19/7/17

UNIT - II

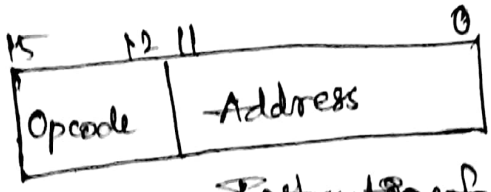
COMPUTER INSTRUCTIONS

Instruction codes:

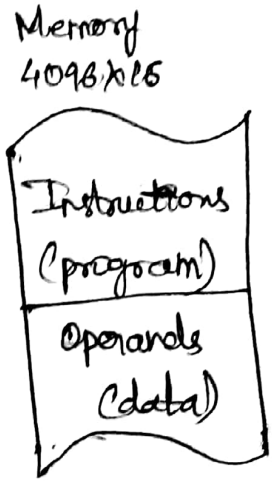
If instruction is represented in binary codes, then it is called instruction code.

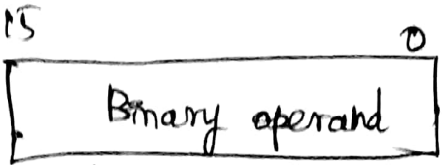
Operation code is a part of instruction code. The bits of the operation codes tells the operation to be performed.

Stored program organisation:

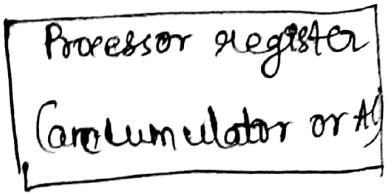


Instruction format





Representation of operand.



In binary form which ~~per~~ contains 16 bits

Processor register is used to manipulate the data called accumulator

→ Memory contains 4096 location.
Each location stores 16 bits. Data containing each location is called "word"

→ We require 12 address bits for 4096 locations ($2^{12} = 4096$)
Each location stores 16 bits of information
Op code - operation code

→ Each word contains 16 bits

Address specifies the address of the operand.

→ Instruction format is simplest way of organizing computer. It contains
1. Op code
2. Address.

opcode
The bits in opcode specifies the operation to be performed.

Op code: indicates what type of operation is to be performed

Binary operand: representation of operand in Binary fo which contains 16 bits.

→ If operand is given in instruction, then that ope added to the content of accumulator and the will be stored in accumulator.

→ When operand is specified directly in instruction, it is called immediate operand & instruction is called immediate instruction.

→ To perform operations such as clearing, complementing the accumulator, add bits are not required.

Ex: 0111 → clear accumulator.

→ The address bits can be used for other operation.

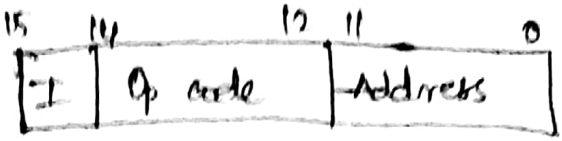
→ In direct address, operand is present in that address

→ In indirect address, operand is present in the address specified by address which is present in address bits.

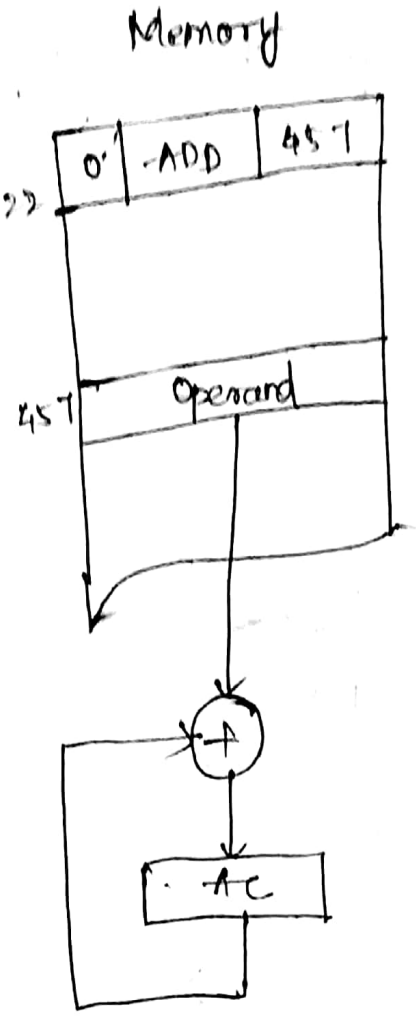
Ex. I → Indirect address mode bit

I = 1 ⇒ Indirect address

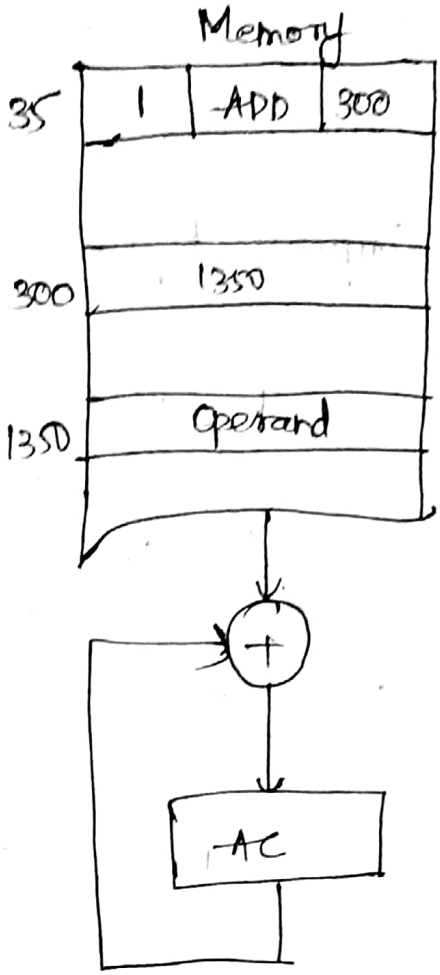
I = 0 ⇒ Direct address



a) Instruction format



b) Direct address



c) Indirect address

~~Fig. 7~~

Fig Demonstration for direct & indirect address.

Operand from memory is added to operand in AC and result is stored in AC

In direct address, ~~address of~~ operand is present in the adress given

by the instruction.

In indirect address, address of operand is present in address given by the instruction

19/7/17

Computer Registers:

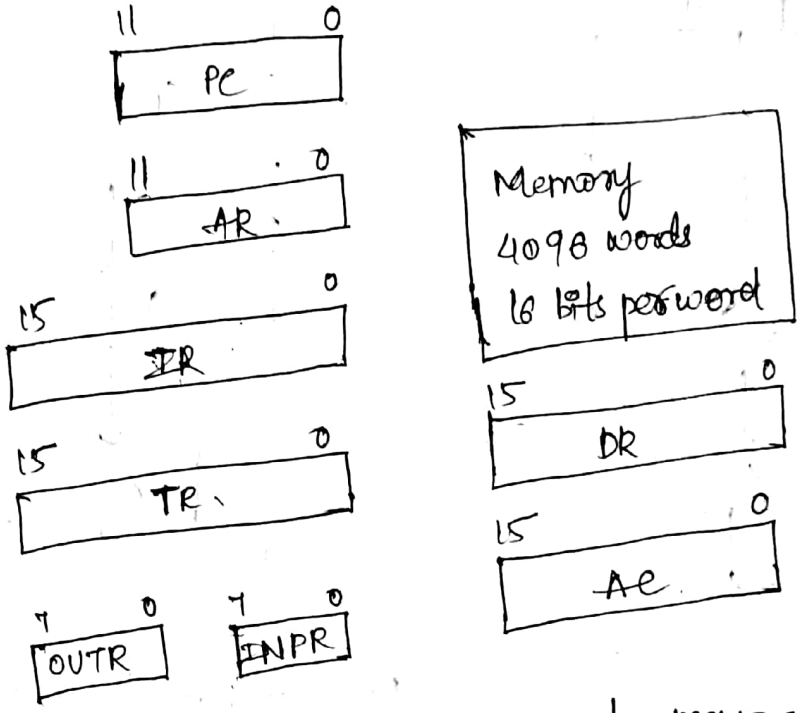


Fig. Basic computer registers and memory

→ The above diagram indicates basic registers.

→ Memory contains 4096 locations indicating 4096 word. 1 word = 16 bits

→ ∴ 12 bits are required for address in order to access.

Program counter (PC):

It contains the address of the next instruction to be read from memory. It is called fetching memory (reading instruction from memory)

(Address ⇒ 12 bits)

Address Register (AR)

It contains address of memory location to be accessed. Size is 12 bits.

Instruction Register (IR)

It contains the instruction that is read from memory. ∴ Size = 16 bits.

Temporary Register (TR)

It is used to hold the temporary data.

OUTR (Output Register)

It holds the output character. output character goes through o/p device. Size of OUTR = 8 bits

INPR (Input Register)

It holds the input character.

Size of character = 8 bits. ∴ INPR size = 8 bits.

It comes from i/p device

DR (Data Register):

It contains the operand read from memory

Size - 16 bits (data)

Accumulator (AC)

It is a processor register which is used to manipulate the data.

Size = 16 bits (data)

In order to transfer information from memory to register or register to register, bus is required.

16 bit common bus:

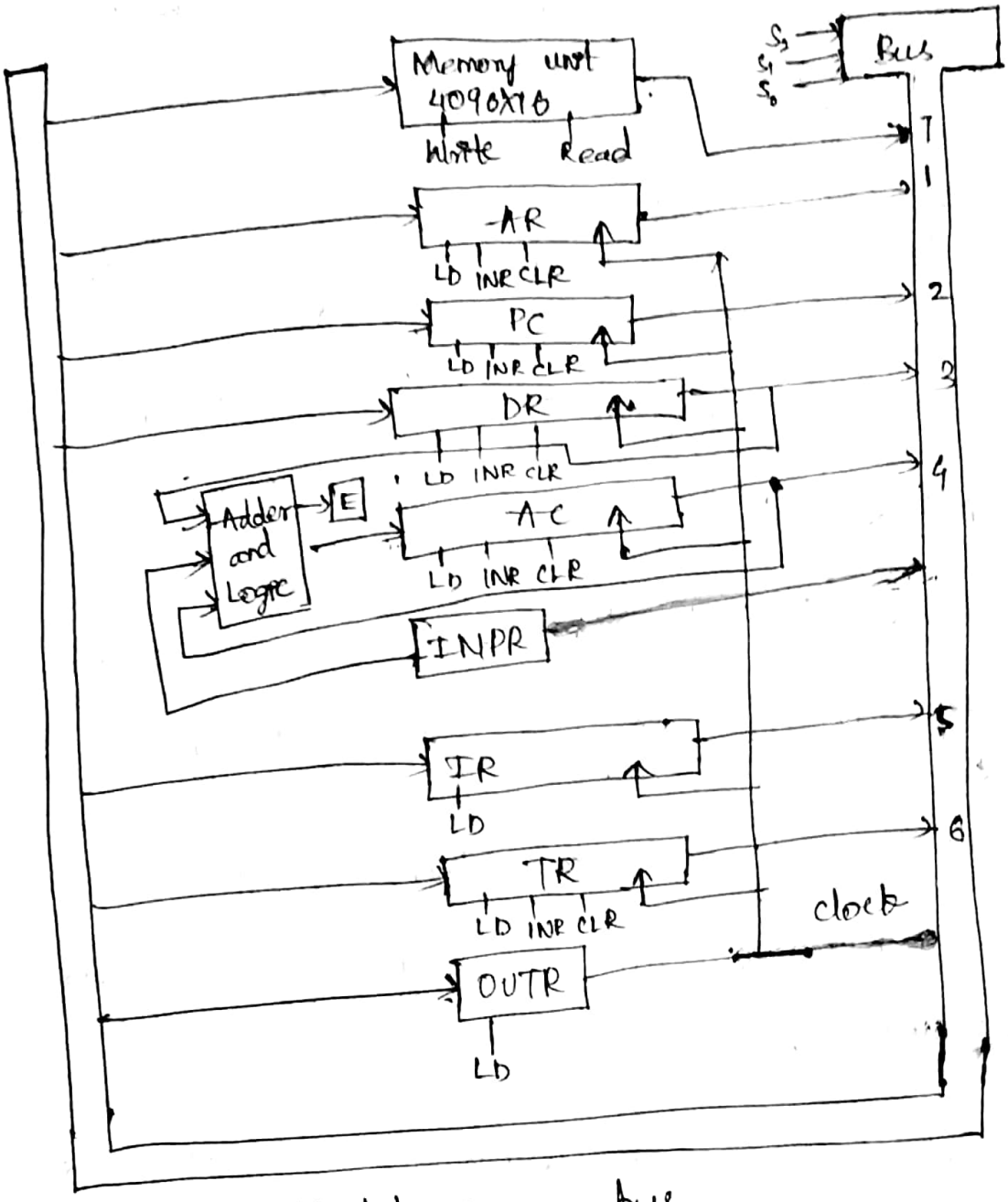
Since the size of word is 16 bit, 16 bit bus is used

→ In memory unit, write is enabled to write the variable and read is " " read the variable.

