Unit - 2

Instruction Codes :-

→ The internal organisation of a digital system (or) a computer is defined as the sequential of microoperations it performs on data stored in registers.

→ The general purpose of a digital computer is capable of executing various microoperations and specific sequence of organisation it must perform.

→ User can control this process by means of programs.

Program :- It is a set of instructions that specify the operations, operands and sequence of its occurrance.

Instruction : A binary code that specifies a sequence of micro-operations for the computer.

Instruction cycle :- The computer reads each instruction from memory and places it in a control Register. The control then interepts the binary code of following instruction and the proceeds to execute it by issuing a sequence of the micro operations.

Instruction code :- The instruction code of an instruction is mainly classified into two types. They are

1. opcode (or) operation code
2. Address (or) operand.

Operation code (opcode) :-

→ The operation code of an instruction is a group of bits. that can define an operation as add, shift, ... etc.

→ The number of bits required for the operation code of an instruction depends upon the total number of operations available in that computer.

→ The operation code must contains atleast n bits for a given 2^n distinct operations.

Address (operand) :-

→ It is used to specify the location of operands such as registers (or) memory words.

→ memory words are specified by their address.

→ Registers are specified by their k-bit binary code.

k-bit address ⇒ 2^k Registers.

Stored program organisation :-

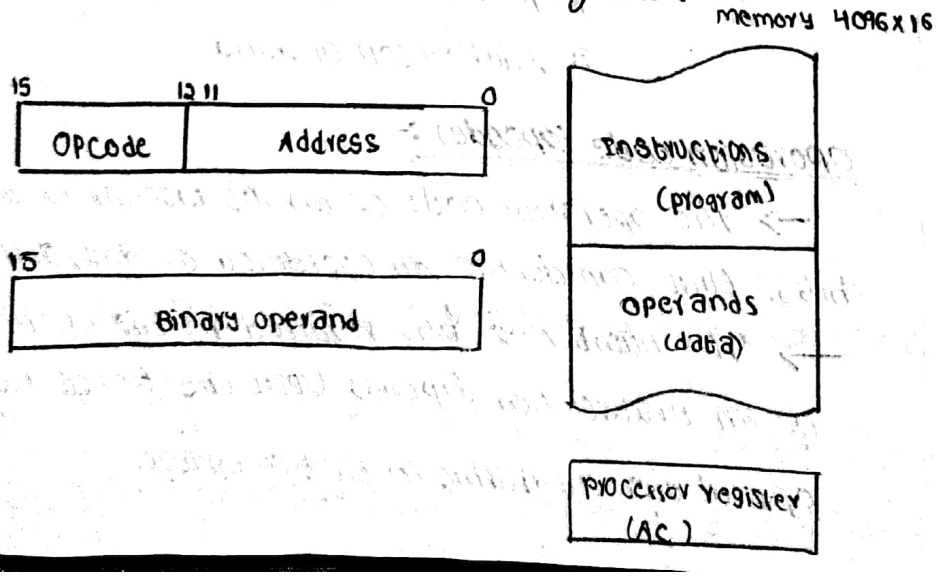
→ It is the simplest way to organise a computer is to have one processor register and an instruction code is formed with two parts. They are

1) First part specifies operation to be performed.

2) Second part specifies an address.

→ The memory address tells about the control where to find an operand in memory.

→ The operand is read from memory and uses as the data to the operated on together.



→ Instructions are stored in one section of memory and data in another.

→ For a memory unit with 4096 words we need 12 bits to specify an address, because $2^{12} = 4096$.

→ If we store each instruction code in one 16-bit memory word, we have available 4 bits for the operation code to specify one out of 16 possible operations and 12-bits to specify the address of operand.

Accumulator (AC) :-

→ Computers that have a single processor register usually assign to it the name accumulator and label it AC. The operation is performed with the memory operand and content of AC.

Addressing of operand :-

→ Sometimes it is convinient to use the address bits of an instruction code not as an address, but as a operand.

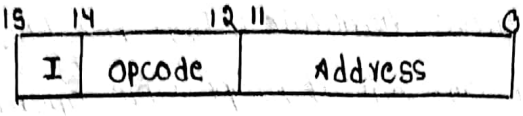
→ When the second part of an instruction code specifies an operand, then the operand is said to have a immediate operand.

→ When the second part specifies the address of an operand, the instruction is said to have a direct address.

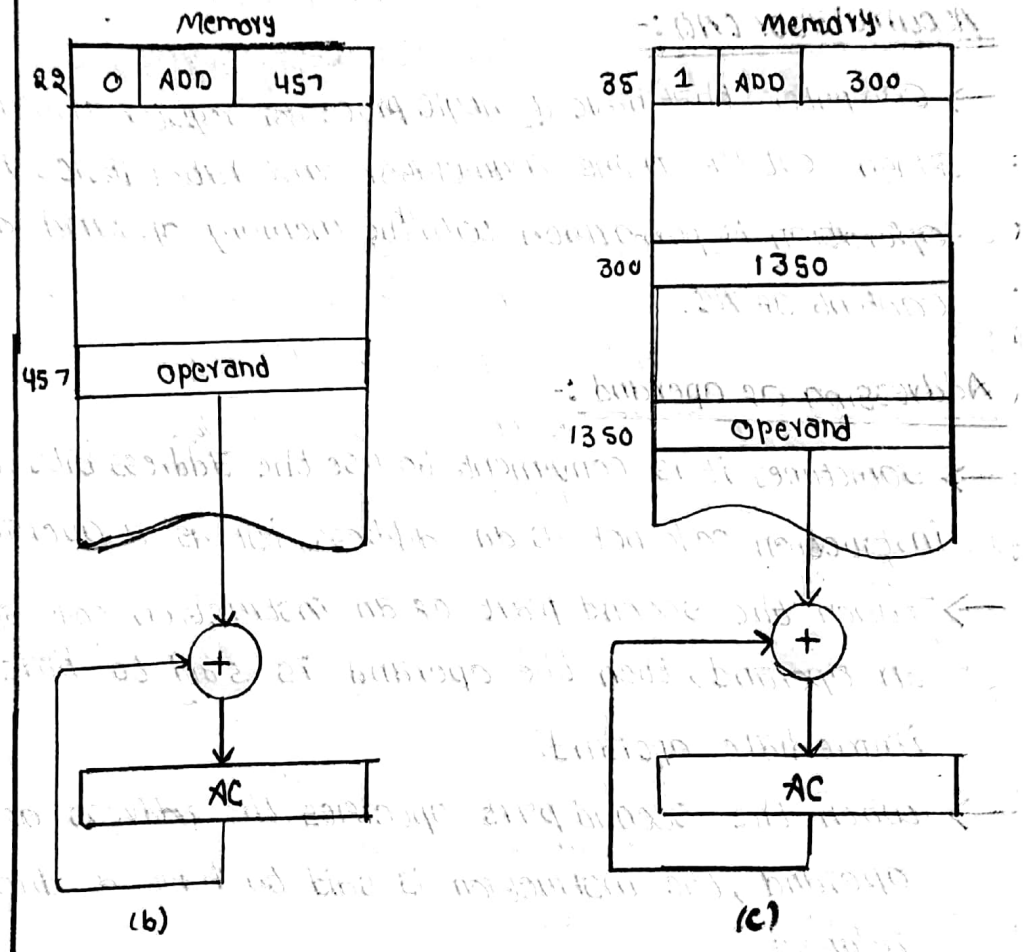
→ When the second part of instruction designate an address of a memory word in which address of the operand is found to have an indirect address.

→ One bit of the instruction code can be used to distingush between a direct and indirect address.

→ The instruction code format shown in figure. It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I. The model bit is 0 for a direct address and 1 for an indirect address.



(a) instruction format



(c) & (b) Demonstration of direct and indirect address.

→ A direct address instruction is shown in figure (b).

→ It is placed in address 22 in memory. The I bit is 0, so the instruction recognised as a direct address.

instruction. The Opcode specifies an ADD instruction, and the Address part is binary equivalent of 457.

→ The control finds the operand in the memory at the address 457 and adds it to content of AC.

→ The instruction in address 35, shown in Figure (c). It has a mode bit I = 1.

→ Therefore, it is recognized as an indirect address instruction.

→ The address part is binary equivalent of 300. The control goes 300 to find address of operand and it results as 1350.

→ The operand found in address 1350 is added to the content of AC.

→ Thus the effective address in instruction of figure (b) is 457 and in instruction of figure (c) is 1350.

Computer Registers :-

→ Computer instructions are normally stored in consective memory locations and are executed sequentially

→ The control reads an instruction from the specific memory and executes it.

→ The need of register in computer for

* instruction sequencing needs a counter to calculate the address of next instruction after execution of current instruction is completed. For this purpose we need a Program Counter (PC).

* It is necessary to provide a register in the control unit for storing the instruction code after it is read from memory (IR).

* Needs processor registers for manupulating data(AC&TR) and a register for holding a memory address (AR).

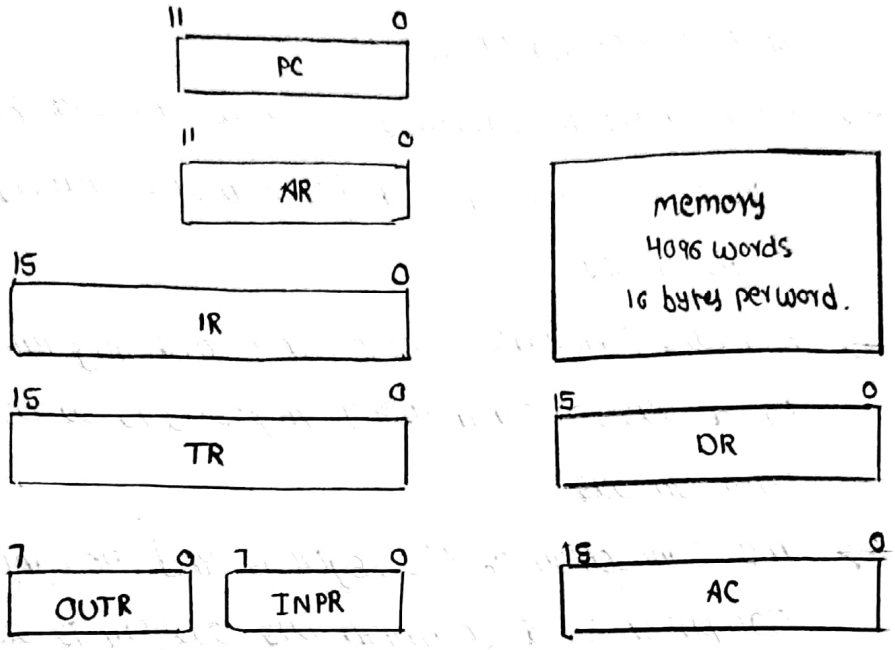
→ The above requirements dictate the register configuration shown in below figure.

→ The Registers are also listed in below table with a brief description of their function and number of bits that they have.

Register Symbol	Number of Bits	Registername	Function
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address of memory
AC	16	Accumulator	processor Register
IR	16	Instruction Register	Holds instruction code
PC	12	program counter	Holds address of instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds input character
OutR	8	Output Register	Holds Output Character

→ The data Register holds the operand read from the memory.

→ The Accumulator (AC) register is a general purpose processing Register.



Basic Computer Registers & memory.

→ The instruction Read from memory is placed in the Instruction Register (IR).

→ The Temporary Register (TR) is used for holding the temporary data during the process.

→ The memory address Register (AR) has 12 bits since this is the width of memory address.

→ The program counter (PC) also has 12 bits and it holds the address of next instruction to be read from memory after current instruction is executed.

→ Two Registers are used for input and output.

* The input Register (INPR) receives an 8-bit character from an input device.
* The output Register (OUTR) holds an 8-bit character for an output device.

Common Bus System :-

→ The basic computer has eight registers, a memory unit and a control unit.

→ paths must be provided to transfer data from one Register to another and between the memory and the Register too.

→ A more efficient scheme for transfering information in a system with many registers is to use a common bus.

→ The connection of the registers and memory of the basic computer to a common bus system is shown in figure.

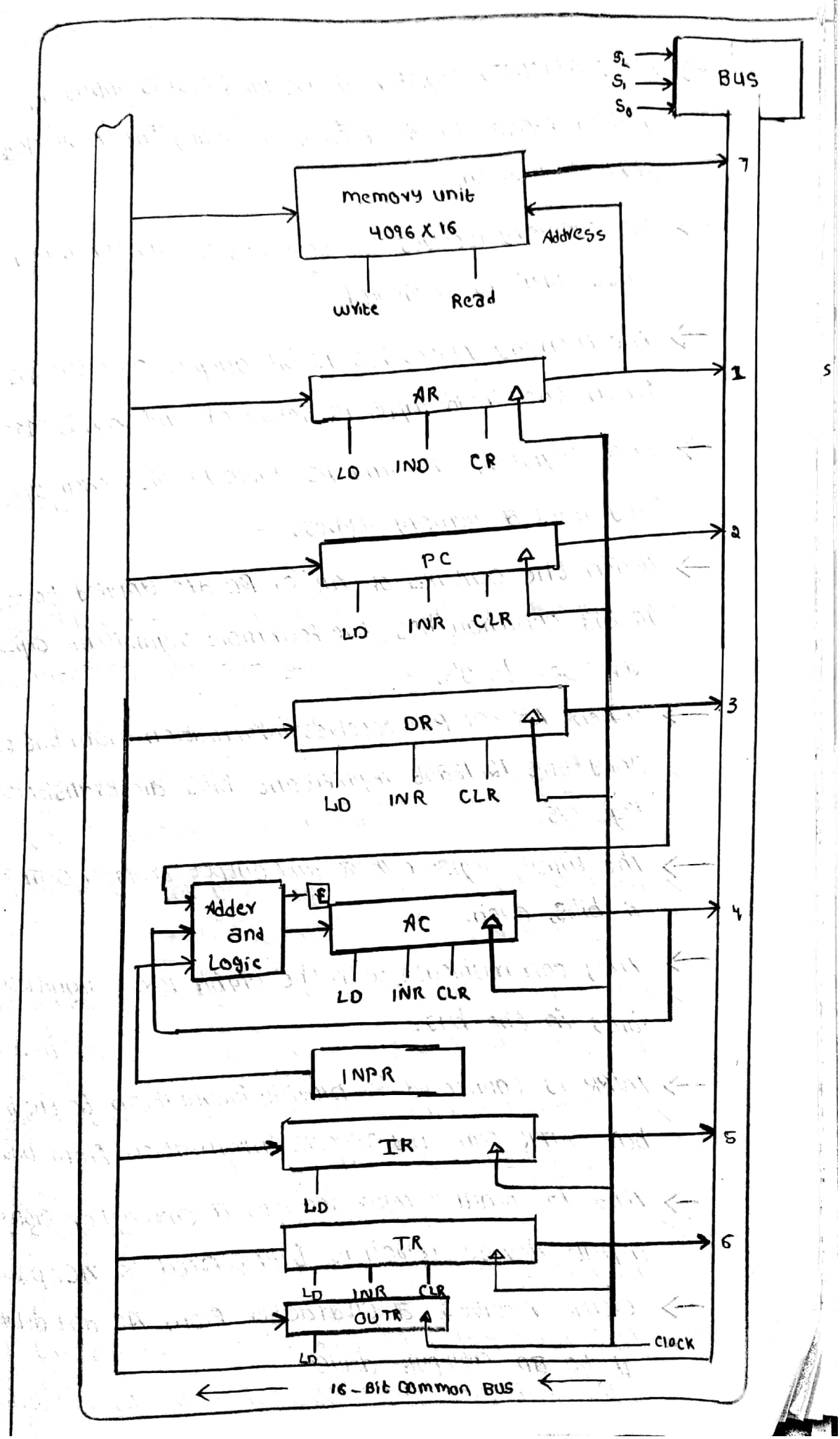
→ The output of seven registers and memory are connected to the common bus.

→ The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables.

→ The number along each output shows the decimal equivalent of the required binary selection.

→ For ex, the number along the output of DR is 3. The 16-bit outputs of DR are placed on the bus lines when $S_2S_1S_0 = 011$.

→ The lines from the common bus are connected to the inputs of each register and the data inputs of the memory.



→ the particular Register whose load (LD) is input is enabled receives the data from bus during the next clock pulse transition.

→ The memory receives the contents of bus when it is write input is activated.

→ The memory places its 16-bit output onto the bus when the read input is activated and $S_2S_1S_0 = 111$

→ Two Registers, AR and PC, have 12 bits each since they hold a memory address.

→ When the contents of AR or PC are applied to the 16-bit common bus, the four more significant digits are set to 0's.

→ When AR or PC receives information from the bus, only the 12 least significant bits are transfer into Registers.

→ The input Register INPR and output Register OUTR have 8 bits each.

→ They communicate with the eight least significant bits in the bus.

→ INPR is connected to provide information to the bus but OUTR can only receive information from bus.

→ This is because INPR receives a character from an input device which is transferred to AC.

→ OUTR receives a character from AC and delivers it to an output device.

→ Five Registers have three control inputs : LD(load), INR(increment) and CLR (clear).

→ This type of Register is equivalent to a binary counter with parallel load and synchronous clear.

→ Two Registers have only a LD input.

→ the input data and output data of the memory are connected to the common bus, but the memory address is connected to AR.

→ Therefore, AR must always be used to specify a memory address.

→ The 16 inputs of AC come from an adder and logic circuit. The circuit has three sets of inputs.

* one set of 16-bit inputs come from the data Register DR.

* Another set of 16-bit inputs come from outputs of AC.

* The resultant of addition is transferred to AC and the end carry-out of addition is transferred to the flipflop E.

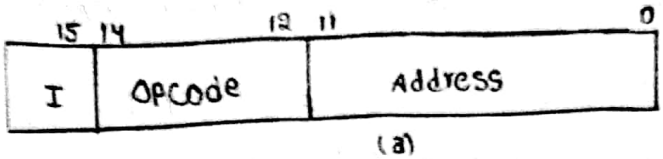
* A third set of 8-bit inputs come from the input register INPR.

→ The content of any register can be applied onto the bus and an operation can be performed in adder and logic circuits during the same clock cycle.

→ This can be done by placing the contents of AC on the bus (with $S_2S_1S_0 = 100$), enabling the LD (load) input of DR, transferring the content of DR through the adder and logic circuit into AC.

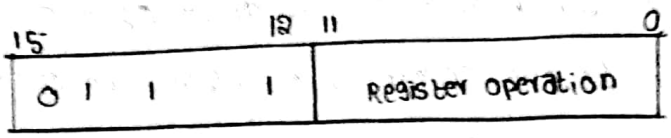
3. Computer Instructions :-

→ The basic computer has three instructions code formats as shown below,



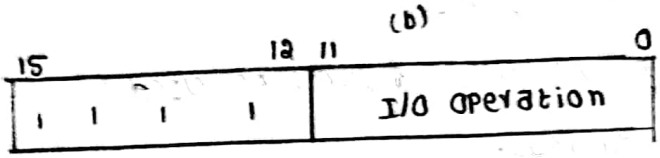
(opcode = 000 through 110)

(a)



(opcode = 111, I = 0)

(b)



(opcode = 111, I = 1)

(c).

