Q.1 Attempt the following (any THREE) **[15]**

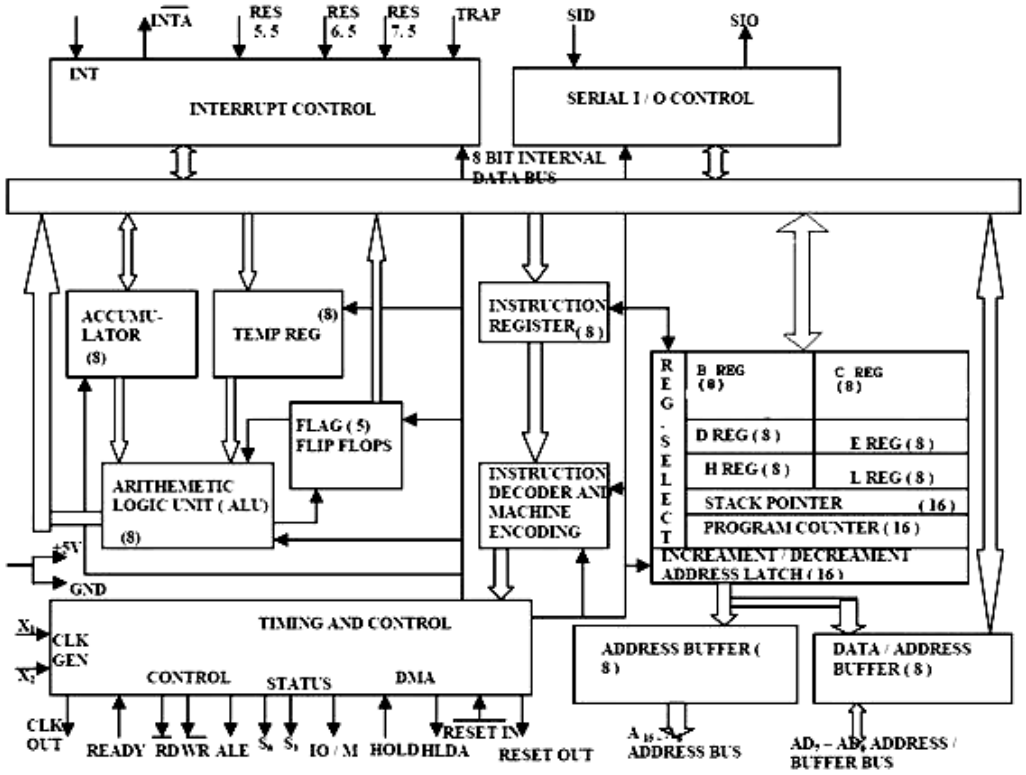
Q.1(a) List down feature of 8085. **[5]**

(A)

- It is an 8 bit microprocessor (each character is represented by 8 bits or a byte).
- It is manufactured with N-MOS (n-type Metal Oxide Semiconductor) technology implemented with 6200 transistors.
- It has 16-bit address lines - A0-A15 (to point the memory locations) and hence can point up to 2^16 = 65535 bytes (64KB) memory locations.
- The first 8 lines of address bus and 8 lines of data bus are multiplexed AD0-AD7. Data bus is a group of 8 lines D0-D7.
- It provides 5 level interrupts and supports external interrupt request.
- A 16 bit program counters (PC).
- A 16 bit stack pointer (SP).
- It provides 1 accumulator, 2 flag register, six 8-bit general purpose register arranged in pairs: BC, DE, HL and 2special purpose registers.
- It consists of 74 instruction sets.
- It performs arithmetic and logical operations.
- It provides status for advanced control signals, On chip clock generator.
- It requires a signal +5V power supply and operates at 3.2 MHZ single phase clock with maximum clock frequency 6 MHz and minimum clock frequency 500 kHz.
- Serial input/output port.
- 1.3 micro sec instruction cycles.
- It is enclosed with 40 pins DIP (Dual in line package).
- It can be used to implement (interface) 3 chip micro-computers (8085, 8155, 8255 and 8355: Peripheral IC's).

Q.1(b) Draw 8085 Architecture and explain in about accumulator. **[5]**

(A) **Accumulator** : It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.



Q.1(c) Explain difference type of register in 8085. **[5]**

(A) Registers used in 8085 Microprocessor and their details

Registers Used

(i) Simple register (main register)
(ii) General purpose register
(iii) Special function register
(iv) Other register

(i) Simple register (main register)

(1) Accumulator
- 8 bit
- Used as a register for storing one data when two are arithmetically and logically operated .
- after ALU operation result is also stored in accumulator.
- when single no is to be logically operated only accumulator is used as a storage as well as storing result after execution.

(ii) General purpose register

- B, C, D, E, H & L are used as general purpose register.
- Each 8 bit long.
- Pairing can also be done in a standard way to stored 16 bit data
  eg (B-C, D-E, H-L)
- H-L pair is an exceptional pair which is used as a memory pointer to locate an address L X H 2000H.
  H recognize H-L pair of general purpose register.
  Load 16 bit data immediately in H-L pair it means 2000H is the location of memory where data is stored and it is recall by an instruction MOV A M
  M locates memory location 2000H which is pointed bye H-L pair.
  Exceptional point in 8085 microprocessor 16 bit additional can only be possible by using an instruction DAD.

(iii) Special function register

Stack pointer & program counter are two special function register these are those register which are used only by microprocessor not by user.

(1) Program counter
- it is a 16 bit register
- Used by microprocessor
- It holds the address of next instruction to be executed for eg. after execution MOV A,B instruction program counter will be automatically incremented and points next location of memory where another data to stored.

(2) Stack pointer: It is used when interrupt is generated by microprocessor interrupt is a command which stop the main program counter location will be stored in a stack and program counter will be loaded with a new address through which sub routine is called.

Sub routine is a small program which is used many times by main program. For example- Generating delay in main program after execution of each instruction.

(3) Stack :
Small memory location acquired by main program in RAM area.

