CHAP 3:

1)What is the work of CPU?

ANS) – Fetch instructions

– Interpret instructions

– Fetch data

– Process data

– Write data

2) Name 3 system Bus

Ans) a. control bus ,

b. data bus ,

c. address bus.

3) components of CPU

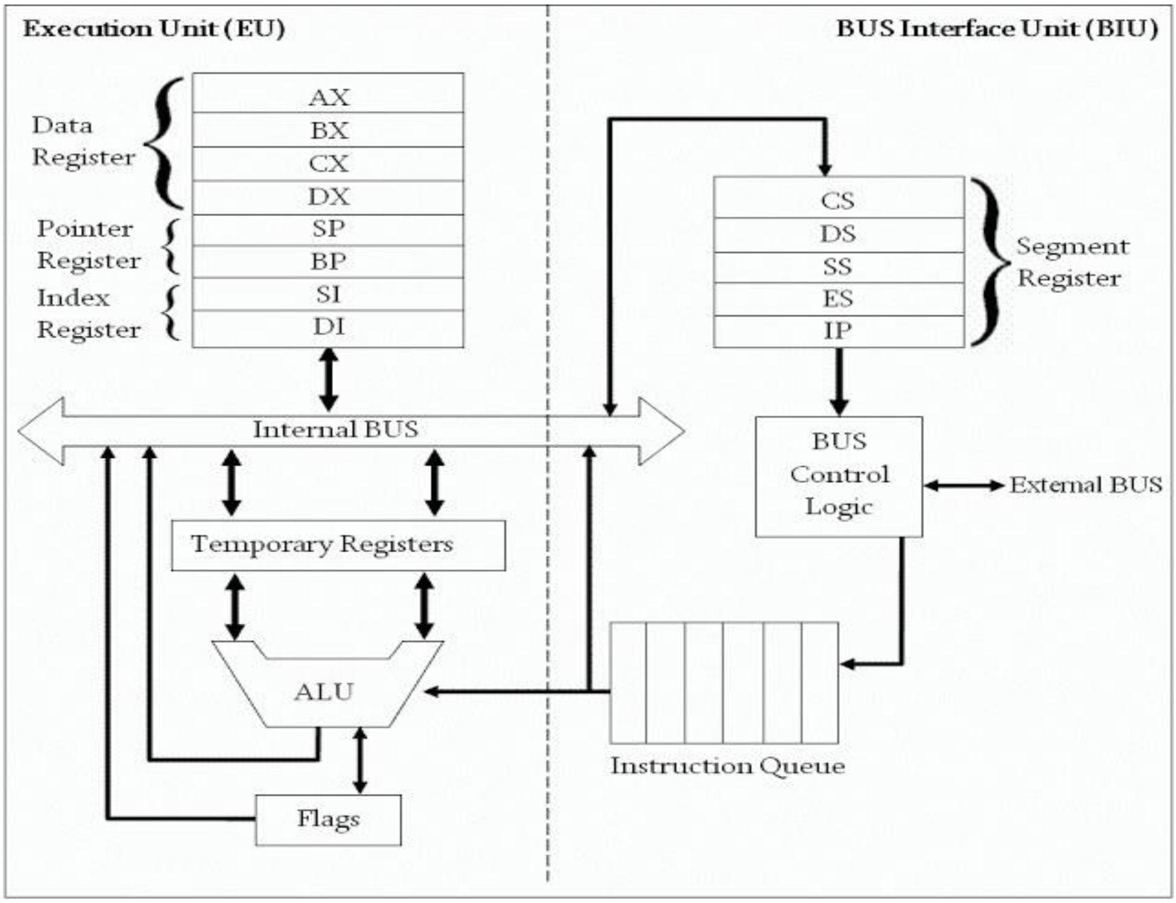
Ans) ALU , Register , control unit

4)Types of registers

Ans) User Visible Registers , Control and status register

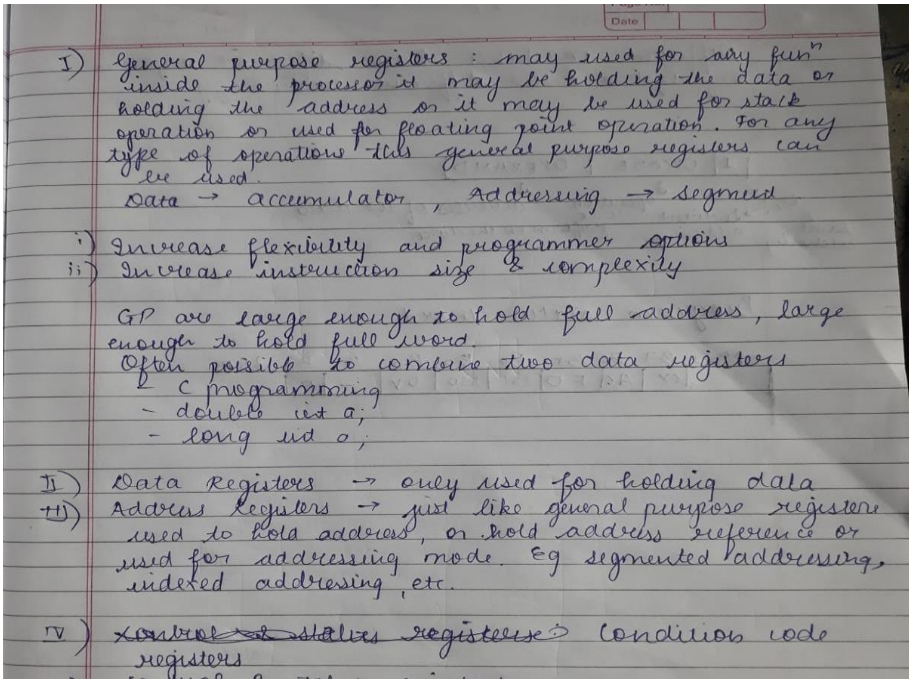
5) What are registers

Ans) Registers CPU must have some working space (temporary storage) Called registers. Registers can be of 8 bit , 16 bit and 32 bit. Registers are made up of multiple flip-flops.



5) Types of User Visible Registers

Ans)



I) General purpose registers : may used for any funⁿ inside the processor it may be holding the data or holding the address or it may be used for stack operation or used for floating point operation. For any type of operations this general purpose registers can be used.

