UNIT - I

INTRODUCTION TO MICROPROCESSORS :

Historical background -

Evolution of microprocessors up to 64-bit

Architecture of 8086 microprocessor

special function of general purpose registers

8086 flag registers & functions of 8086 flags

Addressing modes of 8086

Instruction set of 8086

Assembler directives

Pin diagram 8086

Minimum mode and Maximum mode of operation,

Timing diagrams

CISC and ARM processors.

Unit-I

Introduction to Microprocessors.

Microprocessors:-

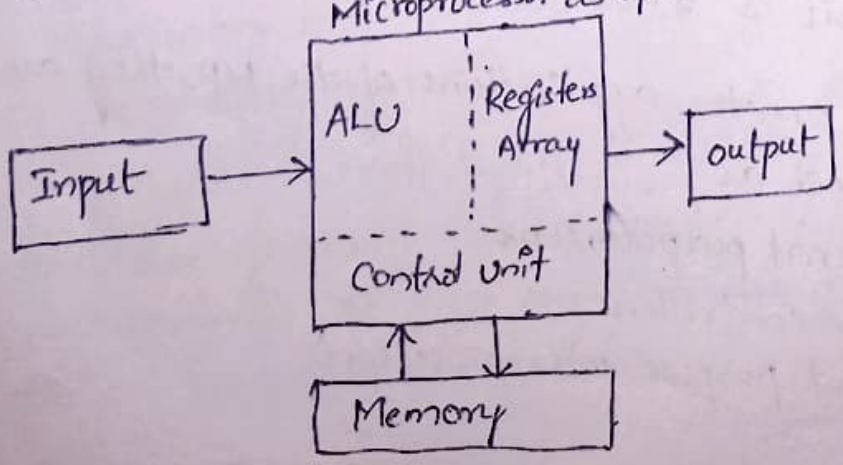
A microprocessor is a multipurpose, programmable, clock driven, register based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input & processes data according to those instructions & provides results as output.

(or)

The microprocessor is an electronic chip that functions as the central processing unit (cpu) of a computer.
In other words, the microprocessor is the heart of any computer system.

The microprocessor system consists of three functional blocks. Central processing unit (cpu), input & output units, & memory units as shown in the fig.

The cpu contains several registers, an Arithmetic & logic unit (ALU), & a control unit. The function of ALU, is to perform an arithmetic & logical operations. The control unit translates the instructions & executes the desired task.



Input & Output devices :-

Input & output devices permit the user to feed data to the computer & retrieve the computed results from it. The input & output devices sometimes can communicate among themselves. In general, computer systems have I/o ports. I/o devices are connected to these ports for data transfer.

Examples of i/o devices are keyboard, mouse, monitor printer etc.

Classification of Microprocessors :-

Microprocessors can be classified based on their specifications, applications & architecture.

→ Based on the size of the data, that the microprocessors can handle, they are classified as

1. 4-bit
2. 8-bit
3. 16-bit
4. 32-bit & 5. 62 bit microprocessors

→ Based on the applications of the µp, they are classified as

1. General purpose µps
2. Microcontrollers
3. Special purpose microprocessors.

→ General purpose processors are those that are used in general computer system

→ Microcontrollers are microprocessors chip with built-in hardware for the memory & ports.

→ Special purpose processors are designed specifically to handle special functions required for an application.

Based on the architecture & hardware of the processors, they are classified as

(1) RISC processors

(2) CISC processors.

Types of Memory :—

Memory unit is an integral part of any computer system. Its primary purpose is to hold program & data. The main objective of the memory unit is to enable it to operate at a speed close to that of the processor. A microprocessor memory can be logically divided into four groups.

1. processor memory / registers
2. cache memory
3. primary or main memory
4. secondary memory.

→ processor memory refers to a set of cpu registers. They are generally few in number - upto a few ten or hundreds. As these registers are available within the processor, they are the fastest memory registers.

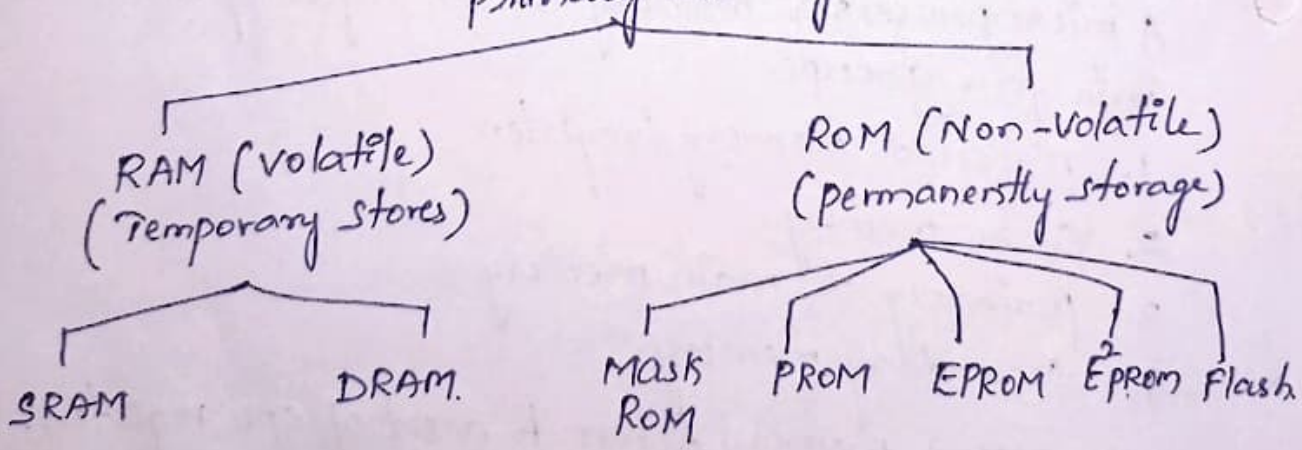
→ Cache memory is the fastest external memory, it is placed close to the processor. They are few Kilobytes in size.

→ primary memory is the storage area from which all the programs are executed. All the programs & corresponding data for execution must be within the primary memory.
The primary memory is much larger than the processor memory & cache memory. but its operating speed is slower.

→ The size of the primary memory varies from few KB to a few MB.

→ Secondary memories refers to the storage medium for huge files such as program source codes, compilers, operating systems etc.

classification of primary memory:-



Historical Background :- Evolution of µps :-

The microprocessor is the greatest invention of the 20th century. Its evolution started from the earlier mechanical calculating devices, in the 1930's. These devices used mechanical relays.

→ Later, In the 1950's, these devices were replaced by vaccum tubes. The vaccum tubes were quickly replaced by transistors. The breakthrough in transistor technology led to the introduction of minicomputers in the 1960's & the personal computer revolution in the 1970's.

The transistor technology led to the development of complex devices called Integrated circuits (IC). The µp later evolved as an IC and was designed to fetch instructions & execute the predefined arithmetic & logic functions. Intel was the first Mpu producer & has been holding a large share of the world market for this product.

The evolution of microprocessors is categorised into five ~~categories~~ generations:

1. first generation :- (1971-1973)

The microprocessors that were introduced from 1971 to 1973 were referred to as first generation systems.

→ In this generation, 4-bit & 8-bit µp's were introduced.

→ In this PMOS technology was used.
(P-channel Metal Oxide Semiconductor technology)

2. Second generation (1974-1978).

→ The microprocessors that were introduced from 1974 to 1978 were referred to as second-generation systems.

→ This generation marked the beginning of very efficient-8-bit μps which were manufactured using N-MOS technology.

(N-channel metal oxide semiconductor) Technology.

3. Third Generation (1978-1980)

→ The μps that were introduced from 1978 to 1980 were referred to as third-generation systems.

→ This generation marked the beginning of 16-bit processors which were manufactured by using High density metal-oxide semiconductor (HMOS) technology which provides some advantages over NMOS technology.

4. fourth generation (1981-1995).

→ The μps that were introduced from 1981 to 1995 were referred to as fourth generation systems.

→ This era marked the beginning of 32-bit μps. which were manufactured by using High density/ high speed complementary metal oxide semiconductor (HCMOS) technology.

5. fifth generation (1995- till date)

→ The µps that were introduced from 1995 to till date were referred to as fifth generation systems.

→ This era marks the introduction of devices that carry on-chip functionalities.

→ This generation marked the beginning of 64-bit microprocessors.

General purpose microprocessor	Year of Introduction	No. of transistors	CPU clock Speed	Data length (bits)
Intel 4004	1971	2,300 at 10µm	740 KHz	4-bit
4040	1974	3000 at 10µm	500 - 740 KHz	4-bit
8008	1972	3,500 at 10µm	500 KHz	8-bit
8080	1974	4,500 at 10µm	2 MHz	8-bit
8085	1976	6,500 at 3µm	3 MHz	8-bit
8086	1978	29,000 at (3µm)	4 MHz	16-bit
8088	1979	30,000 at 3µm	5 - 10 MHz	16-bit
80186	1982	55,000 at 3µm	6 MHz.	16-bit.
80286	1982	1,34,000 at 1.5µm	8 MHz	16-bit.

80386	1985	2,75,000 at 1.5μm	16-33 MHz	16/32 bit
80486	1989	1.2 M. at 1μm	16-100MHz	16/32bit
Intel pentium	1993	3.1M at 0.8μm	60-66mHz	32 bit
Intel pentium pro	1995	5.5M at 0.5μm	150-200mHz	32 bit
Intel pentium II	1997	7.5M at 0.35μm	233-450 mHz	32/64 bit
Intel pentium III	1999	9.5M at 0.25μm	0.45-1.4GHz	32/64 bits
Intel pentium IV	2000	42M at 0.18μm	1.3-3.8GHz	32/64 bits
Intel Atom	2008	47M at 45nm	0.6-2.13 GHz	32/64 bit
Intel celeron	1998	-	0.26-3.6 GHz	32/64 bits
Intel Xeon	1998	-	0.4-4.4 GHz	32/64 bit
Intel pentium dual core	2006	228M at 90nm	1.3-2.6GHz	32/64 bits
Intel Core2 series	2006	291 M at 45nm	1.06-3.33 GHz	64 bit.
Intel Core i3	2010		1.2-3.7GHz	64 bits
Intel core i5	2009		1.06-3.6GHz	64 bit
Intel core i7	2008		1.6-4.4GHz	64 bit

Introduction to 8086 MP:-

In 1978, Intel released its first 16-bit microprocessor, the 8086. It is a 16-bit microprocessor. It means that it has a 16-bit data bus and operates on 16-bit data using a single instruction.

features of 8086 MP:- → The 8086 is a 40-pin IC.

→ The 8086 is a 16-bit processor. It means that ALU, its internal registers and most of its instructions are designed to work with 16-bit data

→ The 8086 has a 16-bit data bus.

→ It has a 20-bit address bus, so it can have a maximum of 1 MB of memory (2^{20} bytes = 1 MB)

→ It has 16-bit I/o address so it can access upto 64K I/o Ports

→ The 8086 has 14 (fourteen) number of 16 bit register

→ The 8086 executes the instruction at 2.5 MIPs (Million Instructions per second)

→ The execution time for 1 instruction is 400 ns.
($1/MIPS = 1/(2.5 \times 10^6)$).

→ The 8086 has a clock speed of 4.77 – 10 MHz.

→ It consists of 29,000 transistors at 3μm.

→ The 8086 has a powerful instruction sets with a range of addressing modes. It can perform bit, byte, word & block operations.

→ The 8086 has two modes of operations, i.e., minimum & maximum.

In minimum mode, it works as a single microprocessor whereas in maximum mode, it can work in multiprocessor configuration.

Architecture of 8086 MP:-

The functional block diagram of 8086 is shown in the fig. below. As seen in the block diagram, it is internally divided into two separate units.

These are (i) Execution Unit (EU) &
(ii) Bus Interface unit (BIU)

The division of work between these units speeds up the processing.

Execution unit (EU):-

The Execution unit is responsible for the instruction Execution & has a control circuit to control its internal operations.

→ An Execution unit includes the ALU, eight 16-bit general purpose registers, a 16-bit flag register & a control unit.

Architecture of 8086 µP -

(6)

Ax → Accumulator.

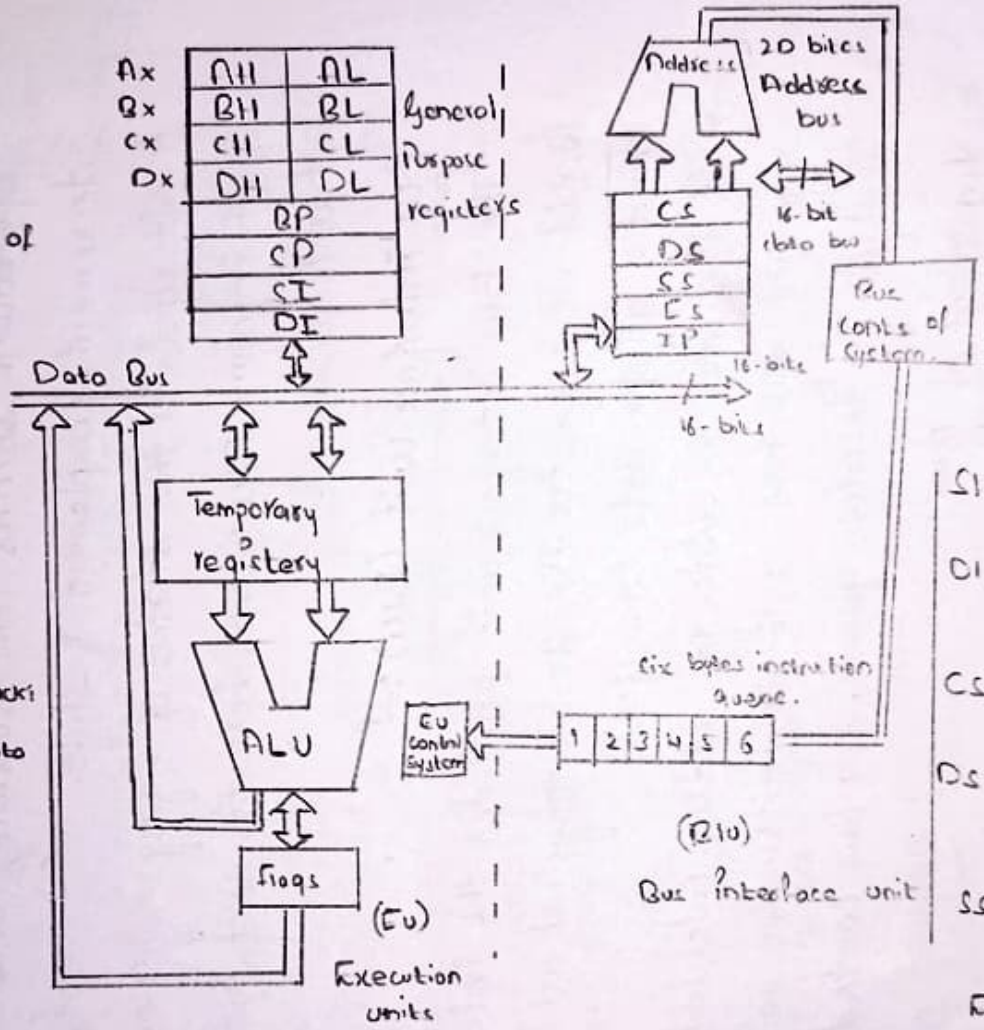
Bx → Holds the off sets address of a location in memory

Cx → Holds the Count Value while excuting repeating string and loop instructions (also in shift/rotate instruction)

Dx → Holds a part of a result during mul/div operations.

Sp → Holds the offset address of the data stored at the top of the stack.

Bp → Holds the offset address of the data to be read from written on to the stack.



SI → Holds the offset address of the source data in d.s.

DI → Holds the offset address of the source distination data.

CS → Starting (or) base address of the code segment

DS → Starting (or) base address of the data segment.

SS → Starting (or) base address of the stack segment

ES → Starting (or) base address of Extra segment.

functions of Execution unit are as follows.

1. It picks up the instructions from the queue of BIU.
2. It decodes the instructions & decides upon the various operations it has to carry out. then it executes these operations
3. It uses ALU, to perform 16-bit arithmetic & logical operations such as add, subtract, AND, OR, XOR, increment decrement, Complement & shift.
4. It updates the status of a flag register
5. It informs BIU from where the next instruction or data has to be read.

Bus interface unit (BIU) :-

The Bus interface unit provides the interface to external world. It generates the 20-bit physical address and handles all the data transfers.

→ It includes an adder for address calculations, four 16-bit segment registers, a 16-bit instruction pointer (IP), a six-byte instruction queue, & a bus control logic.

functions of BIU :-

1. It fetches instruction from memory.
2. It reads/writes data from memory/ports
3. It sends the addresses to memory/ports
4. It supports an Instruction queue

The Queue :- The BIU fetches upto six instruction bytes belonging to next instructions. These bytes are stored in a First-in-first out (FIFO) register set called Queue. These bytes (pre-fetched instructions) are taken out by Execution unit for further processing. While EU does this, further instruction bytes are fetched & stored in the Queue. Hence fetching of the next instructions & executions of the current instruction is done simultaneously.

This technique is called pipelining & speeds up processing.

Register organisation of 8086 :-

The 8086 contains fourteen 16-bit register. they are of different types.

1. General purpose registers
2. Segment registers
3. Index registers
4. Pointer registers.
5. flag or status registers.

1. General purpose registers:-

These are four 16-bit registers - AX, Bx, CX & DX. Each of these registers consists of two 8-bit registers. one of which holds MSByte & other will hold LS Byte. as shown below.

The general purpose registers are used for temporary storage of data & some intermediate results.

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

General purpose registers.

The Special functions of General Purpose registers are as follows :-

1. Register AX (Accumulator) :-

→ The register AX is also called as Accumulator
→ It is used as a default register in operations like multiply, divide, input, output & data conversion (BCD, ASCII).

2. Register Bx (Base register) :-

→ It is used as a base register
→ It contains the offset address of the memory location in some addressing modes.

3. Register Cx (count register) :-

→ It is used as a default counter in loop & string instructions.
→ It is also used in shift & rotate instructions as a 8-bit counter.

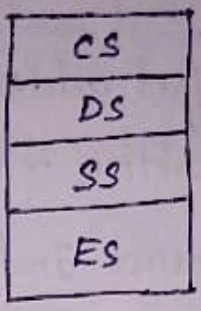
4. Register Dx (Data register) :-

→ It is used to hold a part of the result during a multiplication operation & a part of the dividend before a division operation.
→ It is also used to store I/o addresses in I/o instructions.

(ii) Segment registers :-

There are four segment registers

1. Code segment register (CS)
2. Data segment register (DS)
3. Stack segment register (SS)
4. Extra segment register (ES).



Segment registers.

→ The code segment register is used for addressing a memory location in the code segment of the memory where the executable ~~mem~~program is stored.

→ The Data segment register is used to points the data segment of the memory where the data is resided.

→ The stack segment register is used for addressing stack segment of the memory where the stack data is stored.

→ The Extra segment register refers to a segment which essentially is another data segment of the memory i.e., the extra segment also contains data.

(iii) pointer & Index registers :-

Stack pointer (SP) :-

The stack pointer (SP) is used to hold the offset address of the data stored at the top of the stack segment.

Pointer & Index registers

Base pointer (Bp):- It is used to store (or) hold the offset address of the data to be read from or written into the stack segments.

Source Index (SI) register :- The SI is used to hold the offset address of the source data in the data segment while executing string instructions.

Destination Index (DI) register:- It is used to hold the offset address of the Destination data in the extra segment while executing string instructions.

Instruction pointer (IP):- It acts as a program counter & points to the next instruction to be executed in the current code segment. It is automatically incremented after every instruction execution.

(iv) Flag register:-

It is a 16-bit register containing 16- individual flags.

out of these 16-flags, six are status flags
three are control flags &
remaining flags are kept for future use.

Format of flag register is shown below:-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Status flags:-

1. Carry flag (CF):-

It is set to 1 whenever a carry/borrow occurs during arithmetic operations such as addition/subtraction. otherwise it is set to '0'.

2. Parity flag (PF):-

It is set to 1 whenever there are even number of 1's in the result after an arithmetic/logical operation. otherwise it is set to '0'.

3. Auxilary carry flag (AF):- It is set to '1' whenever a carry/borrow is generated after addition/subtraction from lower nibble to higher nibble otherwise it is set to '0'.

4. Zero flag (ZF):- It is set to '1' whenever the result of an arithmetic operation is zero. otherwise it is set to '0'

5. Sign flag (SF):- It is set to '1' whenever the result of an arithmetic/logical operation is negative. otherwise it is set to '0'.

If MSB of the number is '1' → it is negative
0 → it is positive

6. Overflow flag (OF):-

It is set to '1' whenever the result of an arithmetic operation is out of range. otherwise it is set to '0'.

The range of 8-bit signed numbers (-128 to +127)
The range of 16-bit signed numbers (-32768 to +32767)

Control flags:-

Trap flag (TF):-

This flag is used for on-chip debugging
when T=1, it will work in a single step mode which means After each instruction, one internal interrupt is generated allowing a program to be executed instruction inspected as it by instruction.
If T=0; no function is performed

Interrupt flag (IF):-

when I=1, the µp will recognise interrupt requests from the peripherals.
I=0; the µp will not recognise any interrupt requests & will ignore them.

Direction flag:-

This flag is specifically used in string operations.
when D=0; then access the string data from lower memory locations towards higher memory locations
when D=1, then access the string data from higher memory locations towards lower memory locations.

Addressing modes of 8086:-

Instructions of the µp operate on the data which is called operands. These operands will be available in the different places. Addressing modes specifies the availability of operands for a given instruction. An instruction should know where is the operand & how to get the operand. The various methods used for this purpose is called addressing modes.

An Instruction can have one or two operands. They are called source operand & destination operands. Both these operands can be specified in same or in different methods.

The various addressing modes of 8086 are explained below.

1. Immediate addressing mode:-

In this mode, the operand is specified in the instruction itself.

Ex:-1. MOV AL, 55H.

Destination operand — Source operand.

Here source operand is 55H, which is to be moved to register AL. It is specified in the instruction itself. Hence it is called immediate addressing mode.

2. MOV AX, 0005H.

3. MOV AX, 0ABCDH.

2. Register addressing mode :-

In this mode the operand is specified in the register which is used by the instruction. All the registers can be used except IP.

Ex :- MOV AL, BL

Here both the operands are specified in the registers. The content of register BL will be copied to register AL.

2. MOV CX, DX.

3. ADD AL, BL.

3. Implicit addressing mode :-

In this addressing mode, the registers are used to specify the operands, but these registers are predefined. Hence there is no need to mention the name of the registers. in the instruction.

Ex :- MUL BX.; (16x16 multiplication).

Here one of the operand used for mul is always AX. it will be multiplied by the content of BX & the 32 bit answer will be always ~~always~~ stored in DX & AX. register. MSW (in DX) & LSW in (AX).

2. ADD BX

3. ADD CX.

4. Direct addressing mode:-

In this mode, a 16 bit memory address (offset) or an I/o address is directly specified in the instruction.

Eg:- MOV AX, [5000H].

IN .80H.

Here data resides in a memory location of in the DS, whose effective address may be computed using 5000H as the offset address & content of DS as segment address. The effective address, here is 10H * DS + 5000H.

In the second instruction, 80H is IO address.

5. Register Indirect addressing mode:-

In this mode, the offset address of the operand is specified in the register, which is used by the instruction. Register BX, SI, & DI can be used to specify the 16 bit offset address of the operand.

Eg:- MOV AL, [BX].

Here Register BX contains the 16-bit offset address of memory location having the operand. 8 bit operand (byte) will be copied to the register AL.

MOV AX, [SI]

MOV CX [DI].

6. Indexed addressing mode:- In this mode, offset of the operand is stored in one of the index registers (SI & DI)

Eg:- MOV AX, [SI]
MOV CX, [DI]

7. Register Relative Addressing mode :-

In this mode, the effective address of the operand is specified using a register & 8-bit / 16-bit constant called displacement. The registers BX, SI, DI & BP can be used.

Eg:- MOV AL, [BX] 50H.

Here the operand is available in a memory location whose 16-bit effective address is calculated by adding 50H to the contents of the register BX.

8. Base Indexed addressing mode :-

In this mode, the effective address of the operand is specified using two registers. one of them will be BX or BP. & the other will be index register SI or DI. base registers

The effective address is calculated by adding contents of the two registers.

Eg:- 1. MOV DX, [BX][SI] (or) MOV DX, [BX+SI]

2. MOV DX, [BP] [DI] (or) MOV DX, [BP+DI].

9. Relative base index addressing mode :-

In this mode, the effective address of the operand is specified using baseregister, index register. & 8-bit/16-bit displacement. The effective address is calculated by adding all three parameters.

Eg:- MOV AX, [BX] [SI] 50H.

(or)

MOV AX, [BX+SI+50H].

A program is executed in a sequential manner. But at times, the program control has to be transferred to other memory location in the same segment or other segments. Jmp & CALL instructions are used for this purpose.

1. Intrasegment Direct (relative) mode:-

In this mode, the instruction to which control has to be transfered lies in the same segment. The address of this instruction is specified as a label (NAME) in the current instruction. Its displacement relative to instruction pointer is specified as a part of instruction code.

Ex:- JMP NEXT.

The program control will be transfered to instruction whose name is NEXT.

JC NEXT:- The program control will be transfered to instruction whose name is NEXT if carry flag is set to high.

2. Intra Segment Indirect mode:-

In this mode the instruction to which the control has to be transfered lies in the same segment. The 16-bit offset address of the instruction is specified in a register. Any register except IP can be used.

Eg:- JMP CX.

The program control will be transfered to the memory location whose offset address is stored in CX.

3. Inter segment direct mode:-

In this mode, the instruction to which the control has to be transfered lies in other segment. The address of the instruction is specified as a label (name) in the current instruction

Eg:- JMP CONTINUE: CONTINUE lies in other segment

4. Intersegment indirect mode:-

In this mode, the instruction to which control has to transfered lies in the other segment. The current instruction uses a register which points to a four location memory area in the Data segment.

Eg:- JMP . [2000H]

Memory Segmentation:-

8086 uses the concept of Segmented memory. Here the entire memory is divided into number of logical segments. The capacity of each is 64KB.

Refer to this fig. 1MB physical memory has got 4 logical segments of 64KB each. They are Code Segment (CS), Data Segment (DS), Stack Segment (SS) & Extra Segment (ES). Whose starting & Endings address is given in the figure.

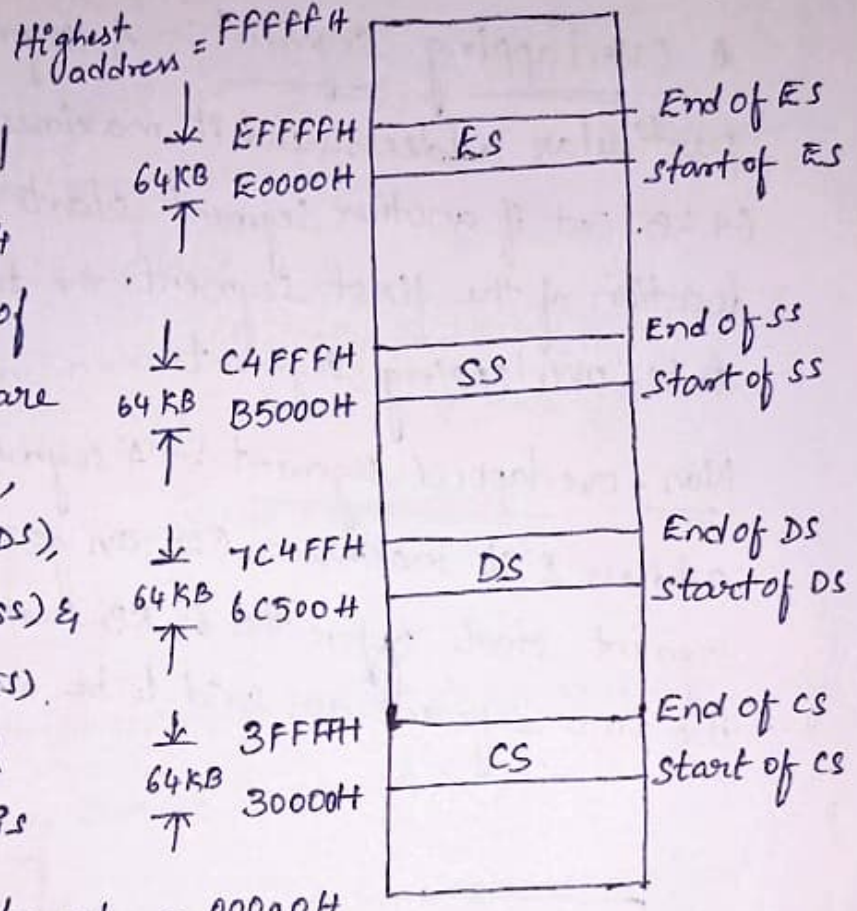


fig:- Physical Memory

To address a memory location within a specific segment, you need to know its position from the start of the segment. This is known as offset address (or) effective address. It is a 16-bit value & should be present in the register associated with that segment. For EX, register BX for data segment. The 16-upper bits of starting address of data segment should be available in DS register.

Segmentation is a process in which the main memory of a computer is divided into different segments & each segment has its own base address.

Types of segmentation :-

A Overlapping segment :- A segment starts at a particular address and its maximum size can go upto 64 KB. But if another segment starts along this 64 KB location of the first segment, the two segments are said to be overlapping segment.

Non-overlapped segment :- A segment starts at a particular address & its maximum size can go upto 64 KB. But if another segment starts before this 64 KB location of the first segment, the two segments are said to be Non-overlapping segments.

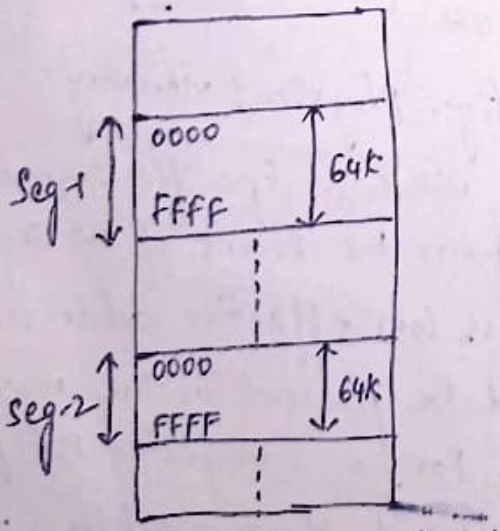


fig :- Non-overlapping segments

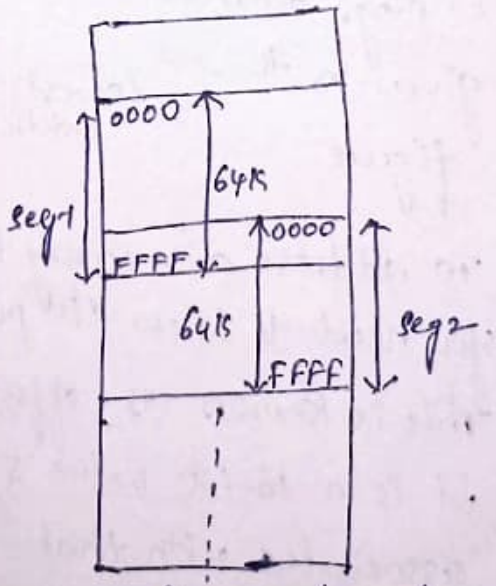


fig: overlapped segments.

Generation of 20-bit physical address :-

The 20-bit physical address can be calculated by

Memory address = (Segment Address) x 10H + offset value.

Advantages & Disadvantages of Memory Segmentation:-

Advantages :-

1. It permits the programmer to access 1 MB memory using only 16-bit address
2. It divides the memory logically to store code, data & stack position separately. for code & data protection.
3. It is useful for multiuser environment.

Disadvantages:-

Although the total memory is 16 * 64 KB at a time, only 4 * 64KB memory can be accessed.

Following table gives the default segment & offset registers.

Default Segment	offset.
CS	IP
SS	SP, BP
DS	BX, SI, DI
ES	DI

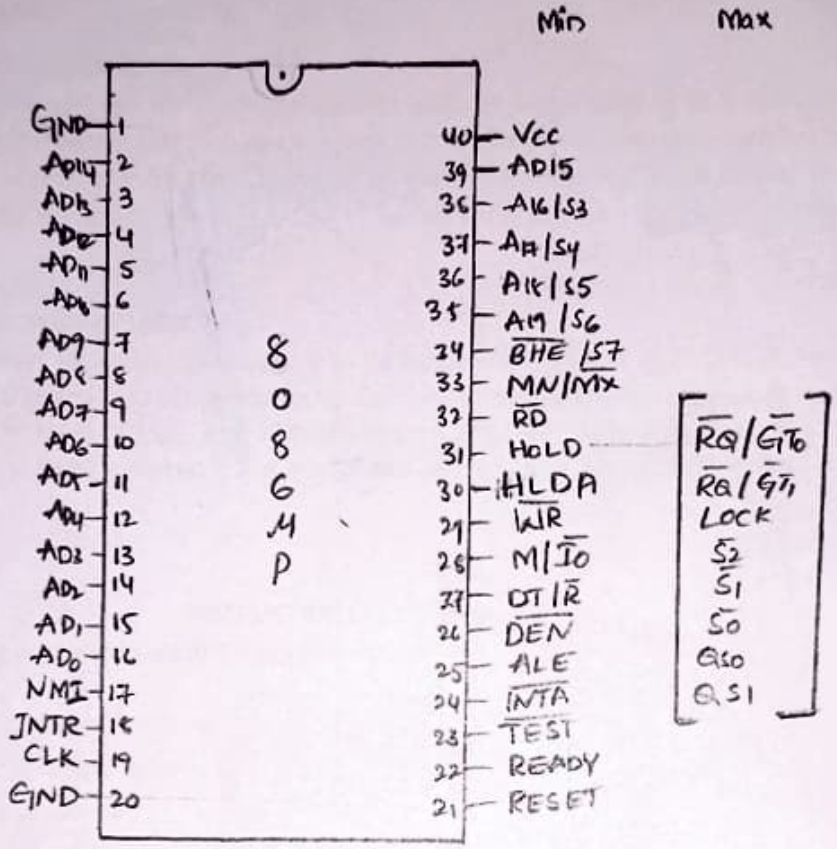


Fig:- Pin diagram of 8086 Micro Processor

INTERSEGMENT ADDRESSING MODE

If the destination location is in the different segment the mode is called intersegment addressing mode

There are two types

1. Intersegment direct mode
2. Intersegment indirect mode

INTERSEGMENT DIRECT MODE:

In this mode the address to which the control is to be transferred is in a different segment this addressing mode provides a means of branching from one code segment to another code segment. Here the CS and IP of the destination address are specified directly in the instruction.

Example

JMP 2000H: 3000H;

INTERSEGMENT INDIRECT MODE :

In this the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly .Content of memory block containing four bytes IP(LSB) ,IP(MSB),CS(LSB) and CS(MSB) sequentially The starting address of the memory block may be referred using any of the addressing mode except immediate mode.

Example

JMP [5000H];

INSTRUCTION SET OF 8086

The 8086 microprocessor supports 8 types of instructions –

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

1. 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.

2. 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.

3. LOGICAL 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.

4. 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.

5. 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

6. 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.

7. 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

8. 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

ASSEMBLER DIRECTIVES

Assembler directives are the Instructions to the Assembler, linker and loader regarding the program being executed. also called 'pseudo instructions. Control the generation of machine codes and organization of the program; but no machine codes are generated for assembler directives.

They are used to

› specify the start and end of a program
› attach value to variables
› allocate storage locations to input/ output data
› define start and end of segments, procedures, macros etc..

ASSUME

Used to tell the assembler the name of the logical segment it should use for a specified segment. You must tell the assembler that what to assume for any segment you use in the program.

Example

ASSUME: CODE

Tells the assembler that the instructions for the program are in segment named CODE.

DB – Defined Byte

Used to declare a byte type variable or to set aside one or more locations of type byte in memory.

Example

PRICES DB 49H, 98H, 29H:

Declare array of 3 bytes named PRICES and initialize 3 bytes as shown.

DD – Define Double Word

Used to declare a variable of type doubleword or to reserve a memory location which can be accessed as doubleword.

DQ – Define Quadword

Used to tell the assembler to declare the variable as 4 words of storage in memory.

DT – Define Ten Bytes

Used to tell the assembler to declare the variable which is 10 bytes in length or reserve 10 bytes of storage in memory.

DW – Define Word

Used to tell the assembler to define a variable type as word or reserve word in memory.

DUP: used to initialize several locations and to assign values to location

END – End the Program

To tell the assembler to stop fetching the instruction and end the program execution.

ENDP – it is used to end the procedure.

ENDS – used to end the segment.

EQU – EQUATE

Used to give name to some value or symbol.

EVEN – Align On Even Memory Address

Tells the assembler to increment the location counter to the next even address if it is not already at an even address.

EXTRN

Used to tell the assembler that the name or labels following the directive are in some other assembly module.

GLOBAL – Declares Symbols As Public Or Extrn

Used to make the symbol available to other modules.It can be used in place of EXTRN or PUBLIC keyword.

GROUP – Group related segment

Used to tell the assembler to group the logical segments named after the directive into one logical segment. This allows the content of all the segments to be accessed from the same group.

INCLUDE – include source code from file

Used to tell the assembler to insert a block of source code from the named file into the current source module. This shortens the source code.

LABEL

Used to give the name to the current value in the location counter. The LABEL directive must be followed by a term which specifies the type you want associated with that name.

LENGTH

Used to determine the number of items in some data such as string or array.

NAME

Used to give a specific name to a module when the programs consisting of several modules.

OFFSET

It is an operator which tells the assembler to determine the offset or displacement of named data item or procedure from the start of the segment which contains it.

ORG – Originate

Tells the assembler to set the location counter value.

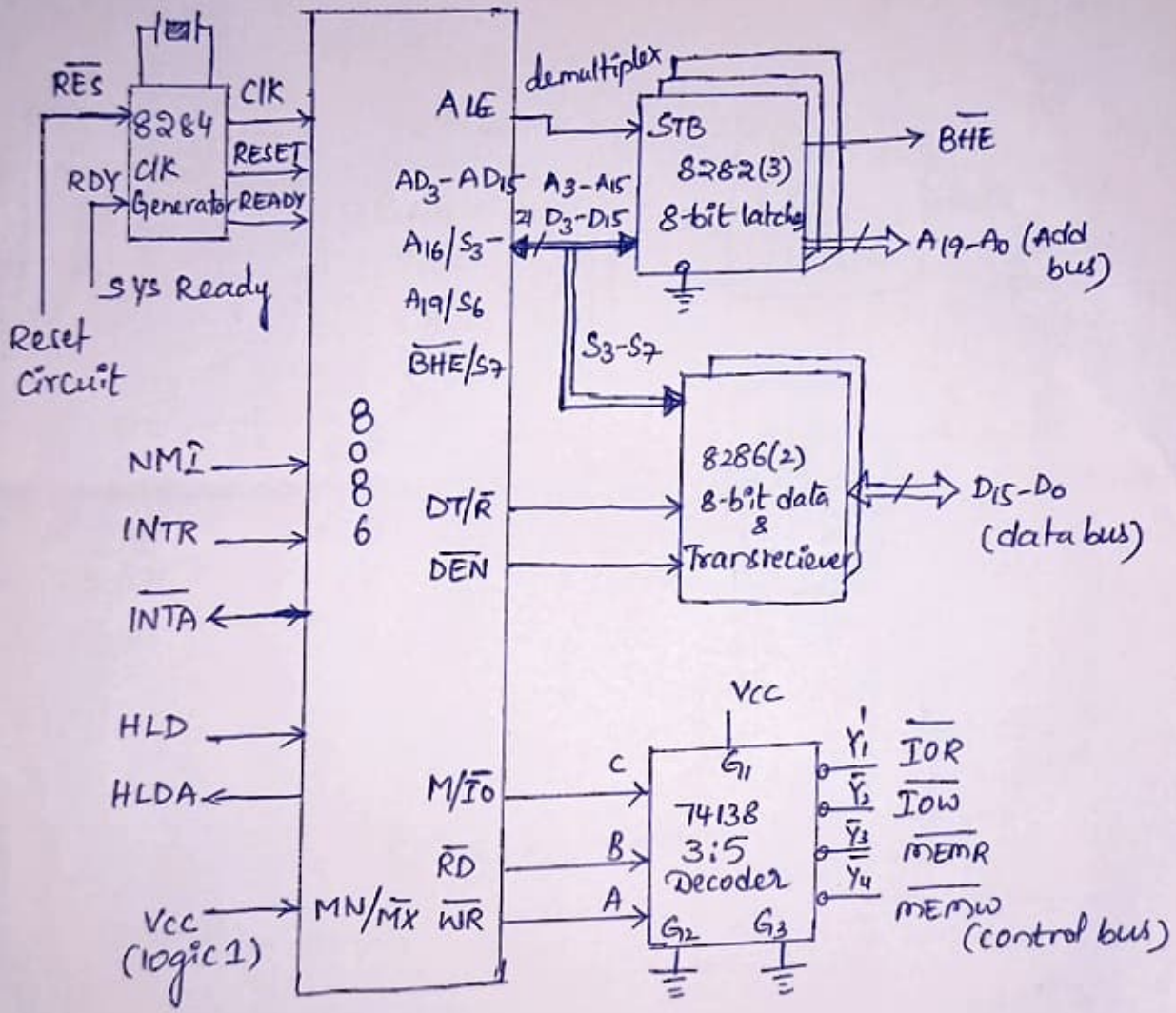
Example, ORG 7000H sets the location counter value to point to 7000H location in memory.