→ The operation code (opcode) part of the instruction contains three bits and the meaning of remaining 13 bits depends on the operation code encountered.

→ A memory reference instruction has use 12 bits to specify an address and 1 bit to specify addressmode.

→ Address mode (I) is equal to 0 for direct address and 1 for indirect address.

→ The Register Reference instructions are recognized by the operation code 111 with a 0 in the left most bit.

→ A Register Reference instruction specifies an operation on AC register.

→ An input - output instruction does not need a reference to memory and it recognised by opc 111.

→ The remaining 12 bits are used to specify the type of input - output operation. The instructions are

Table : Basic computer Instructions

Symbol	Hexa decimal code		Description
	I=0	I=1	
AND	0XXX	8XXX	AND memory word to AC
ADD	1XXX	9XXX	ADD memory word to AC
LDA	2XXX	AXXX	Load memory word to AC
STA	3XXX	BXXX	Store content of AC in memory
BUN	4XXX	CXXX	Branch unconditionally
BSA	5XXX	DXXX	Branch and save return address
ISZ	6XXX	EXXX	Increment and Skip if zero

CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right AC & E
CIL	7040	Circulate left AC & E
INC	7020	Increment AC
SPA	7010	Skip next, if AC positive
SNA	7008	Skip to next, if AC negative
SZA	7004	Skip to next, if AC zero
SZE	7002	Skip to next, if E is 0.
HLT	7001	Halt computer.

INP	F800	Input character to AC
OUT	F400	Output character from AC
SKI	F200	Skip on input flag
SKO	F100	Skip on output flag
ION	F080	Interrupt on
IOF	F040	Interrupt OFF.

→ The symbol designation is a three-letter word and represents an abbreviation intended for programmers and users.

→ The hexa decimal code is equal to the equivalent hexa decimal number of binary code used for instruction.

Instruction set completeness :-

→ A computer should have a set of instructions so that the user can construct machine language program to evaluate any function.

→ The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each categories as :-

* Arithematic, logical and shift operations.
* Data instructions.
* program control (or) Brach.
* Input and output instructions.

→ There is one arithematic instruction, ADD, and two related instructions, complement AC (CMA) and the increment AC (INC), with this we can add and subtract binary numbers when negative numbers are given.

→ The circulate instruction, CIR and CIL ; are used for arithematic shift and as well as other type of shifts.

→ There are three logical operations : AND, complement AC (CMA) and AC (INC). With these three instructions we can add and subtract binary numbers when negative numbers

→ moving information from memory to AC is accomplished with the load AC (LDA) instruction. Storing information from AC into memory is done with store AC.

→ The branch instructions BUN, BSA and ISZ, together with the four skip instructions, provide capabilities for program control and checking of status conditions.

→ The input (INP) and output (OUT) instructions cause information to be transferred between computer and external devices.

Timming and control :

→ The Timming for all registers in the basic computer is controlled by a master clock generator.

→ The clock pulses are applied to all flip-flops and the registers in the system, including the flip-flops and registers in control unit.

→ The clock pulses do not change the state of a register is enabled by control signal.

→ The control signals are generated in the control unit and provide in the inputs for the multiplexers in the common bus, control inputs in processor registers, and micro operations for the accumulator.

→ There are two major types of control organisation :

* Handwired control
* Microprogrammed control.

→ the difference between handwired and micro programmed control are :-

Handwired control	microprogramed control
* The control logic is implemented with gates, flipflops, decoders and other digital circuits.	* The control information is stored in control memory. It is used to intitate the required sequences.
* The advantage that it can be optimized to produce a fast mode of operation.	* compared with the handwired control operation is slow.
* Requires changes in the wiring among the various components.	* modifications can be done by updating the micro program.

→ The block diagram of the handwired control unit is as shown in figure.

→ It consists of two decoders, a sequence counter, and a number of control logic gates.

→ An instruction read from memory is placed in the instruction Register (IR). It divided into 3 parts :

(i) The I bit

(ii) The operation code

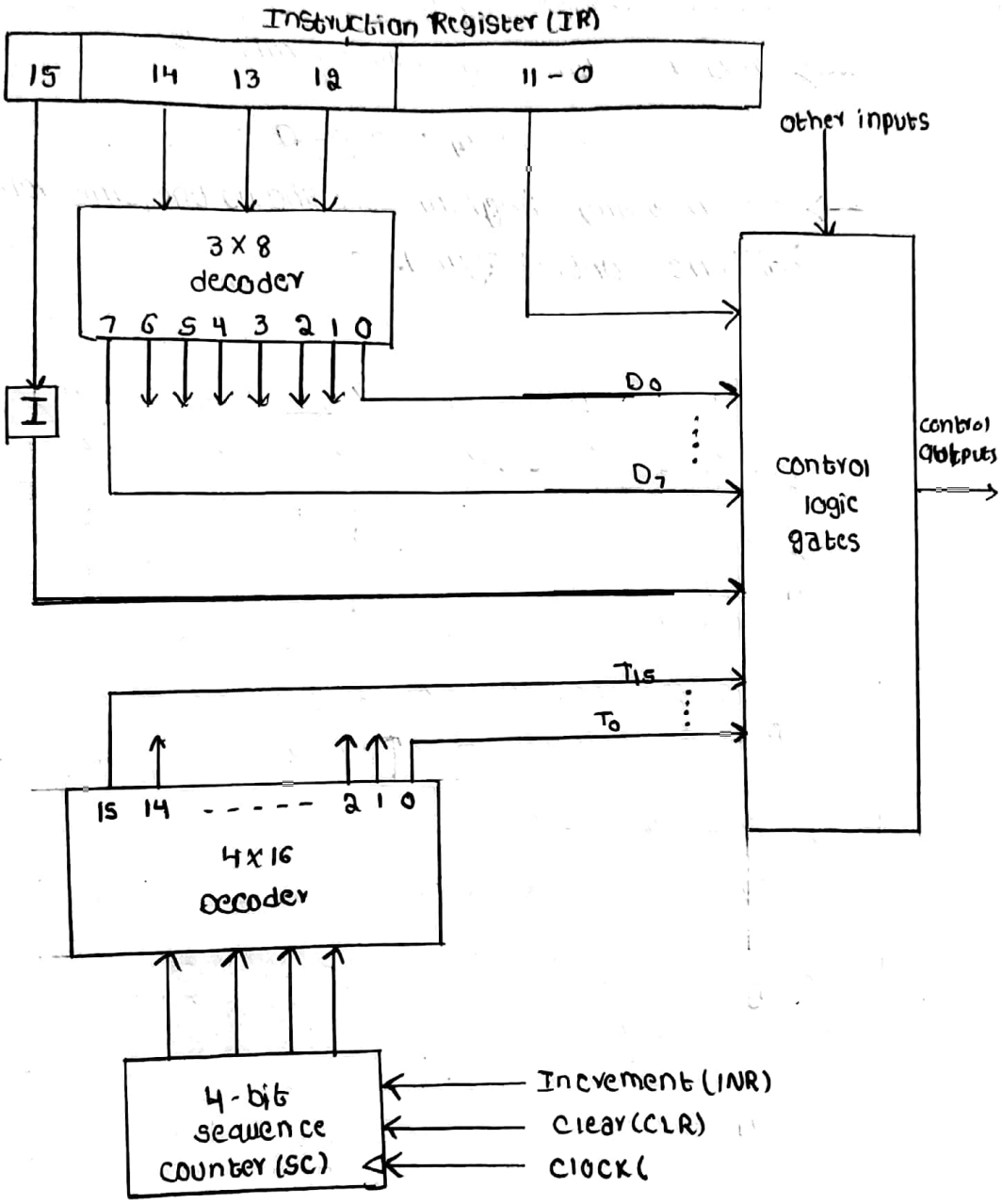
(iii) The bits 0 through 11.

→ The operation code in bits 12 through 14 are decoded with a 3x8 decoder. The eigth outputs of the decoder are designed by symbols D_0 through D_7.

→ Bit 15 of the instruction is transferred through a flip-flop designated by symbol I.

→ Bits 0 through 11 are applied to control logic gates.

→ The 4-bit sequence counter can count in binary from 0 through 15.



Control unit of basic computer.

→ The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}.

→ The sequence counter SC can be incremented (or) cleared synchronously.

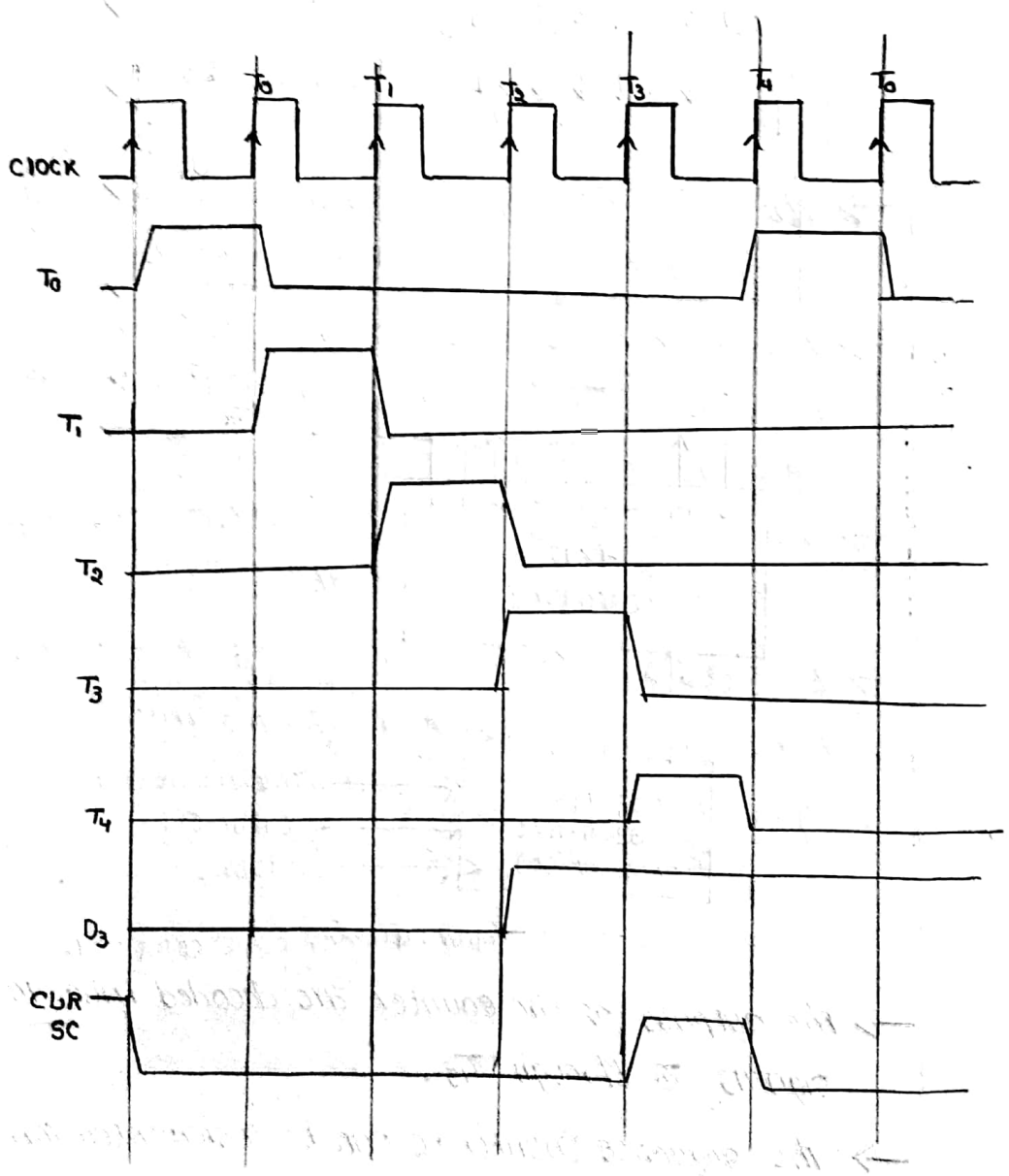
→ The counter is incremented to provide the sequence of timing signals out of 4x16 decoder.

→ As an example, consider the case where SC is incremented to provide timing signals T_0, T_1, T_2, T_3 and T_4 in sequence. At time T_4, SC is cleared to 0 if the decoder output D_3 is active.

→ This is expressed symbolically as

$$D_3 T_4 : SC \leftarrow 0.$$

→ The Timming diagram will shows the time relationship of the control signal.



Example of control timming signals.

→ The sequence counter SC responds to this positive transition of clock.

→ Intially, the CLR input of SC is active. The first positive transition of the clock clears SC to 0, which in turn activates the timming signal.

→ SC is incremented with every positive clock transition, unless its CLR input is active.

→ This produces the sequence of timing signals $T_0, T_1, T_2,$ T_3, T_4 and so on, as shown in diagram.

→ The last three waveforms in figure shown SC is cleared when $D_3 T_4 = 1$.

→ Output D_3 from the operation decoder becomes active at end of Timming signal T_2.

→ When Timming signal T_4 becomes active, the output of the AND gate that implements the control function $D_3 T_4$ becomes active.

→ This signal is applied to the CLR input of SC. on the next positive clock transition the counter clear to 0.

→ This causes the timming signal T_0 to become active instead of T_5 that would have been active if SC were incremented instead of cleared.

Instruction cycle :-

→ A program residing in the memory unit of the computer consists of a sequence of instructions.

→ The program is executed in the computer by going through a cycle for each instruction.

→ Each instruction cycle in turn is subdivided into a sequence of subcycles or phases.

→ In the basic computer each code instruction cycle consists of the following phases :

1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory.
4. Execute the instruction.

→ Upon the completion of step4, the control goes back to step 1 to fetch, decode and execute the next instruction.

Fetch and Decode :-

→ Initially, the program counter PC is loaded with the address of the first instruction in program.

→ The sequence counter SC is cleared to 0, providing a decoded Timming signal T_0.

→ The micro operations for the fetch and decode phases can be specified by following register transfer statement.

→ The below figure shows how the first two register transfer statements.

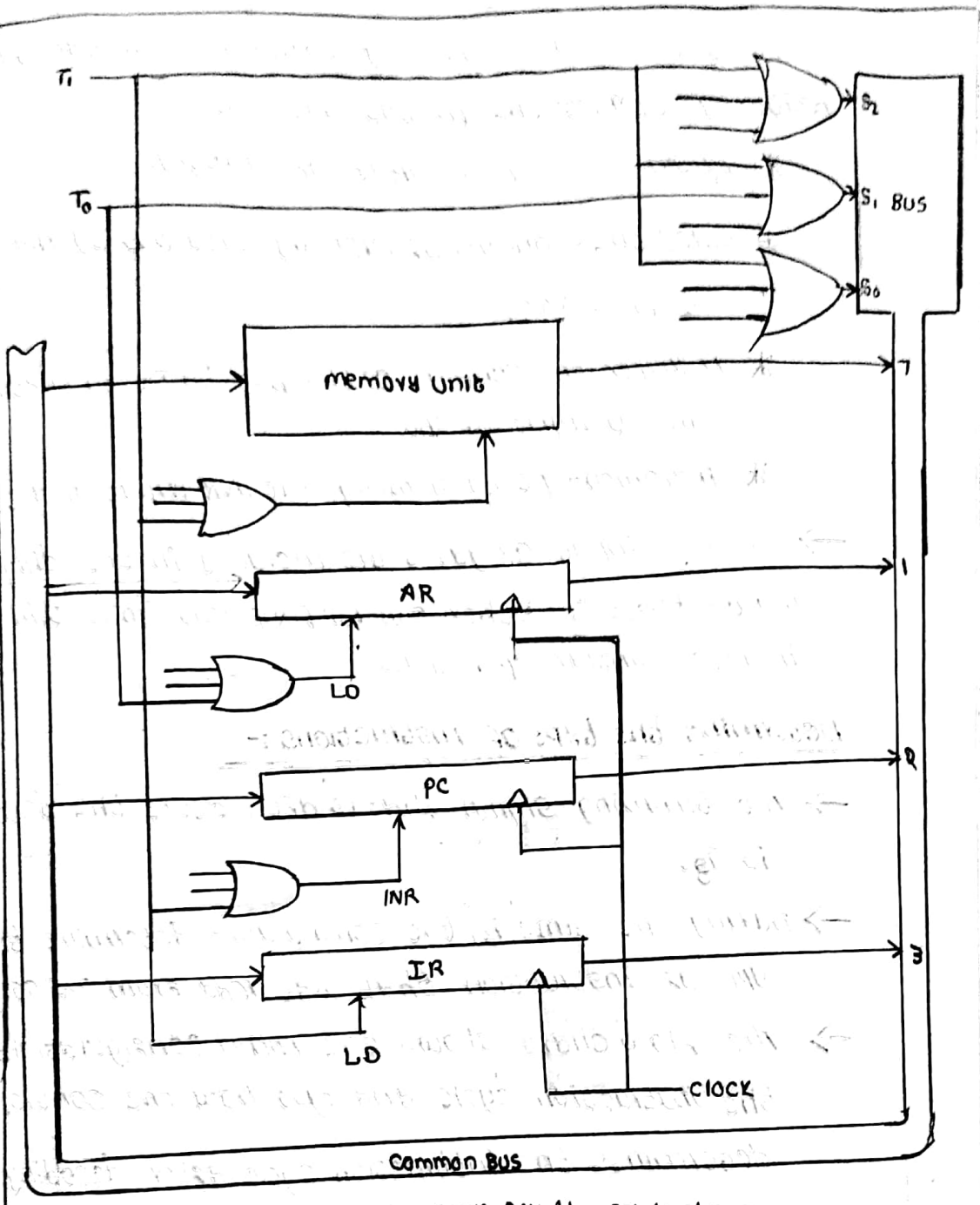
$$T_0 : AR \leftarrow PC$$

$$T_1 : IR \leftarrow M[AR],\ PC \leftarrow PC + 1$$

$$T_2 : D_0, \ldots, D_7 \leftarrow \text{Decode } IR(12\text{-}14)$$

$$AR \leftarrow IR(0\text{-}11),\ I \leftarrow IR(15)$$

→ The figure shows how the first two registers transfer statements are implemented in the bus system.



Register transfers for the fetch phase.

→ To provide the data path to a transfer of pc to AR, we must apply timing signal to the T0, it achelve the following connection.

* place the content of PC onto the bus by making the bus selection input S_2, S_1, S_0 equal to 010.

* Transfer the content of bus to AR by enabling the LD input of AR.

→ In order to implement the second statement it is

necesary to use the timming signal T_i to provide the following connections in bus system.

* Enable the read input of memory.
* place the content of memory onto bus by making $S_2 S_1 S_0 = 111$.
* Transfer the content of the bus to IR by enabling the LD input of IR.
* Increment PC by enabling the INR input of PC.