LD - Load

INR - Increment (Adding 1)

CLR - clear (making 0)



16 bit common bus.

Input register sends the character to accumulator

By connecting the o/p of accumulator to adder and logic ckt, we can perform operations such as shift and complement.

and also we are connecting DR to the adder and logic ckt

→ ALU can perform arithmetic and logic operations (add) on AC and DR since the o/p of DR is also connected to E.g.: Adding DR and AC

logical operations on DR and AC

→ After performing operation on DR and AC, result is stored in AC and if carry is generated, it is stored in E ff_F ff_F

→ OUTP gets the o/p character from accumulator that is sent to o/p device

→ INPR takes the i/p character from i/p device and sends it to accumulator

Based on selection lines, the content of reg is transfered to bus

E.g.: To Transfer content of AC to DR

E.g.: If $S_2 S_1 S_0 = 100$

content of AC is transferred to bus and the load is enabled in DR and content is transferred from Bus to DR

21/7/17

① The adder-subtractor ckt has the following values.

For input mode M and data inputs A and B in each case determine the values of the o/ps $S_3 S_2 S_1 S_0$ and C_4

Sol:

	M	A	B	$S_3 S_2 S_1 S_0$	C_4
a)	0	0111	0110	1101	0
b)	0	1000	1001	0001	1
c)	1	1100	1000	0100	1
d)	1	0101	1010	1011	0
e)	1	0000	0001	1111	0

~~f)~~

~~M=0~~

M=0 in adder-subtractor indicates addition = A+B

M=1 → subtraction ~~= A+B~~ = A+B+1

```
 1100
 0000
    1
-----
10100
```

d) ~~0+0+B~~ = 1010

1's $\bar{B} = 0101$

```
 0101
   +1
-----
 0110
 0101
-----
 1011
```

e) B = 0001

$\bar{B} = 1110$

$\bar{B}+1 = 1111$

A = 0000

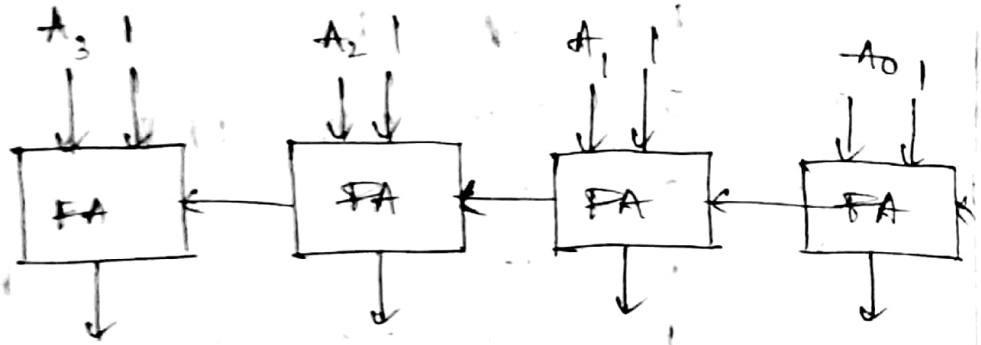
```
 1111
```

(2) Design a 4-bit combinational circuit decrementer using 4 full adder circuits

Sol:

A − 1 = A + 2's complement of 1

= A + 1111



(3) What is the value of output H in 4-bit combinational circuit shifter if

I/p A = 1001, S = 1, I_R = 1, I_L = 0

Sol:

S = 1 ⇒ left shift

I_L = 0

A = 1001

The result is 0010

= 1000

(∵ If I/p is $A_3 A_2 A_1 A_0$

O/p is $A_2 A_1 A_0 I_L$

for left shift)

A_0	A_1	A_2	A_3
1	0	0	1

S=1 ; the H_1 H_2 H_3

A_1	A_2	A_3	I_2
0	0	1	0

4) A digital computer has a common bus system for 16 registers of 32 bits each. The bus is constructed with multiplexer

a) How many selection I/ps are there in each multiplexer.

b) What size of multiplexer are needed.

c) How many multiplexers are there in the bus.

a) 4 selection lines } (16×1 MUX, 16 I/P's,
b) 16×1 (∵ 16 I/P's 4 selection lines, 1 O/P)

c) 32 multiplexers (32 bits)

5) Design an arithmetic ckt with one selection variable 'S' and two 4 bit data I/ps A and B.

The circuit generates the following 4 arithmetic operations in conjunction with C_{in}. Draw the logic diagram for the first 2 stages.

S	$C_{in}=0$	$C_{in}=1$
0	$D=A+B$ (add)	$D=A+1$ (increment)
1	$D=A-1$ (decrement)	$A+\overline{B}+1$ (subtract)

Sol.

output of Arithmetic ckt is $A+Y+C_{in}$

S	C_{in}	Y	$D=A+Y+C_{in}$
0	0	B	$D=A+B$
0	1	0	$D=A+1$
1	0	All 1's	$D=A-1$
1	1	$\overline{B}$	$D=A+\overline{B}+1$

llowing 4

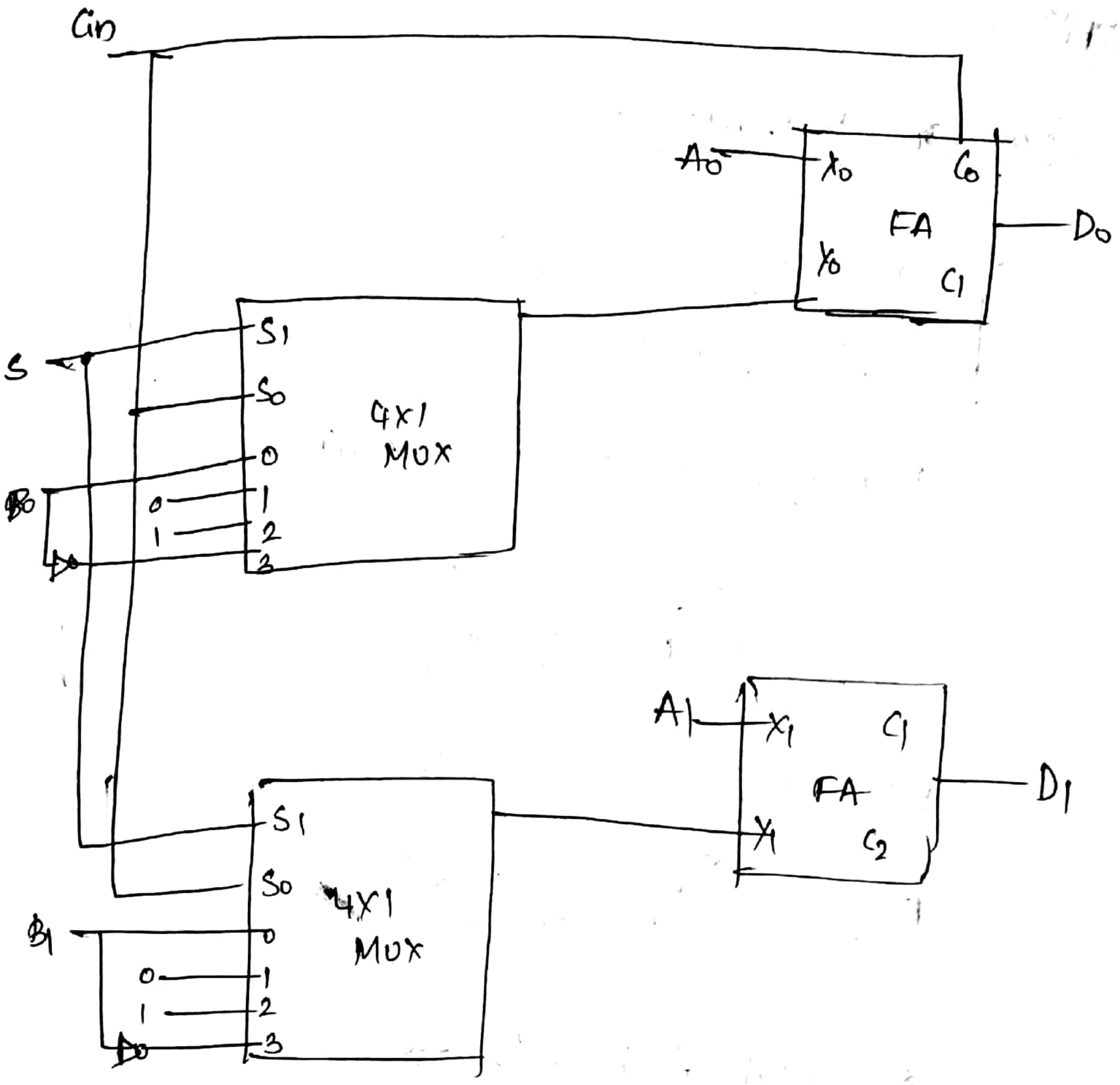
n with o/p

for the

(Complement)

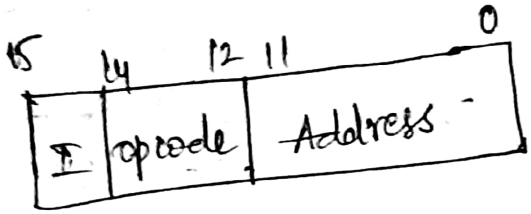
+1 (subtract)

Y+Cin



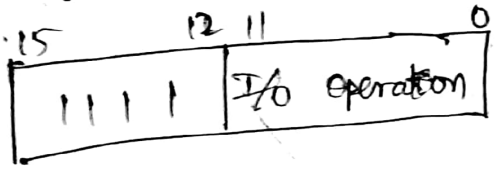
24/7/17

Computer Instructions:



a) Memory-reference instruction.

b) Register-reference instruction.



c) Input-output instruction

Fig Basic computer instruction formats

→ These are the instructions for a basic computer (i.e. a), b), c))

a) Memory-reference instruction:

Memory is ~~refle~~ referred in case of memory-reference instruction inorder to get operand. Memory reference is not required in case of b), c)

whether it is

Bit I indicates a direct addressing mode or indirect addressing mode

3 bits are used to (12 to 14) specify the type of operation.

(which specifies operand also)

Address of operand is called as effective address.

In direct addressing mode, address mentioned in instruction is called effective address of operand

~~In indirect address, the address specified in the address specified in the~~

$2^{12} = 4096 \rightarrow$ 12 bits for address (4096 memory locations)

b) Register reference instruction:

For this instruction, op code is 111, I=0

0-11 bits used to specify the operation to be performed on Accumulator and test is also performed on accumulator

c) Input-output instruction:

If op-code = 111, it indicates I/O instructions. and I = 1

(When opcode ≠ 111, it indicates memory reference instruction) it defines the type of I/O operation and test performed.

→ In indirect address, address contained in the address specified in the instruction is called effective address

Basic computer instructions:

Hexadecimal code

a) Memory-reference instruction:

Symbol	opcode	I=0	I=1	Description
AND	0000	0XXX	8XXX	AND memory word to
ADD	0001	1XXX	9XXX	ADD " "
LDA	010	2XXX	AXXX	load " " "
STA	011	3XXX	BXXX	store content of AC in memory
BUN	100	4XXX	CXXX	Branch unconditionally
BSA	101	5XXX	DXXX	Branch and save return address
ISZ	110	6XXX	EXXX	Increment and skip if zero

→ LDA indicates load memory word to Accumulator.