Data → accumulator , Addressing → segment

i) Increase flexibility and programmer options
ii) Increase instruction size & complexity

GP are large enough to hold full address, large enough to hold full word.
Often possible to combine two data registers
- C programming
- double int a;
- long int a;

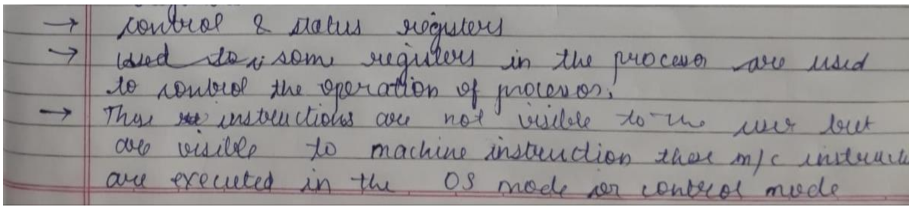
II) Data Registers → only used for holding data
III) Address Registers → just like general purpose registers used to hold address, or hold address reference or used for addressing mode. Eg segmented addressing, indexed addressing, etc.

IV) ~~control & status registerse~~ ⇒ condition code registers

Condition Code Registers(Flag Reg)

- Sets of individual bits – e.g. result of last operation was zero
- Can be read (implicitly) by programs – e.g. Jump if zero
- Can not (usually) be set by programs

2nd Type



→ control & status registers

→ Used ~~to~~, som registers in the processor are used to control the operation of processor.

→ These ~~se~~ instructions are not visible to the user but are visible to machine instruction these m/c instructi are executed in the OS mode or control mode

Control & Status Registers

- **Memory Address Register (MAR)**

– Connected to address bus

– Specifies address for read or write op

- **Memory Buffer Register (MBR)**

– Connected to data bus

– Holds data to write or last data read

- **Program Counter (PC)**

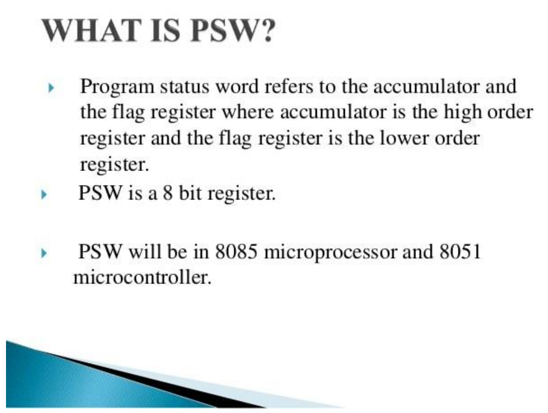
– Holds address of next instruction to be fetched

- **Instruction Register (IR)**

– Holds last instruction fetched/current instruction being executed

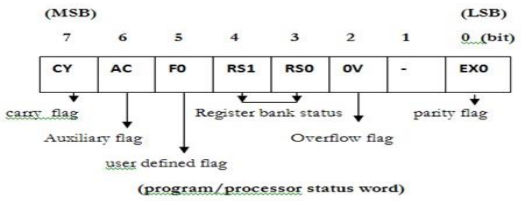
6)Program status word

Ans)

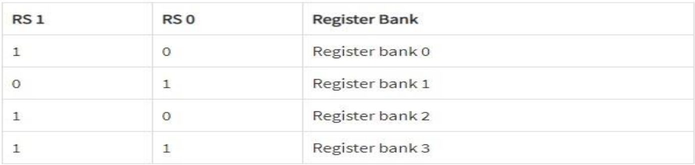


WHAT IS PSW?

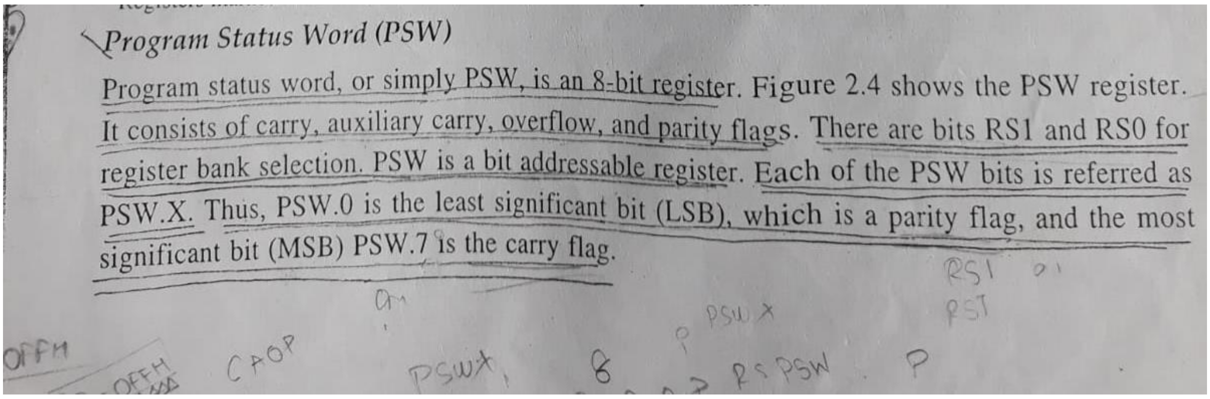
- Program status word refers to the accumulator and the flag register where accumulator is the high order register and the flag register is the lower order register.
- PSW is a 8 bit register.
- PSW will be in 8085 microprocessor and 8051 microcontroller.



(program/processor status word)