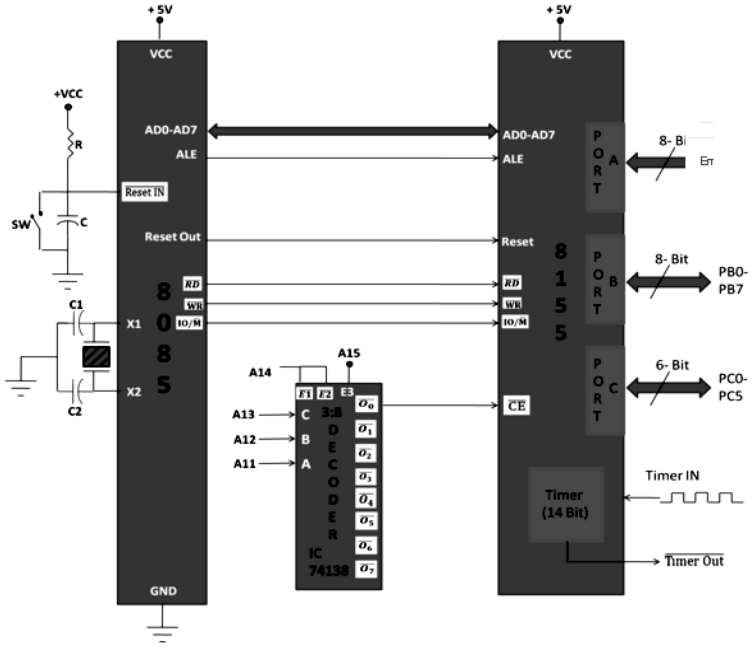
(iv) Other Registers

(1) Instruction Registers
(2) Temporary Register

- Instruction Registers
  - 8 Bit long
  - It will store opcode (8 Bit) of an instruction.
- Temporary register
  - 8 Bit long
  - It will store the data temporary before execution of instruction.

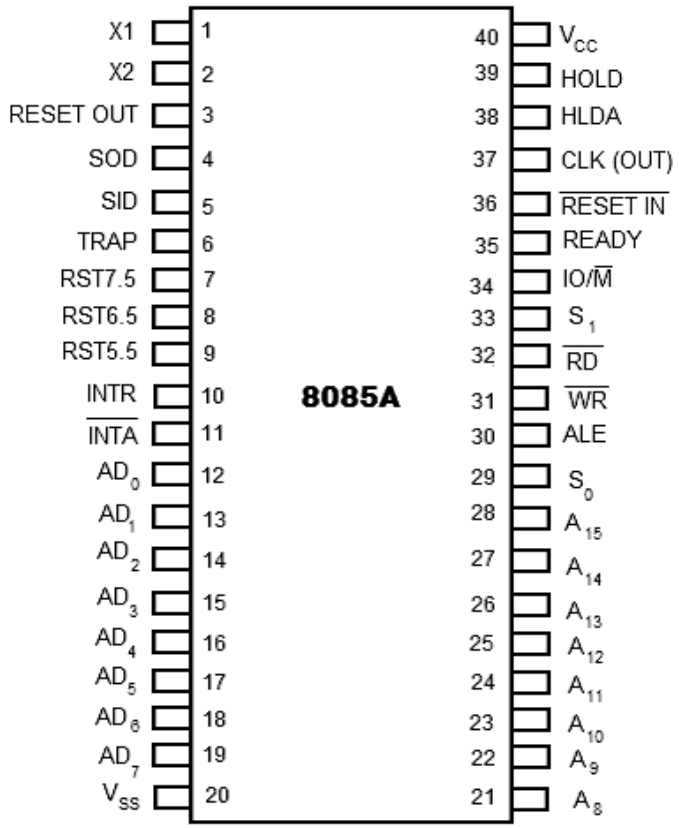
Q.1(d) Draw interfacing of 8155 to 8085. **[5]**

(A)



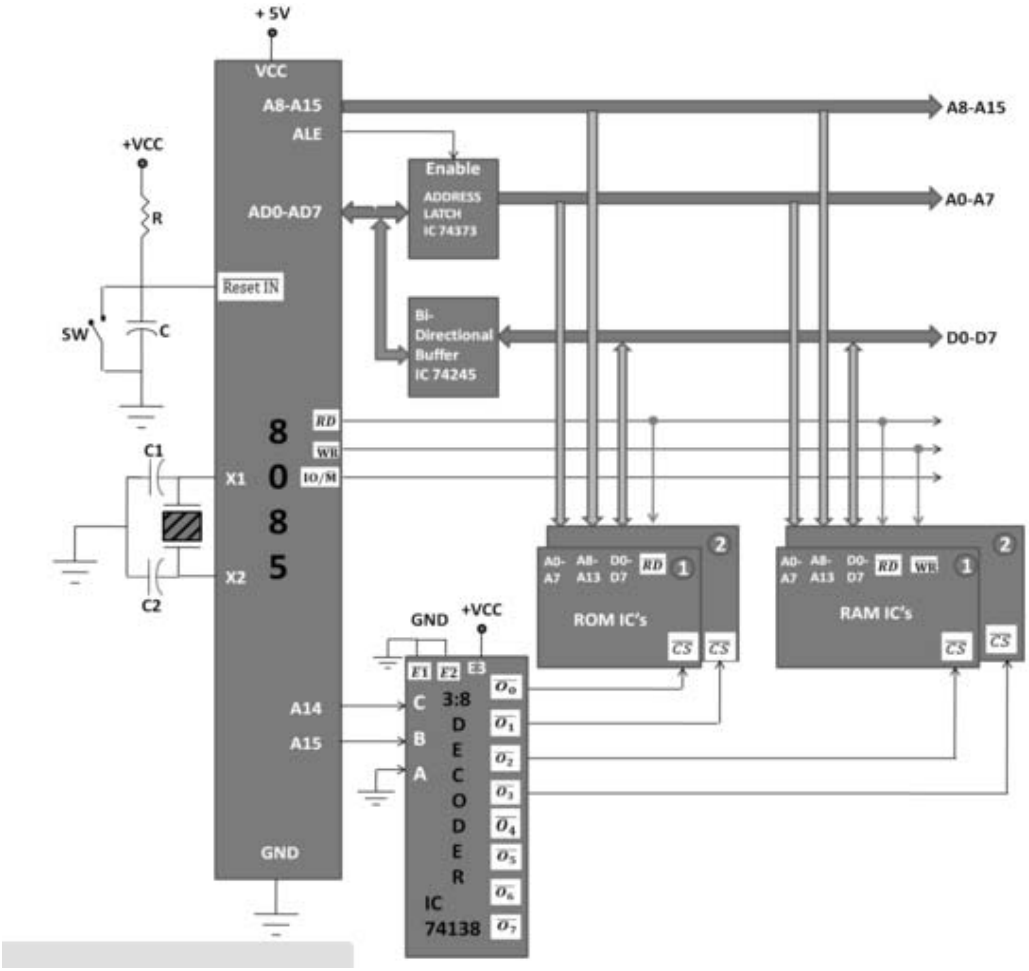
Q.1(e) Draw function pin diagram of 8085. **[5]**

