UNIT 2

MICROPROGRAMMED CONTROL AND CENTRAL PROCESSING UNIT

PART-A
SHORT QUESTIONS WITH SOLUTIONS

Q1. Write short notes on subroutines.

Answer :

Subroutines are just like the interrupt routines which are executed after suitable jump to desired location in the given micro instruction program, the major difference between interrupt routine and subroutines is, that the interrupt is caused explicitly by an hardware unit or any software programs, but subroutines are part of given microprogram which are initiated implicitly. Hence whenever a given subroutine is initiated, the program control jumps to the specified location by storing its current status in a register. On returning from the routine the program control acquires the status from the register and starts processing on with its prescribed task. There may be any number of subroutines present in the given microprogram and also a single subroutine can be called more than once in a given microprogram.

Q2. Discuss in brief about microinstruction format.

Answer :

Model Paper-II, Q1(c)

The microinstruction format represents the way a control unit organizes its microinstructions or it provides the details of different blocks forming an microinstruction.

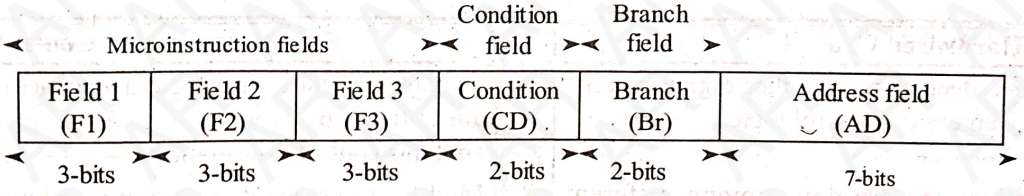


Figure: Microinstruction Format for Control Memory

Each field in the above microinstruction has a unique functionality. The above microinstruction is of 20-bits wide and the distribution of these bits are represented in the above figure. The fields 'F1', 'F2' and 'F3' are microinstruction fields. Length of these fields are 3-bits wide, hence each field can generate 7 combinations forming a total of 21 microoperations. The condition field is only of 2-bits hence, it can generate four possible combinations. The purpose of condition field is to raise a specific condition field in order to decide which operation is to be performed. The branch field is also 2-bits wide forming four possible combinations. The purpose of branch field is to provide the type of branch.

Q3. Write about binary microprogram.

Answer :

Binary microprogram is nothing but the binary representation of a symbolic program. As all the instructions in this program are interpreted in the form of 0's and 1's, hence it can be directly stored in the memory without further translation. Any symbolic microprogram is quite easy writing, since most of the instructions are just observated form of normal English words. But when binary microprogram is considered, it is too tedious to represent since the binary equivalent values for each symbol has to remembered and it has to be placed at proper locations so as to attain the desired results. Here there is a possibility of occurrence of two things (i.e.,) either any compiler should be considered to translate a given symbolic microprogram into it's binary equivalent microprogram or it should be directly interpreted by suitable microprocessor programmer.

Q4. List the major operations performed by sequencer on next address generator.

Answer :

The major operations which are performed by the sequencer on next address generator are listed below,

- ❖ Address sequencing operations
- ❖ Push and pop operations
- ❖ Transmitting program control to the subroutine and returning operations.
- ❖ Increment operations
- ❖ Jump operations at specific location etc.

Q5. Describe stack organization.

Answer :

Model Paper-III, Q1(c)

Stack Organization

A computer that conforms to this type of organization will use the PUSH and POP instructions which use only one address field.

Example

PUSH C

This instruction will push the word present at address C to the top of the stack.

The operation-type instruction used within a stack organized computer, will not require an address field and can be termed as zero-address field instruction.

Example

ADD

This instruction will add the two topmost numbers on the stack by popping them out, computing their sum and pushing the result on top of the stack.

Q6. Distinguish between hardwired control unit and microprgrammed control unit.

Answer :

Model Paper-II, Q1(d)

	Hardwired Control		Microprogrammed Control
1.	Gates, flip-flops, decoders, and other digital circuits are used to implement the control logic.	1.	To implement the control logic, a sequence of microoperations are initiated by programming the control memory which contains control information.
2.	Design modifications are done among different components by making changes in wiring.	2.	Modifications are done among various components by updating control memory with a micro program.
3.	Most of the RISC (Reduced Instruction Set Computer) architecture use hardwired.	3.	Most of the RISC architecture does not make use of micro-programmed control.
4.	A hardwire is organised by using multiple wires.	4.	A microprogram is organized by using sequence of microinstructions.

Q7. What are the three basic fields present in an instruction?

Answer :

Model Paper-III, Q1(d)

The three basic fields that are present in an instruction.

1. **Opcode Field**

It is an operation code field that indicates a specific operation to be performed.

2. **Address Field**

It is a field where a particular memory address or a processor register is specified.

3. **Mode Field**

It is a field which provides a way to determine the operand or its effective address.

Q8. Explain the following,

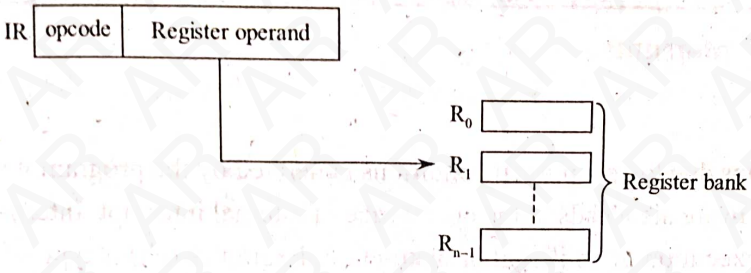
(i) Register operand addressing mode

(ii) Immediate operand addressing mode.

Answer :

(i) Register Operand Addressing Mode

With the register addressing mode, the operand to be accessed is present in CPU registers. Actually all computer systems are usually provided with some addressable registers. The k bits of the operand field can denote any one of 2^k registers operand holding the data word.



(ii) Immediate Operand Addressing Mode

In this mode, operand is part of the instruction instead of the contents of a register or memory location. Instruction format for immediate addressing mode is shown in the following figure.

opcode	Immediate operand

As shown in the above figure immediate addressing mode has an operand field rather than address field. Since the data are encoded directly into the instruction, immediate operands normally represent constant data.

Example

MVI A, 10

MVI : Move Immediate, 10 is stored in AC

Computer requires no memory reference, operand itself is present in the instruction.

Q9. Write short notes on interrupt.

Answer : **Model Paper-I, Q1(c)**

An interrupt is an asynchronous event that halts the normal program execution and diverts the program flow temporarily to an interrupted routine. Interrupts are caused both by the hardware and software. On occurrence of an interrupt, the current status of the running program is stored and the control is transferred to ISR (which is responsible for handling interrupts) after its execution, the control switches back to the execution of the suspended program.

Q10. Write about zero-address instruction.

Answer :

Zero-address Instruction

1. It does not contain any operand address i.e., no explicit operands are present in the instruction.
2. Its general instruction format is,

Operation	Source 1	Source 2	Destination

3. It stores its operands on stack in memory.
4. It is also called as stack organization.
5. Example: Instruction:

Push P

Push 8

Add

Pop P

Computes:

Increment P by 8.

Q11. List any two differences between one-address and three address instructions.

Answer :

One-address Instructions		Three-address Instructions	
1.	It contains only one operand addresses in the instruction i.e., only one explicit operand is specified to one instruction.	1.	It contains two addresses for two operands i.e., three explicit operands are specified to each instruction.
2.	It stores its operands in the accumulator.	2.	It stores its variable names on distinct previously stored locations in memory.

Q12. Discuss about internal interrupts.

Answer :

Model Paper-I, Q1(d)

They are also known as traps due to exceptional conditions generated by the program itself but not by the external event. Signals generated in the CPU hardware also leads to the occurrence of internal interrupt. Internal interrupts are generated due to error condition produced by the execution of an illegal or instruction. Examples of this type of interrupts are register overflow, division by zero, use of unacceptable operation codes, stack overflow and safety or protection breach. The programmer experience such condition during the premature termination of instruction execution.

PART-B

ESSAY QUESTIONS WITH SOLUTIONS

2.1 MICROPROGRAMMED CONTROL

2.1.1 Control Memory

Q13. Explain about the microprogrammed control organization.

Answer :

Model Paper-III, Q4(a)

The way various blocks forming a control unit are organized and are often referred as microprogrammed control organization. A block schematic representing the microprogrammed control organization is given below,

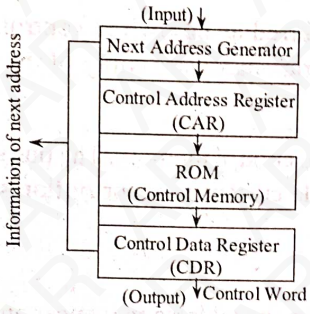


Figure: Interconnection Structure of Different Blocks Forming a Microprogrammed Control Organization

The block diagram representing different blocks forming a microprogrammed control unit is shown above. Each of these blocks possesses a unique functionality to be performed. Any system maintaining microprogrammed control unit utilizes the services of two types of memories referred as control memory and main memory. The control memory is nothing but a Read Only Memory or simply 'ROM' which stores special type of programs referred as microprograms. The microprograms are nothing but the combination of finite number of microinstructions. Whenever, these microinstructions are executed, it results in a series of timing signals which are responsible for controlling various microoperations. As it is an read only memory, hence the data present in these memories can neither be altered nor it can be rewritten. On the other hand, the main memories uses related data. The data stored in these maintains user related data. The data stored in these memories can be altered as well as rewritten depending on the convenience of the user. The Control Address Register or simply 'CAR' is responsible to store the addresses of micro instructions stored in the memories. Meanwhile control data registers stores the microinstructions fetched from the memory. Not waiting till the end of execution of given instruction, there is another block referred as Next address generation or sequencer which maintains the address of the next instruction to be executed, even before current execution of the instruction halts. In the above block diagram, there is a repeating line from CDR to Next address generator. This is because, few bits of the current instruction (which is under execution) are fetched by next address generator block. These bits are helpful in generation of next consecutive instruction. Hence the next address is generated by combining these bits along with the external input. As the next address generator block is responsible for fetching next consecutive instruction, therefore it is also called as sequencer. As the address of a next instruction is fetched, even though the current instruction is under execution in control data register, the control address register is also referred as pipelined register. There are many advantages with this organization. The major among them is that, it provides hardware independence (i.e.,) if there is a requirement to change the functionality of the system, then the microprograms stored in control memory should be changed rather changing the systems hardware configuration.

Q14. Explain the following terms,

(i) Control word
(ii) Microinstructions
(iii) Microprogram
(iv) Hardwired control
(v) Pipeline register
(vi) Control address register
(vii) Control memory
(viii) Sequencer.

Answer :

(i) Control Word

It is a known fact that the control unit is responsible for controlling various microoperations. During certain course of time it disables certain operations at the same time enables various other. The control variables which are associated with circuitry can be represented using binary values. These values are often referred as control word.

(ii) Microinstructions

The control words, belonging to control unit possesses certain instructions, usually referred as microinstructions. A microinstruction denotes a single or a group of microoperations for the system.

(iii) Microprogram

Several microinstructions together constitute a microprogram.

