COA Assignment U2

1.a Draw a block diagram of address Sequencing and Explain each blocks.

1.b Explain The micro program Examples

Address Sequencing

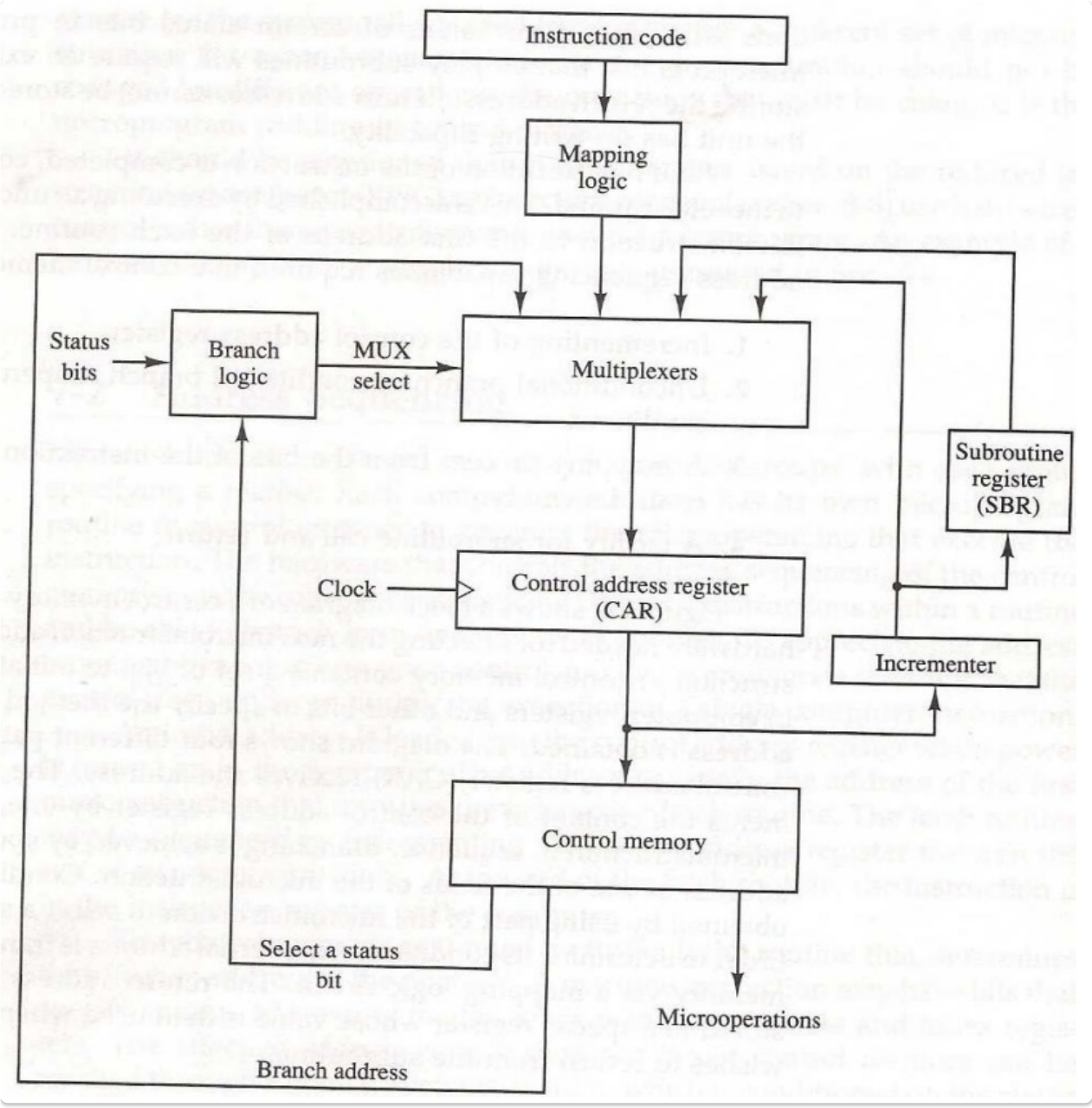
- Microinstructions are stored in control memory in groups, with each group specifying a routine.
- Each computer instruction has its own microprogram routine to generate the microoperations.
- The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be able to branch from one routine to another.
- Steps the control must undergo during the execution of a single computer instruction:
    - Load an initial address into the CAR when power is turned on in the computer. This address is usually the address of the first microinstruction that activates the instruction fetch routine – IR holds instruction.
    - The control memory then goes through the routine to determine the effective address of the operand – AR holds operand address.
    - The next step is to generate the microoperations that execute the instruction by considering the opcode and applying a mapping process.
        - The transformation of the instruction code bits to an address in control memory where the routine of