→ Multiple input OR gates are included in the diagram because there are other control functions that will be intiate similar operations.

Determine the type of Instructions :-

→ The timming signal that is after active the decoding is T_3.

→ During the time T_3, the control unit determine the type of instruction that was read from memory.

→ The flow chart shows the intial configurations for the instruction cycle and also how the control determines the instruction cycle after decoding.

→ Decoder output D_7 is equal to 1, if the operation code is equal to binary 111.

→ If $D_7 = 1$, the instruction must be a register-reference (or) input-output type..

→ If $D_7 = 0$, the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction.

→ Control then inspects the value of the first bit of the instruction, which is in flip-flop I.

→ If $D_7 = 0$ and $I = 1$, indicates a memory-reference instruction with a direct address.

→ If $D_7 = 1$ and $I = 0$, indicates a register-reference instruction.

→ If $D_7 = 01$ and $I = 1$, indicates an input-output instruction.

→ The ~~three~~ instruction types are subdivided into four seperate paths.

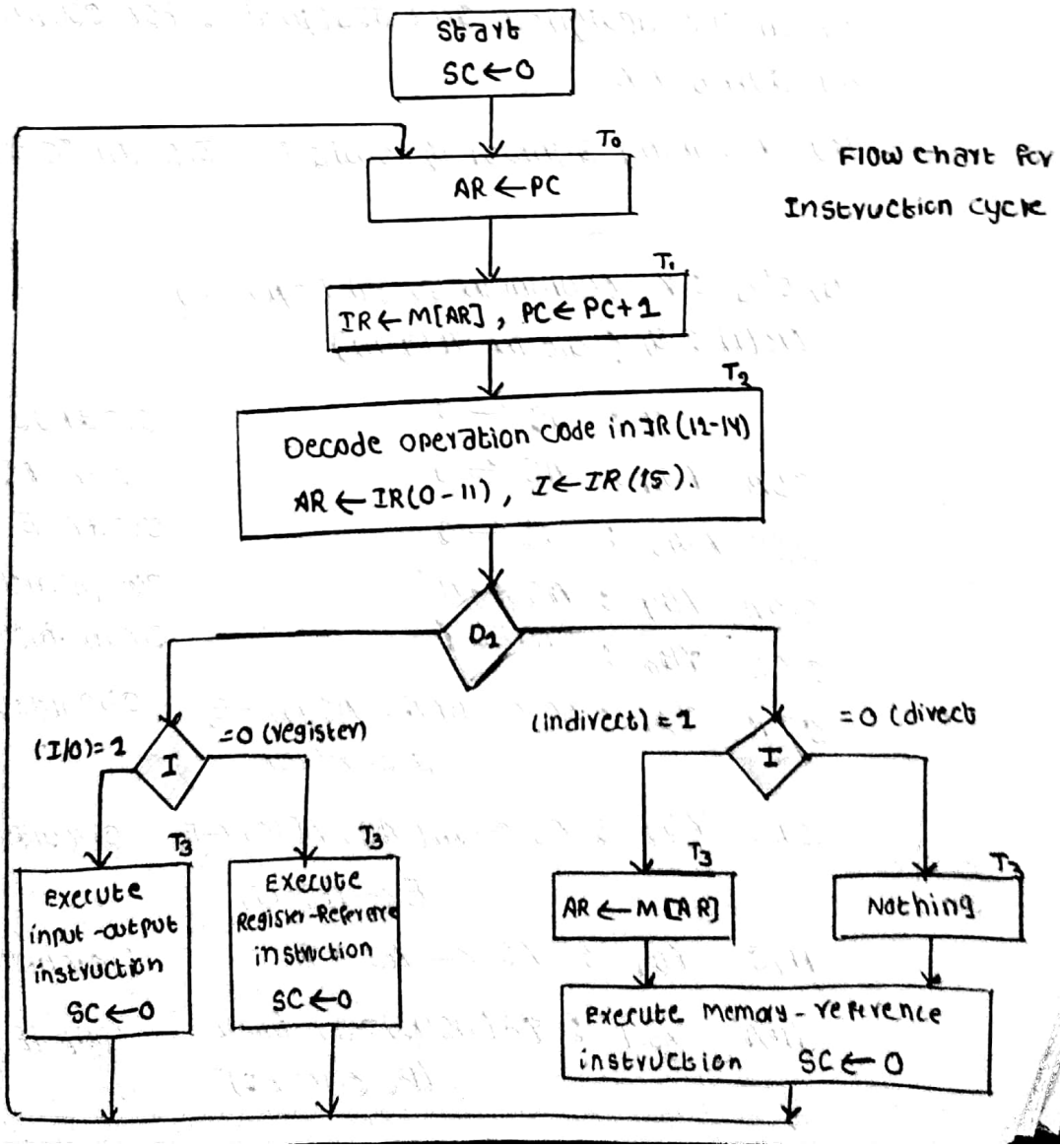
→ The selected operation is activated with the clock transition associated with timming signal T_3 as

$D_7'IT_3$: AR ← M[AR]

$D_7'I'T_3$: Nothing

$D_7I'T_3$: Execute a register

D_7IT_3 : Execute an input-output instruction.



Flow chart for Instruction cycle

Register - Reference instructions :-

→ Register - Reference instructions are recognized by the control, when $D_7 = 1$ and $I = 0$.

→ These instructions are use bits 0 through 11 of the instruction code to specify one of 12 instructions.

→ These 12 bits are available in IR (0-11).

→ The control functions and micro operations for the register-reference instructions are listed in table.

→ These instructions are executed with the clock transition associated with the timming variable T_3.

→ control function needs the Boolean relation $D_7 I' T_3$, which we designed and designated for convinience by symbol r.

→ By assigning symbol B_i to bit i to IR, all controls be

$D_7 I' T_3 : r$ (common to all Registers)
$IR(i) : B_i$ (bit in IR (0-11))

	r :	$SC \leftarrow 0$	clear SC
CLA	rB_{11} :	$AC \leftarrow 0$	clear AC
CLE	rB_{10} :	$E \leftarrow 0$	clear E
CMA	rB_9 :	$AC \leftarrow \overline{AC}$	complement AC
CME	rB_8 :	$E \leftarrow \overline{E}$	complement E
CIR	rB_7 :	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	circulate right
CIL	rB_6 :	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$	circulate left
INC	rB_5 :	$AC \leftarrow AC + 1$	increment AC
SPA	rB_4 :	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive

---

SNA	YB_3 : IF (AC(15) = 1) then (PC ← PC+1)	Skip if negative
SZA	YB_2 : IF (AC = 0) then (PC ← PC+1)	Skip if AC zero
SZE	YB_1 : IF (E = 0) then (PC ← PC+1)	Skip if E zero
HLT	YB_0 : S ← 0 (when S is a Start-Stop flip-flop)	Halt Computer.

Execution of Register-Reference instructions.

→ For example, the instruction CLA has a hexadecimal code 7800, which gives binary equivalent as below

0111 1000 0000 0000, The first bit is a zero.

→ The next three bits constitute the operation code and are recognized by decorder output D_7.

→ Bit 11 in IR is 1, and is recognized from B_{11}. The control function that initiates the micro operation for instruction.

→ The Execution of a Register-reference instruction is completed at time T_3.

→ The sequence counter SC is cleared to 0 and the control goes back to fetch, with timming signal T_0.

→ The condition control statements must be recognised as part of the control conditions.

→ The AC is positive, when the sign bit in (AC(15)) = 0; it is negative when AC(15) = 1. The content of AC is zero (AC = 0) if all flip-flops of registers are zero.

→ The HLT instruction clears a start-stop flip flops and the sequence counter from counting.

Memory-Reference Instructions :-

→ The table shows the seven memory-reference instructions.

→ The decoded output D_i for $i = 0, 1, 2, 3, 4, 5$ and 6 from the operation decoder that belongs to each instruction is included in table.

→ The effective address of the instruction is in the address register AR and was placed there during the timming signal T_2, when $I=0$ (or T_3 when $I=1$.

→ The execution of memory-reference instruction is starts with Timming signal T_4.

→ The symbolic description of each instruction is specified in table in terms of register transfer form.

Symbol	operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, PC \leftarrow AR+1$
ISZ	D_6	$M[AR] \leftarrow M[AR]+1$

AND to AC :-

→ This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by address.

→ The sum is transferred into AC and output carry Cout is transferred to E flipflop.

→ The microoperations needed to execute this are

$D_1 T_4$: $DR \leftarrow M[AR]$

$D_1 T_5$: $AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$.

LDA : load to AC :-

→ This instruction transfers the memory word that is specified by effective address to AC.

→ The microoperations needed to execute this instructions are

$D_2 T_4$: $DR \leftarrow M[AR]$

$D_2 T_5$: $AC \leftarrow DR$, $SC \leftarrow 0$.

STA : store AC :-

→ This instruction stores the contents of AC into the memory word specified by effective address.

→ Since the output of AC is applied to the bus and the data input of memory is connected to bus, we can execute this instruction as

$D_3 T_4$: $M[AR] \leftarrow AC$, $SC \leftarrow 0$

BUN : Branch unconditionally :-

→ This instruction transfers the program to the instruction specified by the effective address.

→ The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches unconditionally.

→ The micro operation is executed with micro operation as : $D_4 T_4$: $PC \leftarrow AR$, $SC \leftarrow 0$.

BSA: Branch and save address of Return :-

→ This instruction is useful for branching to a portion of the program called sub-routine (or) procedure.

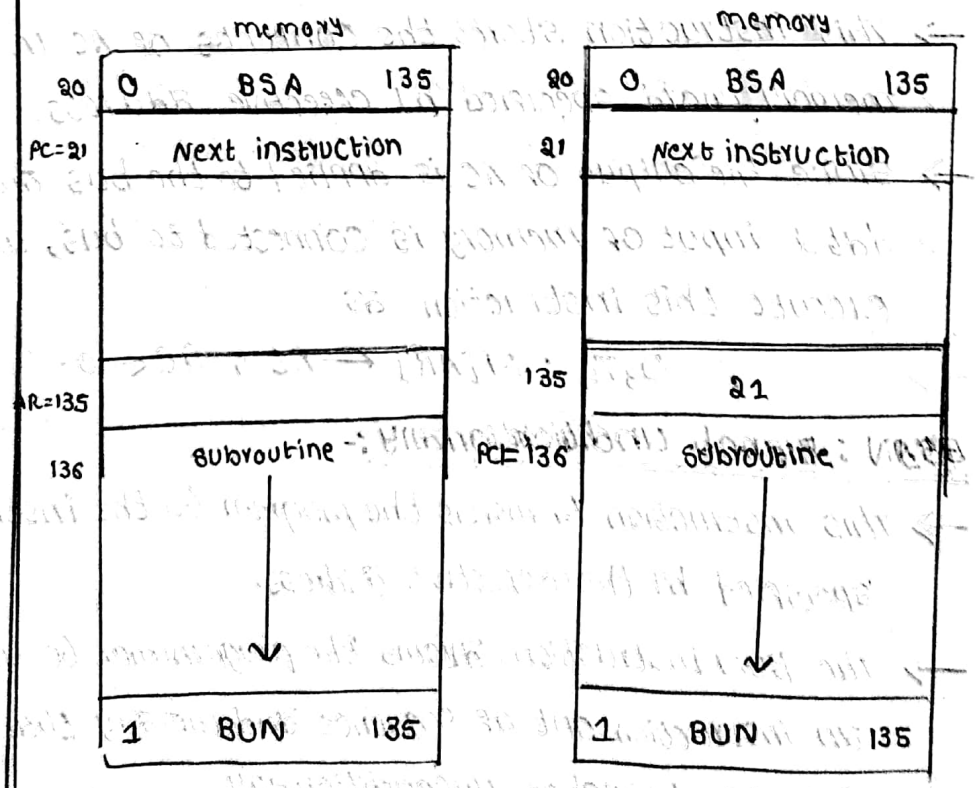
→ When executed, the BSA instruction stores the address of next instruction in sequence into a memory location.

→ The effective address plus one is then transferred to PC to serve as address of first instruction in subroutine.

→ This operation was specified with following register transfer :

$$M[AR] \leftarrow PC,\ PC \leftarrow AR + 1$$

→ A numeric example that demonstrates how this instruction is used with sub-routine as shown.



(a) memory, PC and AR at time T_4

(b) memory and PC after execution

→ The BSA instruction is assumed to be in memory at address 20.

→ The I bit is 0 and the address part of instruction has the binary equivalent of 135.

→ After the fetch and decode phases, PC contains 21, which is the address of next instruction in program.

→ This is shown in part(A) of figure.

→ The BSA instruction performs the following operation

$$M[135] \leftarrow 21,\ PC \leftarrow 135 + 1 = 136.$$

→ The result of this operation is shown in part(B).

→ The return address 21 is stored in memory location 135 and control continues with subroutine program.

→ The return to the original program is accomplished by means of an indirect BUN instruction placed at end.

→ When this instruction is executed, control goes to the indirect phase to read effective address at location 135.

→ When the BUN instruction is executed, the effective address 21 is transferred to PC.

→ The BSA instruction must be executed with a sequence of two micro operations.

$$D_5 T_4 : M[AR] \leftarrow PC,\ AR \leftarrow AR + 1$$

$$D_5 T_5 : PC \leftarrow AR,\ SC \leftarrow 0.$$

ISZ : increment and skip if zero :-

→ This instruction increment the word specified by the effective address, and if the incremented value is equal to 0, PC is increment by 1 to skip next.

→ Since it is not possible to increment a word

inside the memory, it is necessary to read the word into DR, increment DR, and store back in memory.

→ this is done with following sequence

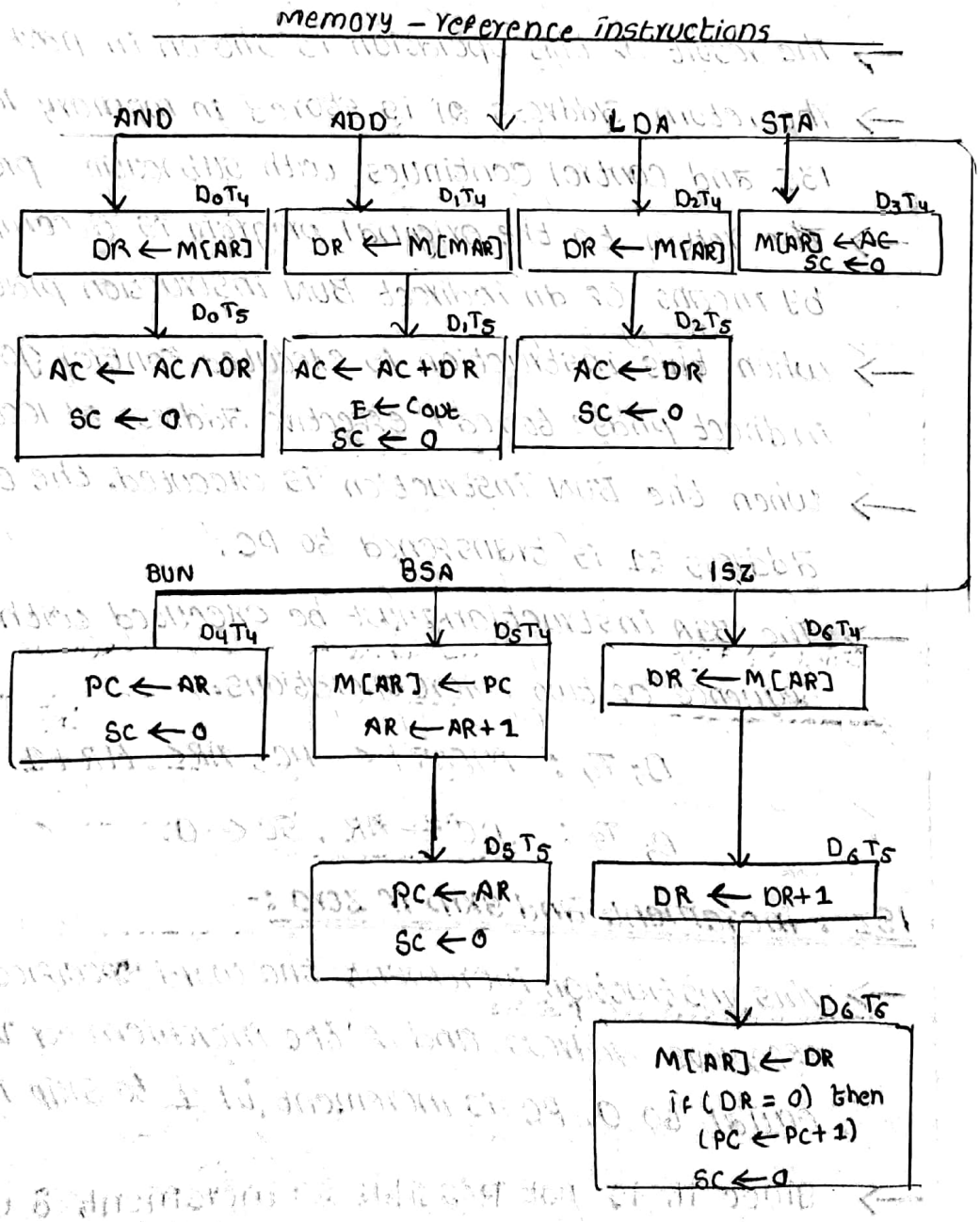
D_6T_4 : $DR \leftarrow M[AR]$

D_6T_5 : $DR \leftarrow DR + 1$

D_6T_6 : $M[AR] \leftarrow DR$, $SC \leftarrow 0$.

if $(DR = 0)$ then $(PC \leftarrow PC + 1)$.

Control Flow chart :-



Flow chart for memory reference instructions

7. Input-Output and Interrupt :-

→ Instructions and data stored in memory must come from some input device.

→ Computational results must be transmitted to the user through some output device.

→ To demonstrate the most basic requirements for input and output communication, we will use as an illustration a terminal unit with a keyboard & printer.

Input-output configuration :-

→ The terminal sends and receives serial information.