(iv) Hardwired Control

The control unit of a digital computer is referred as hardwired control, since it generates various signals to control the operations of a computer while being a separate hardware entity of the system.

(v) Pipeline Register

There are various registers associated with the control memory. The purpose of these registers are, simultaneously generate the next microinstruction while executing the current microoperation specified by the control word. These registers are referred as pipeline register.

(vi) Control Address Register

There is a register which is located in between the address sequencer and control memory in a microprogrammed control organization. The speciality of these registers is that, it stores the address of certain microinstructions. These kind of registers are referred as control address register.

(vii) Control Memory

The memory unit associated with a control unit, which gives the privilege to read and write data is referred as control memory.

(viii) Sequencer

The control unit includes a special unit referred as sequencer or address sequencer, which determines provides a sequence of addresses of instructions which are to be read from the control memory.

2.1.2 Address Sequencing

Q15. Diagrammatically explain the process of selection of address for control memory.

Answer :

Model Paper-III, Q4(b)

The representation of process of address selection is depicted in the below figure,

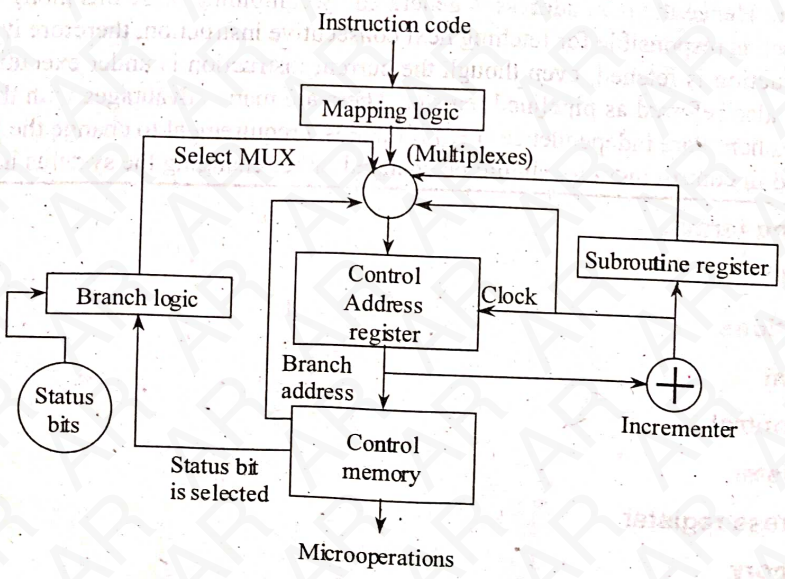


Figure: Representation of Process of Address Selection

In order to implement the unconditional branch instruction in the above circuitry, the control address register is required to be loaded with desired branch address fetched from the control memory. This is possible by initially fixing one of the values of status bits '1' and providing it to the multiplexer. As a result, the required address gets stored in the control address register. While dealing with the selection of address for control memory one has to remember that the given status bit reflects certain special conditions such as sign of a number or carry out of adder etc. This can be estimated by closely analyzing the values of these bits (i.e.,) their values is either '0' or '1'. Also various micro-operations gets generated only after testing the information provided by these bits.

The branch logic unit is responsible for relocating the control to the desired branch instruction, provided that the desired conditions should be met.

Q16. Describe how mapping from instruction code to microinstruction is done. Also, explain about subroutines.

Answer :

Model Paper-I, Q4(a)

Mapping from Instruction Code to Microinstruction

It is a known fact that each microprogram is a combination of finite number of microinstructions. Some times these microinstructions, when executed, refers to some routines which are situated at first word location of the control memory. These microinstructions maintain certain special branches, whose status bits are nothing but the bits of operation code or opcode block of the given microinstruction (which is responsible for executing it). These operation codes processes separate routines stored in the control memory. While dealing with the operation code of the microinstruction one has to remember that the most common length of it (opcode) is 4-bits, which can provide a maximum of 16 combinations. But, on the other hand the control memory can accommodate about 128 words. To address these 128 words, there is a requirement of 7-bits. Hence converting 4-bits of opcode into seven bits of address to refer, all the possible words of control memory is accomplished only by means of mapping.

This is achieved by suitably combining bits belonging to different blocks (i.e.,) initially bit '0' is placed at MSB position later 4-bits belonging to the opcode of the instructions appended to zero. Finally the last two bits (in order to complete a total of seven bits) is acquired from control address register. Hence this can be diagrammatically represented as,

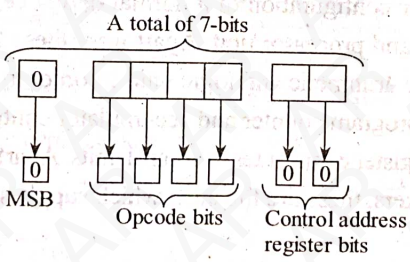


Figure: Mapping Concepts

Hence, the computer instruction are now left out with three options (i.e.,) if the microprogram routines which are associated with the given computer instructions requires to address only four macroinstructions then, it is privileged to use only four bits of the opcode (or) if the computers instructions requires to address more than four microinstructions, then it can utilize the seven possible bits by adopting suitable mapping concepts. Finally, if the given computer instruction requires less than four micro instructions to be addressed then the empty regions of the control memory can be utilized for various other purpose. In order to still simplify the mapping mechanisms, a ROM is introduced here, the bits belonging to the instructions directs the mapping ROM, which in turn directs to the location of control address register. In general terms the mapping functions are internally maintained by using small electronic chips referred as PLDs (Programmable Logic Device).

Subroutines

Subroutines are just like the interrupt routines which are executed after suitable jump to desired location in the given micro instruction program, the major difference between interrupt routine and subroutines is, that the interrupt is caused explicitly by an hardware unit or any software programs, but subroutines are part of given microprogram which are initiated implicitly. Hence whenever a given subroutine is initiated, the program control jumps to the specified location by storing its current status in a register. On returning from the routine the program control acquires the status from the register and starts processing on with its prescribed task. There may be any number of subroutines present in the given microprogram and also a single subroutine can be called more than once in a given microprogram.

2.1.3 Microprogram Example

Q17. With the help of a block diagram explain the computer configuration.

Answer : Model Paper-III, Q5(a)

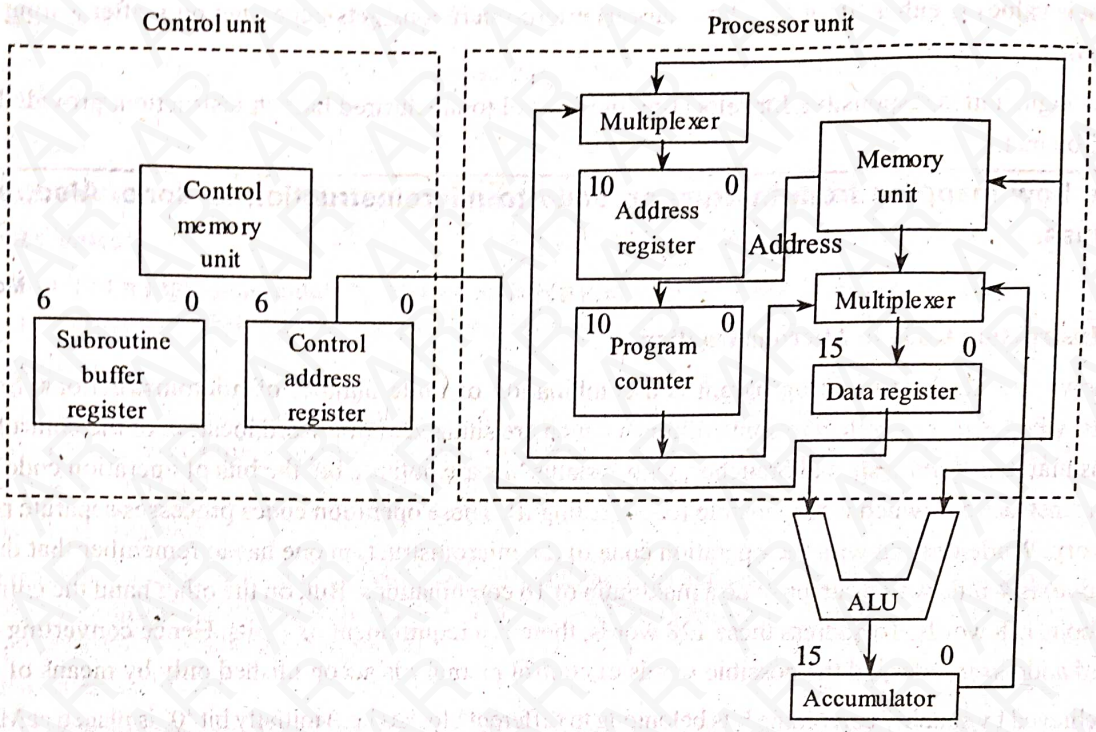


Figure: Computer Hardware Configuration

The above diagram reflects the computer configuration of a normal digital computer. It can be seen that the organization is divided into two major modules as control unit and processor unit. Apart from these two modules, there is a third block which also forms the major entity of this organization at the arithmetic and logic unit. In order to store data temporarily there are certain sets of registers namely address register, data register, program counter and accumulator contributes to processor unit where as two registers namely subroutine register and control address register contributes to control units. Apart from these registers, there are two special units, existing in the processor unit (i.e.,) the multiplexers, these are the units which supplies data to all the registers belonging to processor unit.

Finally, the main functionality of arithmetic and logic unit is to perform all the arithmetic and logical operations by considering data from registers like accumulator and data register respectively. The operations performed by the ALU unit are referred as microoperations whose results gets stored again into the accumulator. Also, for memory unit (belonging to the processor unit) addresses are provided by the address register. Data register can read the data from the memory, program counter or the accumulator.

Q18. Discuss in detail about all the microinstruction formats. Also explain various microinstruction fields.

Answer : Model Paper-I, Q5

The microinstruction format represents the way a control unit organizes its microinstructions or it provides the details of different blocks forming an microinstruction.

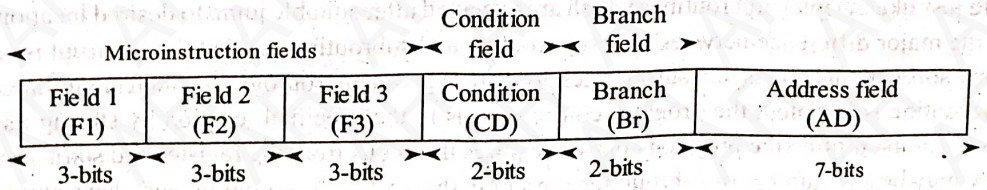


Figure: Microinstruction Format for Control Memory

Each field in the above microinstruction has a unique functionality. The above microinstruction is of 20-bits wide and the distribution of these bits are represented in the above figure. The fields 'F1', 'F2' and 'F3' are microinstruction fields. Length of these fields are 3-bits wide, hence each field can generate 7 combinations forming a total of 21 microoperations. The condition field is only of 2-bits hence, it can generate four possible combinations. The purpose of condition field is to raise a specific condition field in order to decide which operation is to be performed. The branch field is also 2-bits wide forming four possible combinations. The purpose of branch field is to provide the type of branch.

Symbols of Various Microinstruction Fields

Following are the symbols and equivalent binary 3 digit values indicating the first microinstruction field (*F*1).

