SCHEME OF EVALUATION

(Scoring Indicators)

Revision:2015

Course Titles: MICROPROCESSORS AND INTERFACING (5131)

Qst No.	Scoring Indicator	Split up score	Sub Total	Total
I	PART A			
1.	It stands for address enable latch and is available at pin 25. A positive pulse is generated each time the processor begins any operation. This signal indicates the availability of a valid address on the address/data lines.	2		
2.	*use **AAM** only after executing a mul instruction between two BCD digits (unpacked). mul stores the result in the AX register. or ***AAM** unpacks the AL result by dividing AL by 10, stores the quotient (most-significant digit) in AH, and stores the remainder (least-significant digit) in AL	2		
3.	8279	2		
4.	*Set of instructions represented by a single statement *An **interrupt** is a condition that halts the microprocessor temporarily to work on a different task and then return to its previous task. This halt allows peripheral devices to access the microprocessor.	2		
5.	**SSE** is a processor technology that enables single instruction multiple data.**SSE** enables the instructions to handle multiple data elements. **SSE** (SIMD technology) is used in applications like 3D graphics and making processing much faster.	2	10	10
1.	PART B • It is a 16-bit Microprocessor • 20 address lines • 16 data lines • 1MB storage. • It uses two stages of pipelining, i.e. Fetch Stage and Execute Stage, which improves performance. • Fetch stage can prefetch up to 6 bytes of instructions and stores them in the queue. • Execute stage executes these instructions. • It has 256 vectored interrupts. • It consists of 29,000 transistors • It has an instruction queue, which is capable of storing six instruction bytes from the memory resulting in faster processing. (any 6)	1*6	6	6

2.

- Micro processor is designed to function as the CPU of a computer.
- It fetches each instruction from memory, decode and execute
- It processes the data as the instruction tells.
- It performs arithmetic and logical operation
- Data is retrieved from memory or taken from input device
- Processed data is stored in memory and delivered to output device.
  (if answer is block diagram with input, cpu, memory and output device give 6 marks)

6 | 6 | 6

3.

- **Macro** is a set of instructions grouped under a single unit. It is another method for implementing modular programming in the 8086 microprocessors (The first one was using Procedures). — 1
- the processor generates the code in the program every time whenever and wherever a call to the **Macro** is made.
- assembler directives: **MACRO** (used after the name of Macro before starting the body of the Macro) and **ENDM** (at the end of the Macro). All the instructions that belong to the Macro lie within these two assembler directives. — 2
- Syntax: Macro_name MACRO [list of parameters]

  Instruction 1

  ENDM
- Macro call- macroname[parameter] — 2 | 6 | 6

- It avoids overhead time involved in calling and returning
- Used in small instructions to execute — 1

4.

REP/REPE/REPZ/REPNE/REPNZ: These instructions repeat until specified condition exists.