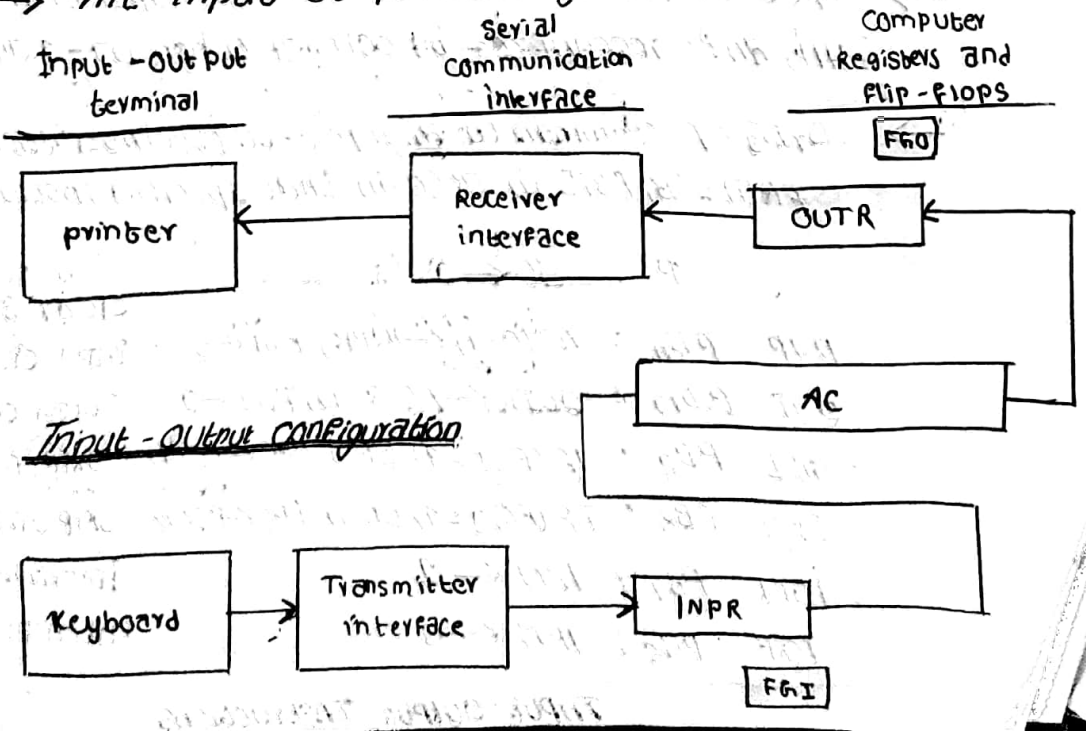
→ Each quantity of information has 8 bits of an alpha numeric code.

→ The serial information from the keyboard is shifted into the input register INPR.

→ The serial information for the printer is stored in the output register OUTR.

→ These two registers communicate with a communication interface serially and with the AC in parallel.

→ The input-output configuration is shown as



Input-Output configuration

→ The input Register INR consists of 8 bits and holds alpha numeric input information.

→ The 1-bit input flag FGI is a control flip-flop.

→ The flag bit is set to 1 when new information is available in input device and cleared to 0 when the information is accepted by computer.

→ The output Register OUTR works similarly but the direction of flow of information is reversed.

→ Intially, the output flag FGO is set to 1.

→ The computer checks the flag bit; if it is 1, then the information from AC is transferred in parallel to OUTR and FGO is cleared to 0.

Input-Output Instructions :-

→ Input and output instructions are needed to transfer information to and from AC register, for checking the flag bits, and for controlling interrupt facility.

→ Input-output instructions have an operation-code 1111 and recognized by control when $D_7 = 1$ and $I = 1$.

→ $D_7 I T_3 = p$ (common to all input-output instructions)
$IR(i) = B_i$ [Bit in IR(6-11) that specifies instructions]

	p	: $SC \leftarrow 0$	clear SC
INP	pB_{11}	: $AC(0-7) \leftarrow INPR, FGI \leftarrow 0$	input character
OUT	pB_{10}	: $OUTR \leftarrow AC(0-7), FGO \leftarrow 0$	Output character
SKI	pB_9	: If $(FGI = 1)$ then $(PC \leftarrow PC+1)$	Skip on input flag
SKO	pB_8	: If $(FGO = 1)$ then $(PC \leftarrow PC+1)$	Skip on output flag
ION	pB_7	: $IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6	: $IEN \leftarrow 0$	interrept enable off

Input-Output Instructions

→ These instructions are executed with the clock transition associated with timming signal T_3.

→ Each control function needs a Boolean relation D_7IT_3, which we designated for convenience by symbol p.

→ The control function is distinguished by one of the bits in IR (6-11).

→ By assigning the symbol B_i to Bit i of IR, all control functions can be denoted by pB_i for i=6 through 11.

→ The sequence counter SC is cleared to 0, when $p = D_7IT_3 = 1$.

→ The last two instructions set and clear an interrupt enable flip-flop IEN.

program Interrupt :

→ The computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer.

→ The difference of information flow rate between the computer and that of the input-output device makes this type of transfer inefficient.

→ An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer.

→ In the meantime the computer can be busy with other tasks. This type of transfer uses the interrupt facility.

→ While the computer is running a program, it does not check the flags.

→ When a flag is set, the computer is momentarily interrupted from the current program.

→ Then computer deviates momentarily from the current program.

→ It returns to the current program to continue what it was doing to perform input or output transfer.

→ The interrupts enable flip-flops IEN can be set and cleared with two instructions.

★ When IEN is cleared to 0, the flags cannot interrupt the computer.

★ When IEN is set to, the computer can be interrupted.

→ The way that the interrupt is handled by the device can be explained by above flowchart.

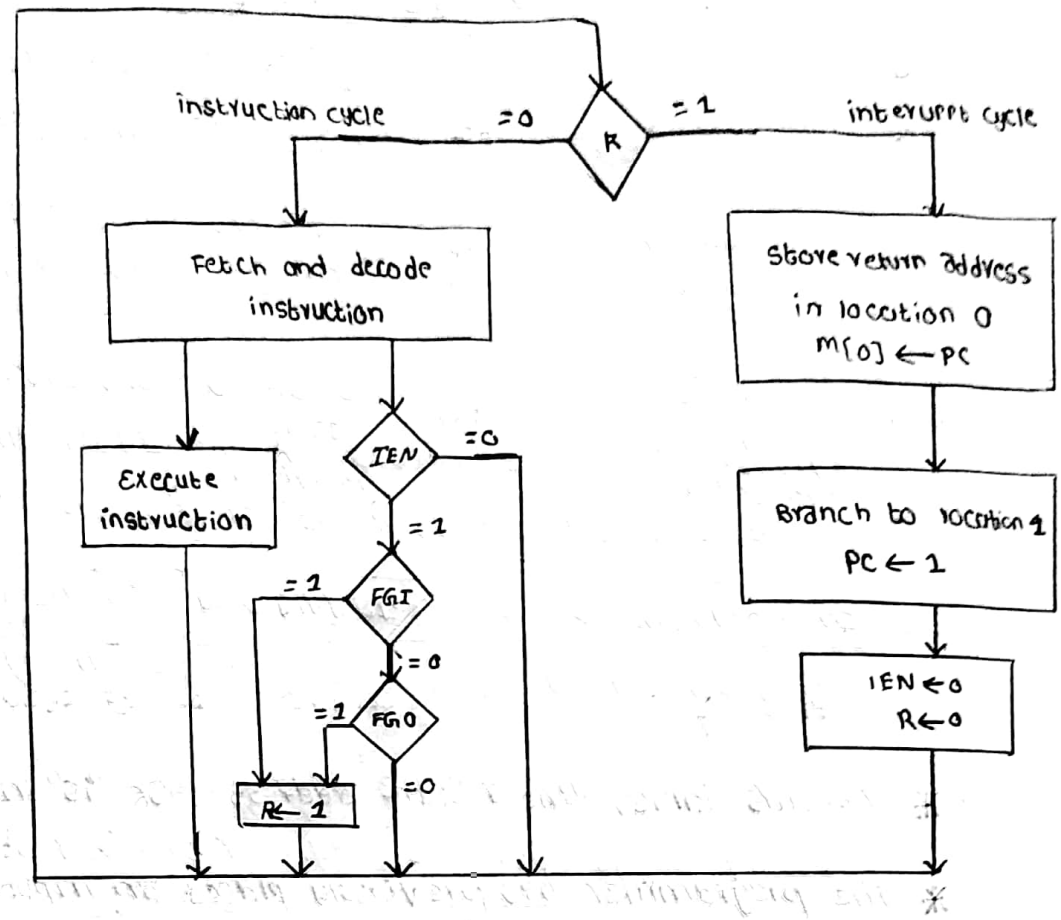
→ An interrept flip-flop R is included in computer. When R=0, it goes through an instruction cycle.

→ During the execute phase of instruction cycle IEN is checked by control.

→ If it is 0, that indicates the programmer does not want to use the interrupt, so the control will continues with next instruction.

→ If IEN is 1, control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer information.

→ If either flag is set to 1, while IEN = 1, flipflop R is set to 1. At the end of execute phase, control checks the values of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.



Flowchart of interrupt cycle.

Interrupt cycle :-

→ The interrupt cycle is a hardware implementation of a branch and save-return address operation.

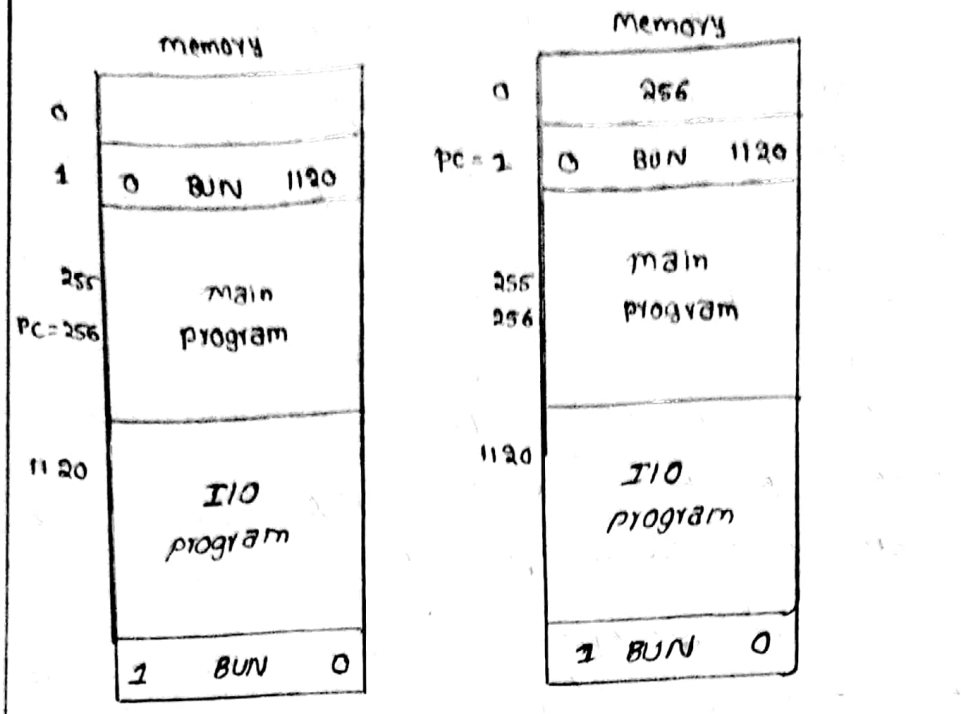
→ The return address available in PC is stored in specific location.

→ That location may be a processor register, a memory stack, or a specific memory location.

→ The example in next page, shows that what happens during the interrupt cycle.

→ From the figure,

* when an interrupt occurs and R is set to 1, while the control is executing the instruction at address 255.



(a) before interrupts (b) after interrupt cycle.

* At this time, the return address 256 is in PC.
* The programmer has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1.
* The programmer has a sign that, when control reaches timming signal T0 and finds R=1, it proceeds with the interrupt cycle.
* The contents of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0.
* The branch instruction at address 1 causes the program to transfer input-output service program at 1120.
* This program checks the flags, determines which flag is set, and transfers required input.
* This is shown in figure (b).

Complete Computer Description :-

→ The final flowchart of instruction cycle, including the interrupt cycle for basic computer. The interrupt flip-flop R may be set at any time during indirect execute phase.

→ The control returns the timming signal T_0 after S_C is cleared to 0. If R=1, the computer goes through an interrupt or instruction cycle.

→ If the instruction is one of the memory-Reference instructions, the computer first checks if there is an address (indirect) and then continues to execute the decoded instruction.

→ If the instruction is one of the Register-Reference instructions, it is executed with one of microoperations listed.

→ Instead of using a flowchart, we can describe the operation of the computer with a list of register transfer statements. This is done by accumulating all the control functions and micro operations.

→ The control functions and the microoperations for the entire computer are summerized. The register transfer statements will describe in a concise from the internal organisation of basic computer.

→ The control functions and conditional contol stakments listed in table format and formulate the boolean functions for the gates in control unit.

→ A register transfer language is useful not only for describing internal organisation, but also specifying the logic circuit needed for design.

Design of Accumulator Logic :-

→ The circuits associated with the AC register are shown in figure. The adder and logic circuit has 3 inputs.

→ One set of 16 inputs comes from the outputs of AC. Another set of 16 input comes from data register DR.

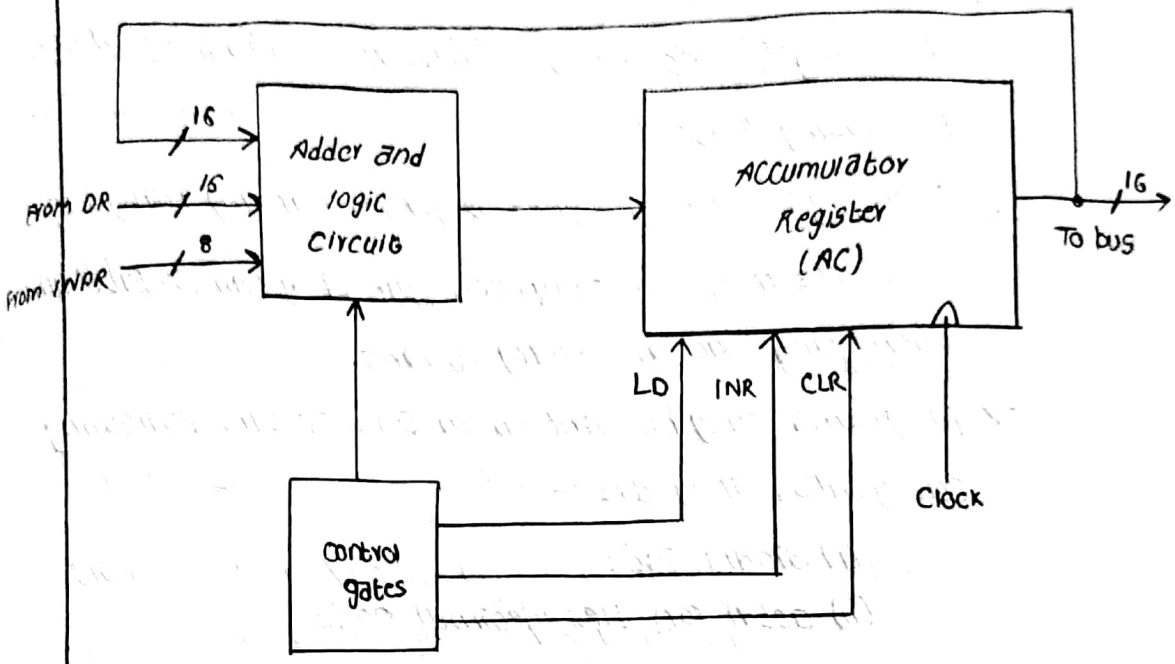
→ A third set of eight inputs comes from the input register INPR. The outputs of the adder and logic circuit provide the data input for register.

→ In addition of this, it is necessary to include logic gates for controlling LD, INR, and CLR in register.

→ Inorder to design the logic associated with AC, it is necessary to go over the register transfer statements.

D_0T_5	:	$AC \leftarrow AC \wedge DR$	AND with DR
D_1T_5	:	$AC \leftarrow AC + DR$	Add with DR
D_2T_5	:	$AC \leftarrow DR$	Transfer from DR
pB_{11}	:	$AC(0\text{-}7) \leftarrow INPR$	Transfer from INPR
rB_9	:	$AC \leftarrow \overline{AC}$	Complement
rB_7	:	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E$	Shift right
rB_6	:	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E$	Shift left
rB_{11}	:	$AC \leftarrow 0$	Clear
rB_5	:	$AC \leftarrow AC + 1$	increment

→ From the above list, we can devieve the control logic gates and the adder and logic circuit.



circuit associated with AC.

* last page

* Programming the basic computer :-

Introduction :-

→ A total computer system is an inclusion of both the hardware and software.

→ Hardware consists of physical components and an associated equipment.

→ Software refers to the program that are written from the computer.

→ This part will introduces some elementary programming concepts and shows their relation to the hardware representation of instructions.

→ The usefulness of various machine instructions is then demonstrated by means of several basic programming examples.

Machine language :-

→ A program is a list of instructions (or) statements for directing the computer to perform a required data-processing task.

→ There are various types of programming languages that are used for computer. But they can execute only when they are in binary format.

→ programs may be written in one of the following categories. They are :-

(i) Binary code
(ii) octal (or) Hexadecimal code
(iii) Symbolic code
(iv) High-level programming languages.

Definition :-

→ A machine language program is a binary program of category, because of simple equivalency between binary (or) octal (or) hexadecimal representation.

→ The hardware of a computer will execute these instructions and perform the given task.

→ Whatever it, the computer knowledge of hardware recognises only this type of instruction code.

Binary to Add two numbers

location	Instruction code
0	0010 0000 0000 0100
1	0001 0000 0000 0101
10	0011 0000 0000 0110
101	1111 1111 1110 1001

Assembly language :-

→ A programming language is defined as by set of rules. Users must conform with all format language rules.

→ Almost every commercial computer has its own particular assembler.

→ The basic unit of an assembly language program is a line of code. The specific language is defined by set of rules.

→ We will now formulate the rules of an assembly language for writing symbolic program for computers.

Rules of language :-

* The label feild may be empty (or) it may specify a symbolic address.
* The instruction feild specifies a machine instruction (or) a pseudo instruction.
* The comment feild may be empty (or) include a comment

Symbolic address :-

→ A symbolic address consists of one (or) more but not more than three alphanumeric characteristics.

→ The first character must be letter and remaining two may be any.

→ The symbol can be choosen arbitrary by the programmer.

* Rules :- It must have :-

1) A memory-reference instruction (MRI).
2) A Register-reference / input-output instruction (non-MRI)
3) A pseudo instruction with / without an operand.

Pseudo instruction :-

→ A pseudo instruction is not a machine instruction but rather than an instruction to the assembler giving information about some phase of translation.

→ The ORG (origin) pseudocode informs the assembler that the instruction or operand in the following line is to placed in memory location.

Symbol	information for Assembler
ORG N	Hexadecimal number N is the memory location for the instruction.
END	Denotes end of symbolic program.
DEC N	signed decimal number N to convert into binary
HEX N	Hexadecimal number N to convert to binary.

The Assembler :-

→ An Assembler is a program that accepts a symbolic language program and produces its binary machine equivalent language.

→ The assembler is a program that operates on the character strings and produces an equivalent binary interpretation.

→ The input symbolic program is called the source program and binary program is called the object program.

program Loops :-

→ A program loop is a sequence of instructions that are executed many times, each time with different set of data.

→ program loops are specified by Fortran by a DO statement.

Eg:-
```
DIMENSION A(100)
INTEGER SUM, A
SUM=0
DO 3 J=1, 100
3 SUM = SUM+A(J).
```

compiler :- A system program that translates a program written in a high-level programming language such as a above to a machine language program is called as a "compiler".

pointer and counter:-

→ The pointer and a counter together with the indirect address operation to form a program loop.

→ The pointer points to the address of current operand and the counter counts the number of times that the loop is executed.