Sl.No	3 bit Binary values (F1)	Equivalent Micro-operation	Symbol	Description
0	000	–	NOP	No operation
1	001	AC ← AC + DR	ADD	Add the contents of DR to the contents of a AC and store the result in AC.
2	010	AC ← 0	CLRAC	Clear the contents of AC.
3	011	AC ← AC + 1	INCAC	Increment the contents of accumulator.
4	100	AC ← DR	DRTAC	Transmit contents of accumulator into data register.
5	101	AR ← DR (0 – 10)	DRTAR	Transmit the address of instruction contained in data register into address register.
6	110	AR ← PC	PCTAR	Transmit the address of instruction from program counter into address register.
7	111	M[AR] ← DR	WRITE	Write the contents present in data register in the memory whose address is specified by the address register.

Following are the symbols and equivalent binary 3-digit values indicating the second microinstruction field (*F*2).

Sl.No	3 bit Binary values (F2)	Equivalent Micro-operation	Symbol	Description
0	000	–	NOP	No operation
1	001	AC ← AC – DR	SUB	Subtract the contents of data register from the contents of accumulator and store the result in accumulator.
2	010	AC ← AC ∨ DR	OR	OR the contents of data register to the contents of accumulator and store the result in accumulator.
3	011	AC ← AC ∧ DR	AND	AND the contents of data register with the contents of accumulator and store it in the accumulator.
4	100	DR ← M[AR]	READ	READ the instruction from the memory whose address is specified by the address register and store it in the data register.
5	101	DR ← AC	ACTDR	Transmit the contents of accumulator into the data register.
6	110	DR ← DR + 1	INCDR	Increment the contents of data register by '1'.
7	111	DR(0 – 10) ← PC	PCTDR	Transmit the instruction associated with the addresses stored in program counter into the data register.

The following are the symbols and equivalent binary 3-digit values indicating the third microinstruction filed (*F3*).

Sl.No	3 bit Binary values (F3)	Equivalent Micro-operation	Symbol	Description
0	000	–	NOP	No operation
1	001	AC ← AC + DR	XOR	Perform Ex-OR operation by considering the contents of data register with contents of accumulator and store the result in the accumulator.
2	010	AC ← $\overline{AC}$	COM	Complement the contents of accumulator and store the result again in the accumulator.
3	011	AC ← Shl AC	SHL	Shift the contents of accumulator to 1 bit left and store the result in the accumulator.
4	100	AC ← Shr AC	SHR	Shift the contents of accumulator to 1-bit right and store the result in the accumulator.
5	101	PC ← PC + 1	INCPC	Increment the program counter.
6	110	PC ← AR	ARTPC	Transmit the address stored in address register into the program counter.
7	111	Reserved	–	Not-specified

Following are the symbols and their equivalent binary 2-digit conditional values indicating the fourth microinstruction field (CD).

CD (Two bit binary values)	Symbol used	Equivalent Condition	Description
00	U	It's value is always '1'	It represents an unconditional branch.
01	I	DR(15)	It is an indirect address bit.
10	S	AC(15)	It refers to sign bit of accumulator.
11	Z	AC = 0	Store a value zero in the accumulator which reflects that the accumulator is empty.

Following are the symbols and equivalent binary 2-digit values indicating the fifth microinstruction field (BR or branch field).

BR (Two bit binary values)	Symbol used	Microoperation	Description
00	JMP	CAR ← AD	This microoperation occurs only if the condition = 1.
		CAR ← CAR + 1	This microoperation occurs only if condition = 0.
01	CALL	CAR ← AD, SBR ← CAR + 1	This microoperation occurs only if the condition = 1.
		CAR ← CAR + 1	This microoperation gets executed only if condition = 0.
10	RET	CAR ← SBR	This microinstruction indicates return from subroutine.
11	MAP	CAR (2 – 5) ← DR (11 – 14), CAR (0, 1, 6) ← 0	It indicates a mapping function.

Q19. Describe briefly about Fetch routine.

Answer :

Whenever fetch routine is considered, it is the control unit which plays a major role. The control memory (belonging to control unit) exerts the capacity of storing about 128 words of data. These 128 words are divided into two half of 64 words each. Here it has to be noted that, a word in general comprises of 20 bits each. Moreover the initial 64 words gets occupied by various routines belonging to a set of 16 instructions and the rest 64 words can be accommodated by the desired blocks of data.

Following are three microinstructions essential in order to generate fetch routines.

AR ← PC // Load the address of the instruction present in program counter into the address register.

DR← M[AR], PC ← PC + 1 // Initially, the instruction referred by address register is transferred from memory into // data register. And later the value of program counter is incremented by '1'.

AR ← DR(0 – 10), CAR(2 – 5) ← DR (11 – 14),

CAR(0, 1, 6) ← 0 // Initially, theaddresses are transmitted from data register into address register. // Later, the control is provided to various instruction routines belonging to 16 instructions.

Q20. Discuss in detail about the symbolic microprogram.

Answer :

A simple symbolic microprogram representing only few lines of code belonging to single large program is as given below,

	Label	Microoperations	Conditions (CD)	Branch (BR)	Address (AD)
		ORG 11			
(11)	ADD:	NOP	I	CALL	INDRCT
(12)		READ	U	JMP	NEXT
(13)		ADD	U	JMP	FETCH
		ORG 15			
(15)	STORE:	NOP	I	CALL	INDRCT
(16)		ACTDR	U	JMP	NEXT
(17)		WRITE	U	JMP	FETCH
		ORG 19			
(19)	EXCHANGE:	NOP	I	CALL	INDRCT
(20)		READ	U	JMP	NEXT
(21)		ACTDR, DRTAC	U	JMP	NEXT
(22)		WRITE	U	JMP	FETCH
		ORG 24	S	JMP	OVER
(24)	BRANCH:	NOP	U	JMP	FETCH
(25)		NOP	I	CALL	INDRCT
(26)	OVER:	NOP	U	JMP	FETCH
(27)		ARTPC			
	.....	.....	.....	.....	.....
	.....	.....	.....	.....	.....
		ORG 56			
(56)	FETCH:	PCTAR	U	JMP	NEXT
(57)		READ, INCPC	U	JMP	NEXT
(58)		DRTAR	U	MAP	
(59)	INDRCT:	READ	U	JMP	NEXT
(60)		DRTAR	U	RET	

Important symbols and their relative explanations used in the above microprogram are as follows,

Symbol		Explanation
I	I	❖ It is referred as indirect address bit whose condition is DR(15).
II	U	❖ It is referred as unconditional branch which is always = 1.
III	S	❖ It refers to sign bit of accumulator whose condition is AC(15).
IV	CALL	❖ There are two possibilities with this instruction (i.e.,) if condition = 1 the contents of AD are transmitted to CAR, at the same time it's (CAR) contents are incremented by '1' and are transmitted to SBR. If the condition = 0, only the contents of CAR is incremented by '1' and are transmitted to CAR.

Symbol		Explanation
V	JMP	❖ The two possibilities of this instruction is that, whenever the condition = 1, the contents of AD are transmitted to CAR and if the conditions = 0 the contents of CAR simply incremented to '1' and are transmitted to CAR.
VI	RET	❖ One execution of this instruction, the only difference observed is that the content of SBR are transmitted to CAR.
VII	ACTDR	❖ The contents of AC are transmitted to DR.
VIII	DRTAC	❖ The contents of DR are transmitted to AC.
IX	ARTDC	❖ The contents of AR are transmitted to PC.
X	PCTAR	❖ The contents of PC are transmitted to AR.
XI	INCPC	❖ The contents of DC are incremented by '1'.
XII	READ	❖ During the execution of this instruction the following operation is performed (i.e.,) DR ← M[AR].
XIII	WRITE	❖ During execution of this instruction the following operation is performed (i.e.,) M[AR] ← DR.

The above mentioned microprogram executes in the following way.

Execution of above program begins by initially considering the FETCH routine. For simplification, assume that, the MAR instruction (located at serial number (58) initiates branch which refers to address 11. It can be observed from above program that, the address 11 corresponds to a ADD routine. The first instruction refers to a CALL to INDRCT which is instruction (59). The details of CALL instruction can be analyzed by considering symbol IV. The first instruction of INDRCT is READ which depends on condition 'U'. The details of this condition can be accomplished by considering symbol XII. Later, after executing DRTAR, the control gets returned to address (11). Now, instruction with address (12) gets executed, where READ operation is performed. After performing ADD at instruction (13) a JUMP to FETCH instruction is made. Hence, in this way execution proceeds.

Q21. Discuss in detail about the binary microprogram.

Answer :

Binary microprogram is nothing but the binary representation of a symbolic program. As all the instructions in this program are interpreted in the form of 0's and 1's, hence it can be directly stored in the memory without further translation. Any symbolic microprogram is quite easy writing, since most of the instructions are just observated form of normal English words. But when binary microprogram is considered, it is too tedious to represent since the binary equivalent values for each symbol has to remembered and it has to be placed at proper locations so as to attain the desired results. Here there is a possibility of occurrence of two things (i.e.,) either any compiler should be considered to translate a given symbolic microprogram into it's binary equivalent microprogram or it should be directly interpreted by suitable microprocessor programmer. Following here is an example symbolic microprogram which is followed by an equivalent binary microprogram.

Symbolic Microprogram				
Label	**Microoperations**	**Condition**	**Branch**	**Address**
	ORG11			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 15			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 19			

EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 24			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	⋮	⋮	⋮	⋮
	ORG 56			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

Now, the binary equivalent program is as follows,

Label	Binary Addresses (11-27) and (56-60)	Field 1	Field 2	Field 3	Condition (CD)	Branch (BR)	Address (AD)
ADD:	1011(11)	000	000	000	01	01	111011
	1100(12)	000	100	000	00	00	1101
	1101(13)	001	000	000	00	00	111000
STORE:	1111(15)	000	000	000	01	01	111011
	10000(16)	000	101	000	00	00	10001
	10001(17)	111	000	000	00	00	111000
EXCHANGE:	10011(19)	000	000	000	01	01	111011
	10100(20)	000	100	000	00	00	10101
	10101(21)	100	101	000	00	00	10110
	10110(22)	111	000	000	00	00	111000
BRANCH:	11000(24)	000	000	000	10	00	11010
	11001(25)	000	000	000	00	00	111000
OVER:	11010(26)	000	000	000	01	01	111011
	11011(27)	000	000	110	00	00	111000
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
FETCH:	111000(56)	110	000	000	00	00	111001
	111001(57)	000	100	101	00	00	111010
	111010(58)	101	000	000	00	11	000000
INDRCT:	111011(59)	000	100	000	00	00	111100
	111100(60)	101	000	000	00	10	000000

Q22. Write an assembly language program for basic computer to find the largest of 10 numbers stored in the memory.