(A)



Q.1(f) Explain interfacing of 8085 with memory interface 1k RAM and 1k ROM. [5]

(A)



Q.2 Attempt the following (any THREE) [15]

Q.2(a) Explain data transfer instruction and load and store instruction. [5]

(A) Data Transfer Instructions

(i) MOV r1,r2 (Move data; move the contents of one register to another).
The contents of r2 register is moved to r1.This instruction doesn't affect any flags.
States: 4 Machine cycles : 1.

(ii) MOV r,M (move the content of memory to register).
No Flags are affected after executing this instruction. Addressing modes is register indirect.
States: 7 Machine cycles :2.

(iii) MOV M,r (move the content of register to memory).
No Flags are affected after executing this instruction. Addressing modes is register indirect.
States: 7 Machine cycles :2.

(iv) MVI r,data (move immediate data to register).
No Flags are affected after executing this instruction. Addressing modes is immediate.
States: 7 Machine cycles : 2.

(v) MVI M,data (move immediate data to memory location).
No Flags are affected after executing this instruction. Addressing modes is immediate/register indirect.
States: 10 Machine cycles :3.

(vi) LXI rp,data16 (Load register pair immediate).
This instruction loads immediate data of 16-bit into register pair rp. No Flags are affected after executing this instruction. Addressing modes is immediate.
States: 10 Machine cycles :3.

(vii) LDA addr (Load accumulator direct).

The content of the memory location specified in the instruction is stored into the accumulator.

No Flags are affected after executing this instruction. Addressing modes is direct. States: 13 Machine cycles :4.

(viii) STA addr (Store accumulator direct).

The content of the memory location specified in the instruction is stored with the accumulator contents. No Flags are affected after executing this instruction. Addressing modes is direct.

States: 13 Machine cycles :4.

(ix) LHLD addr (Load H-L pair direct).

The content of memory location specified in the instruction is stored in the L register and the next memory location content is stored in the H register. No Flags are affected after executing this instruction. Addressing modes is direct.

States: 16 Machine cycles :5.

(x) SHLD addr (Store H-L pair direct).

The content of memory location specified in the instruction is stored with the L register contents and the next memory location content is stored with the H register contents. No Flags are affected after executing this instruction. Addressing modes is direct. States: 16 Machine cycles :5.

(xi) LDAX rp (LOAD accumulator direct).

The content of memory location, whose address is stored in the register pair is loaded into the accumulator. No Flags are affected after executing this instruction. Addressing modes is register indirect.

States: 7 Machine cycles :2.

(xii) STAX rp (LOAD accumulator direct).

The content of the accumulator is stored in the memory location, whose address is stored in the register pair. No Flags are affected after executing this instruction. Addressing modes is register indirect.

States: 7 Machine cycles :2.

(xiii) XCHG (Exchange the contents of H-L with D-E pair).

The contents of H-L pair are exchanged with contents of D-E pair. No Flags are affected after executing this instruction. Addressing modes is register direct. States: 4 Machine cycles : 1.

These are the different data transfer instructions that are present in the 8085 microprocessor.

Q.2(b) Explain addressing mode. **[5]**

(A) ***Addressing Modes in 8085***

These are the instructions used to transfer the data from one register to another register, from the memory to the register, and from the register to the memory without any alteration in the content. Addressing modes in 8085 is classified into 5 groups –

Immediate addressing mode

In this mode, the 8/16-bit data is specified in the instruction itself as one of its operand. **For example:** MVI K, 20F: means 20F is copied into register K.

Register addressing mode

In this mode, the data is copied from one register to another. **For example:** MOV K, B: means data in register B is copied to register K.

Direct addressing mode

In this mode, the data is directly copied from the given address to the register.**For example:** LDB 5000K: means the data at address 5000K is copied to register B.

Indirect addressing mode

In this mode, the data is transferred from one register to another by using the address pointed by the register. **For example:** MOV K, B: means data is transferred from the memory address pointed by the register to the register K.

Implied addressing mode

This mode doesn't require any operand; the data is specified by the opcode itself. **For example:** CMP.

Q.2(c) Explain logical instruction. **[5]**

(A) **(i) CMP:** (compare register or memory with accumulator) The contents of the operand register or memory are M compared with the contents of the accumulator. Both contents are preserved. The result of the comparison is shown by setting the flags of the PSW as follows:
if (A) < reg/mem: carry flag is set.
if (A) = reg/mem:
zero flag is set.
if (A) > reg/mem: carry and zero flags are reset.
Eg: CMP B
CMP M

(ii) CPI: (compare immediate with accumulator) The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:
if (A) < data: carry flag is set
if (A) = data: zero flag is set
if (A) > data: carry and zero flags are reset
Eg: CPI 89H

(iii) ANA: logical AND operation is performed with the specified register or memory with accumulator.
Eg: ANA B
ANA M

(iv) ANI: (AND operation with immediate data) AND operation is performed between accumulator and specified immediate data.
Eg: ANI 30H

(v) XRA: The content of accumulator are exclusive OR with specified register or memory location.
Eg: XRA B
XRA M

(vi) XRI: The content of accumulator are exclusive OR with the immediate data.
Eg: XRI 30H

(vii) ORA: Logical OR operation is performed between accumulator and specified register and memory location.
Eg: ORA B
ORA M

(viii) ORI: Logical OR operation is performed between accumulator and immediate data.
Eg: ORI 30H