$ is often used to symbolically represent the value of the location counter. It is used with ORG to tell the assembler to change the location according to the current value in the location counter. E.g. ORG $+100.

Minimum Mode of 8086 microprocessor :-

The minimum mode configuration of 8086 µp is shown in fig. The 8086 µp operates in minimum mode when $MN/\overline{MX} = 1$.

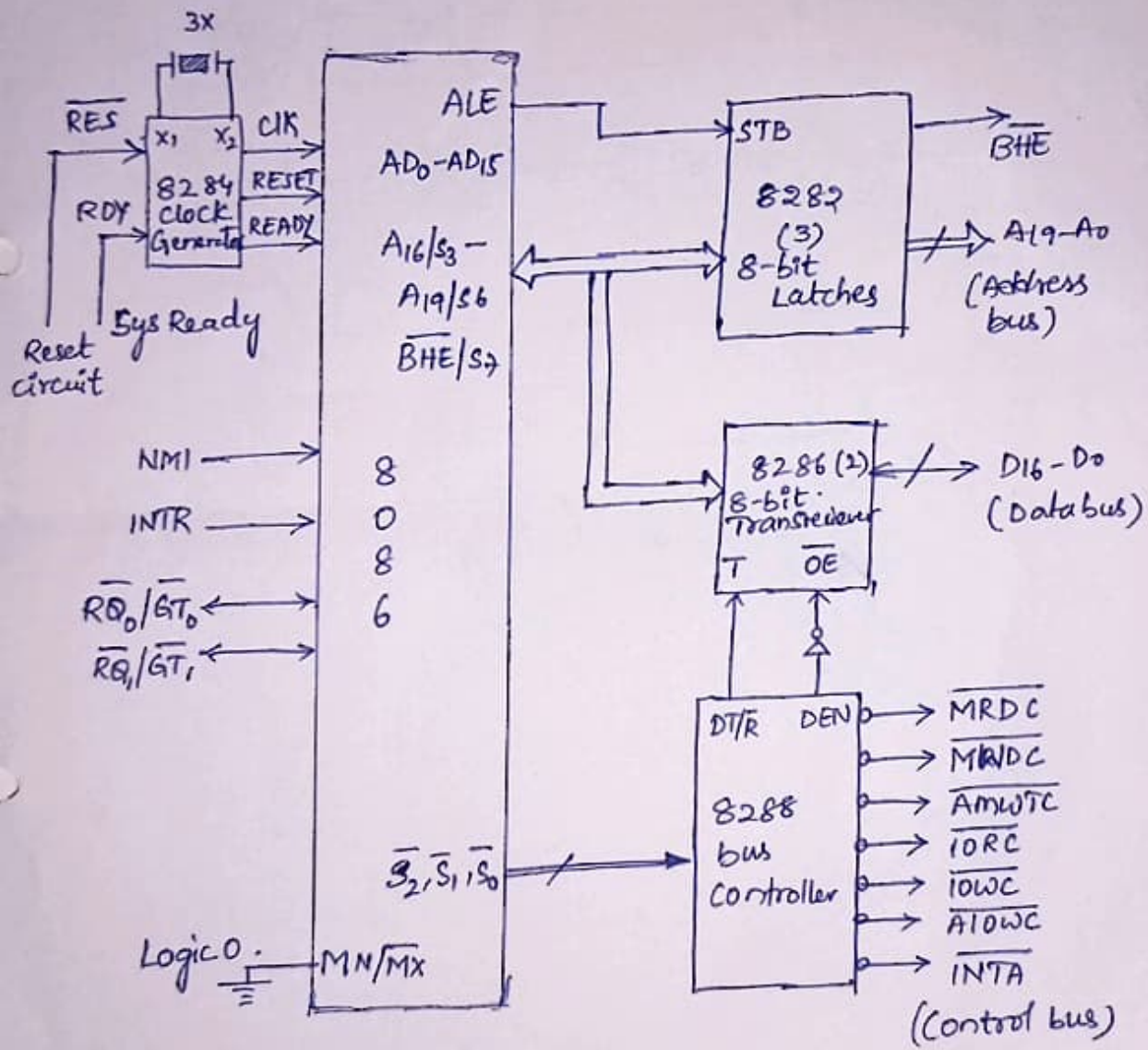
→ In minimum mode, 8086 is the only processor in the s/m

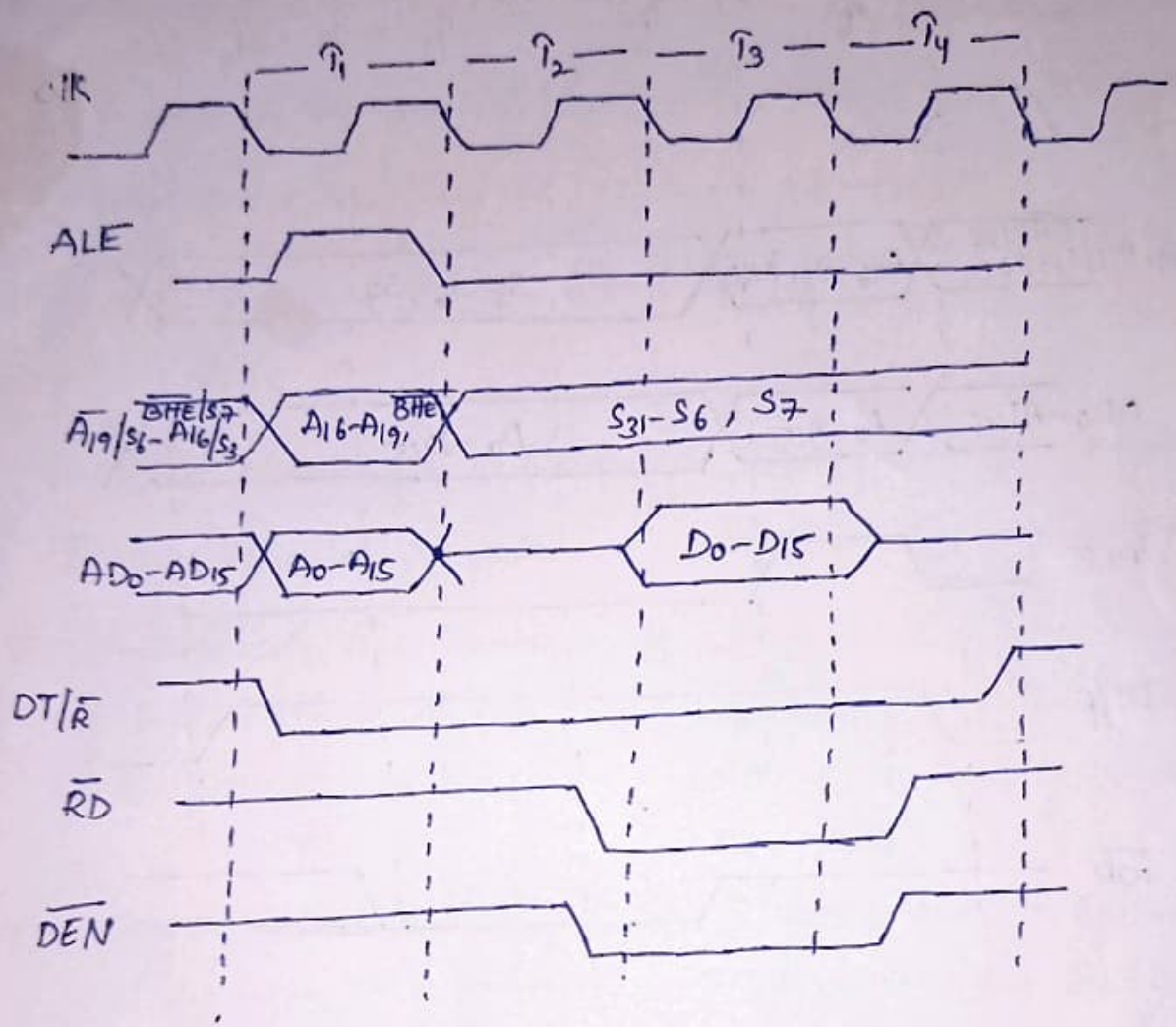


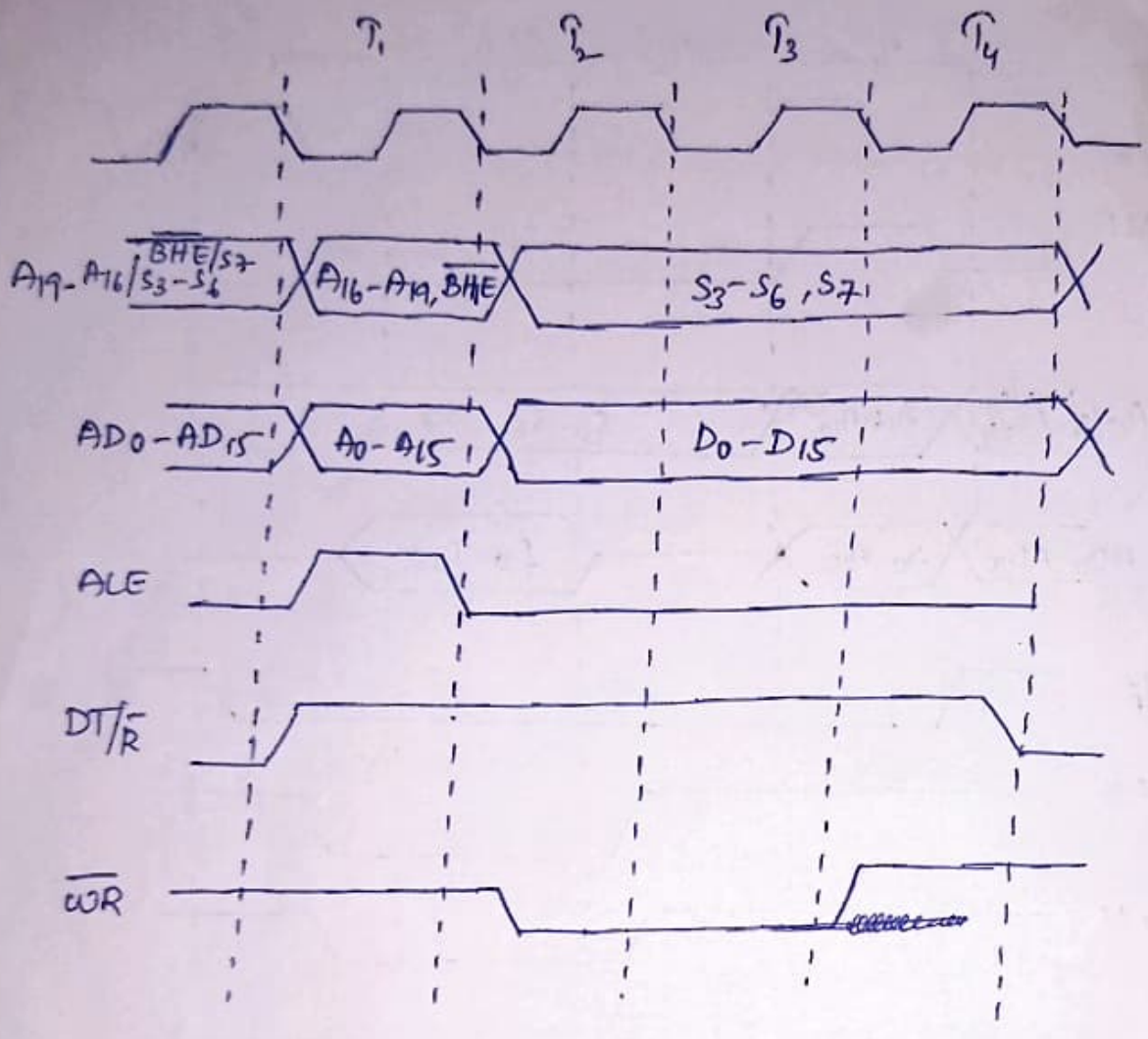
which provides all the control signals which are needed for memory operation & I/o interfacing.

→ Here the circuit is simple but it does not support multiprocessing.

Maximum Mode of 8086 Microprocessor :-




T1
T2
T3
T4
CLK
ALE
A19/S6-A16/S3
BHE/S7
A16-A19
BHE
S31-S6, S7
AD0-AD15
A0-A15
D0-D15
DT/R
RD
DEN


T1
T2
T3
T4
A19-A16/BHE/S7 S3-S6
A16-A19, BHE
S3-S6, S7
AD0-AD15
A0-A15
D0-D15
ALE
DT/R
WR

RISC Vs ARM processors :- (or) RISC Vs CISC :-

Based on the architecture & hardware of the processors, they are classified as (1) RISC processors
(2) CISC processors.

RISC	CISC
1. RISC stands for Reduced Instruction Set computer	1. CISC stands for Complex Instructions set computer
2. It supports less number of instructions with a fixed format (32 bits)	2. It consists of many number of instructions with a variable formats (16-64 bits).
3. They can execute their instructions very faster because instructions are very small & simple.	3. It takes time to execute instructions as instructions are complex.
4. RISC calculations are faster & more precise	4. CISC calculations are slow & precise.
5. Instructions takes a single clock cycle to execute	5. Instructions takes 2-15 clock cycles to execute.
6. More focus on software	6. More focus on hardware
7. It supports pipelining.	7. It may or may not supports pipelining.

8. Instruction decode is simple.	8. Instruction decode is complex.
9. It does not require any external memory for execution of instructions.	9. It needs memory for execution of instructions.
10. Risc is used in high end applications like video processing, telecommunications & image processing, mobile phones & tablets	10. cisc is used in low end applications such as security systems, home automations, desktops & laptops.

ARM processor

ARM came in to existence in the year 1983 and was developed by Acorn Computers.

The Architecture of ARM processor is created by Advanced RISC Machines.

It is the family of Central processing units (CPU)

It is based on RISC architecture.

This needs very few instruction sets and transistors.

It has very small size.

This is reason that it is perfect fit for small size devices

It has less power consumption along with reduced complexity in its circuits.

They can applied to various designs such as

- 32-bit devices
- Embedded systems.

Applications

It is used in
- music players
- smart phones
- wearables
- tablets
- digital television
- set-top boxes
- Hard drivers
- Inkjet printers
- GPS navigation systems and other consumer electronic devices.

ARM architecture is implemented on windows
Unix and

other operating systems including
- Apple iOS
- Android
- Solaris
- WebOS
- GNU / Linux

ARM Versions:

	product-family	features	Cache Memory	million Instructions per second MIPS
1983	ARM1	first implementation (only Arithmetic operations)	1KB	-
	ARM 2	MUL & swap instructions, Graphics MEMC and I/O processor		4-7 MIPS
	ARM 3	first integrated memory cache	4KB	12 MIPS
	ARM 6	support 32-bit Memory Add, Long Inst, Buses	4KB	17-28 MIPS
	ARM 7T	3 stage pipeline, Thumb, 26-bit Addressing	8KB	15-18 MIPS
	ARM 8	5 stage pipeline, static branch prediction, double bandwidth memory	8KB	84 MIPS
	ARM 9T/E	5 stage pipeline, Thumb, enhanced instructions, clockless processor	DSP 16KB	220 MIPS
	ARM 10	6 stage pipeline, Thumb, enhanced instructions	DSP 16-32 KB	550 MIPS
	ARM 11	8 & 9-stage pipeline, Thumb, enchanced DSP instructions.	32 KB	745-965 MIPS

Latest ARM versions:

product family	cache Memory	MIPS
Cortex - M	64KB	3.13 DMIPS
cortex - R	128KB	3.41 DMIPS
cortex - A (32-bit)	1MB	3.5 DMIPS
cortex - A (64-bit)	4 MB	13 - 16 DMIPS
cortex - X	8 MB	20 MIPS

features:

1. Multiprocessing systems
2. Tightly coupled memory
3. Thumb - 2 Technology
4. One cycle execution time
5. Pipelining
6. Large number of registers
7. Load / store Model

Advantages:

1. Affordable to create
2. Low power consumption
3. Work faster
4. Multiprocessing feature
5. Better Battery life
6. Load store Architecture
7. Simple circuits.

Disadvantages:

1. The speeds are limited in some processors which might create problems.
2. Scheduling instructions is difficult in case of ARM processor
3. It needs very highly skilled programmers because when processor not executed instructions properly then its get slower

UNIT – II

Assembly Language programming & I/O Interface

- Assembler directives
- Macros
- Simple programs involving logical – branch instructions
- Sorting
- evaluating arithmetic expressions
- String manipulations
- 8255 PPI
- Various modes of operation
- A/D – D/A converter interfacing
- Memory interfacing to 8086
- Interrupt structure of 8086
- Vector interrupt table
- Interrupt service routine
- Interfacing interrupt controller 8259
- Need of DMA
- Serial communication standards
- serial data transfer schemes

UNIT - II

Assembly language Programming & I/o Interface

- Assembler directives
- Macros
- Simple programs involving logical, branch instructions sorting, arithmetic Expressions, string manipulation.
- 8255 PPI - various modes of operation.
- A/D, - D/A Converter interfacing.
- Memory interfacing to 8086
- Interrupt structure of 8086
- vector Interrupt table
- Interrupt servic routine
- Interfacing interrupt controller 8259
- Need of DMA
- Serial Communication standards
- Serial data transfer schemes.

Macro :-

A Macro is a set of instructions grouped under a single unit. It is another method for implementing modular programming in 8086.

* The macro is different from the procedure in a way that unlike calling and returning the control as in procedures, the processor generates the code in the program every time whenever.
* A Macro can be defined in a program using assembler directives MACRO.

Format :-

```
INIT MACRO      → Define macro
  ≡≡≡ } set of instructions
ENDM  - end of macro
```

Advantages :-

→ CALL and RET instructions are not used in macro
→ No. overhead time
→ It executes faster than procedure
→ It supports modular programming

Disadvantages :-

→ It requires more memory
→ Machine code can be generated each time when the macro is called.

Procedure	Macro
* Procedure is a group of instructions to perform a specific task which can be called repetitively	* Macro is a set of instructions that is written with in MACRO assembler directives.
* Machine code is generated only once.	* Machine code can be generated each time when the macro is called.
* CALL and RET instructions are used to call the procedure	* CALL and RET instructions are not used.
* It requires less memory	* It requires more memory
* It executes slower than Macro	* It executes faster than procedure.

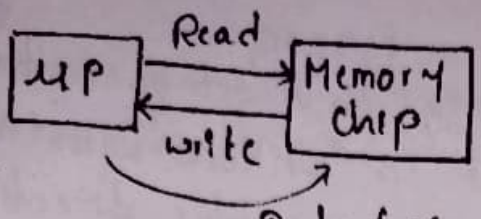
Memory Interfacing to 8086 :-

→ Memory is an integral part of µp.

↓
2 types

1. Read only memory (ROM) → Read signal program that donot need changes
   PROM, EPROM, EEPROM.
2. Random access memory (RAM)

↓
- temparary data
- Read & write
- stores the user program.
- It is a volatile memory



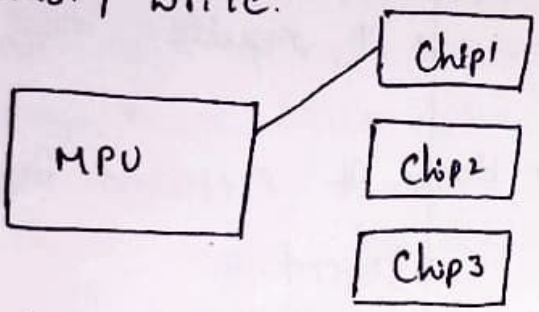
Interface

→ microprocessor needs to access memory to get instruction codes & data.

→ Microprocessor unit will initiate Read and write signal according to the operation.

→ Memory Read

→ Memory write.



CS - Chip select
RD - Read signal
WR - write signal

→ To do the above task it is necessary to have a device or a circuit, which performs this task interfacing device.

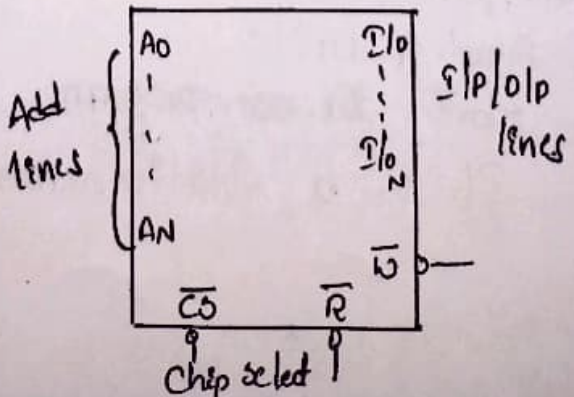
Interfacing device :-

There are three step for interfacing of µp & memory

1. To select the required Chip
2. Identify the required register
3. Must Enable the appropriate buffers

Memory device

1. Address lines
2. I/o lines
3. Selection lines
4. Control lines



Address inputs seled memory location within the memory device $A_0 \dots A_N$

* The No of address lines indicates the total memory capacity of the memory device

1K memory = 1024 bytes
= 10 address lines
1 MB memory = 20 address lines

* Memory device may have separate I/O lines (or) common bidirectional I/O lines.
* Using these lines data can be transferred in any directions.

$D_0 \dots D_N$ - Data lines

* Size of a memory location is dependent upon the number of data bits.
* No of data lines : 8 $D_0 \dots D_7$ then 8-bits (or) 1 byte data can be stored in each location.
* No of address lines indicates the size of memory device
* No of data lines - size of memory location.

Memory Chip - 2K x 8
↑ memory size 2K = 2048
↑ 8bits can be stored

* Memory device may contain one or more inputs which are used to select the memory device.

$\overline{CS}$ - Chip select signal. It's logic 0.

* More than one $\overline{CS}$ → all pins are to be activated.

* One (or) more control inputs.

ROM - $\overline{OE}$ - output enable pin allow data to flow out of the o/p data pins.

RAM { Read - $\overline{RD}$ / Write - $\overline{WR}$ } active low signal

Memory interfacing steps.

* Memories are organized as two dimentional arry.

Ex: 4K x 8 $4 \times 1024 = 4096$ locations.

* N memory locations
n address lines.

$$n = \log_2 N$$

$n = \log_2 4096$

$2^n = N$

$2^n = 4096$

$n = 12$ — address lines are needed.

* 8086 - 20 address lines. A_0 --- A_{19}
A_0-A_{11} are used for to represent the 4096 add. lines. A_{12} - A_{19} - to form the Chip select

Step 1 :- Arrange the available chips so as to obtain 16 bit data bus width. The upper 8-bit bank is called odd address memory band" and lower 8-bit bank is called even add. memory bank.

PPI - Programmable Peripheral Input-output port :-

* 8255 (Programmable peripheral Interface) is designed to increase I/o Interfacing capability of microprocessor.
* It has 24 I/o pins that can be grouped in two 8 bits parallel ports : A & B, with remaining 8-bits into port C.
* port c can be used as individual bits (or) can be used as 4-bit ports ,Cupper & Clower.
* 8255 functions in following modes.
  - I/o mode [mode 0, 1 & 2]
  - BSR mode.

Control signals of 8255

$\overline{RD}$ - Active low signal - MPU reads data from selected port of 8255.

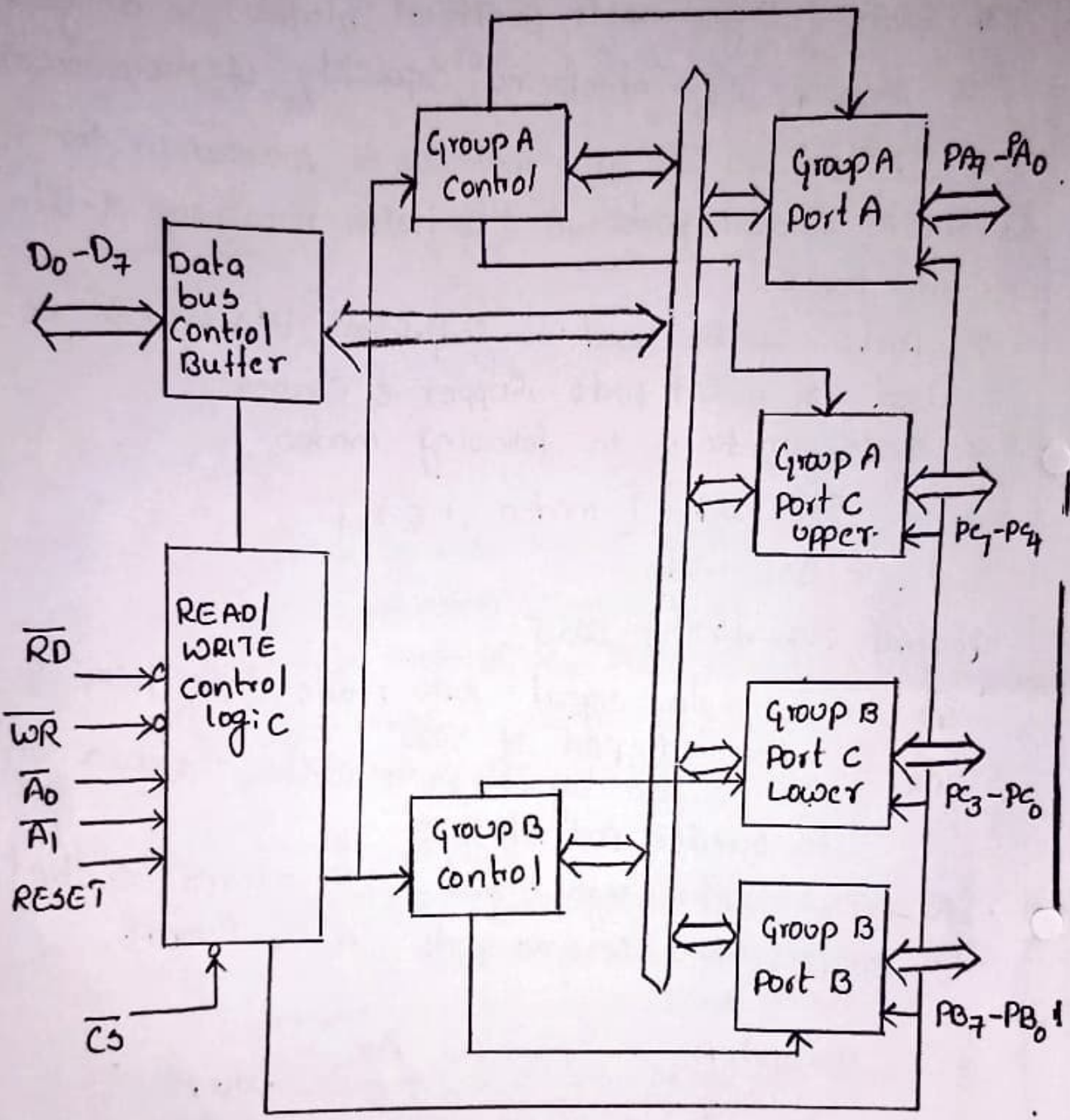
$\overline{WR}$ - Active low signal - MPU writes the data on to selected port of 8255

RESET - Active High signal - It clears control registers and sets all ports in input mode.

$\overline{CS}$, A0 and A1 :-

$\overline{CS}$	A1	A0	
0	0	0	— port A
0	0	1	— Port B
0	1	0	— port C
0	1	1	— Control Register
1	x	x	8255 is not selected.

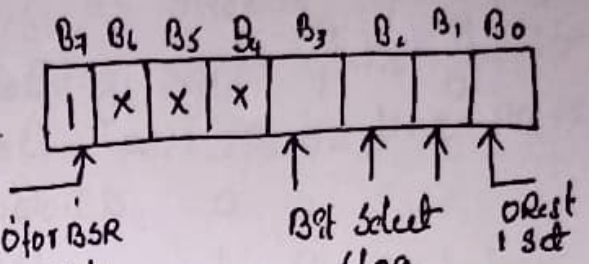
8255 Internal Architecture.



→ The group A contains an 8-bit port A along with a 4-bit port Cupper. The port A lines are identified by symbols $PA_0 - PA_7$ while port C lines are identified as $PC_4 - PC_7$.

The group B contains an 8bit port B along with 4-bit port Clower. ie $PB_0 - PB_7$ & $PC_0 - PC_3$

B_3	B_2	B_1	Slected bits of port c
0	0	0	B0
0	0	1	B1
0	1	0	B2
0	1	1	B3
1	0	0	B4
1	0	1	B5
1	1	0	B6
1	1	1	B7.



I/O MODES :-

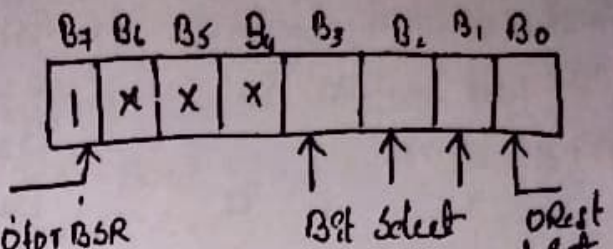
MODE 0 :- This is also known as basic I/P-O/P mode. This mode provides simple input and output capability using each of the three ports. Data can be simply read from & written to the input & output respectively, after appropriate initialisation.

The features of this mode

1) Two 8-bit ports (A & B) and two 4-bit ports (port c upper & lower) are available. This two 4-bit ports can be Combindly used as a third 8-bit port.
2) Any port can be used as input (or) output.
3) Output ports are latched. I/P ports are nd latched
4) Maximum of 4 ports are available so that total 16 I/O configurations are possible.

All these modes can be selected by control word register which is programming register internal to 8255.

B_3	B_2	B_1	Slected bits of port c
0	0	0	B_0
0	0	1	B_1
0	1	0	B_2
0	1	1	B_3
1	0	0	B_4
1	0	1	B_5
1	1	0	B_6
1	1	1	B_7



I/O MODES :-

MODE 0 :- This is also known as basic i/p - o/p mode. This mode provides simple input and output capability using each of the three ports. Data can be simply read from & written to the input & output respectively, after appropriate initialisation.

The features of this mode

1) Two 8-bit ports (A & B) and two 4-bit ports (Port c upper & lower) are available. This two 4-bit ports can be Combindly used as a third 8-bit port.
2) Any port can be used as input (or) output.
3) Output ports are latched. I/p ports are not latched
4) Maximum of 4 ports are available so that total 16 I/o configurations are possible.

All these modes can be selected by control word register which is programming register internal to 8255.

Mode 1 :-

This mode is also called as strobed input/output mode. In this mode handshaking signal control the i/p (or) output action of the specified port. Port C lines PC_0-PC_2 provides handshake lines for port B.

→ Port A (or) port B can be set for input (or) output operation.
→ The bits from port C are used as control signals (used for handshaking)
→ Port A & port B function as 8-bit I/o ports.
→ Each port uses 3 lines from portC as handshake signal and the remaining signals can be used for I/o functions.
→ I/O data are latched.
→ Interrupt logic is supported.

a) Input Configuration.

PortA } function as
PortB } 8-bit i/p ports.

PortA - PortC, PC_3 to PC_5 as handshake signal.

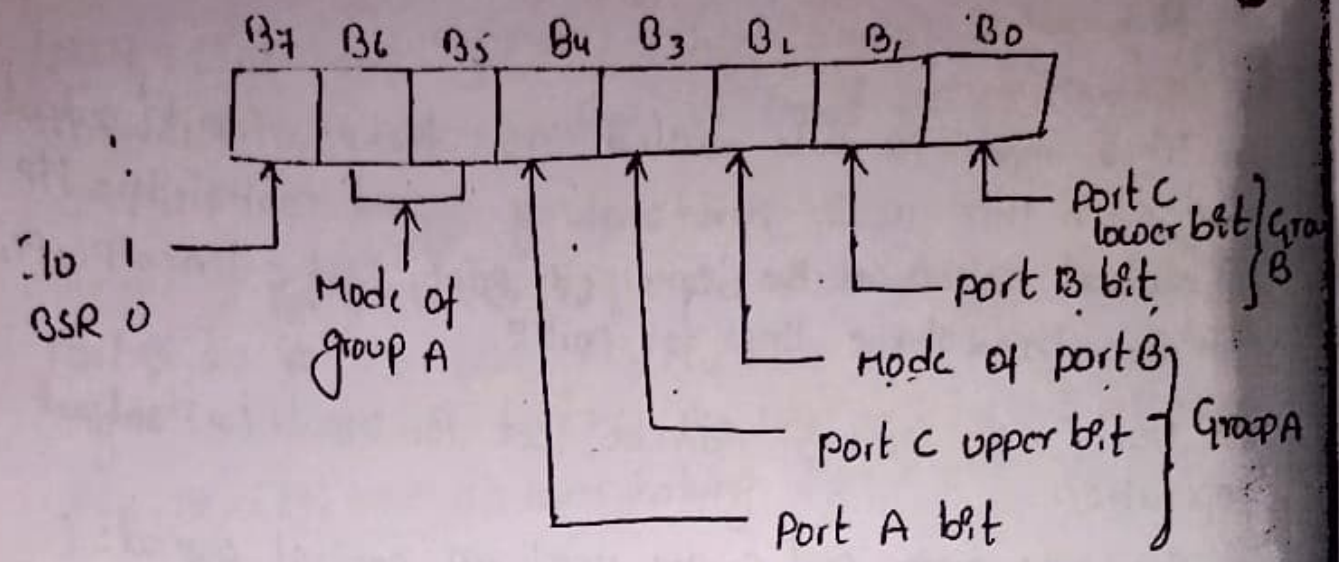
PortB - PortC PC_0 to PC_2 as handshak signal.
- PC_6, PC_7 may be used as Input/output, otherwise not used.

b) O/p Configuration.

PortA } output port 8-bit
PortB }
PortC - handshaking.

PortA - PortC, PC_3, PC_6 & PC_7 handshake signal

PortB - PortC PC_0, PC_1 & PC_2 as handshake signal
- PC_4, PC_5 may be used as input (or) output, otherwise not used.



B7 - 1 I/O mode
- 0 BSR mode.

B6 B5
0 0 - Mode 0
0 1 - Mode 1
1 0 - Mode 2
1 1 - X

B4 - port A
- 0 - O/P
1 - I/P.

B3 - port C upper
- 0 - O/P
1 - I/P

B2 - Mode selection bit.
0 - Mode 0
1 - Mode 1

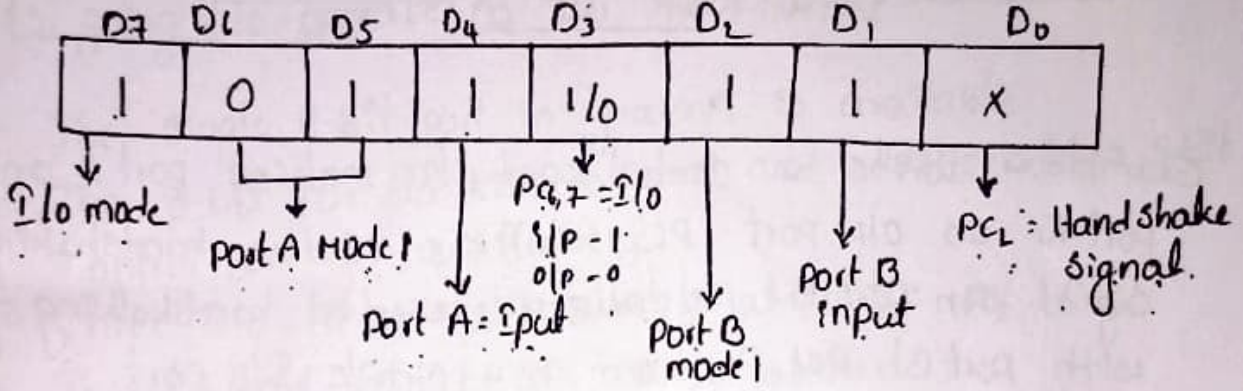
B1 - port B
- 0 - O/P.
1 - I/P.

B0 - port C lower.
1 - I/P.
0 - O/P.

INTR : (Interrupt Request) This signal is used to interrupt the CPU whenever an input device requests the service.

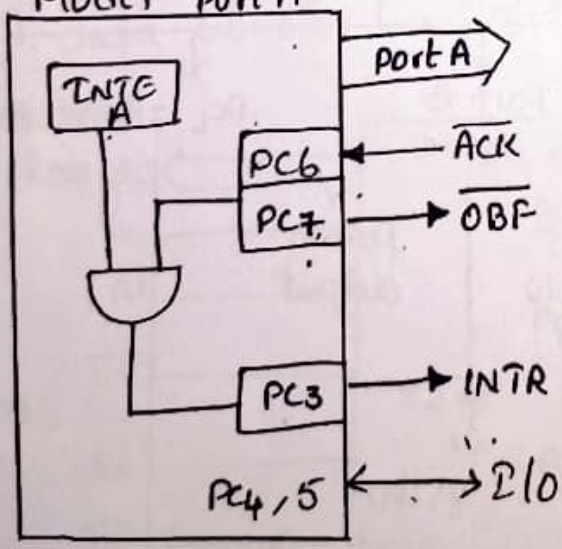
Example : Write a control word for making Port A & Port B as a I/P port, PC_3 PC_4 & PC_5 used as handshaking signal with port A. Port PC_6, PC_7 used as simple I/O port. PC_0, PC_1 & PC_2 used as hand shaking signal with port B.