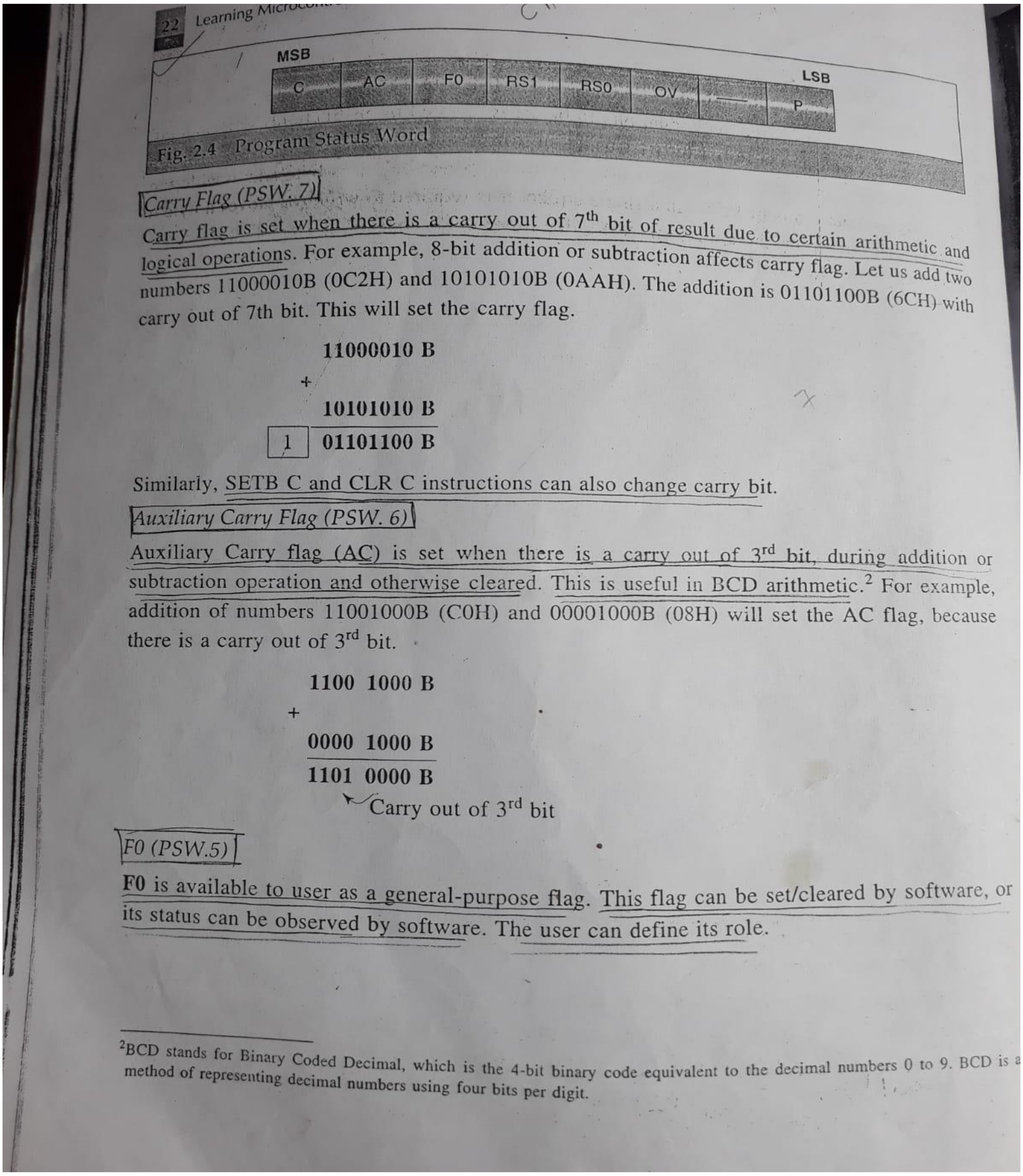
RS 1	RS 0	Register Bank
1	0	Register bank 0
0	1	Register bank 1
1	0	Register bank 2
1	1	Register bank 3



Program Status Word (PSW)

Program status word, or simply PSW, is an 8-bit register. Figure 2.4 shows the PSW register. It consists of carry, auxiliary carry, overflow, and parity flags. There are bits RS1 and RS0 for register bank selection. PSW is a bit addressable register. Each of the PSW bits is referred as PSW.X. Thus, PSW.0 is the least significant bit (LSB), which is a parity flag, and the most significant bit (MSB) PSW.7 is the carry flag.

RS1 01
RST
PSW X
0FFH
0FFH
CAOP
PSWx
8
RS PSW
P



MSB							LSB
C	AC	F0	RS1	RS0	OV	—	P

Fig. 2.4 Program Status Word

Carry Flag (PSW. 7)

Carry flag is set when there is a carry out of 7th bit of result due to certain arithmetic and logical operations. For example, 8-bit addition or subtraction affects carry flag. Let us add two numbers 11000010B (0C2H) and 10101010B (0AAH). The addition is 01101100B (6CH) with carry out of 7th bit. This will set the carry flag.

$$\begin{array}{rr} & \mathbf{11000010\ B} \\ + & \\ & \mathbf{10101010\ B} \\ \hline \boxed{1} & \mathbf{01101100\ B} \end{array}$$

Similarly, SETB C and CLR C instructions can also change carry bit.

Auxiliary Carry Flag (PSW. 6)

Auxiliary Carry flag (AC) is set when there is a carry out of 3rd bit, during addition or subtraction operation and otherwise cleared. This is useful in BCD arithmetic.[2] For example, addition of numbers 11001000B (C0H) and 00001000B (08H) will set the AC flag, because there is a carry out of 3rd bit.

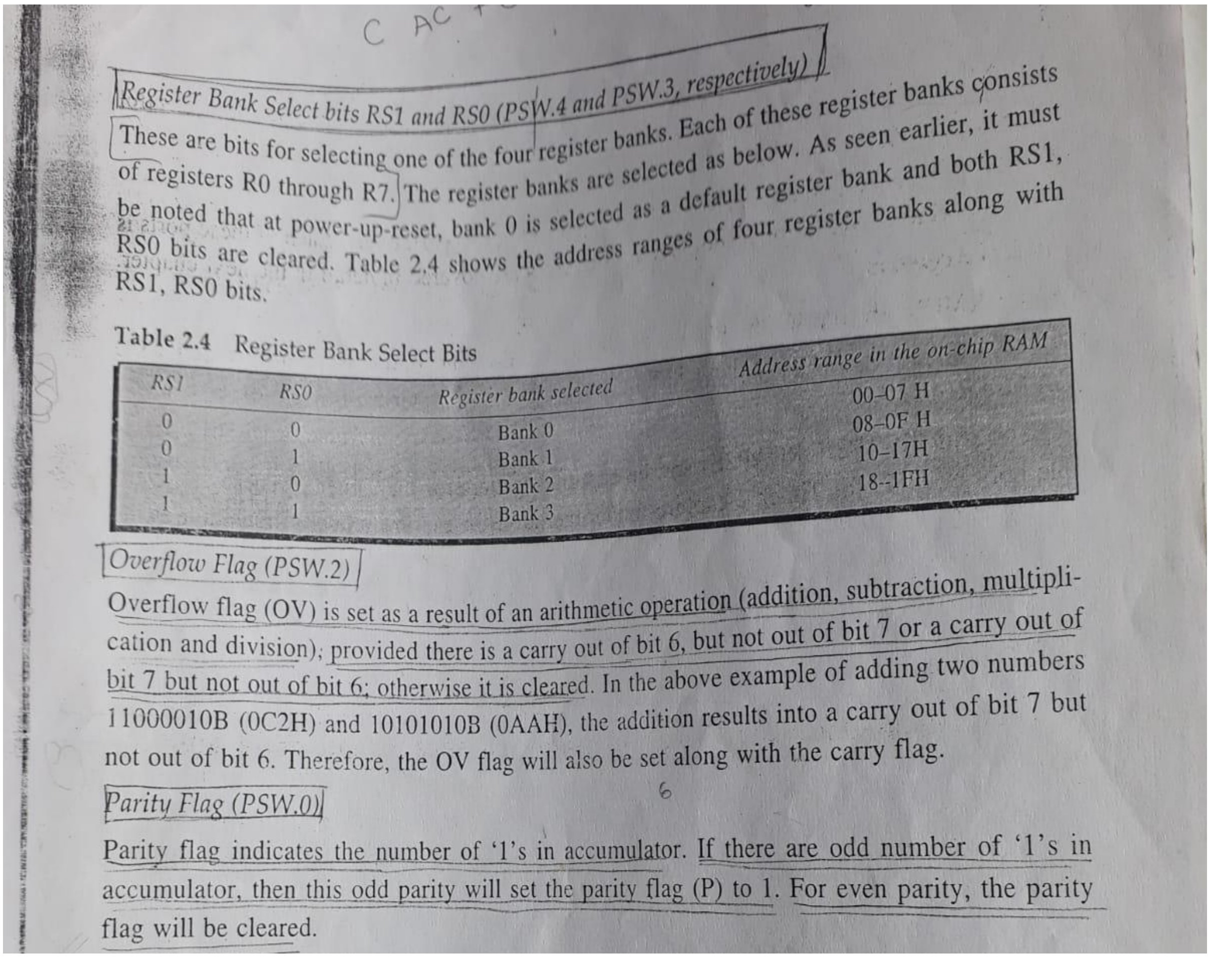
$$\begin{array}{rr} & \mathbf{1100\ \ 1000\ B} \\ + & \\ & \mathbf{0000\ \ 1000\ B} \\ \hline & \mathbf{1101\ \ 0000\ B} \end{array}$$