→ Reverse of load operation is store operation content of AC is transfered to memory.

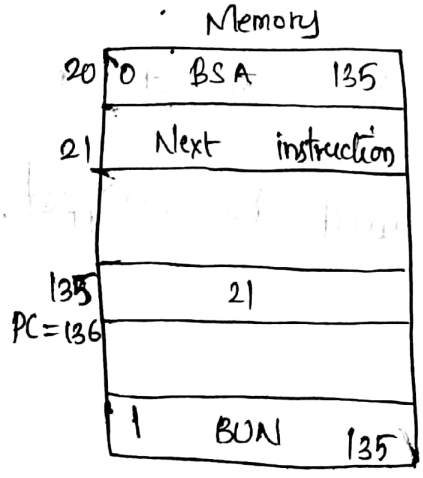
→ AND operation is b/w memory word (which is given by effective address ~~present in address in instruction~~) and accumulator

There are 7 opcodes (000 to 110)

→ When Branch unconditionally is executed the program control will be transferred to instruction specified by the effective address. (Usually, next instruction is executed, but the opcode directly jumps to the instruction in effective address)

→ Set of instructions which are to be executed repeatedly is called subroutine. ~~After the~~ Saving the return address which is to be executed after subroutine is called branch and save return address.

Fig: Example of BSA instruction execution



→ After BSA instruction, the ~~instruction~~ storing address of subroutine is effective address (135+1) which is executed. ~~after BSA~~ that instruction.

The address of next instruction of BSA instruction it will be stored in 135

21 → Return address to

The address ~~at~~ which program control has to return after execution of subroutine.

In general, program counter holds the address of next instruction while the present address is being executed.

→ For ~~basic~~ conditional branching, condition is to be satisfied for branching.

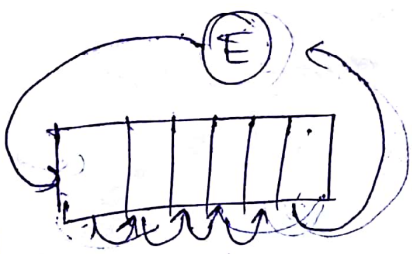
For unconditional branching, there is no condition.

→ ISZ increments the ~~operand~~ data in effective address and if the result is zero, next instruction will be skipped.

b) Register reference instructions

Symbol	Hexadecimal code I=0	Hexadecimal code I=1	Description
CLA	7800		clear AC
CLE	7400		clear E
CMA	7200		complement AC
CME	7100		Complement E
CIR	7080		circulate right AC to E
CIL	7040		circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC → +ve
SNA	7008		Skip next instruction if AC → -ve
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer.

Circulate right indicates LSB bit is transfered to E-flipflop and then transfered to MSB



→ Skip next instruction indicates Program Counter is loaded with PC+1 (next instruction)

→ Halt After execution of Halt instruction, start-stop flip flop will be cleared

Input-Output Instruction:

Symbol	Hexadecimal code		Description.
	I=0	I=1	
INP		F800	Input character to AC
OUT		F400.	o/p character from AC
SKI		F200	Skip on i/p flag
SKO		F100	Skip on o/p flag
ION		F080	Interrupt on
IOF		F040	" off

INP.
OUT
SKI
SKO
ION
IOF

→ INP indicates input character from i/p register to AC

→ OUT → character from AC is transfered to o/p register

→ SKI – Skip the next instruction if the i/p flag is set

→ SKO – Skip the next instruction if the o/p flag is set

→ ION – IEN flip flop is set (IEN-Interrupt Enable)

→ IOF – " " " cleared, i.e. interrupt is disabled

Timing and control:

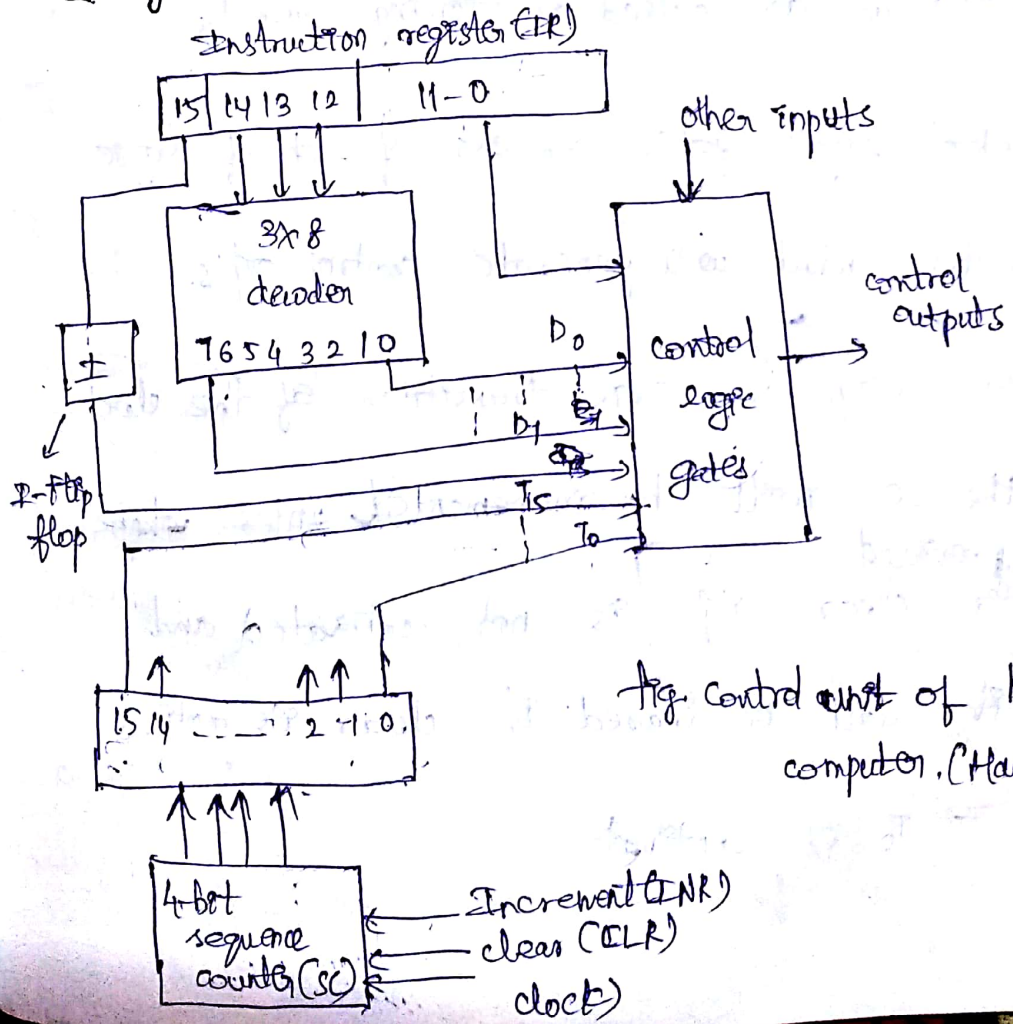


fig. Control unit of basic computer. (Hardwired)

→ Instruction read from memory is stored in Instruction register (IR)

→ I - flip-flop contains 15th bit

→ Op-code is decoded as D_0 to D_7

Eg. $D_7 = 1$ indicates opcode = 111

opcode = 000 ⇒ D_0 is activated

→ 4-bit sequence counts from 0 to 15 (0000 to 1111). O/p of 4-bit sequence is decoded as 0 to 15

Eg: O/p of sc = 1111 ⇒ $T_{15} = 1$

→ T_1 to T_{15} are called as timing signals.

→ control logic gates consists of set of logic gates which will generate control o/p's.

→ For every +ve edge transition of the clock, the sc will be incremented provided the clear i/p is not activated and it will be cleared if clear is active

⇒ T_0 is activated

→ Hardwired control unit consists of gates, FF's, decoders and other digital ckts (No memory)

→ In microprogramed control unit, control information is stored in the control memory. (memory used for control information.

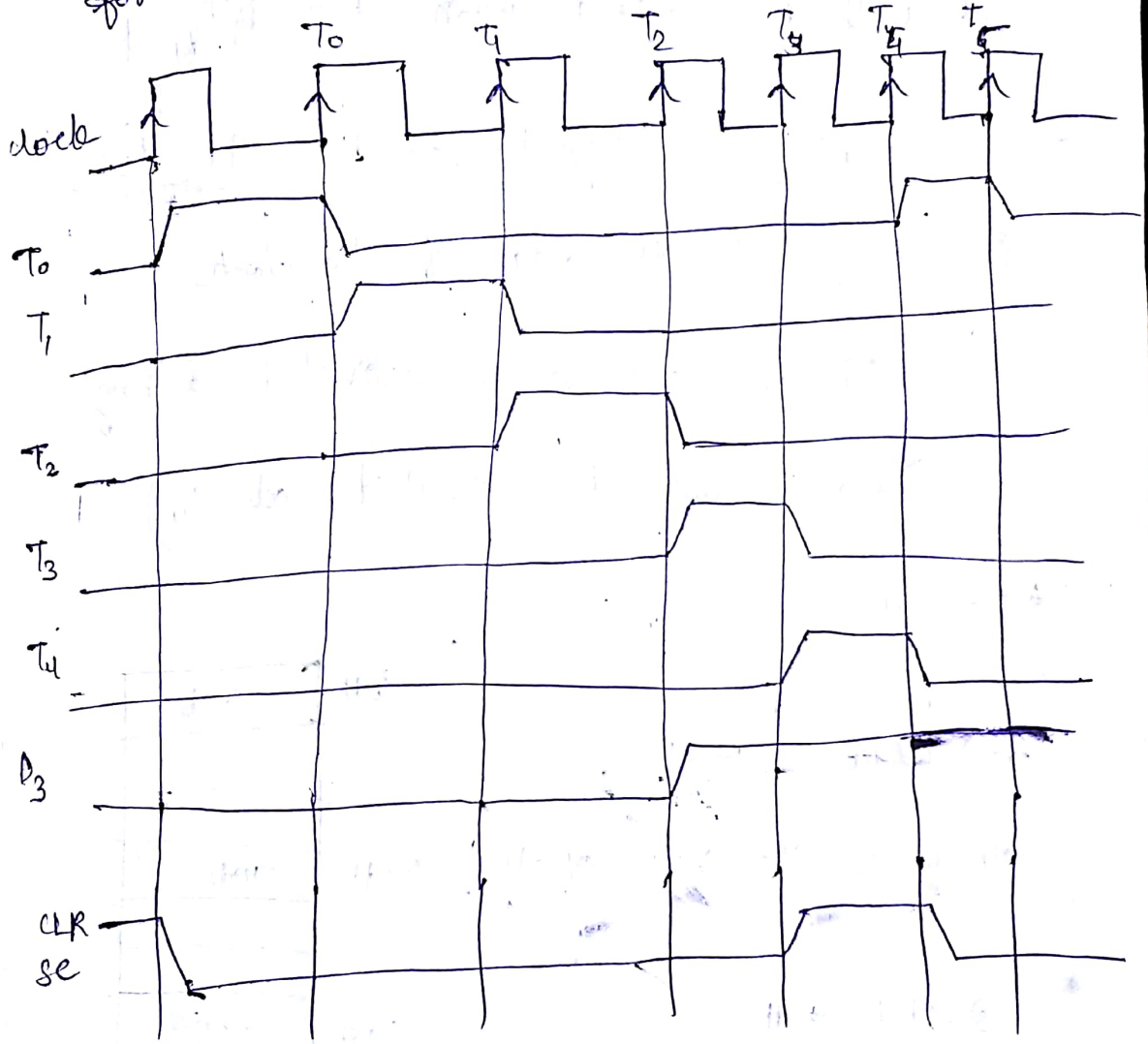


Fig: Example of control timing signals

→ When CLRSC is activated, o/p of SC will be 0000 ∴ T0 is activated.

