* E] Branching inst^n or program control transfer inst^n:

Initially all the inst^n codes are present in memory. To execute any inst^n, first µp has to read the inst^n codes from mem. For reading inst^n code from memory, µp will transfer mem. loc addr. from PC to address bus.

Normally, the address in PC is automatically incremented by one, so µp will read inst^n code in sequence from successive memory loc^n's. This is called normal sequence of reading inst^n code from memory.

If a new 16-bit no. is transferred into PC, then the old no. of PC is lost. Now, µp will read next inst^n code from this new address which is transferred into PC and it is called Branching Operation.

So for branching operat^n, we have to transfer new 16-bit no. into PC.

There are two types of branching oper^n:

(a) Only Branching:

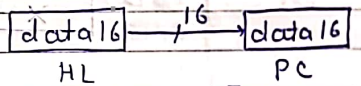
If the old no. of PC is not saved into stack mem. and new 16-bit no. is directly transferred into PC then µp will branch and read next inst^n code from this new address which is transferred into PC. But as the old no. of PC is not saved so µp can't return back to the old address. Hence, it is called only branching.

(b) Branching and Returning back:

If old no. of PC is first stored into stack memory (PUSH PC) and then new no. is transferred into PC, then µp will branch and read next inst^n code from this new address transfer into PC. But as the old no. of PC is saved in stack memory, so whenever µp performs POP PC, then µp will return back to the old address. Hence, it is called branching and returning back.

The different branching inst^n's are:

(1) PCHL ; [Transfer 16 bit no. of reg. pair HL into PC].

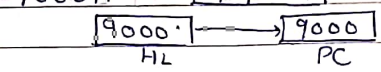


[RDAM] [I-BI] [INFAC]

Note: µp in PCHL inst^n, the old no. of PC is not saved and new no. of reg. pair HL is directly transferred into PC, so it is only branching.

Ex(1): Transfer 9000H into PC.

LXI H, 9000H



After PCHL, µp will read next inst^n code from address 9000H which is transferred into PC.

(2) Transfer 16 bit no. of DE pair into PC.

XCHG ; DE → HL

PCHL ; HL → PC

(3) Transfer 16 bit no. of TOS, TOS+1 into PC.

POP H

PCHL.

(4) Transfer 16-bit no. of SP into PC.

LXI H, 0000H ; H: 00 L: 00

DAD SP + X SP

PCHL. X HL → X PC

2] JMP Addr 16 ; [Unconditional jump to the target instn]

[Similar to GOTO statement]

Addr 16 → Addr 16 PC

(only branching)

[IAM] [3-BI] [NFAC]

Ex: Transfer 9000H into PC by Two method

→ (i) LXI H, 9000H

PCHL ; 9000 (HL) → 9000 (PC)

(ii) JMP 9000H ; 9000H → 9000H PC

(Better method).

Give the sequence of operation for the instn:

JMP 5000H

→

	mem
2146	X
2147	JMP
2148	00
2149	50
214A	4
5000	Z
5001	

JMP 5000H

Forward Branching

Target instn

(1) Opcode fetch and decode:

2147 | mem JMP → S → JMP IR

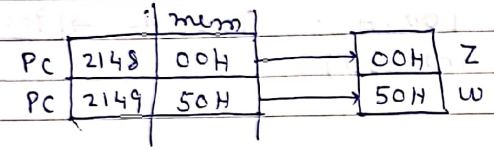
PC 2147 +1 → 2148 PC

Opcode fetch

The opcode of JMP instn is transferred

from IR to ID and JMP inst^n unit is selected. (decoding)

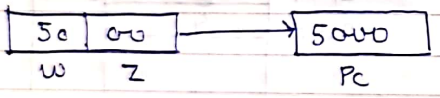
(2) Operand fetch (2nd & 3rd byte)



PC | 2149H | +1 → | 214AH | PC

(3) Execution of Branching operation

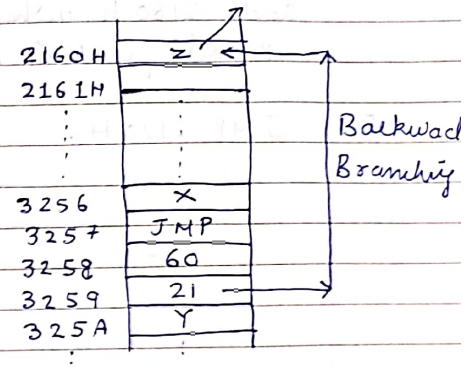
The address 5000H given with JMP inst^n is directly transferred from wz to Pc.



So, µp will jump and read next inst^n code Z from, address 5000H which is transferred into Pc conly branching.