Carry out of 3rd bit

F0 (PSW.5)

F0 is available to user as a general-purpose flag. This flag can be set/cleared by software, or its status can be observed by software. The user can define its role.

[2] BCD stands for Binary Coded Decimal, which is the 4-bit binary code equivalent to the decimal numbers 0 to 9. BCD is a method of representing decimal numbers using four bits per digit.



C AC

Register Bank Select bits RS1 and RS0 (PSW.4 and PSW.3, respectively)

These are bits for selecting one of the four register banks. Each of these register banks consists of registers R0 through R7. The register banks are selected as below. As seen earlier, it must be noted that at power-up-reset, bank 0 is selected as a default register bank and both RS1, RS0 bits are cleared. Table 2.4 shows the address ranges of four register banks along with RS1, RS0 bits.

Table 2.4 Register Bank Select Bits

RS1	*RS0*	*Register bank selected*	*Address range in the on-chip RAM*
0	0	Bank 0	00–07 H
0	1	Bank 1	08–0F H
1	0	Bank 2	10–17H
1	1	Bank 3	18--1FH

Overflow Flag (PSW.2)

Overflow flag (OV) is set as a result of an arithmetic operation (addition, subtraction, multiplication and division); provided there is a carry out of bit 6, but not out of bit 7 or a carry out of bit 7 but not out of bit 6; otherwise it is cleared. In the above example of adding two numbers 11000010B (0C2H) and 10101010B (0AAH), the addition results into a carry out of bit 7 but not out of bit 6. Therefore, the OV flag will also be set along with the carry flag.

Parity Flag (PSW.0)

6

Parity flag indicates the number of '1's in accumulator. If there are odd number of '1's in accumulator, then this odd parity will set the parity flag (P) to 1. For even parity, the parity flag will be cleared.

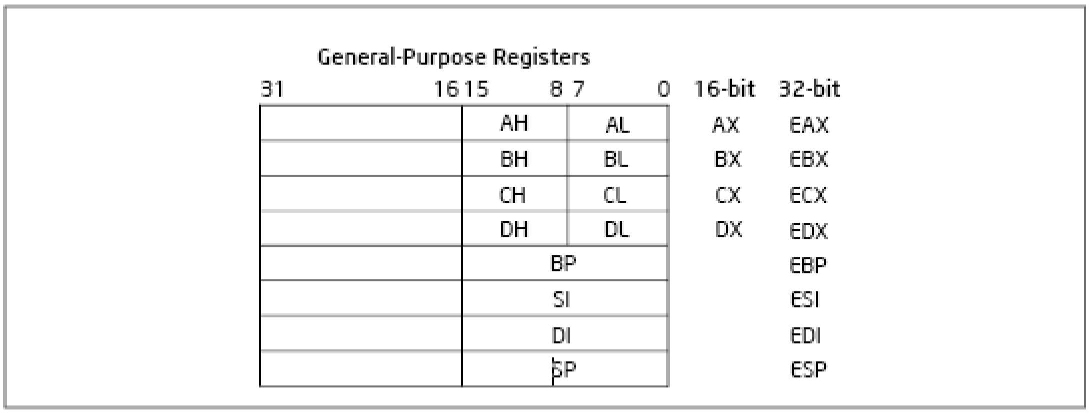
7) General Purpose Register

ANS) AX is the primary accumulator; it is used in input/output and most arithmetic instructions. For example, in multiplication operation, one operand is stored in EAX or AX or AL register according to the size of the operand.

BX is known as the base register, as it could be used in indexed addressing.

CX is known as the count register, as the ECX, CX registers store the loop count in iterative operations.

DX is known as the data register. It is also used in input/output operations. It is also used with AX register along with DX for multiply and divide operations involving large values



General-Purpose Registers

31 16	15 8	7 0	16-bit	32-bit
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

Figure 3-5. Alternate General-Purpose Register Names

9) Pointer Register

ANS) •Instruction Pointer (IP) – The 16-bit IP register stores the offset address of the next instruction to be executed. IP in association with the CS register (as CS : IP) gives the complete address of the current instruction in the code segment.

• **Stack Pointer (SP)** – The 16-bit SP register provides the offset value within the program stack. SP in association with the SS register (SS:SP) refers to be current position of data or address within the program stack.

• **Base Pointer (BP)** – The 16-bit BP register mainly helps in referencing the parameter variables passed to a subroutine. The address in SS register is combined with the offset in BP to get the location of the parameter. BP can also be combined with DI and SI as base register for special addressing.

10) Index Registers

ANS) SI and DI, are used for indexed addressing and sometimes used in addition and subtraction. There are two sets of index pointers

– Source Index (SI) – It is used as source index for string operations.

–Destination Index (DI) – It is used as destination index for string operations.