→ In computers with more than one processor Register, it is possible to use one processor Register as a pointer and another processor Register as a counter and third as an accumulator.

→ when processor Registers are used as pointers and the counters, they are called as index Registers.

Line			
1.		ORG 100	Origin of program is Hex 100
2.		LDA ADS	load first address of operand.
3.		STA PTR	Store to pointer
4.		LDA NBR	//load minus 100/
5.		STA CTR	Store in counter
6.		CLA	Clear in Accumulator
7.	LOP	ADD PTR 1	//Add an operand to Ac/
8.		ISZ PTR	increment pointer
9.		ISZ CTR	increment counter
10.		BLN LOP	Repeat loop again.
11.		STA SUM	Store sum
12.		HLT	Halt
13.	ADS,	HEX 150	First address of operands
14	PTR,	HEX 0	location reserve for pointer
15	NBR,	DEC -100	Const to intialize counter
16	CTR,	HEX 0	location for counter
17	SUM	HEX 0	Sum is stored
18		ORG 150	Origin of operand is HEX 150
19		DEC 75	First operand
.			
.			
.			
118		DEC 23	//Last operand
119		END	/END

programming Arithematic and logic operations :-

→ The number of instructions available in a computer may be a few hundread in a large system or a few dozen in small.

→ Some computer performs given operation with one machine instruction, But Others may require a large number of machine instruction to perform same operation.

→ As an illussion, consider four basic arithematic operations. Some computers have machine instructions to add, Subtract, multiply and divide.

→ operations not included in the set of machine instructions must be implemented by a program.

→ Operations that are implemented in a computer with one computer machine and implemented by hardware.

→ Operations that are impremented by set of instruction that consitute a program are implemented by software.

→ Hardware implementation is more costly beacuse of the additional circuits.

→ Software implementation results in long programs both in number of instructions and execution time too.

→ This section demonstrates the software implementation of a few arithematic and logic operations.

→ programs can be developed for any arithematic operation and not only for fixed binary data, but also for decimal and floating point data as well.

Multiplication program :-

→ The program for multiplying two numbers is based on the procedure we use to multiply with pen & paper.

→ The flowchart shows, the multiplication process consists of checking the bits of the multiplier Y and adding the multiplicand X as many times as there 1's in Y, provide the value of X is shifted left from one line to next line.

→ Since the computer can add only two numbers at a time, we reserve a memory location, denoted by P, to store intermediate sums.

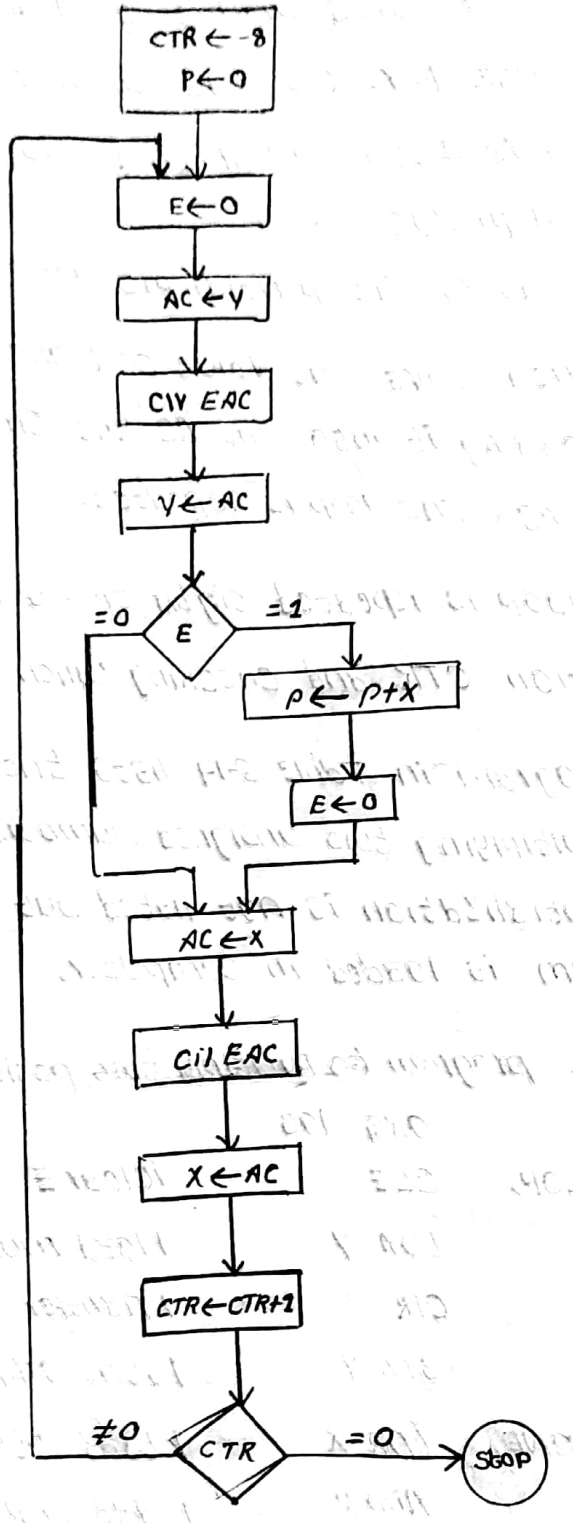
→ The intermediate sums are called partial products, since they hold a partial products until all numbers are added.

→ As shown in the numerical example under P, the partial products starts with zero.

→ The multiplicand X is added to the content of P for each bit of multiplier Y.

→ The final value in P forms the product. The numerical examples has numbers with four significant bits.

→ When multiplied, the product contains 8 specific/significant bits. The computer can use numbers with 8 significant bits to produce upto 16 bits.



Flow chart for multiplication program

→ The above flowchart shows the algorithm for programming of multiplication operation. The program has a loop that is traversed eigth times.

→ The multiplier bit can be checked if it is transferred to E register. This is done by clearing E, loading the values.

→ The bit is stored in E is the low-ordered bit of the multiplier. We now check the value of E.

→ If it is 1, the multiplicand X is added to the partial product P.

→ If it is 0, the partial product does not change.

→ We then shift the value of X once to the left by loading it into the AC and circulating left E and AC. The loop is repeated.

→ The loop is repeated eight times by incrementing location CTR and checking when it reaches zero.

→ The program in Table 8-14 lists the instructions for multiplying two unsigned numbers.

→ The intialization is not listed but included when program is loaded in computer.

program to multiply two positive numbers

```
        ORG  100
LOP,    CLE        /Clear E
        LDA Y      /load multiplier
        CIR        /Transfer multiplier bit to E
        STA Y      /Store shifted multiplier.
ONE,    LDA X      / load multiplicand
        ADD P      / add to partial product
        STA P      /Store partial product
        CLE        /clear E.
ZRO,    LDA X      /load multiplicand
        CIL        / Shift left
        STA X      / Store shifted multiplicand
        ISZ CTR    / Increment counter
```

```
        BUN LOP         / counter not zero ; repeat loop
        HLT             / counter is zero ; halt
CTR,    DEC-8           / these location takes as counter
X,      HEX 000F        / multiplicand stored here
Y,      HEX 000B        / multiplier stored here
P,      HEX 0           / product formed here
        END
```

→ This example has shown that if a computer does not have a machine instruction required.

→ Thus we have demonstrated the software implementation of multiplication operation.

→ The corresponding hardware implementation is present in sec. 10-3.

Double precision Addition :-

→ when two 16-bit unsigned numbers are multiplied, the result is a 32-bit product, that must be stored in two memory words.

→ A number which is stored in two memory words is said to have double precision.

→ when a partial product is computed, it is necessary that a double precision number be added to multiplicand.

→ For grater accuracy, the programmer may wish to employ double-precision numbers and perform arithemeh with operands that occupy two memory words.

→ now we develop a program that is used to add two-double precision numbers.

Program to add two double precision number

```
        LDA  AL      / Load A low
        ADD  BL      / Add B low, carry in E
        STA  CL      / Store C low
        CLA          / Clear AC
        CIL          / Circulate left
        ADD  AH      / Add A high and carry
        ADD  BH      / Add B high and carry
        STA  CH      / Store C high
        HLT

AL,     —            / Location of operands
AH,     —
BL,     —
BH,     —
CL,     —
CH,     —
```

Logic Operation :-

→ The computer has three machine instructions that perform logic operations : AND, CMA and CLA.

→ The LDA instruction may be considered as an logic operation that transfers a logic operand into AC.

→ All 16 micro operations can be implemented by the software means of because any logic function can be implemented using AND and its compliment.

* For example, the OR operation is not available as a machine instruction in basic computer.

→ From Demorgan's theorem we know

$$x+y = (x'y')'$$

→ A program that forms the OR operation of two logic operands A and B is as

```
LDA  A     load first operand A
CMA        complement to get Ā.
STA  TMP   store in temperory location.
LDA  B     load secondary operand B.
CMA        complement to get B̄
AND  TMP   AND with Ā to get Ā∧B̄
CMA        complement again to get A∨B.
```

→ The logic operations can be implement in similar fashion by software.

Shift operation :-

→ The circular shift operations are machine instructions in basic computer.

→ The other shifts of interests are logic shifts and the arithematic shifts.

→ These can be programmed by a small number of the instructions.

→ The logical shifts requires zeros to be added at the extreme positions. For that we need two operations,

CLE and CIR.

→ For a logic shift left operation, we need

CLE and CIL.

→ The arithematic shifts depends on the type or representation of negative numbers.

→ For the basic computer we have adopted the signed 2's complement representation.

→ The rules for the arithematic shifted are listed in above table.

→ For an arithematic right shift it is necessary that the sign bit in left most position remains unchanged.

→ But the sign bit itself shifted into high-order bit position of number.

→ The program for arithematic right-shift requires that we set E to same value as sign bit.

```
CLE        /clear E to 0
SPA        /skip if AC is positive ;
             E remains 0
CME        / AC is-negative , set E to 1.
CIR        /Circulate E and AC.
```

Stay page (cont-)

* **Control of AC Register :-**

→ The gate structure of the control, that controls the LD, INR, and CLR inputs of AC as shown.

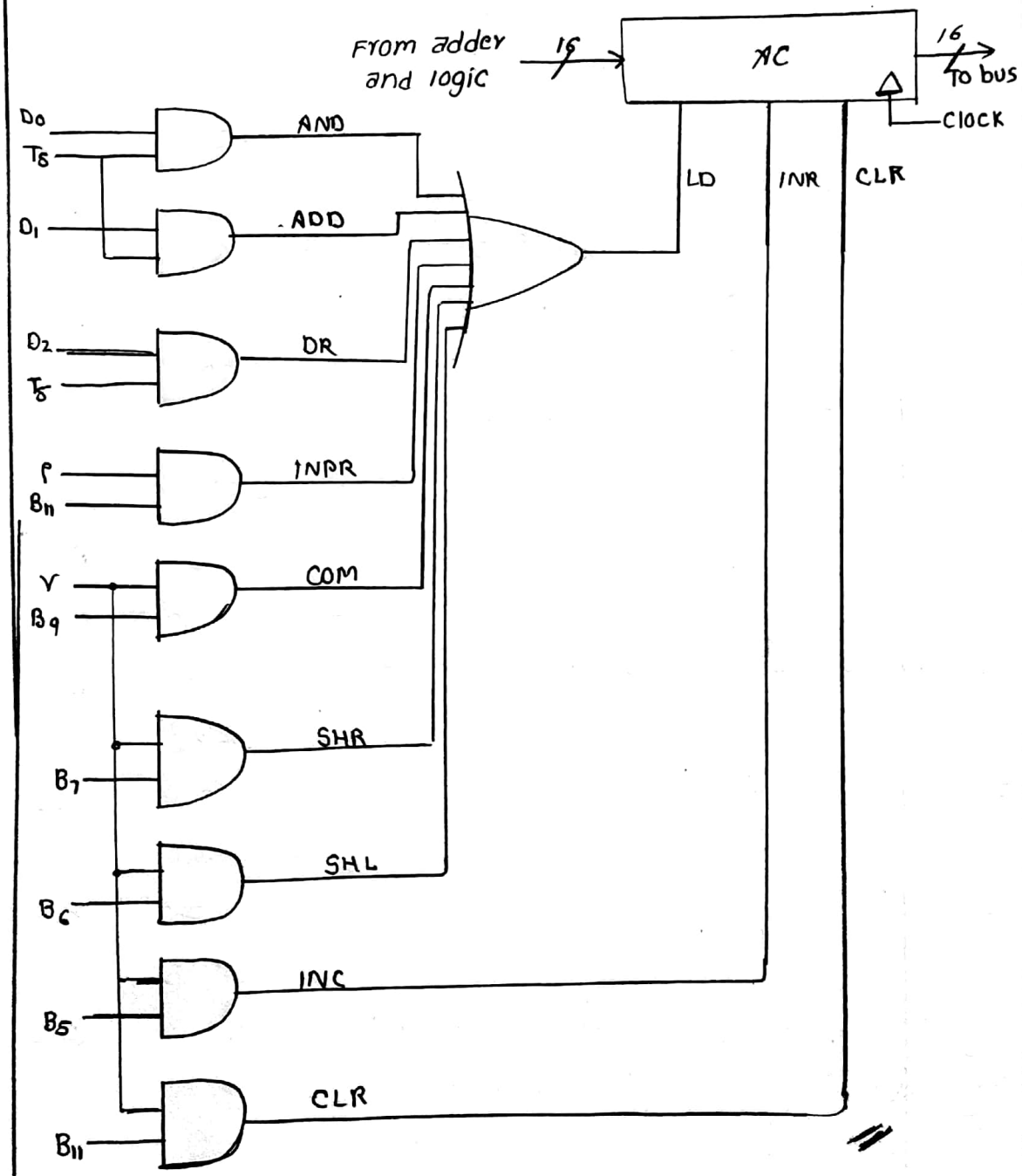
→ The gate configuration is derieved from the control functions in list above.

→ The control function for micro operation clear is rB_{11}, where $r = D_7I'T_3$ and $B_{11} = IR(11)$.

→ The output of AND gate that generates this control function is connected to CLR input of Register.

→ Similarly the output of the gate function that implements the increment microoperation is connected to the INR input of register.

→ The other seven microoperations are generated in the adder and logic circuit and are loaded into AC in a proper time.



PROBLEMS

5-1. A computer uses a memory unit with 256K words of 32 bits each. A binary instruction code is stored in one word of memory. The instruction has four parts: an indirect bit, an operation code, a register code part to specify one of 64 registers, and an address part.

a. How many bits are there in the operation code, the register code part, and the address part?

b. Draw the instruction word format and indicate the number of bits in each part.

c. How many bits are there in the data and address inputs of the memory?

5-2. What is the difference between a direct and an indirect address instruction? How many references to memory are needed for each type of instruction to bring an operand into a processor register?

5-3. The following control inputs are active in the bus system shown in Fig. 5-4. For each case, specify the register transfer that will be executed during the next clock transition.

	S_2	S_1	S_0	LD of register	Memory	Adder
a.	1	1	1	*IR*	Read	—
b.	1	1	0	*PC*	—	—
c.	1	0	0	*DR*	Write	—
d.	0	0	0	*AC*	—	Add

5-4. The following register transfers are to be executed in the system of Fig. 5-4. For each transfer, specify: (1) the binary value that must be applied to bus select inputs S_2, S_1, and S_0; (2) the register whose LD control input must be active (if any); (3) a memory read or write operation (if needed); and (4) the operation in the adder and logic circuit (if any).

a. $AR \leftarrow PC$

b. $IR \leftarrow M[AR]$

c. $M[AR] \leftarrow TR$

d. $AC \leftarrow DR$, $DR \leftarrow AC$ (done simultaneously)

5-5. Explain why each of the following microoperations cannot be executed

Scanned by CamScanner

during a single clock pulse in the system shown in Fig. 5-4. Specify a sequence of microoperations that will perform the operation.

a. $IR \leftarrow M[PC]$
b. $AC \leftarrow AC + TR$
c. $DR \leftarrow DR + AC$ (AC does not change)

5-6. Consider the instruction formats of the basic computer shown in Fig. 5-5 and the list of instructions given in Table 5-2. For each of the following 16-bit instructions, give the equivalent four-digit hexadecimal code and explain in your own words what it is that the instruction is going to perform.

a. 0001 0000 0010 0100
b. 1011 0001 0010 0100
c. 0111 0000 0010 0000

5-7. What are the two instructions needed in the basic computer in order to set the E flip-flop to 1?

5-8. Draw a timing diagram similar to Fig. 5-7 assuming that SC is cleared to 0 at time T_3 if control signal C_7 is active.

$$C_7T_3: \quad SC \leftarrow 0$$

C_7 is activated with the positive clock transition associated with T_1.

5-9. The content of AC in the basic computer is hexadecimal A937 and the initial value of E is 1. Determine the contents of AC, E, PC, AR, and IR in hexadecimal after the execution of the CLA instruction. Repeat 11 more times, starting from each one of the register-reference instructions. The initial value of PC is hexadecimal 021.

5-10. An instruction at address 021 in the basic computer has $I = 0$, an operation code of the AND instruction, and an address part equal to 083 (all numbers are in hexadecimal). The memory word at address 083 contains the operand B8F2 and the content of AC is A937. Go over the instruction cycle and determine the contents of the following registers at the end of the execute phase: PC, AR, DR, AC, and IR. Repeat the problem six more times starting with an operation code of another memory-reference instruction.

5-11. Show the contents in hexadecimal of registers PC, AR, DR, IR, and SC of the basic computer when an ISZ indirect instruction is fetched from memory and executed. The initial content of PC is 7FF. The content of memory at address 7FF is EA9F. The content of memory at address A9F is 0C35. The content of memory at address C35 is FFFF. Give the answer in a table with five columns, one for each register and a row for each timing signal. Show the contents of the registers after the positive transition of each clock pulse.

5-12. The content of PC in the basic computer is 3AF (all numbers are in hexadecimal). The content of AC is 7EC3. The content of memory at address 3AF is 932E. The content of memory at address 32E is 09AC. The content of memory at address 9AC is 8B9F.

a. What is the instruction that will be fetched and executed next?
b. Show the binary operation that will be performed in the AC when the instruction is executed.

Scanned by CamScanner

c. Give the contents of registers *PC*, *AR*, *DR*, *AC*, and *IR* in hexadecimal and the values of *E*, *I*, and the sequence counter *SC* in binary at the end of the instruction cycle.

5-13. Assume that the first six memory-reference instructions in the basic computer listed in Table 5-4 are to be changed to the instructions specified in the following table. *EA* is the effective address that resides in *AR* during time T_4. Assume that the adder and logic circuit in Fig. 5-4 can perform the exclusive-OR operation $AC \leftarrow AC \oplus DR$. Assume further that the adder and logic circuit cannot perform subtraction directly. The subtraction must be done using the 2's complement of the subtrahend by complementing and incrementing *AC*. Give the sequence of register transfer statements needed to execute each of the listed instructions starting from timing T_4. Note that the value in *AC* should not change unless the instruction specifies a change in its content. You can use *TR* to store the content of *AC* temporary or you can exchange *DR* and *AC*.