Ex: JMP 2160H

Target inst^n



Sequence of Operation:

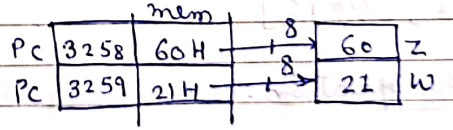
(1) Opcode fetch and decode -

PC | 3257 | JMP | —1B→ | JMP | IR

PC | 3257 | +1 → | 3258 | PC

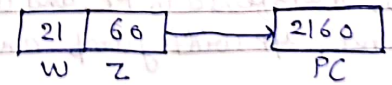
The opcode of JMP inst^n is transferred from IR to ID and JMP inst^n unit is selected (decoding).

(2) Operand fetch (2nd & 3rd byte):



PC | 3259 | +1 → | 325A | PC

(3) Execution of branching:



Now, µp will read next inst^n code from addr. 2160 H which is transferred into PC i.e. µp will jump & execute target inst^n Z.

Ex: JMP 5120 H.

(3) J condition Addr 16 ; [Conditional jump to the target inst^n]
[Similar to IF--THEN statement]

Only if given cond^n is satisfied then,

Addr 16 ⟶ [Addr 16] → PC

[IAM] [3-BI] [NFAC]

Sequence of Operation:

1. Cond^n satisfied (jump) -

If given cond^n is satisfied, then µp will execute like JMP inst^n i.e new 16-bit address given in the inst^n is transferred to PC and the old no. of PC is lost. Hence, µp will jump and read next inst^n code from this new address transferred into PC i.e Branching operation

2. Cond^n not satisfied (Normal operation)

If given cond^n is not satisfied, then new 16-bit address is not transferred into PC, the old no. of PC is not lost. Hence, µp will not jump, instead µp will read next inst^n code in sequence i.e normal operation

The different conditional jump inst^{n}s are ~~given~~ given below:

(a) JC addr 16 ; [Jump if carry]
If CF = 1, then µp will jump or branch

(b) JNC addr 16 ; [Jump if not carry]
If CF = 0, then µp will jump or branch.

(c) JZ Addr 16 ; [Jump if result is zero]
If ZF = 1, then µp will jump or branch

(d) JNZ addr 16 ; [Jump if result is not zero]
If ZF = 0, then µp will jump or branch.

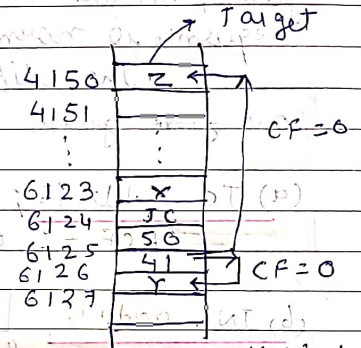
(e) JPE addr 16 ; [Jump if parity even]
If PF = 1, then µp will jump or branch

(f) JPO addr 16 ; [Jump if parity odd]
If PF = 0, then µp will jump or branch.

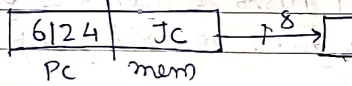
(g) JM ; [Jump if minus sign]
If SF = 1, then µp will jump or branch.

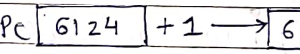
(h) JP addr 16 ; [Jump if + sign]
If SF = 0, then µp will jump or branch.

Ex: Give sequence of operation of the instn
JC 4150H



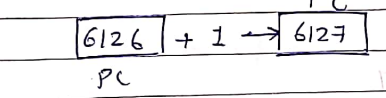
A] Sequence of operation for condn satisfied (CF=1)
(i) Opcode fetch & decode :



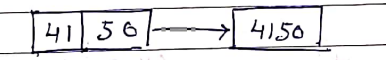


The opcode of JC instn is transfered from IR to ID & Jc instn unit is selected.

(ii) Operand fetch of 2nd byte & condn check (CF = ?) and if CF = 1 (condn satisfied, then operand fetch of 3rd byte) :

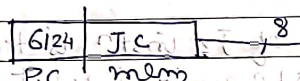


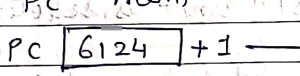
(iii) Execution or branching :



Finally, µp will jump & read next instn code Z from addr 4150 which is transfered into Pc ie branching operation.

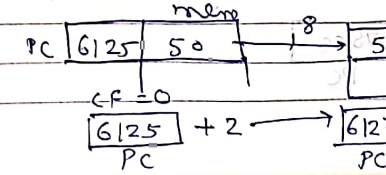
B] Sequence of operation for condn not satisfied (CF=0) :
(i) opcode fetch & decode :





The opcode of JC instn is transfered from IR to ID & JC instn unit is selected.