Answer :

```
00   LDA   105 Num     ;
01   STA   04          ;
02   LDA   91          ;  It loads "LDA 91" where Num = 91 into address 04 in the memory
03   STA   104         ;  The largest number found as yet is stored at location 104
04   0                 ;  The next element of the array is loaded
05   SUB   104         ;  The largest element found till now is subtracted
06   SKP               ;  It is a new largest number which is found
07   JMP     10
08   ADD     104       ;  New largest number (new – old + old)
09   STA     104       ;  Stored at location 104
10   LOAD    04        ;  The instruction at address 04 is retrieved
11   ADD   #   1       ;  Increment address in instruction by 1.
12   STA     04        ;  The instruction is changed now
13   LDA     103
14   SUB   #   1       ;  Decrement the counter
15   STA     103
16   SKN               ;  Checks whether the end of the array is reached
17   JMP     04        ;  If not, continue until the end of the array is reached
18   LDA     04        ;  Processing is completed
19   OUT               ;  Answer is displayed
20   HLT
103  DAT     14 M – 1  ;  Identification of the final array location (i.e., 15 locations)
104  DAT               ;  It contains the largest number found at this point
105  LDA   91          ;  Data is processed and instruction is executed.
```

2.1.4 Design of Control Unit

Q23. Discuss the decoding of microoperation fields.

Answer :

Model Paper-I, Q4(b)

Decoding of Microoperation Fields

In the microinstruction of the control memory, the microoperation field consists of three fields of three bits each. These bits are encoded to provide eight distinct microoperations. The output of these subfields must be decoded to execute the microoperations specified by them. The figure below illustrates the process of decoding microoperation fields of a microinstruction.

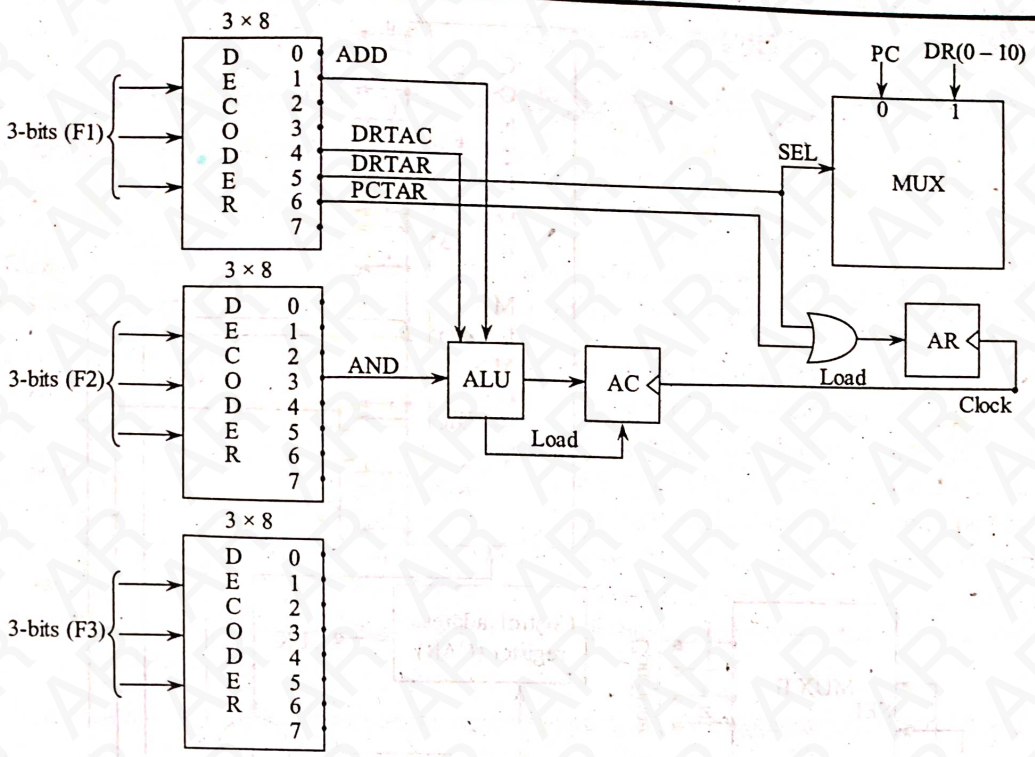


Figure: Decoding of Microoperation Fields

It can be observed from the above figure that, the initial fields of the microinstruction (i.e., F1, F2 and F3) are connected to three different decoders. The configuration 3 × 8 shows that, there are three inputs from each of these fields which are decoded into 8 set of outputs, numbered from 0-7. The inputs which are provided to these decoders are nothing but the outputs from the control memory. The outputs of these decoders should be connected accordingly, in order to attain the right circuit configuration. In order to understand the connectivity of the circuit consider few examples.

Here, certain microoperations supported by F1 field of the microinstruction (these are essential in order to justify examples) are given below.

F1 Code	Microoperation
001	AC ← AC + DR
100	AC ← DR
101	AR ← DR(0 -10)
110	AR ← DC

Now, consider the circuit diagram. Whenever, the pin 1 of F1 decoder is activated, its corresponding binary value 001 initiates the microoperation AC ← AC + DR. That is, this microoperation performs an ADD operation by referring to arithmetic and logic unit of the circuit. If fourth pin of F1 decoder is activated, its corresponding binary value i.e., 100 initiates the microoperation AC ← DR. The symbol DRTAC reflects the microoperation is performed i.e., it transmits the instruction referred by the address stored in Data Register (DR) into the Accumulator (AC). Similar are the cases when pin 5 and 6 are activated.

It can be noted from the circuit diagram that, the signals corresponding to the pins (5, 6) of F1 are ORed and their output is provided to the address register. This is because pin 5, 6 supports microoperations such as AR ← DR (0-10) (i.e., DRTAR) and AR ← PC (i.e., PCTAR). As their destinations are same hence, activating any one of these pins cause the transfer of address from data register or program counter into the address register.

Q24. With the help of a diagram, explain the organization of microprogram sequencer for a control memory.

Answer : **Model Paper-III, Q5(b)**

Following is the organization representing microprogram sequencer internal to a given control unit.

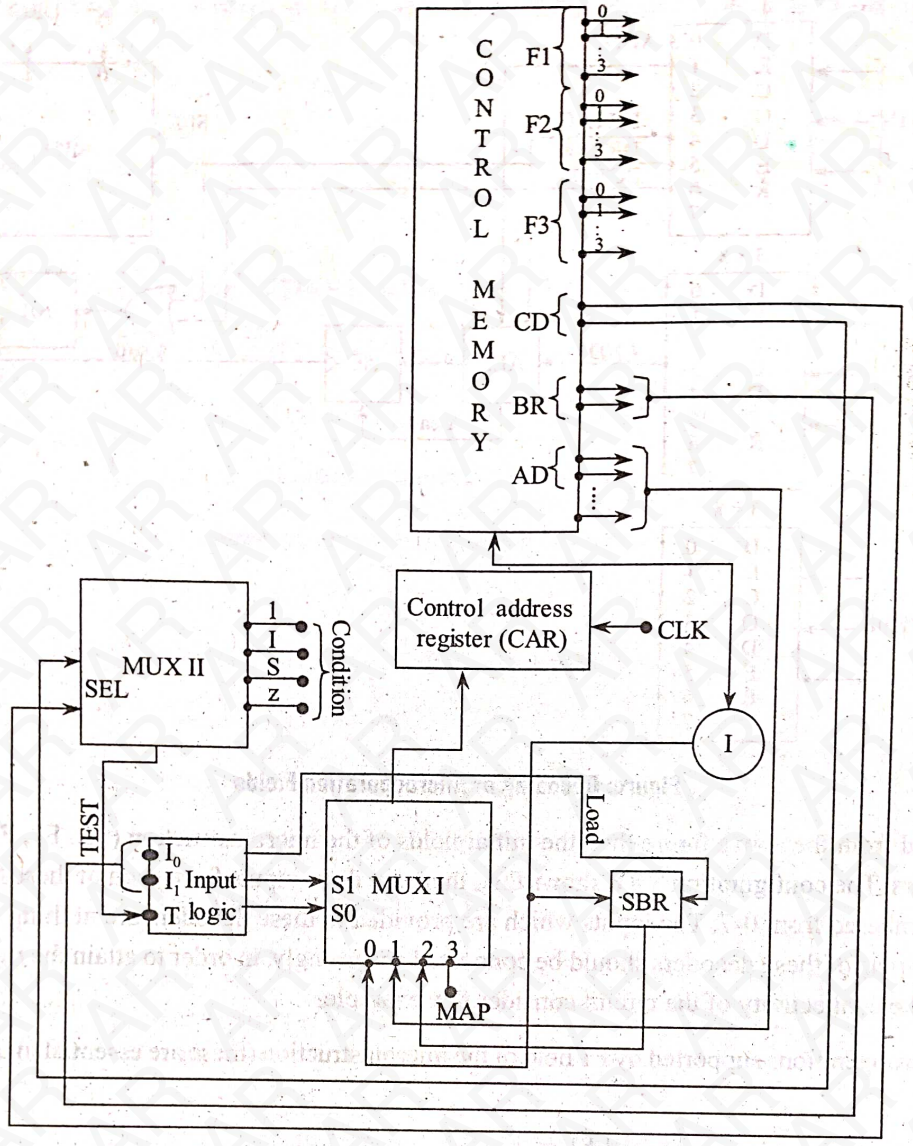


Figure: Organization of a Sequencer or Next Address Generator

In general terms sequencer refers to a block of circuit present in a control unit whose major task is to fetch the address of the next instruction to be executed. This is an important concept in multiprogramming and also it is crucial while considering pipelined execution. It does this by considering the next-address information bits which forms a part of instruction which is currently being executed. The organization of the sequencer is shown in the figure above. It consists of a control memory, control address register, incrementer, subroutine register, two multiplexers and finally an input logic respectively. The control memory maintains four blocks with first block maintaining 24 signals for 24 different microoperations. Similarly the condition and branch fields maintains a total of 8 signals with four signals corresponding to single field.

The Multiplexer-I, takes four inputs and accordingly selects a single address which it passes to control address register. From there the given address gets released to the control memory. It can be observed from above diagram that, the output of CAR also gets connected to the incrementer. This is because, after delivering the address to the control memory, the value of CAR register gets incremented by '1'. Later the output of the incrementer forms one of the inputs of the multiplexer-I as well as of the subroutine register. The Multiplexer-II checks one of the four conditions ((i.e.,) I, U, S, Z) and transmit it to the input logic. The input logic again transmits its signals to multiplexer-I. In addition to above mentioned circuitry, the microprogram sequencer also maintains a stack register with a storage capacity approximately about 8 levels, hence in addition to condition selection microoperation management etc, there are also push and pop operations implemented on this stack.

The major operations which are performed by the sequencer on next address generator are listed below,

- Address sequencing operations
- Push and pop operations
- Transmitting program control to the subroutine and returning operations.
- Increment operations
- Jump operations at specific location etc.

2.2 CENTRAL PROCESSING UNIT

2.2.1 General Register Organization

Q25. Describe briefly the general register organization.

Answer :

Model Paper-II, Q5(a)

When the CPU contains multiple registers, they can be connected to each other using a common bus system. This common bus system will allow the registers to transfer data directly among themselves and also to perform different microoperations.