- solution :-

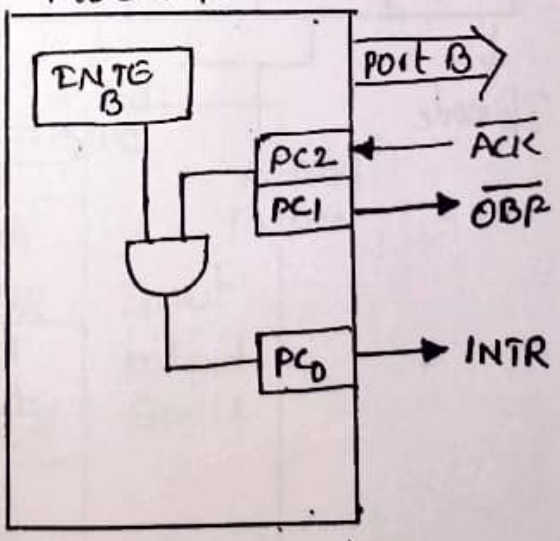


Output Configuration :-

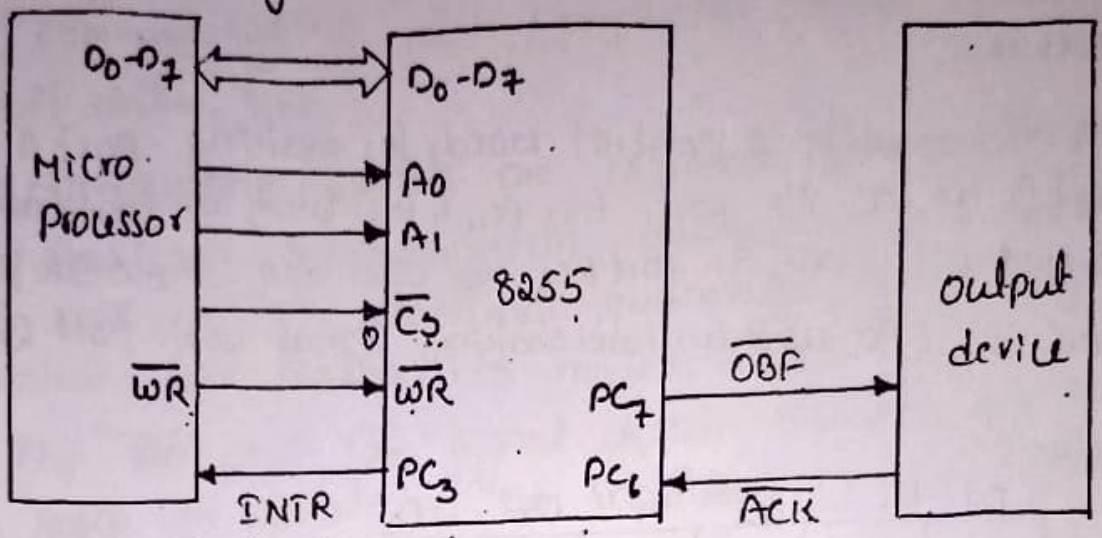
- Mode 1 Port A



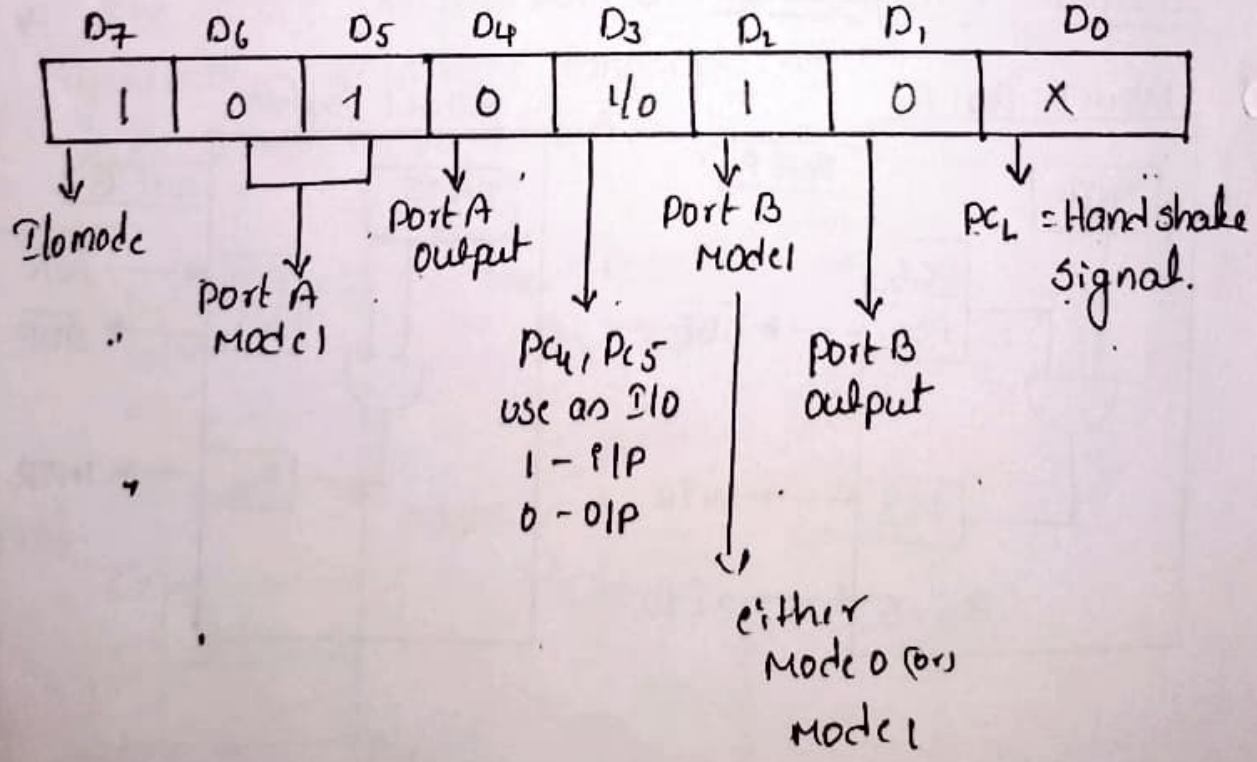
Mode 1 Port B



O/P Configuration.



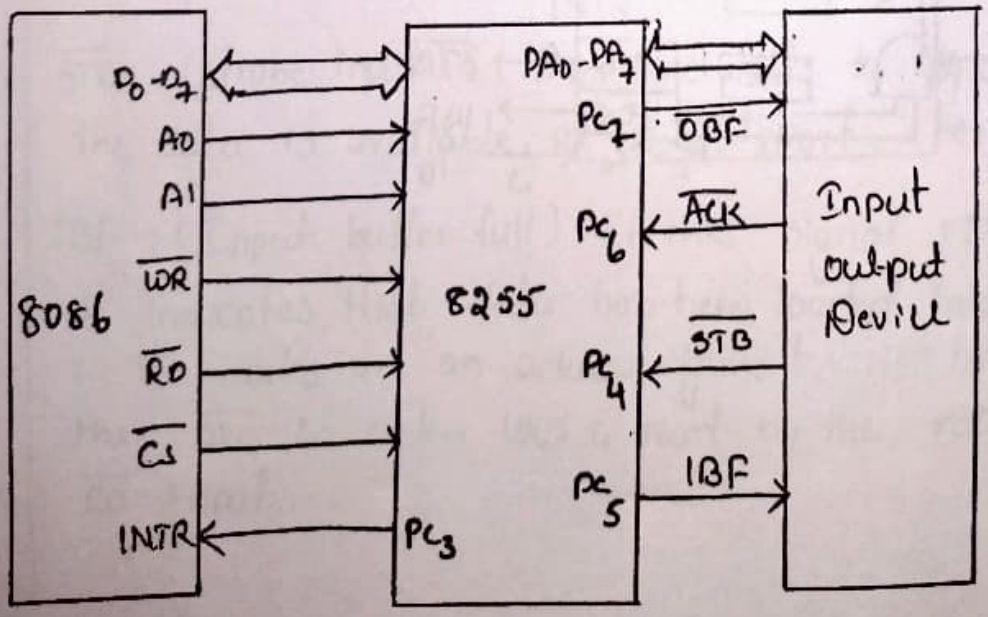
Example:- write a control word for making port A and port B as o/p port PC_7, PC_6 & PC_3 used as handshaking signal with port A. PC_0, PC_1 & PC_2 used as handshaking signal with port B. PC_4, PC_5 used as a simple I/O port.



Mode-2 (Strobed bidirectional I/O)

In this mode of 8255 provides additional features for communicating with peripheral device on an 8-bit data bus.

* Handshaking signals are provided to maintain proper data flow and synchronization between the data transmitter and receiver. The interrupt generation and other functions are similar to mode 1.
* The $\overline{RD}$ and $\overline{WR}$ signal decide whether the 8255 is going to operate as an input port (or) output port.
* The single 8-bit port in group A is available
* The 8-bit port is bidirectional and additionally a 5-bit control port is available.
* Three I/O lines are available at port C $PC_2 - PC_0$
* Inputs and outputs are both latched.
* The 5-bit control port C $PC_3 - PC_7$ is used for generating / accepting handshake signal for the 8-bit data transfer on port A.



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	1	1	0	0	0	0

D_7 - I/O mode

D_6 D_5 - mode selection
00 mode 0
01 mode 1
1x mode 2

D_4 - 1 - I/P, 0 - O/P

D_3 - 1 - I/P, 0 - O/P

D_2 - 0 - mode 0, 1 - mode 1

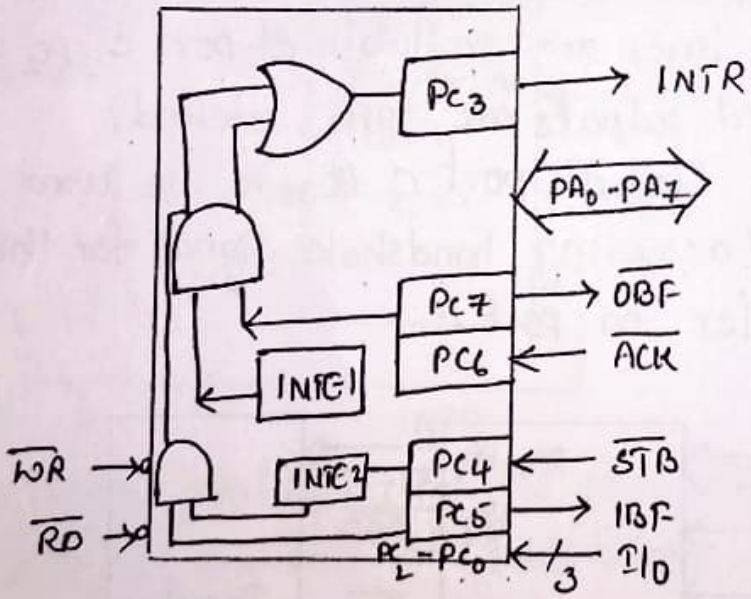
D_1 - 1 - I/P, 0 - O/P

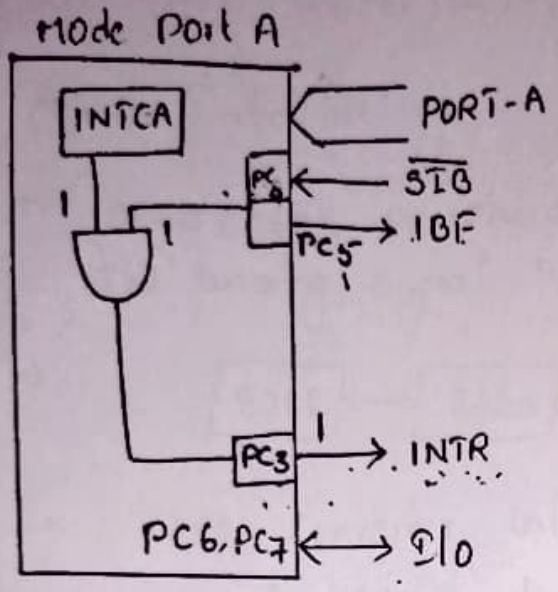
D_0 - 1 - I/P, 0 - O/P

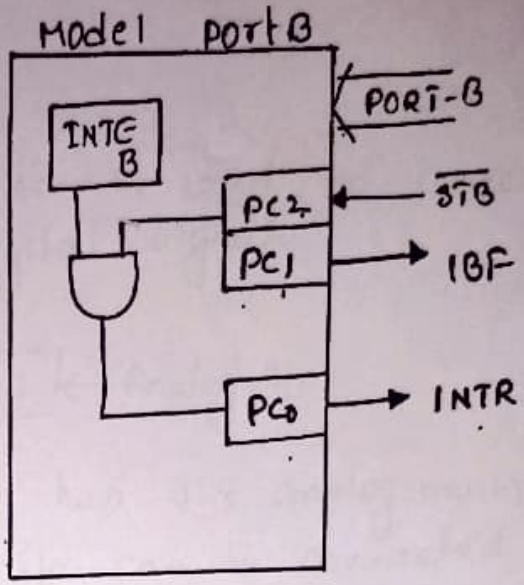
Port A - as I/P port, mode 2

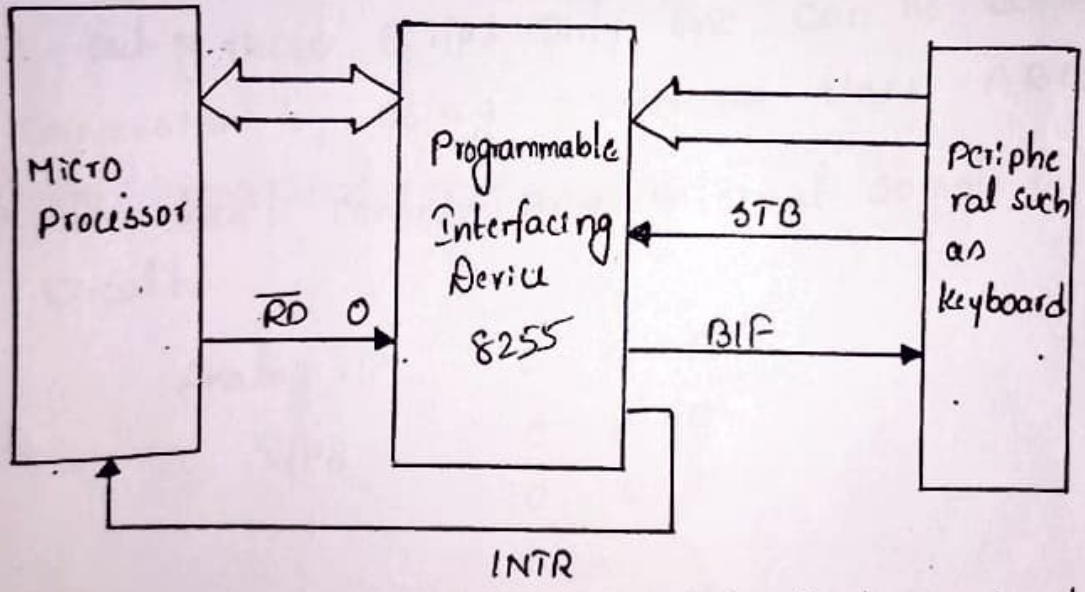
Port B - as O/P port mode 0

Control word = F0









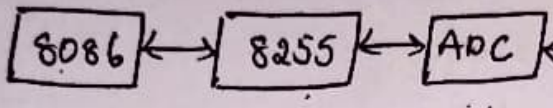
$\overline{STB}$: (Strobe Input) If this line falls to logic low level, the data is available at 8-bit I/Pport

IBF : (Input buffer full) If this signal rises to logic 1 it indicates that data has been loaded into the latches. i.e it works as an acknowledgement. IBF is set when the $\overline{STB}$ is active low & reset by the raising edge of $\overline{RD}$ Input.

A/D - D/A Interfacing

ADC Interfacing.

→ 0808/0809 peripheral device is used to convert the analog signal into digital signal.

→ 

→ This converter internally has 8:8 analog multiplexer i.e it has 8 different I/P can be connected to the chip.

→ out of these 8 I/Ps only one can be selected for Conversion by using 3 address lines A B & C

→ It doesn't contain any internal sample & hold circuits.

Analog I/P	C	B	A
I/P_0	0	0	0
1	0	0	1
⋮	⋮	⋮	⋮
7	1	1	1

Ex: Interface ADC 0808 with 8086 using 8255 port use port A of 8255 for transfering digital data o/p of ADC to the CPU & port C of Control signal. Assume that an analog I/P is present at I/P_2 of the ADC and a clock I/P of suitable frequency is available for ADC.

→ The analog i/p I/P_2 is used & therefore add pins ABC should be 010 to select I/P_2

→ The OE (o/p latch enable) & ALE pins are kept at +5V to select ADC & enable the o/p.

→ Port C upper upper acts as the i/p port to receive the EOC signal.

→ Port C lower acts as the o/p port to send SOC to ADC

→ Port A acts an 8-bit i/p data port to receive digital data o/p from ADC

8255 control word.

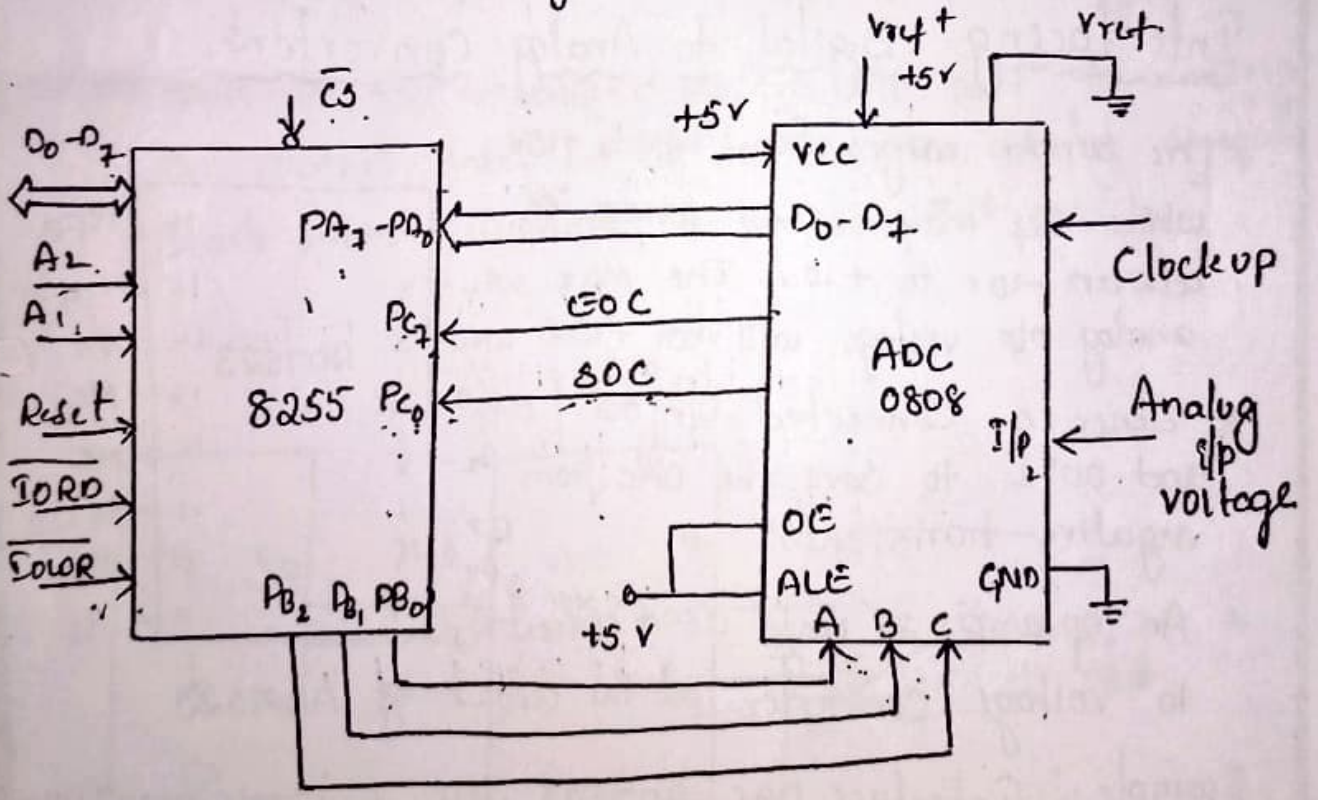
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	0	0	1	1	0	0	0	= 98H

Program

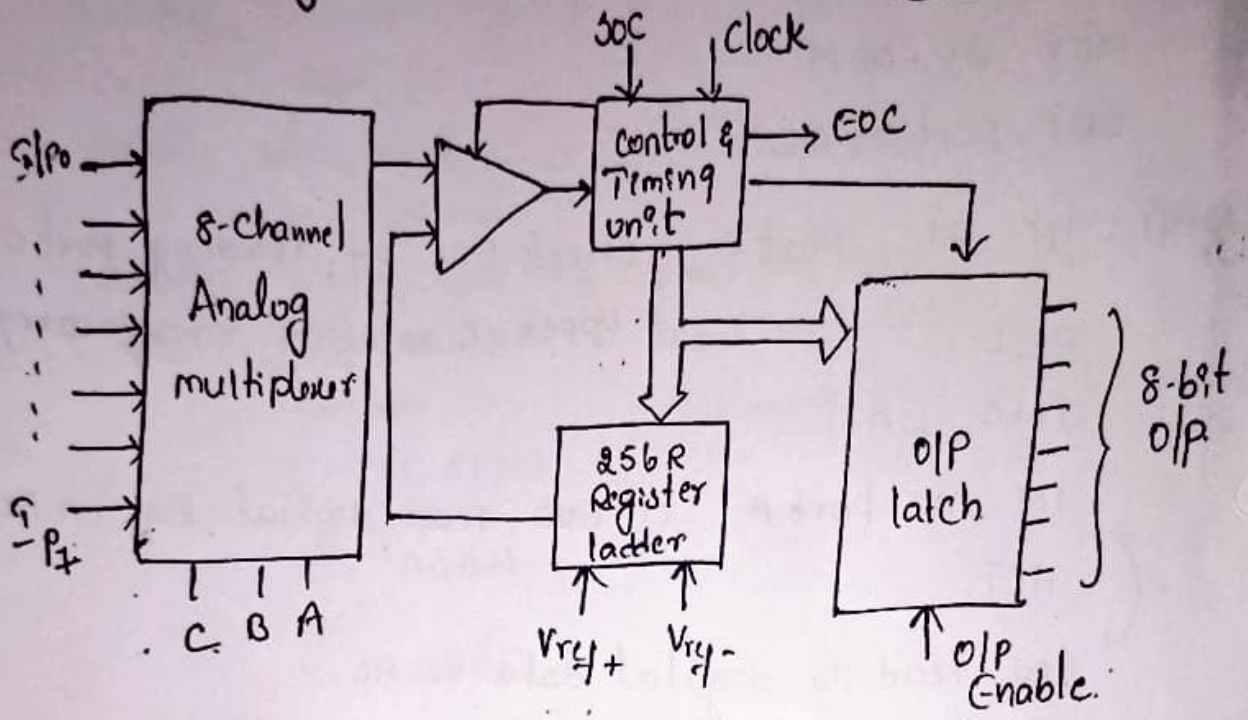
```
MOV AL, 98H ; Initialize the 8255
OUT CWR AL;
MOV AL 02H  - select I/P2 an analog i/p.
OUT portB AL   port B an o/p
MOV AL 00H  - give the start of conversion.
OUT portC AL
MOV AL, 01H
```

```
OUT Port C AL
MOV AL, 00H
OUT Port C, AL
WAIT: IN AL, Port C - Check EOC by reading port c upper & rotating through carry.
      RCL
      JNC WAIT
      IN AL Port A  ij EOC., read digital Eq. in ACK
      HLT
```

Cpu read the digital data to AL.

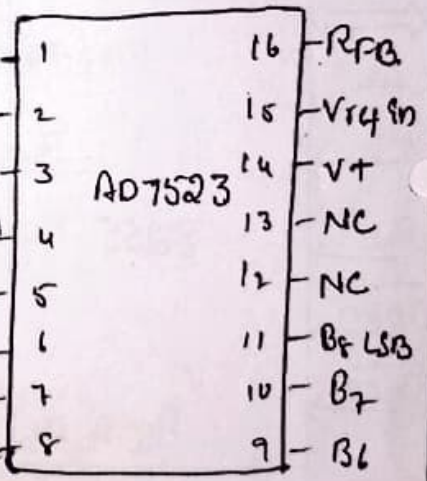


Block Diagram of ADC of 0808/0809 :-



Interfacing Digital to Analog converters.

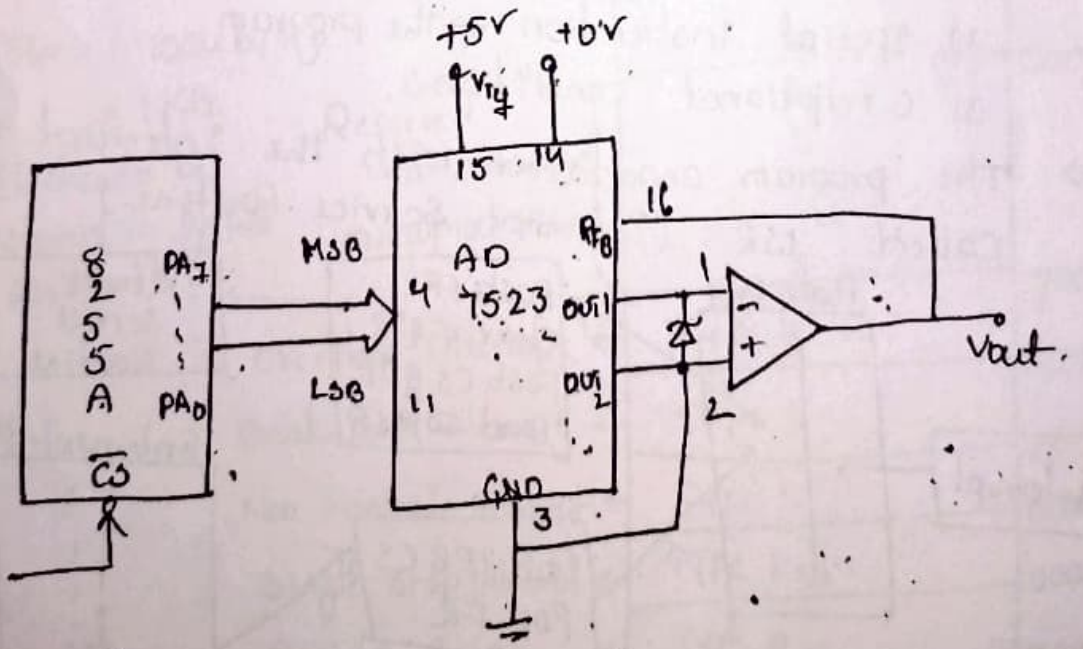
* The supply ranges from +5V to +15V. while Vref may be any where in between -10V to +10V. The max analog o/p voltage will be +10V.
* Zener is connected b/n out1 and out2 to save the DAC from negative transients.
* An op-amp is used as a current to voltage converter at the output of AD7523.



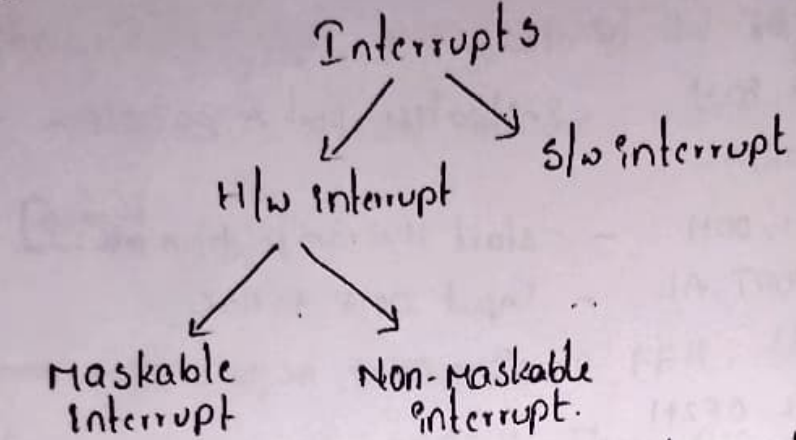
Example:- Interface DAC AD7523 with 8086 CPU running at 8MHz & write an assembly language program to generate a sawtooth waveform.

```
ASSUME  CS : CODE segment
CODE    SEGMENT
START:  MOV AL, 80H   - Initialise port A as o/p
        OUT CWR, AL
AGAIN:  MOV AL, 00H   - start the ramp from 0V.
BACK:   OUT PORT, AL  - Input 00H to DAC
        INC AL        - Increment AL to increase ramp o/p
        CMP AL, 0F2H
        JB BACK
        JMP AGAIN
CODE    ENDS
        END START.
```

* Port A is initialized as the o/p port for sending the digital data as i/p to DAC. The analog signal starts from 0V, hence AL starts with 00H.



Interrupt structure of 8086 :-

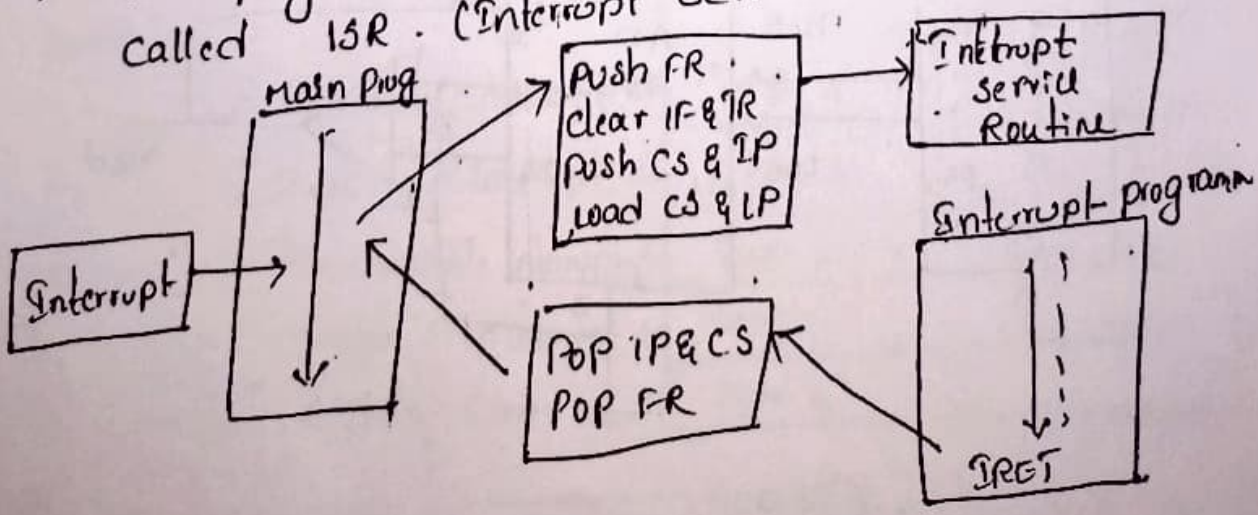


Ex: NMI - is a non maskable interrupt
INTR - is a Maskable interrupt.

The process of interrupting the normal Execution of the program to Carry out the specific Task is called an Interrupt.

→ The processor get interrupted by
1) External peripheral
2) special instruction in the program
3) Exceptional conditions.

→ The program associated with the interrupt is called ISR. (Interrupt Servic Routine)



Software Interrupts :-

→ The software interrupts can be generated by inserting the instruction 'INT' with in the program.

→ There are 256 software interrupts available in 8086

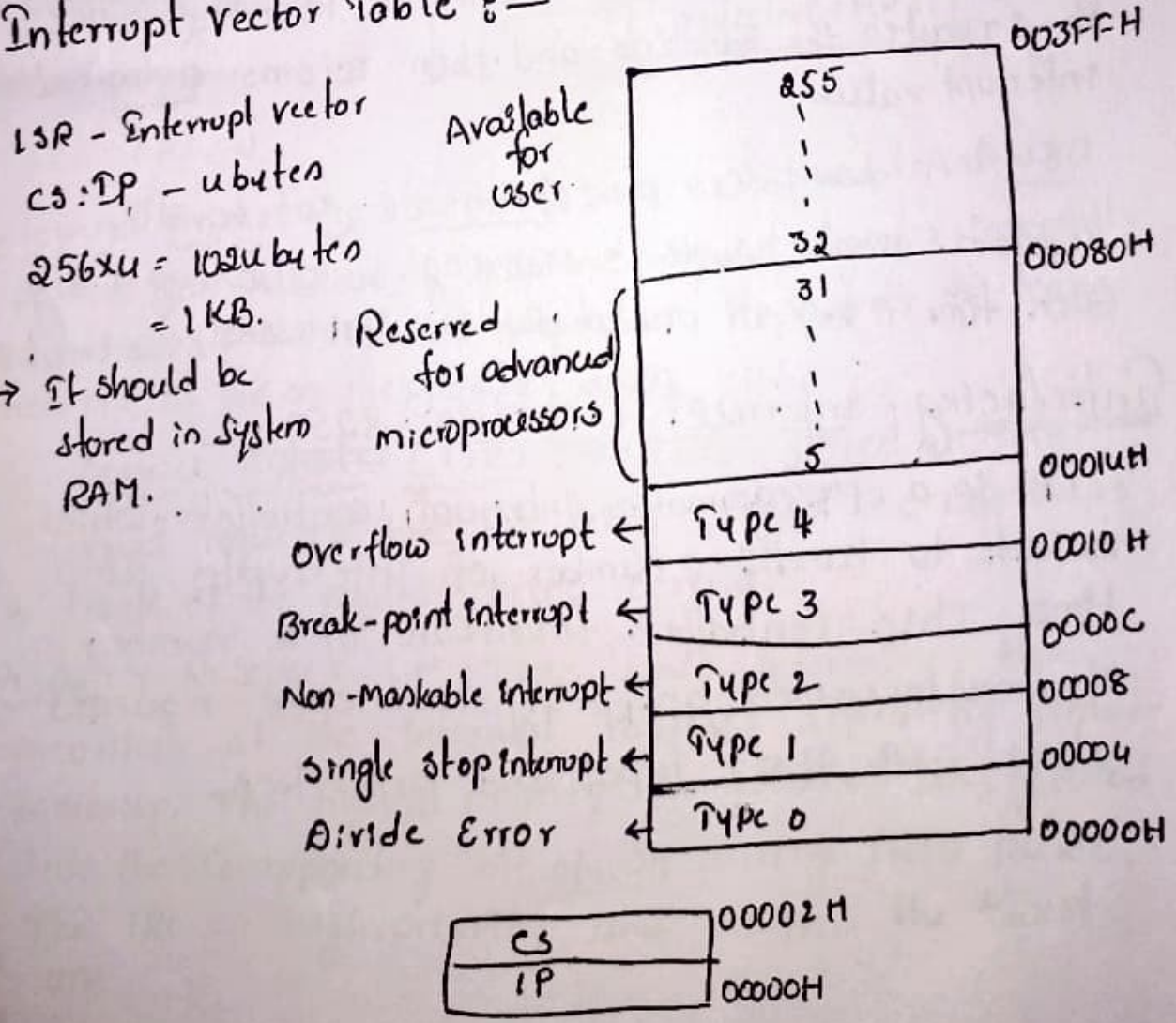
Format :

INT type

type ranges from 00 to FFH, starting address ranges from 00000H to 003FFH. These are 2 byte instruction.

- INT type Number
  - ↓ opcode
  - ↓ Immediate value.

Interrupt vector Table :-



IVR - Interrupt vector

CS:IP - 4 bytes

256 x 4 = 1024 bytes

= 1 KB.

→ It should be stored in System RAM.

Type 0 :-

→ Divide by zero

→ when the quotient from division instruction is too large. 8086 will automatically execute type 0 interrupt

→ ISR address will be stored in 4 bytes.

Type-1 :- Reserved for single step interrupt.

Type-2 :- Reserved for NMI ↓ This can be used for Debugging

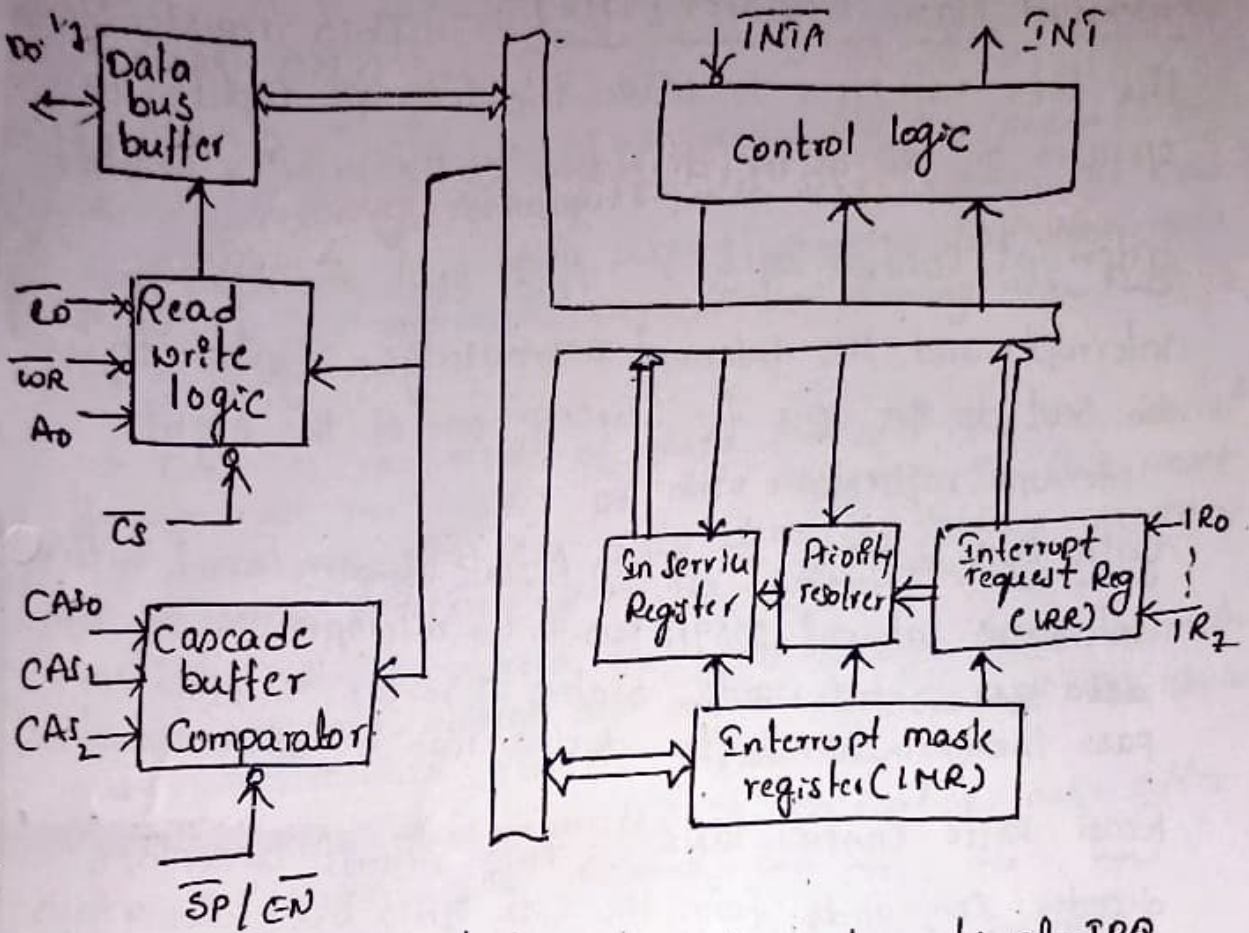
Interrupt Service Routine :-

An interrupt service routine is a software routine that hardware invokes in response to an interrupt. ISR examines an interrupt and determines how to handle it executes the handling, and then returns a logical interrupt value.

Ex: A basic example of an ISR is handles keyboard events, such as pressing (or) releasing a key. Each time a key is pressed the ISR processes the input.

Interfacing interrupt controller 8259 :-

It is a programmable interrupt controller which is able to handle a number of interrupts at a time. This controller takes care of a number of simultaneously appearing interrupts requests along with their types and priorities.



Interrupt Request Register (IRR) :- The interrupts at IRQ i/p lines are handled by interrupt Request Register internally. IRR stores all the interrupt requests in it, in order to serve them one by one on the priority basis.

In-Service Register (ISR) :- This stores all the interrupt requests those are being served. i.e ISR keeps a track of the request being served.

Priority Resolver :- This unit determines the priorities of the interrupt requests appearing simultaneously. The highest priority is selected and stored into the Corresponding bit of ISR during $\overline{INTA}$ pulse. The IR0 is high priority and IR7 has the lowest one.

Interrupt Mask Register (IMR) :- This register stores the bits required to mask the interrupt inputs. IMR operates on IRR at the direction of the priority Resolver.

Interrupt control logic :- This block manages the interrupt and the interrupt acknowledge signals to be sent to the CPU for serving one of the eight interrupt requests.

Data Bus Buffer :- This tristate bidirectional buffer interfaces internal 8259A bus to the microprocessor system data bus. control words, status & vector information pass through data buffer during read (or) write operations.

Read write control logic :- This circuit accepts & decodes commands from the CPU. This block also allows the status of the 8259A to be transferred on the data bus.

Cascade Buffer/Comparator :- This block stores & compare the IDs of all the 8259As used in the system. The three I/O pins CAS0-2 are outputs when the 8259A is used as a master. The same pins acts as inputs when the 8259A is in the slave mode. The 8259A in the master mode, sends the ID of the interrupting slave device on these lines. The slave thus selected, will send its pre programmed vector address on the data bus during the next INTA pulse.

Need of DMA :-

As we have some of dedicated peripherals & their interfacing techniques with 8086, where the CPU is to transfer bulk data using program controlled data transfer. The alternate way of transferring the bulk data is the DMA.

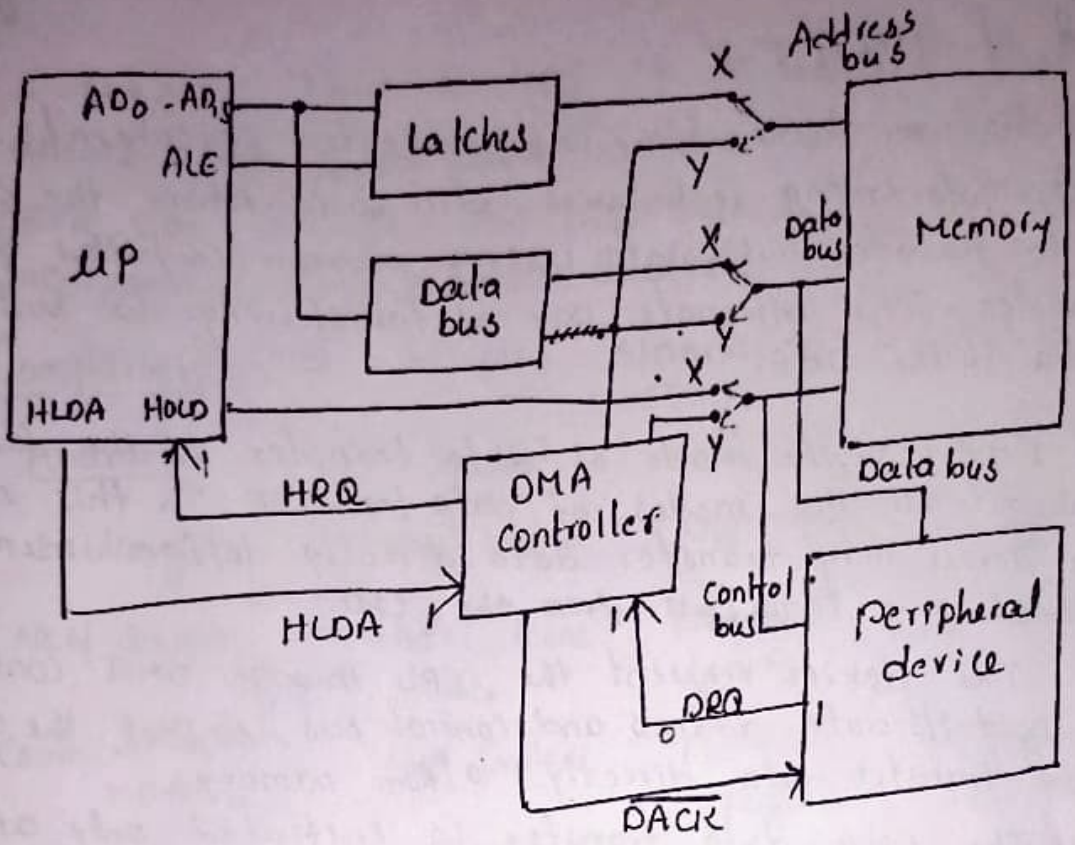
→ DMA is the mode of data transfer is the fastest amongst all the modes of data transfer. In this mode the device may transfer data directly to/from memory without any interface from the CPU.