Q.2(d) Explain Arithmetic instructions. **[5]**

(A) **There are some of the important instructions in 8085 microprocessor**

- **ADD :** The content of operand are added to the content of the accumulator and the result is stored in Accumulator.
  Eg. ADD B (it adds the content of accumulator to the content of the register B)
  ADD M (if content is stored in memory location the it is added with the content stored in accumulator)

- **ADC :** Addition with carry
  Eg. ADC B, ADC M
- **ADI:** Add immediate means add an immediate value withe the content of accumulator and it is stored in accumulator.
  Eg. ADI 30H
- **ACI:** Add immediate to accumulator with carry.
  Eg. ACI 40H
- **SBI:** Substract immediate from the content of the accumulator and the result is stored in accumulator.
- **INR:** The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.
  Eg. INR B(the content of register B is incremented by 1.
  INR M(the content of memory location pointed by HL pair in incremented by 1)
- **INX:** Increment register pair by 1.
  Eg. INX H (It means the location pointed by the HL pair is incremented by 1)
- **DCR:** The contents of the designated register or memory are M decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.
  Eg. DCR B(content of register B is decremented by 1)
  DCR M(the content of memory location pointed by the HL pointer is decremented by 1.)
- **DCX:** decrement the register pair by 1.
  Eg. DCX H(it decreases the memory location pointed by HL pair by 1.
- **DAA:** Decimal adjust accumulator.

Q.2(e) WAP to transfer N Byte. Block Data from one memory location to other memory location. [5]

(A) Program

Address	Mnemonics	Operand	Opcode	Remarks
2000	LDA	3000H	3A	Load H-L pair with data from 3000H.
2001			00	Lower-order of 3000H.
2002			30	Higher-order of 3000H.
2003	ANI	0FH	E6	AND Immediate 0FH with reg. A.
2004			0F	Immediate value 0FH.
2005	STA	3001H	32	Store the result at memory location 3001H.
2006			01	Lower-order of 3001H
2007			30	Higher-order of 3001H.
2008	HLT		76	Halt.

Q.2(f) WAP to find 2's complement of number. [5]

(A)
```
LXI H, 4150 : Initialize memory pointer
MVI B, 08 : count for 8-bit
MVI A, 54
LOOP : RRC
JC LOOP1
MVI M, 00 : store zero it no carry
JMP COMMON
LOOP2: MVI M, 01 : store one if there is a carry
COMMON: INX H
DCR B : check for carry
JNZ LOOP
HLT : Terminate the program
```

Q.3 Attempt the following (any THREE) : [15]

Q.3(a)Explain Rotate Instruction. [5]

(A) **(i) RLC :** Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7. Any other bit is not affected.
Eg. : RLC

(ii) RRC : Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. Any other bit is not affected.
Eg. : RRC

(iii) RAL : Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7.
Eg. : RAL

(iv) RAR : Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0.
Eg.: RAR

Q.3(b)WAP to add and subtract two eight bit number. [5]

(A) Addition of bit number

Address	Label	Mnemonics	Hex Code	Comments
4100		MVI C,00	OE, 00	Initialize the carry as zero
4102		LDA 4300	3A, (00, 43)	Load the first 8 bit data
4105		MOV, B,A	47	Copy the value of 8 bit data into register B
4106		LDA 4301	3A, (01, 43)	Load the second 8 bit data into the accumulator
4109		ADD B	80	Add the hoo values
410A		JNC	D2, 0E, 41	Jump on if no carry
410D		INR C	OC	If carry is there increment it by one
410E	Loop	STA 4302	32 (02, 43)	Stone the added value in the accumulator
4111		MOV A,C	79	More the value of carry to the accumulator from register C
4112		STA 4303	32 (03, 43)	Store the value of carry in the accumulator
4115		HLT	76	Stop the program execution

Subtraction of 8 bit number

Address	Label	Mnemonics	Hex Code	Comments
4100		MVI C,00	OE, 00	Initialize the carry as zero
4102		LDA 4300	3A, (00, 43)	Load the first 8 bit data into the accumulator
4105		MOV, B,A	47	Copy the value into register 'B'
4106		LDA 4301	3A, (01, 43)	Load the 2^{nd} 8 bit data into the accumulator
4109		SUB B	90	Subtract both the values
410A	Loop	INC	D2, 0E, 41	Jump on if no borrow
410D		INR C	OC	If borrow is there, increment it by one
410E	Loop	CMA	2F	Compliment of 2^{nd} data
410F		ADI, 01	6, 01	Add one to 1's compliment of 2^{nd} data
4111		STA 4302	32,02,43	Store the result in accumulator
4114		MOV A,C	79	Moul the value of borrow into the accumulator
4115		STA 4303	32,03,43	Store the result in accumulator
4118		HLT	76	Stop Program execution

Q.3(c) What is subroutine? and write its importance. [5]

(A) In computer programming, a subroutine is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

Subprograms may be defined within programs, or separately in libraries that can be used by multiple programs. In different programming languages, a subroutine may be called a procedure, a function, a routine, a method, or a subprogram. The generic term callable unit is sometimes used.

The name subprogram suggests a subroutine behaves in much the same way as a computer program that is used as one step in a larger program or another subprogram. A subroutine is often coded so that it can be started (called) several times and from several places during one execution of the program, including from other subroutines, and then branch back (return) to the next instruction after the *call*, once the subroutine's task is done. Maurice Wilkes, David Wheeler, and Stanley Gill are credited with the invention of this concept, which they termed a *closed subroutine*,[2][3] contrasted with an *open subroutine* or macro.

Subroutines are a powerful programming tool, and the syntax of many programming languages includes support for writing and using them. Judicious use of subroutines (for example, through the structured programming approach) will often substantially reduce the cost of developing and maintaining a large program, while increasing its quality and reliability. Subroutines, often collected into libraries, are an important mechanism for sharing and trading software. The discipline of object-oriented programming is based on objects and methods (which are subroutines attached to these objects or object classes).

Q.3(d) Explain delay using 16 bit counter and calculate total time delay. [5]

(A)
```
LXI B, count : 16 - bit count
BACK: DCX B : Decrement count
MOV A, C
ORA B : Logically OR Band C
JNZ BACK : If result is not zero repeat
HLT
```

Q.3(e) Explain use of stack by program using push and Pop instruction. [5]

(A) **PUSH:** This instruction pushes the register pair onto stack. The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the highorder register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location.