A common bus organization for seven registers present in the CPU is shown in the figure. In this unit, two MUXs of size 8×1 are used. The inputs to each of the MUXs are the output of seven registers and a data input. In addition to these inputs, the MUXs also have three selection inputs collectively termed as SI_A and SI_B for MUX_A and MUX_B respectively. The output of the MUX_A is bus A and the output of the MUX_B is bus B. These buses are the inputs to the arithmetic and logic unit. The operation to be performed in arithmetic logic unit is selected using the 'selop' lines. The result of the microoperation performed is the output of the ALU and is also the input to all the seven registers. A 3×8 decoder with three selection inputs which are collectively termed as SI_D is also used which allows the data from the output bus to be transferred as input to a particular destination register. To do so, this decoder selects a particular destination register in which the data from the output bus is to be transferred and then activates the load input of that register.

The information which is transferred between the registers and ALU is managed by the control unit that handles the CPU bus system.

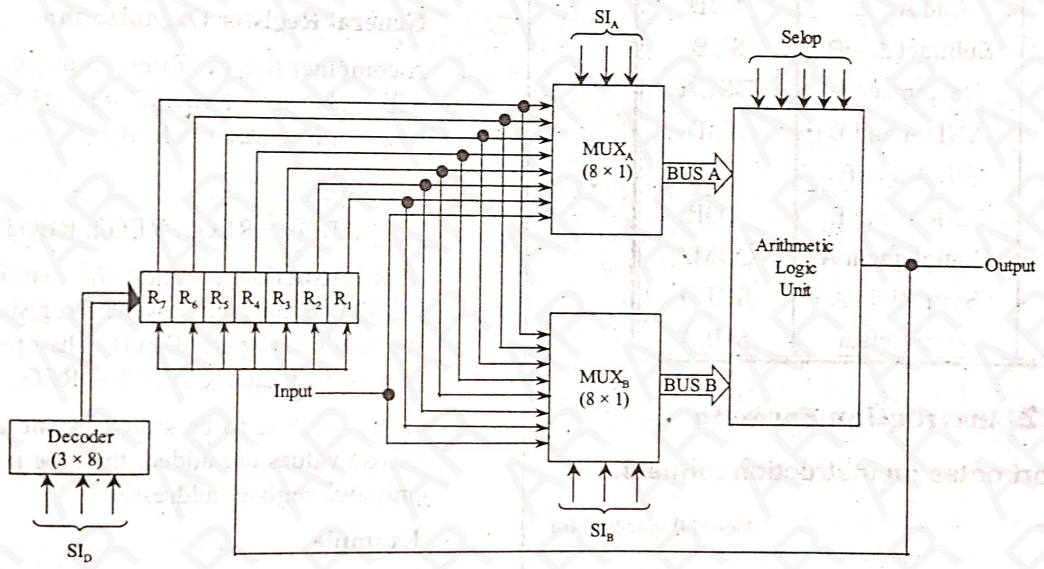


Figure: A Common Bus Organization of Seven Registers

Control Word

The selection inputs of MUX_A, MUX_B, decoder and ALU, collectively form a control word and can be represented as,

SI_A	SI_B	SI_D	Selop
3	3	3	5

Figure: A 14-bit Control Word

A control word contains four fields and 14 bits. The SI_A, SI_B, SI_D and selop fields in control word represent the selection inputs of MUX_A, MUX_B, decoder and ALU, respectively.

- The 'SI_A' field contains three bits which are used to select a source register when the input to ALU is 'A'.
- The 'SI_B' field contains three bits which are used to select a register when the input to ALU is 'B'.
- The 'SI_D' field contains three bits which are used to select a destination register with the help of a decoder that produces seven load outputs.
- The 'selop' field contains five bits which are used to select an operation that is to be performed in ALU.

To indicate a particular microoperation, all the 14 bits of the control word are applied to different selection table.

Binary Code	SI_D	SI_B	SI_A
000	None	Input	Input
001	R_1	R_1	R_1
010	R_2	R_2	R_2
011	R_3	R_3	R_3
100	R_4	R_4	R_4
101	R_5	R_5	R_5
110	R_6	R_6	R_6
111	R_7	R_7	R_7

The five bits of selop field that are used for selecting a particular ALU operation can be encoded with the following binary codes as illustrated in the following table.

Selop	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A – B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift Right A	SHRA
11000	Shift Left A	SHLA

2.2.2 Instruction Formats

Q26. Write short notes on instruction formats.

Answer : Model Paper-II, Q4(a)

An instruction can be represented in a specific format which shows the bits appearing in a word of the memory or in the control register. The bits present in the instruction are classified into several groups to form fields.

There are three basic fields that are present in an instruction.

1. **Opcode Field**

   It is an operation code field that indicates a specific operation to be performed.

2. **Address Field**

   It is a field where a particular memory address or a processor register is specified.

3. **Mode Field**

   It is a field which provides a way to determine the operand or its effective address.

The length of an instruction differs in different computers based on the number of addresses used within an instruction. The register's internal organization will decide the number of address fields that are to be used in an instruction format. The three different types of CPU organization are as follows,

1. Single accumulator organization
2. General register organization
3. Stack organization.

1. **Single Accumulator Organization**

A computer that conforms to this type of organization will perform all its operations with the help of an implied accumulator register. Therefore, only one address field is used within the instruction format.

Example

ADD B

This instruction will add the value present at address B with the value present in the accumulator and stores the result back into the accumulator.

AC ← AC + M[B]

2. **General Register Organization**

A computer that conforms to this type of organization will make use of two or three register address fields within the instruction format.

Example

ADD REG1, REG2, REG3

This instruction will add the data present in register REG2 with the data present in register REG3 and stores the result in register REG1. The operation carried out, thus corresponds to REG1← REG2 + REG3.

If the result is to be stored in one of the two register whose values are added, then the instruction can have only two register addresses.

Example

ADD REG1, REG2

This instruction will add the data present in REG1 with the data present in register REG2 and stores the result in register REG1. The operation carried out, thus corresponds to REG1← REG1 + REG2.

The address fields in the instruction format of the general register-type computers can describe a processor register or a memory word.

Example

ADD REG2, A

This instruction has two address fields among which one describes the address for register REG2 and another specifies the memory address of the operand A. This instruction adds the contents of A to the register REG2 and stores the result in REG2. The operation carried out, thus corresponds to REG2 ← REG2 + M[A].

3. Stack Organization

A computer that conforms to this type of organization will use the PUSH and POP instructions which use only one address field.

Example

PUSH C

This instruction will push the word present at address C to the top of the stack.

The operation-type instruction used within a stack organized computer, will not require an address field and can be termed as zero-address field instruction.

Example

ADD

This instruction will add the two topmost numbers on the stack by popping them out, computing their sum and pushing the result on top of the stack.

Q27. With an example, explain the concept of zero-address, one-address, two-address and three-address instructions.

Answer :

1. Zero Address Instructions

In this instruction type, the major instructions used are PUSH, POP. These instructions correspond to stack related operations, hence, they are implemented in stack-enabled systems. Here, the commands such as SUB, ADD and DIV do not require any addresses (operands), moreover the PUSH and POP instructions do maintain single address variable, so as to ascertain the indulgence of a given variable with respect to the operating stack.

Example

The a program to execute $Y = (A - B)/(C + D*E)$ using zero-address instructions is given below.

S.No.	Program Instructions
1	Push A
2	Push B
3	SUB
4	Push D
5	Push E
6	MUL
7	Push C
8	ADD
9	DIV
10	Pop Y

2. One Address Instruction

One-address instructions use only one variable and uses an implied accumulator (AC) register for all data manipulation. The result of all operations is stored in the AC register.

Example

The program to execute $Y = (A - B)/(C + D*E)$ using one-address instructions is given below,

S.No.	Program Instructions
1.	LOAD D
2.	MUL E
3.	ADD C
4.	STORE TEMP
5.	LOAD A
6.	SUB B
7.	DIV TEMP
8.	STORE Y

An explanation for the above program is as follows,

- Line 1 the value of D in an accumulator.
- In line 2 value of E is multiplied with the value stored in accumulator and the result is stored back in the accumulator.
- Line 3 adds the value of C to the value stored in the accumulator.
- In line 4, the value of accumulator is stored into a temporary variable TEMP.
- Further, in lines 5, 6 the value of 'A' is added into the accumulator, from which the value of 'B' is subtracted. The result of this operation is again stored in the accumulator.
- Finally, in lines 7, 8 the value in accumulator is divided by the value of TEMP and the result is stored in accumulator which is later stored in 'Y'.

3. Two Address Instructions

Two-address instructions include two variables in each instruction. Like three-address instructions, each variable can specify a processor register or a memory address. Hence, the number of steps in two address are less with respect to one address instructions, but more when compared to three-address instructions.

Example

The program to execute $Y = (A - B)/(C + D * E)$ using two address instructions is given below,

S. No.	Program Instructions
1.	MOV Y, A
2.	SUB Y, B
3.	MOV TEMP, D
4.	MUL TEMP, E
5.	ADD TEMP, C
6.	DIV Y, TEMP

An explanation for the above program is as follows,

- In line 1, the value of *A* is stored *Y*.
- In line 2, the value of '*B*' is subtracted from the value of '*Y*'.
- Now, the value of '*D*' is stored in a temporary variable (TEMP).
- In line 4, the value of '*E*' is multiplied to the value of TEMP. The resultant this step is again stored into TEMP.
- In line 5, the value of '*C*' is added to TEMP Variable.
- Finally, the program is concluded by dividing the value of '*Y*' by TEMP and storing the result back into '*Y*'.

4. Three Address Instructions

Three address instructions include three variables in each instruction. Each variable can specify a processor register or a memory address. When these instructions are compared with one-address and two address instructions, they form the least steps to execute a given program.

Example

The program to demonstrate execution of $Y = (A - B)/(C + D*E)$ using three address instructions is given below,

S. No.	Program Instructions
1.	SUB Y, A, B.
2.	MUL TEMP, D, E.
3.	ADD TEMP, TEMP, C.
4.	DIV Y, Y, TEMP.

An explanation for the above program is as follows,

- In line 1, value of '*B*' is subtracted from the value of '*A*' and the result is stored in variable '*Y*'.
- Line 2, introduces a temporary variable 'TEMP', where the value of '*E*' is multiplied to the value of '*D*' and the result is directed to the variable 'TEMP'.
- In line 3, the value of '*C*' is added to TEMP and the result is stored back in TEMP.
- Finally, the program is concluded by dividing the value of *Y* by TEMP and the result is stored back in 'Y'.

Q28. Give the comparison between zero-address, one-address, two-address and three-address.

Answer :

The following table shows the comparison between zero-address, one-address, two-address and three-address.

Zero-address		One-address		Two-address		Three-address	
1.	It does not contain any operand address i.e., no explicit operands are present in the instruction.	1.	It contains only one operand address in the instruction i.e., only one explicit operand is specified to one instruction.	1.	It contains two operand addresses in the instruction i.e., two explicit operands are specified to each instruction.	1.	It contains two addresses for two operands i.e., three explicit operands are specified to each instruction.
2.	Its general instruction format is, Operation Source1, Source2, Destination.	2.	Its general instruction format is, Operation Source, Destination.	2.	Its general instruction format is, Operation Source.	2.	–
3.	It stores its operands on stack in memory.	3.	It stores its operands in the accumulator.	3.	It stores the operands by overwriting the contents in the memory location.	3.	It stores its variable names on distinct previously stored locations in memory.
4.	It is also called as stack organization.	4.	It is also called as single single accumulator organization.	4.	–	4.	It is also called as general register organization.