Symbol	Opcode	Symbolic designation	Description in words
XOR	000	$AC \leftarrow AC \oplus M[EA]$	Exclusive-OR to *AC*
ADM	001	$M[EA] \leftarrow M[EA] + AC$	Add *AC* to memory
SUB	010	$AC \leftarrow AC - M[EA]$	Subtract memory from *AC*
XCH	011	$AC \leftarrow M[EA]$, $M[EA] \leftarrow AC$	Exchange *AC* and memory
SEQ	100	If $(M[EA] = AC)$ then $(PC \leftarrow PC + 1)$	Skip on equal
BPA	101	If $(AC > 0)$ then $(PC \leftarrow EA)$	Branch if *AC* positive and non-zero

5-14. Make the following changes to the basic computer.

1. Add a register to the bus system CTR (count register) to be selected with $S_2S_1S_0 = 000$.
2. Replace the ISZ instruction with an instruction that loads a number into CTR.

LDC Address $CTR \leftarrow M[Address]$

3. Add a register reference instruction ICSZ: Increment *CTR* and skip next instruction if zero. Discuss the advantage of this change.

5-15. The memory unit of the basic computer shown in Fig. 5-3 is to be changed to a 65,536 × 16 memory, requiring an address of 16 bits. The instruction format of a memory-reference instruction shown in Fig. 5-5(a) remains the same for $I = 1$ (indirect address) with the address part of the instruction residing in positions 0 through 11. But when $I = 0$ (direct address), the address of the instruction is given by the 16 bits in the next word following the instruction. Modify the microoperations during time T_2, T_3, (and T_4 if necessary) to conform with this configuration.

Scanned by CamScanner

5-16. A computer uses a memory of 65,536 words with eight bits in each word. It has the following registers: *PC*, *AR*, *TR* (16 bits each), and *AC*, *DR*, *IR* (eight bits each). A memory-reference instruction consists of three words: an 8-bit operation-code (one word) and a 16-bit address (in the next two words). All operands are eight bits. There is no indirect bit.

a. Draw a block diagram of the computer showing the memory and registers as in Fig. 5-3. (Do not use a common bus).

b. Draw a diagram showing the placement in memory of a typical three-word instruction and the corresponding 8-bit operand.

c. List the sequence of microoperations for fetching a memory reference instruction and then placing the operand in *DR*. Start from timing signal T_0.

5-17. A digital computer has a memory unit with a capacity of 16,384 words, 40 bits per word. The instruction code format consists of six bits for the operation part and 14 bits for the address part (no indirect mode bit). Two instructions are packed in one memory word, and a 40-bit instruction register *IR* is available in the control unit. Formulate a procedure for fetching and executing instructions for this computer.

5-18. An output program resides in memory starting from address 2300. It is executed after the computer recognizes an interrupt when *FGO* becomes a 1 (while *IEN* = 1).

a. What instruction must be placed at address 1?

b. What must be the last two instructions of the output program?

5-19. The register transfer statements for a register *R* and the memory in a computer are as follows (the X's are control functions that occur at random):

$X_3' X_1$:	$R \leftarrow M[AR]$	Read memory word into *R*
$X_1' X_2$:	$R \leftarrow AC$	Transfer *AC* to *R*
$X_1' X_3$:	$M[AR] \leftarrow R$	Write *R* to memory

The memory has data inputs, data outputs, address inputs, and control inputs to read and write as in Fig. 2-12. Draw the hardware implementation of *R* and the memory in block diagram form. Show how the control functions X_1 through X_3 select the load control input of *R*, the select inputs of multiplexers that you include in the diagram, and the read and write inputs of the memory.

5-20. The operations to be performed with a flip-flop *F* (not used in the basic computer) are specified by the following register transfer statements:

xT_3:	$F \leftarrow 1$	Set *F* to 1
yT_1:	$F \leftarrow 0$	Clear *F* to 0
zT_2:	$F \leftarrow \overline{F}$	Complement *F*
wT_5:	$F \leftarrow G$	Transfer value of *G* to *F*

Otherwise, the content of *F* must not change. Draw the logic diagram showing the connections of the gates that form the control functions and the inputs of flip-flop *F*. Use a *JK* flip-flop and minimize the number of gates.

Scanned by CamScanner

5-21. Derive the control gates associated with the program counter *PC* in the basic computer.

5-22. Derive the control gates for the write input of the memory in the basic computer.

5-23. Show the complete logic of the interrupt flip-flops *R* in the basic computer. Use a *JK* flip-flop and minimize the number of gates.

5-24. Derive the Boolean logic expression for x_2 (see Table 5-7). Show that x_2 can be generated with one AND gate and one OR gate.

5-25. Derive the Boolean expression for the gate structure that clears the sequence counter *SC* to 0. Draw the logic diagram of the gates and show how the output is connected to the INR and CLR inputs of *SC* (see Fig. 5-6). Minimize the number of gates.

Scanned by CamScanner

CHAPTER 5

5-1

$256K = 2^8 \times 2^{10} = 2^{18}$

$64 = 2^6$

(a) Address: 18 bits
Register code: 6 bits
Indirect bit: 1 bit
25

32 − 25 = 7 bits for opcode.

(b)

1	7	6	18
I	Opcode	Register	Address

= 32 bits

(c) Data; 32 bits; address: 18 bits.

5-2

A direct address instruction needs two references to memory: (1) Read instruction; (2) Read operand.

An indirect address instruction needs three references to memory: (1) Read instruction; (2) Read effective address; (3) Read operand.

5-3

(a) Memory read to bus and load to IR: IR ← M[AR]

(b) TR to bus and load to PC: PC ← TR

(c) AC to bus, write to memory, and load to DR: DR ← AC, M[AR] ← AC

(d) Add DR (or INPR) to AC: AC ← AC + DR

5-4

		(1) $S_2S_1S_0$	(2) Load (LD)	(3) Memory	(4) Adder
(a)	AR ← PC	010 (PC)	AR	—	—
(b)	IR ← M[AR]	111 (M)	IR	Read	—
(c)	M[AR] ← TR	110 (TR)	—	Write	—
(d)	DR ← AC AC ← DR	100 (AC)	DR and AC	—	Transfer DR to AC

Scanned by CamScanner

5-5

(a) IR ← M[PC] PC cannot provide address to memory. Address must be transferred to AR first

AR ← PC
IR ← M[AR]

(b) AC ← AC + TR Add operation must be done with DR. Transfer TR to DR first.

DR ← TR
AC ← AC + DR

(c) DR ← DR : AC Result of addition is transferred to AC (not DR). To save value of AC, its content must be stored temporary in DR (or TR).

AC ← DR, DR ← AC (See answer to Problem 5-4(d))
AC ← AC + DR
AC ← DR, DR ← AC

5-6

(a) 0001 0000 0010 0100 $= (1024)_{16}$
ADD ... $(024)_{16}$
ADD content of M[024] to AC ADD 024

(b) 1011 0001 0010 0100 $= (B124)_{16}$
I STA (124)6
store AC in M[M[124]] STA I 124

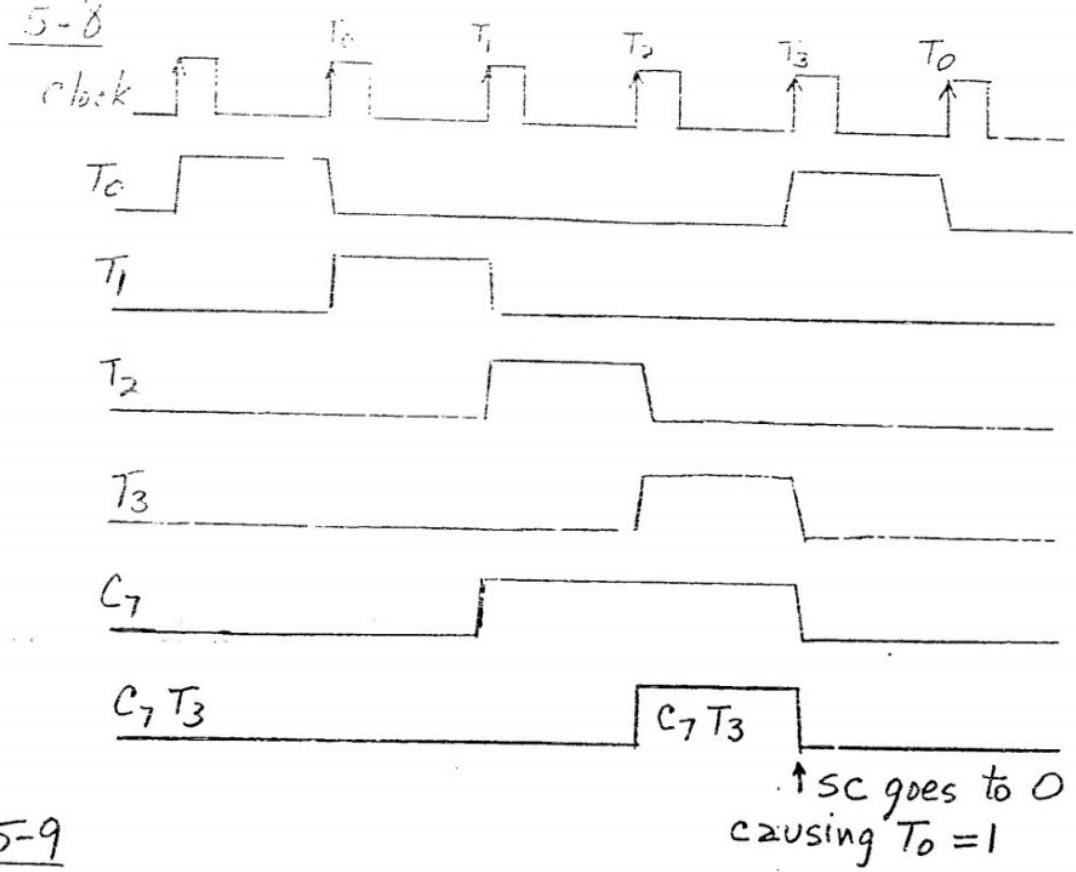
(c) 0111 0000 0010 0000 $= (7020)_{16}$
Register Increment AC INC

5-7

CLE clear E
CME Complement E

Scanned by CamScanner

5-8



5-9

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
CLA	1	0000	022	800	7800
CLE	0	A937	022	400	7400
CMA	1	56C8	022	200	7200
CME	0	A937	022	100	7100
CIR	1	D49B	022	080	7080
CIL	1	526F	022	040	7040
INC	1	A938	022	020	7020
SPA	1	A937	022	010	7010
SNA	1	A937	023	008	7008
SZA	1	A937	022	004	7004
SZE	1	A937	022	002	7002
HLT	1	A937	022	001	7001

Scanned by CamScanner

5-10

	PC	AR	DR	AC	IR
Initial	021	—	—	A937	—
AND	022	083	B8F2	A832	0083
ADD	022	083	B8F2	6229	1083
LDA	022	083	B8F2	B8F2	2083
STA	022	083	—	A937	3083
BUN	083	083	—	A937	4083
BSA	084	084	—	A937	5083
ISZ	022	083	B8F3	A937	6083

5-11

	PC	AR	DR	IR	SC
Initial	7FF	—	—	—	0
T_0	7FF	7FF	—	—	1
T_1	800	7FF	—	EA9F	2
T_2	800	A9F	—	EA9F	3
T_3	800	C35	—	EA9F	4
T_4	800	C35	FFFF	EA9F	5
T_5	800	C35	0000	EA9F	6
T_6	801	C35	0000	EA9F	0

5-12

(a) $9 = (1001)_2$

I=1 1,001 ADD ADD I 32E

Memory

3AF	932E
32E	09AC
9AC	8B9F

AC = 7EC3

(b)

AC=7EC3 (ADD)
DR=8B9F
0A62

E=1

(c) PC = 3AF+1 = 3B0 IR = 932E
AR = 9AC E = 1
DR = 8B9F I = 1
AC = 0A62 SC = 0000

Scanned by CamScanner

5-13

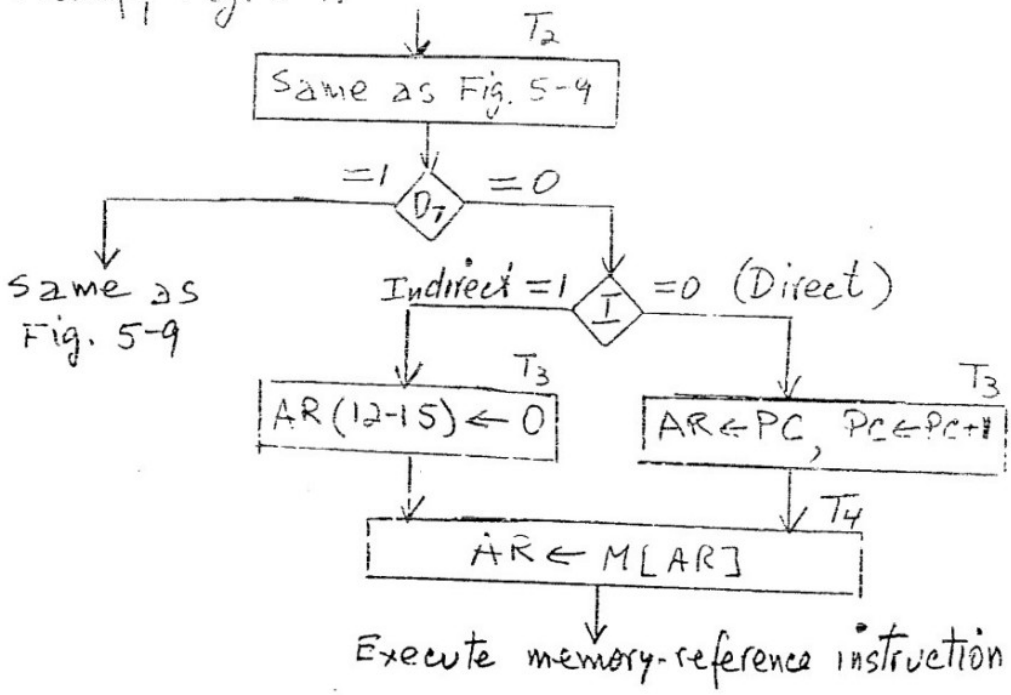
XOR	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \oplus DR$, $SC \leftarrow 0$
ADM	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$DR \leftarrow AC$, $AC \leftarrow AC + DR$
	D_1T_6:	$M[AR] \leftarrow AC$, $AC \leftarrow DR$, $SC \leftarrow 0$
SUB	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$DR \leftarrow AC$, $AC \leftarrow DR$
	D_2T_6:	$AC \leftarrow \overline{AC}$
	D_2T_7:	$AC \leftarrow AC + 1$
	D_2T_8:	$AC \leftarrow AC + DR$, $SC \leftarrow 0$
XCH	D_3T_4:	$DR \leftarrow M[AR]$
	D_3T_5:	$M[AR] \leftarrow AC$, $AC \leftarrow DR$, $SC \leftarrow 0$
SEQ	D_4T_4:	$DR \leftarrow M[AR]$
	D_4T_5:	$TR \leftarrow AC$, $AC \leftarrow AC \oplus DR$
	D_4T_6:	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$, $AC \leftarrow TR$, $SC \leftarrow 0$
BPA	D_5T_4:	If $(AC = 0 \wedge AC(15) = 0)$ then $(PC \leftarrow AR)$, $SC \leftarrow 0$

5-14

Converts the ISZ instruction from a memory-reference instruction to a register-reference instruction. The new instruction ICSZ can be executed at time T_3 instead of time T_6, a saving of 3 clock cycles.

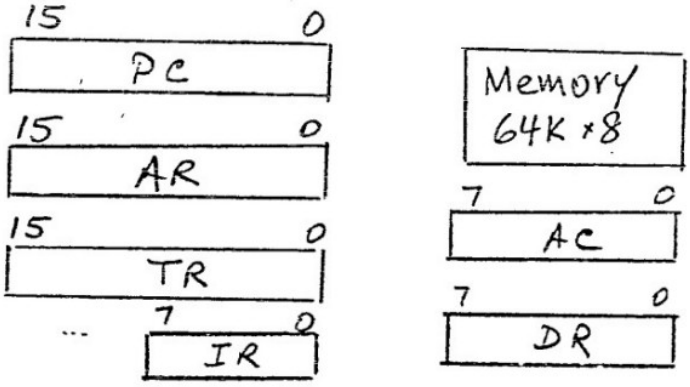
Scanned by CamScanner

5-15 Modify Fig. 5-4:

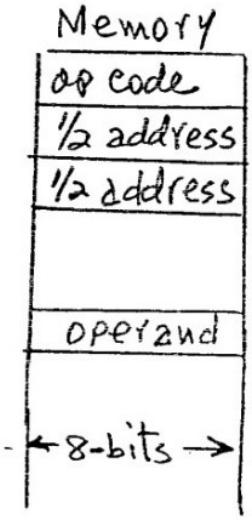


5-16

(a)



(b)



(c) T_0: IR ← M[PC], PC ← PC+1

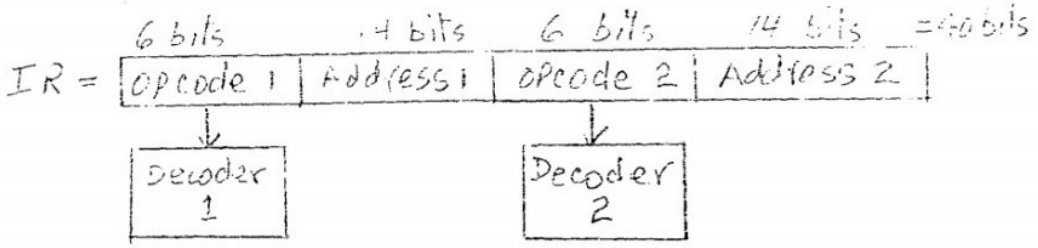
T_1: AR(0-7) ← M[PC], PC ← PC+1

T_2: AR(8-15) ← M[PC], PC ← PC+1

T_3: DR ← M[AR]

Scanned by CamScanner

5-11



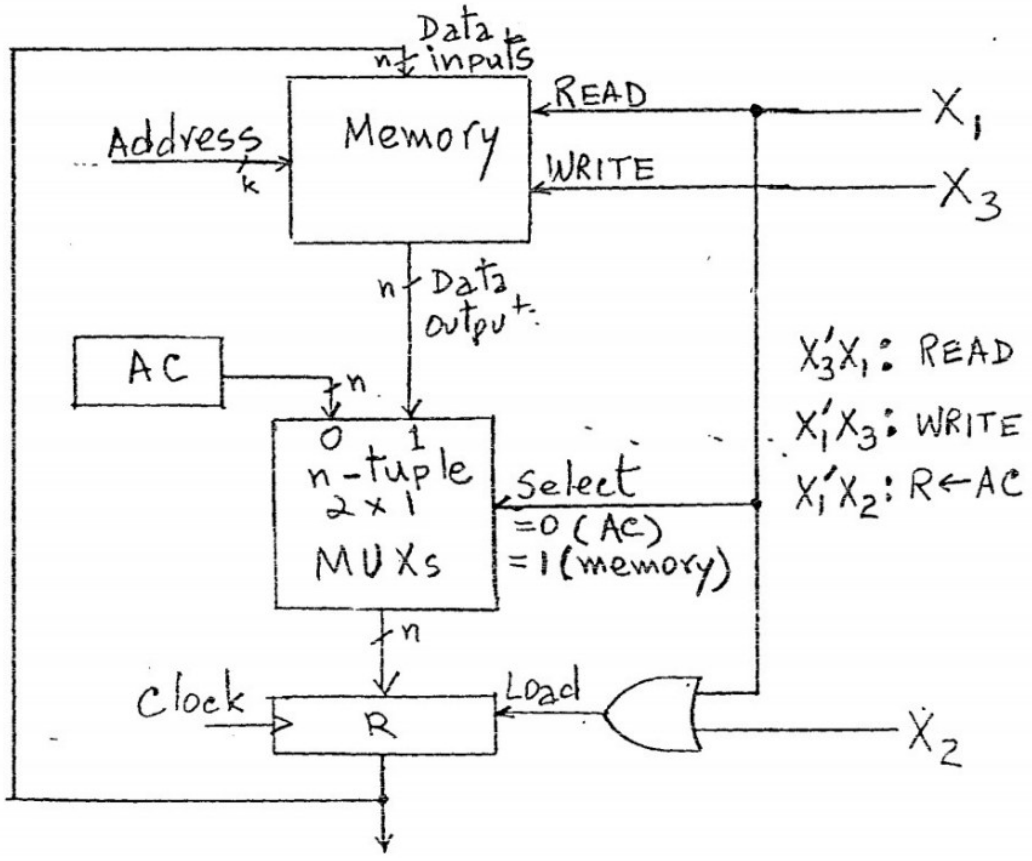
1. Read 40-bit double instruction from memory to IR and then increment PC,
2. Decode opcode 1.
3. Execute instruction 1 using address 1.
4. Decode opcode 2.
5. Execute instruction 2 using address 2.
6. Go back to step 1.