Eg:
```
PUSH B
PUSH A
```

POP: This instruction pop off stack to register pair. The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1.

Eg:
```
POP H
POP A
```

Q.3(f) Explain Branching Instruction (CALL, JUMP) [5]

(A) JUMP Instruction

1. **JMP:** (unconditionally jump) The program sequence is transferred to the memory location specified by the16-bit address given in the operand.
   Eg. JMP 2034H (jump to location 2034H) there is no condition to jump.
   JMP ABC (jump to abc level)

2. **JC:** (conditional jump) The program sequence is transferred to a particular level or a 16-bit address if C=1 (or carry is 1)
   Eg. JC ABC (jump to the level abc if C=1)
3. **JNC:** (conditional jump) The program sequence is transferred to a particular level or a 16-bit address if C=0 (or carry is 0)
   Eg: JNC ABC (jump to the level abc if C=0)
4. **JP:** (conditional jump) The program sequence is transferred to a particular level or a 16-bit address if S=0 (or sign is 0)
   Eg. JP ABC (jump to the level abc if S=0)
5. **JM:** (conditional jump) The program sequence is transferred to a particular level or a 16-bit address if S=1 (or sign is 1)
   Eg. JM ABC (jump to the level abc if S=1)
6. **JZ:** (conditional jump) The program sequence is transferred to a particular level or a 16-bit address if Z=1 (or zero flag is 0)
   Eg. JZ ABC (jump to the level abc if Z=1)
7. **JNZ:** (conditional jump) The program sequence is transferred to a particular level or a 16-bit address if Z=0 (or zero flag is 0)
   Eg: - JNZ ABC (jump to the level abc if Z=0)
8. **JPE:** (conditional jump) The program sequence is transferred to a particular level or a 16-bit address if P=1 (or parity flag is 1)
   Eg. JPE ABC (jump to the level abc if P=1)
9. **JPO:** (conditional jump)The program sequence is transferred to a particular level or a 16-bit address if P=0 (or parity flag is 0)
   Eg. JPO ABC (jump to the level abc if P=0)

CALL Instruction :

1. **CC:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if C=1 (or carry is 1)
   Eg. CC ABC (jump to the level abc if C=1)
2. **CNC:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if C=0 (or carry is 0)
   Eg. CNC ABC (jump to the level abc if C=0)
3. **CP:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if S=0 (or sign is 0)
   Eg. CP ABC (jump to the level abc if S=0)
4. **CM:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if S=1 (or sign is 1)
   Eg. CM ABC (jump to the level abc if S=1)
5. **CZ:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if Z=1 (or zero flag is 1)
   Eg. CZ ABC (jump to the level abc if Z=1)
6. **CNZ:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if Z=0 (or zero flag is 0)
   Eg. CNZ ABC (jump to the level abc if Z=0)
7. **CPE:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if P=1 (or parity is 1)
   Eg. CPE ABC (jump to the level abc if P=1)
8. **CPO:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if P=0 (or parity is 0)
   Eg. CPO ABC (jump to the level abc if P=0)

RETURN Instruction

1. **RC:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if C=1 (or carry is 1)
   Eg. RC ABC (jump to the level abc if C=1)

2. **RNC:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if C=0 (or carry is 0)
   Eg. RNC ABC (jump to the level abc if C=0)
3. **RP:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if S=0 (or sign is 0)
   Eg. RP ABC (jump to the level abc if S=0)
4. **RM:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if S=1 (or sign is 1)
   Eg. RM ABC (jump to the level abc if S=1)
5. **RZ:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if Z=1 (or zero flag is 1)
   Eg. RZ ABC (jump to the level abc if Z=1)
6. **RNZ:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if Z=0 (or zero flag is 0)
   Eg. RNZ ABC (jump to the level abc if Z=0)
7. **RPE:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if P=1 (or parity is 1)
   Eg. RPE ABC (jump to the level abc if P=1)
8. **RPO:** (conditional call) The program sequence is transferred to a particular level or a 16-bit address if P=0 (or parity is 0)
   Eg. RPO ABC (jump to the level abc if P=0)

9.RET: Return from subroutine unconditionally

Q.4 Attempt the following (any THREE) [15]

Q.4(a) Short note on Assembler and Cross Assembler. [5]

(A) Assembler : An assembler is program that takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations. Some people call these instructions assembler language and others use the termassembly language.

An assembler that generates machine language for a different type of computer than the one the assembler is running in. It is used to develop programs for computers on a chip or microprocessors used in specialized applications that are either too small or are otherwise incapable of handling the development software.

Q.4(b) WAP to Binary to ASCII Conversion. [5]

(A) Assembly Language Program

```
C100    LDA    C200    3A    ; Load the accumulator with binary number
C101                   00    ;
C102                   C2    ;
C103    CPI    0A      FE    ; Compare immediately the accumulator content
C104                   0A    ; with OA_H
C105    JC     C10A    DA    ; Jump if carry to C10A_H
C106                   0A    ;
C107                   C1    ;
C108    ADI    07      C6    ; Add immediate data 07_H to the accumulator
C109                   07    ;
C10A    ADI    30      C6    ; Add immediate data 30_H to the accumulator
C10B                   30    ;
C10C    STA    C202    32    ; Store the accumulator content at C202_H
C10D                   02    ;
C10E                   C2    ;
C10F    HLT            76    ; Halt the execution
```

Q.4(c) WAP to Packed the unpacked BCD. **[5]**

(A) Source Program

LDA 4201H	: Get the Most significant BCD digit
RLC	
RLC	
RLC	
RLC	: Adjust the position of the second digit (09 is changed to 90)
ANI FOH	: Make least significant BCD digit zero
MOV C, A	: store the partial result
LDA 4200H	: Get the lower BCD digit
ADD C	: Add lower BCD digit
STA 4300H	: Store the result
HLT	: Terminate program execution

Q.4(d) WAP to convert BCD to ASCII. **[5]**

(A)

```
    LDA 5000 Get Hexa Data
    MOV B, A
    ANI OF        ; Mask Upper Nibble
    CALL SUB1     ; Get ASCII code for upper nibble
    STA 5001
    MOV A,B
    ANI F0        ; Mask Lower Nibble
    RLC
    RLC
    RLC
    RLC
    CALL SIB1     ; Get ASCII code for lower nibble
    STA 5002
    HLT       ; Halt the program

SUB1: CPI 0A
    JC SKIP
    ADI 07
SKIP: ADI 30
    RET           ; Return Subroutine
```

Result
Input :
Data 1 : HEX data – E4H in memory location 5000

Output:
Data 1 : 34H in memory location 5001 (ASCII Code for 4)
Data 2 : 45H in memory location 5002 (ASCII Code for E)

Q.4(e) WAP to convert ASCII to BCD. **[5]**

(A) ASCII to Decimal Conversion (8085)

Statement : Convert the ASCII number in memory to its equivalent decimal number.