5.	**Example** Instruction: Push P Push 8 Add Pop P. Computes: Increment P by 8.	5.	**Example** Instruction: ADD P Computes: AC ← AC+M[P]	5.	**Example** Instruction: ADD 5, X. Computes: X ← X + 5.	5.	**Example** Instruction: ADD A, B, C. Computes: C ← A + B.

2.2.3 Addressing Modes

Q29. Explain in detail about different addressing modes with example.

Answer :

A computer program can ideally be described as a sequence of instructions. These instructions are binary codes which holds series of microoperations for the computer. Every individual instruction describes operations to be performed on data. These operations are present in opcode field and the operands stored in the operand field specifies the data.

An addressing mode is a method of specifying an operand. The instruction format must contain various types of addressing capabilities so as to improve the capabilities and computing power of instruction. This can be included by introducing implicit addressing schemes in the operand field. The following are the types of addressing modes.

1. Implied Addressing Mode

When the instruction's definition contain the operands that are implicitly provided on which an operation specified by the opcode is to be performed, then this addressing mode is said to be an implied addressing mode. As the operands are directly specified instead of their address.

Example of an instruction that employs implied addressing mode is "Increment Accumulator".

The operand is the data present in the accumulator on which the increment operation is performed.

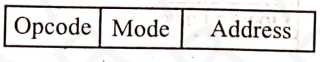


Figure: Instruction Format with Mode Field

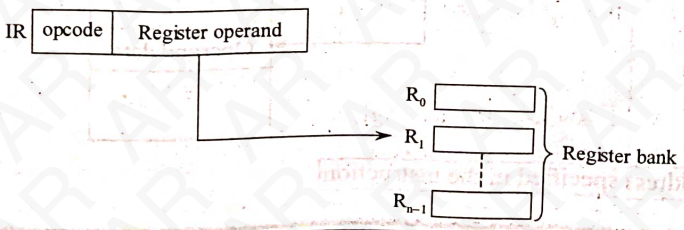
2. Immediate Operand Addressing Mode

In this mode, operand is part of the instruction but not as a contents of a register or memory location. Instruction format for immediate addressing mode is shown in the following figure.

As shown in the above figure immediate addressing mode has an operand field rather than address field. Since the data are encoded directly into the instruction, immediate operands normally represent constant data.

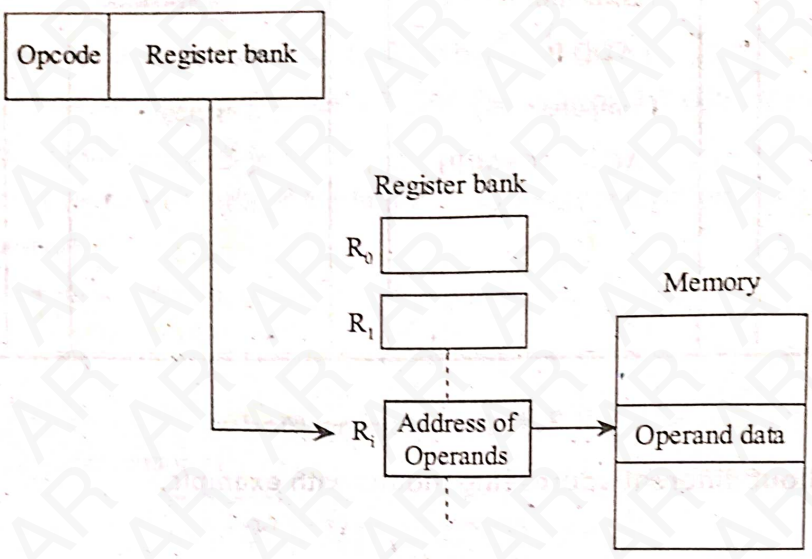
3. Register Operand Addressing Mode

With the register addressing mode, the operand to be accessed is present in CPU registers. Mostly, all computer systems are embedded with some addressable registers. The k bits of the operand field can specifies any one of 2^k registers operand holding the data word.



4. Register Indirect Addressing Mode

In this mode the register operand R_i in the instruction specifies the memory address where the operand data is located.



It needs one memory access and one register access to get the operand data.

5. Autoincrement/Autodecrement Mode

In this addressing mode, the operand address is incrimented or decremented automatically for every access mode to the operand data. It is implemented in the form of memory direct (figure (a)) or as register/memory indirect (figure (b)). In both the scenario the operand address which is pre-incremented or decremented is specified that is the address is specified on location where the access to the operand address is made by increasing and decreasing the operand address value. Moreover, using these schemes the operand address is registered with in the operand field just by post incrementing and decrementing the values.

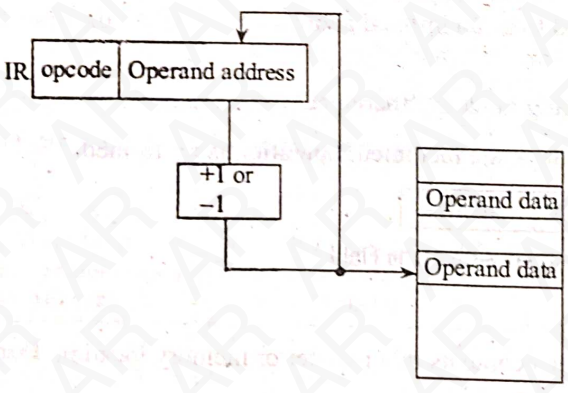


Figure (a): Memory Direct

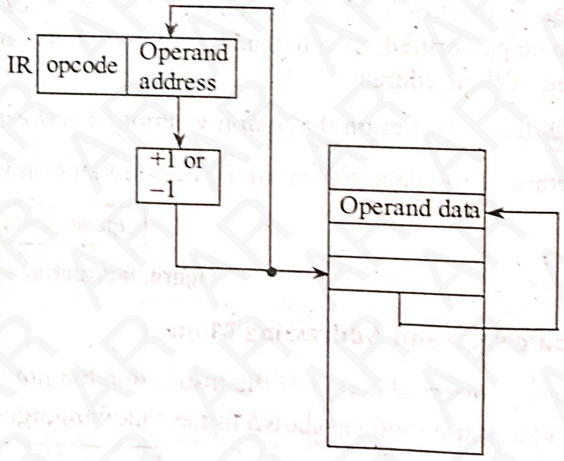
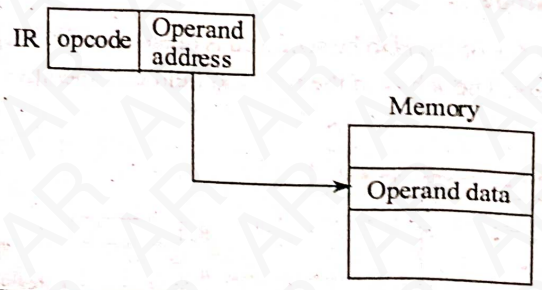


Figure (b): Memory Indirect

6. Direct Memory Addressing Mode

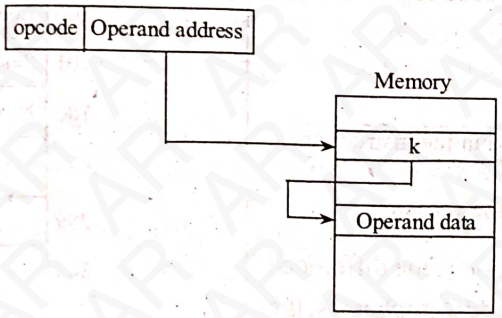
The operand in this addressing mode resides in memory and its respective address is directly specified by the address field of the instruction.



∴ Effective address = Address specified in the instruction

7. Indirect Memory Addressing Mode

In this scheme the operand address specifies another memory location i.e., the effective address where the desired data element is stored.



$\therefore$ Effective address = M[address specified in the instruction] + Content CPU register

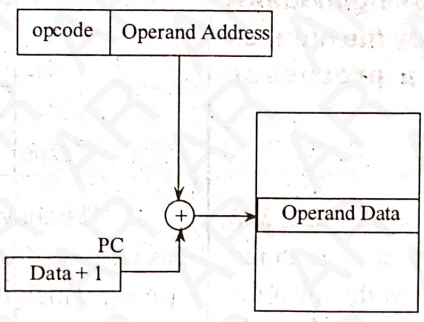
While this scheme has the advantage of enhancing addressing capability it needs to access memory twice to get the operand data.

8. Relative Addressing Mode

In this addressing mode, the instruction contains a relative address which is mostly a signed number (in 2's complement form) and it can be sometimes positive or negative. This address is added to the contents of the program counter to obtain the effective address.

$\therefore$ Effective address = ([PC] + 1) + Address specified in the instruction

Where, [PC] indicates data within the program counter.



This type of addressing mode is commonly used while dealing with the branch-type instructions where the branch address is located in the area where the instruction word is also stored.

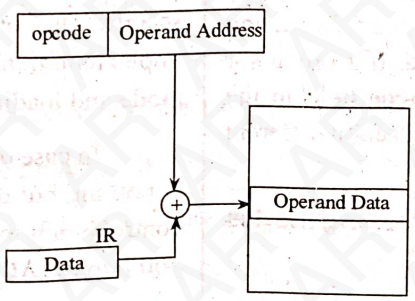
The advantage of using this type of address mode is that the address specified in the instruction has a shorter address field as complete memory address is not specified.

9. Indexed Addressing Mode

In this addressing mode. The address is added to the data present in the index register inorder to obtain effective address.

Effective address = [IR] + Address specified in the instruction

Where, [IR] = Data present in instruction register



The index value stored in the index register can be specified as a distance covering from starting address till the address of the operand. If the address field is not specified within an index-type instruction then it starts operating in a register indirect mode.

10. Base Register Addressing Mode

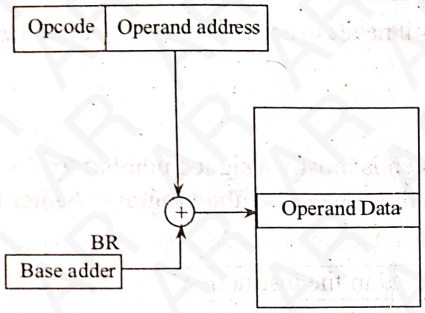
In this addressing mode, the instruction containing an address is added to the data present in the base register to get the effective address.

Thus,

Effective address = [BR] + Address specified in the instruction

Where, [BR] = Data present in base register.

The address specified in the instruction is the difference between the base address and the effective address whereas, the base register holds the base address.



Q30. Why do we need so many addressing modes? Is the instruction size influenced by the number of addressing modes which a processor supports?