> instruction located is referred to as the mapping process.
>
> - After execution, control must return to the fetch routine by executing an unconditional branch.

In brief, the address sequencing capabilities required in control memory are:

- Incrementing of the control address register.
- Unconditional branch or conditional branch, depending on status bit conditions.
- A mapping process from the bits of the instruction to an address for control memory.
- A facility for subroutine call and return.

The below figure shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.



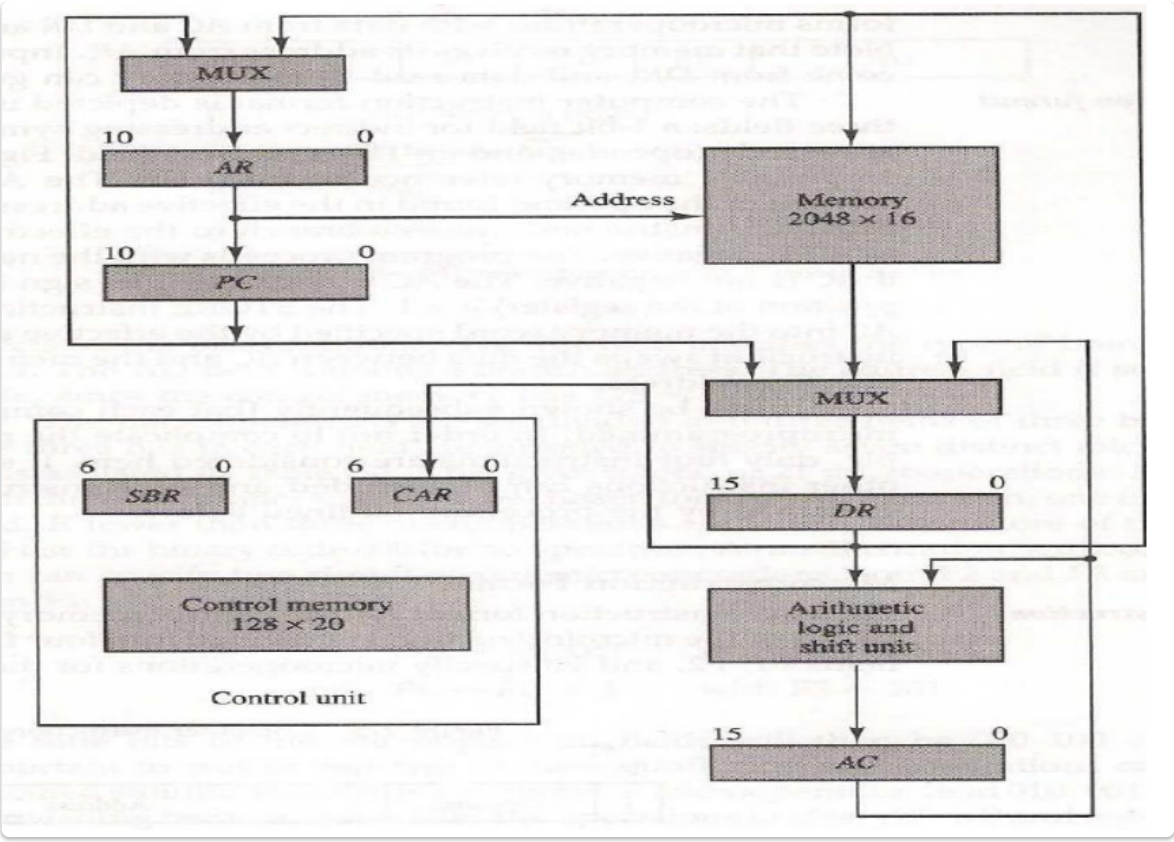
- The microinstruction in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained.
- In the figure, four different paths form which the control address register (CAR) receives the address:
    1. The incrementer increments the content of the control register address register by one, to select the next microinstruction in sequence.
    2. Branching is achieved by specifying the branch address in one of the fields of the microinstruction.

3. Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.
4. An external address is transferred into control memory via a mapping logic circuit.

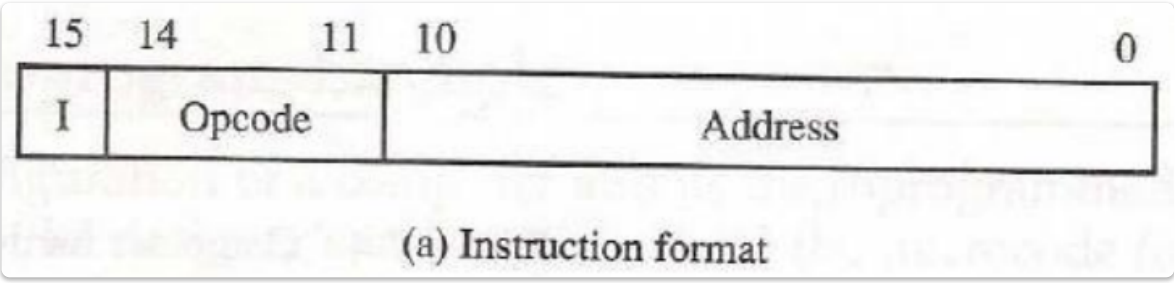
- The return address for a subroutine is stored in a special register; that value is used when the microprogram wishes to return from the subroutine.

Microprogram Example

- The process of code generation for the control memory is called microprogramming.
- The block diagram of the computer configuration is shown in the figure below.
- Two memory units:
  - Main memory – stores instructions and data
  - Control memory – stores microprogram
- Four processor registers:
  - Program counter – PC
  - Address register – AR
  - Data register – DR
  - Accumulator register - AC
- Two control unit registers:
  - Control address register – CAR
  - Subroutine register – SBR
- Transfer of information among registers in the processor is through MUXs rather than a bus.



- The computer instruction format is shown in the figure below.



(a) Instruction format

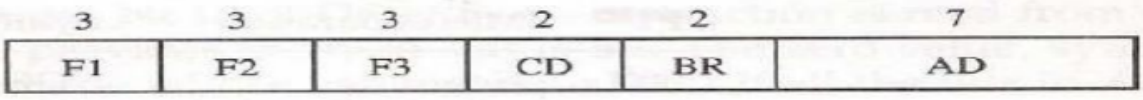
  - Three fields for an instruction:
    - 1-bit field for indirect addressing
    - 4-bit opcode
    - 11-bit address field
- The example will only consider the following 4 of the possible 16 memory instructions.

Symbol	Opcode	Description
ADD	0000	$AC \leftarrow AC + M\,[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M\,[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

EA is the effective address

(b) Four computer instructions

- The microinstruction format for the control memory is shown in the figure below.



F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

Figure 7-6 Microinstruction code format (20 bits).

