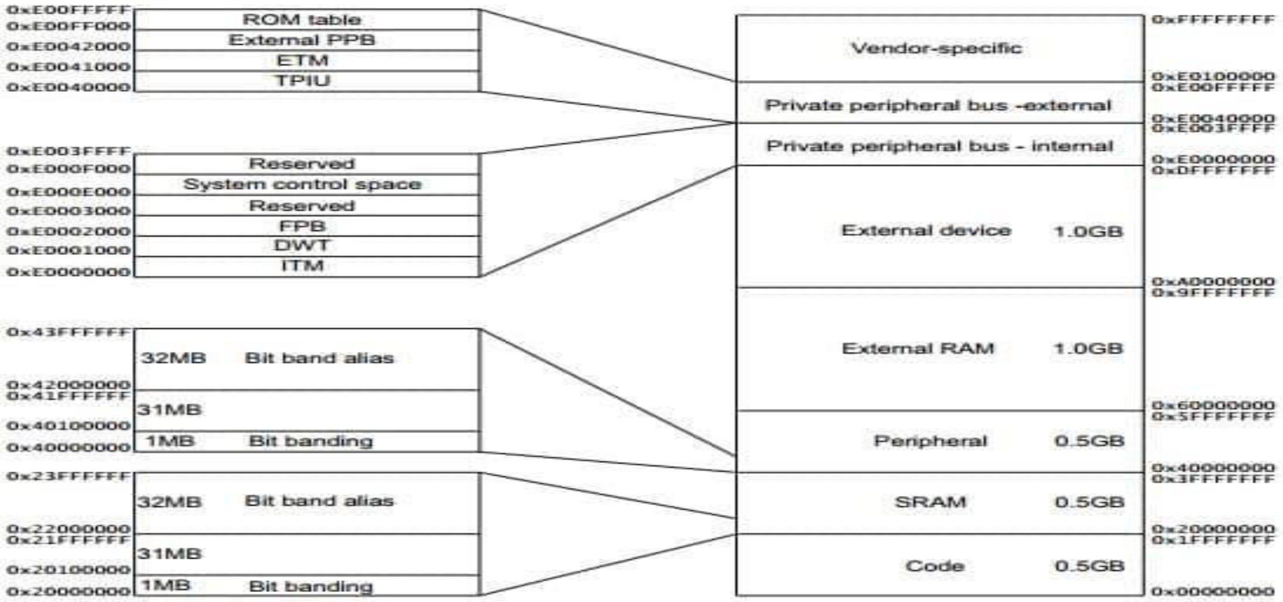
Prerequisite knowledge for Complete understanding and learning of Topic:

- Basic operations of 8085 & 8051

Detailed content of the Lecture: Bit-Band Operation

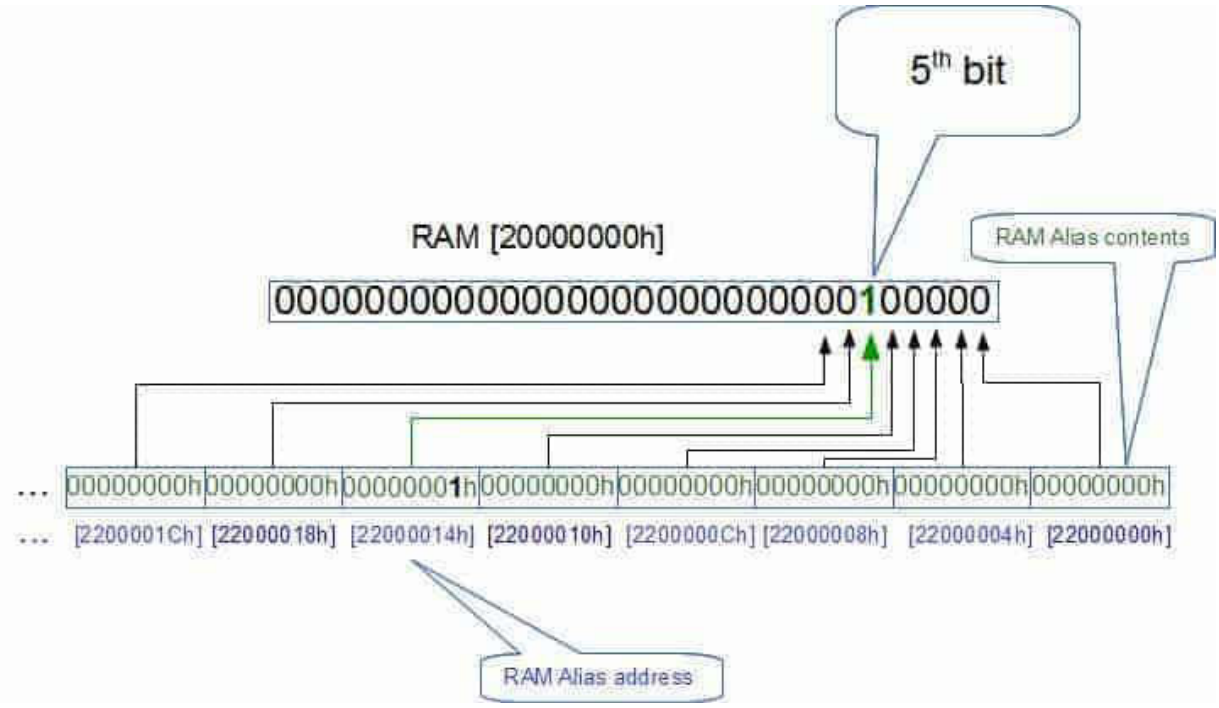
- If you open any ARM Cortex documentation or datasheet, you will find that bit-band can be performed in two memory regions – the first 1MB SRAM region (from address 20000000h to 200FFFFFh) and the first 1MB peripheral region (from address 40000000h to 400FFFFFh). What does this mean? This means that bitwise operations can be performed to RAM and peripherals like ports and other peripheral registers.

The bit band mechanism works by using a separate memory region called the bit-band alias. Alias regions are located far from available RAM or actual peripherals.



As you can see for RAM, this region starts at address 22000000h, from 31MB. This is a safe location as ARM internal SRAM will not likely reach 32MB. The same situation is with the peripheral region. It also starts are a 31MB location (at address 42000000h).

Using the bit-band alias memory region, we can access individual bits of RAM or peripherals. Each bit in memory is addressed using a 32-bit bit-band address in the alias memory region.


5th bit
RAM [20000000h]
RAM Alias contents
00000000000000000000000000100000
... 00000000h 00000000h 00000001h 00000000h 00000000h 00000000h 00000000h 00000000h
... [2200001Ch] [22000018h] [22000014h] [22000010h] [2200000Ch] [22000008h] [22000004h] [22000000h]
RAM Alias address

- Let's say you want to change the 5th bit of RAM contents stored at address 20000000h. Since this is the first memory address from the alias region, we can find an alias address pretty quickly. The first alias address is 22000000h, which is the address of the first (actually 0) bit in RAM. The second 1st bit in the alias has an address of 22000004h (remember each 32-bit word address is increased by 4), the second bit is addressed with 22000008h, and so on. So the fifth bit has an alias address 22000014h. If you want to write or read a bit from the SRAM word, you need to write or read 0-bit from the corresponding bit band address. In other words, you need to write 1 to RAM address 22000014h to set the 5th bit in RAM location 20000000h. This applies to bit read operation as well.
- How to calculate alias address to any bit in RAM location? For this, you can do a simple calculation. Let us say we want to access peripheral bits. We know that the peripheral area starts at address 40000000h. So we call this address as Peripheral Base; then peripheral Alias Base is 42000000h. Let's say we want to write 1 to PORT D 5th pin 1. We need to set this bit to 1; From the memory map, we find that the PORT D output register address is 4001140C**h**. So this is 1140C**h** – the byte in peripheral memory. Since each byte in memory offsets address by 20h in the alias memory region, we can calculate that the PORTD output register alias starts at 1140Ch * 20h. Then we need to add a 5th bit: + 5×4. So our alias address for PORT D output register GPIOD_ODR pin 5 is:

42000000h + 1140C**h** * 20h + 5 * 4 = 42228194**h**

- In C program we can define this bit as a pointer and use it to set or clear this bit:

```
#define PortDBit5 (*((volatile unsigned long *) 0x42228194))
```

Then anywhere in the program when you need to set this bit to 1 you can write:

```
PortDBit5 = 1;
```

Or write zero:

```
PortDBit5 = 0;
```

In C program you can prepare a MACRO to calculate the bit-band location for particular bit location.

```
#define PERIPHERAL_BASE 0x40000000
```

```
#define BITBAND_PERIPHERAL_BASE 0x42000000
#define BITBAND_PERIPHERAL(a,b) (BITBAND_PERIPHERAL_BASE + (a-PERIPHERAL_BASE)*0x20+(b*4
//port D output register address
#define GPIOD_ODR 0x4001140C
//create a pointer to bit 5 in bit band location
#define PORTDBIT5 (*((volatile unsigned long *)(BITBAND_PERIPHERAL(GPIOD_ODR,5))))
//use it somewhere to set bit to 1
int main()
{
//...
```

Video Content / Details of website for further learning (if any):

www.youtube.com/watch?v=liRtPtvj7bFU&noredirect=1

Important Books/Journals for further learning including the page nos.:
Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 465-472)

Course Faculty

Verified by HOD

L 43

ECE | LECTURE HANDOUTS | II/IV

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : V - ARM CORTEX M3 PROGRAMMING **Date of Lecture:**

Topic of Lecture: CMSIS

Introduction :

- The Common Microcontroller Software Interface Standard (CMSIS) is a vendor-independent abstraction layer for microcontrollers that are based on Arm Cortex processors. ... CMSIS provides interfaces to processor and peripherals, real-time operating systems, and middleware components.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Interfacing of 8051

Detailed content of the Lecture: CMSIS

- The **Common Microcontroller Software Interface Standard (CMSIS)** is a vendor-independent abstraction layer for microcontrollers that are based on Arm Cortex processors. CMSIS defines generic tool interfaces and enables consistent device support. The CMSIS software interfaces simplify software reuse, reduce the learning curve for microcontroller developers, and improve time to market for new devices.

- CMSIS provides interfaces to processor and peripherals, real-time operating systems, and middleware components. CMSIS includes a delivery mechanism for devices, boards, and software, and enables the combination of software components from multiple vendors.

CMSIS components

CMSIS-...	Target Processors	Description
Core(M)	All Cortex-M, SecurCore	Standardized API for the Cortex-M processor core and peripherals. Includes intrinsic functions for Cortex-M4/M7/M33/M35P SIMD instructions.

Core(A)	Cortex-A5/A7/A9	Standardized API and basic run-time system for the Cortex-A5/A7/A9 processor core and peripherals.
Driver	All Cortex	Generic peripheral driver interfaces for middleware. Connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces.
DSP	All Cortex-M	DSP library collection with over 60 Functions for various data types: fixed-point (fractional q7, q15, q31) and single precision floating-point (32-bit). Implementations optimized for the SIMD instruction set are available for Cortex-M4/M7/M33/M35P.
NN	All Cortex-M	Collection of efficient neural network kernels developed to maximize the performance and minimize the memory footprint on Cortex-M processor cores.
RTOS v1	Cortex-M0/M0+/M3/M4/M7	Common API for real-time operating systems along with a reference implementation based on RTX. It enables software components that can work across multiple RTOS systems.
RTOS v2	All Cortex-M, Cortex-A5/A7/A9	Extends CMSIS-RTOS v1 with Armv8-M support, dynamic object creation, provisions for multi-core systems, binary compatible interface.
Pack	All Cortex-M, SecurCore, Cortex-A5/A7/A9	Describes a delivery mechanism for software components, device parameters, and evaluation board support. It simplifies software reuse and product life-cycle management (PLM).
SVD	All Cortex-M, SecurCore	Peripheral description of a device that can be used to create peripheral awareness in debuggers or CMSIS-Core header files.
DAP	All Cortex	Firmware for a debug unit that interfaces to the CoreSight Debug Access Port.
Zone	All Cortex-M	Defines methods to describe system resources and to partition these resources into multiple projects and execution areas.