11) Control Registers

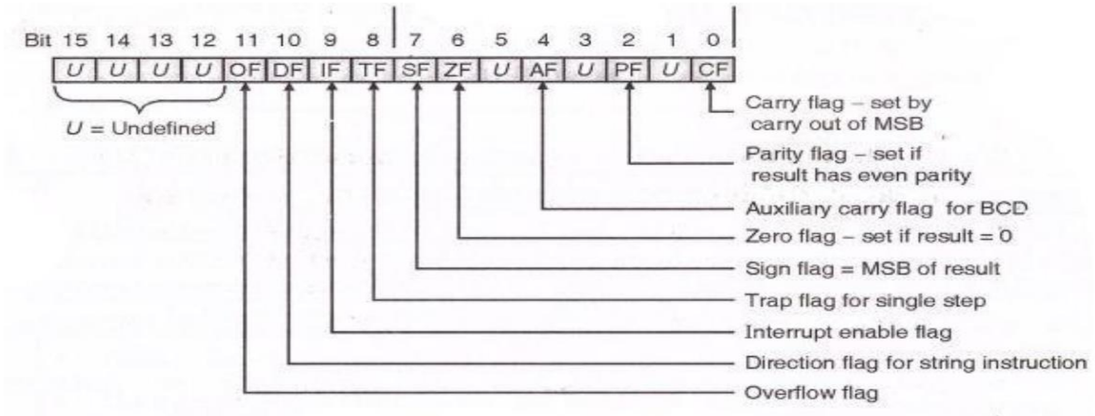
ANS) • The 32-bit instruction pointer register and the 32-bit flags register combined are considered as the control registers.

• Many instructions involve comparisons and mathematical calculations and change the status of the flags and some other conditional instructions test the value of these status flags to take the control flow to other location.

• The common flag bits are:

This register is 16 bits wide. Its successors, the EFLAGS and RFLAGS registers, are 32 bits and 64 bits wide, respectively.

FLAG REGISTER 8086



Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
U U U U OF DF IF TF SF ZF U AF U PF U CF
U = Undefined
Carry flag – set by carry out of MSB
Parity flag – set if result has even parity
Auxiliary carry flag for BCD
Zero flag – set if result = 0
Sign flag = MSB of result
Trap flag for single step
Interrupt enable flag
Direction flag for string instruction
Overflow flag

8086 flag register format

•**Overflow Flag (OF)** − It indicates the overflow of a high-order bit (leftmost bit) of data after a signed arithmetic operation.

•**Direction Flag (DF)** − It determines left or right direction for moving or comparing string data. When the DF value is 0, the string operation takes left-to-right direction and when the value is set to 1, the string operation takes right-to-left direction.

• **Trap Flag (TF)** − It allows setting the operation of the processor in single-step mode. The DEBUG program we used sets the trap flag, so we could step through the execution one instruction at a time.

• **Sign Flag (SF)** − It shows the sign of the result of an arithmetic operation. This flag is set according to the sign of a data item following the arithmetic operation. The sign is indicated by the high-order of leftmost bit. A positive result clears the value of SF to 0 and negative result sets it to 1.

• **Zero Flag (ZF)** − It indicates the result of an arithmetic or comparison operation. A nonzero result clears the zero flag to 0, and a zero result sets it to 1.

• **Interrupt Flag (IF)** − It determines whether the external interrupts like keyboard entry, etc., are to be ignored or processed. It disables the external interrupt when the value is 0 and enables interrupts when set to 1.

- **Auxiliary Carry Flag (AF)** − It contains the carry from bit 3 to bit 4 following an arithmetic operation; used for specialized arithmetic. The AF is set when a 1- byte arithmetic operation causes a carry from bit 3 into bit 4.

- **Parity Flag (PF)** − It indicates the total number of 1-bits in the result obtained from an arithmetic operation. An even number of 1-bits clears the parity flag to 0 and an odd number of 1-bits sets the parity flag to 1.

- **Carry Flag (CF)** − It contains the carry of 0 or 1 from a high-order bit (leftmost) after an arithmetic operation. It also stores the contents of last bit of a shift or rotate operation.

12) Segment register

Ans) Segments are specific areas defined in a program for containing data, code and

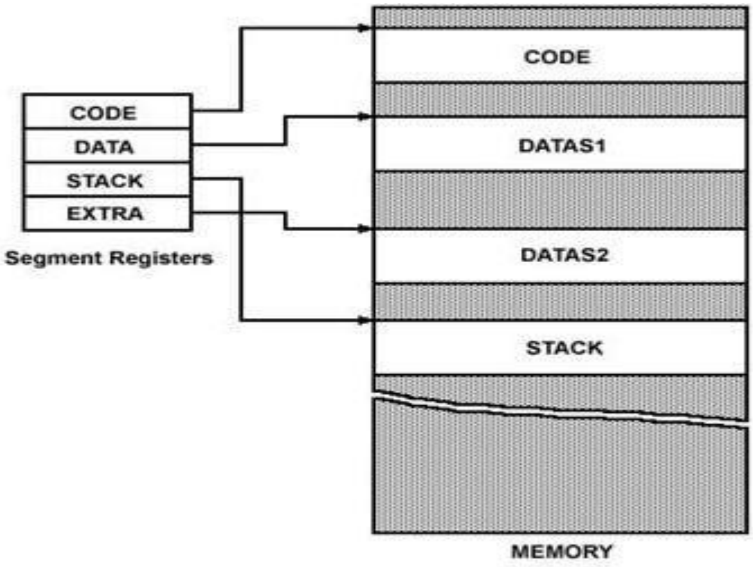
stack.