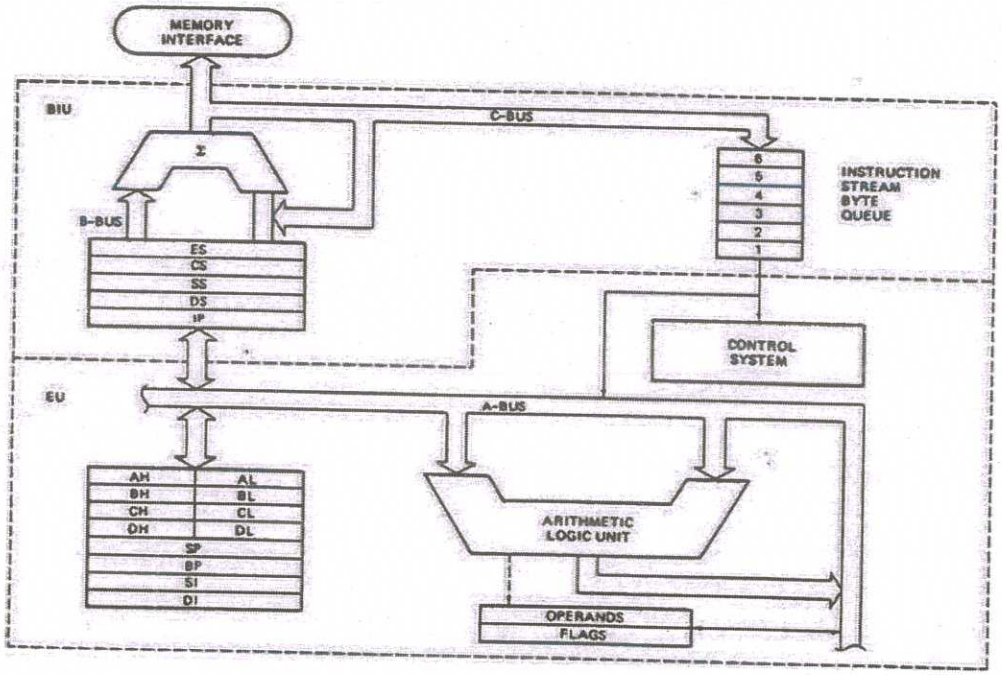
REP: exit when cx=0

REPE/REPZ: exit when cx=0 and ZF=0

REPNE/REPNZ: exit when cx=0, and ZF=1

- **MOVS/MOVSB/MOVSW:** This instruction copies a byte or word from a location in the data segment to a location in the extra segment.
- **CMPS/CMPSB/CMPSW:** The CMPS instruction can be used to compare a byte in one

	string with a byte in another string or to compare a word in one string with a word in another string. SI is used to hold the offset of a byte or word in the source string and DI is used to hold the offset of a byte or a word in the other string. • **SCAS/SCASB/SCASW:** Compares the contents of the AL, AX or EAX register with the memory byte, word or double word pointed to by DI relative to ES and changes the flags accordingly. After the comparison is made, the DI register is automatically updated • **LODS/LODSB/LODSW:** Loads the AL, AX or EAX registers with the content of the memory byte, word or double word pointed to by SI relative to DS. Then SI is incremented or decremented based on DF • **STOS/STOSB/STOSW:** Transfers the contents of the AL, AX or EAX registers to the memory byte, word or double word pointed to by DI relative to ES. After the transfer is made, the DI register is automatically updated (any 3 with explanation/example)	3*2	6	6
5.	There are different types of interrupts in 8086. All the interrupts cannot be executed at the same time when it occurs. So there must be an order of preference for its execution .This is called interrupt priority. Priority of interrupt in order is 1. Divide by error , INT n, INT 0 2. NMI 3. INTR 4. Single step	2 4	6	6
6.	Features of 80386 • 32 bit processor • 32 bit data and address bus • 4 GB physical Memory • Multitasking and protection capability • Virtual memory addressability 64TB • Offers 3 stage pipeline • Real, protected and virtual addressing mode • Operating frequency of 16MHZ,20MHZ,25MHZ….. (Any 6)	6*1	6	6
7.	Hyper Threading and MMX technology *hyperthreading : A high-performance computing architecture that simulates some degree of overlap in executing two or more independent sets of instructions. *It is a feature of certain intel chips that makes one physical CPU appear as two logical CPUs. *It increases the processing performance of the system.	3	3	

	*MMX is multimedia extended technology, *it is a Pentium microprocessor from Intel that is designed to run faster when playing multimedia applications. *According to Intel, a PC with an MMX microprocessor runs a multimedia application up to 60% faster than one with a microprocessor having the same clock speed but without MMX.	3	3	6
IIIa.	PARTC  	4		
	8086 Microprocessor is divided into two functional units, **EU** (Execution Unit) and **BIU** (Bus Interface Unit).	1		
	EU (Execution Unit) Execution unit gives instructions to BIU stating from where to fetch the data and then decode and execute those instructions. Its function is to control operations on data using the instruction decoder & ALU. EU has no direct connection with system buses .It performs operations over data through BIU.	1		
	ALU: handles all arithmetic and logical operations, like +, −, ×, /, OR, AND, NOT			

operations.

Flag Register:It is a 16-bit register . It changes its status according to the result stored in the accumulator. It has 9 flags and they are divided into 2 groups – Conditional Flags and Control Flags. (2)

Conditional Flags

It represents the result of the last arithmetic or logical instruction executed. Following is the list of conditional flags –

- **Carry flag** – This flag indicates an overflow condition for arithmetic operations.
- **Auxiliary flag** – When an operation is performed at ALU, it results in a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), then this flag is set, i.e. carry given by D3 bit to D4 is AF flag. The processor uses this flag to perform binary to BCD conversion.
- **Parity flag** – This flag is used to indicate the parity of the result, i.e. when the lower order 8-bits of the result contains even number of 1's, then the Parity Flag is set. For odd number of 1's, the Parity Flag is reset.
- **Zero flag** – This flag is set to 1 when the result of arithmetic or logical operation is zero else it is set to 0.
- **Sign flag** – This flag holds the sign of the result, i.e. when the result of the operation is negative, then the sign flag is set to 1 else set to 0.
- **Overflow flag** – This flag represents the result when the system capacity is exceeded.