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=ttXW58-drYw

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 476-481)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L 45

ECE

II/IV

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : **V - ARM CORTEX M3 PROGRAMMING** **Date of Lecture:**

Topic of Lecture: CMSIS

Introduction :

CMSIS started as a vendor-independent hardware abstraction layer Arm® Cortex®-M based processors and was later extended to support entry-level Arm Cortex-A based processors. To simplify access, CMSIS defines generic tool interfaces and enables consistent device support by providing simple software interfaces to the processor and the peripherals.

Prerequisite knowledge for Complete understanding and learning of Topic:

- CMSIS

Detailed content of the Lecture: CMSIS

- CMSIS has been created to help the industry in standardization. It enables consistent software layers and device support across a wide range of development tools and microcontrollers. CMSIS is not a huge software layer that introduces overhead and does not define standard peripherals. The silicon industry can therefore support the wide variations of Arm Cortex processor-based devices with this common standard.

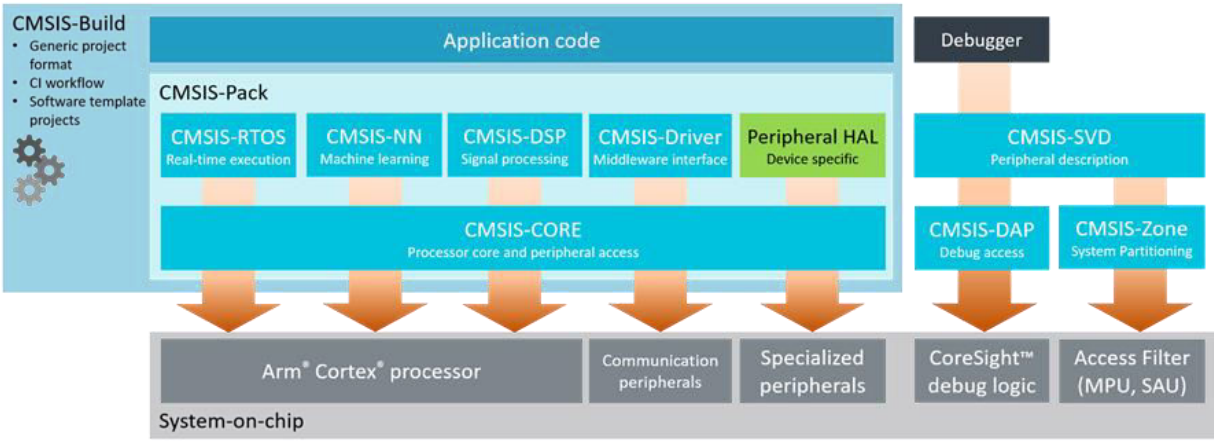


Fig : CMSIS Structure

The benefits of the CMSIS are:

- CMSIS reduces the learning curve, development costs, and time-to-market. Developers can write software quicker through a variety of easy-to-use, standardized software interfaces.
- Consistent software interfaces improve the software portability and re-usability. Generic software libraries and interfaces provide consistent software framework.
- It provides interfaces for debug connectivity, debug peripheral views, software delivery, and device support to reduce time-to-market for new microcontroller deployment.
- It allows to use the compiler of your choice, as it is compiler independent and thus supported by mainstream compilers.
- It enhances program debugging with peripheral information for debuggers and ITM channels for printf-style output.
- CMSIS is delivered in CMSIS-Pack format which enables fast software delivery, simplifies updates, and enables consistent integration into development tools.
- CMSIS-Zone will simplify system resource and partitioning as it manages the configuration of multiple processors, memory areas, and peripherals.
- Continuous integration is common practice for most software developers nowadays. CMSIS-Build supports these workflows and makes continuous testing and validation easier.