Answer :

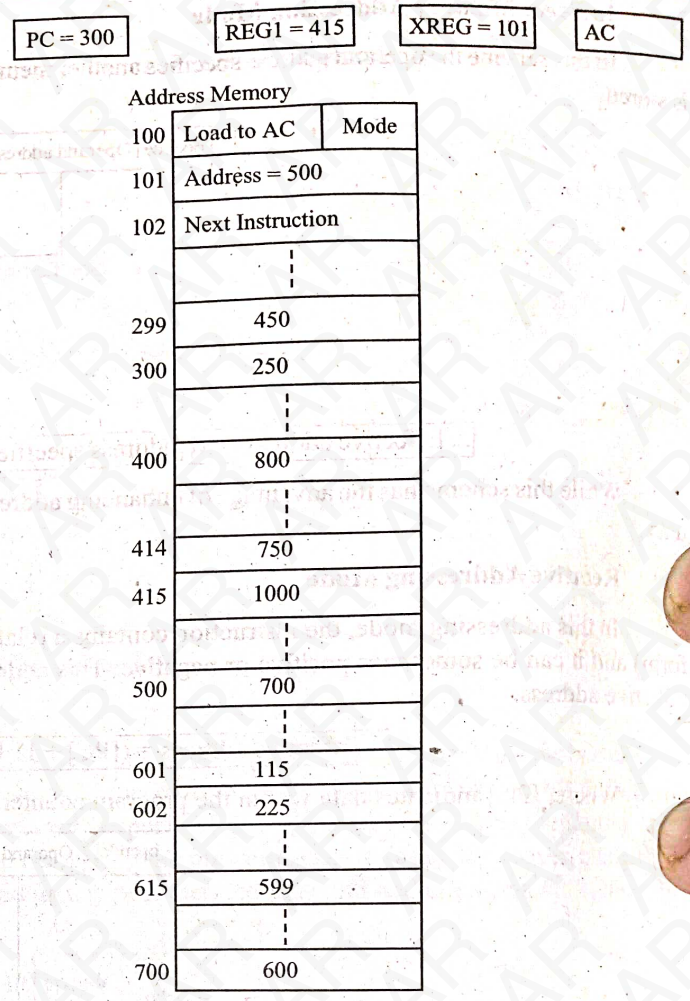
An instruction contains 3 fields i.e., mode, opcode and address performed, this must be executed on some data stored either in register (or) in memory. The method with which the operands are selected at the time of program execution entirely depends on addressing mode of the instruction. Thus, the addressing mode specifies the operand is accessed either from memory (or) from register. If there are many addressing modes in the computer, then they are distinguished using mode field.

The size of an instruction will not be changed, no matter how many addressing modes are supported by a processor. But the size of the mode field will be influenced by the number of addressing modes that are supported by a processor. As the mode field present within an instruction set specifies any one of the different addressing modes, multiple bits of an instruction format utilized as a mode field. For example, if a processor supports eight addressing modes then the mode field in the instruction set will be of 3-bit size to distinguish eight different addressing modes.

Q31. Illustrate the use of different addressing modes with numerical example.

Answer : **Model Paper-II, Q4(b)**

The use of different addressing modes can be illustrated with the following example.



The above figure containing the distinct memory location has two-word instruction at memory addresses 100 and 101. Among which the first part at memory address 100 is load to Accumulator (AC) instruction specifying the operation code and mode. And the second part at memory location 101 specifies address equal to 500. On the other hand, the Program Counter (PC) stores the value 300 inorder to fetch the instruction stored at memory address 100. And the rest of the elements such as processor register REG1 holds the value 415 and the index register XREG holds the value 101. The accumulator is kept inorder to store the value after the execution of the instruction.

The mode field present at memory address 100 specifies any one of the ten address modes. It optimizes the implementation by calculating the effective address in every mode and loading the operands in AC.

In case of direct addressing mode, the effective address is nothing but the address part of the instruction 500. So, the control jumps to the memory address 500 and loads the operand 700 into the AC.

In immediate addressing mode, the effective address is 101 itself, however it is not loaded, instead the operand is loaded into the AC which is 500.

In indirect addressing mode, the effective address is given at memory address 500 i.e., the effective address becomes 700. Now the control jumps to the address location 700 and loads the value 600 into the AC.

In the relative addressing mode, the effective address is optimized by summing 500 and address location of the next instruction i.e., 500 + 102 = 602. Now, the control jumps to the memory address 602 and stores the value 225 in the AC. After every fetch and execution phase, the value of the program counter PC increments which now becomes 202.

In the index mode, the calculation of effective address is done by summing the value at index register XREG and 500 (i.e., memory address is 101). Now 101 + 500 = 601 and the operand loaded into AC is 115.

In the register addressing mode, the effective address is not calculated therefore, the value stored at REG1 i.e., 415 is directly loaded into AC.

In the register indirect addressing mode, the effective address is more or less same as the value stored at REG1 i.e., 415. Now the control goes directly to the address location 415 and loads the value 1000 into the AC.

In autoincrement addressing mode, the working is analogous to the register indirect mode but with a small difference. That is in former one after the completion of the instruction REG1 is incremented to 415 where as in latter one no increment is performed.

In autodecrement addressing mode, before completion of execution of instruction the value of REG1 is decremented to 414. The control jumps to 414 address location and loads the operand value as 750 into the AC.

Q32. Write program to evaluate the arithmetic statement $X = \frac{A-B+C*(D*E-F)}{G+H*K}$ using a general register computer with three address instructions.

Answer :

The program to demonstrate execution of $X = \frac{A-B+C*(D*E-F)}{G+H*K}$ using three address instruction is given below,

S.No.	Program	Instruction
1.	SUB	R_1, A, B
2.	MUL	R_2, D, E
3.	SUB	R_3, R_2, F
4.	MUL	R_4, C, R_3
5.	ADD	R_5, R_2, R_4
6.	MUL	R_6, H, K
7.	ADD	R_7, G, R_6
8.	DIV	R_8, R_5, R_7

As seen from the above program, three address instructions includes three variables in each instruction. Each variable can specify a processor register or a memory address. An explanation of the above program is as follows,

- In line 1, value of 'B' is subtracted from the value of 'A' and the result is stored in the variable 'R_1'.
- In line 2, the value of 'R_1' is multiplied with the value of 'E' and the result is stored in the variable 'R_2'.
- In line 3, the value of 'F' is subtracted with the value of 'R_2' and the result is stored in 'R_3'.
- In line 4, the value of 'C' is multiplied with the value of 'R_3' and the result is stored in 'R_4'.
- In line 5, the values of both 'R_2' and 'R_4' are added and the value is stored in 'R_5'.
- In line 6, the value of 'H' is multiplied with the value of 'K' and the result is stored in 'R_6'.
- In line 7, the value of 'G' is added to the value of 'K' and the result is stored in 'R_7'.
- Finally, in line 8, the value of 'R_5' is divided by 'R_7' and the result is stored in 'R_8'.

2.2.4 Data Transfer and Manipulation

Q33. Tabulate various data transfer and manipulation instructions.

Answer :

Computer instructions are extremely useful entities of a computer since they facilitate users to carry on their proceedings smoothly while working with different tasks.

Following is the list of instructions that come under the category of data transfer instructions,

SL.No.	Instruction Name	Symbol used	Description
1.	Push	PUSH	It is used to direct a word of data on to the top of a stack.
2.	Pop	POP	It causes a data word to be removed from the top of a stack.
3.	Input	IN	It causes the data to be loaded from a given source (I/O port etc.) into the memory or registers.
4.	Output	OUT	It causes data to be transmitted from given memory or registers to the destination (I/O port, I/O devices etc.).
5.	Load	LD	It causes transfer of data (interms of words) from given memory location to the processor for further computations.
6.	Store	ST	It is reverse of load operation i.e. data (in terms of words) gets stored in memory.
7.	Move	MOV	It causes transfer of data from source to destination.
8.	Exchange	XCH	It causes swapping operation i.e the contents of source is copied into the destination and vice versa.

Table: Data Transfer Instructions

Data Manipulation Instructions

These instructions facilitate various types operations (i.e., arithmetic, shift) to be performed on a given set of data. Based on different types of operations, data manipulation instructions are broadly classified into three categories. They are as follows,

1. Arithmetic instructions
2. Logical and bit manipulation instruction
3. Shift instruction.

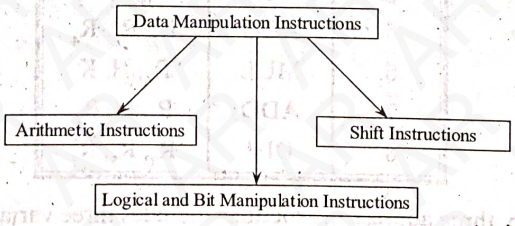


Figure: Data Manipulation Instructions

1. Arithmetic Instructions

As the name suggests arithmetic instructions cause arithmetic operations (add, subtract, divide etc.) to be performed on a given set of data. Different types of arithmetic instructions, supported by a general computer are tabulated as following.

Sl.No.	Instruction Name	Symbol Used	Description
1.	Addition	ADD	It adds the values of two operands.
2.	Subtraction	SUB	It subtracts the values of two operands.
3.	Division	DIV	It divides the two operands by considering one as dividend and other as divisor and the result of this operation is the quotient.
4.	Multiplication	MUL	It multiplies the two operands.
5.	Increment	INC	It increments the value of given operand by 1.

6.	Decrement	DEC	It decrements the value of given operand by '1'.
7.	Add with carry	ADDC	It performs the addition operations between two operands and considers even its carry bit.
8.	Subtract with	SUBB	It performs the subtraction operation between two borrow operands and considers even
9.	Negation	NEG	It simply changes the sign of a given operand value.

Table: Arithmetic Instructions

2. Logical and Bit Manipulation Instructions

These are the type of instructions which performs logical operations such as AND, OR, NOT etc., on the binary data, stored in their respective registers.

SL.No.	Instruction Name	Symbol Used	Description
1.	Logical AND	AND	It performs logical AND operations between the values of two registers.
2.	Logical OR	OR	It performs logical OR operation between the values of two registers.
3.	Logical	XOR	It performs logical exclusive OR operation between the values of two registers.
4.	Clear carry	CLRC	It clears or vacant the register maintaining the carry bit.
5.	Set carry	SETC	It enables the carry register.
6.	Clear	CLR	It empties or vacant the given register.
7.	Complement	COM	It inverts the bits of a given register.
8.	Enable Interrupt	EI	It enables an interrupt
9.	Disable Interrupt	DI	It disables an interrupt
10.	Complement carry	COMC	It complements to inverts the carry bit.

Table: Logical and Bit Manipulation Instructions

3. Shift Instruction

These are the instructions which causes the contents of the registers to be shifted. Following are the various types of shift instructions.

Sl.No.	Instruction Name	Symbol Used	Description
1.	Logical shift left	SHL	It causes the contents of the given register Logical to be shifted 1 bit position towards its left excluding the sign bit.
2.	Logical shift right	SHR	It causes the contents of the given register Shift right to be shifted 1 bit position towards it's right excluding the sign bit.
3.	Arithmetic	SHRA	This instruction causes the contents of the register to be shifted 1 bit position towards its right, including the sign bit. But the sign bit remains unchanged.
4.	Arithmetic shift left	SHLA	It causes the contents of given register to be shifted 1 bit towards its left, including the sign bit. But the sign bit remains unchanged.
5.	Rotate right	ROR	It causes the contents of a given register to be shifted 1 bit position towards it's right in a circular manner i.e., the last bit of the register forming the first bit and rest of the bits shifting towards their right by 1 bit position.
6.	Rotate left	ROL	It causes the contents of a given register to be shifted 1 bit position towards
7.	Rotate right	RORC	Same as ROR, except the fact that, it even carry includes the sign bit.
8.	Rotate left	ROLC	Same as ROL, except the fact that it even carry includes the sign bit.