Control Flags

Control flags controls the operations of the execution unit. Following is the list of control flags –

- **Trap flag** – It is used for single step control and allows the user to execute one instruction at a time for debugging.
- **Interrupt flag** – It is an interrupt enable/disable flag, It is set to 1 for interrupt enabled condition and set to 0 for interrupt disabled condition.
- **Direction flag** – It is used in string operation. When it is set then string bytes are accessed from the higher memory address to the lower memory address and vice-a-versa.

General purpose register (2)

There are 8 general purpose registers.The valid register pairs are AH and AL, BH and BL, CH and CL, and DH and DL. It is referred to the AX, BX, CX, and DX respectively.

- **AX register** – It is also known as accumulator register. It is used to store operands for arithmetic operations.
- **BX register** – It is used as a base register. It is used to store the starting base address

	of the memory area within the data segment. • **CX register** – It is referred to as counter. It is used in loop instruction to store the loop counter. • **DX register** – This register is used to hold I/O port address for I/O instruction. Stack pointer register It is a 16-bit register holds the address from the start of the segment to the memory location. BIU (Bus Interface Unit) BIU takes care of all data and addresses transfers on the buses for the EU like sending addresses, fetching instructions from the memory, reading data from the ports and the memory as well as writing data to the ports and the memory. EU has no direction connection with System Buses so this is possible with the BIU. EU and BIU are connected with the Internal Bus. It has the following functional parts – • **Instruction queue** –BIU gets upto 6 bytes of next instructions and stores them in the instruction queue. When EU executes instructions and is ready for its next instruction, then it simply reads the instruction from this instruction queue resulting in increased execution speed. • **Segment register** – BIU has 4 segment buses o **CS** – It stands for Code Segment. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored. o **DS** – It stands for Data Segment. It consists of data used by the program andis accessed in the data segment by an offset address or the content of other register that holds the offset address. o **SS** – It stands for Stack Segment. It handles memory to store data and addresses during execution. o **ES** – It stands for Extra Segment. ES is additional data segment, which is used by the string to hold the extra destination data. • **Instruction pointer** – It is a 16-bit register used to hold the address of the next instruction to be executed.	(1) (1) (1) (1) (1)	15	15

IVa.

Addressing modes are

*Immediate addressing mode:The addressing mode in which the data operand is a part of the instruction itself

Example

MOV CX, 4929 H

*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),
```

*Direct addressing mode:The addressing mode in which the effective address of the memory location is written directly in the instruction.

Example

MOV AX, [1592H]

*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
```

*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]

	*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]
	
	*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]
	
	*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]
	(Any 6)
		6*
		1 1/2
			9	9

IV b.	*In 8086 pin 33 is MN/MX, in maximum mode MX is low * It is a multi processor mode *Has High performance *ALE for the latch is given by bus controller * DT/R is also given by bus controller as there are more processors. *instead of control signals each processor generates status signals S0,S1 and S2 *RQ/GT is used for bus request by other processors. (or pin diagram of maximum mode)	6	6	6

Va.

Instructions to transfer the control 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.

3

Instructions to transfer the control 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

1*6

6+3

9

Vb.

- **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.

1*6 | 6 | 6

VIa.

ADD :
The add instruction adds the contents of the source operand to the destination operand.
Eg.
ADD AX, BX ; AX=AX+BX

3

MUL :Unsigned Multiplication Byte or Word
This instruction multiplies an unsigned byte or word by the contents of AL.
Eg.
MUL BH ; Result in (AX) (AL) x (BH)
MUL CX ; Result in(D X) (AX) (AX) x (CX)

3

DIV : Unsigned division
This instruction is used to divide an unsigned word by a byte or to divide an unsigned double word by a word.

Eg.
DIV CL ; Word in AX / byte in CL

3 | 9 | 9

; Quotient in AL, remainder in AH

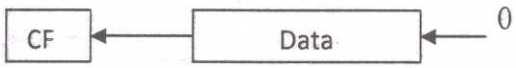
VIb.

- **SHR** – Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.

  Ex: SHR BL,02

- **SHL/SAL** – Used to shift bits of a byte/word towards left and put zero(S) in LSBs.