5-18

(a) BUN 2300

(b) ION
BUN 0 I (Branch indirect with address 0)

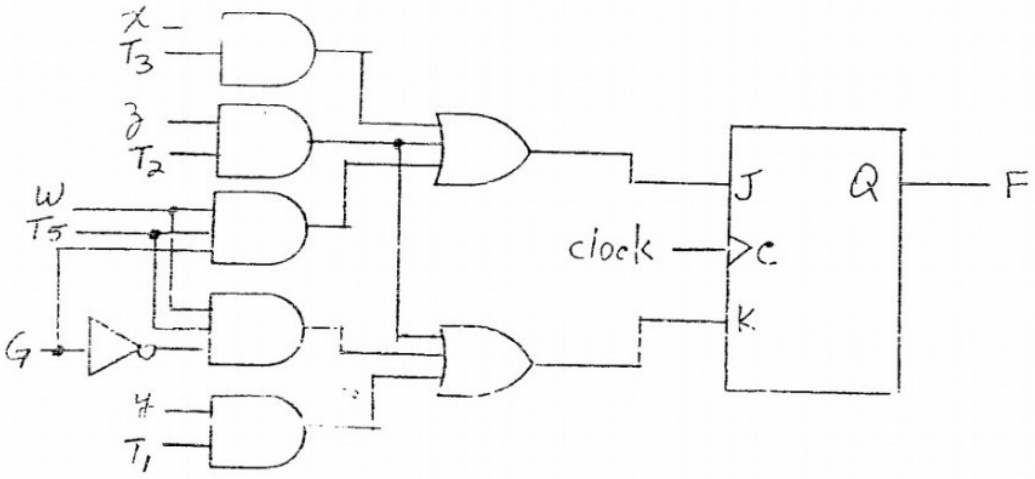
5-19



Scanned by CamScanner

5-20

$J_F = x T_3 + y T_2 + w T_5 G \qquad K_F = y T_1 + y T_2 + w T_5 G'$



5-21 From Table 5-6: ($z_{DR}=1$ if $DR=0$; $z_{AC}=1$ if $AC=0$)

$$INR(PC) = R'T_1 + RT_2 + D_6T_6 z_{DR} + pB_9(FGI) + pB_8(FGO) + rB_4(AC_{15})' + rB_3(AC_{15}) + rB_2 z_{AC} + rB_1 E'$$

$$LD(PC) = D_4T_4 + D_5T_5$$

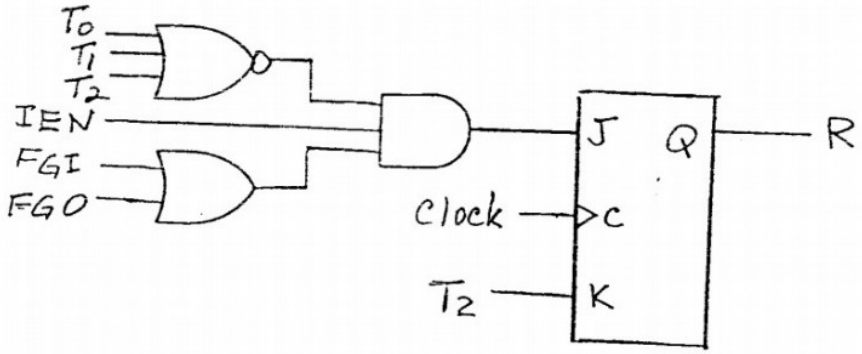
$$CLR(PC) = RT_1$$

The logic diagram is similar to the one in Fig. 5-16.

5-22 $\text{Write} = D_3T_4 + D_5T_4 + D_6T_6 + RT_1$ $\qquad (M[AR] \leftarrow xx)$

5-23 $(T_0+T_1+T_2)'(IEN)(FGI+FGO)$: $R \leftarrow 1$

RT_2 : $R \leftarrow 0$



Scanned by CamScanner

5-24

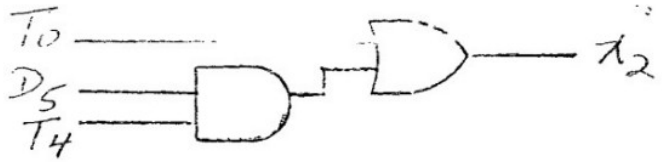
x_2 places PC onto the bus. From Table 5-6:

$R'T_0$: AR ← PC

RT_0 : TR ← PC

D_5T_4 : M[AR] ← PC

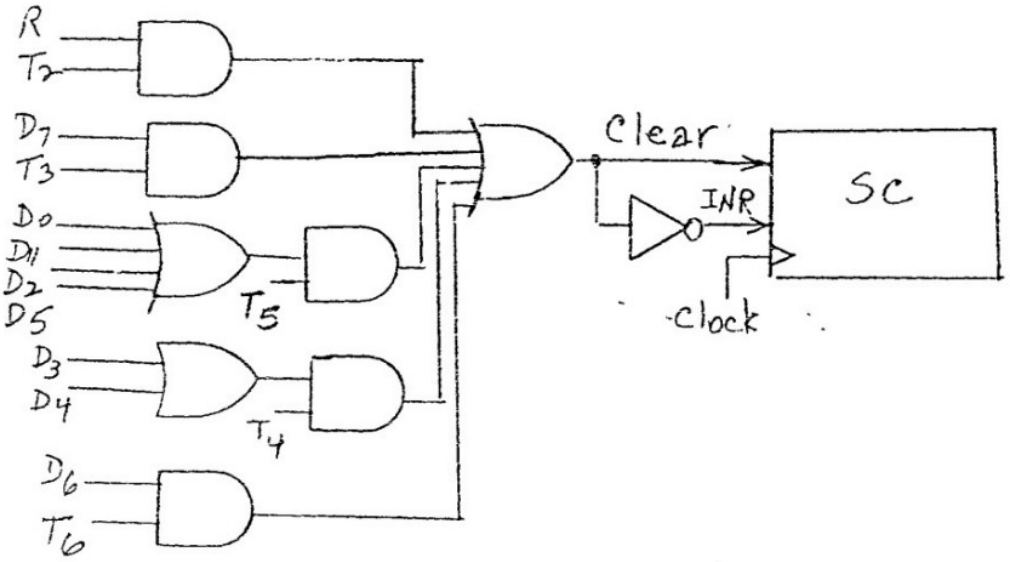
$$x_2 = R'T_0 + RT_0 + D_5T_4 = (R' + R)T_0 + D_5T_4 = T_0 + D_5T_4$$



5-25

From Table 5-6:

$$CLR(SC) = RT_2 + D_7T_3(I' + I) + (D_0 + D_1 + D_2 + D_5)T_5 + (D_3 + D_4)T_4 + D_6T_6$$



Scanned by CamScanner

6-1. The following program is stored in the memory unit of the basic computer. Show the contents of the *AC*, *PC*, and *IR* (in hexadecimal), at the end, after each instruction is executed. All numbers listed below are in hexadecimal.

Location	Instruction
010	CLA
011	ADD 016
012	BUN 014
013	HLT
014	AND 017
015	BUN 013
016	C1A5
017	93C6

6-2. The following program is a list of instructions in hexadecimal code. The computer executes the instructions starting from address 100. What are the content of *AC* and the memory word at address 103 when the computer halts?

Location	Instruction
100	5103
101	7200
102	7001
103	0000
104	7800
105	7020
106	C103

6-3. List the assembly language program (of the equivalent binary instructions) generated by a compiler from the following Fortran program. Assume integer variables.

```
SUM = 0
SUM = SUM + A + B
DIF = DIF - C
SUM = SUM + DIF
```


Scanned by CamScanner

6-3. List the assembly language program (of the equivalent binary instructions) generated by a compiler from the following Fortran program. Assume integer variables.

```
SUM = 0
SUM = SUM + A + B
DIF = DIF - C
SUM = SUM + DIF
```

6-4. Can the letter I be used as a symbolic address in the assembly language program defined for the basic computer? Justify the answer.

6-5. What happens during the first pass of the assembler (Fig. 6-1) if the line of code that has a pseudoinstruction ORG or END also has a label? Modify the flowchart to include an error message if this occurs.

6-6. A line of code in an assembly language program is as follows:

```
DEC -35
```

a. Show that four memory words are required to store the line of code and give their binary content.
b. Show that one memory word stores the binary translated code and give its binary content.

6-7. a. Obtain the address symbol table generated for the program of Table 6-13 during the first pass of the assembler.
b. List the translated program in hexadecimal.

6-8. The pseudoinstruction BSS N (block started by symbol) is sometimes employed to reserve *N* memory words for a group of operands. For example, the line of code

```
A,  BSS 10
```

informs the assembler that a block of 10 (decimal) locations is to be left free, starting from location A. This is similar to the Fortran statement DIMENSION A(10). Modify the flowchart of Fig. 6-1 to process this pseudoinstruction.

Scanned by CamScanner

6-9. Modify the flowchart of Fig. 6-2 to include an error message when a symbolic address is not defined by a label.

6-10. Show how the MRI and non-MRI tables can be stored in memory.

6-11. List the assembly language program (of the equivalent binary instructions) generated by a compiler for the following IF statement:

```
IF(A - B) 10, 20, 30
```

The program branches to statement 10 if $A - B < 0$; to statement 20 if $A - B = 0$; and to statement 30 if $A - B > 0$.

6-12. a. Explain in words what the following program accomplishes when it is executed. What is the value of location CTR when the computer halts?
b. List the address symbol table obtained during the first pass of the assembler.
c. List the hexadecimal code of the translated program.

```
        ORG 100
        CLE
        CLA
        STA CTR
        LDA WRD
        SZA
        BUN ROT
        BUN STP
ROT,    CIL
        SZE
        BUN AGN
        BUN ROT
AGN,    CLE
        ISZ CTR
        SZA
        BUN ROT
STP,    HLT
CTR,    HEX 0
WRD,    HEX 62C1
        END
```

6-13. Write a program loop, using a pointer and a counter, that clears to 0 the contents of hexadecimal locations 500 through 5FF.

6-14. Write a program to multiply two positive numbers by a repeated addition method. For example, to multiply 5 × 4, the program evaluates the product by adding 5 four times, or 5 + 5 + 5 + 5.

6-15. The multiplication program of Table 6-14 is not initialized. After the program is executed once, location CTR will be left with zero. Show that if the program is executed again starting from location 100, the loop will be traversed 65536 times. Add the needed instructions to initialize the program.

Scanned by CamScanner

6-16. Write a program to multiply two unsigned positive numbers, each with 16 significant bits, to produce an unsigned double-precision product.

6-17. Write a program to multiply two signed numbers with negative numbers being initially in signed-2's complement representation. The product should be single-precision and signed-2's complement representation if negative.

6-18. Write a program to subtract two double-precision numbers.

6-19. Write a program that evaluates the logic exclusive-OR of two logic operands.

6-20. Write a program for the arithmetic shift-left operation. Branch to OVF if an overflow occurs.

6-21. Write a subroutine to subtract two numbers. In the calling program, the BSA instruction is followed by the subtrahend and minuend. The difference is returned to the main program in the third location following the BSA instruction.

6-22. Write a subroutine to complement each word in a block of data. In the calling program, the BSA instruction is followed by two parameters: the starting address of the block and the number of words in the block.

6-23. Write a subroutine to circulate *E* and *AC* four times to the right. If *AC* contains hexadecimal 079C and $E = 1$, what are the contents of *AC* and *E* after the subroutine is executed?

6-24. Write a program to accept input characters, pack two characters in one word and store them in consecutive locations in a memory buffer. The first address of the buffer is $(400)_{16}$. The size of the buffer is $(512)_{10}$ words. If the buffer overflows, the computer should halt.

6-25. Write a program to unpack two characters from location WRD and store them in bits 0 through 7 of locations CH1 and CH2. Bits 9 through 15 should contain zeros.

6-26. Obtain a flowchart for a program to check for a CR code (hexadecimal 0D) in a memory buffer. The buffer contains two characters per word. When the code for CR is encountered, the program transfers it to bits 0 through 7 of location LNE without disturbing bits 8 through 15.

6-27. Translate the service routine SRV from Table 6-23 to its equivalent hexadecimal code. Assume that the routine is stored starting from location 200.

6-28. Write an interrupt service routine that performs all the required functions but the input device is serviced only if a special location, MOD, contains all 1's. The output device is serviced only if location MOD contains all 0's.

Scanned by CamScanner

CHAPTER 6

6-1

		AC	PC	IR
010	CLA	0000	011	7800
011	ADD 016	C1A5	012	1016
012	BUN 014	C1A5	014	4014
013	HLT	8184	014	7001
014	AND 017	8184	015	0017
015	BUN 013	8184	013	4013
016	C1A5			
017	93C6			

$(C1A5)_{16} = 1100\ 0001\ 1010\ 0101$ AND

$(93C6)_{16} = 1001\ 0011\ 1100\ 0110$

$1000\ 0001\ 1000\ 0100 = (8184)_{16}$

6-2

			AC
100	5103	BSA 103	
101	7200	CMA	FFFE ← Answer
102	7001	HLT	
103	0000	5101 ← Answer	
104	7800	CLA	0000
105	7020	INC	0001
106	C103	BUN 103 I	

6-3

```
CLA
STA SUM      } SUM = 0

LDA SUM
ADD A
ADD B        } SUM = SUM + A + B
STA SUM

LDA C
CMA
INC          } DIF = DIF - C
ADD DIF
STA DIF

LDA SUM
ADD DIF      } SUM = SUM + DIF
STA SUM
```

A more efficient compiler will optimize the machine code as follows:

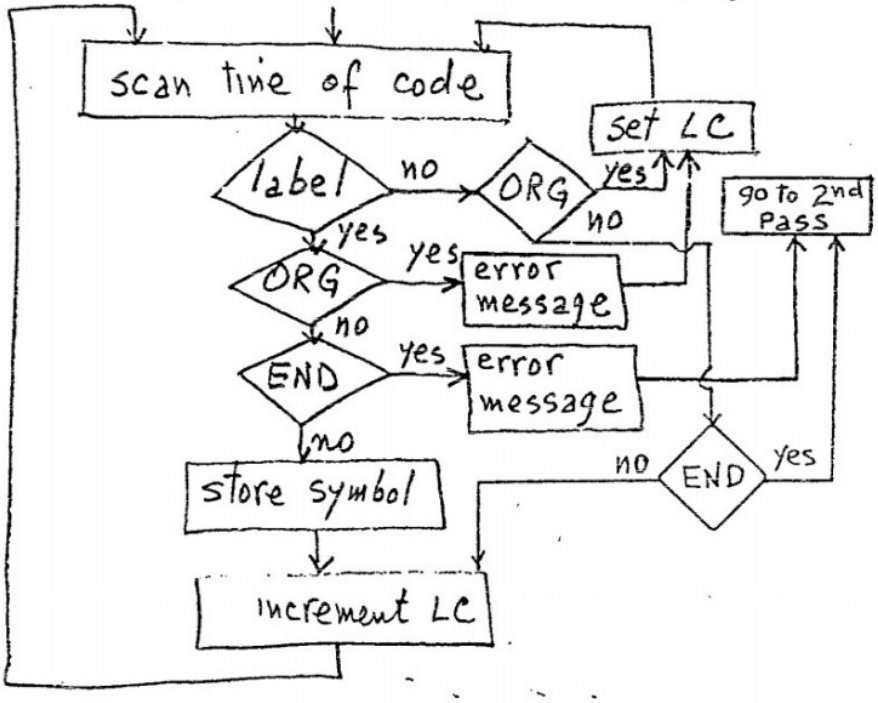
```
LDA A
ADD B
STA SUM
LDA C
CMA
INC
ADD DIF
STA DIF
ADD SUM
STA SUM
```


Scanned by CamScanner

6-4 A line of code such as: LDA I is interpreted by the assembler (Fig. 6-2) as a two symbol field with I as the symbolic address. A line of code such as: LDA I I is interpreted as a three symbol field. The first I is an address symbol and the second I as the Indirect bit.
Answer: Yes, it can be used for this assembler.