→ The device request the CPU through DMA controller to hold its data, address and control bus, so that the device may transfer data directly to/from memory.

→ The DMA data transfer is initiated only after recving HLDA signal from the CPU.

→ Intel's 8257 is a four channel DMA controller designed to be interfaced with their family of micro processors. The 8257 request the CPU for bus access using local bus request input HOLD in min-mode & $\overline{RQ}/\overline{GT}$ in maximum mode.

→ on recving the HLDA signal (or) $\overline{RQ}/\overline{GT}$ signal from the CPU, the requesting device gets the access of the bus, and it completes the required number of DMA cycles for the data transfer. and then hand over the control of the bus back to the CPU.



Serial Communication standards :-

→ The first RS standards was RS-232 which was developed in 1962 when the need for forms of transmitting data from modems attached telephone lines to remote Communications equipments became apparent.

→ 'RS' stands for Recomended standard

→ faster communication over longer distances.

→ The most widely used are RS-422 & RS-485

RS-232 :- is the most widely used serial standard that is in use. Many laptop Computers incorporate a serial interface and it was also used on many printers.

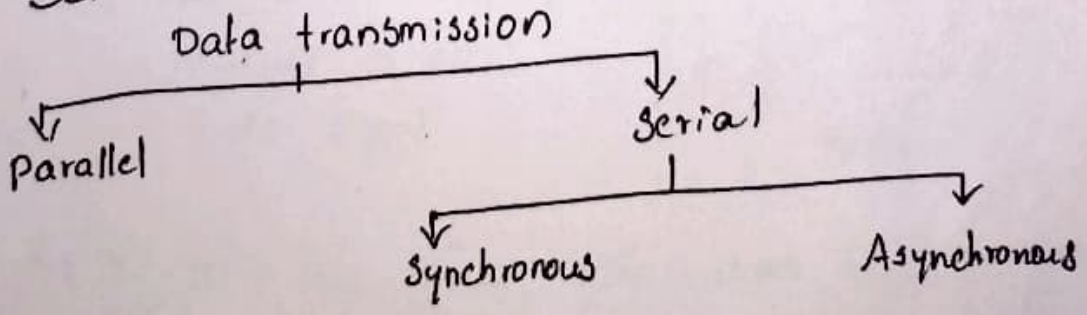
RS-422 :- This standard gives a much higher data rate than RS-232, but it uses differential transmission techniques. Many RS-422 devices are compatible with RS-232.

- 485 :- This RS-485 is a standard that allows high speed data Tx along with multiple transmitters and receivers and this makes it able to be incorporated as a network solution.

Comparision of various standards.

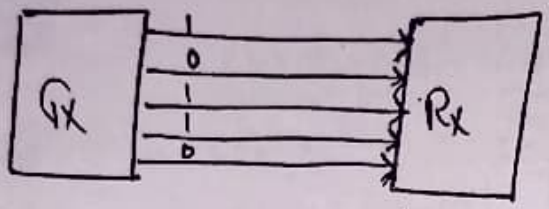
Parameter	RS-232	RS-422	RS-485
1. Cabling	single ended	Differential	Differential
2. No. of devices	one Tx & one Rx	five Tx & ten Rx	32 Tx & 32 Rx
3. Communication mode	Full Duplex	Full duplex / Half duplex	Half Duplex
4. Maximum distance	50 feet ~~at~~ 19.2Kbps	4000 feet at 100 Kbps	4000 feet at 100 Kbps.
5. Maximum data rate	19.2 Kbps	10 Mbps	10Mbps
6. Signalling mode	unbalanced	Balanced	Balanced.
7. o/p current Capability	500 mA	150 mA	250 mA
8. Mark (1 logic)	-5 to -15 V	2 to 6 V.	1.5 to 5V
9. Space (0 logic)	+5 to +15 V	2 V to 6 V	1.5 to 5 V.

Serial data transfer schemes :-

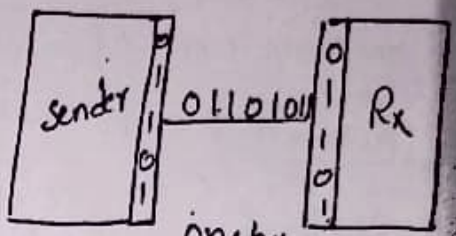


Parallel Transmission :-

→ All the bits are transfered from Soorce to destination. at a time.



Parallel



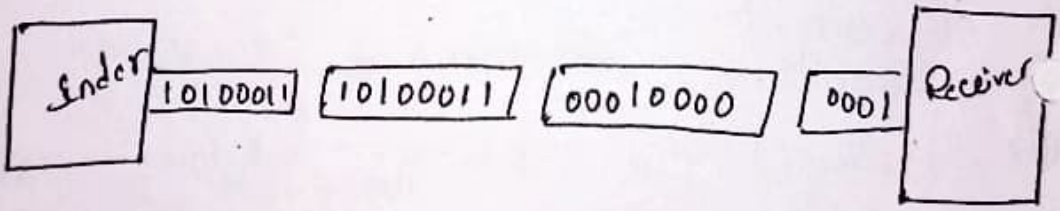
Oneby one Serial

Synchrous Transmission :-

* Sending bits one after another without start/ stop bits (or) gap:

*. It is the responsibility of the receiver to group the bits.

* The receiver counts the bits as they arive & groups them & in Eight bit units.



⇒ Serial communication standards :

Serial Transmission is the type of transmission in which a single communication link is used to transfer the data from one end to another.

Parallel Transmission is the mode of transmission in which multiple parallel links are used that transmit each bit of data simultaneously.

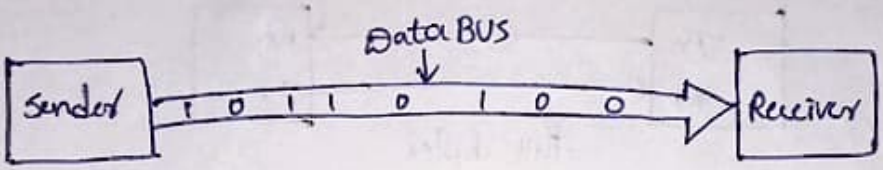


fig: Serial communication

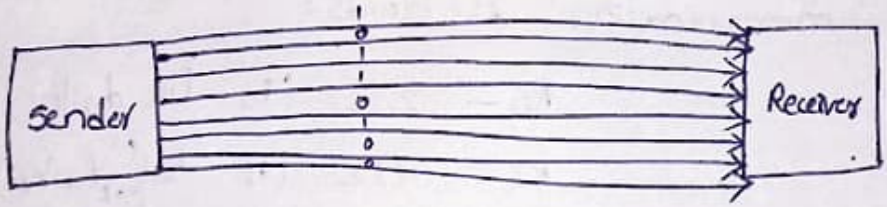
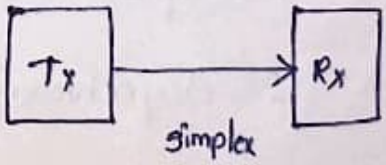


fig: Parallel communication

Methods of serial data transmission:

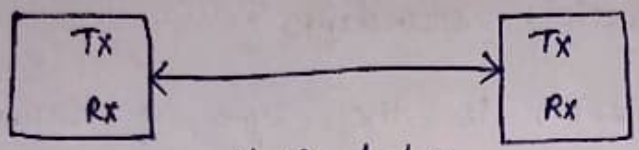
Simplex : In this mode, the data is transmitted only in one direction over a single communication channel

Ex: CPU to CRT display
Keyboard to CPU
Radio signal



Half-duplex : In this mode, the data is transmitted in both directions, but only one direction at a time i.e., Simultaneous data transfer is not possible

ex: Walkie Talkie



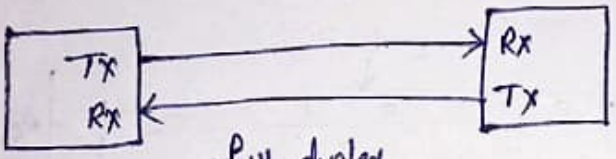
Half-duplex

Full duplex:

In this mode, the data transmission takes place in both directions simultaneously.

It Requires two channels

ex: Telephone communication



full duplex

Serial communication standards:

RS – 232 (is full duplex)

RS – 422 (is full/Half duplex)

RS – 485 (is Half duplex)

where RS → Recommended standard

⇒ Serial Data Transfer schemes: (How we transferring the data)

1. Synchronous transmission (SPI, I2C).
2. Asynchronous transmission (UART)

UNIT-3

8051 Microcontroller

Introduction to 8051 Microcontroller

Microcontroller is a single chip microcomputer which consists of CPU, Memory, I/O ports, timers and other peripherals. The difference between microprocessor and microcontroller is microprocessor is a single integrated CPU whereas microcontroller is single chip microcomputer. The world leaders of manufacturing of microprocessor and microcontroller are Intel, Motorola, IBM, Cyrix etc. Here we have to focus on microcontroller 8051.

In 1981 Intel Corporation introduced an 8 bit microcontroller called 8051.this microcontroller had 128 bytes of RAM, 4K bytes of on-chip ROM, two timers, one serial port and four ports (each 8bit wide) all on a single chip. It is an 8 bit processor means it can process 8 bit of data at a time. It has total of four I/O ports, each 8 bit wide.

Features of 8051

Feature	Quantity
ROM	4K bytes
RAM	128bytes
Timer	2
I/O pins	32
Serial Port	1
Interrupt sources	6

1. Architecture of 8051

Fig 4.1 shows a simplified architecture for the internal Hardware. Fig 4.2 shows an overview of the internal hardware architecture of the 8051/8031 microcontrollers.

The CPU has the controlled and sequencing logic circuits with signals as in a microprocessor.

The MCU has, besides the CPU, ROM, Interrupt control circuit, internal timing devices (timers T0, T1), serial interface (SI), RAM and special function registers (SFRs). It has four ports P0, P1, P2 and P3 as shown in Fig. 4.1. The overview block diagram of 8051 is depicted in Fig.4.2.

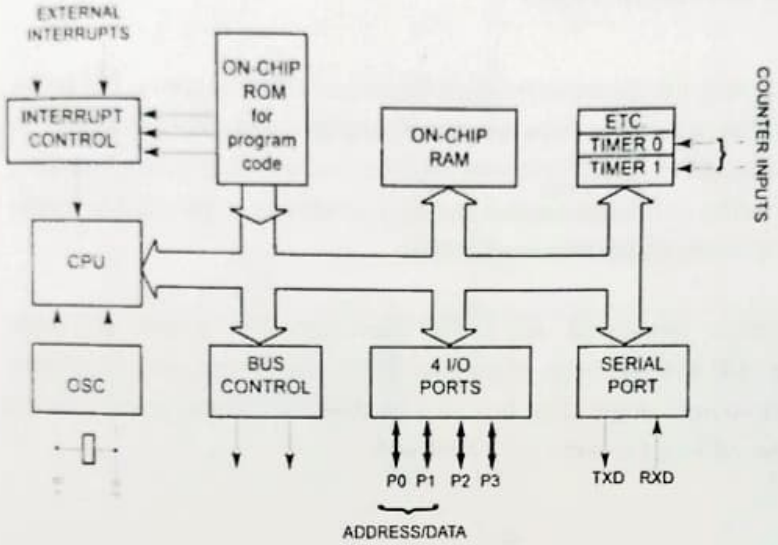


Fig. 4.1 Simplified architecture of 8051

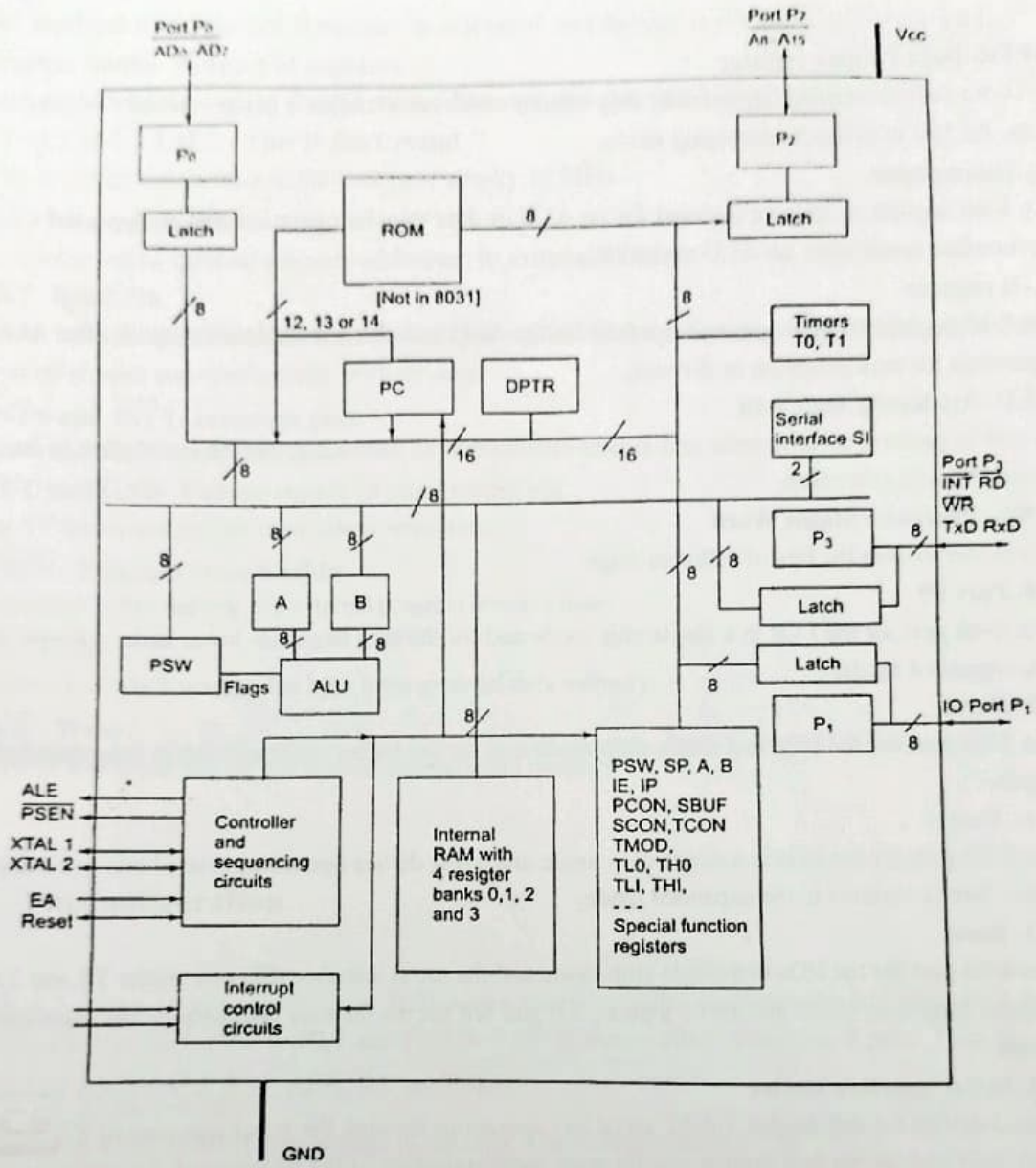


Fig.4.2 Overview (Block diagram) of 8051

Description of Sub units in the hardware architecture and meaning of the symbols

PC- Program Counter

A 16 bit register to hold the program memory address of the instruction being currently fetched. Increments continuously to point to the next instruction, unless there is change in the program flow path.

DPTR- Data Pointer register

A 16-bit register to hold the external data memory address of the data being currently fetched or to be fetched in indirect addressing mode.

A-Accumulator

An 8-bit register to save an operand for an ALU or data transfer operation and is also used to accumulate result after an ALU operation.

B- B register

An 8-bit register to save a second operand for the ALU and also accumulate the result after ALU operation for multiplication or division.

ALU- Arithmetic logic unit

A unit to perform an arithmetic and logical operation at an instance as per the instruction to be executed and give result.

PSW- Processor Status Word

A register to save the bits of different flags.

P0- Port P0

An 8-bit port for the I/Os in a single chip mode and for the data bus-cum- lower order address in the expanded mode.

P2- Port2

An 8-bit port for the I/Os in a single chip mode and for the higher order address in the expanded mode

P1- Port1

An 8-bit port for the I/Os in a single chip mode and a few device operations related bits in certain 8051 family variants in the expanded mode.

P3- Port3

An 8-bit port for the I/Os in a single chip mode and the serial interface (SI) bits , timer T0 and T1 inputs, Interrupts INT0 and INT1 inputs , $\overline{RD}$ and $\overline{WR}$ for the memory read-write in the expanded mode.

SI- Serial Interface Device

Serial device for full duplex UART serial I/O operations through the set of two pins of P3, RxD and TxD and for the half duplex synchronous communication of the bits through the same set of pins, DATA and CLOCK.

T0 and T1- Timers T0 and T1

Timing devices in 8051 family using four registers TH1, TH0, TL1, and TL0.

SFRs- Special Function Registers

All registers the SP, PSW, A, B, IE, IP, SCON, TCON, SMOD, SBUF, PCON, , TL0, TH0, TL1, TH1 are called SFRs

ROM- Read only Program memory

Masked ROM EPROM or flash EEPROM of 4kB in 8051 classic family.

Internal RAM- Internal Random Access Memory

For read and write the 128 B memory is indirectly and directly addressable in address space.

Register banks- Four set of registers

Four register banks each of 8 registers and these are also part of the internal RAM.

XTAL1 and XTAL2 – Pins to the Crystal

Pins to the crystal in the oscillator circuit, usually 12 MHz

$\overline{EA}$ - External Enable

To enable use of external memory addresses to external ROM.

RST- Reset Pin

Reset circuit input and also reset few output cycles to the external peripheral devices to let processor reset and synchronize with devices.

$\overline{INT}$ 0 and $\overline{INT}$ 1- Interrupt pins

Active low two external interrupts.

VCC and GND- Voltage supply pi and ground pin

For 5 V supply and ground connections respectively.

$\overline{PSEN}$ - Program Store Enable

Active low when reading the external program memory bytes

$\overline{RD}$ -Read

Active low when reading the byte from external data memory.

$\overline{WR}$ - Write

Active low when writing the byte to external data memory

2. Pin Configuration

Fig 4.3 shows 40 pin signals in an 8051 series microcontroller. It shows the I/O pins, P0.0 to P0.7, P1.0 to P1.7, P2.0 to P2.7 and P3.0 to P3.7. It shows other remaining 8 pins, V_{DD}, V_{SS}, XTAL1 and XTAL2, RST, ALE, $\overline{EA}$ and $\overline{PSEN}$.

Vcc - Pin 40 provides supply voltage to the chip. The voltage source is +5V

GND- Pin 20 is the ground.

XTAL1 and XTAL2- 8051 has an on-chip oscillator but requires an external clock to run it. Most upon a quartz crystal oscillator is connected to inputs XTAL1 (pin 19 and XTAL@ (pin-18) The quartz crystal oscillator connected also needs two capacitors of 30 pF. If frequency source other than crystal oscillator such as TTL oscillator will be connected to XTAL1 and XTAL2 is left unconnected.

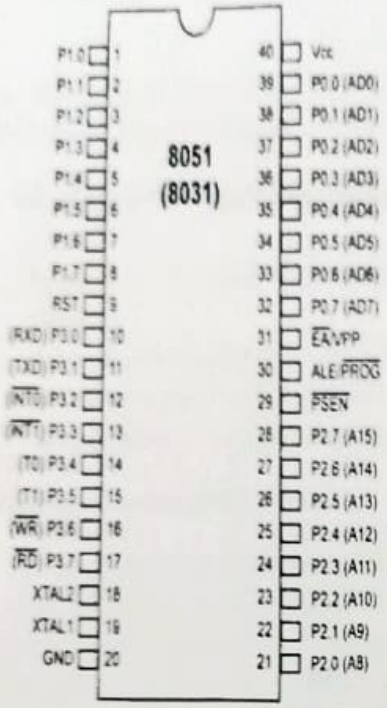


Fig. 4.3 8051 Pin diagram

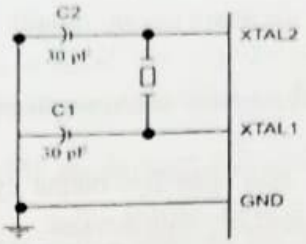


Fig. 4.4 XTAL connection to 8051

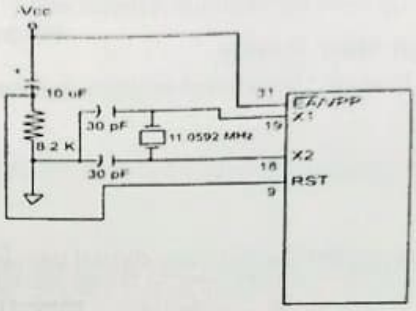


Fig. 4.5 Power-On RESET circuit

RST (I/P)- Pin 9 is the RESET pin and is active high (normally low). Upon applying high pulse to this pin the microcontroller will reset and terminate all activities. This often referred to as power on reset. Once it is activated the contents of all registers become zero except the content of SP which is 07H.

$\overline{EA}$ **(External Access)** - This pin is connected to V_{CC} for those have on-chip ROM otherwise it is grounded incase 8031 and 8032. Because in case of 8031 and 8032 there is no on-chip ROM.

$\overline{PSEN}$ **(o/p) (Program Store Enable)**- In case of 8031 based system in which an external ROM holds the program code . To read the code this pin is connected to $\overline{OE}$ pin of ROM chip.

AlE (o/p) (address Latch enable)- When 8051is connected to external memory, both address and data are transferred through port 0 pins. ALE signal is active high used to demultiplex address/data bus.

10, 18, 24, 25, 29, 40, 51, 52, 53, 59,

P0. P1. P2 and P3 are explained in port section.

3. Memory Organization

The 8051 micro controller has a total of 128 bytes of RAM. The 128 bytes of RAM inside the 8051 are assigned addresses 00H to 7FH and divided into three different groups as follows.

1. A total of 32 bytes from location s 00H to 1FH are set aside for register banks and the stacks.
2. A total of 16 bytes from locations 20H to 2FH are set aside for bit addressable read/write memory.
3. A total of 80 bytes from locations 30H to 7FH are used for read and write storage, or what is normally called a scratch pad. These 80 locations of RAM are widely used for the purpose of storing data and parameters by 8051 programmers.

Register banks in the 8051

As mentioned, a total of 32 bytes of RAM are set aside for the register banks an stack. These 32 bytes are divided into 4 banks of registers in which each bank has 8 registers, R0-R7. RAM locations from 0 to 7 are set aside for bank 0 of R0-R7 where R0 is RAM location 0 , R2 is location 2 and so on. The second bank of registers R0-R7 start RAM location 08 and goes to location 1FH. The third bank of R0-R7 starts at memory location 10 H and goes to location 17H. finally RAM location 18H to 1FH are set aside for the fourth bank of R0-R7. The following shows how 32 bytes are allocated into 4 banks.

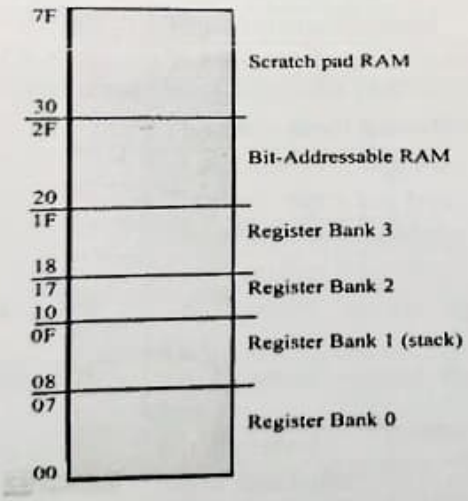
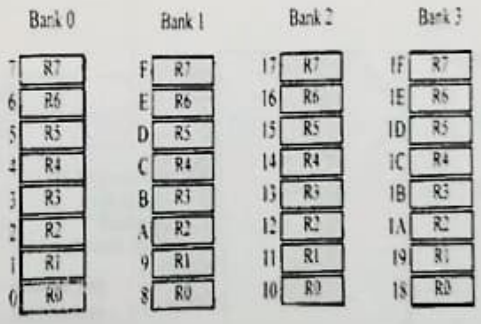


Fig. 4.6 RAM allocation in the 8051

Fig. 4.7 RAM Allocation in the 8051



External Program Memory

Fig.4.8 shows a layout of the external code memory addresses in the classic 8051 architecture.

1. When the the $\overline{EA}$ =0 at RESET, the PC (MCU program counter) starts from 0x0000 and accesses the external addresses from the memory. Memory addresses are between 0x0000 and 0xFFFF.
2. When the $\overline{EA}$ =1 at RESET, the PC starts from 0x0000 for banks0 and 1 and accesses the internal addresses and the 0x1000 onwards from the external addresses from the memory.

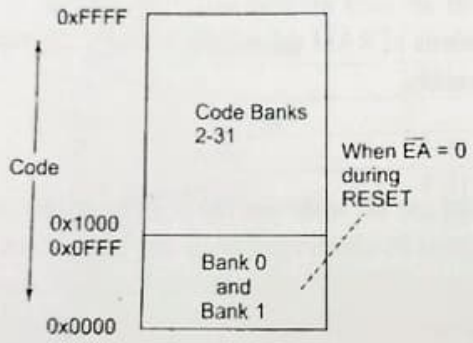


Fig. 4.8 Code Memory (Program memory)

External Data Memory

Fig. 4.9 shows a layout of the external data (X-DATA) memory addresses in the classic 8051 architecture. It can be accessed through the indirect addressing mode used.

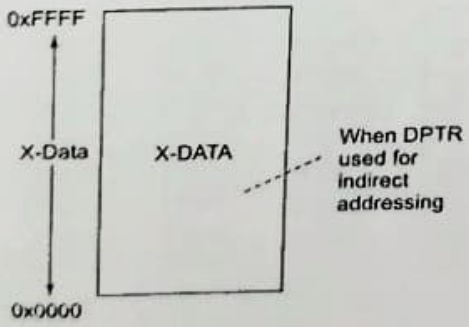


Fig. 4.9 Memory for X-Data in classic 8051

5. Special Function Registers (SFR)

For a programmer, the SFRs are at the directly addressable space special registers. These can be accessed by their names or by their addresses. The SFRs have addresses between 80H and FFH. These addresses are above 80H, since the addresses 00 to 7FH are addresses of RAM memory inside the 8051.Not all the address space of 80 to FF is used by the SFR. The unused locations 80H to FFH are reserved and must not be used by the 8051 programmer. The meaning of each symbol is enlisted in Table 4.1.

Table 4.1 Special Function Register (SFR) Address.

Symbol	Name	Address
ACC*	Accumulator	0E0H
B*	B-register	0F0H
PSW*	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 bytes	
	DPL lower byte	82H
	DPH higher byte	83H
P0*	Port0	80H
P1*	Port1	90H
P2*	Port2	0A0H
P3*	Port3	0B0H
IP*	Interrupt Priority Control	0B8H
IE*	Interrupt Enable Control	0A8H
TMOD	Timer /counter mode control	89H
TCON*	Timer/counter control	88H
T2CON*	Timer/counter 2 control	0C8H
T2MOD	Timer /counter mode control	0C9H
TH0	Timer/counter0 high byte	8CH
TL0	Timer/counter0 low byte	8AH
TH1	Timer/counter 1 high byte	8DH
TL1	Timer/counter 1 low byte	8BH
TH2	Timer/counter 2 high byte	0CDH
TL2	Timer/counter 2 low byte	0CCH
RCAP2H	T/C2 capture register high byte	0CBH
RCAP2L	T/C2 capture register high byte	0CAH
SCON*	Serial control	98H
SBUF	Serial data buffer	99H
PCON8	Power control	87H
* indicate Bit addressable		

6. Port Operation

The four ports P0, P1, P2 and P3 each use 8 pins, making them 8-bit ports. All the ports upon RESET are configured as output, ready to be used as output ports. To use any of these ports as an input port , it must be programmed. The port structure is depicted in Fig. 4.10

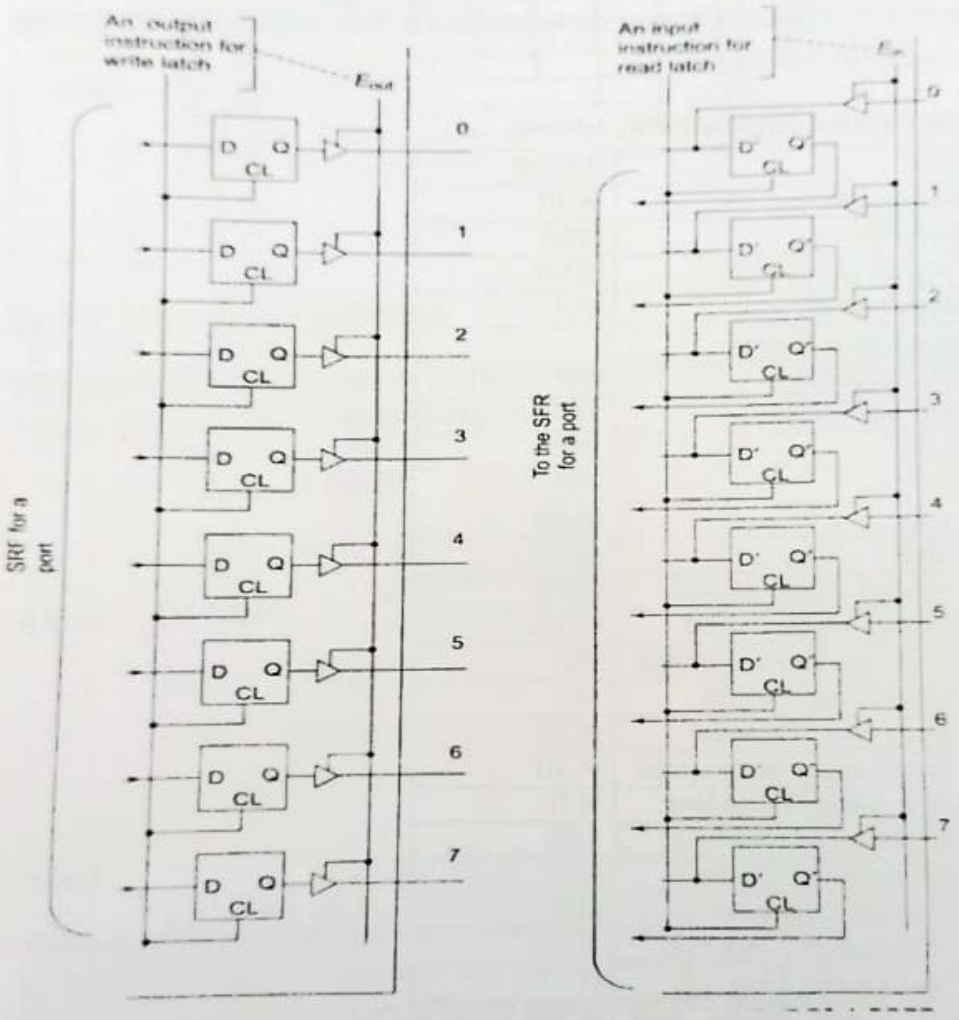


Fig.4.10 Port Structure

Port 0

It can be used for input or output. It occupies total of 8 pins (pins 32-39). To use the pins of port 0 as both input and out ports, each pin must be connected externally to a 10 K ohm pull-up resistor. P0 is an open drain unlike P1, P2 and P3. With external pull-up resistors connected upon reset, port0 is configured as an output port.

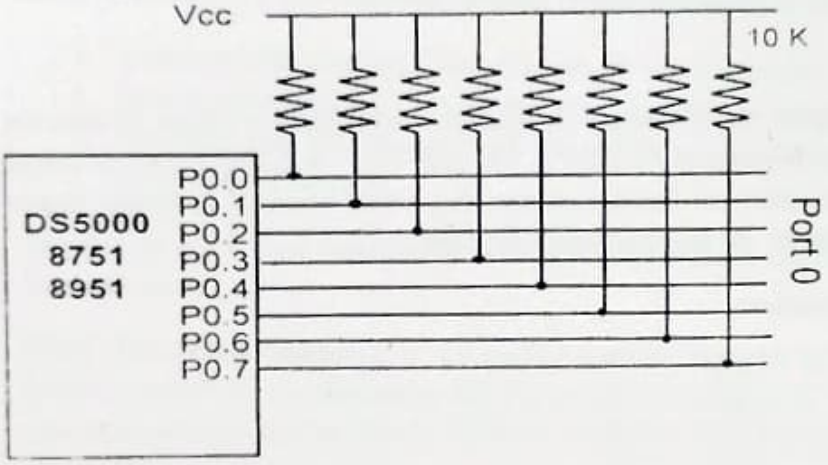


Fig. 4.11 Port 0 with pull up Resistors

With resistors connected to port 0 , in order to make it as input the port must be programmed by writing 1 to all the bits. In the following code.

```
        MOV   A, #0FFH
        MOV   P0, A
BACK:   MOVA, P0
        MOV   P1, A
        SJMP  BACK.
```

Port 1

Port 1 occupies a total of 8 pins (pins 1 through 8) . It can be used as input or output. In contrast to Port 0 , this port does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset port 1 is configured as an output port. To make Port 1 an input port it must be programmed as such by writing 1 to all its bits.

Port 2

Port 2 occupies a total of 8 pins (pins 21 through 28). It can be used as input or output. Just like P1, port 2 does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset, port 2 is configured as an output port. To make port 2 as input, it must programmed

as such by writing 1 to all its bits. The dual role of port 2 is also accomplished by providing higher byte address through A8-A15 to access the external memory.

Port 3

Port 3 occupies a total of 8 pins, pin 10 through 17. It can be used as input or output. P3 does not need any pull-up resistors , the same as P1 and P2. Although Port 3 is configured as an output port upon reset, Port 3 has additional function of providing some extremely important signals such as interrupts. Table depicts the alternate functions of port 2

Table 4.2 Port 3 alternate functions

P3 bit	Functions	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	$\overline{\text{INT0}}$	12
P3.3	$\overline{\text{INT1}}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17

P3.0 and P3.1 are used for the RxD and TxD serial communication signals. P3.2 and P3.3 are used for external interrupts. Bits P3.4 and P3.5 are used for timers 0 and 1. Bits P3.6 and P3.7 are used to provide $\overline{\text{WR}}$ and $\overline{\text{RD}}$ signals for external memories in 8051 based system.

7. Memory interfacing

Semiconductor memory

In the design of all microprocessor based system , Semiconductor memory are used as primary storage for code and data. It can be in units of K bits, M bits and so on. Semiconductor memories are conducted directly to the CPU and is also called as primary memory. The widely used semiconductor memories are ROM and RAM.

Characteristics of Semiconductor Memory

Memory capacity- The number of bits that a semiconductor memory chip can store is called chip capacity.

Memory organization- Memory chips are organized into number of locations within the IC. Each location hold 1 bit, 4bits, 8bits or even 16 bits, depending on how it is designed internally.

1. A memory chip contains 2^x locations where x is the number of address pins.
2. Each location contains y bits, where y is the number of data pins on the chip.
3. The entire chip will contain $2^x \times y$ bits, where x is the number of address pins and y is the number of data pins

Speed- One of the most important characteristics of a memory chip is the speed at which its data can be accessed.

ROM (Read-Only-Memory)- It is a type of memory that does not loss its contents when the power is turned off. For this reason ROM is called volatile memory. There are different types of read-only- memory such as PROM, EPROM, EEPROM, Flash EPROM and mask ROM.

PROM- It refers to the kind of ROM that the user can burn information into it. That's why it is called as user-programmable memory. For every bit of the PROM, there exists a fuse. So it is programmed by blowing of fuses. It is also referred to as OTP (one –time programmable)

EPROM (Erasable Programmable ROM)- In EPROM, one can program the memory chip and erase it thousands of times. A widely used EPROM is called UV-EPROM. The content of UV-EPROM is erased when it is exposed to ultra violet light. Its erase time is near about 20 minutes.

EEPROM (Electrically Erasable Programmable ROM)- Its desired contents are erased by electrically.

Flash memory EPROM- This memory has become popular user-programmable memory chip, due to the process of erasure of the entire contents takes less than a second. As the erasure method is electrical sometimes it is called as Flash EEPROM.

Mask ROM- Mask ROM refers to kind of ROM in which the contents are programmed by the IC manufacturer. It is not a user-programmable ROM.

RAM (Random Access Memory)- It is called volatile memory since cutting of the power to the IC will result in the loss of data. Sometimes it is called as read and write memory (RAWM). There are three types of RAM: Static RAM (SRAM), NV-RAM (Nonvolatile RAM) and dynamic RAM (DRAM).

NV-RAM (Nonvolatile RAM)-This RAM is nonvolatile. Like other RAMs it allows the CPU to read write to it, but when the power is turned off, the contents are not lost. To retain its content every NV-RAM chip internally is made of the following components.

1. It uses extremely power-efficient. SRAM cells built out of CMOS
2. It uses an internal lithium battery as back energy source.
3. It uses an intelligent control circuitry. The main job of internal circuitry to monitor the V_{cc} pin constantly to detect the loss of external power supply. If the power to the Vcc pin falls the below out-of-tolerance condition, the control circuitry switches automatically to its internal power source, lithium battery.

DRAM (Dynamic RAM)-The use of a capacitor as a means to store data cuts down the number of transistors needed to build up the cell; however it requires constant refreshing due to leakage. This is in contrast to SRAM whose cells are made of flip-lops. The use of capacitor as storage cells in DRAM results in much smaller net memory size.

Fig. 4.12 2764 ROM 8Kx8

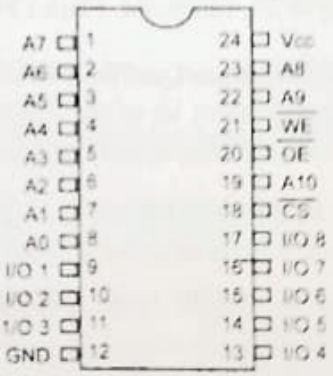


Fig. 4.13 2Kx8 SRAM pins

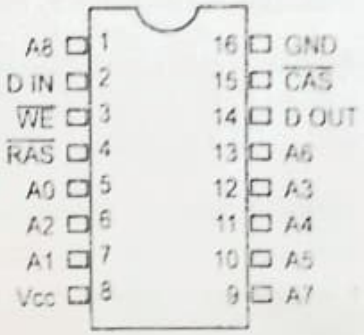


Fig. 4.14 256Kx1 DRAM

Memory Address Decoding

The job of the decoding circuitry to locate the selected memory block that CPU has access to desired data in memory chip. Memory chips have one more pins called CS (chip select) which must be activated for the memory contents to be accessed. Sometimes the chip select is also referred to as Chip Enable (CE).

Following points are required for interfacing the memory to the CPU.

1. The data bus of the CPU is connected directly to data pins of the memory chip.
2. Control signals RD (read) and WR write from the CPU are connected to the OE (output enable) and WE (write enable) pins of memory chips respectively.
3. In case of the address buses, while lower bits of the addresses from the CPU are connected directly to the address pins of the memory chips and upper address pins are used to activate the CS or CE pin of the memory chip. The CS or CE pin along with RD/WR allows the flow of data in or out of the memory chip.

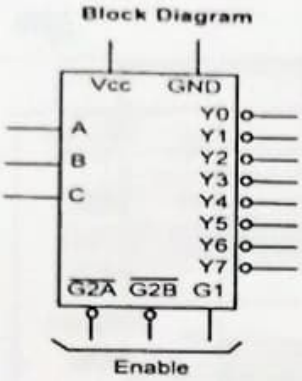


Fig. 4.15 74LS138 Decoder

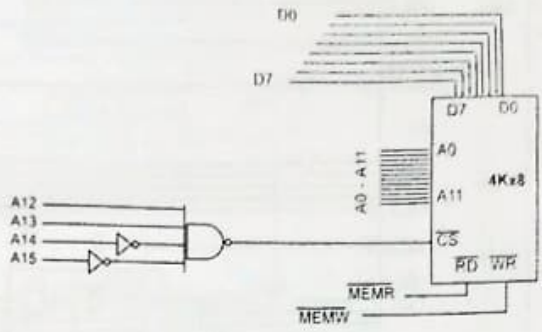


Fig. 4.16 logic Gate as Decoder

6.3 Interfacing with External ROM/RAM as Program and Data Memory

For interfacing to external ROM some pins have important role that to be discussed here.