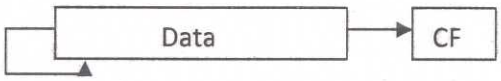
  Ex: SHL AX,05



3

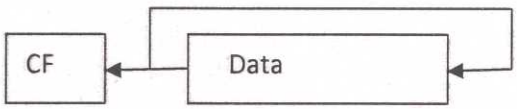
- **SAR** – Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

  Ex: Sar ah, 03



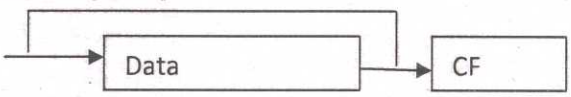
Instructions to perform rotate operations

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



  Ex: ROL CH,2

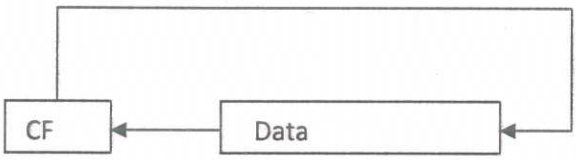
- **ROR** – Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF]. 3



  Ex: ROR CH,2

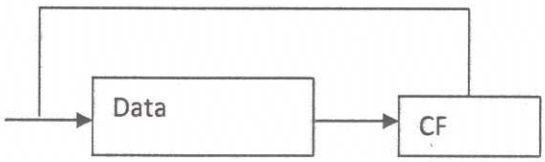
- **RCL** – Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

  Ex: RCL BL,2



- **RCR** – Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.

  Ex: RCR BH,2



(Any 4) | 3 | 6 | 6

VIIa.

The 8259 is known as the Programmable Interrupt Controller (PIC) microprocessor

By adding 8259, we can increase the interrupt handling capability.

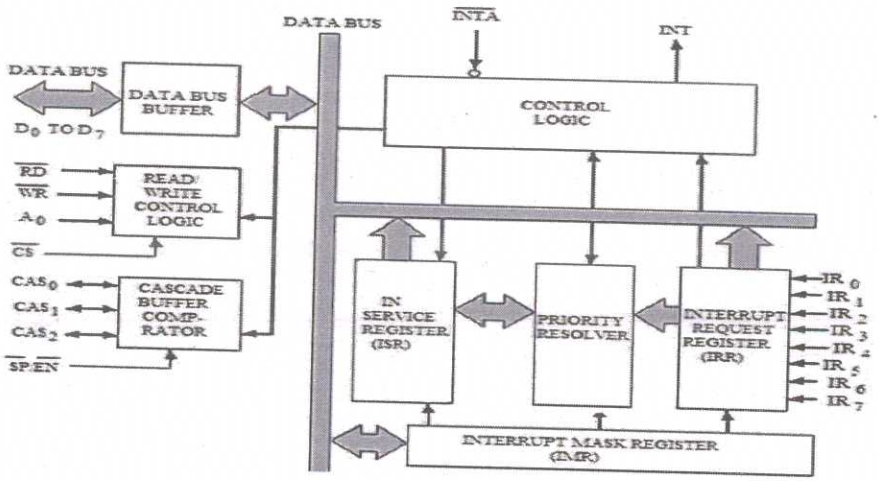
This chip combines the multi-interrupt input source to single interrupt output.

This provides 8-interrupts from IR0 to IR7.

Features of this microprocessor are .

- This chip is designed for 8085 and 8086.
- It can be programmed either in edge triggered, or in level triggered mode
- We can mask individual bits of Interrupt Request Register. | 5 | 5 | 5
- By cascading 8259 chips, we can increase interrupts up to 64 interrupt lines
- Clock cycle is not needed.

VII b.



6

*To increase number of interrupt pin, we can cascade more number of pins, by using cascade buffer.
*Data bus buffer :takes the control word from 8085/8086 and send it to the 8259. It transfers the opcode of the selected interrupts and address of ISR to the other connected microprocessor.
*Interrupt service register: stores interrupt level that are currently being execute.
*Interrupt request register stores the level of all incoming interrupt request.
*Priority Resolver Sets the priority