There are three main segments –

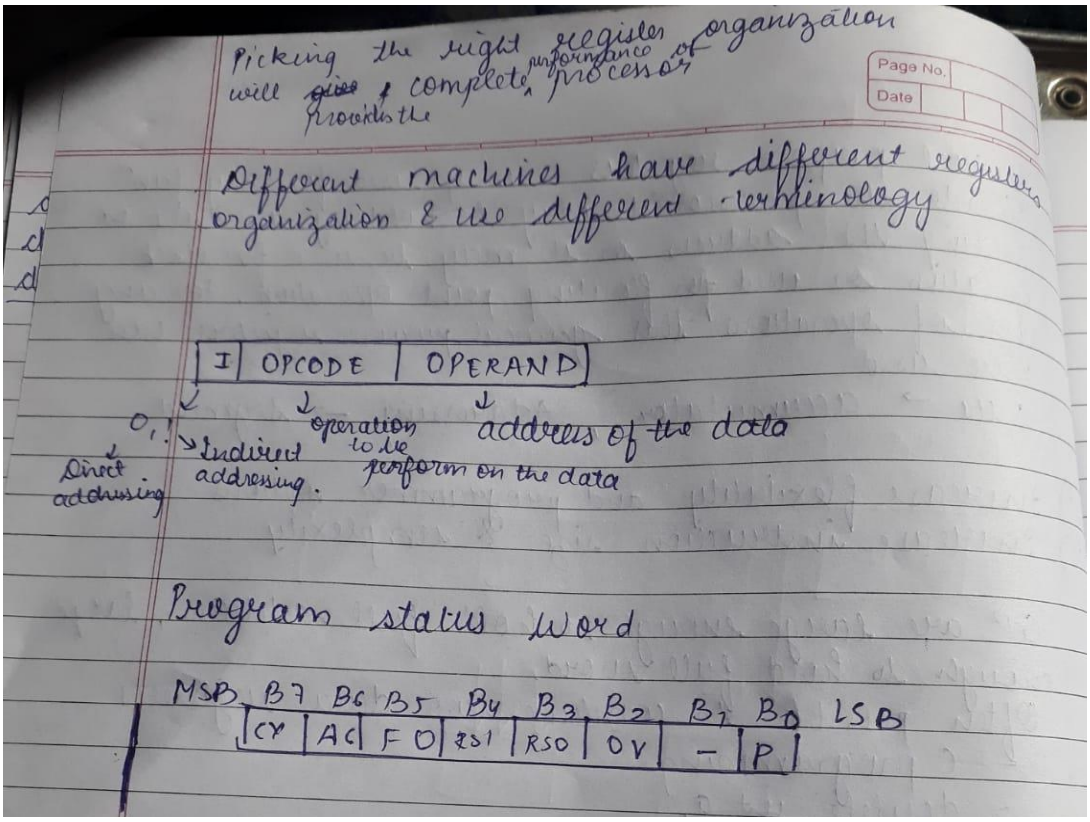
- **Code Segment** − It contains all the instructions to be executed. A 16-bit Code Segment register or CS register stores the starting address of the code segment.

- **Data Segment(DS,ES)** − It contains data, constants and work areas. A 16-bit Data Segment register or DS register stores the starting address of the data segment.

- **Stack Segment** − It contains data and return addresses of procedures or subroutines. It is implemented as a 'stack' data structure. The Stack Segment register or SS register stores the starting address of the stack.



13) Instruction Format (opcode , operand)
ANS)



Picking the right register organization will ~~give~~ provides the complete performance of processor

Different machines have different registers organization & use different terminology

I	OPCODE	OPERAND

0, 1 → Direct addressing → Indirect addressing.

operation to be perform on the data

address of the data

Program status word

MSB B7	B6	B5	B4	B3	B2	B1	B0 LSB
CY	AC	F0	RS1	RS0	OV	–	P

Instruction Length

- Affected by and affects:
  - – Memory size
  - – Memory organization
  - – Bus structure
  - – CPU complexity
  - – CPU speed

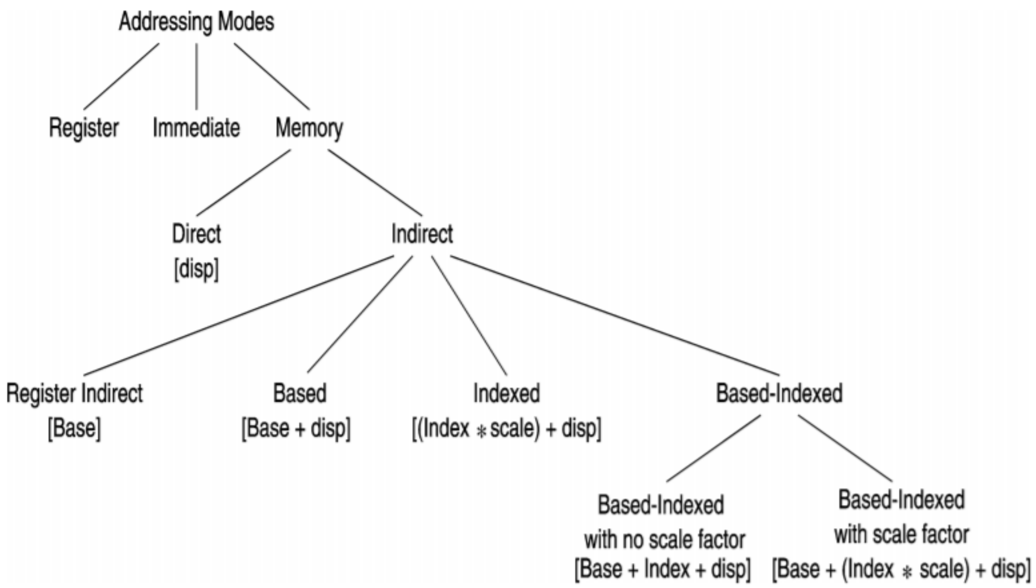
User wants more opcodes, operands, addressing modes , address range

14)7 Addressing Modes

Ans) • Immediate

- Direct
- Indirect
- Register
- Register Indirect
- Displacement (Indexed)
- Stack

Pentium Addressing Modes (32-bit Addresses)



Addressing Modes

The term ***addressing modes*** refers to the way in which the operand of an instruction is specified. Information contained in the instruction code is the value of the operand or the address of the result/operand. Following are the main addressing modes that are used on various platforms and architectures.

Effective address or Offset: An offset is determined by adding any combination of three address elements: displacement, base and index.

Displacement: It is an 8 bit or 16 bit immediate value given in the instruction.

Base: Contents of base register, BX or BP.

Index: Content of index register SI or DI.

Immediate addressing mode

The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.

Example

MOV CX, 4929 H, ADD AX, 2387 H, MOV AL, FFH

Register addressing mode

It means that the register is the source of an operand for an instruction.

Example MOV CX, AX   ; copies the contents of the 16-bit AX register into

; the 16-bit CX register),

ADD BX, AX

Direct addressing mode

The addressing mode in which the effective address (Effective Address is the location where operand is present) of the memory location is written directly in the instruction.

Example

MOV AX, [1592H], MOV AL, [0300H]

Register indirect addressing mode

This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI & SI.

Example

MOV AX, [BX] ; Suppose the register BX contains 4895H, then the contents

; 4895H are moved to AX

ADD CX, {BX}

Based addressing mode

In this addressing mode, the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement.

Example

MOV DX, [BX+04], ADD CL, [BX+08]

Indexed addressing mode

In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements.

Example

MOV BX, [SI+16], ADD AL, [DI+16]

Based-index addressing mode

In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an Index register.

Example

ADD CX, [AX+SI], MOV AX, [AX+DI]

Based indexed with displacement mode

In this addressing mode, the operands offset is computed by adding the base register contents. An Index registers contents and 8 or 16-bit displacement.