Source Program:
LXI H, 4150 : Point to data
MOV A, M : Get operand
SUI 30 : convert to decimal
CPI 0A : Check whether it is valid decimal number

JC LOOP : yes, store result
MVI A, FF : No, make result = FF
LOOP : INX H
MOV M, A
HLT : (A) = (4151)

Note : The ASCII Code (American Standard Code for Information Interchange) is commonly used.

Q.4(f) WAP for BCD addition and subtraction. **[5]**

(A) Source Program:
MVI A, 99H
SUB E : Find the 99's complement of subtrahend
ADD D : Add minuend to 100's complement of subtrahend
DAA : Adjust for BCD
HLT : Terminate program execution

Q.5 Attempt the following (any THREE) **[15]**

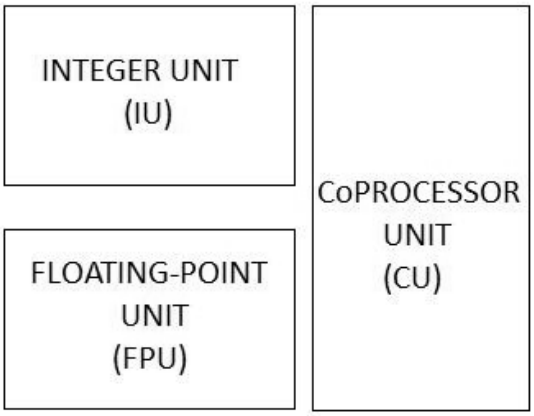
Q.5(a) Differentiate i3, i5 and i7 processor. **[5]**

(A)

Model	Core i3	Core i5	Core i7
Number of cores	2	4	4
Hyper-threading	yes	No	Yes
Turbo boost	No	yes	Yes
K model	No	yes	Yes

Q.5(b) Explain Sun SPARC Architecture. **[5]**

(A)



IU

- Contains the general purpose registers and controls the overall opertion of the processor.
- May contain from 64 to 528 general-purpose 64-bit r registers. The are partitioned into 8 global registers, 8 alternate global registers, plus a circular stack of from 3 to 32 sets of 16 registers each, known as register windows.
- Executes the integer arthmetic instructions and computes memory addresses for toads and stores.
- Maintains the program counters and controls instruction execution for the FPU.

FPU

- The FPU has 32 32-bit (single-precision) floating-point registers. 32 64-bit (double-precision) floating-point registers, and 16 128-bit (quad-precision) floating-point registers.

- Double-precision values occupy an even-odd pair of single-precision registers.
- Quad-precision values occupy an odd-even number pair of double precision registers.
- Floating-point load/store instructions are used to move data between the FPU and memory.
- The memory address is calculated by the IU.
- Floatng-Point operate (FPop) instructions perform the floating-point arithmetic operations and comparisons.

CU

- The instruction set includes support for a single, implementation-dependent coprocessor. The coprocessor has its own set of registers.
- Coprocessor load/store instructions are used to move data between the coprcessor registers and memory.
- Floating-point instructions mirrors coprocessor instructions.
- Not implemented in SPARC V9.

Q.5(c) Explain special Pentium programmer feature. [5]

(A) A salient feature of Pentium is its superscalar, superpipelined architecture. It has two integer pipelines U and V, where each one is a 4-stage pipeline. This enhances the speed of integer arithmetic of Pentium to a large extent. Moreover, it has an on-chip floating-point unit, which has increased the floating-point performance manifold compared to the floating-point performances of 80386/486 processors.

Another feature of Pentium is that it contains two separate caches, viz. data cache and instruction cache. In 80486 there was a single unified data/instruction cache.

The Intel CPU architectures up to 80486 issues only one instruction to the execution unit per cycle. This obviously leads to a comparatively slow process of decoding and execution. For enhancement of processor performance beyond one instruction per cycle, the computer architects employ the technique of multiple instruction issue (MII). Thus a microprocessor which is capable of issuing more thaw instruction per single processor cycle will be termed as MII microprocessor. Obvious executing more than one instruction in a cycle, the microprocessor must have more than execution channels. Thus there are two problems, viz. (a) How to issue multiple instruct, and (b) How to execute them concurrently. Keeping in view these two issues, I architectures may again be redivided in two classes of architectures — (i) Very Long Instruction Word (VLIW) architecture and (ii) Superscalar architecture.

Q.5(d) Explain memory management of Pentium processor. [5]

(A) **Pentium Memory management :** The memory-management unit within the Pentium is upward-compatible with the 80386 and 80486 microprocessors. Many of the features of these earlier microprocessors are basically unchanged in the Pentium. The main change is in the paging unit and a new system memory-management mode.

Paging Unit : The paging mechanism functions with 4K-byte memory pages or with a new extension available to the Pentium with 4M byte-memory pages. As detailed in Chapters 1 and 17, the size of the paging table structure can become large in a system that contains a large memory. Recall that to fully repage 4G bytes of memory, the microprocessor requires slightly over 4M bytes of memory just for the page tables. In the Pentium, with the new 4M-byte paging feature, this is dramatically reduced to just a single page table. The new 4M-byte page sizes are selected by the PSE bit in control register 4.