4 | 10 | 10

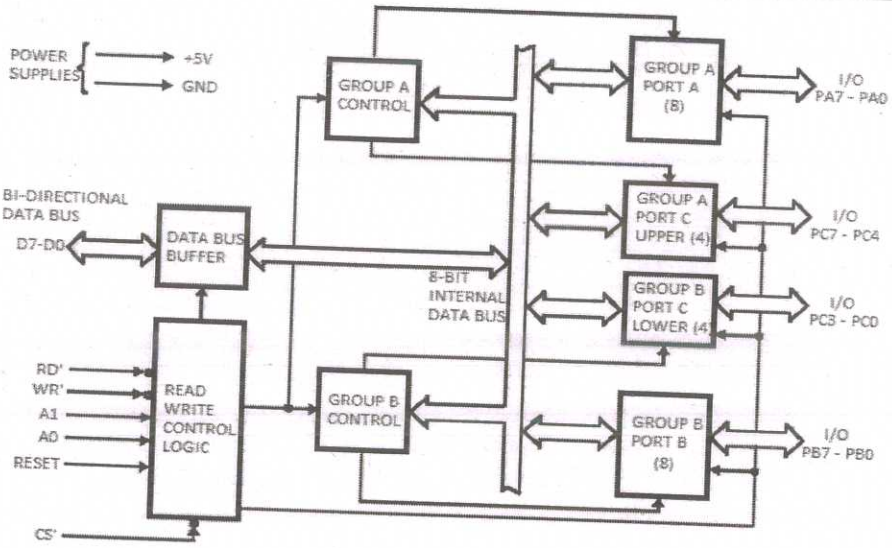
VIIIa

PPI 8255 is a general purpose programmable I/O device designed to interface the CPU with its outside world such as ADC, DAC, keyboard etc.

We can program it according to the given condition. It can be used with almost any microprocessor.

It consists of three 8-bit bidirectional I/O ports i.e. PORT A, PORT B and PORT C. We can assign different ports as input or output functions. 4

Block diagram –



5

*It consists of 40 pins and *operates in +5V regulated power supply.

*Port C is further divided into two 4-bit ports i.e. port C lower and port C upper

*port C can work in either BSR (bit set rest) mode or in mode 0 of input-output mode of 8255.

*Port B can work in either mode or in mode 1 of input-output mode.

*Port A can work either in mode 0, mode 1 or mode 2 of input-output mode.

15

It has two control groups, control group A and control group B. Control group A consist of port A and port C upper. *Control group B consists of port C lower and port B. 6

15

IXa.

Pipelining is the process of accumulating instruction from the processor through a **pipeline**. It allows storing and executing instructions in an orderly process. It is also known as **pipeline** processing. **Pipelining** is a technique where multiple instructions are overlapped during execution. It performs fetching ,decoding and execution of multiple instructions simultaneously

2

1. Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycles.
2. Any condition that causes a stall in the pipeline operations can be called a hazard.
3. There are primarily three types of hazards: 1

   i. Data Hazards

   ii. Control Hazards or instruction Hazards

iii. Structural Hazards.

i. **Data Hazards:**

A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. As a result of which some operation has to be delayed and the pipeline stalls. Whenever there are two instructions one of which depends on the data obtained from the other. 1

A=3+A 1

B=A*4

For the above sequence, the second instruction needs the value of 'A' computed in the first instruction.

Thus the second instruction is said to depend on the first.

If the execution is done in a pipelined processor, it is highly likely that the interleaving of these two instructions can lead to incorrect results due to data dependency between the instructions. Thus the pipeline needs to be stalled as and when necessary to avoid errors.

ii. **Structural Hazards:**

This situation arises mainly when two instructions require a given hardware resource at the same time and hence for one of the instructions the pipeline needs to be stalled. 1

The most common case is when memory is accessed at the same time by two instructions. One instruction may need to access the memory as part of the Execute or Write back phase while other instruction is being fetched. In this case if both the instructions and data reside in the same memory. Both the instructions can't proceed together and one of them needs to be stalled till the other is done with the memory access part. Thus in general sufficient hardware resources are needed for avoiding structural hazards. 1

iii. **Control hazards:**

The instruction fetch unit of the CPU is responsible for providing a stream of instructions to the execution unit. The instructions fetched by the fetch unit are in consecutive memory locations and they are executed. 1

However the problem arises when one of the instructions is a branching instruction to some other memory location. Thus all the instruction fetched in the pipeline from consecutive memory locations are invalid now and need to removed(also called flushing of the pipeline).This induces a stall till new instructions are again fetched from the memory address specified in the branch instruction.