Coding Rules

The CMSIS uses the following essential coding rules and conventions:

- Compliant with ANSI C (C99) and C++ (C++03).
- Uses ANSI C standard data types defined in **<stdint.h>**.
- Variables and parameters have a complete data type.
- Expressions for #define constants are enclosed in parenthesis.
- Conforms to MISRA 2012 (but does not claim MISRA compliance). MISRA rule violations are documented.

In addition, the CMSIS recommends the following conventions for identifiers:

- **CAPITAL** names to identify Core Registers, Peripheral Registers, and CPU Instructions.
- **CamelCase** names to identify function names and interrupt functions.
- **Namespace_** prefixes avoid clashes with user identifiers and provide functional groups (i.e. for peripherals, RTOS, or DSP Library).

The CMSIS is documented within the source files with:

- Comments that use the C or C++ style.
- Doxygen compliant **function comments** that provide:
  - brief function overview.
  - detailed description of the function.
  - detailed parameter explanation.
  - detailed information about return values.

Validation

The various components of CMSIS are validated using mainstream compilers. To get a diverse coverage, Arm Compiler v5 (based on EDG front-end), Arm Compiler v6 (based on LLVM front-end), and GCC are used in the various tests. For each component, the section **"Validation"** describes the scope of the various verification steps.

CMSIS components are compatible with a range of C and C++ language standards. The CMSIS components comply with the Application Binary Interface (ABI) for the Arm Architecture (exception CMSIS-RTOS v1). This ensures C API interfaces that support inter-operation between various toolchains.

As CMSIS defines API interfaces and functions that scale to a wide range of processors and devices, the scope of the run-time test coverage is limited. However, several components are validated using dedicated test suites (CMSIS-Driver, CMSIS-RTOS v1, and CMSIS-RTOS v2).

The CMSIS source code is checked for MISRA C:2012 conformance using PC-Lint. MISRA deviations are documented with reasonable effort, however Arm does not claim MISRA compliance as there is today for example no guideline enforcement plan. The CMSIS source code is not checked for MISRA C++:2008 conformance as there is a risk that it is incompatible with C language standards, specifically warnings that may be generated by the various C compilers.

-

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=ttXW58-drYw

Important Books/Journals for further learning including the page nos.:
Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 482-490)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L 45

ECE

II/IV

Course Name with Code: 19ECC07/MICROCONTROLLER BASED SYSTEM DESIGN

Course Faculty : Mrs.S.Punitha

Unit : V - ARM CORTEX M3 PROGRAMMING **Date of Lecture:**

Topic of Lecture: Assembly Language Programming

Introduction :

The programming process is independent of the language which is used for programming, but in a more realistic sense different languages encourage different programming techniques to be used. The machine which is being used for assembly language programming, since it affects the instruction set, also affects the type of programming which can be easily done. The result of this is that there is a body of general programming techniques which can be used for programming in assembly language for most machines

Prerequisite knowledge for Complete understanding and learning of Topic:

- Assembly language programming

Detailed content of the Lecture: Assembly Language Programming

Computers were originally built to automate arithmetic calculations, and many people still believe this is their major function. Thus the coding of arithmetic expressions is a logical place to start. (In fact, actual arithmetic computation is only a minor part of what computers do. Remember that in the machine language program of Section 2.2, only 1 instruction out of 16 was an arithmetic computation; the others were input/output, conversion, loads and stores, etc). Here we assume we have a set of integer numbers upon which we wish to compute some simple arithmetic expression.