  - The microinstruction format is composed of 20 bits with four parts to it:
    - Three fields F1, F2, and F3 specify microoperations for the computer [3 bits each]
    - The CD field selects status bit conditions [2 bits]
    - The BR field specifies the type of branch to be used [2 bits]
    - The AD field contains a branch address [7 bits]
- Each of the three microoperation fields can specify one of seven possibilities.
- No more than three microoperations can be chosen for a microinstruction.
- If fewer than three are needed, the code 000 = NOP.
- The three bits in each field are encoded to specify seven distinct microoperations listed in the table below.

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

- The condition field (CD) is two bits to specify four status bit conditions shown below.

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	$DR(15)$	I	Indirect address bit
10	$AC(15)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

- The branch field (BR) consists of two bits and is used with the address field to choose the address of the next microinstruction.

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2\text{–}5) \leftarrow DR(11\text{–}14)$, $CAR(0,1,6) \leftarrow 0$

- Each line of an assembly language microprogram defines a symbolic microinstruction and is divided into five parts:

    1. The label field may be empty or it may specify a symbolic address. Terminate with a colon (:).

2. The microoperations field consists of 1-3 symbols, separated by commas. Only one symbol from each field. If NOP, then translated to 9 zeros.
3. The condition field specifies one of the four conditions.
4. The branch field has one of the four branch symbols.
5. The address field has three formats:
    - A symbolic address – must also be a label
    - The symbol NEXT to designate the next address in sequence
    - Empty if the branch field is RET or MAP and is converted to 7 zeros.

- The symbol ORG defines the first address of a microprogram routine.
- ORG 64 – places the first microinstruction at control memory 1000000.

2.a Discuss about addressing Modes

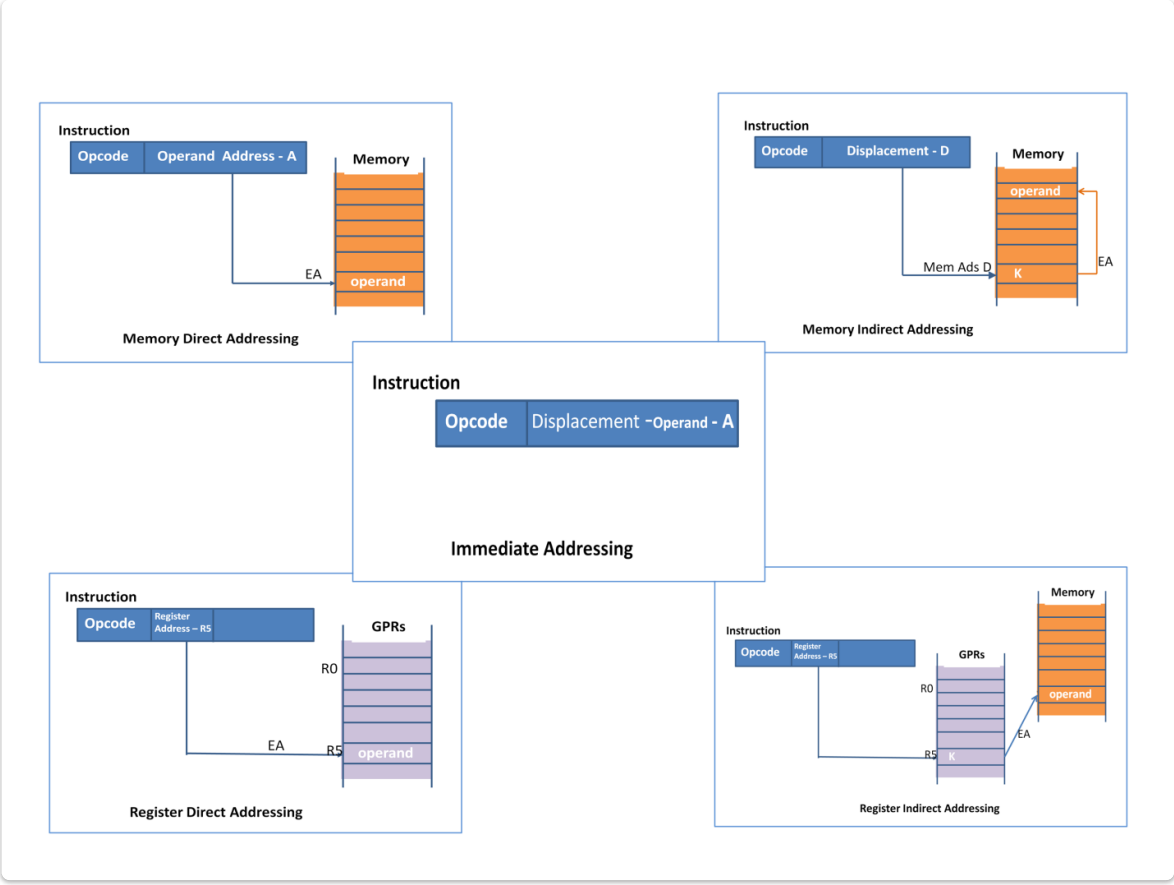
2.b Explain The data transfer Instruction