(ii) Operand fetch of only 2nd byte & condn check (CF = ?) & if CF = 0 (condn not satisfied) then µp will not read 3rd byte operand only address in Pc is increased by 2.



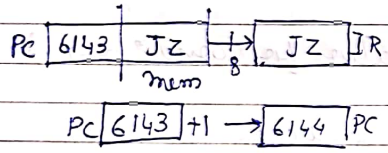
So, µp will not jump & µp will ~~fetch~~ read next inst^n Y in sequence from addrs. 6127 H.

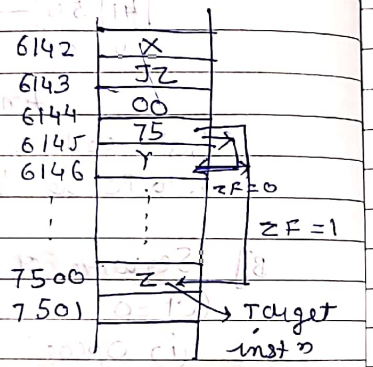
Ex: JPO 9000 H

Ex: JZ 7500 H

A] Sequence of operation for cond^n satisfied

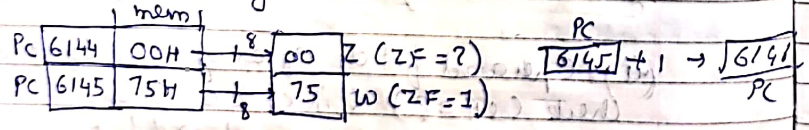
(i) Opcode fetch and decode



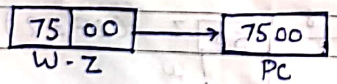


The opcode of JZ inst^n is transferred from IR to ID & JZ inst^n unit is selected (decoding).

(ii) Operand fetch of 2nd byte and cond^n check (ZF = ?) and if ZF = 1 (cond^n satisfied) then operand fetch of 3rd byte:



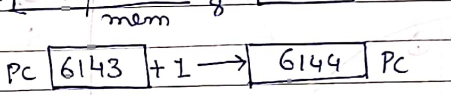
(iii) Execution or Branching operation (ZF = 1)



So, µp will jump and read next inst^n code from addr. 7500H ie µp executes target inst^n Z.

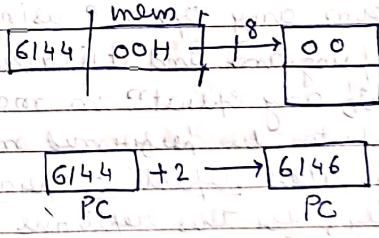
B] Sequence of operation for cond^n not satisfied:

(i) Opcode fetch & decode



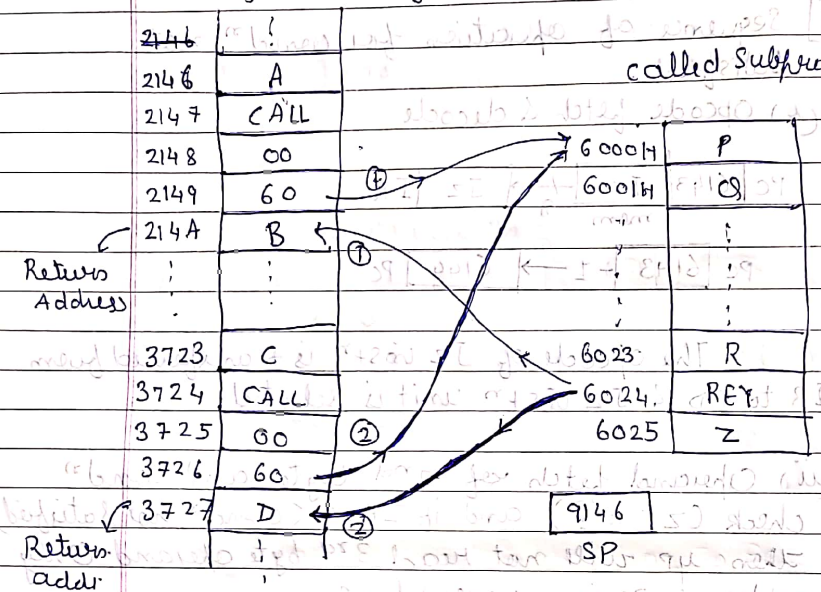
The opcode of JZ inst^n is transferred from IR to ID & JZ inst^n unit is selected.

(ii) Operand fetch of 2nd byte and cond^n check (ZF = ?) and if ZF = 0 (cond^n not satisfied) then µp will not read 3rd byte operand. Only addrs. in PC is increased by 2.