The main difference between 4K paging and 4M paging is that in the 4M paging scheme there is no page table entry in the linear address. See Figure 5.6 for the 4M paging system in the Pentium microprocessor. Pay close attention to the way the linear address is used

with this scheme. Notice that the leftmost 10 bits of the linear address select an entry in the page directory (just as with 4K pages). Unlike 4K pages, there are no page tables; instead, the page directory addresses a 4M-byte memory page.

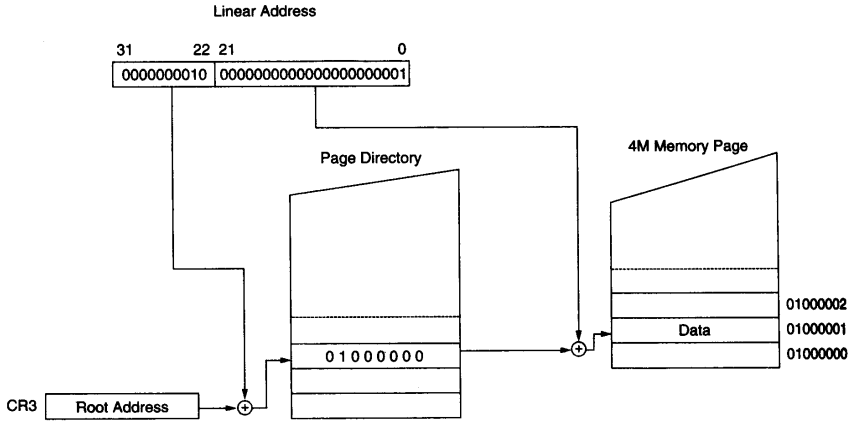


Fig. : The linear address 00200001H repaged to memory location 01000002H in 4Mbyte pages. Note that there are no page tables.

Memory-Management Mode

The system memory-management mode (SMM) is on the same level as protected mode, real mode, and virtual mode, but it is provided to function as a manager. The SMM is not intended to be used as an application or a system-level feature. It is intended for high-level system functions such as power management and security, which most Pentiums use during operation.

Access to the SMM is accomplished via a new external hardware interrupt applied to the $\overline{SMI}$ pin on the Pentium. When the SMM interrupt is activated, the processor begins executing system-level software in an area of memory called the *system management RAM* or SMMRAM, called the SMM *state dump record*. The $\overline{SMI}$ interrupt disables all other interrupts that are normally handled by user applications and the operating system. A return from the SMM interrupt is accomplished with a new instruction. RSM returns from the memory-management mode interrupt and returns to the interrupted program at the point of the interruption.

The SMM interrupt calls the software, initially stored at memory location 38000H, using CS=3000H and EIP = 8000H. This initial state can be changed using a jump to any location within the first 1M byte of memory. An environment similar to real-mode memory addressing is entered by the management mode interrupt, but it is different because, instead of being able to address the first 1M of memory, SMM mode allows the Pentium to treat the memory system as a flat, 4G-byte system.

In addition to executing software that begins at location 38000H, the SMM interrupt also stores the state of the Pentium in what is called a dump record. The dump record is stored at memory locations 3FFA8H through 3FFFFH, with an area at locations 3FE00H through 3FEF7H that is reserved by Intel. The dump record allows a Pentium-based system to enter a sleep mode and reactivate at the point of program interruption. This requires that the SMMRAM be powered during the sleep period. Many laptop computers have a separate battery to power the SMMRAM for many hours during sleep mode.

The Halt auto restart and I/O trap restarts are used when the SMM mode is exited by the RSM instruction. These data allow the RSM instruction to return to the halt-state or return to the interrupt I/O instruction. If neither a halt nor an I/O operation is in effect upon entering the mode, the RSM instruction reloads the state of the machine from the state dump and returns point of interruption.

The SMM mode can be used by the system before the normal operating system is placed in the memory and executed. It can also periodically be used to manage the system, provided that normal software doesn't exist at location 38000H—3FFFFH. If the system relocates the SMRAM before booting the normal operating system, it becomes available for use in addition to the normal system.

The base address of the SMM mode SMRAM is changed by modifying the value in the state dump base address registers (locations 3FEF8H through 3F3FBH) after the first memory-management mode interrupt. When the first RSM instruction is executed, returning control back to the interrupted system, the new value from these locations changes the base address of the SMM interrupt for all future uses. For example, if the state dump base address is changed to 000E8000H, all subsequent SMM interrupts use locations E8000H—EFFFFH for the Pentium state dump. These locations are compatible with DOS and Windows.

Q.5(e) List down Pentium Instruction. Explain Pentium special register. [5]

(A) Data Transfer Instructions

MOV	Move
CMOVE/CMOVZ	Conditional move if equal/Conditional move if zero
CMOVNE/CMOVNZ	Conditional move if not equal/Conditional move if not zero
CMOVA/CMOVNBE	Conditional move if above/Conditional move it not below or equal
CMOVAE/CMOVNB	Conditional move if above or equal/Conditional move if not below
CMOVB/CMOVNAE	Conditional move if below/Conditional move if not above or equal
CMOVBE/CMOVNA	Conditional move if below or equal/Conditional move if not above
CMOVG/CMOVNLE	Conditional move if greater/Conditional move if not less or equal
CMOVGE/CMOVNL	Conditional move if greater or equal/Conditional move if not less
CMOVL/CMOVNGE	Conditional move if less/Conditional move if not greater or equal
CMOVLE/CMOVNG	Conditional move if less or equal/Conditional move if not greater
CMOVC	Conditional move if carry
CMOVNC	Conditional move if not carry
CMOVO	Conditional move if overflow
CMOVNO	Conditional move if not overflow
CMOVS	Conditional move if sign (negative)
CMOVNS	Conditional move if not sign (non-negative)
CMOVP/CMOVPE	Conditional move if parity/Conditional move if parity even
CMOVNP/CMOVPO	Conditional move if not parity/Conditional move if parity odd
XCHG	Exchange
BSWAP	Byte swap

Binary Arithmetic Instructions

ADD	Integer add
ADC	Add with carry
SUB	Subtract
SBB	Subtract with borrow
IMUL	Signed multiply
MUL	Unsigned multiply
IDIV	Signed divide
DIV	Unsigned divide
INC	Increment
DEC	Decrement
NEG	Negate
CMP	Compare