Addressing Modes

Addressing Modes

- The way operands are chosen during program execution depends on the addressing mode of the instruction.
- Computers use addressing mode techniques to provide programming versatility and to reduce the number of bits in the addressing field of the instruction.
- Most addressing modes modify the address field of the instruction; two modes that need no address field at all are

implied and immediate modes.



Implied Mode:

- In this mode, operands are specified implicitly in the definition of the instruction.
- For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.
- All register reference instructions that use an accumulator are implied mode instructions.
- Zero address in a stack organization computer is implied mode instructions.

Immediate Mode:

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

- Immediate-mode instructions are useful for initializing registers to a constant value.
- The address field of an instruction may specify either a memory word or a processor register.
- When the address specifies a processor register, the instruction is said to be in the register mode.

Register Mode:

- In this mode, the operands are in registers that reside within the CPU.
- The particular register is selected from a register field in the instruction.

Register Indirect Mode:

- In this mode, the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
- The selected register contains the address of the operand rather than the operand itself.
- The advantage is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

Auto-increment or Auto-Decrement Mode:

- Similar to the register indirect mode, but the register is incremented or decremented after (or before) its value is used to access memory.
- The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory.
- Sometimes the value given in the address field is the address of the operand, but sometimes it is just an address from which the address of the operand is calculated.

Direct and Indirect Address Modes:

Direct Address Mode:

- In this mode, the effective address is equal to the address part of the instruction.
- The operand resides in memory, and its address is given directly by the address field of the instruction.
- In a branch-type instruction, the address field specifies the actual branch address.

Indirect Address Mode:

- In this mode, the address field of the instruction gives the address where the effective address is stored in memory.
- Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.
- A few addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU.
- The effective address in these modes is obtained from the following computation:
  `Effective address = address part of instruction + content of CPU register`
- The CPU register used in the computation may be the program counter, an index register, or a base register.

Special Addressing Modes:

Relative Address Mode:

- In this mode, the content of the program counter is added to the address part of the instruction to obtain the effective address.

Indexed Addressing Mode:

- In this mode, the content of an index register is added to the address part of the instruction to obtain the effective address.
- An index register is a special CPU register that contains an index value.

Base Register Addressing Mode:

- In this mode, the content of a base register is added to the address part of the instruction to obtain the effective address.
- This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.

Data Transfer Instructions

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.
- Table 8-5 gives a list of eight data transfer instructions used in many computers.

TABLE 8-5 Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

- The load instruction has been used mostly to designate a transfer from memory to a processor register, usually an accumulator.
- The store instruction designates a transfer from a processor register into memory.
- The move instruction has been used in computers with multiple CPU registers to designate a transfer from one register to another and also between CPU registers and memory or between two memory words.
- The exchange instruction swaps information between two registers or a register and a memory word.
- The input and output instructions transfer data among processor registers and input or output terminals.
- The push and pop instructions transfer data between processor registers and a memory stack.
- Different computers use different mnemonic symbols to differentiate addressing modes.
- As an example, consider the load to accumulator instruction when used with eight different addressing modes.