$\overline{EA}$-When this pin is connected to Vcc, that indicates the program code is stored in the microcontroller on-chip ROM. For external ROM access tis pin is grounded.

P0 and P2 role in providing addresses- In 8051 P0 and P2 provides the 16-bit address to access external memory. Of these ports P0 provides the lower 8 bit addresses A0-A7, and P2 provides the upper 8 bit addresses A8-A15. More importantly, P0 is also used to provide 8 bit- data bus D0-D7. In other words P0.0- P0.7 are used for both address and Data is called as address/data multiplexing. The sharing of this bus is accomplished by ALE (address latch enable.) Pin.
When ALE=0, the 8051 uses P0 for the data path and when ALE=1, it is used for address path.

$\overline{PSEN}$ **(program store enable)-** It is an output signal must be connected to OE pin of a ROM containing the program code. When $\overline{EA}$ pin is connected to ground the 8051 fetches opcode from external ROM by using $\overline{PSEN}$.

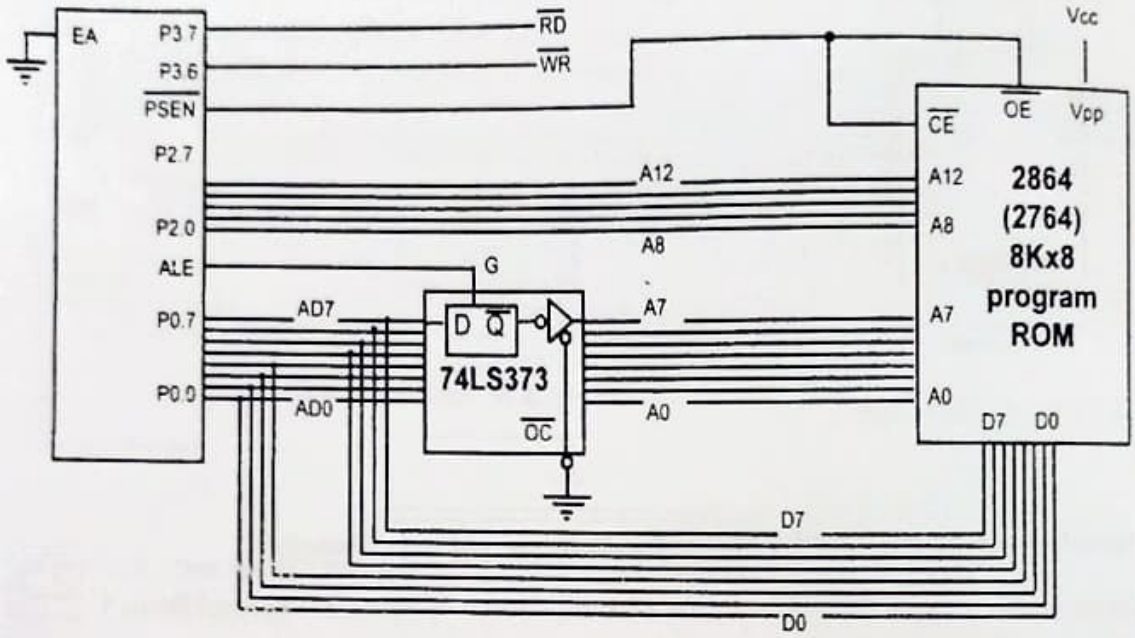


Fig. 4.17 Interfacing of ROM to 8051 as program memory

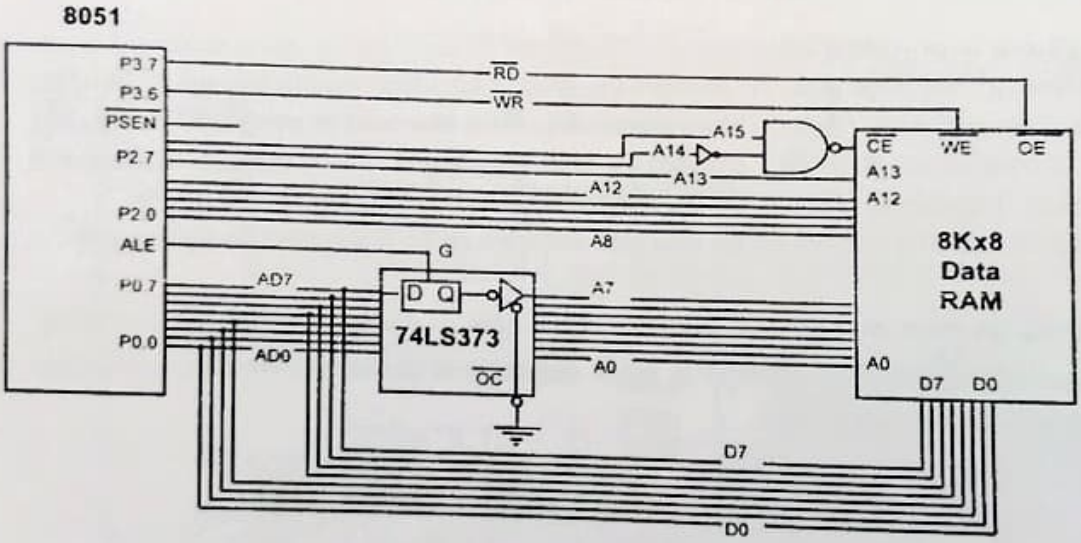


Fig. 4.18 Interfacing of ROM as Data Memory

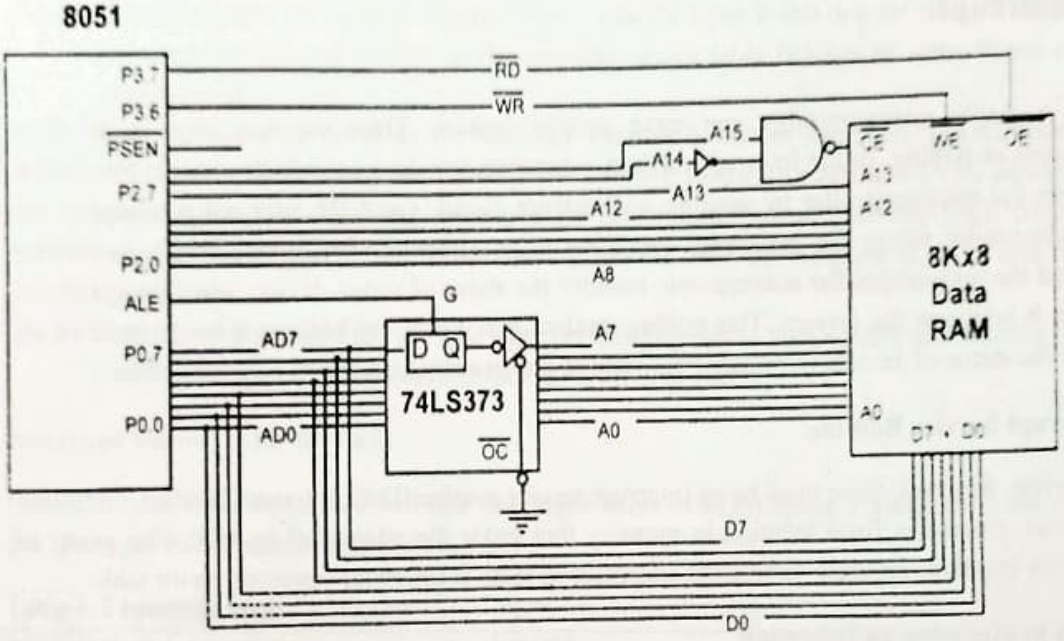


Fig. 4.19 Interfacing with Data RAM

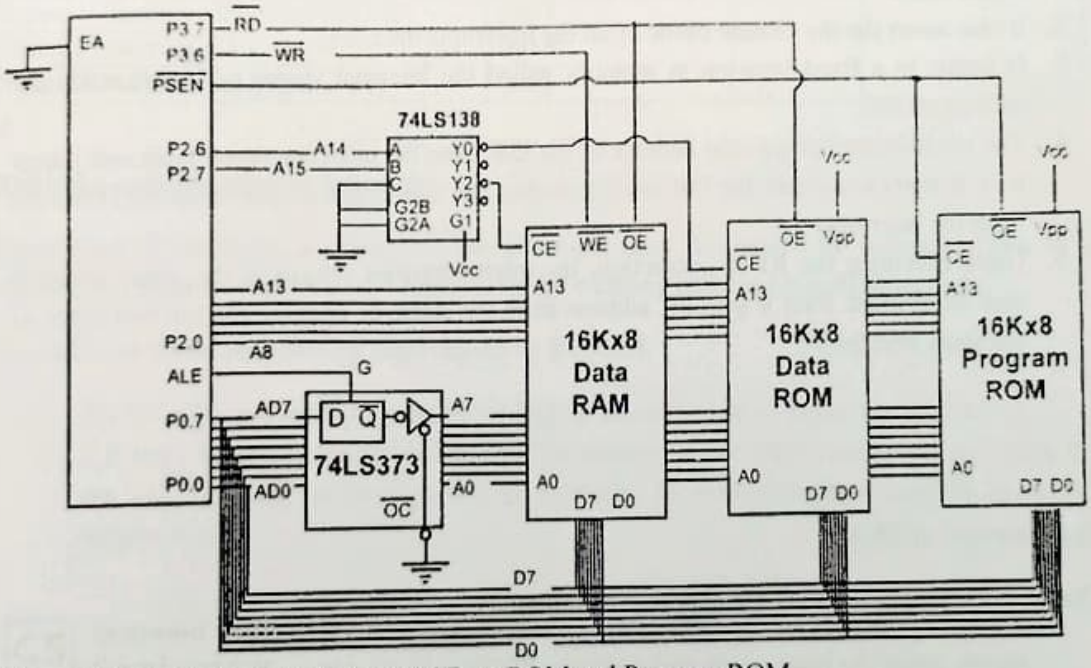


Fig. 4.20 Interfacing with Data RAM Data ROM and Program ROM

8. Interrupt

A single microcontroller can serve several devices. There are two ways to do that: interrupts or polling. In the Interrupt method, whenever any device needs its service, the device notifies the microcontroller by sending an interrupt signal. Once the interrupt is accepted the microcontroller serves the device by executing an interrupt service routine (ISR). In polling method the microcontroller continuously monitor the status of a give device, when the condition is met it performs the service. This polling method is not efficient because it has to monitor all times the status of devices in round-robin fashion and priority assignment is not possible.

Interrupt Service Routine

For every interrupt, there must be an Interrupt service routine (ISR), Interrupt handler. For every interrupt, there is a fixed location in memory that holds the address of its ISR. The group of memory locations set aside to hold the addresses of ISRs is called the interrupt vector table.

Steps in executing an Interrupt

Once an interrupt is activated, microcontroller performs the following steps.

1. It finishes the instruction it is executing and save the address of the next instruction (PC) on the stack.
2. It also saves the the current status of all the interrupts internally.
3. It jumps to a fixed location in memory called the interrupt vector table that holds the address of ISR.
4. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the ISR until it reaches last instruction of subroutine RETI (return from the interrupt).
5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted. First it gets PC address from the stack by popping the top two bytes of the stack into the PC

Six Interrupts of 8051

The six interrupts in the 8051 are allocated as follows

1. Reset- when the reset pin is activated, the 8051 jumps to address location 0000. This is power-up reset.

2. Two interrupts are set aside for the timers: one for timer 0 and one for timer 1. Memory locations 000BH and 001BH in the interrupt vector table belongs to timer 0 and timer 1.respectively.
3. Two interrupts are set aside for hardware external hardware interrupts. Pin numbers 12 (P3.2) and 13(P3.3) in port 3 are for the external hardware interrupts INT0 and INT1, respectively. These external interrupts are also referred to as EX1 and EX2. Memory location 0003H and 0013H in the interrupt vector table are assigned to INT0 and INT1 respectively.
4. Serial communication has a single interrupt that belongs to both receive and transfer. The interrupt vector table location 0023H belongs to this interrupt.

Interrupt Vector Table for 8051

From the table it has been observed that only three bytes of ROM space is assigned to the reset pin. They are ROM address locations 0,1 and 2.

Table 4.2 Interrupt vector addresses

Interrupt	**ROM Location(Hex)**	**Pin**
Reset	0000	9
External hardware interrupt (INT0)	0003	12
Timer 0 interrupt (TF0)	000B	13
External hardware interrupt (INT1)	0013	
Timer 1 interrupt (TF1)	001B	
Serial COM interrupt (RI and TI)	0023	

Enabling and disabling an interrupt

Upon reset all interrupts are disabled. The interrupts must be enabled by software. There is a register IE (interrupt enable) that is responsible for enabling and disabling the interrupts.

To enable an interrupt following steps should be followed.

1. Bit D7of IE register (EA) must be set high to allow the rest of register to take effect.
2. If EA=1 , interrupts are enabled and will be responded to if their corresponding bits in IE are high. If EA=0, no interrupt will be responded to, even if the associated bit in the IE register is high.

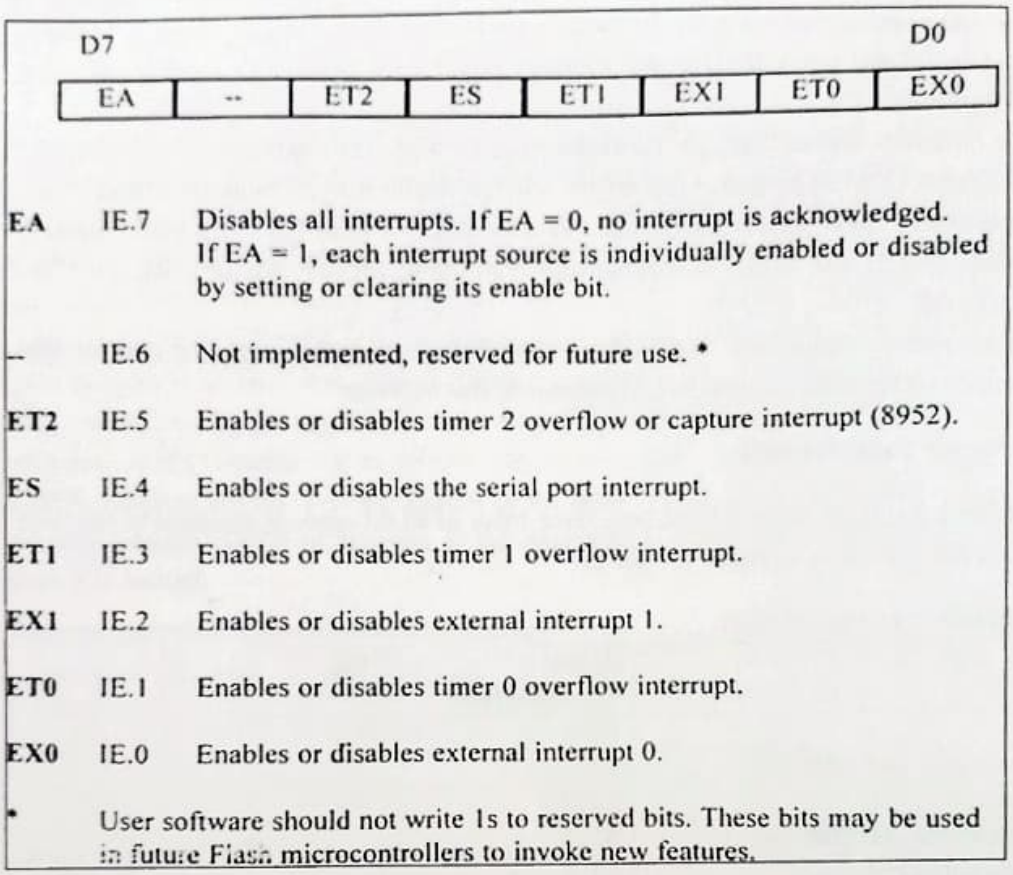


Fig. 4.21 Interrupt Enable Register

Interrupt Priority in the 8051

When the 8051 is powered up, the priorities are assigned, that are enlisted in table.

Table 4.3 Interrupt priority

Highest to Lowest priority	
External Interrupt 0	INT0
Timer Interrupt 0	TF0
External Interrupt 1	INT1
Timer Interrupt 1	TF1
Serial Communication	RI-TI

9. Programmer's Model

The CPU registers are used to store the data temporarily. The information may be data to be processed or address pointing the data to be fetched. The majority of registers are 8 bits. The 8-bit registers are shown in the diagram from MSB (most significant bit) D7 to the LSB (least significant bit) D0. The most widely used registers of 8051 are A (accumulator), B, R0, R1, R2, R3, R4, R5, R6, R7, DPTR (data pointer), and PC (program counter). All these registers are 8 bits except DPTR and the program counter. The accumulator is used to hold one operand before execution and hold the result after execution. The program counter points to the address of next instruction to be fetched. It is a auto increment register. As the size of program counter is 16 bit. 8051 can access the program addresses from 0000H-FFFFH. When 8051 is powered-up the program counter contents will be 0000H. This means that it expects the first opcode to be stored at ROM address 0000H. For this reason in the 8051 system, the first opcode must be burned memory location 0000H of program ROM since this is where it looks for the first instruction when it is booted.

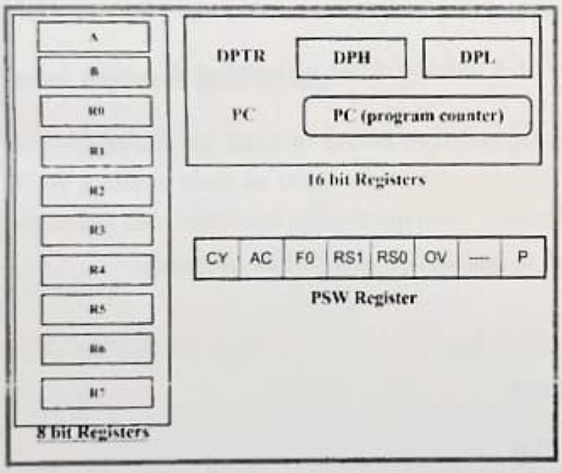


Fig. 4.22 Programmer's model

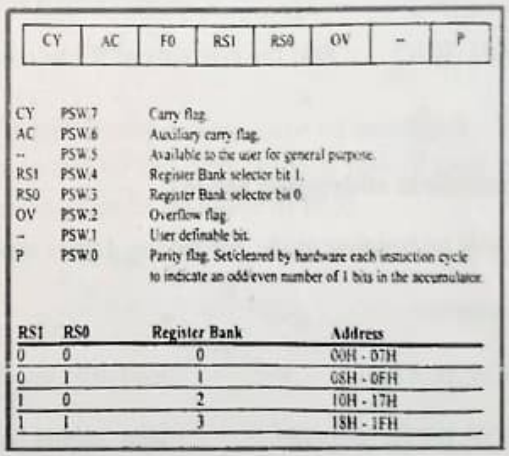


Fig. 4.23 PSW register

PSW (program status word register)

The program status word register (PSW) is an 8-bit register. It is also referred as Flag register. Although this register is size of 8-bits, only 6bits are used by 8051. Two unused bits are user definable flags. Other 4 bits are called as conditional flags such as CY (carry), AC (auxiliary carry), P(parity) and OV(overflow).In this register the bits PSW.3 and PSW.4 are designated as RS0 and RS1 and used to select the banks. PSW.5 and PSW.1 bits are general purpose status flags and can be used by the programmer for any purpose.

10. Operand addressing

An addressing mode is a method of specifying the data source or destination in an instruction. There are 5 types of addressing modes is supported by 8051.

1. Register
2. Immediate
3. Direct (memory related)
4. Register Indirect (memory related)
5. Index register addressing

Register addressing mode

This addressing mode involves the use of registers to hold the data to be manipulated.

Examples:

MOV A, R0 ; Copy the contents of R0 int A

ADD A, R7 ; Add the contents of R7 to contents of A and the result is stored in A

Immediate addressing mode

In this addressing mode immediate data is specified in instruction as a source operand.

Examples:

MOV B, #40H ; load 40H into B register

MOV DPTR, #2000H ; load 2000H into DPTR

Direct addressing mode

As we know the on-chip RAM of 8051 is 128 byte, it can be accessed through memory address from 00H to FF H. The allocations of 128 bytes are as follows.

1. RAM location 00H-1FH are assigned to register banks and stack
2. RAM location 20H-2FH is set aside as bit-addressable space to save single bit data.
3. RAM location 30H-7F is available as place to save bite-sized data.

Although the entire 128 bytes of RAM can be accessed through direct addressing mode, it is most often used to access RAM location 30H-7FH. This is due to fact that register banks are accessed through their names.

Examples:

MOV R4, 70H ; move the contents of RAM location 70H to R4.

MOV 56H, A ; save the content of A in RAM location 56H

PUSH 05 ; push R5 onto the stack

Register indirect addressing mode

In this mode the address (of 8bits) is indirectly specified in the instruction by the contents of pointer. This addressing mode so called because the source operand is from the address specified indirectly by another register in the instruction. The limitation is that only R0 and R1 register can be used in 8051 for indirect addressing. SFRs are directly accessible.

Examples

MOV R1, #55H ; load pointer R1=55H

MOV A, @R1 ; the content of pointer is transferred to A

Index registers addressing

Suppose we need to access external data RAM and external code space of on-chip ROM 16 bit address must be required. In this case we have to use DPTR. This mode is widely used in accessing data elements of look-up table entries in the program ROM space of 8051.

Examples;

MOV DPTR, #0200H ; load DPTR with 0200

CLR A ; clear accumulator

MOVC A,@A+DPTR ; Move the content 0200 location into A

11. Instruction set

The instruction set of 8051 can be classified into following group.

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logic Instructions
4. Boolean Variable manipulation Instructions
5. Program flow control (Processor and Machine control) Instructions
6. Interrupt flow Control instruction

Data Transfer Instruction

Three types of the data transfer can be done by move instruction. First type is transfer within the internal RAM and SFRs, second type is transfer using code memory area (CODE) and the third is using the external data memory X-DATA).

MOV instruction

A MOV instruction means move (copy) the bits from one source to a destination.

Table 4.4 MOV instructions within the registers, internal RAM and SFRs in 8051

Instruction (Mnemonic)	Action	Addressing	Length in bytes	cycles
MOV A, Rn	Move Rn into A	Register	1	1
MOV Rn, A	Move into Rn from A	Register	1	1
MOV A, #data	Move immediate 8-bit data into A	Immediate	2	1
MOV Rn, #data	Move into Rn the data.	immediate	2	1
MOV A, direct	Move byte at the direct address into A	Direct	2	1
MOV Rn, direct	Move from direct address into Rn	Direct	2	2
MOV direct, A	Move byte to the direct address form A	Direct	2	1
MOV direct, Rn	Move a byte to the direct address from Rn	Direct	2	2
M OV direct, direct	Move byte to the direct address from the direct address	Direct	3	2
MOV direct, #data	Move immediate data byte to the direct address	Immediate	3	2
MOV a,@Ri	Move into A the byte from the address pointed by Ri	Indirect	2	2
MOV @Ri, A	Move A into address pointed by Ri	Indirect	1	1
MOV direct, @Ri	Move into direct address from address pointed by Ri	indirect	1	1
MOV @Ri, direct	Move from the direct address to the address poined by ri	Indirect	2	2
MOV @Ri, #data	Move data ino address pointed by Ri	immediate	2	2
MOV DPTR, data16	Mov e16 bit dat	immediate	3	2

MOVC-type Instruction

It moves the 8-bit code from one source at the program memory (internal and external) to the register A destination.

Table 4.5 MOVC Instructions for transfer from the program memory area address code or constant to accumulator in 8051

Instruction	Action	Addressing	Length in bytes	Cycles
MOVC A, @A+DPTR	Moves the code or constant into A the byte from the program memory address pointed	Indirect	1	2

	by hypothetical addition of DPTR with the A itself.			
MOVC A, @A+PC	Move the code or constant into A the byte from the program memory address pointed by hypothetical addition of PC with the A itself	Indirect	1	2

MOX-type Instructions

A MOVX instruction means move (copy) the 8-bit data into A and from A using the external data memory address using DPTR or Ri as the pointer

Table 4.6 MOVX instruction

Instruction	Action	Addressing	Length in bytes	Cycles
MOVX A, @DPTR	Move the external data byte (X-DATA) into A from the data memory address pointed by DPTR	Indirect	1	2
MOVX @DPTR,A	Move into the external data memory from A to the address pointed by DPTR	Indirect	1	2
MOVX A,@Ri	Move the external data byte into a from the memory address pointed by Ri	Indirect	1	2
MOVX @Ri, A	Move into the external data memory from A to the memory address pointed by Ri	Indirect	1	2

Table 4.7 PUSH and POP instructions for using the Stack Area employing SP

Instruction	Action	Addressing	Length in bytes	Cycles
PUSH direct	Move byte from a direct internal RAM or SFR into the stack after first incrementing the stack pointer by 1	Direct	2	2
POP direct	Move byte to a direct internal RAM or SFR into the stack and then decrement the stack pointer by 1.	Direct	2	2

XCH-type instructions

An XCH instruction is for exchanging the A register with a source using the register (direct or indirect addresing0 mode.

Table 4.8 XCH and XCHD instruction

Instruction	Action	Addressing	Length in bytes	cycles
XCH A@Ri	Exchange byte at A with the address pointed by Ri	Indirect	1	2
XCH A,Rn	Exchange byte at A with the register Rn	Register	1	2
XCH A, direct	Exchange byte at A with the byte at a direct address.	Direct	1	1
XCHD A,@Ri	Exchange lower hex-digits of the bytes at A with the address pointed by Ri	Indirect	1	2

Arithmetic Instruction

These instructions include 8 bit addition, subtraction, increment, decrement, multiply and division instruction.

Table 4.9 Arithmetic ADD, SUB,MUL, DIV, INC and DEC instruction s in 8051

Instruction	Action	Addressing	Flags affected	Length (bytes)	Cycles
ADD A,Rn	Add Rn into A	Register	C,AC,OV	1	1
ADD A, direct	Add the byte at the direct address into A	Direct	C,AC,OV	2	1
ADD A, @Ri	Add the byte from the address pointed by the Ri into A	Indirect	C,AC,OV	1	1
ADD A, #data	Add immediate data byte to the A	Immediate	C,AC,OV	2	1
ADDC A, Rn	Add CF(carry) bit and Rn into A	Register	C,AC,OV	1	1
ADDC A, direct	Add CF bit and byte at the direct address ito A	Direct	C,AC,OV	2	1
ADDC A @Ri	Add CF bit and the byte from the address pointed by the Ri	Indirect	C,AC,OV	1	1
ADDC A, #data	Add CF bit and immediate data byte to the A	Immediate	C,AC,OV	2	1
SBBB A,Rn	Subtract borrow at CF bit and Rn into A	Rgister	C,AC,OV	1	1
SBBB A, direct	Subtract borrow at CF bit and byte at the direct address into A	Direct	C,AC,OV	2	1
SBBB A, @Ri	Subtract borrow at C bit and byte at the byte from the address pointed	Indirect	C,AC,OV	1	1

	by the Ri into A				
SBBB A, #data	Subtract borrow at CF bit and immediate data byte into A	Immediate	C,AC,OV	2	1
INC A	Increment	Register	None	1	1
INC Rn	Increment Rn	Register	None	1	1
INC direct	Increment byte at the direct address	Direct	None	2	1
INC @Ri	Increment the byte at the address pointed by Ri	Indirect	None	1	1
DEC A	Decrement A	Register	None	1	1
DEC Rn	Decrement Rn	Register	None	1	1
DEC direct	Decrement byte at the direct address	Direct	None	2	1
DEC @Ri	Decrement the byte at the address pointed by the Ri	Indirect	None	1	1
MUL AB	Multiply A and B Result MSB in B and LSB in A	Register	OV	1	4
DIV AB	Divide A (Numerator) and B(denominator) Remainder in B Quotient in A	Register	OV	1	4
DAA	Decimal adjust accumulator	Register	C	1	1

Logical Instruction

Table gives features of 8-bit AND, OR and XOR instruction. These instructions have 4 addressing modes such as register, immediate, direct and indirect.

Table 4.10 ANL, ORL XRL instruction

Instruction	Action	Addressing	Length in bytes	Cycles
ANL A, Rn	AND Rn into A	Register	1	1
ANL A, direct	AND byte at the direct address into A	Direct	2	1
ANL A, @Ri	AND into the byte from the address pointed by the Ri	Indirect	1	1
ANL A, #data	AND immediate data byte into A	immediate	2	1
ANL direct, A	AND A into byte at the direct address	Direct	2	1
ANL direct, #data	AND immediate byte into byte at the direct address	Direct	3	2
ORL A, Rn	OR Rn into A	Register	1	1
ORL A, direct	OR byte at the direct address into A	Direct	2	1
ORL A, @Ri	OR into the byte from the address pointed by Ri	Indirect	1	1
ORL A, #data	OR immediate data byte to the A	immediate	2	1
ORL direct, A	OR A into byte at the direct address	Direct	2	1
ORL direct,#data	OR immediate byte into byte at the	Direct	3	2

	direct address			
XRL A, Rn	XOR Rn into A	Register	1	1
XRL A, direct	XOR byte at the direct address into A	Direct	2	1
XRL A, @Ri	XOR the byte at the address pointed by Ri into A	Indirect	1	1
XRL A, #data	XOR immediate data byte to the A	immediate	2	1
XRL direct, A	XOR A into byte at the direct address	Direct	2	1
XRL direct, #data	XOR immediate byte into byte at the direct address	Direct	3	2

Boolean Variable manipulation Instructions

These are also called as Boolean processing instruction.

Table 4.11 MOV, CLR, CPL,SETB,ANL, and ORL Boolean Processing Instruction

Instruction	Action	Addressing	Length (bytes)	Cycles
MOV C, bit	Move bit into CF	Direct bit addressing	2	1
MOV bit, C	Move CF into the bit	Direct bit addressing	2	2
CLR C	Clear CF	PSW Register CF bit addressing	1	1
CLR bit	Clear bit	Direct bit addressing	2	1
CPL C	Complement CF	PSW Register CF bit addressing	1	1
CPL bit	Complement bit	Direct bit addressing	2	1
SETB C	Set CF=1	PSW Register CF bit addressing	1	1
SETB bit	Set bit =1	Direct bit addressing	2	1
ANL C,bit	AND between CF and bit, place the result in CF	Direct bit addressing	2	2
ANL C, $\overline{bit}$	AND between CF and , place the result in C	Direct bit addressing	2	2
ORL C,bit	OR between CF and bit, place the result in C	Direct bit addressing	2	2
ORL C, $\overline{bit}$	OR between CF and $\overline{bit}$, place the result in C	Direct bit addressing	2	2

Control Transfer Instruction

In the main program other sub programs may be called to perform a particular task. When a sub program is called the processor will jump to a new address where this program is available and

it has to accomplish program flow control transfer with help of JUMP and CALL instruction when some condition met.

Table 4.12 Delay-Cycle (NOP) instruction (No operation)

Instruction	Action	Addressing	Length in bytes	Cycles
NOP	No operation, PC gets the address of next instruction on incrementing at NOP.		1	1

Long, Absolute and Short Jump

8051 has three jump instructions: Long- it jumps to 16-bit address, Absolute- it jumps within 2 K bytes and Short- it jumps to address within 128 bytes above or below the present address.

Table 4.13 Long, absolute and short jump instructions

Instruction	Action	Addressing	Length in bytes	Cycles
LJMP addr16	Jump to the next address given by two bytes in the instruction	Direct 16 bit address	3	2
AJMP addr11	Jump to the next address	Direct 11-bit address	2	2
SJMP *rel*	Jump in the range between -128 and +127 from the address of next instruction	Direct 8-bit	2	2
JMP @A+DPTR	Jump in the next address given by addition of 8-bits of A with 16-bits of DPTR	Indirect 16-bit relative addreess		

Table 4.14 Conditional Short Relative Jumps

Instruction	Action	Addressing	Length in bytes	Cycles
JNZ rel	Jump to a relative address if a is not zero	Relative(offset)	2	2
JZ rel	Jump to a relative address if A is zero	Relative(offset)	2	2
JNC rel	Jump to a relative address if CF is not 1	Relative(offset)	2	2
JC rel	Jump to a relative address if CF=1	Relative(offset)	2	2
JB bit, rel	Jump to a relative address if addressed bit 1 (bit not set)	Relative(offset)	2	2
JNB bit,rel	Jump to a relative address if addressed bit 0 (bit not set)	Relative(offset)	2	2
JBC bit, rel	Jump to a relative address if addressed bit 1(bit set) and reset	Relative(offset)	2	2

	carry (make CF=0)			

Decrement and Conditional jump on Zero

Table 4.15 Instruction for decrement and then jump in program-loops in 8051

Instruction	Action	Addressing	Length in bytes	Cycles
DJNZ Rn, Rel	Decrement Rn and jump if Rn is still not zero.	Relative (offset)	2	2
DJNZ direct, Rel	Decrement byte at the direct and jump if byte is still not zero	Relative (offset)	2	2

Jump after comparison

Table 4.16 Compare then conditional jump after comparison

Instruction	Action	Addressing	Flag affected	Length in bytes	Cycles
CJNE A, #data, rel	Compare A and immediate data and jump if both are not equal.	Relative (offset)	C	3	2
CJNE Rn, #data, rel	Compare Rn and immediate data and jump if both are not equal.	Relative (offset	C	3	2
CJNE A, direct, rel	Compare the bytes at A and direct and jump if both are not equal	Relative (offset	C	3	2
CJNE @Ri, #data, rel	Compare byte from the address pointed by Ri and immediate data and jump if both are not equal	Relative (offset)	C	3	2

Call to a Routine

`Table 4.17 Long, absolute call and return instruction

Instruction	**Action**	**Addressing**	**Length in bytes**	**Cycles**
LCALL addr16	Call to the next address given by two bytes in the instruction	Direct 16-bit address	3	2
ACALL addr11	Call the next address given by 11 bits in	Direct 11	2	**2**

	the instruction.	bit address		
RET	Return to PC the saved PCL and PCH from the stack.	Stack address	1	2

Interrupt Control Flow (RETI instruction)

Table 4.18 RETI instruction

Instruction	Action	Addressing	Length In bytes	cycles
RETI	Return into PC the saved PCL and	Stack adddress	1	2

12. Programming

While the CPU can work only in binary, it can do so at a very high speed. however. it is quite tedious and slow for humans to deal with 0s and 1s in order to program the computer. A program that consists of 0s and 1s is called machine language. In the early days of the computer programmers coded programs in machine language. Although the hexadecimal system was used as a more efficient way to represent binary numbers, the process of working in machine code was still cumbersome for humans. Eventually, assembly language were developed which provided **mnemonics** for the machine code instructions. Plus other features which made programming faster and less prone to error. Assembly language is referred to as low level language because it deals directly with internal structure of CPU. Programmer needs assembler to convert the assembly language to machine language for execution purpose. Assembly language consists mnemonics optionally followed by one or two operands.

Programs

P1. Write an ALP (Assembly Language Program) to find the sum of values and store the result in A (lower byte and in R7 (higher byte). Assume that RAM locations 40-44 have the following values.

40=(7B), 41=(EC), 42=(C4), 43=(5B), 44=(30)

Solution:

```
        MOV     R0, #40H        ; load pointer
        MOV     R2, #05H        ; load counter
        CLR     A               ; A=0
        MOV     R7, A           ; clear R7
AGAIN:  ADD     A, @R0          ; add the byte pointer
        JNC     NEXT            ; if CY=0 it can jump to NEXT label
        INC     R7              ; increment counter
NEXT:   INC     R0              ; increment pointer
        DJNZ    R2,AGAIN        ; repeat until R2is zero
HERE:   SJMP    HERE
```

P2. Assume that 5 BCD data items are stored in RAM locations starting a 40H as shown below. Write an ALP to find the sum of all numbers. The result must be in BCD.

Solution:

```
        MOV     R0, #40H        ; load pointer
        MOV     R2, #05H        ; load counter
        CLR     A               ; A=0
        MOV     R7, A           ; clear R7
AGAIN:  ADD     A, @R0          ; add the byte pointer
        DA      A
        JNC     NEXT            ; if CY=0 it can jump to NEXT label
        INC     R7              ; increment counter
NEXT:   INC     R0              ; increment pointer
```

```
        DJNZ    R2,AGAIN        ; repeat until R2is zero
HERE:   SJMP    HERE
```

P3. Write an ALP to get hex data in the range of 00-FFH from port 1 and convert it to decimal. Save the digits in R7, R6 and R5, where the least significant digit in R7.

```
        MOV     A, #0FFH
        MOV     P1, A           ; make an P1 an input port
        MOV     A1, P1          ; read data from P1
        MOV     B, #0AH         ;  move 0AH to register b
        DIV     AB              ; divide by the contents of A by B
        MOV     R7, B           ; Save lower digit in R7 register
        MOV     B , #0AH        ;
        DIV     AB              ;
        MOV     R6, B           ; save the next digit
        MOV     R5, A           ; save the last digit
HERE:   SJMP    HERE
```

P4. Read and test P1 to see whether it has the value 45H. if it does send 99H to P2; otherwise, it stays cleared.

Solution:

```
        MOV     P2, 00H         ; clear P2
        MOV     P1, #0FFH       ; make P1 an input port
        MOV     R3, #45H        ; R3=45H
        MOV     A, P1           ; read P1
        XRL     A, R3           ;
```

```
        JNZ     EXIT
        MOV     P2, #99H
EXIT:   ……
```

P5. Find the 2's complement of the value 78 H

Solution:

```
        MOV     A, #78H         ; A=85H
        CPL     A               ; make 1's complement a
        ADD     A, #01H         ; make 2's complemt
HERE:   SJMP    HERE
```

P6. Write an ALP to determine if register A contains the value 99H, if so, make R1=FFH otherwise make R1=0.

Solution:

```
        MOV     R1, #00H        ; clear R1
        CJNE    A, #99H, NEXT   ; if A is not equal 99H then jump
        MOV     R1, #0FFH       ; make R1=FFH
NEXT    …..
```

P7. Assume that P1 is an input port connected to a temperature sensor. Write an ALP to read the temperature and test it for the value 75. According to the rest result, place the temperature value into the registers indicated by the following.

If T=75 then A=75

If T<75 then R1=T

If T>75 then R2=T

Solution:

```
        MOV     P1, # 0FFH          ; make P1 an input port
        MOV     A, P1               ; read P1 port, temperature
        CJNE    A, #75, OVER        ; jump if  A is not equal 75
        SJMP    EXIT
OVER:   JNC     NEXT                ; if CY=0, then A>75
        MOV     R1, A               ; if CY=1, A<75
        SJMP    EXIT                ; Exit
NEXT:   MOV     R2, A
EXIT    ......
```

P8. Write an ALP that finds the number of 1s in a given byte 97H.

Solution:

```
        MOV     R1, #00H            ; clear R1
        MOV     R7, 08H             ; Counter=08
        MOV     A, 97H
AGAIN:  RLC     A                   ; rotate through CY once
        JNC     NEXT                ; check for CY
        INC     R1                  ; if CY=1 then increment R1
NEXT:   DJNZ    R7, AGAIN           ; go through 8times
HERE:   SJMP    HERE
```

P9. Assume that register a has packed BCD 29H, write an ALP to convert packed BCD to ASCII numbers and place them in R2 and R6.

Solution:

```
        MOV     A, #29H         ; A=29H, packed BCD
        MOV     R2, A           ; keep a copy of BCD data in R2
        ANL     A, #0FH         ; mask the upper nibble (A=09)
        ORL     A, #30H         ; make it an ASCII, A=39H
        MOV     A, R6           ; save in R6
        MOV     A, R2           ; A=29H
        ANL     A, #0F0H        ; mask the lower nibble
        RR      A               ; rotae right
        RR      A               ; rotae right
        RR      A               ; rotate right
        RR      A               ; rotate right
        ORL     A, #30 H        ; A=32H
        MOV     R2, A           ; save the ASCII character in R2
HERE:   SJMP    HERE
```

P10. Write an ALP to create a square wave of 50% duty cycle on bit 0 of port 1.

Solution:

```
HERE:   SETB    P1.0            ; set to high bit 0of port 1
        LCALL   DELAY           ; call the delay subroutine
        CLR     P1.0            ; p1.0=0
        LCALL   DELAY
        SJMP    HERE
```

P11. Assume that the bit P2.2 is used to control the outdoor light and bit P2.5 to control the light inside the building. Write an ALP to turn on outside light and to turn the inside one.