To start, consider a simple addition. We wish to compute the sum of the value of the variables CAT and DOG and store the sum in the variable FIGHT. This code might look like

```
CAT       CON   0                    DATA LOCATION FOR CAT
DOG       CON   0
FIGHT     CON   0
          ...
          <code to give CAT and DOG values>
          ...
          LDA  CAT
          ADD  DOG
          STA  FIGHT              SET FIGHT = CAT + DOG
```

If we wanted FIGHT = CAT - DOG, we have (assuming the data declarations above)

```
          LDA  CAT
          SUB  DOG
          STA  FIGHT              SET FIGHT = CAT - DOG
```

For ZOO = DOG * CAT, the code needs to consider that the product of two 5-byte numbers will be a 10-byte number with the upper 5 bytes in the A register and the lower 5 bytes in the X register. If we expect that the product will always be no more than 5 bytes in magnitude, we can write

```
          LDA  DOG
          MUL  CAT                PUTS PRODUCT IN AX
          STX  ZOO                ZOO = DOG*CAT LOWER 5 BYTES
```

Notice that this code fragment does not worry about overflow. If we fear that the product DOG*CAT may be too large for five bytes, we can write

```
          LDA  CAT
          MUL  DOG                AX = DOG * CAT
          JANZ TOOLARGE           CHECK FOR PRODUCT TOO BIG
          STX  ZOO
```

For this code, if the product is greater than five bytes, the A register will have the upper five bytes, which will be non-zero. In this case the program will jump to the location whose label is TOOLARGE and execute the code there. If the A register is zero, the product is small enough to fit in five bytes (and is in the X register), so this is stored in the memory location ZOO. Remember, we still have to write the code for TOOLARGE. This code might be a simple

```
TOOLARGE  HLT  0                  OVERFLOW, STOP
```

or more complicated (and better) code which prints a message saying what went wrong and informs the user of the program what steps can be taken to correct the problem (change data, rewrite program, or such).

Division must also be coded carefully. If we want RATIO = NMEN / NWOMEN, we need the dividend (NMEN in this case) as a 10-byte quantity in register AX for the DIV operator. Normally this means putting the dividend in the X register and zero in the A register. This can be done in several ways. The major constraint in how it is done is that the sign of the dividend is the sign of the A register. Thus, if you know that your dividend is positive, division can be

```
          LDX  NMEN
          ENTA 0
          DIV  NWOMEN             COMPUTE NMEN/NWOMEN
```

If the sign of the dividend can be either positive or negative, then code can be

```
          LDA  NMEN(0:0)          PUT SIGN IN A
          LDX  NMEN               LOAD NUMBER IN X
          DIV  NWOMEN             DIVIDE NMEN BY NWOMEN
```

which loads the sign into the A register (setting the rest to zero) and the rest of the dividend into the X register, or the code can be,

```
          LDA  NMEN                LOAD NUMBER IN A
          SRAX 5                   SHIFT INTO X, LEAVING SIGN
          DIV  NWOMEN              DIVIDE
```

In this case the number is loaded into the A register and shifted into X. SRAX is an end-off shift, so zeros are shifted into the A register and the old contents of the X register are lost. The sign bits do not participate in the shift, so the sign of the dividend remains in the sign bit of the A register.

Once the division has been done, the quotient will be in the A register, and the remainder in the X register. Thus, these two quantities can be used, as desired, and the same code can be used to compute either the quotient or the remainder (or both). Remember that the DIV operator is an *integer* division. Fractions would require the use of floating point arithmetic.

Once these basic operations are understood, it is possible to compute much more complicated expressions, like XI = I + J*L1. This expression would be programmed to first perform the multiplication, then the addition. To do the multiplication, we would

```
          LDA  J
          MUL  L1                  AX = L1*J
```

Now we have J*L1, but it is in the X register. The ADD operator only adds to the A register, so we must get our result into the A register for the addition. This could be done by

```
          STX  TEMP
          LDA  TEMP                TRANSFER FROM X TO A
```

where TEMP is a *temporary* variable, or we could simply

```
          SLAX 5                   SHIFT X INTO A
```

which is faster (2 time units instead of 4) and shorter (1 instruction instead of 2). Notice that we are taking advantage of the sign bits of the A and X registers as set by the MUL operator. Now we can add and store. Our complete code is then

```
          LDA  J
          MUL  L1                  COMPUTE J*L1
          SLAX 5                   MOVE X TO A
          ADD  I
          STA  X1                  X1 = I + J*L1
```

Even more complicated expressions may be computed. Consider the expression

```
((B+W)*Y) + 2 + ((L-M)*(-K))/Z
```

This is the sum of three terms. This means that at least one term must be stored in memory, in a temporary, while the other term is being computed. The addition by 2 can be done by either

```
          ADD  =2=                 INCREASE A BY 2
```

or

```
        INCA 2                 INCREASE A BY 2.
```

Since the latter is shorter and faster, we use it. (Shorter because although both instructions are the same length, the ADD also requires a literal, which is one memory location).