- Table 8-6 shows the recommended assembly language convention and actual transfer accomplished in each case.

TABLE 8-6 Eight Addressing Modes for the Load Instruction

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD $ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$

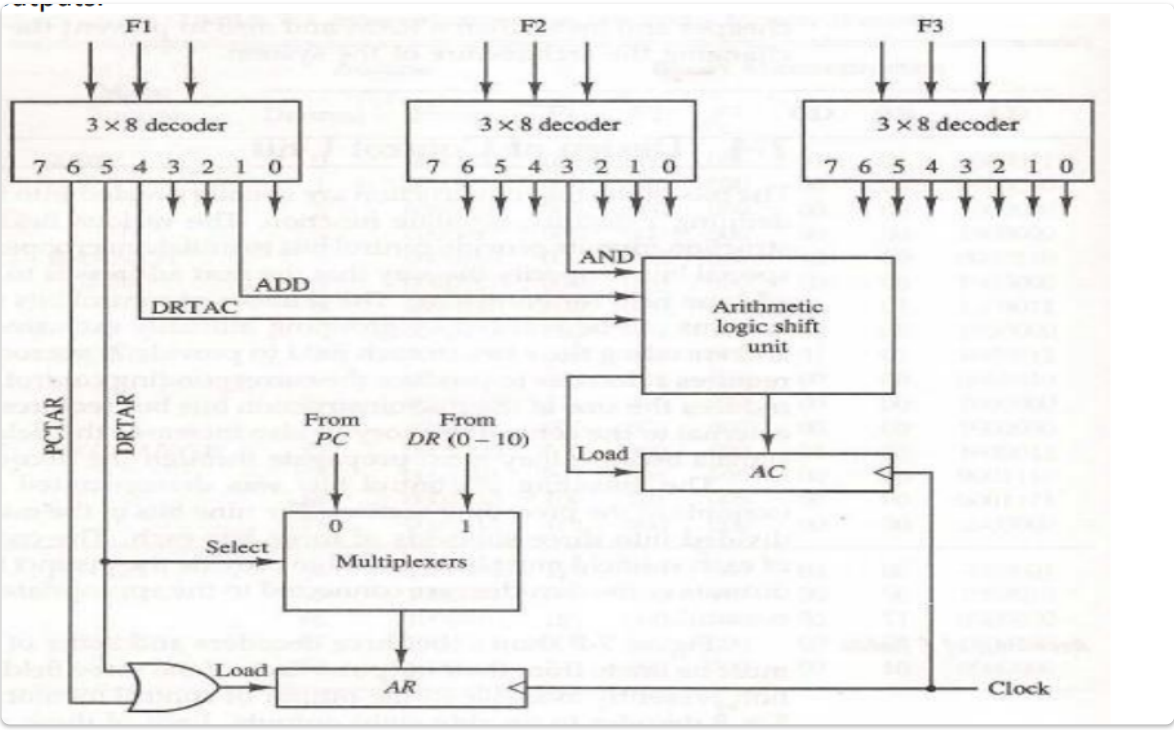
- ADR stands for an address.
- NBA is a number or operand.
- X is an index register.
- R1 is a processor register.
- AC is the accumulator register.
- The `@` character symbolizes indirect addressing.
- The `$` character before an address makes the address relative to the program counter PC.
- The `#` character precedes the operand in an immediate-mode instruction.
- An indexed mode instruction is recognized by a register placed in parentheses after the symbolic address.
- The register mode is symbolized by giving the name of a processor register.
- In the register indirect mode, the name of the register that holds the memory address is enclosed in parentheses.
- The auto-increment mode is distinguished from the register indirect mode by placing a plus after the parenthesized register. The auto-decrement mode would use a minus instead.