Solution:

```
SETB    C           ; CY=1
ORL     C, P2.2     ; CY=P2.2
MOV     P2.2, C     ;  turn  it "on" if not already "on"
CLR     C           ; CY=0
ANL     C, P2.5     ; CY=P2.5 ANDed with CY
MOV     P2.5, C     ; turn it off if not already off.
```

closed loop control of servo motor :

Servo motor : A servo motor is defined as an electric motor that allows for precise control of angular (or) linear position, speed and torque. It consists of a suitable motor coupled to a sensor for position feedback and a controller that regulate the motor's movement according to a desired setpoint.

Servo motors are widely used in industrial applications such as Robotics
CNC Machinery and
Automated Manufacturing.
where high accuracy
fast response and
smooth motion are requ

A servo motor is a type of electric motor that can Rotate or Move to a specific position, speed or torque based on an i/p signal from a controller.

The term servo comes from the Latin word servus, meaning servant (or) slave. This reflects the historical use of servo motors as auxiliary drives that assist the main drive system.

However, modern servo motors are capable of providing high performance and precision as main drives in various applications.

A servo motor consists of 3 main components:

1. A motor : This can be either a DC motor (or) AC motor depending on the power source and the application requirements. The motor provides the Mechanical power to Rotate or Move the o/p shaft.

2. A sensor : This can be either a potentiometer, an encoder, a Resolver, (or) another device that measures the position, speed (or) Torque of the output shaft and sends the feedback signals to the controller.

3. A controller : This can be either an Analog (or) a Digital ckt that compares the feedback signals from the sensor with the desired setpoint signals from an external source such as

a computer (or) a joystick and generate control signals to adjust the motor's voltage (or) current accordingly. (2)

The controller uses a closed-loop feedback system to regulate the motor's movement and ensure that it matches the desired setpoint with in a certain tolerance.

- The controller can also implement various control algorithms such as proportional-integral-derivative control (PID) fuzzy logic control adaptive control, etc., to optimize the performance of the servo motor.

Servo motor work:

- The basic working principle of a servo motor involves the controller receiving two types of i/p signals:

1. A set point s/g :- This is an Analog or Digital s/g that represents the desired position, speed or Torque of the o/p shaft.
2. A feedback s/g :- This is an Analog or Digital s/g that represents the actual position, speed or torque of the o/p shaft measured by the sensor.

The controller compares these two signals and calculates an

error signal that represents the difference between them

Types of servo motors: It can be classified into different types based on their power source, construction, feedback mechanism and application.

1. AC servo motors: These are electric motors that operate on alternating current (AC). They have a stator that generate a rotating magnetic field and a rotor that follows the field.

AC servo motors can be further divided in to 2 types.

(a) Synchronous

(b) Asynchronous

* Synchronous AC servo motors have a permanent magnet rotor that rotates at the same speed as the stator field. They are more efficient precise and responsive than asynchronous motors, but they require a more complex controller and a position sensor.

* Asynchronous AC servo motors have a wound rotor that induces a current and a magnetic field that lags behind the stator field. They are simpler cheaper and more rugged than

Synchronous motors, but they have lower efficiency

Accuracy and Speed.

AC servo motors are suitable for high-power Applications that require high speed
Torque
reliability.

They are commonly used in Industrial machines
Robotics
CNC Machines etc.,

2. DC servo motors: DC servo motors are electric motors that operate on Direct current (DC). They have a permanent magnet stator that generates a fixed magnetic field and a wound rotor that rotates when a current is applied.

- DC servo motors can be further divided in to 2 types
  - (a) Brushed
  - (b) Brushless

Brushed DC servo motors have a commutator and brushes that switch the current direction in the rotor windings. They are simple
inexpensive
easy to control but they have lower efficiency
life span and
speed due to friction and wear of brushes.

Brushless DC servo motors have an electronic controller that switches the current direction in the stator windings. They are more efficient
Durable and
fast than brushed motors

but they require a more sophisticated controller and a position sensor.

DC servo motors are suitable for low-power applications that require high precision
Responsiveness and
Smooth motion.

They are commonly used in Hobby projects
toy cars
· CD/ DVD Players etc.,

3. Linear servo motors: These are electric motors that produce linear motion instead of rotary motion. They have a stationary part called a forcer(or) primary that contain coils (or) magnets, and a moving part called a platen or secondary that contains magnets (or) Iron cores.

These are divided in to two types

(a) Iron-core
(b) Iron less.

* Iron-core linear servo motors have iron cores in the platen that interact with the magnetic field of the forcer. They have high force density
stiffness
accuracy

but they also have high cogging force
weight
and Heat generation.

* Iron less linear servo motors have no iron cores in the platen, only magnets. They have low cogging force
weight and
Heat generation, but they also have a low force density
~~density~~ stiffness and
accuracy.

Linear servo motors are suitable for applications that require high speed
acceleration and
precision over long distances.

They are commonly used in Semiconductor Manufacturing,
Metrology,
Laser cutting, etc.,

Applications of Servo motors:

It have a wide range of applications in various field and Industries. some of the common applications are

1. Robotics — servo motors are used to provide precise motion and force for robotic arms
legs
Joints
grippers etc.,

They enable robots to perform tasks such as picking
placing
welding
assembling etc.,

2. CNC Machinery — servo motors are used to drive the axes of CNC Machines such as lathes
mills
Routers etc.,

They enable CNC machines to perform accurate and complex Machining operations such as
cutting
Drilling
engraving etc.,

3. Automated Manufacturing — servomotors are used to control the movements and position of various components and devices in automated Manf. systems. such as conveyors
feeders
loaders
unloaders etc.,

They enable automated manufacturing systems to achieve high productivity and Quality.

4. Medical equipment:
servomotors are used to operate various medical devices and instruments such as surgical robots
scanners
pumps
Ventilators etc.

They enable medical equipment to perform precise and safe operations and treatments.

UNIT – IV

INTRODUCTION TO TMS320LF2407 DSP CONTROLLER

Basic architectural features

physical Memory

Software Tools

Introduction to Interrupts

Interrupt Hierarchy

Interrupt control Registers

C2xx DSP CPU and Instruction set:

Introduction to code Generation

Components of the C2xx DSP core

Mapping External Devices to the C2xx code

Peripheral interface

System configuration registers

Memory

Memory Addressing Modes

Assembly programming Using the C2xx DSP - Instruction set.

UNIT 4

INTRODUCTION TO TMS320LF2407 DSP CONTROLLER

INTRODUCTION

The Texas Instruments TMS320LF2407 DSP Controller (referred to as the LF2407 in this text) is a programmable digital controller with a C2xx DSP central processing unit (CPU) as the core processor. The LF2407 contains the DSP core processor and useful peripherals integrated onto a single piece of silicon. The LF2407 combines the powerful CPU with on-chip memory and peripherals. With the DSP core and control-oriented peripherals integrated into a single chip, users can design very compact and cost-effective digital control systems.

The LF2407 DSP controller offers 40 million instructions per second (MIPS) performance. This high processing speed of the C2xx CPU allows users to compute parameters in real time rather than look up approximations from tables stored in memory. This fast performance is well suited for processing control parameters in applications such as notch filters or sensor less motor control algorithms where a large amount of calculations must be computed quickly. While the "brain" of the LF2407 DSP is the C2xx core, the LF2407 contains several control-orientated peripherals onboard (see Fig. 3.1). The peripherals on the LF2407 make virtually any digital control requirement possible. Their applications range from analog to digital conversion to pulse width modulation (PWM) generation. Communication peripherals make possible the communication with external peripherals, personal computers, or other DSP processors. Below is a brief listing of the different peripherals onboard the LF2407 followed by a graphical layout depicted in Fig. 3.1.

The LF2407 peripheral set includes:

- Two Event Managers (A and B)
- General Purpose (GP) timers
- PWM generators for digital motor control
- Analog-to-digital converter
- Controller Area Network (CAN) interface

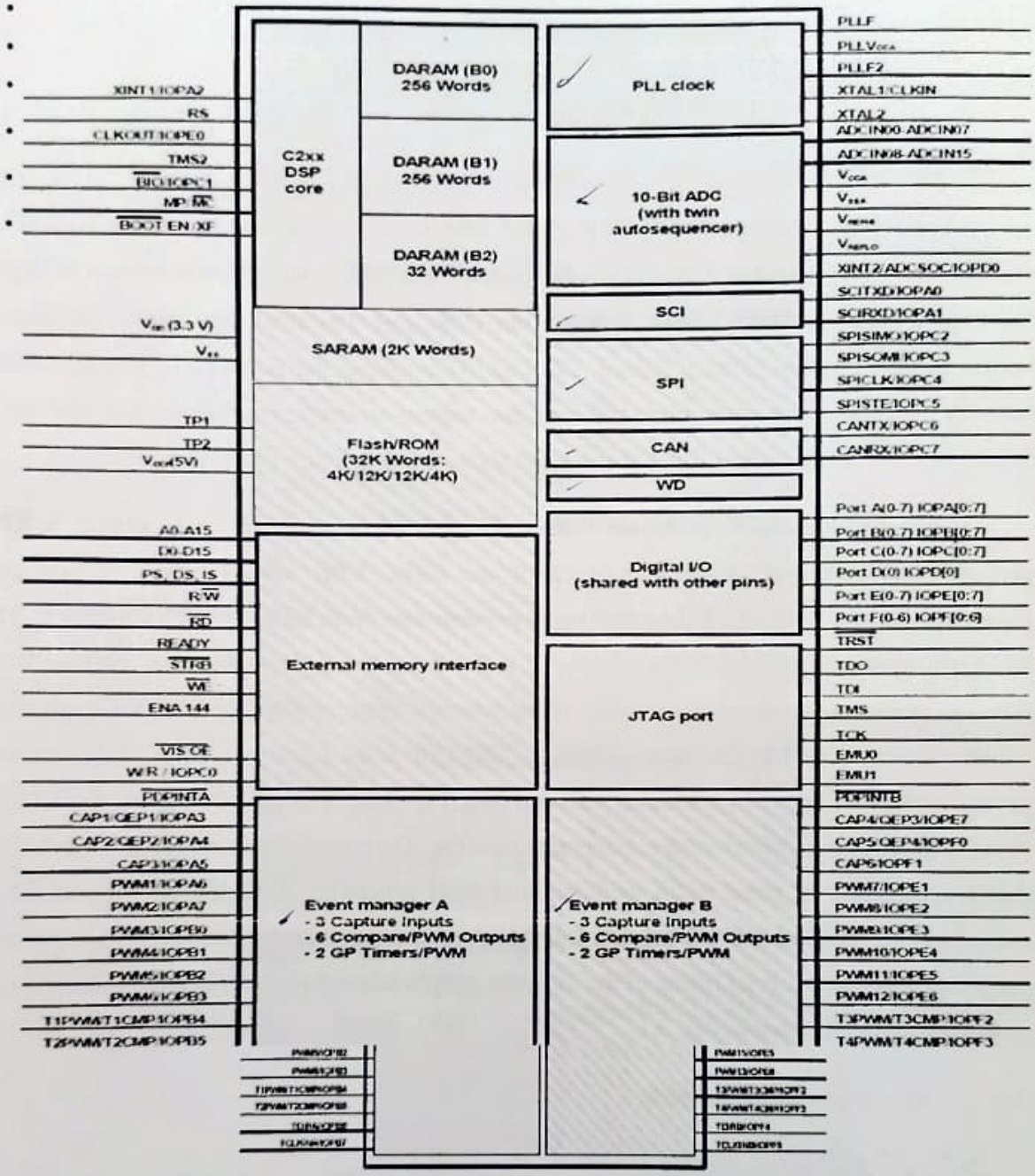


Figure 3.1 Graphical overview of DSP core and peripherals on the LF2407.

Brief Introduction to Peripherals

The following peripherals are those that are integrated onto the LF2407 chip. Refer to Fig. 1.1 to view the pin-out associated with each peripheral.

UNIT – 4

Introduction to TMS320LF2407 DSP CONTROLLER

* The Texas Instruments TMS320LF2407 DSP controller is a programmable digital controller with a C2xx DSP central processing unit (CPU) as the core processor.
* The LF2407 contains the DSP core processor and useful peripherals integrated on to a single piece of silicon, so, users can design very compact and cost effective digital controllers.
* The LF2407 combines the powerful CPU with on-chip Memory and the peripherals.
* The LF2407 DSP controller offers (40 MIPS) 40 million instructions per second performance.
* This high processing speed of the C2XX CPU allows users to compute parameters in real time rather than look up approximations from tables stored in memory.
* This fast performance is well suited for processing control parameters in applications such as notch filters

(or)

Sensorless motor control algorithms, where a large amount of calculations must be computed quickly.

The LF2407 contains several control-oriented peripherals onboard (figure). The peripherals on the LF2407 make virtually any digital control requirement possible.

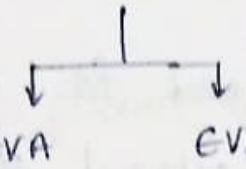
Their applications range from Analog to digital conversion to PWM (pulse width modulation) generation.

The LF2407 Peripheral set includes:

1. Two Event managers (A and B)
2. General purpose (GP) timers
3. PWM generators for digital motor control
4. Analog - to - digital converter
5. Controller Area Network (CAN) Interface
6. Serial peripheral Interface (SPI) — synchronous serial port
7. Serial comm. Interface (SCI) — Asynchronous serial port
8. General-purpose Bi-directional digital I/O (GPIO) pins
9. watchdog Timer ("Time-out" DSP reset device for system integrity).

1. Event Managers [EVA , EVB]

Two event Managers on the LF2407



* Event Manager is the most imp. peripheral in digital motor control.
* It contains the necessary functions needed to control electromechanical devices.
* Each EV is composed of functional "Blocks" including Timers
  Comparators and
  Capture units for triggering on an event
  PWM logic circuit
  Quadrature - encoder - pulse (QEP) circuit
  and Interrupt logic

2. Analog to Digital converter (ADC)

* The ADC on the LF2407 is used whenever an External Analog s/g needs to be sampled and converted to a Digital Number.
* ADC is useful in motor control applications because it allows for current sensing using a shunt resistor instead of an expensive current sensor.

3. Control Area Network (CAN) Module

It is a useful peripheral for specific applications of the LF2407.

The CAN Module is used for multi-master serial MMSC Hyss. H/w communication between external h/w.

The CAN bus has a high level of data integrity and is ideal for operation in Noisy environments such as in an Automobile

(or)

Industrial Environments

that require reliable communication and data Integrity.

4. SPI and SCI

SPI : It is a high-speed Synchronous comm. port that is mainly used for communicating between the DSP and Ext. peripherals (or) another DSP device.

Typically uses of SPI include communication with Ext. shift registers, display drivers (or) ADCs

SCI : It is an Asynchronous comm. port that supports Asynchronous serial (UART) digital comm. between CPU and other Asynchronous peripherals that use the standard NRZ (Non Return to zero) format.

It is useful in comm. between Ext. devices and the DSP.

5. Watch dog Timer (WD):

* The watch dog Timer peripheral monitors s/w & h/w operations and asserts a system reset when its internal counter overflows.
* The WD timer (when enabled) will count for a specific amount of time.
* It is necessary for the users s/w to reset the WD timer periodically, so that an unwanted reset does not occur.
* If for some reason there is a CPU disruption, the watch dog will generate a system reset.

6. GPIO pins

Since there are only a finite No. of pins available on the LF2407 device, many of the pins are multiplexed to either their primary function (or) the secondary GPIO function.

In most cases, a pins second function will be a general-purpose input/output pin.

The GPIO capability of the LF2407 is very useful as a means of controlling the functionality of pins and also provides another method to I/P (or) O/P to and from the device.

Nine 16-bit control registers control all I/O and shared pins.

There are 2 types of these registers.

(a) I/O MUX control Registers (MCRx)
— used to control the multiplexer selection that chooses b/w the primary function of a pin (or) the GPIO function.

(b) Data and Direction control Registers (Px DAT DIR)
— used to control the data and data Direction of Bi-directional I/O pins.

7. PLL clock Module:
It is basically an i/p clock multiplier that allows the user to control the i/p clocking frequency to the DSP core.
External to the LF2407, a clock reference (can oscillator | crystal) is generated.
This s/g is fed in to the LF2407 and is multiplied (or) divided by the PLL.
This new (Higher or lower freq.) clock s/g is then used to clock the DSP core.
The LF2407; PLL allows the user to select a multiplication factor ranging from 0.5x to 0.4x that of the Ext. clock s/g.
The Default value of the PLL is 4x.

8. Memory Allocation spaces:

The LF2407 DSP controller has 3 diff. allocations of memory it can uses:

Data
program and
I/O memory space.

Data space is used for program calculations
look-up tables and
Any other memory used by an Algorithm

Data memory can be in the form of the on-chip random access memory (RAM) or Ext. RAM.

Program memory is the location of users programe code. Program memory on the LF2407 is either mapped to the off-chip RAM (MP/MC -pin = 1)

(or)

to the ON-chip flash memory (MP/MC -pin = 0), depending on the logic values of the MP/MC-pin.

I/O space is not Really memory, but a virtual memory address used to o/p data to Peripherals external to the LF2407.

① TMS320Lf2407 DSP controller

↓

Programmable digital controller, with C2xx DSP CPU as core processor

② Lf2407

↓

contains
- DSP core processor (C2xx)
- useful peripherals Integrated on to single piece of silicon

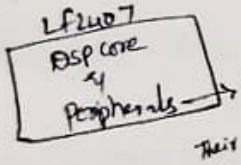
so, users design
- compact (Real time)
- cost effective Digital control sys.

combines
- powerful (C2xx) CPU with on-chip memory
- Peripherals.

↓

offers — 40 MIPS performance. (High Processing speed C2xx CPU) and allows users to design in real time.

Graphical overview:



→ make virtually any digital (PCB) control requirements possible

Their Applications Range from ~~detect~~ ADC to PWM generation

③ Lf2407 includes:

① 2 event Managers (EVA, EVB (.. same))
- 2 GP Timers / PWM
- 3 Capture i/ps
- 6 compare / PWM o/ps

⇒ not Imp. peripheral in Digital motor control. contains receiving functions to control E Mech. Device.

Each EV

② (a) 2 event Managers

(b) General purpose Timers

(c) PWM generator for Digital motor control

(d) ADC → ex. Analog sig sampled and converted in to digital No.

(e) CAN interface → MMSC b/w ext. H.W

(f) SPI — Synchronous serial port — used for comm. b/w DSP and ext. peripheral

(g) SCI — Asy. " — used for Digital comm. b/w CPU & Asyn. peripheral.

(h) GPIO pins → used of controlling the functionality of pins ; 2 Regs
- IOMCR
- DATA & Direction control Reg.

(i) watch dog Timer → it monitors s/w & H/w operations

PLL clock → It is basically an i/p clk multiplies that allows user to control the i/p clk freq to the DSP core.

Memory Allocation Spaces
- 3 diff. allocations in Lf2407.

Event Managers (EVA, EVB)

There are two Event Managers on the LF2407, the EVA and EVB. The Event Manager is the most important peripheral in digital motor control. It contains the necessary functions needed to control electromechanical devices. Each EV is composed of functional "blocks" including timers, comparators, and capture units for triggering on an event, PWM logic circuits, quadrature-encoder–pulse (QEP) circuits, and interrupt logic.

The Analog-to-Digital Converter (ADC)

The ADC on the LF2407 is used whenever an external analog signal needs to be sampled and converted to a digital number. Examples of ADC applications range from sampling a control signal for use in a digital notch filtering algorithm or using the ADC in a control feedback loop to monitor motor performance. Additionally, the ADC is useful in motor control applications because it allows for current sensing using a shunt resistor instead of an expensive current sensor.

The Control Area Network (CAN) Module

While the CAN module will not be covered in this text, it is a useful peripheral for specific applications of the LF2407. The CAN module is used for multi-master serial communication between external hardware. The CAN bus has a high level of data integrity and is ideal for operation in noisy environments such as in an automobile, or industrial environments that require reliable communication and data integrity.

Serial Peripheral Interface (SPI) and Serial Communications Interface (SCI)

The SPI is a high-speed synchronous communication port that is mainly used for communicating between the DSP and external peripherals or another DSP device. Typical uses of the SPI include communication with external shift registers, display drivers, or ADCs.

The SCI is an asynchronous communication port that supports asynchronous serial (UART) digital communication between the CPU and other asynchronous peripherals that use the standard NRZ (non-return-to-zero) format. It is useful in communication between external devices and the DSP. Since these communication peripherals are not directly related to motion control applications, they will not be discussed further in this text.

Watchdog Timer (WD)

The Watchdog timer (WD) peripheral monitors software and hardware operations and asserts a system reset when its internal counter overflows. The WD timer (when enabled) will count for a specific amount of time. It is necessary for the user's software to reset the WD timer periodically so that an unwanted reset does not occur. If for some reason there is a CPU disruption, the watchdog will generate a system reset. For example, if the software enters an endless loop or if the CPU becomes temporarily disrupted, the WD timer will overflow and a DSP reset will occur, which will cause the DSP program to branch to its initial starting point. Most error conditions that temporarily disrupt chip operation and inhibit proper CPU function can be cleared by the WD function. In this way, the WD increases the reliability of the CPU, thus ensuring system integrity.

General Purpose Bi-Directional Digital I/O (GPIO) Pins

Since there are only a finite number of pins available on the LF2407 device, many of the pins are multiplexed to either their primary function or the secondary GPIO function. In most cases, a pin's second function will be as a general-purpose input/output pin. The GPIO capability of the LF2407 is very useful as a means of controlling the functionality of pins and also provides another method to input or output data to and from the device. Nine 16-bit control registers control all I/O and shared pins. There are two types of these registers:

- I/O MUX Control Registers (MCRx) – Used to control the multiplexer selection that Chooses between the primary function of a pin or the general-purpose I/O function.
- Data and Direction Control Registers (PxDATDIR) – Used to control the data and data Direction of bi-directional I/O pins.

Phase Locked Loop (PLL) Clock Module

The phase locked loop (PLL) module is basically an input clock multiplier that allows the user to control the input clocking frequency to the DSP core. External to the LF2407, a clock reference (can oscillator/crystal) is generated. This signal is fed into the LF2407 and is multiplied or divided by the PLL. This new (higher or lower frequency) clock signal is then used to clock the DSP core. The LF2407's PLL allows the user to select a multiplication factor ranging from 0.5X to 4X that of the external clock signal. The default value of the PLL is 4X.

Memory Allocation Spaces

The LF2407 DSP Controller has three different allocations of memory it can use: Data, Program, and I/O memory space. Data space is used for program calculations, look-up tables, and any other memory used by an algorithm. Data memory can be in the form of the on-chip random access memory (RAM) or external RAM. Program memory is the location of user's program code. Program memory on the LF2407 is either mapped to the off-chip RAM (MP/MC- pin =1) or to the on-chip flash memory (MP/MC- = 0), depending on the logic value of the MP/MC-pin.

I/O space is not really memory but a virtual memory address used to output data to peripherals external to the LF2407. For example, the digital-to-analog converter (DAC) on the Spectrum Digital™ evaluation module is accessed with I/O memory. If one desires to output data to the DAC, the data is simply sent to the configured address of I/O space with the "OUT" command. This process is similar to writing to data memory except that the OUT command is used and the data is copied to and outputted on the DAC instead of being stored in memory.

Types of Physical Memory

Random Access Memory(RAM)

The LF2407 has 544 words of 16 bits each in the on-chip DARAM. These 544 words are partitioned into three blocks: B0, B1, and B2. Blocks B1 and B2 are allocated for use only as data memory. Memory block B0 is different than B1 and B2. This memory block is normally configured as Data Memory, and hence primarily used to hold data, but in the case of the B0 block, it can also be configured as Program Memory. B0 memory can be configured as program or data memory depending on the value of the core level "CNF" bit.

- (CNF=0) maps B0 to data memory.
- (CNF=1) maps B0 to program memory.

The LF2407 also has 2K of single-access RAM (SARAM). The addresses associated with the SARAM can be used for both data memory and program memory, and are software configurable to the internal SARAM or external memory.

Non-Volatile Flash Memory

The LF2407 contains 32K of on-chip flash memory that can be mapped to program space if the MP/MC-pin is made logic 0 (tied to ground). The flash memory provides a permanent location to store code that is unaffected by cutting power to the device. The flash memory can be electronically programmed and erased many times to allow for code development. Usually, the external RAM on the LF2407 Evaluation Module (EVM) board is used instead of the flash for code development due to the fact that a separate "flash programming" routine must be performed to flash code into the flash memory. The on-chip flash is normally used in situations where the DSP program needs to be tested where a JTAG connection is not practical or where the DSP needs to be tested as a "stand-alone" device. For example, if a LF2407 was used to develop a DSP control solution to an automobile braking system, it would be somewhat impractical to have a DSP/JTAG/PC interface in a car that is undergoing performance testing.

Software Tools

- Texas Instruments Code Composer Studio (CCS) is a user friendly windows based debugger for developing and debugging software for LF2407.

- CCS allows user to write and debug code in C or assembly language.

Features

- User friendly windows environment
- Ability to use code written in C and Assembly language
- Memory displays and on the fly editing capability
- Disassembly window for debugging
- Source level debugging (allows stepping through and setting break points in original source code)
- CPU register visibility and modification
- Real time debugging
- Various single step by step, over/ step into command icons

- Ability to display data in graph formats
- General Extension Language capability (GEL) allows the user to create functions that extend the usefulness of CCS.

Introduction to the C2xx DSP Core and Code Generation

The heart of the LF2407 DSP Controller is the C2xx DSP core. This core is a 16-bit fixed point processor, meaning that it works with 16-bit binary numbers. One can think of the C2xx as the central processor in a personal computer. The LF2407 DSP consists of the C2xx DSP core plus many peripherals such as Event Managers, ADC, etc., all integrated onto one single chip.

The Components of the C2xx DSP Core

The DSP core (like all microprocessors) consists of several subcomponents necessary to perform arithmetic operations on 16-bit binary numbers. The following is a list of the multiple subcomponents found in the C2xx core which we will discuss further:

- A 32-bit central arithmetic logic unit (CALU)
- A 32-bit accumulator (used frequently in programs)
- Input and output data-scaling shifters for the CALU
- A (16-bit by 16-bit) multiplier
- A product-scaling shifter
- Eight auxiliary registers (AR0 – AR7) and an auxiliary register arithmetic unit (ARAU)

Each of the above components is either accessed directly by the user code or is indirectly used during the execution of an assembly command.

Central Arithmetic Logic Unit (CALU)

The C2xx performs 2s-complement arithmetic using the 32-bit CALU. The CALU uses 16-bit words taken from data memory, derived from an immediate instruction, or from the 32-bit multiplier result. In addition to arithmetic operations, the CALU can perform Boolean operations. The CALU is somewhat transparent to the user. For example, if an arithmetic command is used, the user only needs to write the command and later read the output from the appropriate register. In this sense, the CALU is "transparent" in that it is not accessed directly by the user.

Accumulator

The accumulator stores the output from the CALU and also serves as another input to the CALU (many arithmetic commands perform operations on numbers that are currently stored in the accumulator; versus other memory locations). The accumulator is 32 bits wide and is divided into two sections, each consisting of 16 bits. The high-order bits consist of bits 31 through 16, and the low-order bits are made up of bits 15 through 0. Assembly language instructions are provided for storing the high- and low-order accumulator words to data memory. In most cases, the accumulator is written to and read from directly by the user code via assembly commands. In some instances, the accumulator is also transparent to the user (similar to the CALU operation in that it is accessed "behind the scenes").

Scaling Shifters

The C2xx has three 32-bit shifters that allow for scaling, bit extraction, extended arithmetic, and overflow-prevention operations. The scaling shifters make possible commands that shift data left or right. Like the CALU, the operation of the scaling shifters is "transparent" to the user. For example, the user needs only to use a shift command, and observe the result. Any one of the three shifters could be used by the C2xx depending on the specific instruction entered. The following is a description of the three shifters:

- Input data-scaling shifter (input shifter): This shifter left-shifts 16-bit input data by 0 to 16 bits to align the data to the 32-bit input of the CALU. For example, when the user uses a command such as "ADD 300h, 5", the input shifter is responsible for first shifting the data in memory address "300h" to the left by five places before it is added to the contents of the accumulator.

- Output data-scaling shifter (output shifter): This shifter left-shifts data from the accumulator by 0 to 7 bits before the output is stored to data memory. The content of the accumulator remains unchanged. For example, when the user uses a command such as "SACL 300h, 4", the output shifter is responsible for first shifting the contents of the accumulator to the left by four places before it is stored to the memory address "300h".
- Product-scaling shifter (product shifter): The product register (PREG) receives the output of the multiplier. The product shifter shifts the output of the PREG before that output is sent to the input of the CALU. The product shifter has four product shift modes (no shift, left shift by one bit, left shift by four bits, and right shift by six bits), which are useful for performing multiply/accumulate operations, fractional arithmetic, or justifying fractional products.

Multiplier

The multiplier performs 16-bit, 2s-complement multiplication and creates a 32-bit result. In conjunction with the multiplier, the C2xx uses the 16-bit temporary register (TREG) and the 32-bit product register (PREG).

The operation of the multiplier is not as "transparent" as the CALU or shifters. The TREG always needs to be loaded with one of the numbers that are to be multiplied. Other than this prerequisite, the multiplication commands do not require any more actions from the user code. The output of the multiply is stored in the PREG, which can later be read by the user code.

Auxiliary Register Arithmetic Unit (ARAU) and Auxiliary Registers

The ARAU generates data memory addresses when an instruction uses indirect addressing to access data memory (more on indirect addressing will be covered later along with assembly programming). Eight auxiliary registers (AR0 through AR7) support the ARAU, each of which can be loaded with a 16-bit value from data memory or directly from an instruction. Each auxiliary register value can also be stored in data memory. The auxiliary registers are mainly used as "pointers" to data memory locations to more easily facilitate looping or repeating algorithms. They are directly written to by the user code and are automatically incremented or decremented by particular assembly instructions during a looping or repeating operation. The auxiliary register pointer (ARP) embedded in status register ST0 references the auxiliary register. The status registers (ST0, ST1) are core level registers where values such as the Data Page (DP) and ARP located.

Mapping External Devices to the C2xx Core and the Peripheral Interface

LF2407 contains many peripherals that need to be accessed by c2xx core.

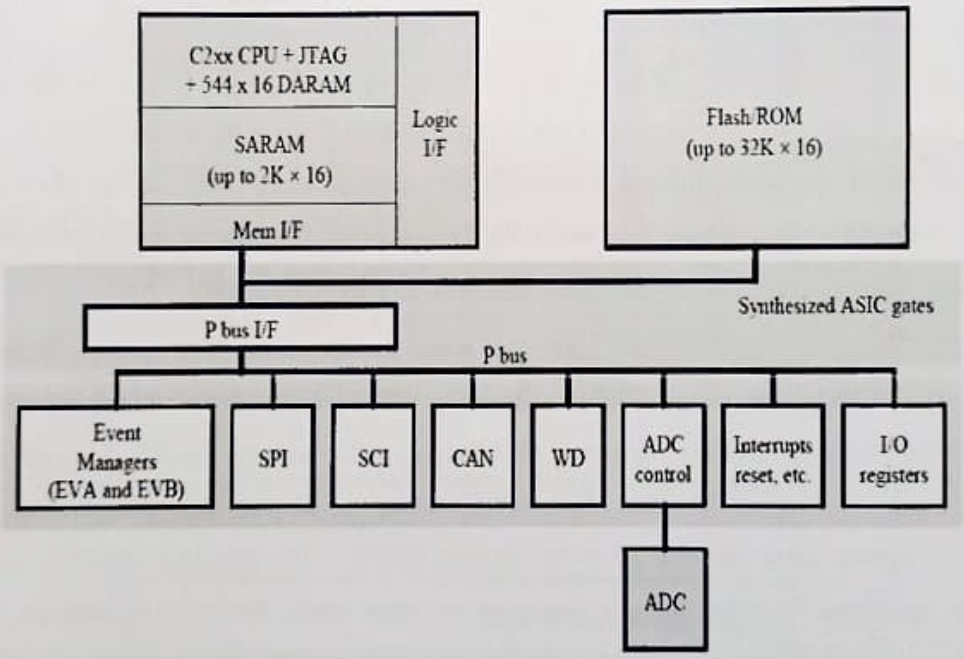
Each peripheral is mapped to a corresponding block of data memory addresses.

Each corresponding block contains configuration registers, input registers, output registers and status registers.

Each peripheral is accessed by simply writing to the appropriate registers in data memory provided the peripheral clock is enabled.

Functional Block Diagram of LF2407 DSP Controller

- All on chip peripherals are accessed through peripheral bus (PBUS).



System Configuration Registers

The System Control and Status Registers (SCSR1, SCSR2) are used to configure or display fundamental settings of the LF2407. For example, these fundamental settings include the clock speed (clock pre-scale setting) of the LF2407, which peripherals are enabled, microprocessor/microcontroller mode, etc. Bits are controlled by writing to the corresponding data memory address or the logic level on an external pin as with the microprocessor/microcontroller (MP/MC) select bit. The bit descriptions of these two registers (mapped to data memory) are listed below.

System Control and Status Register 1 (SCSR1) — Address 07018h

15	14	13	12	11	10	9	8
Reserved	CLKSRC	LPM1	LPM0	CLK PS2	CLK PS1	CLK PS0	Reserved
R–0	RW–0	RW–0	RW–0	RW–1	RW–1	RW–1	R–0
7	6	5	4	3	2	1	0
ADC CLKEN	SCI CLKEN	SPI CLKEN	CAN CLKEN	EVB CLKEN	EVA CLKEN	Reserved	ILLADR
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	R–0	RC–0

Note: R = read access, W = write access, C = clear, -0 = value after reset.

Bit 15 Reserved

Bit 14 CLKSRC. CLKOUT pin source select

0 CLKOUT pin has CPU Clock (40 MHz on a 40-MHz device) as the output

1 CLKOUT pin has Watchdog clock as the output

Bits 13–12 LPM (1:0). Low-power mode select

These bits indicate which low-power mode is entered when the CPU executes the IDLE instruction. Description of the low-power modes:

LPM(1:0)	Low-Power mode selected
00	IDLE1 (LPM0)
01	IDLE2. (LPM1)
1x	HALT (LPM2)

Bits 11–9

PLL Clock prescale select. These bits select the PLL multiplication factor for the input clock.

CLK PS2	CLK PS1	CLK PS0	System Clock Frequency
0	0	0	$4 \times F_{in}$
0	0	1	$2 \times F_{in}$
0	1	0	$1.33 \times F_{in}$
0	1	1	$1 \times F_{in}$
1	0	0	$0.8 \times F_{in}$
1	0	1	$0.66 \times F_{in}$
1	1	0	$0.57 \times F_{in}$
1	1	1	$0.5 \times F_{in}$

Note: F_{in} is the input clock frequency.

Bit 8 Reserved

Bit 7 ADC CLKEN. ADC module clock enable control bit.

0 Clock to module is disabled (i.e., shut down to conserve power).

1 Clock to module is enabled and running normally.

Bit 6 SCI CLKEN. SCI module clock enable control bit.

0 Clock to module is disabled (i.e., shut down to conserve power).

1 Clock to module is enabled and running normally.

Bit 5 SPI CLKEN. SPI module clock enable control bit

0 Clock to module is disabled (i.e., shut down to conserve power)

1 Clock to module is enabled and running normally

Bit 4 CAN CLKEN. CAN module clock enable control bit

0 Clock to module is disabled (i.e., shut down to conserve power)

1 Clock to module is enabled and running normally

Bit 3 EVB CLKEN. EVB module clock enable control bit

0 Clock to module is disabled (i.e., shut down to conserve power)

1 Clock to module is enabled and running normally

Bit 2 EVA CLKEN. EVA module clock enable control bit

0 Clock to module is disabled (i.e., shut down to conserve power)

1 Clock to module is enabled and running normally

Note: In order to modify/read the register contents of any peripheral, the clock to that peripheral must be enabled by writing a 1 to the appropriate bit.

Bit 1 Reserved

Bit 0 ILLADR. Illegal Address detect bit

If an illegal address has occurred, this bit will be set. It is up to software to clear this bit following an illegal address detects. This bit is cleared by writing a 1 to it and should be cleared as part of the initialization sequence. Note: An illegal address will cause a Non-Mask able Interrupt (NMI).

System Control and Status Register 2 (SCSR2) — Address 07019h

15-8							
Reserved							
RW–0							
7	6	5	4	3	2	1	0
Reserved	I/P QUAL	WD OVERRIDE	XMIF HI–Z	BOOT EN	MP/MC	DON	PON
	RW–0	RC–1	RW–0	RW–BOOT EN pin	RW–MP/MC pin	RW–1	RW–1

Note: *R = read access, W = write access, C = clear, -0 = value after reset.*

Bits 15–7 Reserved. Writes have no effect; reads are undefined

Bit 6 Input Qualifier Clocks.

An input-qualifier circuitry qualifies the input signal to the CAP1–6, XINT1/2, ADCSOC, and PDPINTA/B pins in the 240xA devices. The I/O functions of these pins do not use the input-qualifier circuitry. The state of the internal input signal will change only after the pin is held high/low for 6 (or 12) clock edges. This ensures that a glitch smaller than (or equal to) 5 (or 11) CLKOUT cycles wide will not change the internal pin input state. The user must hold the pin high/low for 6 (or 12) cycles to ensure that the device will see the level change. This bit determines the width of the glitches (in number of internal clock cycles) that will be blocked. Note that the internal clock is not the same as CLKOUT, although its frequency is the same as CLKOUT.

0 The input-qualifier circuitry blocks glitches up to 5 clock cycles long

1 The input-qualifier circuitry blocks glitches up to 11 clock cycles long

Note: This bit is applicable only for the 240xA devices, not for the 240x devices because they lack an input-qualifier circuitry.

Bit 5 Watchdog Override. (WD protect bit)

After RESET, this bit gives the user the ability to disable the WD function through software (by setting the WDDIS bit = 1 in the WDCR). This bit is a clear-only bit and defaults to a 1 after reset.

Note: This bit is cleared by writing a 1 to it.

0 Protects the WD from being disabled by software. This bit cannot be set to 1 by software. It is a clear-only bit, cleared by writing a 1.

1 This is the default reset value and allows the user to disable the WD through the WDDIS bit in the WDCR. Once cleared, however, this bit can no longer be set to 1 by software, thereby protecting the integrity of the WD timer.

Bit 4 XMIF Hi-Z Control

This bit controls the state of the external memory interface (XMIF) signals.

0 XMIF signals in normal driven mode; i.e., not Hi-Z (high impedance).

1 All XMIF signals are forced to Hi-Z state.

Bit 3 Boot Enable

This bit reflects the state of the BOOT_EN / XF pin at the time of reset. After reset and device has "booted up", this bit can be changed in software to re-enable Flash memory visibility or return to active Boot ROM.

0 Enable Boot ROM — Address space 0000 — 00FF is now occupied by the on-chip Boot ROM Block. Flash memory is totally disabled in this mode. Note: There is no on-chip boot ROM in ROM devices (i.e., LC240xA)

1 Disable Boot ROM — Program address space 0000 — 7FFF is mapped to on-chip Flash memory in the case of LF2407A and LF2406A. In the case of LF2402A, addresses 0000 – 1FFF are mapped

Bit 2 Microprocessor/Microcontroller Select

This bit reflects the state of the MP/MC pin at time of reset. After reset, this bit can be changed in software to allow dynamic mapping of memory on and off chip.

0 Set to Microcontroller mode — Program Address range 0000 — 7FFF is mapped internally (i.e., Flash)

1 Set to Microprocessor mode — Program Address range 0000 — 7FFF is mapped externally (i.e., customer provides external memory device.)

Bits 1–0 SARAM Program/Data Space Select

DON PON SARAM status

0	0	SARAM not mapped (disabled), address space allocated to external memory
0	1	SARAM mapped internally to Program space
1	0	SARAM mapped internally to Data space
1	1	SARAM block mapped internally to both Data and Program spaces.

This is the default or reset value

Memory Addressing Modes

There are three basic memory addressing modes used by the C2xx instruction set. The three modes are:

- Immediate addressing mode (does not actually access memory)
- Direct addressing mode
- Indirect addressing mode

Immediate Addressing Mode

In the immediate addressing mode, the instruction contains a constant to be manipulated by the instruction. Even though the name "immediate addressing" suggests that a memory location is accessed, immediate addressing is simply dealing with a user-specified constant which is usually included in the assembly command syntax. The "#" sign indicates that the value is an immediate address (just a constant). The two types of immediate addressing modes are:

Short-immediate addressing: The instructions that use short-immediate addressing have an 8-bit, 9-bit, or 13-bit constant as the operand.

For example, the instruction:

```
LACL #44h          ; loads lower bits of accumulator with
                   ; Eight-bit constant (44h in this case)
```

Note: The LACL command will work only with a short 8-bit constant. If you want to load a long 16-bit constant, then use the LACC command.