Thus the time lost as a result of this is called a branch penalty. Often dedicated hardware is incorporated in the fetch unit to identify branch instructions and compute branch addresses as soon as possible and reducing the resulting delay as a result. 1

9

9

IXb.	64 bit data bus supports pipelining Separate code cache and data cache 8 KB of dedicated instruction cache Super scalar Architecture Two Integer execution units, one Floating point execution unit Dynamic Branch Prediction Logic (any 6)	6*1	6	6
Xa.	*For higher performance in a multiprocessor system, each processor will usually have its own cache. * Cache coherence refers to the problem of keeping the data in these caches consistent. *The main problem is dealing with writes by a processor. *There are two general strategies for dealing with writes to a cache: **Write-through** - all data written to the cache is also written to memory at the same time. **Write-back** - when data is written to a cache, a dirty bit is set for the affected block. The modified block is written to memory only when the block is replaced. **Software solution:** • In software approach, the detecting of potential cache coherence problem is transferred from run time to compile time, and the design complexity is transferred from hardware to software. • On the other hand, compile time; software approaches generally make conservative decisions. Leading to inefficient cache utilization. • Compiler-based cache coherence mechanism perform an analysis on the code to determine which data items may become unsafe for caching, and they mark those items accordingly. So, there are some more cacheable items, and the operating system or hardware does not cache those items.	3		

- The simplest approach is to prevent any shared data variables from being cached. This is too conservative, because a shared data structure may be exclusively used during some periods and may be effectively read-only during other periods.
- It is only during periods when at least one process may update the variable and at least one other process may access the variable then cache coherence is an issue More efficient approaches analyze the code to determine safe periods for shared variables. The compiler then inserts instructions into the generated code to enforce cache coherence during the critical periods.

***Hardware solutions:**

- Hardware solution provide dynamic recognition at run time of potential inconsistency conditions. Because the problem is only dealt with when it actually arises, there is more effective use of caches, leading to improved performances over a software approach.
- Hardware schemes can be divided into two categories: directory protocol and snoopy protocols.

***Directory protocols:**

- Directory protocols collect and maintain information about where copies of lines reside. Typically, there is centralized controller that is part of the main memory controller, and a directory that is stored in main memory. 1
- The directory contains global state information about the contents of the various local caches.
- When an individual cache controller makes a request, the centralized controller checks and issues necessary commands for data transfer between memory and caches or between caches themselves.
- It is also responsible for keeping the state information up to date, therefore, every local action that can effect the global state of a line must be reported to the central controller.
- The controller maintains information about which processors have a copy of which lines.
- Before a processor can write to a local copy of a line, it must request exclusive access to the line from the controller. 2
- Before granting thus exclusive access, the controller sends a message to all processors with a cached copy of this time, forcing each processors to invalidate its copy.
- After receiving acknowledgement back from each such processor, the controller grants exclusive access to the requesting processor.
- When another processor tries to read a line that is exclusively granted to another processors, it will send a miss notification to the controller.
- The controller then issues a command to the processor holding that line that requires

	the processors to do a write back to main memory. • Directory schemes suffer from the drawbacks of a central bottleneck and the overhead of communication between the various cache controllers and the central controller. ***Snoopy Protocols:** • Snoopy protocols distribute the responsibility for maintaining cache coherence among all of the cache controllers in a multiprocessor system. • A cache must recognize when a line that it holds is shared with other caches. • When an update action is performed on a shared cache line, it must be announced to all other caches by a broadcast mechanism. • Each cache controller is able to "snoop" on the network to observed these broadcasted notification and react accordingly. • Snoopy protocols are ideally suited to a bus-based multiprocessor, because the shared bus provides a simple means for broadcasting and snooping. • Two basic approaches to the snoopy protocol have been explored: Write invalidates or write- update (write-broadcast) • With a write-invalidate protocol, there can be multiple readers but only one write at a time. • Initially, a line may be shared among several caches for reading purposes. • When one of the caches wants to perform a write to the line it first issues a notice that invalidates that tine in the other caches, making the line exclusive to the writing cache. Once the line is exclusive, the owning processor can make local writes until some other processor requires the same line. • With a write update protocol, there can be multiple writers as well as multiple readers. When a processors wishes to update a shared line, the word to be updated is distributed to all others, and caches containing that line can update it.	1 2	9	9
Xb.	80386 that allows the switching between the modes using software commands. *In the **protected mode**, 80386 microprocessor operates in similar way like 80286, but offers higher memory addressing ability. *In **virtual mode**, the overall memory of 80386 can be divided into various virtual machines. And all of them acts as a separate computer with 8086 microprocessor. This mode is also called virtual 8086 mode or V86 mode. *The other one is the **virtual real mode**, this mode allows the system to execute multiple programs in the protected memory. And in case a program at a particular memory gets crashed then it will not cause any adverse effect on the other part of the memory.	2 2 2	6	6