→ When the next +ve edge of

clock pulse is received, T_1 will be activated. Similarly when next +ve edge is received, T_2 will be activated and so on.

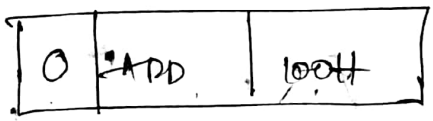
$D_3 T_4 : SC \leftarrow 0$

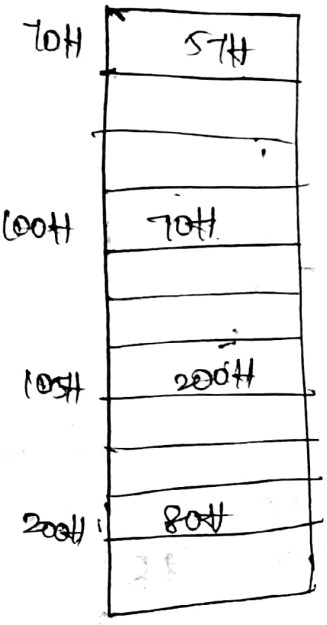
SC will be cleared ~~when~~ when $D_3 T_4 = 1$

→ Incrementing the register, counter etc takes place only at +ve edge of the clock.

(i.e. Though CLR SC is activated during T_3, T_0 will be activated at T_4)

① AC = 20H

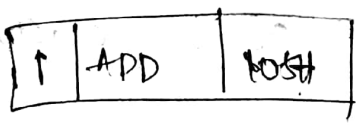




What is the value of AC?

20H + 70H.

= 90H

② 

AC ?

= 20H + 80H = 100H = A0H (Hexadecimal)

③

0	ADD	70H

What is the value of AC?

20H + 57H

= 77H

④

1	ADD	100H

What is the value of AC?

57H + 20H

= 77H

31/7/17

Instruction cycle:

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

→ It is further divided into sequence of sub-cycles or phases

→ In basic computer, instruction cycle consists of the following phases

1) Fetching an instruction from memory.
2) Decode the instruction.
3) Read the effective address from memory if the instruction has indirect address.
4) Execute the instruction.

Fetch and Decode:

To fetch the instruction from memory address of instruction is required.

Program counter contains the address of instruction that is to be fetched which is transfered to Address register, which is connected to address i/p of memory

Transfer of address from (containing) P.C. to address register takes place when T_0 is active.

* T_0 : AR ← PC

* T_1 : IR ← M[AR], PC ← PC+1 (Fetch)

Memory word specified by address word in Address register is transfered to instruction register and P.C is incremented by 1 when T_1 is active. (∵ P.C contains the address of next instruction to be fetched from memory)

* T_2 : D_0 ..... D_7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)
(Decode)

3x8 decoder decodes 12-14 bits in instruction into D_0 .... D_7

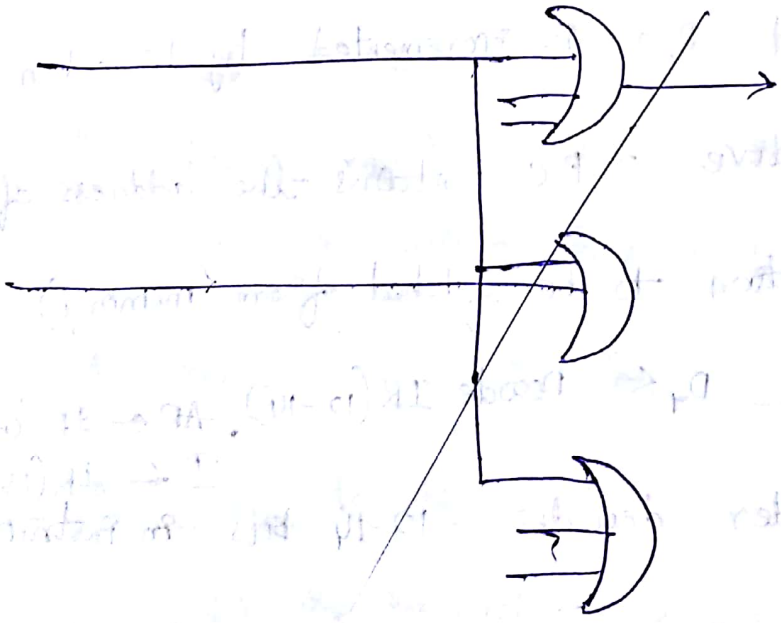
→ In case of memory-reference instruction, the bits (0-11) indicates direct or indirect

address which is loaded into Address register.

→ The bit is indicates direct or indirect address. In case of memory reference instruction

→ In case of nonmemory instruction, I = 0 indicates register-reference instruction and I = 1 indicates I/O operation.

→ The 15th bit is loaded into I flip flop. These all operations takes place if T_2 is active



Circuit diagram

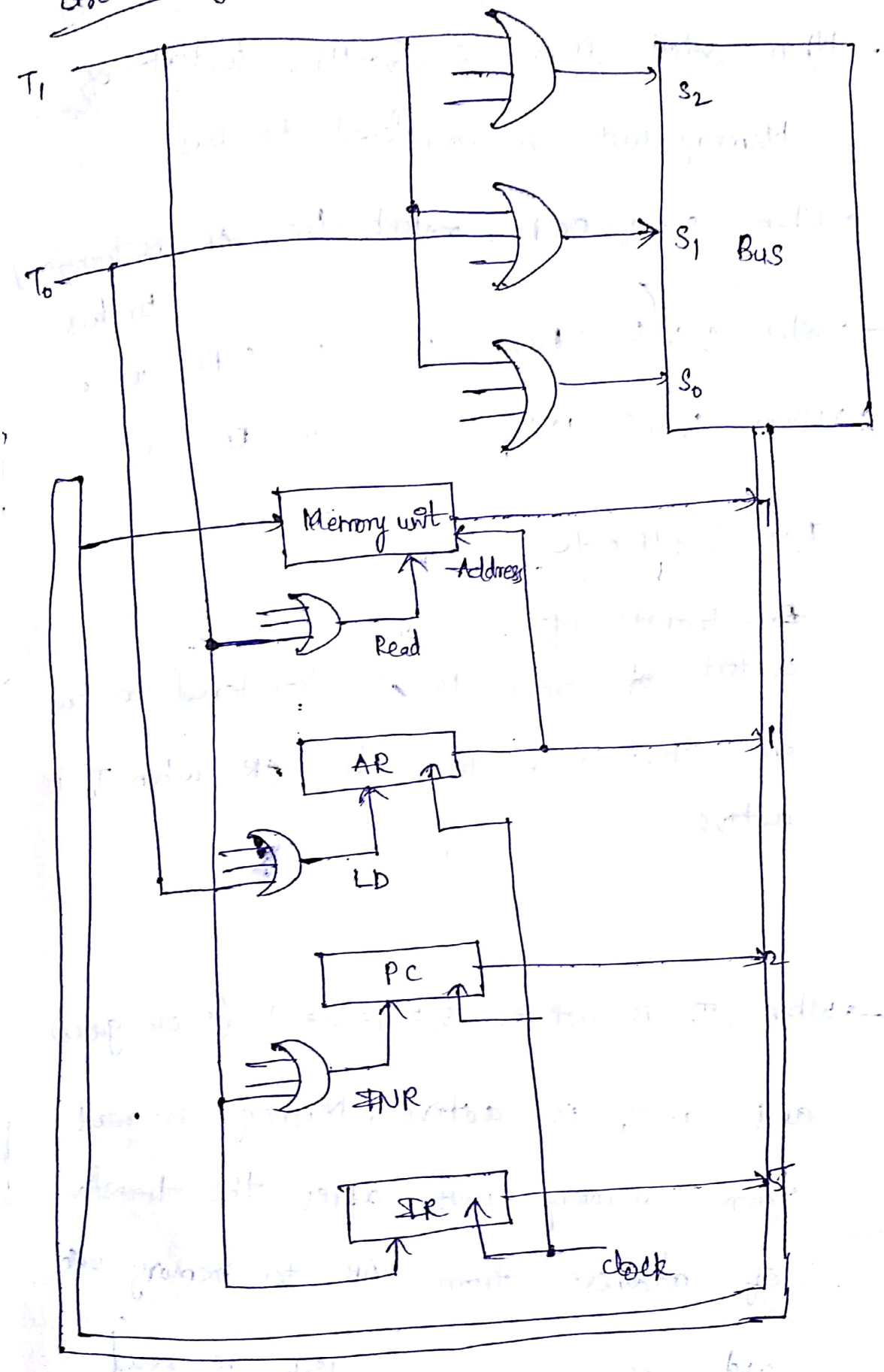


Fig. Register transfers for the fetch phase

→ When select lines $S_2 S_1 S_0 = 111$, content of Memory unit is transfered to bus.

→ When $S_2 S_1 S_0 = 001$, content from AR is transfered to bus

→ When $S_2 S_1 S_0 = 010$, " " PC "

→ When $S_2 S_1 S_0 = 101$, " " DR "

Eg: T_0 : AR ← PC

~~For transfer of~~ content ~~of~~ from PC is transfered to bus and then from bus to AR when T_0 is active

→ When T_1 is active, $S_2 = S_1 = S_0 = 1$ (∵ OR gate)

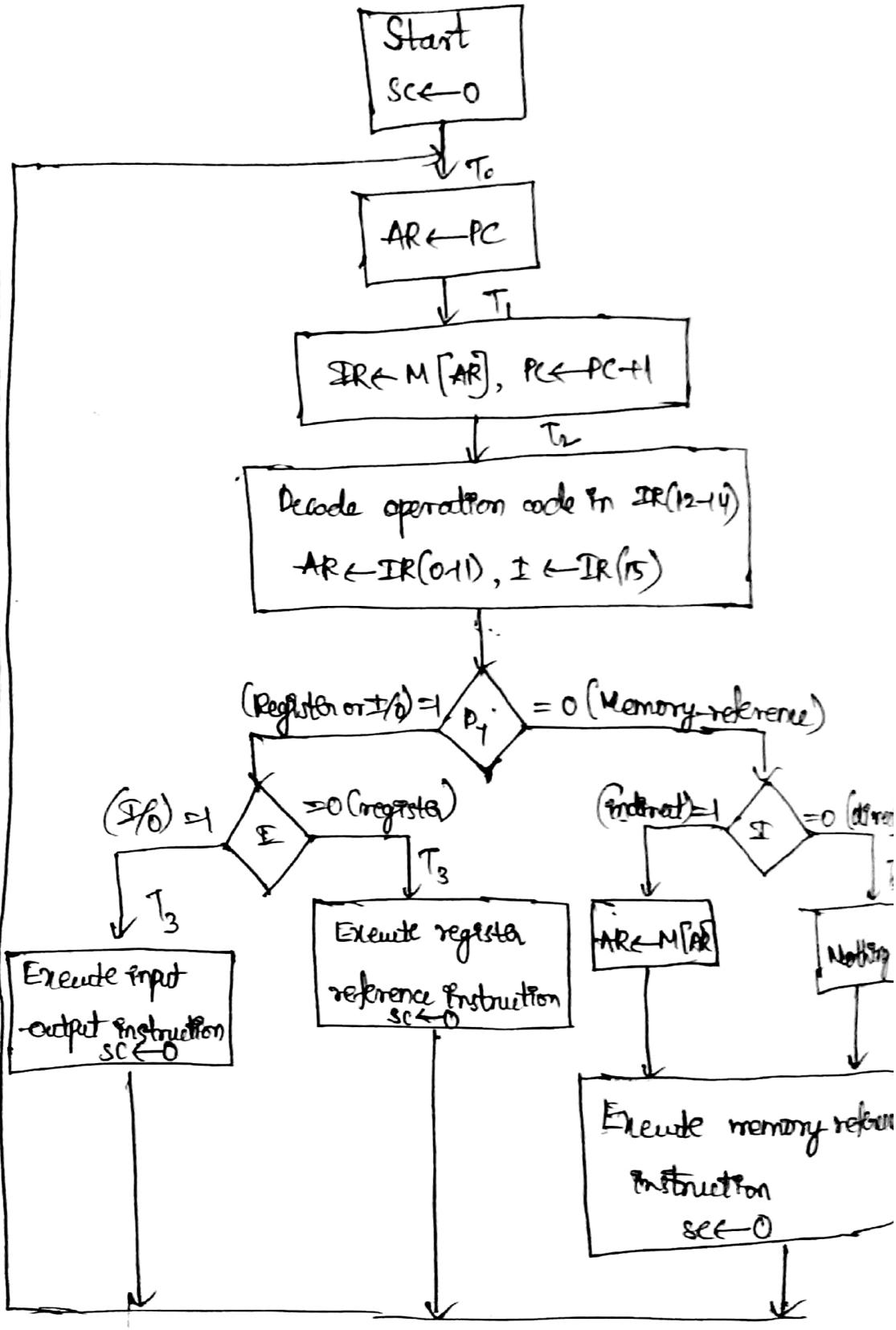
and as T_1 is active, Memory is read from memory unit after the transfer of address from AR to memory unit and Memory ~~is~~ which is read from memory unit is transfered to

bus and load of IR is enabled (as $T_1 = 1$)

and ~~the~~ then transfered from bus to IR

2/8/17

Flowchart for instruction cycle:



→ sequence counter is cleared initially

∴ T_0 is activated.