3.a Explain The design of control unit

3.b Explain The data manipulation interaction reactions.

Design of Control Unit

- The control memory, out of each subfield, must be decoded to provide distinct microoperations.
- The outputs of the decoders are connected to the appropriate inputs in the processor unit.
- The figure below shows three decoders and some of the connections that must be made from their outputs.



- The three fields of the microinstruction in the output of the control memory are decoded with a 3x8 decoder to provide eight outputs.
- Each output must be connected to the proper circuit to initiate the corresponding microoperation as specified in the previous topic.
- When F1 = 101 (binary 5), the next pulse transition transfers the content of DR (0-10) to AR.

- Similarly, when F1 = 110 (binary 6), there is a transfer from PC to AR (symbolized by PCTAR). Outputs 5 and 6 of decoder F1 are connected to the load input of AR so that when either one of these outputs is active, information from the multiplexers is transferred to AR.
- The multiplexers select information from DR when output 5 is active and from PC when output 5 is inactive.
- The transfer into AR occurs with a clock transition only when output 5 or output 6 of the decoder is active.
- For the arithmetic logic shift unit, the control signals are instead of coming from the logical gates; now, these inputs will come from the outputs of AND, ADD, and DRTAC, respectively.

Data Manipulation Instructions

1. Arithmetic Instructions

- Data manipulation instructions perform operations on data and provide computational capabilities for the computer.
- The data manipulation instructions in a typical computer are usually divided into three basic types:

1.1 Arithmetic Instructions

- The four basic arithmetic operations are addition, subtraction, multiplication, and division.
- Most computers provide instructions for all four operations.
- Some small computers have only addition and possibly subtraction instructions. Multiplication and division must then be generated by means of software subroutines.

- The increment instruction adds 1 to the value stored in a register or memory word.
- A number with all 1's, when incremented, produces a number with all 0's.
- The decrement instruction subtracts 1 from a value stored in a register or memory word.
- A number with all 0's, when decremented, produces a number with all 1's.
- The add, subtract, multiply, and divide instructions may use different types of data.
- The data type assumed to be in the processor register during the execution of these arithmetic operations is defined by an operation code.
- An arithmetic instruction may specify fixed-point or floating-point data, binary or decimal data, single-precision, or double-precision data.
- Examples of mnemonics for three add instructions that specify different data types are shown below:
    - ADDI: Add two binary integer numbers
    - ADDF: Add two floating-point numbers
    - ADDD: Add two decimal numbers in BCD
- Special instructions like "add carry" and "subtract with borrow" deal with the carry from a previous operation.
- The negate instruction forms the 2's complement number, effectively reversing the sign of an integer when represented in signed-2's complement form.

2. Logical and Bit Manipulation Instructions

- Logical instructions perform binary operations on strings of bits stored in registers.

- They are useful for manipulating individual bits or groups that represent binary-coded information.
- Typical logical and bit manipulation instructions include:
  - Clear
  - Complement
  - AND, OR, XOR
  - Bit manipulation operations: clear, set, complement
- Other bit manipulation instructions perform operations on individual bits, such as clearing, setting, or complementing.

3. Shift Instructions

- Shifts are operations in which the bits of a word are moved to the left or right.
- Shift instructions may specify logical shifts, arithmetic shifts, or rotate-type operations.
- Types of shift instructions include logical shift left/right, arithmetic shift left/right, rotate left/right, and rotate through carry.
- Shifts may be used for various purposes, including moving bits to designated positions, preserving sign bits in arithmetic shifts, and performing circular shifts.
- A list of typical arithmetic instructions is given in Table 8-7.
- The logical shift inserts 0 into the end bit position.
- Arithmetic shifts usually conform to the rules for signed-2's complement numbers.
- Rotate instructions produce a circular shift, circulating bits shifted out back into the opposite end.
- Rotate through carry treats a carry bit as an extension of the register being rotated.