Table: Shift Instructions

2.2.5 Program Control

Q34. Discuss in brief about program control instructions.

Answer :

Model Paper-II, Q5(b)

Program Control Instructions

These type of instructions causes the program control to be switched to different locations depending on the requirement.

Different types of program control instructions and their description is tabulated below,

S.No.	Instruction Name	Symbol Used	Description
1.	Jump	JMP	It causes the program control to be switched to different location by inserting the status of its current execution into the corresponding location.
2.	Branch	BR	It causes the current execution to be terminated temporarily and the processor starts executing the instructions whose address is being specified in the branch.
3.	Call	CALL	This instruction calls a procedure to be executed.
4.	Return	RET	This instruction causes the program control to be returned from the procedure which was activated through CALL instruction.
5.	Compare	CMP	Using this instruction, comparison between two or more operands is made and accordingly the output is generated.
6.	Test	TST	It tests the value of a given operand based on the condition and accordingly sets the flags.
7.	Skip	SKP	It skips the next instruction based on the given condition.

Table: Program Control Instructions

Q35. Draw and explain the block diagram of an 8-bit AW with a 4-bit status register.

Answer :

The arithmetic logic unit inside a CPU can be provided with a status register. The status bits or flag bits or condition-code bits in this register are used for performing conditional instructions like CMP, JMP etc. For an 8-bit ALU, a 4-bit status register is used as shown below,

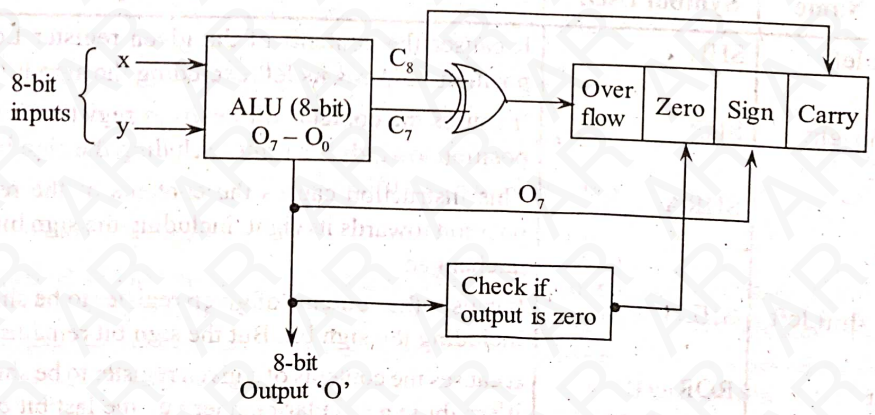


Figure: Block Diagram of an 8-bit ALU with a 4-bit Status Register

The two inputs to an ALU are *X* and *Y* which are 8-bits long. The MSB in these inputs specifies their sign. Some operations may generate a carry, or change the sign, etc. The 4-bit status register stores four flag bits one for sign (s), one for carry (c), one for zero (z) and one for overflow (v) condition. The four bits are set/reset (not all together) depending on the result 'O' obtained after performing an 8-bit operation.

1. Carry Bit (C)

When an operation generates a carry at the end of its completion, the carry is called an end carry, other carries are intermediate carries. When $C_8 = 1$ the C bit in the status register is set to 1 else it is O.

Example

Two 8-bit signed numbers $A = (10010001)_2$ and $B = (11001100)_2$ are added as,

End carry

$$\begin{array}{lll} A = & \overset{1}{\textcircled{}} & 1001\ \ 0001 \\ B = & + & 1100\ \ 1100 \\ \hline & & 0101\ \ 1101 \\ \hline \end{array}$$

$\because C_8 = 1, \quad \therefore \ C = 1$ (i.e., carry flag is set)

2. Sign Bit (S)

The MSB O_7 in the output gives the sign of the result. If $O_7 = 1$ then, $S = 1$, else $S = 0$. In the above addition example the MSB is 0, hence $S = 0$ in this case.

3. Zero Bit (Z)

In the above block diagram, a simple circuitry is used to check for a zero output. By zero, it means that the bits O_0 to O_7 are all zeroes. If the result is zero then $Z = 1$, else $Z = 0$.

Example

Two 8-bit numbers $A = (10010001)_2$ and $B = (1100\ 1100)_2$ are XORed as,

$$\begin{array}{lll} A = & 1001 & 0001 \\ B = & \oplus 1100 & 1100 \\ \hline & 0101 & 1101 \\ \hline \end{array}$$

⊕ – For same bits the output is 0 else of in 1

In this case, $z = 0$ as the output is nonzero. The XOR operation can be used to check whether $A = B$. If 'Z' is set to 1 then it means $A = B$, else $A \neq B$. Moreover, AND operation also effects the value of 'Z' bit.

4. Overflow Bit (V)

When last two carries C_7 and C_8 are set, the overflow bit 'V' is also set to 1. To check whether $C_7 = C_8 = 1$, an XOR gate is used. If its output is zero then $V = 1$ else $V = 0$.

Q36. List the conditional branch instructions.

Answer :

Conditional Branch Instructions

These instructions takes the program control to some other part of the program based on the results of a 'compare' instruction'.

Different types of conditional branch instructions are tabulated below.

S.No.	The branch Condition	Symbol Used	The Test Condition
1.	Branch if zero	BZ	$Z = 1$
2.	Branch if not zero	BNZ	$Z = 0$
3.	Branch if overflow	BV	$V = 1$
4.	Branch if no overflow	BNV	$V = 0$
5.	Branch if carry	BC	$C = 1$
6.	Branch if no carry	BNC	$C = 0$
7.	Branch if minus	BM	$S = 1$
8.	Branch if plus	BP	$S = 0$
	Signed compare conditions between two variables (i.e.,) $(i - j)$		
9.	Branch if greater than	BGT	$i > j$
10.	Branch if less than	BLT	$i < j$
11.	Branch if greater than or equal	BGE	$i \geq j$
12.	Branch if less than or equal	BLE	$i \leq j$
13.	Branch if equal	BE	$i = j$
14.	Branch if not equal	BNE	$i \neq j$
	Unsigned compare conditions between two variables (i.e.,) $(i - j)$		
15.	Branch if higher	BHI	$i > j$
16.	Branch if lower	BLO	$i < j$
17.	Branch if higher or equal	BHE	$i \geq j$
18.	Branch if lower or equal	BLOE	$i \leq j$
19.	Branch if equal	BE	$i = j$
20.	Branch if not equal	BNE	$i \neq j$

Q37. Illustrate the concept behind subroutine call and return.

Answer :

Subroutine is an important part of any computer system architecture. A subroutine is a self-contain group of instructions that usually performs one task. It is a reusable section of the software that is stored in memory once, but used as many times as required. This saves memory space and makes it easier to develop software.

At the time of program execution, a subroutine is called multiple times to perform its function at various points. After the execution of the subroutine, control is returned back to the main program so that the program may resume its normal operations.

There are different names available for the instructions that transfers program control to subroutine such as call subroutine, jump to subroutine, branch to subroutine or branch and save address subsequently, the contents of the call subroutine instruction includes operation code and address, these together determines the beginning of the subroutine.

To execute an instruction two operations must be performed,

1. Store the address of the next instruction present in PC (return address) in a temporary location. This helps the subroutine in identifying the return location.
2. Transfer the control to the beginning of the subroutine.

After this, the return from subroutine which is a last instruction in subroutine transfers the return address stored in temporary location to the program counter. Thereby, the program control is now transferred to that particular instruction whose memory address was previously stored at some temporary location.

Return address may be stored in one of the following locations.

(a) First memory-location of the subroutine.

(b) Fixed memory location.

(c) Processor register.

(d) Stack.

Normally stack stores the return address. In assembly language programming the CALL instruction pushes the address of the instruction following the CALL (return address) on to the stack. The RET instruction removes the return address from the stack so that the program returns to the instruction following the CALL.

The subroutine call implementation can be performed using the following microoperations,

Stackptr ← Stackptr – 1

This microoperation decrements the stack pointer.

M[stackptr] ← PC

This microoperation pushes the program counter contents into the stack.

PC ← effective address

This microoperation transfers the control into the subroutine.

Suppose, if another subroutine is called by allowing the current subroutine in running state then its newly generated return address is, pushed into the stack. The microoperations that implements the instruction which shows returns from last subroutine is,

PC ← M[SP]

This microoperation pops the contents from the stack and transfers them to PC.

SP ← SP + 1

This microoperation increments the stack pointer.

The subroutine stack makes the storage of return address automatic with in a single unit. On the other hand, when a subroutine calls itself, it is referred to as a recursive subroutine. However, the use of recursive subroutine may posses few problems to the programmer. That is, when return address is stored in only one register or memory address and at this point if recursive address call itself, it completely erases the previously stored return address. Thereby, producing potential problems to the programmer. However, the problem can be addressed by, making use of different memory locations for storing return addresses with use of each subroutine. So, usually stack is implemented where in every single return address is pushed into the stack which avoids the destruction of previously stored addresses.

Q38. Discuss in brief about program interrupt.

Answer :

Program Interrupt

While a program is running on a computer, to transfer information from computer to input-output device, firstly we have to set the flag bit. By enabling the flag bit, computer initiates the transfer of data between computer and input-output devices. The rate of information transfer between computer and input-output device makes the system inefficient if the information flow among the devices are different i.e., computer has to check the flag bit several times in order to decide whether to transfer or not the information. Thus, the time required to check the flag bit is wasted resulting in inefficiency as no other useful task can be performed at the same time.

To eliminate consumption of time required to check flag bit each time, an external device of a computer should be assigned with a decision making task. This decision making device acts as a program interrupt which decides when information should be transferred or not. It keeps track of flag bit and informs the computer regarding the possibility of transferring information. A computer does not require to check flag bit as program interrupt informs the computer to transfer information whenever IEN = 1.

Q39. Explain the terms hardware interrupts and software interrupts.

Answer :

Interrupts are generally of two types. They are,

(i) Hardware interrupts

(ii) Software interrupts.

(i) Hardware Interrupts

Hardware interrupts are categorized into two types. i.e., internal interrupts and external interrupts.

❖ **Internal Interrupts**

They are also known as traps due to exceptional conditions generated by the program itself but not by the external event. Signals generated in the CPU hardware also leads to the occurrence of internal interrupt. Internal interrupts are generated due to error condition produced by the execution of an illegal or instruction. Examples of this type of interrupts include register overflow, divided by zero, use of restricted operation codes stack overflow and safety or protection breach. The programmer experience such condition during the premature termination of instruction execution.

❖ **External Interrupts**

External interrupts are derived from a hardware signals. Sources of hardware signal are input/output devices, timer, console switch or any other external sources such as a power sensing circuit. For example an I/O device controller generate request to for data transfer or it can ask for CPU's attention once it completes the data transfer, power failure or elapsed time of an event.

(ii) Software Interrupts

Software interrupts is a special call instruction that acts as an interrupt but not as a subroutine call.

Software interrupt is caused by executing an instruction of type INT. Main difference between internal interrupts and software interrupts is that software interrupts are under the control of the programmer where as internal interrupts are not i.e., the programmer can use software interrupt to raise an interrupt procedure at any point in the program.