Long-immediate addressing: Instructions that use long-immediate addressing have a 16-bit constant as an operand. This 16-bit value can be used as an absolute constant or as a 2s-complement value.

For example, the instruction:

```
LACC #4444h        ; loads accumulator with up to a 16-bit
```

; Constant (4444h in this case)

If you need to use registers or access locations in data memory, you must use either direct or indirect addressing.

Direct Addressing Mode

In direct addressing, data memory is first addressed in blocks of 128 words called data pages. The entire 64K of data memory consists of 512 DPs labeled 0 through 511, as shown in the Fig. 3.2. The current DP is determined by the value in the 9-bit DP pointer in status register ST0. For example, if the DP value is "0 0000 0000", the current DP is 0. If the DP value is "0 0000 0010", the current data page is 2. The DP of a particular memory address can be found easily by dividing the address (in hexadecimal) by 80h.

For example: For the data memory address 0300h, 300h/80h = 6h so the DP pointer is 6h. Likewise, the DP pointer for 200h is 4h.

DP Value	Offset	Data Memory
0000 0000 0	000 0000	Page 0: 0000h-007Fh
0000 0000 0	111 1111	
0000 0000 1	000 0000	Page 1: 0080h-00FFh
0000 0000 1	111 1111	
0000 0001 0	000 0000	Page 2: 0100h-017Fh
0000 0001 0	111 1111	
1111 1111 1	000 0000	Page 511: FF80h-FFFFh
1111 1111 1	111 1111	

Figure 3.2 Data pages and corresponding memory ranges.

In addition to the DP, the DSP must know the particular word being referenced on that page. This is determined by a 7-bit offset. The 7-bit offset is simply the 7 least significant bits (LSBs) of the memory address. The DP and the offset make up the 16-bit memory address (see Fig. 3.3).

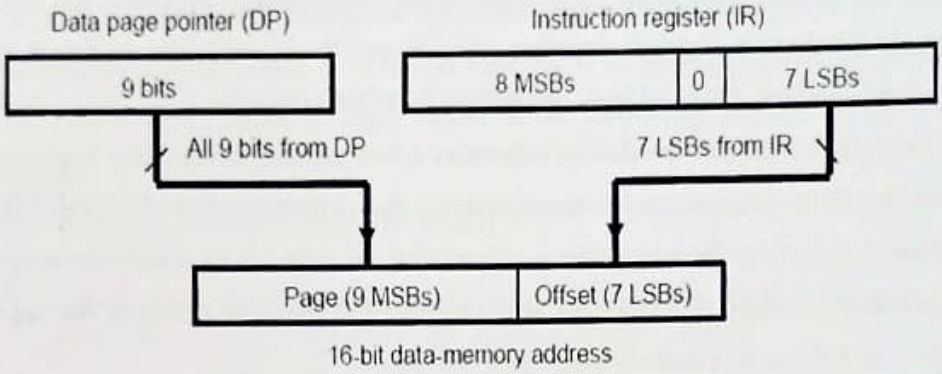


Figure 3.3 Data page and offset make up a 16-bit memory address.

When you use direct addressing, the processor uses the 9 DP bits and the 7 LSBs of the instruction to obtain the true memory address. The following steps should be followed when using direct addressing:

1 Set the DP. Load the appropriate value (from 0 to 511 in decimal or 0-1FF in hex) into the DP. The easiest way to do this is with the LDP instruction. The LDP instruction loads the DP directly to the ST0 register without affecting any other bits of the ST0.

```
LDP #0E1h          ; sets the data page pointer to E1h
                          Or
LDP #225           ; sets the data page pointer to 225 decimal
                   ; Which is E1 in hexadecimal
```

2 Specify the offset. For example, if you want the ADD instruction to use the value at the second address of the current data page, you would write: ADD 1h

If the data page points to 300h, then the above instruction will add the contents of 301h to the accumulator

Note: You do not have to set the data page prior to every instruction that uses direct addressing. If all the instructions in a block of code access the same data page, you can simply load the DP before the block. However, if various data pages are being accessed throughout the block of code be sure the DP is changed accordingly.

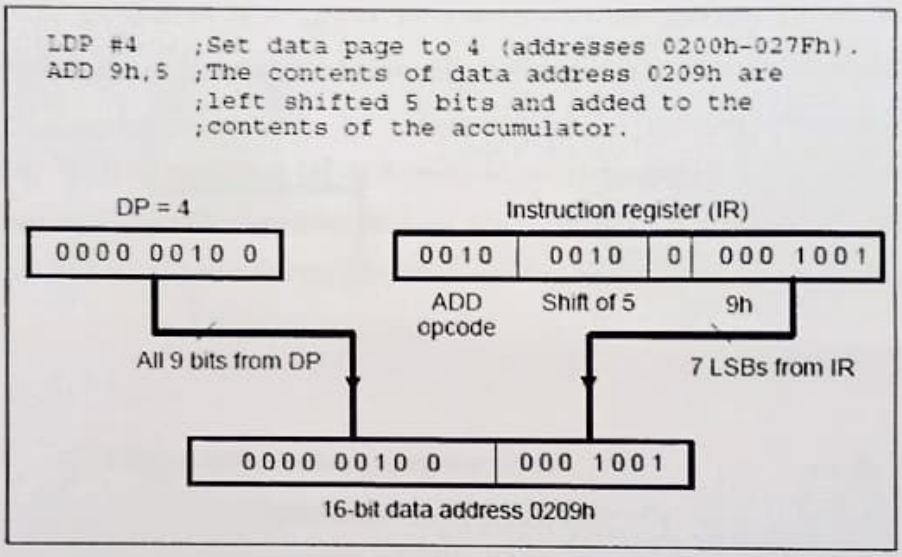
Examples of Direct Addressing

In Example 1, the first instruction loads the DP with $0\ 0000\ 0100_2$ to set the current data page to 4. The ADD instruction then references a data memory address that is generated as

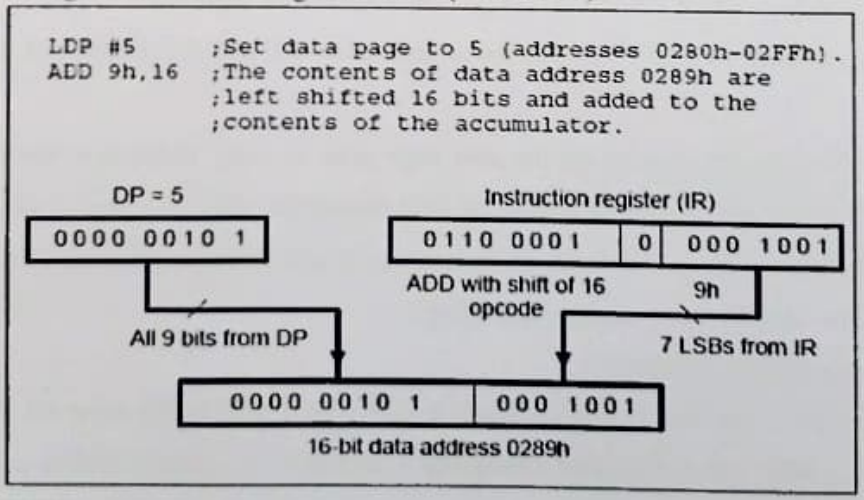
shown following the program code. Before the ADD instruction is executed, the opcode is loaded into the instruction register. Together, the DP and the seven LSBs of the instruction register form the complete 16-bit address, 0000 0010 0000 1001_2 (0209h).

In Example 2, the ADD instruction references a data memory address that is generated as shown following the program code. For any instruction that performs a shift of 16, the shift value is not embedded directly in the instruction word; instead, all eight MSBs contain an opcode that not only indicates the instruction type, but also a shift of 16. The eight MSBs of the instruction word indicate an ADD with a shift of 16.

Example 1 Using Direct Addressing with ADD (Shift of 0 to 15)



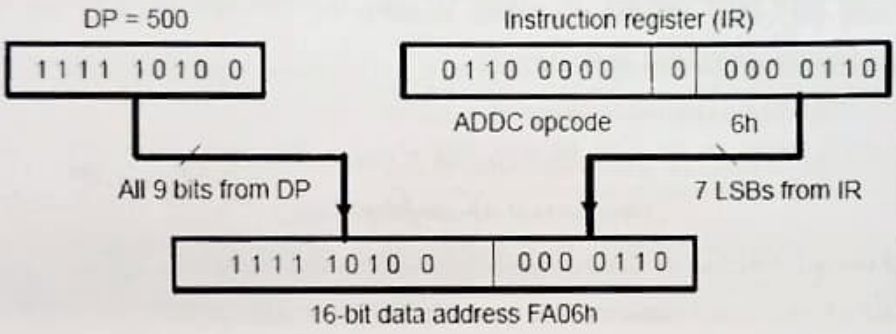
Example 2 Using Direct Addressing with ADD (Shift of 16)



In Example 3, the ADDC instruction references a data memory address that is generated as shown following the program code. You should note that if an instruction does not perform shifts (such as the ADDC instruction), all eight MSBs of the instruction contain the opcode for the instruction type.

Example 3 Using Direct Addressing with ADDC

```
LDP #500  ;Set data page to 500 (addresses FA00h-FA7Fh).
ADDC 6h   ;The contents of data address FA06h
          ;and the value of the carry bit (C) are
          ;added to the contents of the accumulator.
```



Indirect Addressing Mode

Indirect addressing is a powerful way of addressing data memory. Indirect addressing mode is not dependent on the current data page as is direct addressing. Instead, when using indirect addressing you load the memory space that you would like to access into one of the auxiliary registers (ARx). The current auxiliary register acts as a pointer that points to a specific memory address.

The register pointed to by the ARP is referred to as the current auxiliary register or current AR. To select a specific auxiliary register, load the 3-bit auxiliary register pointer (ARP) with a value from 0 to 7. The ARP can be loaded with the MAR instruction or by the LARP instruction. An ARP value can also be loaded by using the ARx operand after any instruction that supports indirect addressing as seen below.

Example of using MAR:

```
ADD *, AR1          ; Adds using current *, then makes AR1 the
                    ; New current AR for future uses
```

Example of using LARP

```
LARP #2             ; this will make AR2 the current AR
```

The C2xx provides four types of indirect addressing options:

- No increment or decrement. The instruction uses the content of the current auxiliary register as the data memory addresses but neither increments nor decrements the content of the current auxiliary register.
- Increment or decrement by 1. The instruction uses the content of the current auxiliary register as the data memory address and then increments or decrements the content of the current auxiliary register by one.
- Increment or decrement by an index amount. The value in AR0 is the index amount. The instruction uses the content of the current auxiliary register as the data memory address and then increments or decrements the content of the current auxiliary register by the index amount.
- Increment or decrement by an index amount using reverse carry. The value in AR0 i the index amount. After the instruction uses the content of the current auxiliary register as the data memory address, that content is incremented or decremented by the index

amount. The addition and subtraction process is accomplished with the carry propagation reversed and is useful in fast Fourier transforms algorithms.

Table 3.1 displays the various operands that are available for use with instructions while using indirect addressing mode.

Operand	Option	Example
*	No increment or decrement	LT * loads the temporary register TREG with the content of the data memory address referenced by the current AR.
*+	Increment by 1	LT *+ loads the TREG with the content of the data memory address referenced by the current AR and then adds 1 to the content of the current AR.
*-	Decrement by 1	LT *- loads the TREG with the content of the data memory address referenced by the current AR and then subtracts 1 from the content of the current AR.
*0+	Increment by index amount	LT *0+ loads the TREG with the content of the data memory address referenced by the current AR and then adds the content of AR0 to the content of the current AR.
*0-	Decrement by index amount	LT *0- loads the TREG with the content of the data memory address referenced by the current AR and then subtracts the content of AR0 from the content of the current AR.
*BR0+	Increment by index amount, adding with reverse carry	LT *BR0+ loads the TREG with the content of the data memory address referenced by the current AR and then adds the content of AR0 to the content of the current AR, adding with reverse carry propagation.
*BR0-	Decrement by index amount, subtracting with reverse carry	LT *BR0- loads the TREG with the content of the data memory address referenced by the current AR and then subtracts the content of AR0 from the content of the current AR, subtracting with bit reverse carry propagation.

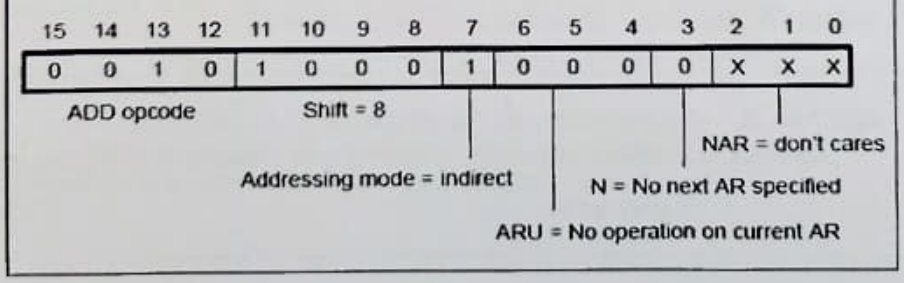
Table 3.1 Indirect addressing operands.

Examples of Indirect Addressing

Example 1 illustrates how the instruction register is loaded with the value shown when the ADD instruction is fetched from program memory.

Example 1. Indirect Addressing—No Increment or Decrement

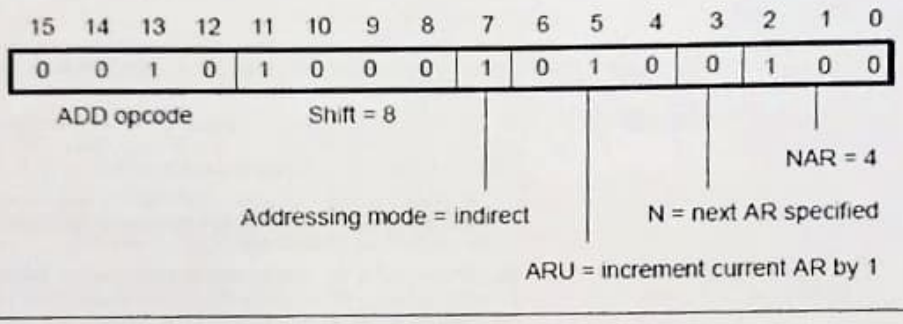
```
ADD *,8       ;Add to the accumulator the content of the
              ;data-memory address referenced by the
              ;current auxiliary register. The data
              ;is left shifted 8 bits before being added.
```



Example 2, illustrates how the instruction register is loaded with the value shown when the ADD instruction is fetched from program memory.

Example 2. Indirect Addressing—Increment by 1

```
ADD *+,8,AR4    ;Operates as in Example 1   , but
                ;in addition, the current auxiliary
                ;register is incremented by one, and
                ;AR4 is chosen as the next auxiliary
                ;register.
```



Example 3. Indirect Addressing—Decrement by 1

```
ADD *-,8        ;Operates as in Example 1   , but in
                ;addition, the current auxiliary register
                ;is decremented by one.
```

Example 4. Indirect Addressing—Increment by Index Amount

```
ADD *0+,8       ;Operates as in Example 1   , but in
                ;addition, the content of register AR0
                ;is added to the current auxiliary
                ;register.
```

Example 5. Indirect Addressing—Decrement by Index Amount

```
ADD *0-,8       ;Operates as in Example 1   , but in
                ;addition, the content of register AR0
                ;is subtracted from the current auxiliary
                ;register.
```

Assembly Programming Using the C2xx DSP Instruction Set

The complete detailed instruction set for the C2xx DSP core can be found in the Texas Instruments TMS320F/C24x DSP Controllers Reference Guide: CPU and Instruction Set; Literature Number: SPRU160C. This reference guide contains a complete descriptive listing on syntax, operands, binary opcode, instruction execution order, status bits affected by the instruction, number of memory words required to store the instruction, and clock-cycles used by the instruction. The Texas Instruments documentation on the assembly instruction set is very well written. Each assembly instruction has a complete explanation of the instruction, all optional operands, and several examples of the instructions used. Since including the instruction set and complete documentation would make this book excessively long, we will assume the reader has access to the documentation referred to above.

We will therefore focus on developing code, not the instruction set itself. Each command starts with the basic assembly instruction. Each command supports specific addressing modes and options. For example, the ADD command will work with direct, indirect, and immediate addressing. In addition to the basic command, many instructions have additional options that may be used with the instruction. For example, the ADD command supports left shifting of the data before it is added to the accumulator.

The following is the instruction syntax for the ADD command:

```
ADD dma [, shift]          ; Direct addressing
ADD dma, 16                ; Direct with left shift of 16
ADD ind [, shift [, ARn]]  ; Indirect addressing
ADD ind, 16 [, ARn]        ; Indirect with left shift of 16
ADD #k                     ; short immediate addressing
ADD #lk [, shift]          ; Long immediate addressing
```

The following is a list of the various notations used in C2xx syntax examples:

Italics Italic symbols in instruction syntax represent variables.

Example:

LACC dma, you can use several ways to address the dma (data memory address).

```
LACC *
```

Or

```
LACC 200h
```

Or

LACC v *; where "v" is any variable assigned to data memory*

*Where *, 200h, and v are the data memory addresses*

Boldface Characters Boldface characters must be included in the syntax.

Example:

LAR dma, 16 *; direct addressing with left shift of 16*

LAR AR1, 60h, 16 *; load auxiliary AR1 register with the memory contents of 60h that was left shifted 16 bits*

Example:

LACC dma, [shift] *; optional left shift from 0, 15; defaults to 0*

LACC main_counter, 8 *; shifts contents of the variable "main_counter" data 8 places to the left before loading accumulator*

[ ] An optional operand may be placed in the placed here.

Example:

LACC ind [, shift [, AR n]_] *Indirect addressing*

*LACC **; load Accum. W/contents of the memory*
; Location pointed to by the current AR.

*LACC *, 5* *; load Accum. With the contents of the memory*
; Location pointed to by the current AR after
; The memory contents are left shifted by 5
; Bits.

*LACC *, 0, AR3* *; load Accum. With the contents of the memory*
; Location pointed to by the current AR after
; The memory contents are left shifted by 5
; Bits. Now you have the option of choosing
; A new AR. In this case, AR3 will become the
; New AR.

[, x1 [, x2]] Operands x1 and x2 are optional, but you cannot include x2 without also including x1.

It is optional when using indirect addressing to modify the data. Once you supply a left shift value from 0...15 (even a shift of 0), then you have the option of changing to a new current auxiliary register (AR).

The # sign is prefix that signifies that the number used is a constant as opposed to memory location.

Example:

```
RPT #15          ; this syntax is using short immediate addressing. It will repeat the next
                 instruction 15+1 times.
LACC #60h        ; this will load the accumulator with the
                 ; Constant 60h
LACC 60h         ; However, this instruction will load the
                 ; Accumulator with the contents in the data
                 ; Memory location 60h, not the constant #60h
```

We will now provide a few examples of using the instruction set. Example 2.1 performs a few arithmetic functions with the DSP core and illustrates the nature of assembly programming. Programming with the assembly instruction set is somewhat different than languages such as C. In a high-level language, to add two numbers we might just code "c = a + b". In assembly, the user must be sure to code everything that needs to happen in order for a task to be executed. Take the following example:

Example 2.1 - Add the two numbers "2" and "3":

```
LDP #6h          ; loads the proper DP for dma 300h
SPLK #2, 300h    ; store the number "2" in memory address 300h
LACL #3          ; load the accumulator with the number "3"
ADD 300h         ; adds contents of 300h ("2") to the contents
                 ; of the accumulator ("3"); accumulator = 5

Another way:
LDP #6h          ; loads the proper DP for dma 300h
SPLK #2h, 300h   ; store the number "2h" in memory address
                 ; 300h
SPLK #3h, 301h   ; stores the number "3h" into memory address
                 ; 301h
```

```
LACL 300h       ; load the accumulator with the contents in
                ; Memory location 300h
ADD 301h        ; adds contents of memory address 301h ("3h")
                ; To the contents of the accumulator ("2h")
                ;accumulator = 5h
```

Looping algorithms are very common in all programming languages. In high-level languages, the "For" and "While" loops can be used. However, in assembly, we need a slightly different approach to perform a repeating algorithm. The following example is an algorithm that stores the value "1" to memory locations 300h, 301h, 302h, 303h, and 304h.

Example 2.2- Looping Algorithm Using the Auxiliary Register

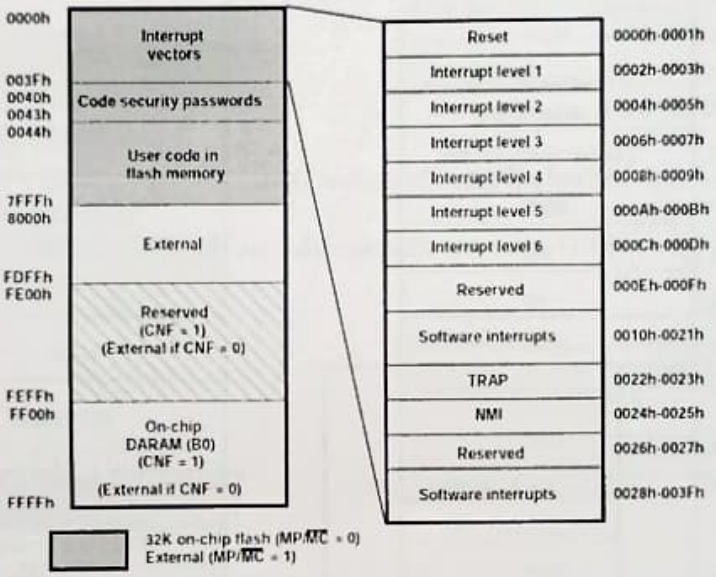
```
LAR AR0, #4         ; load auxiliary register 0 with #4
LAR AR1, #300h      ; this AR will be used as a memory pointer
LACL #1h            ; loads "1" into the accumulator
LOOPER MAR *, AR1   ; makes AR1 the next current AR
SACL *+, AR0        ; writes contents of accumulator to address
                    ; pointed to by AR1, the "+" increments AR1
                    ; By 1, next current AR is AR0
BANZ LOOPER         ; branch to LOOPER while current AR is not 0;
                    decrements current AR by 1 and branches
                    ; back to LOOPER
```

One might wonder if assembly language is so tedious to use, why not just program in a high-level language all the time. When code written in a high level language is compiled into assembly, the length of the code increases substantially. For example, if an assembly program takes up 50 lines, the same program written in C might take 150 lines after it is compiled. For this reason, code written in assembly almost always executed faster and uses less memory than high-level language code.

Memory Maps

Program Memory

– When a program is loaded to LF2407, the code resides in and run from program memory space. In addition, program memory can also store immediate operands and table information.



Two factors determine the configuration of program memory - CNF bit and MP/MC Pin

- CNF bit

– Determines if B0 memory is in on chip program space.
– If 0, 256 words are mapped as external memory
– If 1, 256 words of DARAM B0 are configured for Program Space.

- MP/MC Pin

0 – Device is configured in microcontroller mode. Flash memory is accessible.

1- Device is configured in microprocessor mode. Program memory is mapped to external memory.

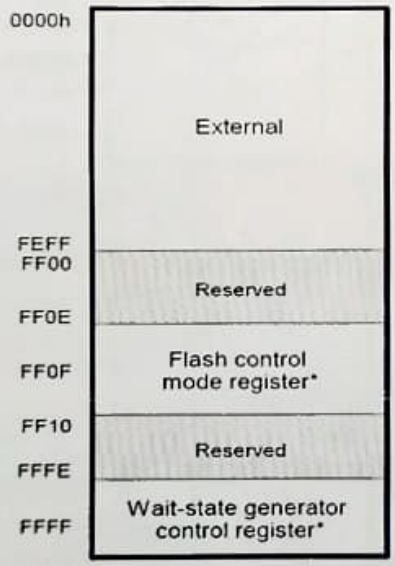
Data Memory

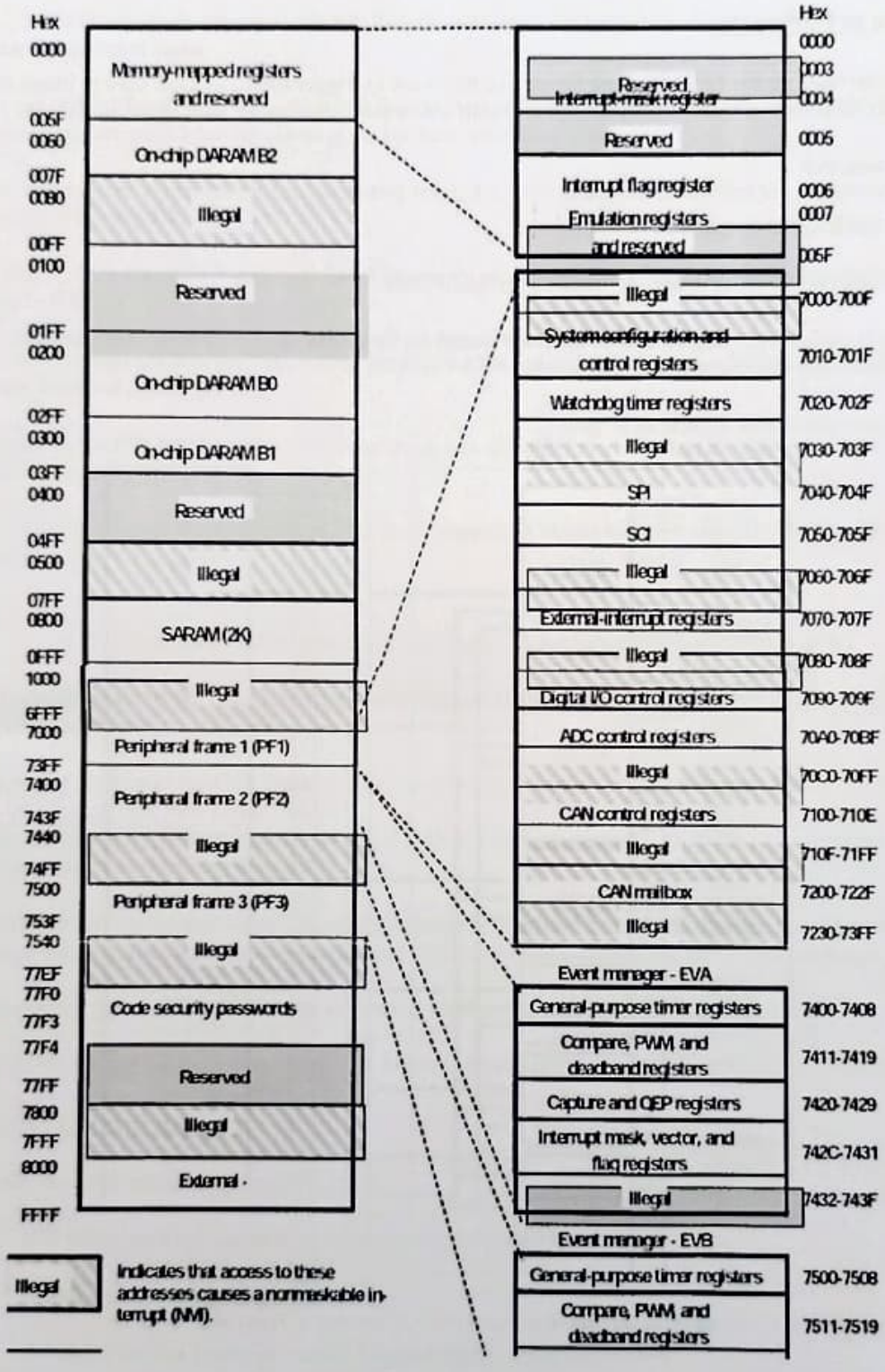
- For the execution of a program, it is necessary to store calculation results or look up tables in memory – data memory

- To store a value to data memory address, the corresponding memory block must reside in data memory space.
- Blocks B1 and B2 – Data Space
- Block B0 and SARAM – for Program or Data Space
- In addition, data memory space can access on chip configuration registers and peripherals.

Input / Output space (I/O)

- It is used for accessing external peripherals such as DAC…
- Within program , data and I/O space are addresses that are reserved for system functionality .




Hex
0000
Memory-mapped registers and reserved
005F
0060
On-chip DARAM B2
007F
0080
Illegal
00FF
0100
Reserved
01FF
0200
On-chip DARAM B0
02FF
0300
On-chip DARAM B1
03FF
0400
Reserved
04FF
0500
Illegal
07FF
0800
SARAM (2K)
0FFF
1000
Illegal
6FFF
7000
Peripheral frame 1 (PF1)
73FF
7400
Peripheral frame 2 (PF2)
743F
7440
Illegal
74FF
7500
Peripheral frame 3 (PF3)
753F
7540
Illegal
77EF
77F0
Code security passwords
77F3
77F4
Reserved
77FF
7800
Illegal
7FFF
8000
External
FFFF
Illegal
Indicates that access to these addresses causes a nonmaskable interrupt (NMI).
Hex
0000
Reserved
0003
Interrupt mask register
0004
Reserved
0005
Interrupt flag register
0006
Emulation registers and reserved
0007
005F
Illegal
7000-700F
System configuration and control registers
7010-701F
Watchdog timer registers
7020-702F
Illegal
7030-703F
SPI
7040-704F
SCI
7050-705F
Illegal
7060-706F
External interrupt registers
7070-707F
Illegal
7080-708F
Digital I/O control registers
7090-709F
ADC control registers
70A0-70BF
Illegal
70C0-70FF
CAN control registers
7100-710E
Illegal
710F-71FF
CAN mailbox
7200-722F
Illegal
7230-73FF
Event manager - EVA
General-purpose timer registers
7400-7408
Compare, PWM, and deadband registers
7411-7419
Capture and QEP registers
7420-7429
Interrupt mask, vector, and flag registers
742C-7431
Illegal
7432-743F
Event manager - EVB
General-purpose timer registers
7500-7508
Compare, PWM, and deadband registers
7511-7519

Introduction to Interrupts

- The interrupts on the LF2407 allow the device hardware to trigger the C2xx DSP core to break from the current task, branch to a new section of code and start a new task, then return back to the initial task.
- New task-ISR
- Interrupt Request Sequence

– There are two levels of interrupt hierarchy in the LF2407.

– There is an interrupt flag bit and an interrupt enable bit located in each peripheral configuration register for each event that can generate an interrupt.

– The interrupt enable bit acts as a "gate".

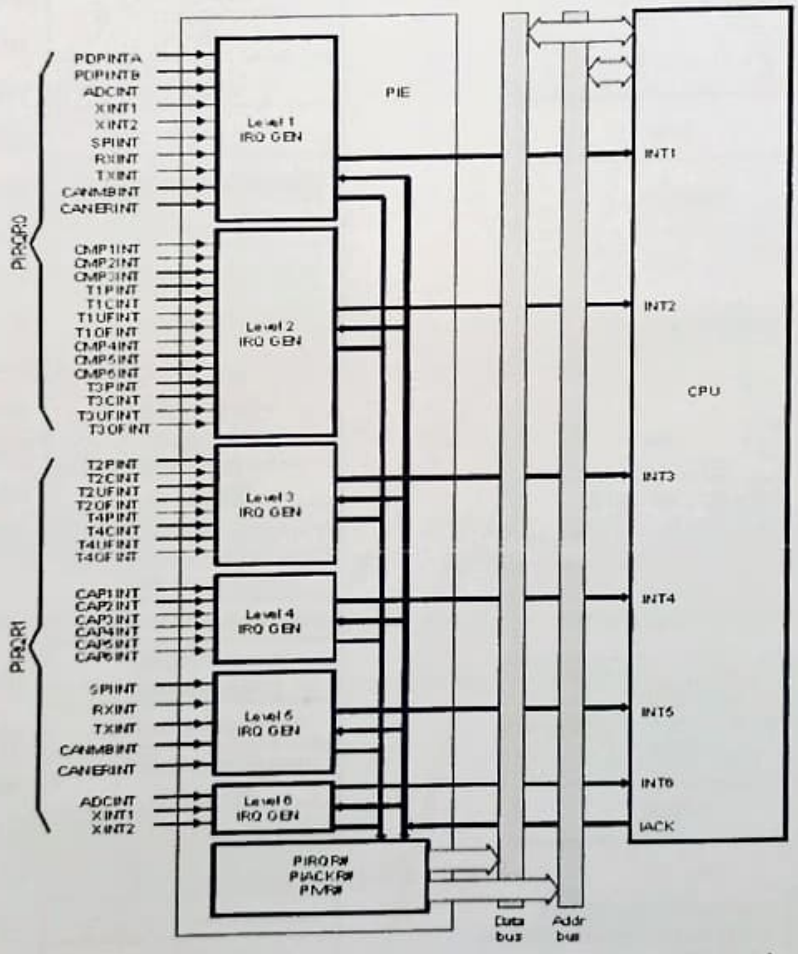


Figure 4.1 Interrupt hierarchy in the LF2407 (Courtesy of Texas Instruments)

If the interrupt enable bit is not set, then the setting of the peripheral flag bit will not be able to generate an interrupt signal.

• If the enable bit is set, then the peripheral flag bit will generate an interrupt signal. That interrupt signal will then leave the peripheral level and go to the next hierarchal level.

• Once an interrupt signal leaves the peripheral level, it is then multiplexed through the Peripheral Interrupt Expansion (PIE) module.

• The PIE module takes the many individual interrupts and groups them into six priority levels (INT1 through INT6).

• Once an interrupt reaches the PIE, a code identifying the individual interrupt is loaded into the Peripheral Interrupt Vector Register (PIVR).

• This allows the ISR to determine which interrupt was actually asserted when multiple interrupts from the same level occur.

• After passing through the PIE module, the interrupt request signal has now entered the upper level of hierarchy or the "CPU level".

• CPU Level:

– Each of the six levels has a corresponding flag bit in the Interrupt Flag Register (IFR).

– Additionally there is an Interrupt Mask Register (IMR) which acts similar to the interrupt enable bits at the peripheral level.

• Each of the six bits in the IMR behaves as a "gate" to each of the corresponding six bits in the IFR.

• If the corresponding bits in both the IFR and IMR are both set, then the interrupt request signal can continue through to the C2xx core itself.

• Once the interrupt request signal has entered the CPU level and has passed through the IFR/IMR, there is one more gateway the signal must pass through in order to cause the core to service the interrupt.

• The Interrupt Mask (INTM) bit must be cleared for the interrupt signal to reach the core.

• The INTM bit and the peripheral level flag bit must be cleared "manually" via Interrupt Control Register

Interrupt Control Register

The three registers are used at the CPU level.

- IFR-Interrupt Flag Register at the beginning of the CPU level
- IMR-Interrupt Mask Register
- PIVR-Peripheral Interrupt Vector Register-At the peripheral Level.
- In addition INTM bit at the CPU level provides the "gateway" that the interrupt must pass through to

reach the core.

Interrupt Flag Register

- IFR is used to identify and clear the pending interrupts at the CPU level and contains the interrupt flag bits for the maskable interrupt priorities INT1-INT6.
- After the interrupt is serviced, the IFR bit corresponding to the interrupt is automatically cleared.
- When the interrupt is acknowledged ,only the IFR bit is cleared automatically.
- IFR pertain to interrupts at the CPU level only. All peripherals have their own interrupts mask and flag bits in their respective control / configuration bits.

Interrupt Flag Register (IFR) — Address 0006h

15-6	5	4	3	2	1	0
Reserved	INT6 flag	INT5 flag	INT4 flag	INT3 flag	INT2 flag	INT1 flag
0	RW1C–0	RW1C–0	RW1C–0	RW1C–0	RW1C–0	RW1C–0

INTERRUPT MASK REGISTER (IMR)

- It is a 16 bit register which contains mask bits for each of the six interrupt priority levels INT1 –INT6.
- When an IMR bit is '0' the corresponding interrupt is masked (halted at the CPU level).
- IMR may also be used to identify which interrupts are masked or unmasked.

15-6	5	4	3	2	1	0
Reserved	INT6 mask	INT5 mask	INT4 mask	INT3 mask	INT2 mask	INT1 mask
0	RW	RW	RW1	RW	RW	RW

PERIPHERAL INTERRUPT VECTOR REGISTER (PIVR)

- PIVR is a 16 bit read only register.
- Each interrupt has a unique code which is loaded into PVIR when in the PIE module.
- This assures that the different priorities happen simultaneously, the higher priority interrupt will be serviced first.

Peripheral Interrupt Vector Register (PIVR) – Address 701EH

15	14	13	12	11	10	9	8
V15	V14	V13	V12	V11	V10	V9	V8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
V7	V6	V5	V4	V3	V2	V1	V0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

R – Read access.
0 = value after reset.
Bits 15-0 V15 – V0.
Interrupt vector. This register contains the peripheral interrupt vector of the most recently acknowledged peripheral interrupt.

External Interrupt Control Registers

The external interrupts (XINT1, XINT2) are controlled by the XINT1CR and XINT2CR control registers, respectively.

If these interrupts are enabled in their control registers, an interrupt will be generated when the XINT1or XINT2 logic transition occurs for at least 12 CPU clock cycles.

15	14–3	2	1	0
XINT1 flag	Reserved	XINT1 polarity	XINT1 priority	XINT1 enable
RC-0	R-0	RW-0	RW-0	RW-0

External Interrupt 2 Control Register (XINT2CR) – Address 7071h

15	14-3	2	1	0
XINT2 flag	Reserved	XINT2 polarity	XINT2 priority	XINT2 enable
RC-0	R-0	RW-0	RW-0	RW-0

***Note**: R = read access, W = write access, C = Clear by writing a 1, -0 = value after reset.*

UNIT - V

FIELD PROGRAMMABLE GATE ARRAYS (FPGA)

* Introduction to field programmable Gate arrays
* CPLD Vs FPGA
* Types of FPGA
* Xilinx
  XC3000 series
* Configurable logic Blocks (CLB)
* Input/output Block (IOB)
* Programmable Interconnect point (PIP)
* Xilinx 4000 series
* HDL programming
* Overview of spartan 3E and virtex II pro FPGA boards -
* case study

Introduction to field programmable Gate Arrays :-

⇒ F = field → The user or Designer

P = Programmable → which can be reprogrammed

G = Gate → Logic gate - Boolean functions

A = Array → In rows and columns

FPGA : Means is an Integrated circuit which can be programmed by user/ designer to implement any Boolean logic on components arranged in rows and columns.

* FPGA is a semiconductor device that can be configured by the customer or designer after manufacturing. Hence the name as field - programmable.
* FPGAs are programmed using a logic circuit diagram or a source code in a 'HDL' (Hardware programming lang.)
* Unlike ASIC which can perform a single specific function for life time of the chip and FPGAs can be reprogrammed to perform a different function.

→ Here two kinds of "chips"

ASIC	FPGA
* ASIC - Application specific Integrated circuit.	* FPGA - Field programmable Gate Array
chip designed of particulars	* A programmable chip an be programmed for required functionality.
* application	* It is good for research and development activities.

⇒ FPGA Architecture :

It has `3' main components

CLB's * Logic elements (Les) (or) configurable logic Blocks (CLBs)

SM * Inter connections (Also called switch Box) (or) Switch Matrix

IOpads * Input / output Blocks (IOBs) (or) I/O pads / I/o Blocks.