→ When T_0 is active, content of PC is loaded into AR

→ PC is incremented by 1 and T_1 is activated.

∴ Memory word from AR is transfered to IR and PC is incremented by 1 (∵ next instruction is fetched)

→ When T_2 is activated,

→ (12-14) bits are decoded as $D_0, \ldots D_7$

(0-11) bits in IR is transfered to AR

and 15th bit is transfered to I - flip flop

→ When $D_7 = 1 \Rightarrow$ Register-reference or I/O operation.

$D_7 = 0 \Rightarrow$ Memory-reference instruction.

→ When $D_7 = 0$ and $I = 1 \Rightarrow$ indirect address

$I = 0 \Rightarrow$ direct address

→ When $D_7 = 1$ and $I = 0 \Rightarrow$ register reference instruction

$I = 1 \Rightarrow$ I/O operation.

(iv) When T_3 is activated

→ In case of memory-reference instruction.

When $I = 1$, Memory word from AR is transfered to AR (i.e. effective address)

When $I = 0$, nothing is performed

→ In case of register reference or I/O operation, I/O instruction or register reference instruction is executed and SC is cleared i.e. next instruction is fetched ~~($T_0 = 0$)~~ (T_0 is activated)

→ Execution of register-reference or I/O operation takes place when T_3 is active

→ Where as memory reference instruction is executed when T_4 (or) T_5 (or) T_6 is active

* $D_7' I T_3$: $AR \leftarrow M[AR]$

($\because D_7 = 0, I = 1, T_3 = 1$)

* $D_7' I' T_3$: Nothing

* $D_7 I' T_3$: Execute a register-reference instruction

* $D_7 I T_3$: " " Input-output instruction

These are called control functions.

3/8/11

~~Execution of~~

Execution of Register Reference Instructions:

condition for register-reference instruction:

$$D_7 I' T_3 = r$$

(AND gate)

→ The o/p of $D_7 I' T_3$ is given to clear i/p of SC

→ When $r = 1$, SC is cleared.

→ clock is given as i/p to registers and SC.

∴ SC is cleared during +ve edge of the clock. ∴ $IR(i) = B_i$ (∵ IR indicates 0-11 bits.)

* $r:$ SC ← 0 clear SC

When $IR(11) = 1$, (or $B_{11} = 1$ → It indicates $B_{10} = B_9 \dots = B_0 = 0$

clear AC (AND operation of r and $B_{11} = 1$)

rB_{11} → control function to perfo

$r: SC \leftarrow C$ 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 $rB_7: AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$ (MSB ↑, LSB ↓) Circulate right Ac to E

CIL $rB_6: AC \leftarrow shl\ AC, AC(0) \leftarrow E, E \leftarrow AC(15)$ Circulate left Ac and E

INC $rB_5: AC \leftarrow AC + 1$ Increment Ac

SPA $rB_4: If\ (AC(15) = 0)$ then $(PC \leftarrow PC + 1)$ Skip next instruction if Ac is +ve

SNA $rB_3: If\ (AC(15) = 1)$ then $(PC \leftarrow PC + 1)$ Skip if -ve

SZA $rB_2: If\ (AC = 0)$ then $(PC \leftarrow PC + 1)$ Skip if Ac is zero

SZE $rB_1: If\ (E = 0)$ then $(PC \leftarrow PC + 1)$ " " E " "

HLT $rB_0: S \leftarrow 0$ Halt computer

↓

Start/Stop flipflop

while present instruction is being executed, PC contains the address of next instruction. In SPA, PC is further incremented.

→ Register reference or I/O instructions will be executed when T_3 is active where as memory reference is executed when T_4 (or) T_5 (or) T_6 are active.

Op-code for memory reference instruction - 000 to 110

Memory-Reference Instructions:

→ operand read from memory is always transfered to data register.

→ When T_3 is active, effective address is read from the memory in case of memory reference instructions.

→ Execution of memory starts when T_4 is active (i.e. Memory is read from AR and transfered to DR)

→ AND operation is performed b/w memory word to AC

AND to AC (op-code - 000)

D_0T_4 : $DR \leftarrow M[AR]$

D_0T_5 : $AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$

(Operations performed when D_0 and T_5 are active)

AND is performed and SC is cleared and

T_0 is activated and next instruction is fetched.

ADD to AC (op-code - 001)

D_1T_4 : $DR \leftarrow M[AR]$

D_1T_5 : $AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$

After adding, if carry is generated, it is transfered to E flip flop

LDA (Load AC) to AC)

D_2T_4 : $DR \leftarrow M[AR]$

D_2T_5 : $AC \leftarrow DR$, $SC \leftarrow 0$

Memory word is transfered to DR and then to AC.

Writing into memory → store

" " register → load

STA (store from AC

D_3T_4 : $M[AR] \leftarrow AC$, $SC \leftarrow 0$

BUN (Branch Unconditionally):

D_4T_4 : PC ← AR , SC ← 0

Effective address is loaded into PC from AR.

BUN → branching to the instruction specified by effective address.. without any condition

BSA : Branch and Save Return Address:

Subroutine – set of instructions performed repeatedly in a program.

After execution of subroutine, again we have to return to main program.

Return address – address where we have to go after subroutine

. Address that PC has to go after the execution of subroutine is next to BSA instruction.

Address in PC (∵ next instruction after BSA) is saved in AR (AR contains effective address)

D_5T_4 : M[AR] ← PC , AR ← AR + 1

M[AR] → effective address

AR + 1 → subroutine address.

Address of subroutine is obtained by incrementing effective address in BSA instruction

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

ISZ : Increment and skip if zero:

Increment the operand in an effective address. and if the result is zero, then next instruction is skipped.

$D_6 T_4$: $DR \leftarrow M[AR]$

$D_6 T_5$: $DR \leftarrow DR+1$

$D_6 T_6$: $M[AR] \leftarrow DR$, if $(DR=0)$ then $(PC \leftarrow PC+1)$,

$SC \leftarrow 0$

6/8/14

Input-Output and Interrupt

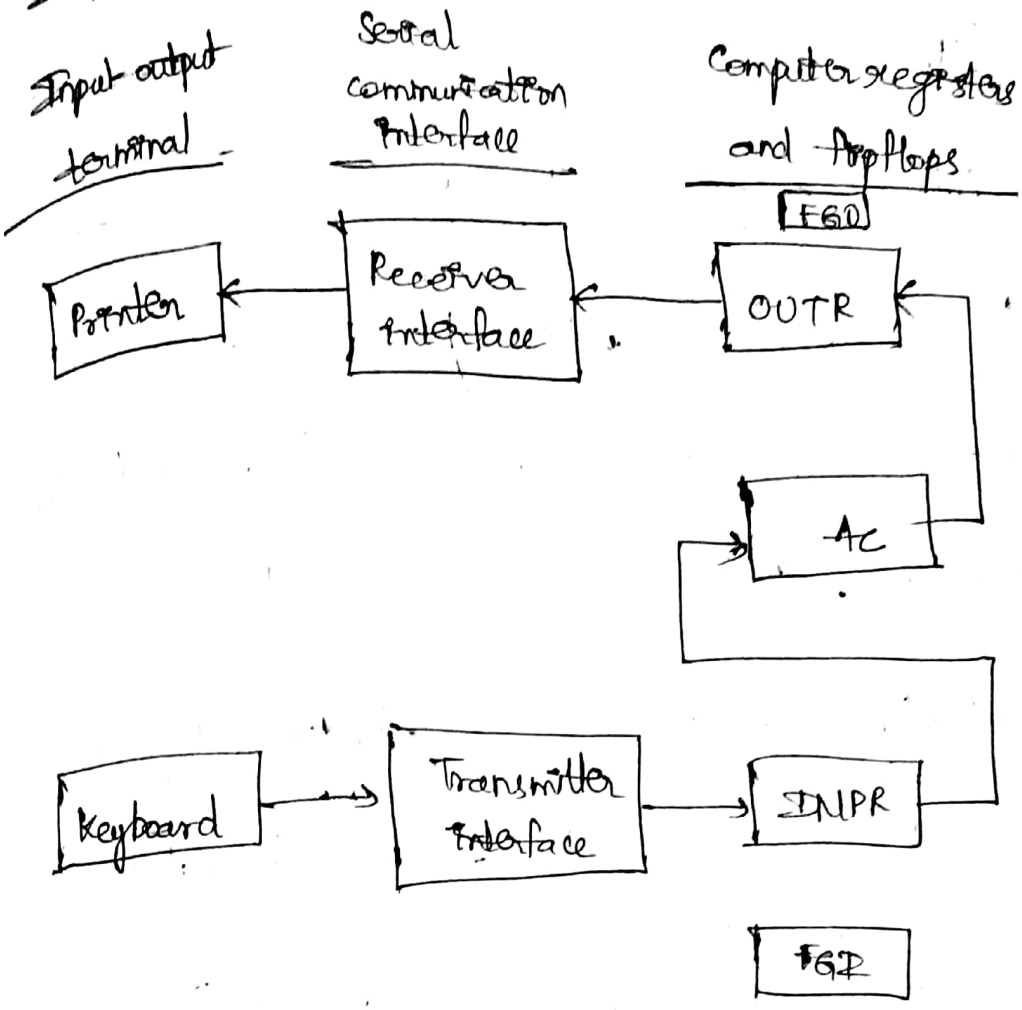


Fig. Input-output configuration.

Keyboard - i/p device

printer - o/p "

FGI - Input Flag

FGO - o/p "

FGI, FGO are one-bit registers which are flip flops.

→ Data from keyboard is transferred to

i/p register (INPR) through transmitter interface serially

Ily, data from output register to o/p device through receiver interface serially.

→ Data from INPR to AC is transferred Ily parallely. Ily from AC to OUTR

→ Initially, i/p flag is cleared. When key is pressed on keyboard, character is transfered to INPR & alpha numeric code.

After a character is transferred to INPR, i/p flag is set. indicates FF of computer. Computer checks the FGI Flip Flop. and if the FGI is set, data is transfered from INPR to AC. and i/p flag is cleared.

→ Initially o/p flag is set and computer checks the " " and if it set,

it will transfer the data from AC to OUTR. and o/p flag is cleared

Transferring data from INPR → AC → o/p transfer

" " " AC to OUTR → o/p "

→ character from OUTR is accepted by printer and printed and the o/p flag is set.

→ i/p and o/p transfers takes place at faster rate as computer is fast and to i/p and o/p devices are slow as i/p and o/p devices are slow. (∵ keyboard → INPR and OUTR → printer)

→ computer checks i/p and o/p flags for i/p and o/p transfers. o/p flag is set once the character is printed.