Example

MOV AX, [BX+DI+08], ADD CX, [BX+SI+16]

Stack Addressing

Operand is (implicitly) on top of stack.

e.g. – ADD Pop top two items from stack and add then push.

Indirect Addressing Mode-

In this addressing mode,

- The address field of the instruction specifies the address of memory location that contains the effective address of the operand.
- Two references to memory are required to fetch the operand.
- Indirect addressing is a scheme in which the address specifies which memory word or register contains not the operand but the address of the operand.

For example:

1) LOAD R1, @100
Load the content of memory address stored at memory address 100 to the register R1.

2) LOAD R1, @R2
Load the content of the memory address stored at register R2 to register R1.

15) Instruction cycle

Ans)

Registers Involved In Each Instruction Cycle:

Memory address registers(MAR) : It is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.

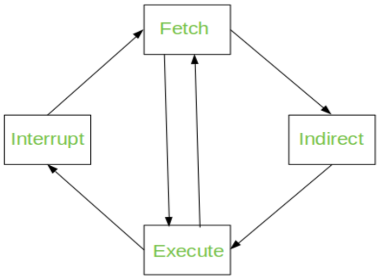
Memory Buffer Register(MBR) : It is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from the memory.

Program Counter(PC) : Holds the address of the next instruction to be fetched.

Instruction Register(IR) : Holds the last instruction fetched.

The Instruction Cycle –

Each phase of Instruction Cycle can be decomposed into a sequence of elementary micro-operations.
In the above examples, there is one sequence each for the Fetch, Indirect, Execute and Interrupt Cycles.



The Instruction Cycle

The Indirect Cycle is always followed by the Execute Cycle. The Interrupt Cycle is always followed by the Fetch Cycle. For both fetch and execute cycles, the next cycle depends on the state of the system.

We assumed a new 2-bit register called Instruction Cycle Code (ICC). The ICC designates the state of processor in terms of which portion of the cycle it is in:-

00 : Fetch Cycle
01 : Indirect Cycle
10 : Execute Cycle
11 : Interrupt Cycle

At the end of the each cycles, the ICC is set appropriately.

The above flowchart of Instruction Cycle describes the complete sequence of micro-operations, depending only on the instruction sequence and the interrupt pattern(this is a simplified example).

The operation of the processor is described as the performance of a sequence of micro-operation.

Different Instruction Cycles:

The Fetch Cycle –

At the beginning of the fetch cycle, the address of the next instruction to be executed is in the Program Counter(PC).

Step 1: The address in the program counter is moved to the memory address register(MAR), as this is the only register which is connected to address lines of the system bus.

Step 2: The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register(MBR). Program counter is incremented by one, to get ready for the next instruction.(These two action can be performed simultaneously to save time)

Step 3: The content of the MBR is moved to the instruction register(IR).

The Indirect Cycles –

Step 1: The address field of the instruction is transferred to the MAR. This is used to fetch the address of the operand.

Step 2: The address field of the IR is updated from the MBR.(So that it now contains a direct addressing rather than indirect addressing)

Step 3: The IR is now in the state, as if indirect addressing has not been occurred.

Note: Now IR is ready for the execute cycle, but it skips that cycle for a moment to consider the Interrupt Cycle .

The Execute Cycle

The other three cycles(Fetch, Indirect and Interrupt) are simple and predictable. Each of them requires simple, small and fixed sequence of micro-operation. In each case same micro-operation are repeated each time around.

Execute Cycle is different from them. Like, for a machine with N different opcodes there are N different sequence of micro-operations that can occur.

Lets take an hypothetical example :-

consider an add instruction:

Here, this instruction adds the content of location X to register R. Corresponding micro-operation will be:-

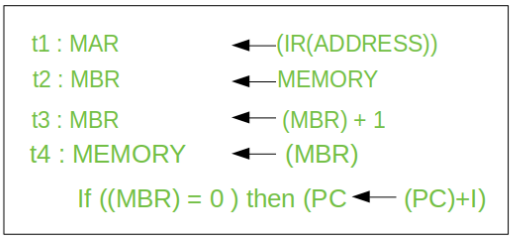
We begin with the IR containing the ADD instruction.

Step 1: The address portion of IR is loaded into the MAR.

Step 2: The address field of the IR is updated from the MBR, so the reference memory location is read.

Step 3: Now, the contents of R and MBR are added by the ALU.

Lets take a complex example :-



```
t1 : MAR          ←(IR(ADDRESS))
t2 : MBR          ←MEMORY
t3 : MBR          ← (MBR) + 1
t4 : MEMORY       ← (MBR)
     If ((MBR) = 0 ) then (PC ← (PC)+I)
```

ISZ X

Here, the content of location X is incremented by 1.

If the result is 0, the next instruction will be skipped. Corresponding sequence of micro-operation will be :-

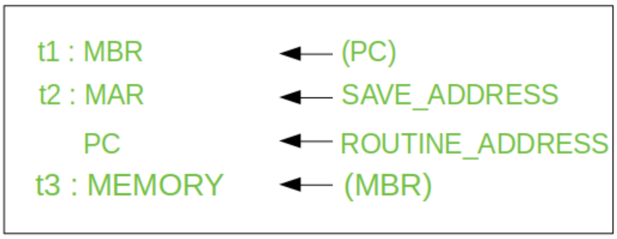
Here, the PC is incremented if (MBR) = 0.

This test (is MBR equal to zero or not) and action (PC is incremented by 1) can be implemented as one micro-operation.

Note : This test and action micro-operation can be performed during the same time unit during which the updated value MBR is stored back to memory.

The Interrupt Cycle:

Lets take a sequence of micro-operation:-



```
t1 : MBR          ← (PC)
t2 : MAR          ← SAVE_ADDRESS
     PC           ← ROUTINE_ADDRESS
t3 : MEMORY       ← (MBR)
```

Step 1: Contents of the PC is transferred to the MBR, so that they can be saved for return.

Step 2: MAR is loaded with the address at which the contents of the PC are to be saved.

PC is loaded with the address of the start of the interrupt-processing routine.

Step 3: MBR, containing the old value of PC, is stored in memory.