Decimal Arithmetic

DAA	Decimal adjust after addition
DAS	Decimal adjust after subtraction
AAA	ASCII adjust after addition
AAS	ASCII adjust after subtraction
AAM	ASCII adjust after multiplication
AAD	ASCII adjust before division

Logic Instructions

AND	And
OR	Or
XOR	Exclusive or
NOT	Not

Shift and Rotate Instructions

SAR	Shift arithmetic right
SHR	Shift logical right
SAL/SHL	Shift arithmetic left/Shift logical left
SHRD	Shift right double
SHLD	Shift left double
ROR	Rotate right
ROL	Rotate left
RCR	Rotate through carry right
RCL	Rotate through carry left

Bit and Byte Instructions

BT	Bit test
BTS	Bit test and set
BTR	Bit test and reset
BTC	Bit test and complement
BSF	Bit scan forward
BSR	Bit scan reverse
SETE/SETZ	Set byte if equal/Set byte if zero
SETNE/SETNZ	Set byte if not equal/Set byte if not zero
SETA/SETNBE	Set byte if above/Set byte if not below or equal
SETAE/SETMB/SETNC	Set byte if above or equal/Set byte if not below/Set byte if not carry
SETB/SETNAE/SETC	Set byte if below/Set byte if not above or equal/Set byte if carry
SETBE/SETNA	Set byte if below or equal/Set byte if not above
SETG/SETNLE	Set byte if greater/Set byte if not less or equal
SETGE/SETNL	Set byte if greater or equal/Set byte if not less
SETL/SETMGE	Set byte if less/Set byte if not greater or equal
SETLE/SETNG	Set byte if less or equal/Set byte if not greater
SETS	Set byte if sign (negative)
SETNS	Set byte if not sign (non-negative)

String Instructions

MOVS/MOVSB	Move string/Move byte string
MOVS/MOVSW	Move string/Move word string
MOVS/MOVSD	Move string/Move doubleword string
CMPS/CMPSB	Compare string/Compare byte string

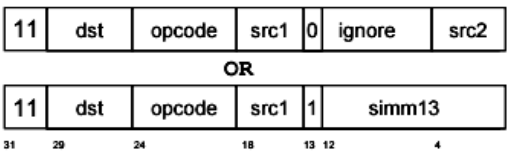
CMPS/CMPSW	Compare string/Compare word string
CMPS/CMPSD	Compare string/Compare doubleword string
SCAS/SCASB	Scan string/Scan byte string
SCAS/SCASW	Scan string/Scan word string
SCAS/SCASD	Scan string/Scan doubleword string
LODS/LODSB	Load string/Load byte string
LODS/LODSW	Load string/Load word string
LODS/LODSD	Load string/Load doubleword string
STOS/STOSB	Store string/Store byte string
STOS/STOSW	Store string/Store word string
STOS/STOSD	Store string/Store doubleword string
REP	Repeat while ECX not zero
REPE/REPZ	Repeat while equal/Repeat while zero
REPNE/REPNZ	Repeat while not equal/Repeat while not zero
INS/INSB	Input string from port/Input byte string from port
INS/INSW	Input string from port/Input word string from port
INS/INSD	Input string from port/Input doubleword string from port
OUTS/OUTSB	Output string to port/Output byte string to port
OUTS/OUTSW	Output string to port/Output word string to port
OUTS/OITSD	Output string to port/Output doubleword string to port

Q.5(f) Explain Sun SPARC Instruction format. [5]

(A)

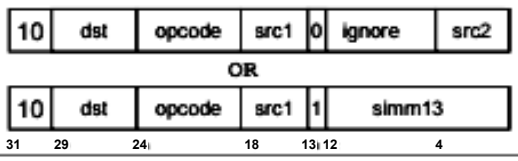
Load Instructions

- Move data from memory to a register
  - ld [u, s] [b, h, d] {a} [*address*], *reg*
- Examples:
  - `ld [%i1],%g2`
  - `ldud [%i1+%i2],%g3`



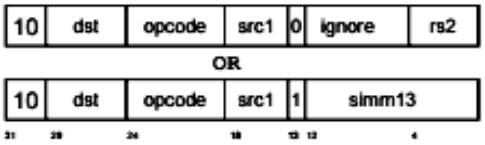
Arithmetic Instructions

- Arithmetic operations on data in registers
  - add{x}{cc} src1, src2, dst — dst = src1 + src2
  - sub{x}{cc} src1, src2, dst — dst = src1 − src2
- Examples:
  - add %01, %02, %03
  - sub %01, 2, %03
- Details
  - src1 and dst must be registers.
  - src2 may be a register or a signed 13-bit immediate.



Bitwise Logical Instructions

- Logical operations on data in registers
  - and{cc} *src1, src2, dst* — *dst = src1 & src2*
  - andn{cc} *src1, src2, dst* — *dst = src1 & ~src2*
  - or{cc} *src1, src2, dst* — *dst = src1 | src2*
  - orn{cc} *src1, src2, dst* — *dst = src1 | ~src2*
  - xor{cc} *src1, src2, dst* — *dst = src1 ^ src2*
  - xnor{cc} *src1, src2, dst* — *dst = src1 ^ ~src2*

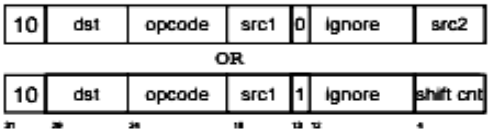


Shift Instructions

- Shift bits of data in registers
  - s[l|r][l|a] *src1*, [*src2* | *0..31*], *dst*

  sll: *dst = src1 << src2;*
  slr: *dst = src1 >> src2;*

- Details
  - do not modify condition codes
  - sll and srl fill with 0, sra fills with sign bit
  - no sla



Floating Point Instructions

- Performed by floating point unit (FPU)
- Use 32 floating point registers: `%f0..%f31`
- Load and store instructions
  - ld *[address],freg*
  - ldd *[address],freg*
  - st *freg,[address]*
  - std *freg,[address]*
- Other instructions are FPU-specific
  - fmovs,fsqrt,fadd,fsub,fmul,fdiv,…

❑ ❑ ❑ ❑ ❑