This process consumes a lot of time.

→ When flag is set, computer is interrupted.

When the computer is interrupted and

· It stop executing main program and

It starts I/O to perform

I/O operation.

The program that is executed when there is an interrupt is called Interrupt Source Routine (ISR)

→ Before executing ISR, or I/O program, it has to save the return address (address which is stored in location 0, PC should be loaded with address: 1. (Branch instruction)

→ While executing the I/O program, IEN (Interrupt Enable) flip flop should be cleared

→ Branch Instruction transfers Program control to I/O operation.

→ The operations that are taking place when there is an interrupt before going to I/O program are called Interrupt cycle which is ISR.

Interrupt cycle

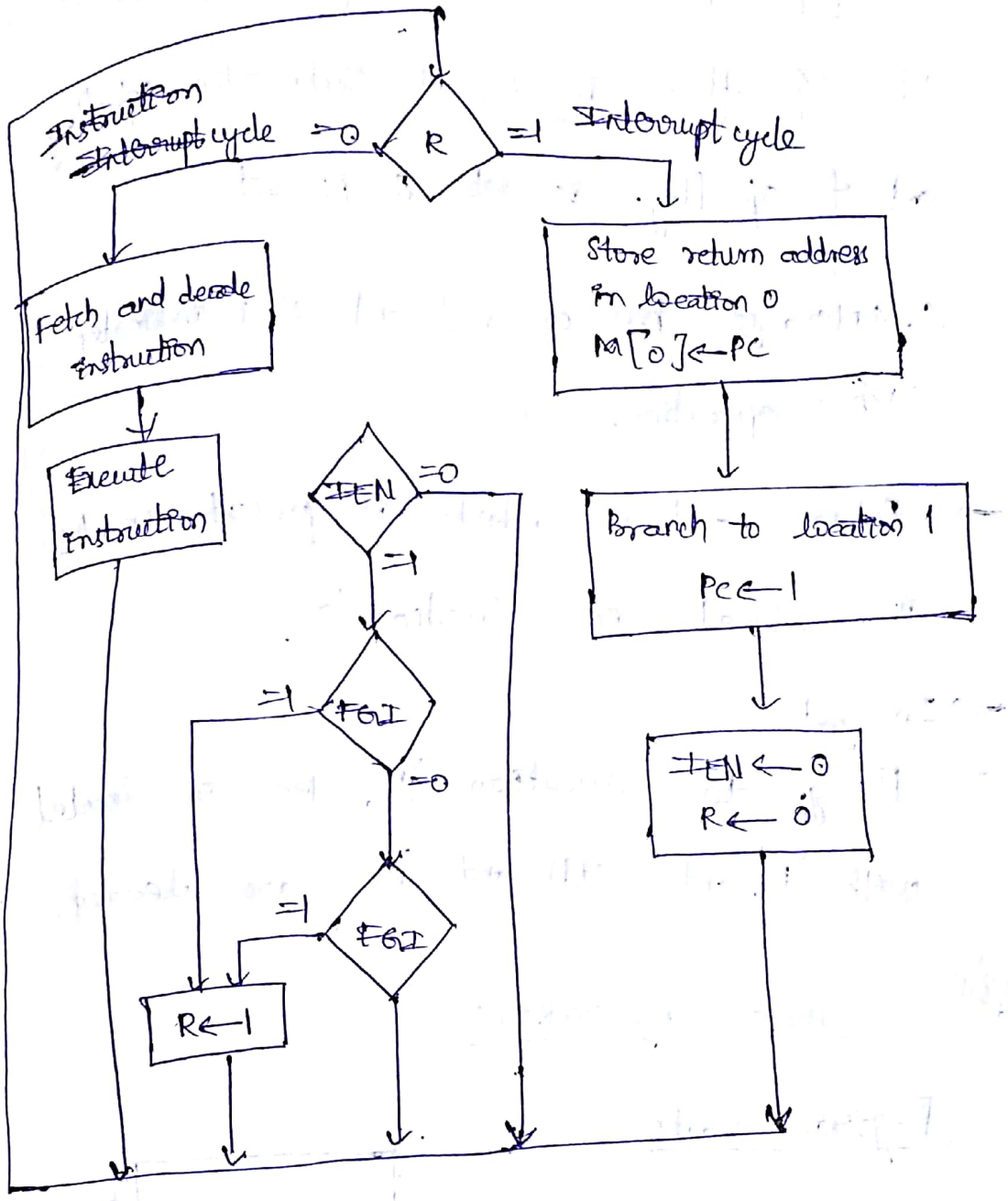


Fig: Flow chart for interrupt cycle

→ 'R' is interrupt FF (Flip Flop)

→ IEN = 0 ⇒ interrupt is disabled, and will not execute I/O operation.

→ If IEN = 1 (i) and if i/p flag = 1 ⇒ i/p transfer
(ii) and if o/p " " ⇒ o/p "

'R' flip flop is called interrupt FF.

If i/p flag is 0, it checks the i/p flag and if i/p flag is set, R is set

∴ Interrupt has occured and R=1 indicates I/O operation.

→ Return address which is present in PC is stored in location '0'.

~~→ To go!~~

→ To go to location 'l', pc is loaded with 'l' and IEN and 'R' are cleared.

8/8/17

Stack organisation:

Register stack:

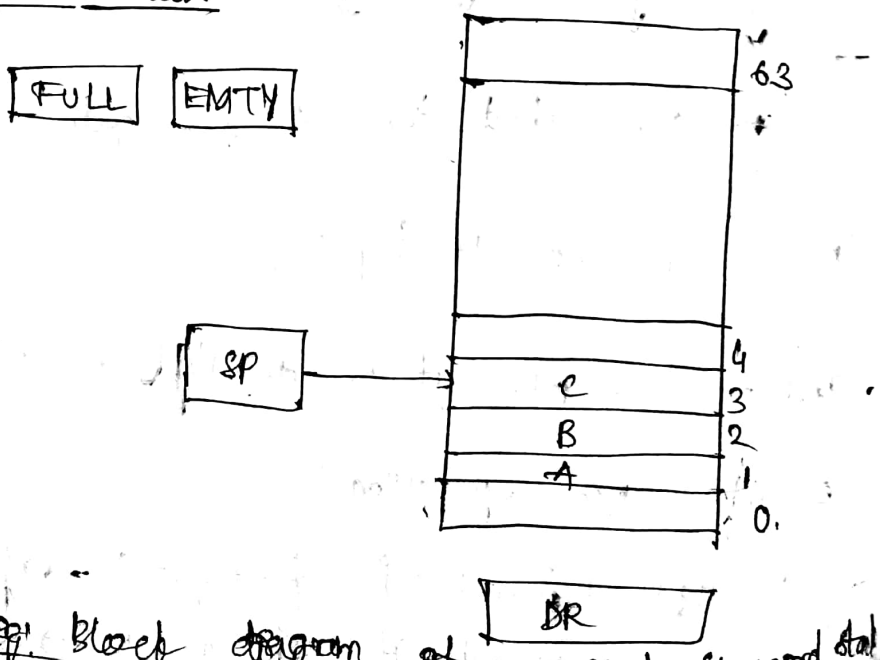


Fig: Block diagram of a 64-word stack

Memory Stack:

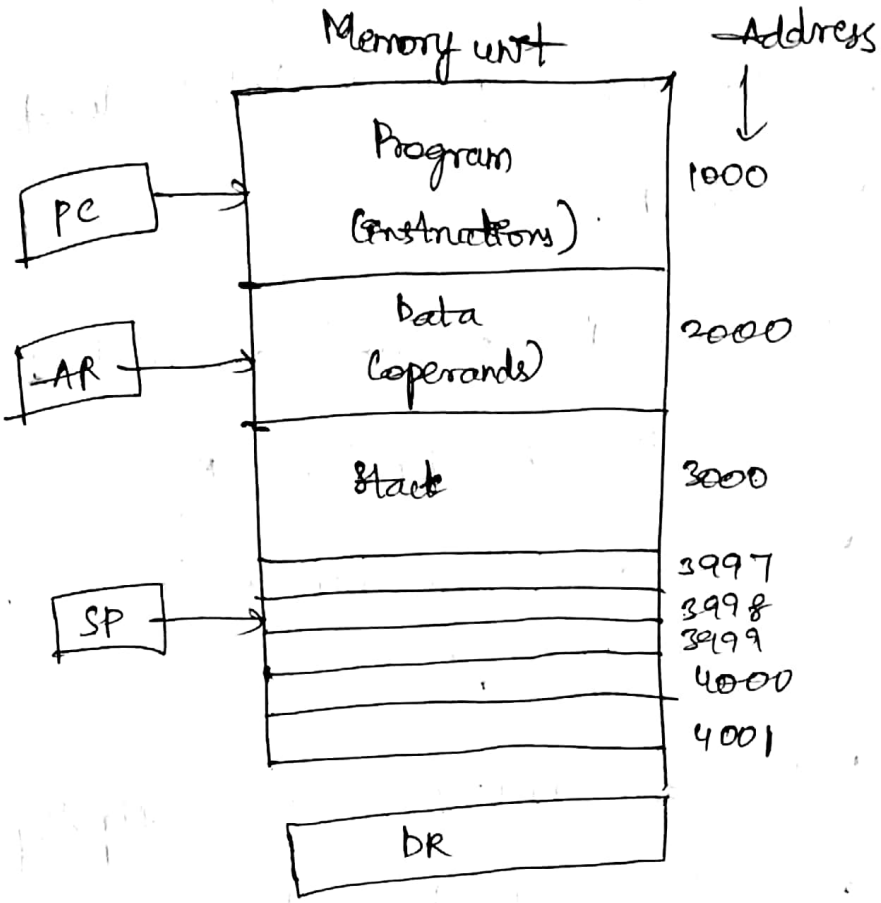


Fig. Computer memory with program, data and stack segments

→ Stack is a storage device which follows last in fast out

→ Register stack is a stand alone type Memory it will be part of RAM (Random Access memory)

SP → stack pointer is always used to indicate the top most filled location of the stack.

→ Data can be returned or read from the stack through DR (Data Register)

(∵ DR contains item that is to be returned or read from stack)

→ FULL and EMPTY are the flip flops and FULL will be set when stack is full and EMPTY " " " " " empty

→ Initially 1, 2, ... 63 locations are filled and then 0 is filled. ∴ First item is taken from 0, then 63, 62 1

→ 6-bit address is required to represent 64 addresses.

To perform push operation.

push:

~~SP ← SP+1~~

After incrementing SP, data is transfered

from DR to stack

SP ← SP+1

M[SP] ← DR

If (SP=0) then (FULL ← 1)

EMTY ← 0

SP ← SP+1

M[SP] ← DR

If (SP=0) then SP full

content of ~~data~~ DR is transfered to the memory word in the address specified by SP.

Pop:

DR ← M[SP]

SP ← SP-1

→ First item from SP is removed and SP is decremented.

DR ← M[SP]

SP ← SP-1

If (SP=0) then (EMTY ← 1)

FULL ← 0

Memory stack:

→ Memory unit contains program, data and stack.

→ Initially, ~~sp~~ SP contains (4001) and to insert an item into stack, SP is decremented

Push:

SP ← SP-1

M[SP] ← DR

Pop:

DR ← M[SP]

SP ← SP+1

Reverse Polish Notation:

A+B ← infix (or polish

+AB ← prefix (or polish

AB+ ← postfix (or reverse polish

Ex: (1) A*B+C*D → arithmetic expression in infix notation.

A*B

AB*CD*+ → reverse polish notation

Ex: (2) (A+B)*[C*(D+E)+F] infix

Follow operational hierarchy to convert AB+CDE+F*+ into post fix

1) Inside inner parantheses

2) Inside the outer parantheses

3) * (or) /

4) + (or) −

AB+ DE+ C*F+ *

③ (3*4) + (5*6)

34*56*+

→ Reverse polish notation is used to evaluate arithmetic expressions.