16) Functional Requirements(of Control Unit)

Ans) -Define basic elements of processor

-Describe micro-operations processor performs

-Determine functions control unit must perform

17) CPU (Control Program Unit)

Ans) • All the control unit does is generate a set of control signals

• Each control signal is on or off

• Represent each control signal by a bit

• Have a control word for each micro-operation

• Have a sequence of control words for each machine code instruction

• Add an address to specify the next micro-instruction, depending on Conditions

18) Microprogram control unit function

Ans) Advantages and Disadvantages of Microprogramming

- Simplifies design of control unit
  - Cheaper
  - Less error-prone
- Slower

Tasks Done By Microprogrammed Control Unit

- Microinstruction sequencing
- Microinstruction execution
- Must consider both together

19) Micro-Instruction Types

Ans) Micro-instruction Types -

 Each micro-instruction specifies many different micro-operations to be performed in parallel

◦ (horizontal micro-programming)

- Wide memory word
- High degree of parallel operations possible
- Little encoding of control information

 Each micro-instruction specifies single (or few) micro-operations to be performed

◦ (vertical micro-programming)

- Width is narrow

- Limited ability to express parallelism
- Considerable encoding of control information requires external memory word decoder to identify the exact control line being manipulated.

20) Application of Micro-Processing

Ans) ın Operating system: Microprograms can be used to implement some of the primitives of operating system. This simplifies operation system implementation and also improves the performance of the operating system.

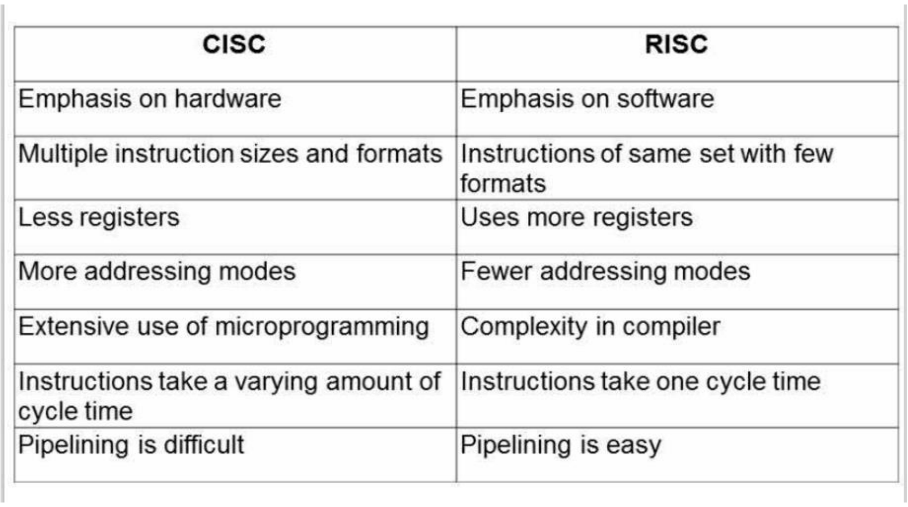
In High-Level Language support: In High-Level language various sub functions and data types can be implemented using microprogramming. This makes compilation into an efficient machine language form possible.

In Emulation: Emulation refers to the use of a microprogram on one machine to execute programs originally written for another machine. This is used widely as an aid for users in migrating from one computer to another.

21) Reduced instruction set computer (RISC) / Complex instruction set computer (CISC)

ANS)

- Hardware fused with software (Intel v/s Apple)
- Intel’s hardware oriented approach is termed as CISC while that of Apple is RISC .
- Instruction Set Architecture- Interface to allow easy communication between the programmer and the hardware.
- ISA- execution of data, copying data, deleting it, editing
- Instruction Set , Addressing Modes



CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

22) RISC Pipelining

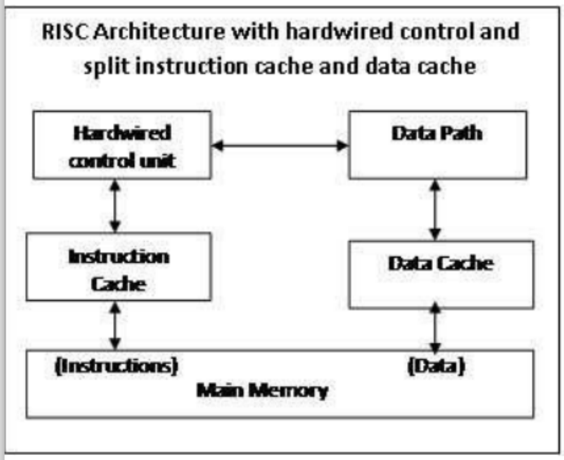
Ans) RISC Pipelining

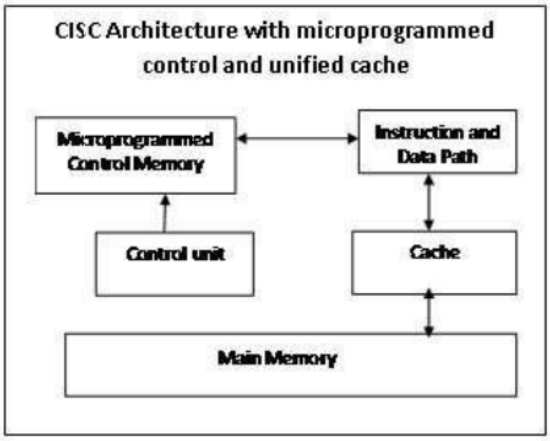
- Most instructions are register to register
- Two phases of execution, I E
    - – I: Instruction fetch
    - – E: Execute
- ALU operation with register input and output
- For load and store (memory), I E D
    - – I: Instruction fetch
    - – E: Execute
- Calculate memory address
    - – D: Memory
- Register to memory or memory to register operation

Types of piplining :-

1) Sequential Pipelining
2) Two – staged pipelining
3) Three – staged pipelining

CISC / RISC ARCHITECTURE




CISC Architecture with microprogrammed control and unified cache
Microprogrammed Control Memory
Instruction and Data Path
Control unit
Cache
Main Memory

RISC Architecture

- 9 functional units interconnected by multiple data paths with width ranging from 32-128 bits
- All internal- external buses are 32 bit wide
- Separate instruction (4KB) and data cache (8KB).
- MMU- implements paged virtual memory structure
- RISC integer unit executes load, store, fetch etc.
- 2 floating point units, multiplier unit and adder unit
- Graphics unit to support 3D drawing

23) Scalable Processor Architecture (SPARC)

ANS)

- Designed to optimize compilers and pipelined hardware implementations.
- Offers fast execution rates.
- Engineered at Sun Microsystems in 1985
  - Based on RISC I & II which were developed at Univ. of Cal at Berkeley.

Features of SPARC

- Performance and Economy
  - Simplified instruction set
  - Higher number of instructions with fewer transistors
- Scalability
  - Flexible integration of cache, memory and FPUs
- Open Architecture
  - Compatible technology to multiple vendors
  - Now allow access to CPU component techniques
  - Complete set of development tool available for h/w & s/w