6-5

The assembler will not detect an ORG or END if the line has a label; according to the flow chart of Fig. 6-1. Such a label has no meaning and constitutes an error. To detect the error, modify the flow chart of Fig. 6-1:



6-6 (a)

memory word	characters	Hex	binary
1	D E	44 45	0100 0100 0100 0101
2	C space	43 20	0100 0011 0010 0000
3	- 3	2D 33	0010 1101 0011 0011
4	5 CR	35 0D	0011 0101 0000 1101

(b) $(35)_{10} = (0000\ 0000\ 0010\ 0011)_2$

$-35 \rightarrow 1111\ 1111\ 1101\ 1101 = (FF\ DD)_{16}$

Scanned by CamScanner

6-7 (a)

	LOP	105
	ADS	10B
	PTR	10C
	NBR	10D
	CTR	10E
	SUM	10F

$(100)_{10} = (0000\ 0000\ 0110\ 0100)_2$

$(-100)_{10} = (1111\ 1111\ 1001\ 1100)_2 = (FF9C)_{16}$

$(75)_{10} = (0000\ 0000\ 0100\ 1011)_2 = (0048)_{16}$

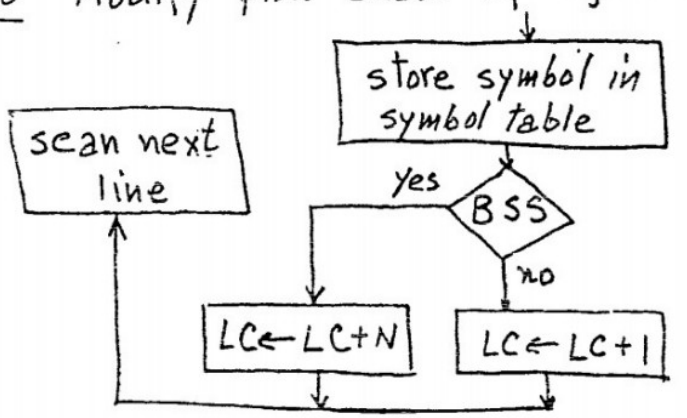
$(23)_{10} = (0000\ 0000\ 0001\ 0111)_2 = (0017)_{16}$

(b)

Loc	Hex			
			ORG	100
100	210B		LDA	ADS
101	310C		STA	PTR
102	210D		LDA	NBR
103	310E		STA	CTR
104	7800		CLA	
105	910C	LOP,	ADD	PTR I
106	610C		ISZ	PTR
107	610E		ISZ	CTR
108	4105		BUN	LOP
109	310F		STA	SUM
10A	7001		HLT	

Loc	Hex		
10B	0150	ADS,	HEX 150
10C	0000	PTR,	HEX 0
10D	FF9C	NBR,	DEC -100
10E	0000	CTR,	HEX 0
10F	0000	SUM,	HEX 0
			ORG 150
150	0048		DEC 150
⋮	⋮		⋮
1B3	0017		DEC 23
			END

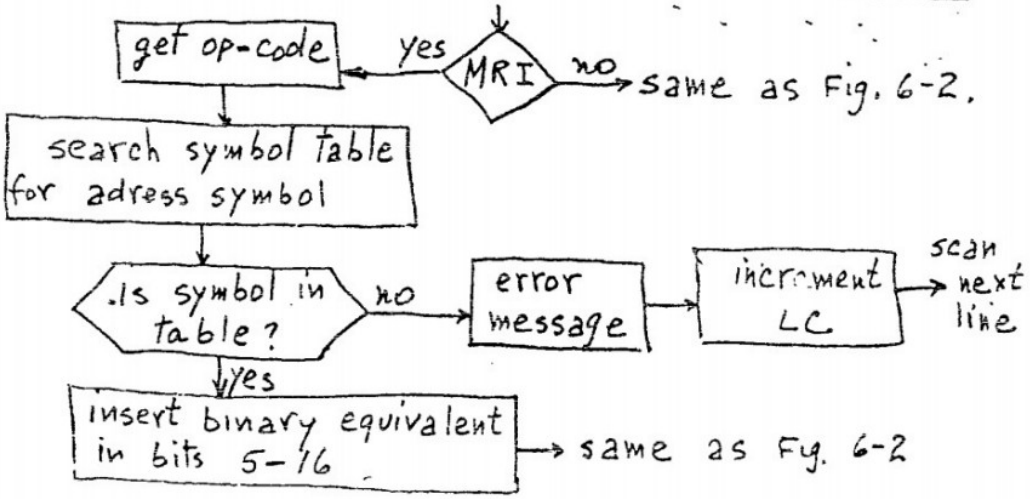
6-8 Modify flow chart of Fig. 6-1




```
LC=1    A, Bss 10
          2   A+1
          3   A+2
          4   A+3
          5   A+4
          6   A+5
          7   A+6
          8   A+7
          9   A+8
         10   A+9
->       11   LC set to 11
```

example

6-9



Scanned by CamScanner

6-10 (a) MRI table →

	memory word	symbol	HEX
AND	1	A N	41 4D
	2	D space	44 20
	3	value	00 00
ADD	4	A D	41 44
	5	D space	44 20
	6	value	1 0 00

etc.

(b) non-MRI table →

	word	symbol	HEX
CLA	1	C L	43 4C
	2	A space	41 20
	3	value	78 00
CLE	4	C L	43 4C
	5	E space	45 20
	6	value	74 00

etc.

6-11

```
LDA  B
CMA
INC
ADD  A     / Form A-B
SPA        / skip if AC positive
BUN  N10   /(A-B)<0, go to N10
SZA        / skip if AC=0
BUN  N30   /(A-B)>0, go to N30
BUN  N20   / (A-B)=0, go to N20
```

6-12 (a) The program counts the number of 1's in the number stored in location WRD. Since WRD = $(62C1)_{16}$ = ... $(0110\ 0010\ 1100\ 0001)_2$
number of 1's is 6; so CTR will have $(0006)_{16}$

(b)

```
                ORG 100
100  7400       CLE
101  7800       CLA
102  3110       STA CTR   /Initialize counter to zero
103  2111       LDA WRD
104  7004       SZA
105  4107       BUN ROT
106  410F       BUN STP   / Word is zero; stop with CTR=0
107  7040  ROT, CIL       / Bring bit to E
108  7002       SZE
109  410B       BUN AGN   / bit=1, go to count it
10A  4107       BUN ROT   / bit=0, repeat
10B  7400  AGN, CLE
10C  6110       ISZ CTR   /Increment counter
```


Scanned by CamScanner

6-12 (b) Continued

```
10D   7004      SZA      /check if remaining bits = 0
10E   4107      BUN ROT  / No; rotate again
10F   7001  STP, HLT      / Yes; stop
110   0000  CTR, HEX 0
111   62C1  WRD, HEX 62C1
            END
```

6-13 $(100)_{16} = (256)_{10}$ 500 to 5FF $\rightarrow (256)_{10}$ locations

```
      ORG  100
      LDA  ADS
      STA  PTR     / Initialize pointer
      LDA  NBR
      STA  CTR     / Initialize counter to -256
      CLA
LOP,  STA  PTR I   / store zero
      ISZ  PTR
      ISZ  CTR
      BUN  LOP
      HLT
ADS,  HEX  500
PTR,  HEX  0
NBR,  DEC  -256
CTR,  HEX  0
      END
```

6-14

```
      LDA  A       / Load multiplier
      SZA          / Is it zero ?
      BUN  NZR
      HLT          / A=0, product=0 in AC
NZR,  CMA
      INC
      STA  CTR     / Store -A in counter
      CLA          / Start with AC=0
LOP,  ADD  B       / Add multiplicand
      ISZ  CTR
      BUN  LOP     / Repeat Loop A times
      HLT
A,    DEC  -       / multiplier
B,    DEC  -       / multiplicand
CTR,  HEX  0       / counter
      END
```


Scanned by CamScanner

6-15 The first time the program is executed, location CTR will go to 0. If the program is executed again starting from location $(100)_{16}$, location CTR will be incremented and will not reach 0 until it is incremented $2^{16} = 65,536$ times, at which time it will reach 0 again.
We need to initialize CTR and P as follows:

```
      LDA   NBR
      STA   CTR
      CLA
      STA   P
        ↓
     Program
        ↓
NBR,  DEC  -8
CTR,  HEX  0
  P,  HEX  0
```

6-16 Multiplicand is initially in location XL. Will be shifted left into XH (which has zero initially). The partial Product will contain two locations PL and PH (initially zero). Multiplier is in location Y. CTR = -16

```
LOP,  CLE
      LDA   Y
      CIR
      STA   Y
      SZE
      BUN   ONE
      BUN   ZRO
ONE,  LDA   XL
      ADD   PL
      STA   PL
      CLA
      CIL
      ADD   XH
      ADD   PH
      STA   PH
      CLE
```

same as beginning of Program in Table 6-14

Double-precision add

$P \leftarrow X + P$

Same as program in Table 6-15

Continued next page

Scanned by CamScanner

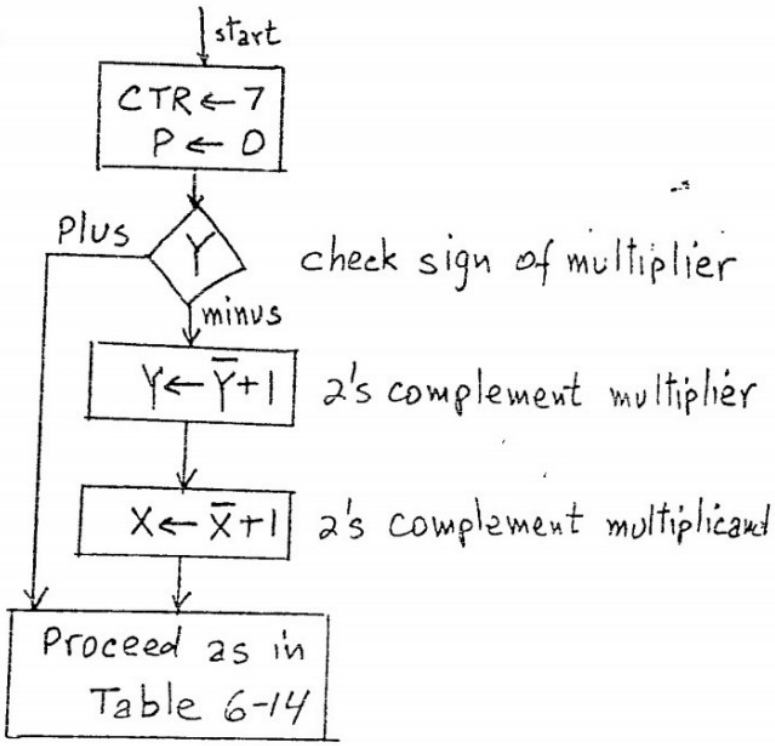
6-16 continued

```
ZRO,    LDA   XL      ┐  Double-precision
        CIL           │  left-shift XH+XL
        STA   XL      │
        LDA   XH      │
        CIL           │
        STA   XH      ┘
        ISZ   CTR     ┐  Repeat 16 times
        BUN   LOP     │
        HLT           ┘
```

6-17

If multiplier is negative, take the 2's complement of multiplier and multiplicand and then proceed as in Table 6-14 (with CTR=-7).

Flow-Chart:



Scanned by CamScanner

6-18 $C \leftarrow A - B$

```
        CLE
        LDA  BL
        CMA
        INC
        ADD  AL
        STA  CL
save    CLA
carry   CIL
        STA  TMP
        LDA  BH
        CMA
        ADD  AH
add carry→ ADD  TMP
        STA  CH
        HLT
TMP,    HEX  0
```

To form a double-precision 2's complement of subtrahend BH+BL, a 1's complement is formed and 1 added once.

Thus, BL is complemented and incremented while BH is only complemented.

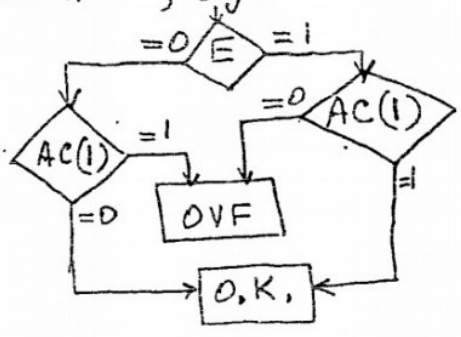
Location TMP saves the carry from E while BH is complemented.

6-19 $z = x \oplus y = xy' + x'y = [(xy')' \cdot (x'y)']'$

```
        LDA  Y
        CMA
        AND  X
        CMA
        STA  TMP
        LDA  X
        CMA
        AND  Y
        CMA
        AND  TMP
        CMA
        STA  Z
        HLT
X,      —
Y,      —
Z,      —
TMP,    —
```

6-20

```
        LDA  X
        CLE
        CIL      / zero to low order bit; sign bit in E
        SZE
        BUN  ONE
        SPA
        BUN  OVF
        BUN  EXT
ONE,    SNA
        BUN  OVF
EXT,    HLT
```



Scanned by CamScanner

6-21

Calling program

```
BSA   SUB
HEX   1234  /Subtrahend
HEX   4321  / minuend
HEX   0     / difference
```

subroutine

```
SUB,  HEX 0
      LDA SUB I
      CMA
      INC
      ISZ SUB
      ADD SUB I
      ISZ SUB
      STA SUB I
      ISZ SUB
      BUN SUB I
```

6-22

Calling Program

```
BSA   CMP
HEX   100  /starting address
DEC   32   /number of words
```

Subroutine

```
CMP,  HEX 0
      LDA CMP I
      STA PTR
      ISZ CMP
      LDA CMP I
      CMA
      INC
      STA CTR
LOP,  LDA PTR I
      CMA
      STA PTR I
      ISZ PTR
      ISZ CTR
      BUN LOP
      ISZ CMP
      BUN CMP I
PTR,  —
CTR,  —
```

6-23

```
CR4,  HEX 0
      CIR
      CIR
      CIR
      CIR
      BUN CR4 I
```

E	AC	AC HEX
1	0000 0111 1001 1100	079C
1	1001 0000 0111 1001	9079

6-24

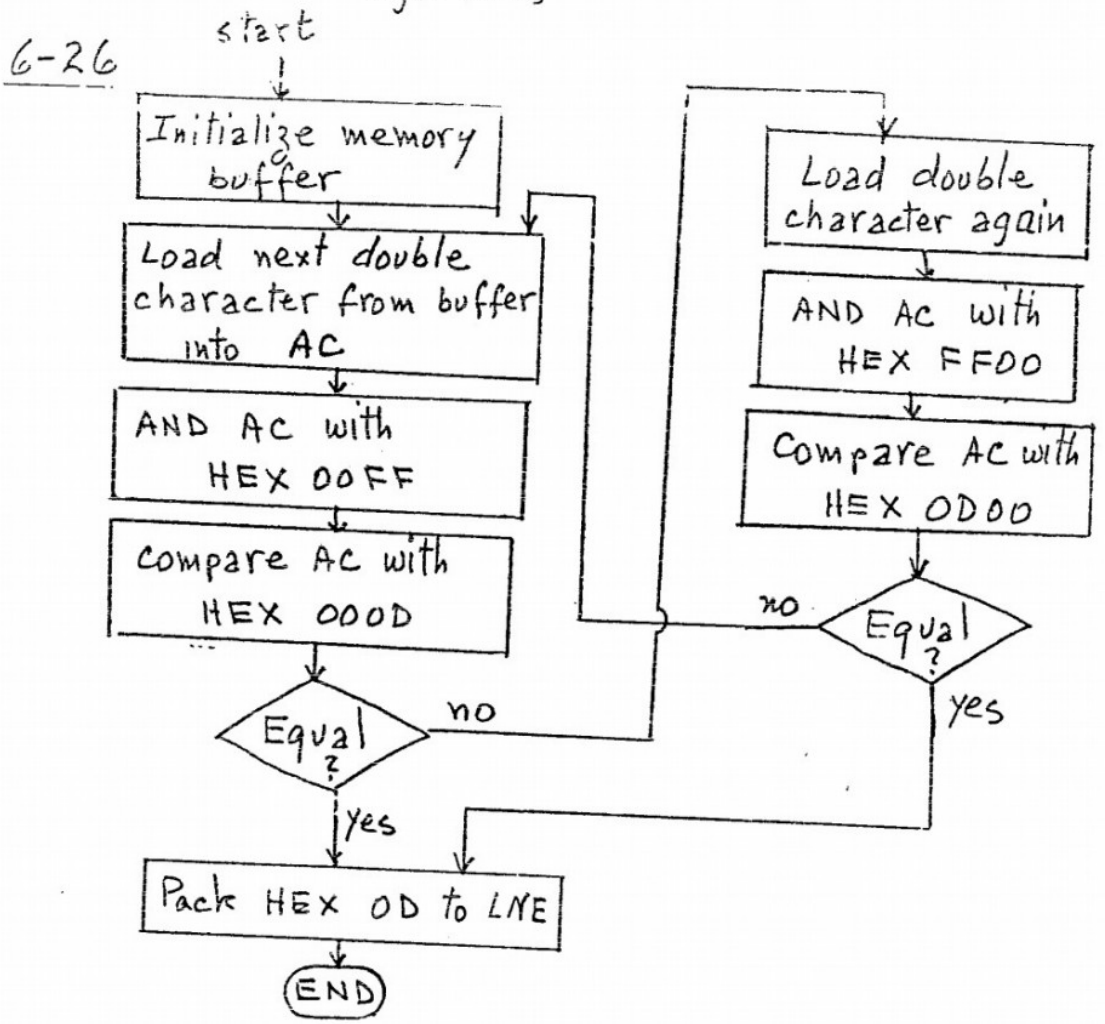
```
      LDA ADS
      STA PTR
      LDA NBR
      STA CTR
LOP,  BSA IN2  /subroutine Table 6-20
      STA PTR I
      ISZ PTR
      ISZ CTR
      BUN LOP
      HLT
ADS,  HEX 400
PTR,  HEX 0
NBR,  DEC -512
CTR,  HEX 0
```


Scanned by CamScanner

6-25

```
LDA   WRD
AND   MS1
STA   CH1
LDA   WRD
AND   MS2
CLE
BSA   SR8   /subroutine to
            shift right
            eight times
STA   CH2
HLT
WRD,  HEX -
CH1,  HEX -
CH2,  HEX -
MS1,  HEX 00FF
MS2,  HEX FF00
```

6-26



Scanned by CamScanner

6-27

Location	Hex code			
200	3213	SRV,	STA	SAC
201	7080		CIR	
202	3214		STA	SE
203	F200		SKI	
204	4209		BUN	NXT
205	F800		INP	
206	F400		OUT	
207	B215		STA	PTI I
208	6215		ISZ	PTI
209	F100	NXT,	SKO	
20A	420E		BUN	EXT
20B	A216		LDA	PT2 I
20C	F400		OUT	
20D	6216		ISZ	PT2
20E	2214	EXT,	LDA	SE
20F	7040		CIL	
210	2213		LDA	SAC
211	F080		ION	
212	C000		BUN	ZRO I
213	0000	SAC,	—	
214	0000	SE,	—	
215	0000	PTI,	—	
216	0000	PT2,	—	

6-28

```
SRV,  STA  SAC
      CIR
      STA  SE
      LDA  MOD   /check MOD
      CMA
      SZA
      BUN  NXT   / MOD≠ all 1's
      SKI        ┐
      BUN  NXT   │ service
      INP        │ input
      OUT        │ Device
      STA  PTI I │
      ISZ  PTI   ┘
      BUN  EXT   / MOD≠ 0
```

```
NXT,  LDA  MOD
      SZA
      BUN  EXT
service SKO      ┐
output  BUN  EXT │
device  LDA  PT2 I
        OUT      │
        ISZ  PT2 ┘
EXT,  continue as
      in Table 6-23
```


Scanned by CamScanner