Since the first term is simpler, let us attack it first. We can write

```
        LDA  B
        ADD  W                 A = B+W
        MUL  Y                 AX = (B+W)*Y
```

Now we can add the 2, but remember the result of the MUL will be in AX, so we

```
        INCX 2                 X = (B+W)*Y + 2
```

This is even better than we had thought, over the ADD approach, since it prevents having to move the product from the X register to the A register for the ADD. Now we need to store this while we compute the next term.

```
        STX  TEMP              SAVE PARTIAL TERM
```

Next we compute ((L-M)*(-K))/Z. This can be done easiest by noting that ((L-M)*(-K)) = ((M-L) * K), which removes one operation, and we can now write,

```
        LDA  M
        SUB  L                 M-L
        MUL  K                 (M-L) * K
        DIV  Z                 ((M-L)*K) / Z.
```

Notice that the MUL leaves the AX register just right for a DIV. This is one reason for doing our multiply first. Another reason is due to the integer nature of the division. If (M-L) = 100, K = 50, and Z = 100, consider the result of computing (M-L)*(K/Z) instead of ((M-L)*K)/Z. For (M-L)*(K/Z), K/Z = 0 and the product is 0, while for ((M-L)*K)/Z, the product is 5000 and the quotient is 50. This illustrates the wide difference which can result from carelessness with the order of multiplication and division.

Once the division is done, we can add from our temporary, TEMP, to the quotient, which is conveniently in the A register. Our complete code is then

```
        LDA  B                 B
        ADD  W                 B+W
        MUL  Y                 (B+W)*Y
        INCX 2                 ((B+W)*Y)+2
        STX  TEMP              TEMP = ((B+W) * Y) + 2
        LDA  M                 M
        SUB  L                 M-L
        MUL  K                 (M-L) * K
        DIV  Z                 ((M-L)*K) / Z
        ADD  TEMP              ((B+W)*Y) + 2 + ((M-L)* K)/Z
```

Overflow is a problem which we have not considered in this code. Notice that at any (arithmetic) instruction overflow could occur (except for the DIV, but if Z = 0, this will act the same as overflow). One of the nice features of MIX is that the overflow toggle is turned on whenever overflow occurs, but is not turned off when it does not occur. This means that if it is off before we begin to execute this code, and is on after the code is executed then overflow has occurred somewhere in the expression. Often it is not important to know exactly where overflow has occurred, but only if it has occurred at any time during the evaluation of the expression. This can be tested by one test at the end of the expression computation as,

```
        JOV   OVERFLOW
```

Remember that this will also turn the overflow toggle off so that it can be used for the next segment of code. This approach to testing overflow requires that the overflow toggle be off before the computation of the arithmetic expression is begun. How do we guarantee that it is off? The easiest way is to test for overflow whenever it may occur. This assures that as soon as it happens, it will be recognized and appropriate action taken. Alternatively, we will need to turn the overflow toggle off explicitly. Searching the list of machine instructions, we find that there is no instruction to turn off the overflow toggle. But consider

```
        JOV   *+1
```

This instruction will not jump if the overflow is off, and if it is on, it will turn the overflow off, and jump to `*+1`. In either case the next instruction will be at `*+1`, with the overflow toggle off.

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=HXYhBCpDoVc

Important Books/Journals for further learning including the page nos.:

Muhammad Ali Mazidi & Janice Gilli Mazidi, R.D.Kinely, The 8051 Micro Controller and Embedded Systems, PHI Pearson Education, 5th Indian reprint, 2003 (Page No: 482-490)

Course Faculty

Verified by HOD