→ 3 and 4 are * popped from the stack and multiplied and stored in the stack

Ly, 5 & 6 are popped from the stack and multiplied and stored in the stack. The results are then added.

Instruction Formats:

→ Instruction format consists of mode, operational code and address fields. Address fields contains address of operand or process register address

→ The no. of address fields in the instruction format depends on

1) Organisation of its internal register

2) Most of the computers fall into one of 3 types of CPU organisations.

a) Single accumulator organisation

b) General register " "

c) Stack organisation

3) Example of accumulator type organisation is basic computer.

The instruction format in this type of computer contains one address stream

Ex: ADD X → AC ← AC + M[X]
↳ address of operand

b) General register type computer contains two or 3 address fields

Instruction format of

c) Stack organised CPU contains one address field or no address field

Ex for a) ADDX → AC ← AC + M[X]

content in X is added to AC and result is stored in AC

X → address, it specifies address or process register

Ex for b) ADD R_1, R_2, R_3 → $R_1 \leftarrow R_2 + R_3$

R_1, R_2, R_3 are called processor registers

ADD R_1, X → $R_1 \leftarrow R_1 + M[X]$

X → memory address

R_1 → processor register

Ex for c) PUSH X

X → memory address

→ pushes the item from memory location to stack

ADD

ADD

→ It adds top 2 items of the stack

(1) $X = (A + B) * (C + D)$

A, B, C, D are memory addresses

Three address Instructions

ADD R_1, A, B $R_1 \leftarrow M[A] + M[B]$

ADD R_2, C, D $R_2 \leftarrow M[C] + M[D]$

↳ process registers

MUL X, R1, R2 $M[X] \leftarrow R1 * R2$

Two Address Instructions.

~~ADD A B~~

ADD A B $\rightarrow M[A] \leftarrow M[A] + M[B]$

ADD C D $\rightarrow M[C] \leftarrow M[C] + M[D]$

MUL AC $\rightarrow M[A] \leftarrow M[C] * M[A]$

9/8/17

Addressing Modes:

→ Addressing mode specifies the way operand or effective address is determined (an instruction)

The Mode field indicates the addressing mode

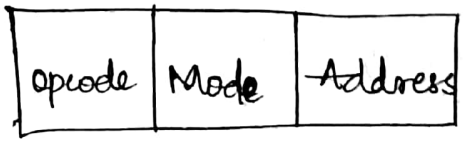


Fig. Instruction format with mode field

There are different addressing modes.

1) Implied mode →

In this mode, the definition of

instruction specifies the location of operand.

→ In instruction CMA (complement Ac), operand in Ac is complemented. In CMA, location of operand need not be specified. (no address field.)

→ ∴ In implied mode, address field does not exist.

2) Immediate mode:

Operand is specified in instruction in immediate mode.

3) Register mode:

In this mode, operand is present in the register specified in instruction.

4) Register indirect mode:

In this mode, the address of the operand is contained in the register specified in the instruction.

5) Autoincrement or Autodecrement mode:

comes under register indirect mode

In autoincrement mode, register is incremented after the execution of instruction

In auto decrement mode, register is decremented (before) prior to the execution of instruction

6) Direct Address mode:

→ In this mode, the address part of the instruction specifies effective address.

7) Indirect Address mode:

→ In this mode, address part of the instruction ~~specifies~~ ~~contains~~ indicates address of effective address.

i.e. address specified in the address part of the instruction gives effective address

8) Relative Address mode:

→ In this mode, effective address is given by adding the content of the PC to the address part of the instruction.

4) Indexed addressing mode:

→ In this mode, effective address is given by adding the content of the Index Register to the address part of the instruction.

5) Base Register Addressing mode:

→ In this mode, effective address is given by adding the content of base register to the address part of the instruction.

Example for Addressing modes:

Initially PC=200,

Instruction in 200 is fetched

location of operand is determind by addressing mode and is loaded into AC.

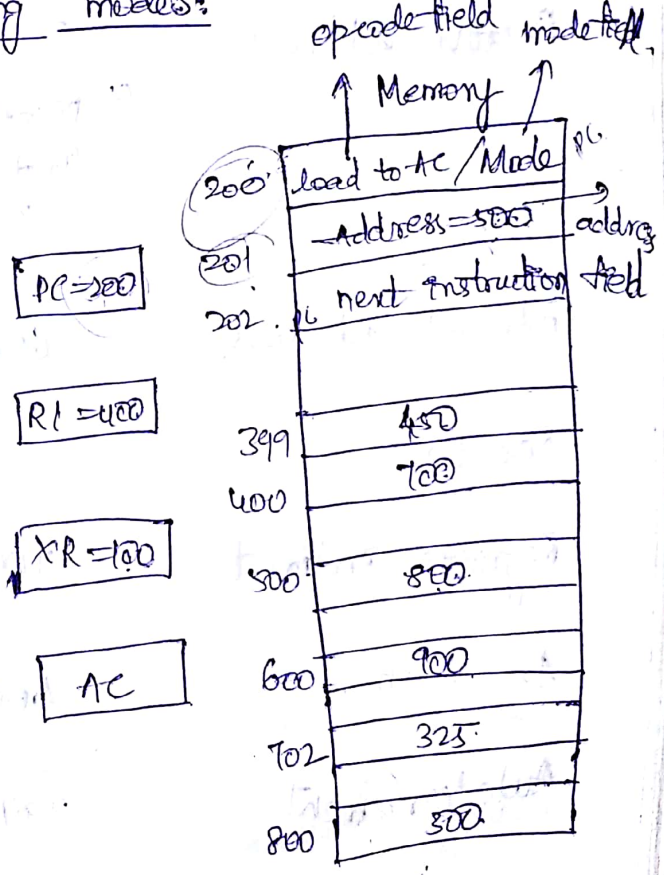


fig. Numerical example for addressing

R1 → processor register

XR → index register

Addressing Mode	Effective Address	Content of Ac
Direct Address	500	800
~~Indirect Address~~	~~800~~	~~300~~
Immediate operand	201	500
Indirect Address	800	300
Relative address	702 (∵ PC contains 202 (next instruct) 202 + 800)	325
Indexed address	600	900
Register	—	400
Register Indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

In immediate operand, address field becomes operand field.

10/8/17

Data Transfer and Manipulation:

Computer instructions are classified into 3 types

1) Data Transfer Instructions
2) Data manipulation "
3) Program control "

1) Data Transfer Instructions:

Mnemonic for load instruction

→ LD (Load)

Load instruction transfer data from memory to processor register

→ ST (Store)

Mnemonic for store

transfers data from processor register to memory.

→ MOV (move)

Transfers data b/w registers, b/w register and memory, b/w two memory words

→ XCH → exchange operation

Data is swapped b/w registers and b/w register and memory word.

→ IN (input)

→ OUT (o/p)

I/p and o/p instructions are used to transfer the data b/w processor registers and I/p (or) o/p terminals

→ ~~PUSH~~

→ PUSH

→ POP

These are used to transfer the data b/w processor registers and memory stack

2) Data Manipulation Instructions:

They are further classified into 3 types.

a) Arithmetic instructions

b) Logical and bit manipulation instructions

c) Shift instructions.

a) Arithmetic instructions

INC (increment) (+1)

DEC (decrement (-1))

ADD (add)

SUB (subtract)

MUL (multiply)

DIV (divide)

ADC (Add with carry)

SUBB (subtract with borrow)

NEG (Negate) - 2's complement)

b) Logical and bit manipulation instructions:

CLR (clear) → clears the operand

COM → complement the operand - 1's complement

AND

OR

XOR

CLRC → clear carry (i.e. E - FF) (carry = 0)

SETC → set carry (carry = 1)

CMC → complement carry

EI → enable interrupt

DI → disable "

c) Shift instructions

SHR → logical shift right

SHL → " " left

SHRA → Arithmatic shift right

SHLA → " " left

circular shift is also called rotate

ROR → Rotate right (LSB to MSB) *

ROL → " left

RORC → Rotate through carry.
(LSB to carry flag and carry to MSB)

ROLC → Rotate left to carry *
(MSB → carry and carry → LSB

3) Program Control Instructions: *

* BR → branch instruction

Executing instruction in fetch branching to specified address

2 types of branching

conditional branching → branching takes place if the condition is true otherwise next instruction is executed.

Unconditional branching → no condition. Branches to specified address.

BR
JMP } They are interchangebly used with different addressing modes. Both have same operation.

*SKP → skip

skip the next instruction.

They are of 2 types.

conditional skip and unconditional skip.

* CALL → to call

When subroutine is to be called into main program, CALL is used.

*RET

It is used to return to the main program after executing subroutine. It is the last instruction of subroutine.

* CMP → compare instruction

It is used to subtract two operands but the result is not stored.

Based on CMP operation, (result) status bits are effected (status bits → 0 bit, sign bit, carry bit, overflow bit) In subtract, result is stored and status bits are effected.

* TST

Status bits changes for any operation.

* TST - test instruction → logical AND

It perform logical AND but result is not stored and status bits are effected.

11/8/17

Program Interrupt:

→ Interrupt will cause break in the normal execution of program.

When there is an interrupt, it executes interrupt service routine (ISR)

Address of ISR is determined by

hardware.

At the end of execution phase of each instruction, control checks whether there is an interrupt. If there is an interrupt, it goes to interrupt cycle, which saves the state of CPU.

State of CPU is given by 1) content of PC, 2) content of all processor registers, 3) content of status registers. (content PSW (Program Status Words) These are saved into stack

PSW is a collection of status bits. (status bits corresponding to ALU operation are 0 bit, sign bit, carry bit, overflow bit).

Another status bit specifies the interrupts that are allowed to occur.

We also have status bit which indicates whether the CPU is in supervisor mode or user mode.

Supervisor mode: When the CPU is executing part of the program of the operating syste

It is said to be operating in supervisor mode.

User mode: When the CPU is executing user program, (written by user) it is said to be operating in user mode.

→ Address of ISR is loaded after saving the state of CPU into status register

New PSW is loaded after loading the address of PSW.

→ After Interrupt cycle, ISR is executed

→ After the execution of ISR, control should go to the main program (by getting the return address from stack)

At the end of ISR (ISR program), we have instruction called as return from interrupt. When this is executed, the contents of the stack are popped (return address,

content of processor registers and status register)

Then return address is loaded into PC and old PSW is " " status register.

Types of Interrupts:

1) External Interrupt
2) Internal Interrupt
3) software Interrupt.

1) External Interrupt:

It is due to external source.

Ex:1) I/O device requesting for data transfer.

2) Timeout Interrupt (may come from a program that is in an endless loop) and exceeded its time allocation) or when time has been elapsed for an event)

2) Power failure interrupt

When it occurs, ISR will be transfering the state of CPU into a nondestructive memory before the power seizes.)

2) Internal Interrupt:

It is caused by illegal or erroneous use of instructions or data

Ex: (i) Stack overflow (stack is full)

(ii) Register overflow.

(iii) Attempt to divide by '0'.

(iv) Invalid operation code

When there is an internal interrupt, source program (ISR) will determine the corrective methods to be taken.

3) Software Interrupt:

It is initiated by executing an instruction.

When

Inorder to get go to supervisor mode (complex I/O transfer), software interrupt is used.

Reduced Instruction Set Computer (RISC):

There are 2 types of Instruction set computer 1) Complex Instruction set Computer (CI

2) Reduced " " "

characteristics of CISC

1) A large no. of instructions typically from 100-250
2) Some instructions that perform specialised task and are used frequently.
3) A large variety of addressing modes typically from 5-20 diff. modes (depends on compiler)
4) Variable length Instruction format.
5) Instructions that manipulates operands in memory.

Characteristics of RISC :

1) Relatively few instructions
2) Relatively few addressing modes
3) Memory access limited to load and store instruction. (no manipulation)
4) All operations done within the registers of CPU
5) Fixed length, easily decoded instruction format.
6) Single cycle instruction execution
7) It uses hardwired rather than microprogram control

12/8/14

Microprogrammed control:

Control Memory:

Computer consists of 2 control units:

control unit generates control signals to perform any operation.