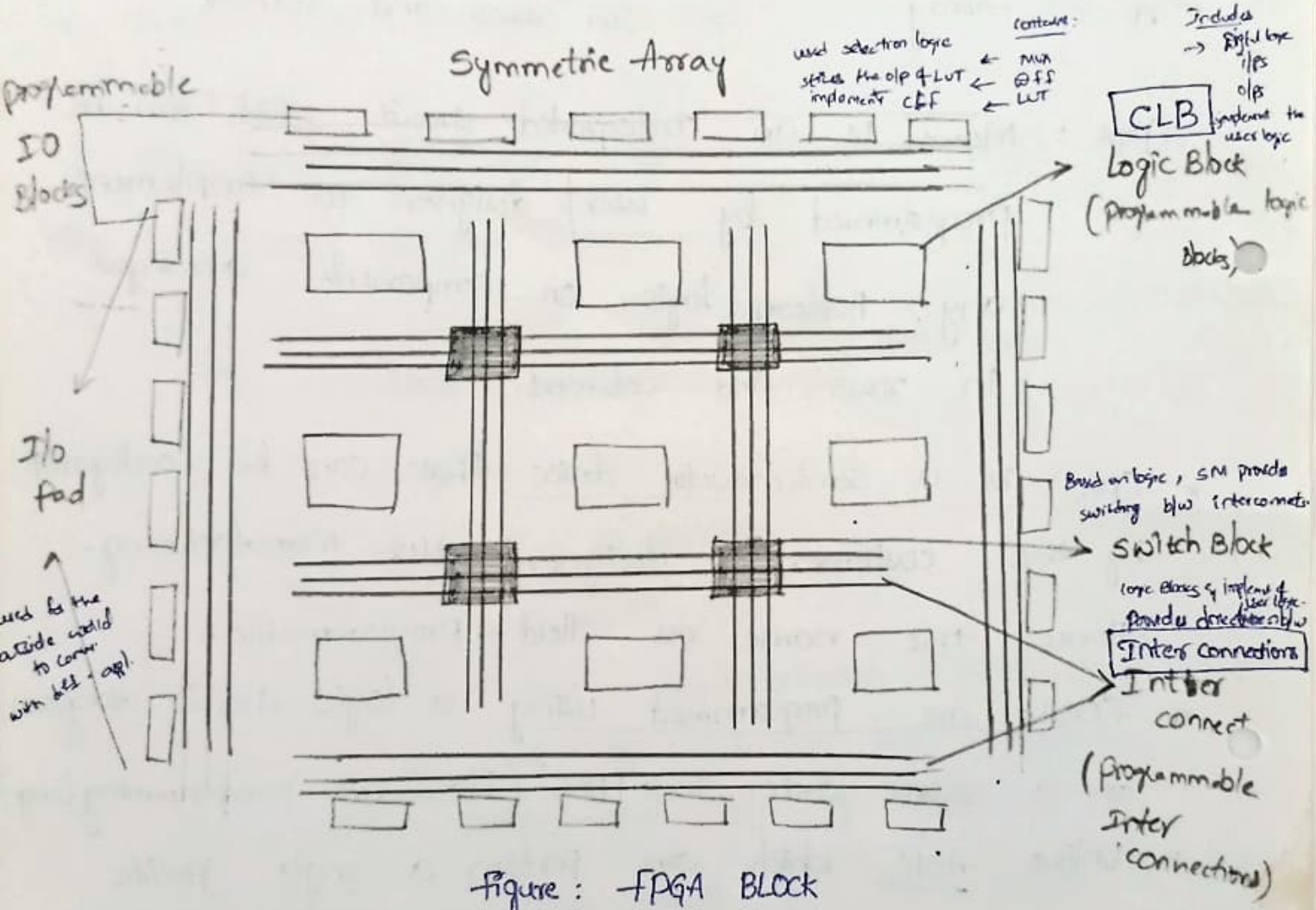


Figure : FPGA BLOCK

The Basic FPGA architecture has two dimensional arrays of Logic Blocks with a means for a user to arrange the interconnection between the logic blocks.

The basic FPGA architecture has functions discussed below:

* CLB — configurable logic Block includes digital logic
  Input
  Outputs.

It implements the user logic.

* Interconnects provides direction between the logic blocks to implement the user logic.
* Depending on the logic, switch Matrix provides switching between interconnects.
* I/O pads used for the outside world to communicate with different applications.
* Logic Block contains MUX (Multiplexer)
  D flip flop
  LUT
* LUT implements the combinational logical functions; the MUX is used for selection logic and D flip flop stores the o/p of the LUT.

The basic building block of the FPGA is the Look up Table based function generator. The no. of i/ps to the LUT vary from 3, 4, 6 and even 8 experiments. Now, we have adaptive LUTs that provides two o/ps per single LUT with the implementation of two function generators.

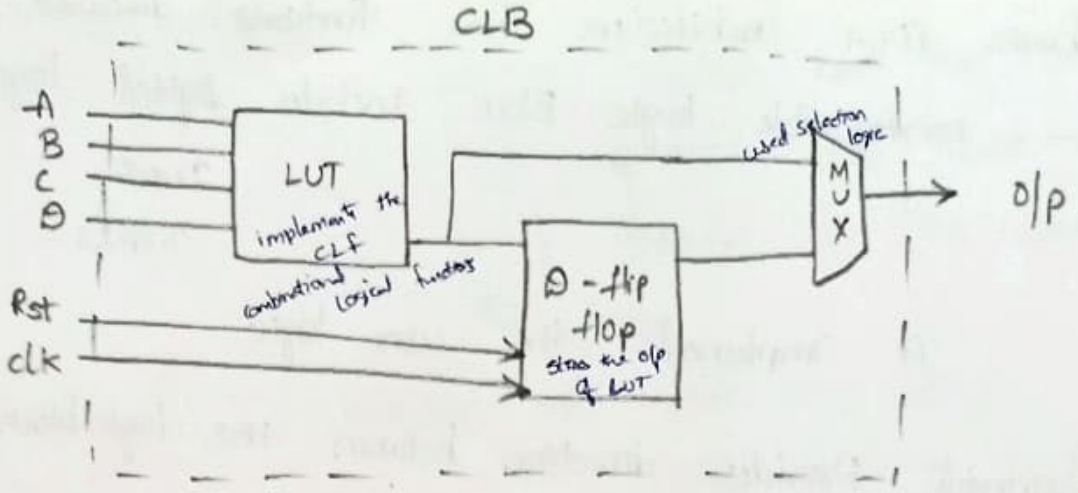
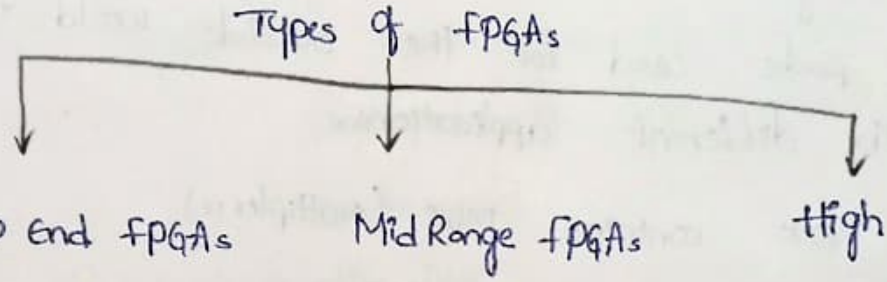


fig: FPGA LOGIC Block

⇒ Types of FPGA's Based on Applications :-



① Low End FPGAs — These types of FPGAs are designed for low power consumption
Low logic density
Low complexity per chip.
Examples are Cyclone family from Altera
Spartan " " Xilinx
fusion " " Microsemi and
Mach XO/ICE 40 from Lattice Semiconductors.

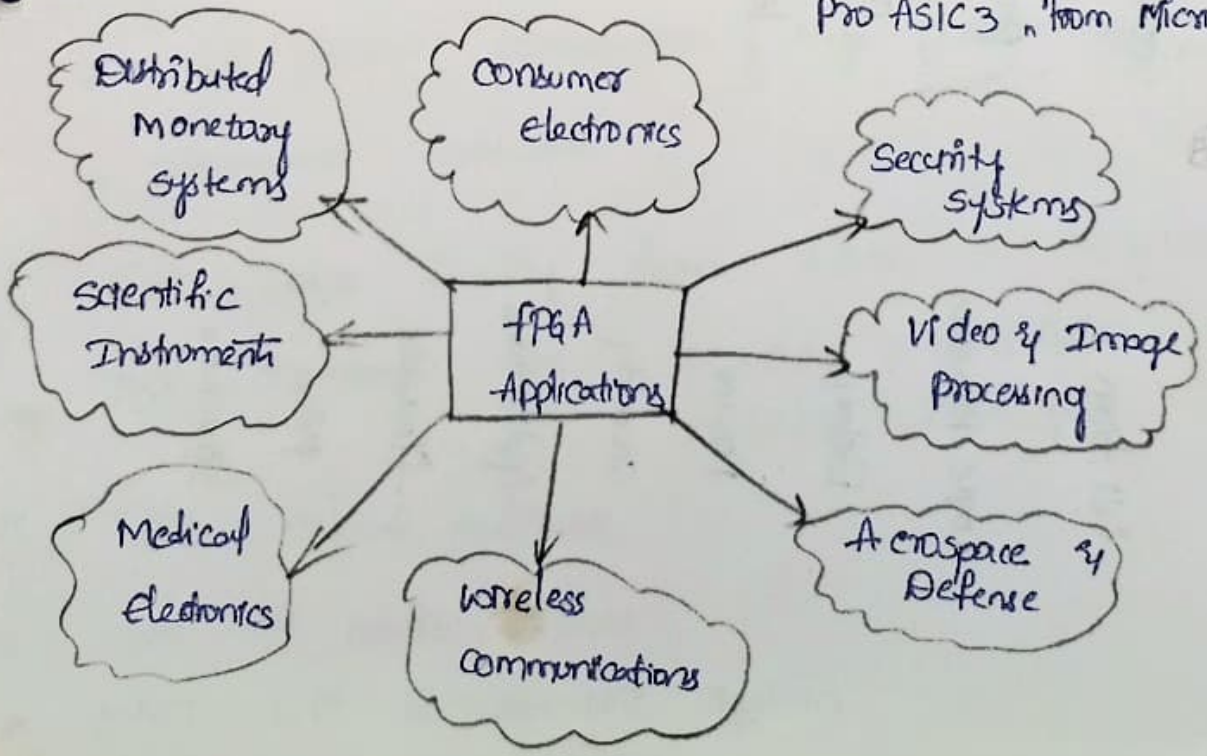
② Mid Range fpGAs — These types of fpGAs are the optimum solution b/w the low-end and high-end fpGAs and these are developed as a balance b/w the performance and the cost.

Examples are Arria from Altera
Artix-7/ Kintex-7 series from Xilinx
IGLOO2 from microsemi and
ECP3 and ECP5 series from
Lattice Semiconductor.

③ High End fpGAs — These types of fpGAs are developed for logic density and high performance.

examples are a Stratix family from Altera
Virtex " " Xilinx
Speedster 22i from Achronix
Pro ASIC3 family from Microsemi.

⇒ Applications :-



Parameter	CPLD	Vs	FPGA
full form	complex programmable logic devices		field programmable gate array
Logic Resources	provides less logic resources		More
Delays	easier to predict delays		Difficult
Power	less amount of power		More
Security	More Secure		less
Applications	small & medium applications		complex application
flexible	less flexible		More
price	less price		More
Structure	consist of PLDs PLA (or) PAM		CLPs configurable logic blocks

Xilinx FPGA's are well known of running a standard embedded operating system such as Linux (or) Vx works. And implementing processor peripherals in programmable logic.

The FPGA families Virtex – II pro
Virtex – 4
Virtex – 5 and
Virtex – 6 which have up to two embedded IBM power PC cores, are specially made for SOC (system – on – chip) designers.

Xilinx XC 3000 features (generic features)

* High performance at different voltages
* Foot print compatibility
  – Devices with in each family are compatible.
* Low power consumption | high performance
* Integrated software

→ First FPGA family from Xilinx is XC2000. The
→ Two members are XC2064 1000 Gates
→ XC2018 1500 gates
→ Ext. Crystal Oscillator
→ NO Tri state Buffers
→ XACT 1.0 Development system

XC3000 – Series :

The XC3000 – Series field programmable gate Arrays (FPGAs) provide a group of high – performance
high – density
Digital ICs

Some features of XC3000 – series are regular,
extendable,
flexible,
user friendly (and

user – programmable array architecture is composed of a configuration program store plus three types of configurable elements.

I/O Blocks
CLBs
Resources for Interconnection.

The XC3000 FPGA families provide a variety of
logic capacities
package styles
temperature ranges and
speed grades

There are now 4 distinct family groupings with in the XC3000 series of FPGA devices :

XC3000A family → enhanced version of XC3000 and user friendly enhancement.

XC 3000L " → Identical in Arch. and features to XC3000A but operate nominal supply Vtg of 3.3V & best solution for battery-operated low power Application

XC 3100A " → It is performance-optimized relative of XC3000A family

XC 3100L " → Identical in Arch and features to XC3000A but operate nominal supply Vtg of 3.3V

All 4 families share a common architecture development s/w design and programming Methodology and also common package pin-outs.

* XC 3000A family :- It is an enhanced version of the basic XC3000 family, featuring additional interconnect resources and other user-friendly enhancements.

* XC3000L family :- It is identical in architecture and features to the XC3000A family, but operates at a nominal supply Vtg of 3.3V. It is the right solution for battery-operated and low-power applications.

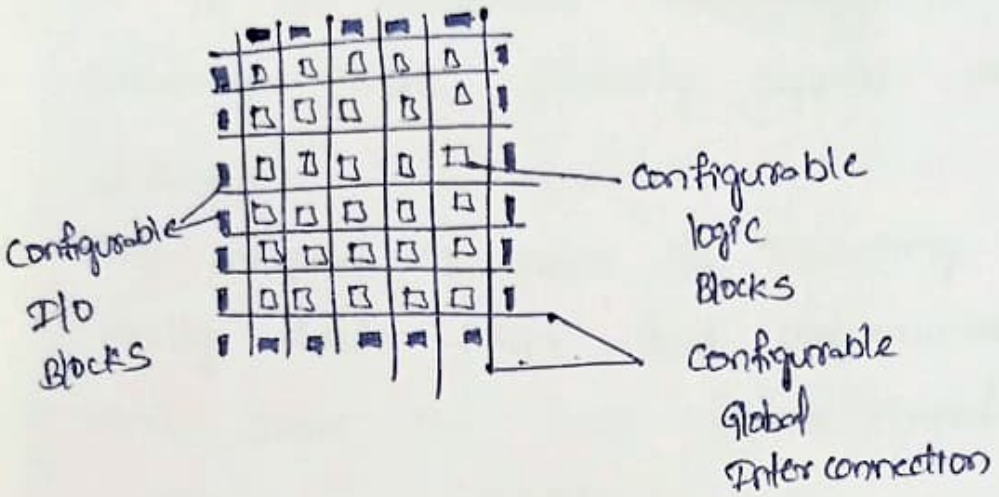
XC3100A family :- It is a performance - optimized relative of the XC3000A family, while both families are bitstream and foot print compatible, the XC3100A family extends toggle rates to 370 MHz and in-system performance to over 80 MHz.

XC3100L family :- The XC3100L is identical in architectures and features to the XC3100A family, but operates at a nominal supply voltage of 3.3V.

XC4000 series :-

The XC4000 series high performance, high capacity FPGAs provide the benefits of custom CMOS VLSI, while avoiding the initial cost long development cycle of inherent risk of a conventional masked gate arry.

Xilinx General Architecture



- The Xilinx 4000 Architecture also have above same architecture contains CLB
  IOB
  Inter connections

Xilinx 4000 spec & features :

Synchronous single and Dual – port RAM

Internal Three – state buffers

System performance to 80MHz

0.5 μ SRAM process Technology.

Xilinx 4000 Inter connections → 3 types :

- fast Direct connection
- General purpose connections with switching Matrix
- Horizontal | vertical long

→ Lines :

Single length (8)
Double " (4)
Long lines (6)
Global lines (4)

17/7/2023

Xilinx :-

It is an American technology and semiconductor company that primarily supplies programmable logic devices.

The company is known for inventing the first commercially viable FPGA 'field programmable gate Array' and creat the first fabless manufacturing model.

Type	Subsidiary
Industry	Integlated circuit
founded	1984; 39 yrs ago
founder	James V. Barnett II Ross freeman Bernie Vonderschmitt
Head quarters	San Jose, California, U.S
Area served	World wide
key people	Dennis segers (chairman) Victor peng (president, CEO) Brice Hill (CFO) Ivo Bolsens (CTO)
Products	FPGAs, CPLDs (complex prog logic Devices)
No. of Employees	4,890 (April 2021)
parent	AMD
website	www.xilinx.com

* Xilinx sells both FPGAs and CPLDs for electronic equipment manufacturers in end markets such as communications
Industrial
consumer
automotive and
data processing.

* It's World's leading innovator of complete programmable logic solutions

* Programmable logic fits in to engineering curriculum are Engineering labs
Research projects
senior Design projects
Design contests

* Universities use programmable Logic in
  * professor assigns application (project, lab Assignment, etc.)
  * student creates application with Xilinx s/w tools.
  * when Application is final, student implements design in h/w.

Xilinx products Design Tools

⇒ V6.2i ISE software

* complete s/w package
  - Design entry (schematic, VHDL, Verilog)
  - synthesis (XST)
  - Implementation (Translate, Map, Place & Route)
  - Simulation (model Sim)
  - iMPACT programmer (Download Bitstream)
* CORE Generator
  - parameterizable cores
* State CAD/state Bencher
  - state Machine Design
* HDL Bencher
  - Test Bench Generation
* Unix
  Linux and
  PC platforms.

⇒ 6.2i Device support

all xilinx leading fpga/cpld families

1. New leading-edge device families; are
   - Virtex-II pro
   - Spartan-IIE
   - cool Runner-II
2. ISE advantages can be leveraged across various engineering courses
   - Across all device families and design sizes;

Virtex-II
Virtex
Sparten-II
XC9500XV and coolRunner XPLA3.

Xilinx products
- CPLDs and FPGAs

Complex Programmable Logic Device (CPLD)

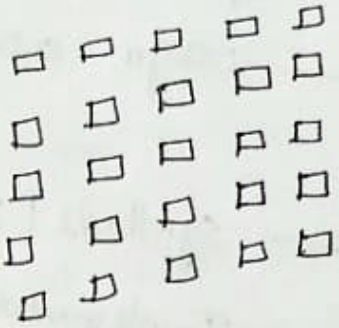
* Architecture :-
  PAL | 22V10 - like
  more combinational

* Density :-
  Low - to - medium
  0.5 - 10K logic gate

* Performance :-
  predictable timing
  up to 250 MHz today

* Interconnect :-
  crossbar switch

Field Programmable Gate Array (FPGA)



* Gate array - like
  More registers + RAM

* Medium to high
  1K to 3.2M system gates

* Applicant dependent up to
  200MHz today

* Incremental..

A programmable systems that are used for various applications

Examples of programmable Devices:

DSP processors (programmed by c/c++)

Microcontrollers (" " c/c++ (or) Assembly)

FPGA's (" " HDL (VHDL or VERILOG))

PLC's (" " Language similar to BASIC)

A FPGA is a general purpose IC ckt that is "programmed" by the designer / user rather than the device manufacturer.

FPGA Types and Manufacturers:

Two major Manufacturers (Xilinx and Altera)

Xilinx FPGA families: Spartan
Virtex
Kintex
Artix
Zynq (SOC with ARM Processor)

FPGA Design Tools:

1. Traditional FPGA Design tools

Xilinx ISE (Integrated Site Environment)

VHDL
VERILOG
Schematic

2. Model Based Design (MBD)

Xilinx system Generator (with matlab simulink)

Matlab HDL coder

3. High-level synthesis Design tools

4. c/c++

The spartan - 3E family of FPGAs is specially designed to meet the needs of high volume cost sensitive consumer e applications.

features :

1. Very low cost high performance logic solution for high-volume consumer - oriented applications.
2. proven advanced 90- nanometer process technology.
3. Multi - voltage, multi - standard SelectIO interface pins.
4. Up to 376 I/O pins (or) 156 differential signal pairs.

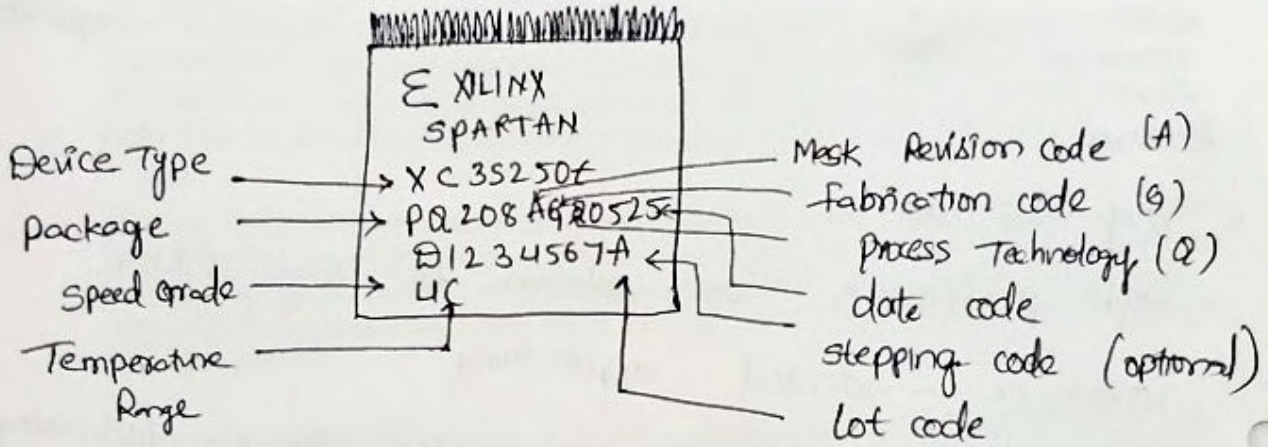
Architectural overview :

It consists of 5 fundamental programmable functional elements :

1. configurable logic Blocks (CLBs)
2. Input/output Blocks (IOBs)
3. Block RAM
4. Multiplier Blocks
5. Digital clock manager (DCM) Blocks

Spartan - 3E QFP package Marking Example:

Quad - flat package



Field Programmable Gate Arrays (FPGA)

Introduction

Field Programmable Gate Arrays (FPGAs) are digital ICs (Integrated Circuits) that enable the hardware design engineer to program a customized Digital Logic as per his/her requirements. The term "Field Programmable" implies that the Digital Logic of the IC is not fixed during its manufacturing (or fabrication) but rather it is programmed by the end-user (designer).

In order to provide this programmability, an FPGA consists of Configurable (or Programmable) Logic Blocks and configurable interconnects between these blocks. This configurable Logic and the Interconnections (Routing) of FPGAs makes them general purpose and flexible but at the same time, it also makes them slow and power hungry when compared to a similar calibre ASIC with Standard Cells.

It has been more than three decades since the introduction of FPGAs into the market and in this long span, they have undergone a severe technological advancement and gained a continuously growing popularity.

A Brief Note on PLD (Programmable Logic Device)

Before diving into the main topic, I want to briefly discuss the concept of Programmable Logic Devices. So, what is a PLD. It is an IC containing a large number of Logic gates and Flip-flops that can be configured by the user to implement a wide variety of functions.

The simplest of Programmable Logic Devices consists of an array of AND & OR gates and the logic of these gates and their interconnections can be configured by a programming process.

PLDs are particularly useful when an engineer wants to implement a customized logic and is restricted by the pre-configured integrated circuits. PLDs provide a way to implement a custom digital circuit through the power of hardware configuration rather than implementing it using a software.

Different Types of PLDs

Basically, PLDs can be categorized into three types. They are:

- Simple Programmable Logic Devices (SPLD)
- Complex Programmable Logic Devices (CPLD)
- Field Programmable Gate Arrays (FPGA)

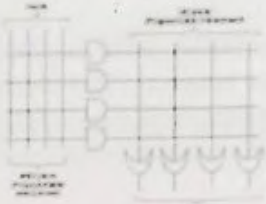
The Simple Programmable Logic Devices are further divided into:

- Programmable Logic Array (PLA)
- Programmable Array Logic (PAL)
- Generic Array Logic (GAL)

Let us now see some basic details about all these PLDs.

Programmable Logic Array (PLA)

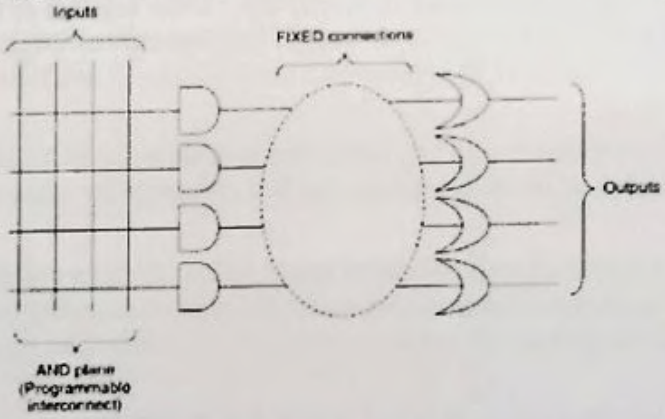
A PLA consists of an AND gate plane with programmable interconnects and an OR gate plane with programmable interconnects. The following is a simple four input – four output PLA with AND & OR gates.



Any input can be connected to any AND gate by connecting the horizontal and vertical interconnect lines. The outputs from different AND gates can then be applied to any of the OR gates with programmable interconnects.

Programmable Array Logic (PAL)

A PAL is similar to the PLA but the difference is that in PAL, only the AND gate plane is programmable while the OR gate plane is fixed during fabrication. Even though PALs are less flexible than P ; OR Gates.

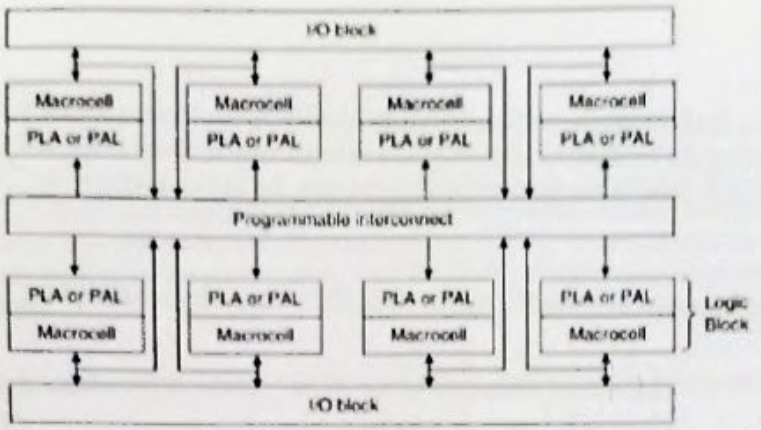


Generic Array Logic (GAL)

Architecture wise, a GAL is similar to a PAL but the difference lies in programmable structure. PALs use PROM, which is one-time programmable, while GAL uses EEPROM, which can be reprogrammed.

Complex Programmable Logic Devices (CPLD)

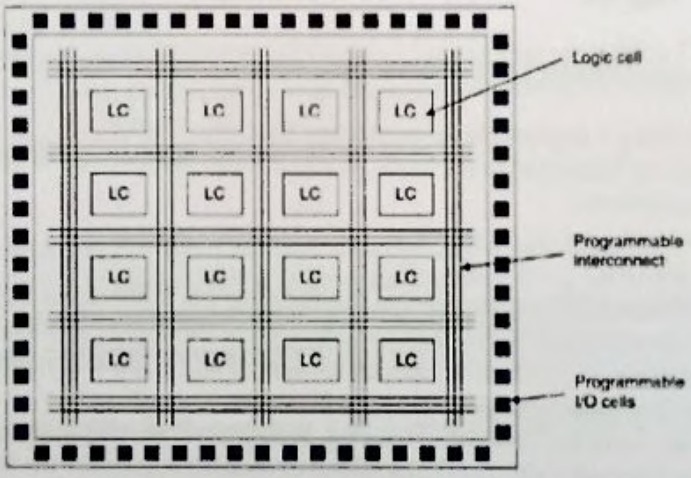
Moving up from SPLD devices, we get CPLD. It is developed on top of SPLD devices to create mush larger and complex designs. A CPLD consists of a number logic blocks (or functional blocks), which internally consists of either a Pal or a PAL along with a Macrocell.



Macrocell consists of any additional circuitry and signal polarity control to provide true signal or its complement.

Field Programmable Gate Arrays (FPGA)

Complexity wise, CPLD are much more complex than SPLDs. But FPGA are even more complex than CPLDs. The architecture of an FPGA is completely different as it consists of programmable Logic Cells, programmable interconnects and programmable IO blocks.

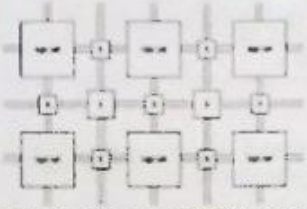


What is an FPGA?

Field Programmable Gate Arrays or FPGAs in short are pre-fabricated Silicon devices that consists of a matrix of reconfigurable logic circuitry and programmable interconnects arranged in a two-dimensional array. The programmable Logic Cells can be configured to perform any digital function and the programmable interconnects (or switches) provide the connections among different logic cells.

Using an FPGA, you can implement any custom design by specifying the logic or function of each logic block and setting the connection of each programmable switch. Since this process of designing a custom circuit is done in the field rather than in a fab, the device is known as "Field Programmable".

The following image shows a typical internal structure of an FPGA in a very broad sense.



As you can see, the core of the FPGA is made up of configurable logic cells and programmable interconnections. These are surrounded by a number of programmable IO blocks, which are used to talk to the external world.

Components of an FPGA

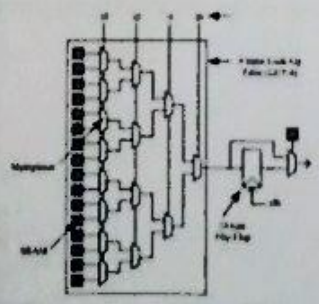
Let us now take a closer look at the structure of an FPGA. Typically, an FPGA consists of three basic components. They are:

- Programmable Logic Cells (or Logic Blocks) – responsible for implementing the core logic functions.
- Programmable Routing – responsible for connecting the Logic Blocks.
- IO Blocks – which are connected to the Logic Blocks through the routing and help to make external connections.

Logic Block

The Logic Block in Xilinx based FPGAs are called as Configurable Logic Blocks or CLB while the similar structures in Altera based FPGAs are called Logic Array Blocks or LAB. Let us use the term CLB for this discussion. A CLB is the basic component of an FPGA, which provides both the logic and storage functionalities. The basic logic block can be anything like a transistor, a NAND gate, Multiplexors, Look-up Table (LUT), a PAL like structure or even a processor. Both Xilinx and Altera use Look-up Table (LUT) based logic blocks to implement the logic as well as the storage functionalities.

A Logic Block can be made up of a single Basic Logic Element or a set of interconnected Basic Logic Elements, where a Basic Logic Element is a combination of a Look-up table (which is in turn made up of SRAM and Multiplexors) and a Flip-flop.



Many logic blocks are confined to a local set of connections and hierarchical routing architecture makes use of this feature by dividing the logic blocks into several groups or clusters. If the logic blocks are residing in the same cluster, then the hierarchical routing connects them in a low level of hierarchy.

If the logic blocks are residing in different clusters, then wiring is done over a higher level of hierarchy.

FPGA Programming Technologies

We have talked about the reprogrammable architecture of FPGAs quite a bit but now let us see some of the most commonly used programming techniques that is responsible for such reconfigurable architecture.

The following are three of the well-known programming technologies used in FPGAs.

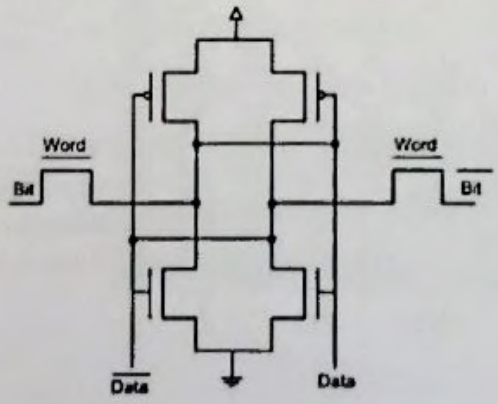
- SRAM
- EEPROM / Flash
- Anti-Fuse

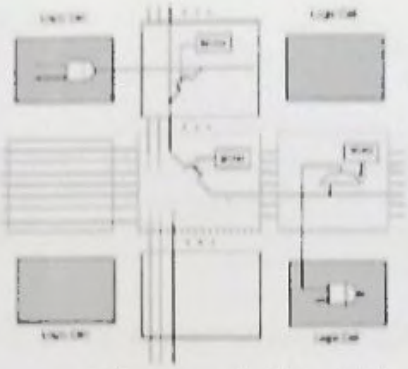
Other technologies include EPROM and Fusible Link but they are used in CPLDs and other PLDs but not in FPGAs, Hence, let us keep the discussion limited to FPGA related programming technologies.

SRAM

We know that there are two types of semiconductor RAM called the SRAM and DRAM. SRAM is short for Static RAM while DRAM is short for Dynamic Ram. SRAM is designed using transistors and the term static means that the value loaded on a basic SRAM Memory Cell will remain the same until deliberately changed or when the power is removed.

A typical 6 transistor SRAM Cell to store 1 bit is shown in the following image.





A LUT with 'n' inputs consists of 2^n configuration bits, which are implemented by SRAM Cells. Using these 2^n SRAM Bits, the LUT can be configured to implement any logical function.

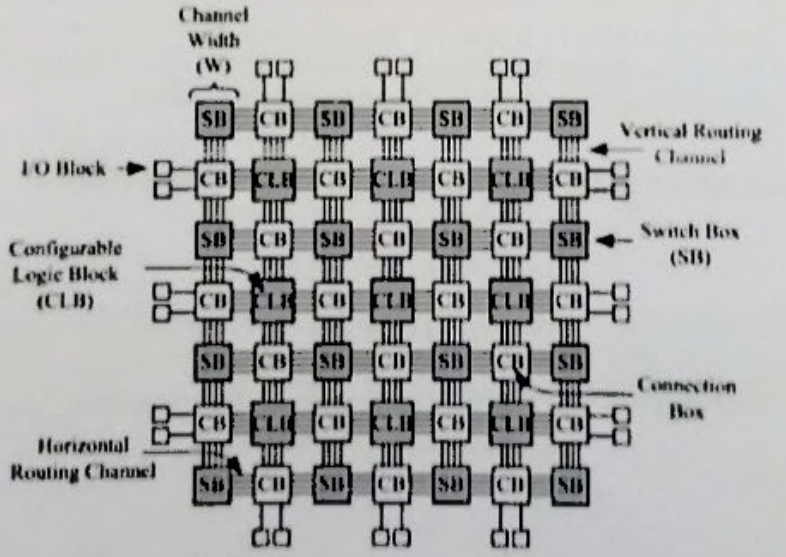
Routing

If the computational functionality is provided by the Logic Blocks, then the programmable routing network is responsible for interconnection these logic blocks. The Routing Network provides interconnections between one logic block to other as well as between the logic block and the IO Block to completely implement a custom circuit.

Basically, the routing network consists of connecting wires with programmable switches, which can be configured using any of the programming technologies. There are basically two types of routing architectures. They are:

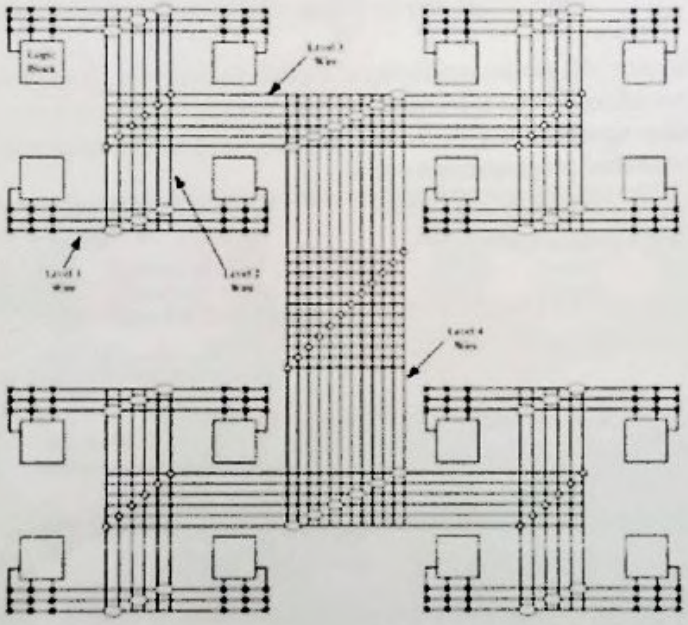
- Island Style Routing (also known as Mesh Routing)
- Hierarchical Routing

In island style routing architecture, the logic blocks are arranged in a two-dimensional array and are interconnected using a programmable routing network. This type of routing is widely used in commercial FPGAs.



This is in contrast to the DRAM, which consists of a combination of a transistor and a capacitor. The term Dynamic refers to the fact that the value loaded in the basic DRAM Memory Cell is valid until there is charge in the capacitor. As capacitor loses its charge over time, the memory cell has to be periodically recharged to maintain the charge. This is also known as refreshing.

Many FPGA vendors implement Static Memory Cells in SRAM based FPGAs for programming. SRAM based FPGAs are used to program both the logic cells and the interconnects and they have become quite predominant due to their re-programmability and use of CMOS technology, which is known for its low dynamic power consumption, high speed and tighter integration.



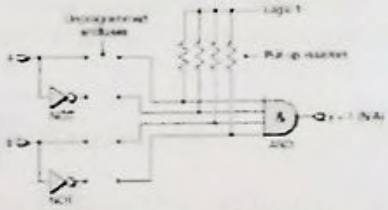
EEPROM / Flash

A close alternative to SRAM based programming technology is based on EEPROM or Flash programming technologies. The main advantage of flash-based programming is its non-volatile nature. Even though flash supports re-programmability, the number of times this can be done is very small when compared to an SRAM technology.

Anti-Fuse

The anti-fuse programming technology is an old technique of producing one-time programmable devices. They are implemented using a link called the antifuse, which in its unprogrammed state has a very high resistance and can be considered an open circuit.

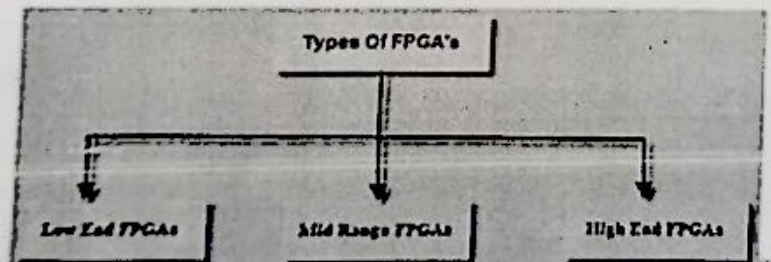
When programming, a high voltage and current is supplied to the input. As a result, the antifuse, which is initially in the form of amorphous silicon (basically an insulator with very high resistance) linking two metal tracks, comes to life by converting to a conducting polysilicon.



When compared to the other two technologies, the antifuse one occupies the least amount of space but comes only as one-time programmable option.

Types of FPGAs Based on Applications

Field Programmable Gate Arrays are classified into three types based on applications such as Low-end FPGAs, Mid-range FPGAs and high-end FPGAs.



Types of FPGAs

Low End FPGAs

These types of FPGAs are designed for low power consumption, low logic density and low complexity per chip. Examples of low end FPGAs are Cyclone family from Altera, Spartan family from Xilinx, fusion family from Microsemi and the Mach XO/ICE40 from Lattice semiconductor.

Mid Range FPGAs These types of FPGAs are the optimum solution between the low-end and high- end FPGAs and these are developed as a balance between the performance and the

cost. Examples of Mid range FPGAs are Arria from Altera, Artix-7/Kintex-7 series from Xlinix, IGL002 from Microsemi and ECP3 and ECP5 series from Lattice semiconductor.

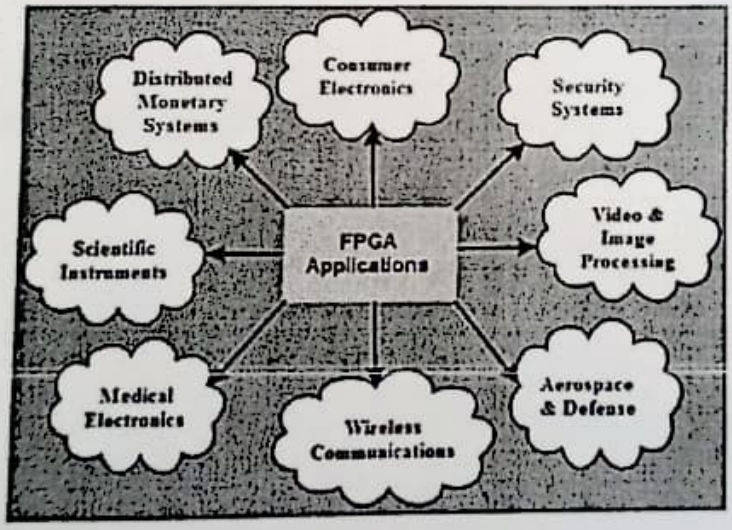
High End FPGAs

These types of FPGAs are developed for logic density and high performance. Examples of High end FPGAs are a Stratix family from Altera, Virtex family from Xilinx, Speedster 22i family from Achronix, and ProASIC3 family from Microsemi.

Applications of FPGA:

FPGAs have gained rapid growth over the past decade because they are useful for a wide range of applications. Specific application of an FPGA includes digital signal processing, bioinformatics, device controllers, software-defined radio, random logic, ASIC prototyping, medical imaging, computer hardware emulation, integrating multiple SPLDs, voice recognition, cryptography, filtering and communication encoding and many more.

Usually, FPGAs are kept for particular vertical applications where the production volume is small. For these low-volume applications, the top companies pay in hardware costs per unit. Today, the new performance dynamics and cost have extended the range of viable applications.



Applications of FPGA

Some More Common FPGA Applications are: Aerospace and Defense, Medical Electronics, ASIC Prototyping, Audio, Automotive, Broadcast, Consumer Electronics, Distributed Monetary Systems, Data Center, High Performance Computing, Industrial, Medical, Scientific Instruments, Security systems, Video & Image Processing, Wired Communications, Wireless Communications.