1) Control unit
2) Microprogram control unit.

Program is stored in main memory

Ay, Micro program (set of microinstructions) is stored in micro program control unit. to execute it.

Microinstruction:

→ It consists of control coord and next address information.

→ Control word specifies one or more microoperations

→ Next address information is used to generate the next address, where next address

indicates the address of the microinstruction that is to be executed next.

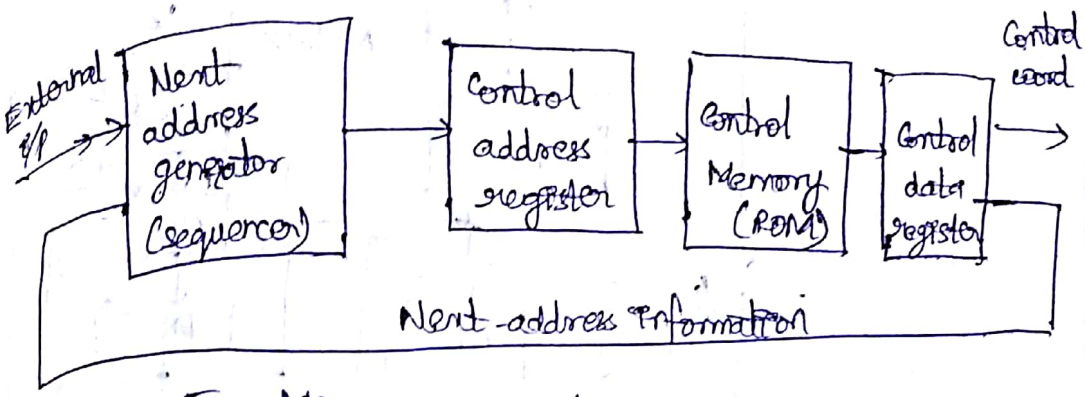


Fig. Microprogrammed control organisation.

control address register contains the address of microinstruction that is to be read by ROM. Control address register gives the memory (micro-instruction) to control Memory (generally ROM) and it i.e reads the microinstruction from the control memory which is loaded into control data register. control data register contains microinstruction which consists of control word and next-address instruction.

Microprogram Example

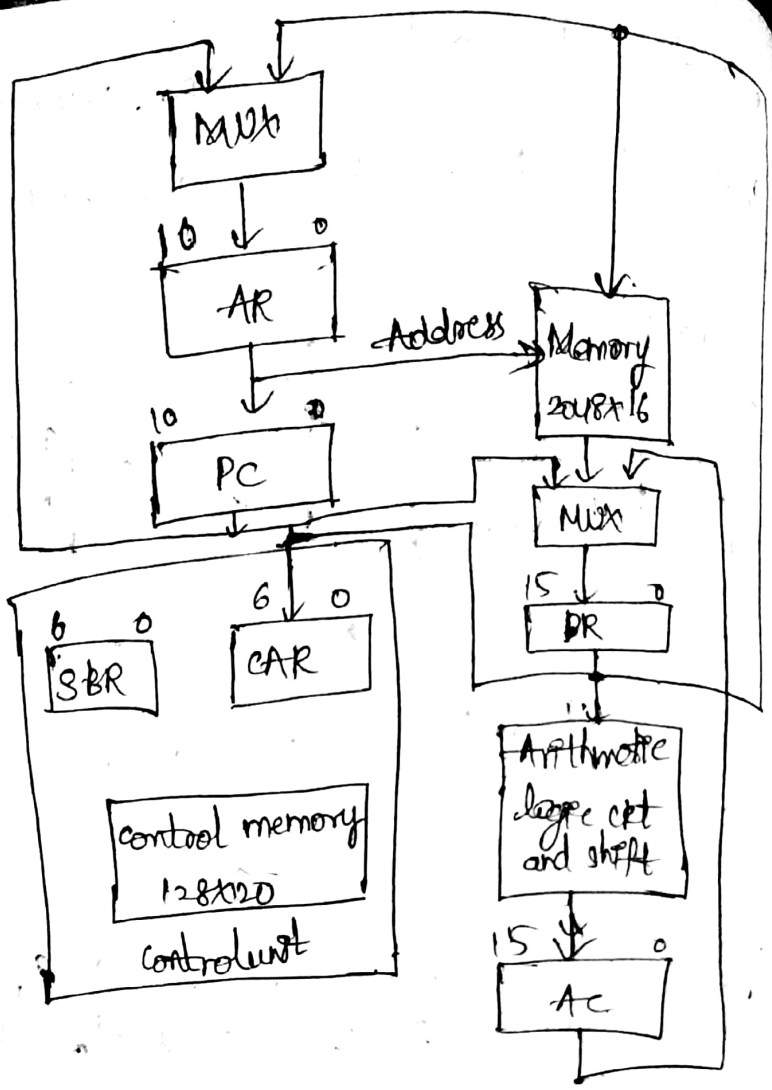


Fig: Computer hardware configuration

Microprogram control unit consists of control memory. Main memory contains 2048 locations (16 bits). Control " " 128 "

(20 bits)

→ DR contains data that is read/to be written into main memory.

∴ size = 16 bits.

→ AC and DR are used to perform arithmetic, logic and shift operation when

result is stored in AC. (Size of AC = 16)

2048 location = 2^{11}

→ 11 bits are required for PC as PC contains address of next instruction.

→ AR provides address to the main memory

→ Data is transfered from one register to another using MUX.

→ DR receives data from PC, memory, AC.

→ AR " " " PC, DR

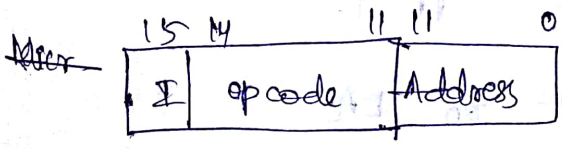
Control unit contains 2 register

1) SBR - Subroutine Register

It is used to hold the return address

2) CAR - control Address Register

It is used contains the address of microinstruction that is to be read by Control memory



(a) Instruction format.

It consists of 3 fields i.e. Mode, opcode, Address fields.

I = 0 → direct address → effective address in instruction

I = 1 → indirect address → address in memory.

opcodes is of 4 bits i.e. $2^4 = 16$ operation.

It is a memory reference instruction

i.e. contains address

Out of 16 opcodes, only 4 are defined as memory instruction

Symbol	opcode	Description.
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	If (AC < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

Exchange means swapping the content of AC and content in EA

ADD instruction adds the content of AC and operand in effective address and the result is stored in AC. (AC and DR)

→ Branch → jumping to effective address.
It takes place when AC is -ve (conditional branching). EA is loaded into PC.

→ STORE → storing the content of AC into EA

Microinstruction Format:

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

F1, F2, F3 : Microoperation fields → specify microoperation

CD: Condition for branching

BR: Branch field

AD: Address field

Fig. Instruction code format (20 bits)

Each location of control memory contains one microinstruction (128×20 bits)

∴ Size of microinstruction = 20 bits.

9 bits (F1, F2, F3) specify microoperation

~~F1 contains 000 & 111~~

F1 specifies 7 microoperations (000 no operation)

F2 " " " " "

F3 " " " (000 - no operation)

∴ Total = 21 microoperations.

Eg: For some microinstructions,

F1 - no operation, F2, F3 - some microoperation

can perform maximum of 3 "

→ CD indicates condition for branching

→ BR " type of branch

→ AC specifies branch address

Symbols and Binary codes for Microoperation fields (Instruction)

F1	Microoperation	Symbol
000	No operation	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

- Microprogram is stored in control memory in the form of binary
- Always AR contains effective address
- 101 → 0-10 bits in DR is transfered to AR

F_2	Microoperation	Symbol
000	None	NOP
001	AC ← AC − DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	AC ↑ DR
110	DR ← DR + 1	INC DR
111	DR(0-10) ← PC	PC ↑ DR

READ → Memory word from AR is transfered to DR

F_3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← $\overline{AC}$	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INC PC
110	PC ← AR	AR TPC
111	Reserved	

111 → Not yet defined

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

Unconditional indicates condition is always true.

DR(15) indicates 'I' bit (Instruction is loaded into DR ∴ 15th bit in DR → I bit

10 → check whether AC is -ve

MSB of AC = 1 ⇒ -ve AC is -ve

BR	Symbol	Function
00	JMP	CAR ← AD if condition=1 CAR ← CAR+1 if condition=0
01	CALL	CAR ← AD, SBR ← CAR+1 if condition=1 CAR ← CAR+1 if =0
10	RET	CAR ← SBR (Return from Subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

00 → CAR ← AD
↓
Address field

CAR contains the address of microinstruction that is to be executed.

01 → CAR ← AD, SBR ← CAR + 1

AD field is loaded into CAR.

Return address is stored in sub Routine register

10 → RET CAR ← SBR

Returns from SBR to main program.

JMP

Branching takes place when the condition is true and CAR should be loaded with address of microinstruction that is to be executed after branching.

→ If the condition is false, CAR should go to the next microinstruction after jump whose address is obtained by incrementing CAR

Diff b/w JMP and CALL → Return address is not saved in JMP.

CALL:

When CALL is executed, it saves the return address in SBR and SBR is executed ~~when~~ when the condition is true.

When the condition is false, CAR is incremented. AD → Address of subroutine Always return address of microinstruction that is after CALL

RET:

Return address from SBR is transfered to CAR.

MAP:

For each instruction in program, there are ~~microinstr~~ microinstructions in microprogram.

To get address of first microinstruction corresponding to some instruction is given by putting one '0' before opcode and two '0's after opcode (∵ Total = 7) (∵ 4+3)

Ex: 0000000 indicates address of the first microinstruction corresponding to add operation.

7 bits are used as the control memory contains 128 locations ($2^7 = 128$)

This is called mapping i.e. converting 4-bit opcode to 7-bit address.

opcode DR(11-14) is transfered to AR(2-5) and 0's are transfered to 0, 1, 6th bits.

18/8/14

The Fetch Routine:

→ Fetch

Reading the instruction from memory.

Main program contains main program.

Control memory ll micro program.

→ Microinstructions are required to perform fetch operation also.

PC contains address of next instruction. But in order to access the same location in memory, address is given AR (i.e. AR is connected to memory not PC)

1) Register transfer Representation:

* AR ← PC → stored in location 64 of control memory.

* DR ← M[AR], PC ← PC+1

Memory word specified by AR is transfered to DR and PC is incremented

* AR ← DR(0-10) CAR(2-5) ← DR(11-14), CAR(0, 1, 6) ← 0

Address bits in DR is transfered to AR.

If it is direct address, operand AR contains address of operand.

If it is indirect address, AR contains effective address

CAR gives the address of location to be accessed from control memory

DR(11-14) → opcode bits.

These are register transfer representation of

There are 3 types of representation

1) Register transfer representation

2) Symbolic

3) Binary

2) Symbolic Representation of microinstruction of fetch routine:

ORG 64

↓

pseudo instruction

64 — address of first microinstruction of fetch routine

```
* FETCH  F1 F2 F3   CD   BR    AD
        : PCTAR     U    JMP   NEXT
```

↓

Label indicating fetch routine

U → symbol for unconditional branching

JMP → Jump to next microinstruction

NEXT → symbol for address of next microinstruction.

(JMP address in binary → NEXT → 64 = 1000000)

~~FETCH : ORG 64~~

ORG 64

FETCH : PCTAR U JMP NEXT (65)

READ, INCPC U JMP NEXT (66)

~~DRT~~

DRTAR U MAR

CALL and MAP are unconditional

JMP and RET can be conditional also.

MAP gives the first microinstruction

3) Binary Representation

F_1, F_2, F_3 represents microoperations

For MAP and RET, address is not defined

(∵ DR(11, 14) → CAR(2-5), CAR(0, 1, 6) → 0)

(In RET, AR ← SBR i.e. Return address is saved in SBR during CALL execution)

Binary Address	F_1	F_2	F_3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

In ~~first~~ two microinstruction, address in instruction is transferred to CAR, but in MAP, CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0