So, µp will not jump, instead µp will read next inst^n code in sequence from address 6146 H ie µp execute inst^n Y.

Subprogram / Subroutine / Functions / Procedure:



To perform any operatⁿ using µP, we have to write users program and it is called main user program. If any operatⁿ in main user program is reqd to be performed more than once, then it is called repetitive operation.

Normally, for this repetitive operation, the program is prepared only once and this program is stored in separate mem locⁿs. This program of repetitive operatⁿ is not a separate program but it is a part of main user program, hence it is called subprogram.

(a) To execute this sub program we have to give call instⁿ in the main program. When µP executes call instⁿ, then it performs two steps. The old address in PC is 1st saved into stack mem (PUSH PC).

(b) Then new sub program address is transfered into PC.

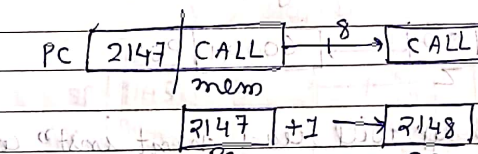
So, µP branches from main program to sub program.

After executing sub program, when µP executes RET instⁿ, then the old address of PC is transfered back to from stack mem. to PC (POP PC). So µP returns back from sub program to the next instⁿ of main program from where µP has left.

Sequence of Operation of instⁿ CALL 6000H.

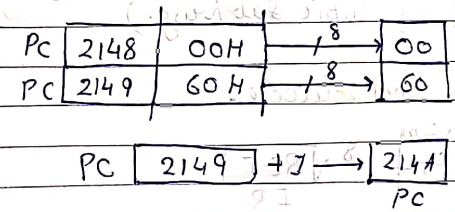
(Refer big. of topic sub program)

(1) OPcode fetch and decode



The opcode of CALL instⁿ is transferred from IR to ID and CALL instⁿ unit is selected (decoding)

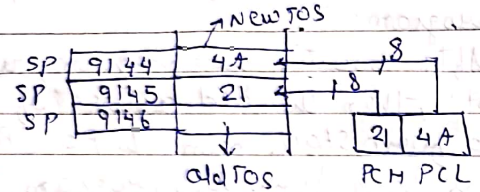
(2) Operand fetch of 2nd & 3rd byte :-



(3) Execution Operation:-

a) Saving of ~~data~~ Return address:-

The return address 214AH of next inst^n B is first saved from PC to stack mem. (PUSH PC). Only this step is not present in JMP inst^n



Note: TOS after call inst^n is equal to mem. loc^n 9144H.

(b) Branching from main prog. to Sub prog.

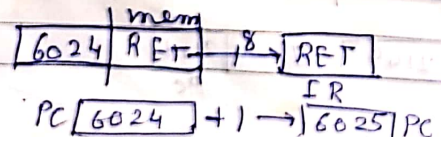
The Sub prog. address 6000H is transfered from reg. pair WZ to PC

Now, µP will read next inst^n code from address 6000H which is transfered into PC ie µP branches from main prog. to sub prog. and executes inst^n P.

Sequence of operation of RET inst^n:-

(Refer fig. of topic Sub prog.)

(i) OPcode fetch and decode:



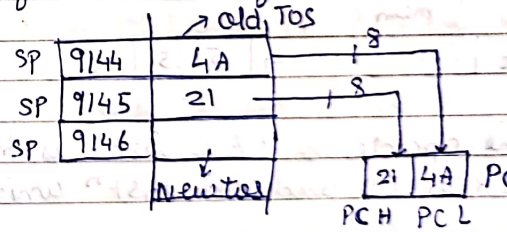
The opcode of RET inst^n is transferred from IR to ID and RET inst^n unit is selected (decoding).

Note: As RET is one byte inst^n, there is no operand fetch.

(ii) Execution of Returning back from Sub prog. to main prog.:

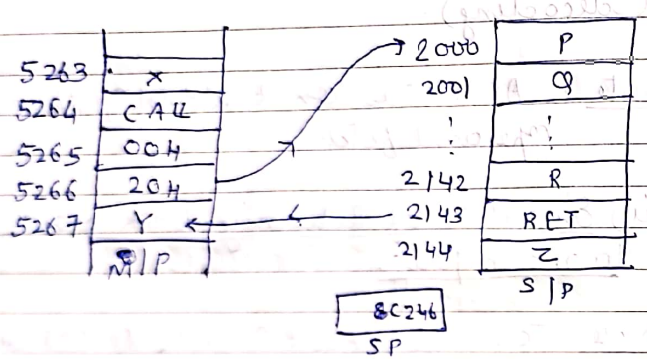
Note: TOS after CALL = 9144H, so TOS before RET inst^n should also be 9144H. To satisfy this cond^n, normally, the no. of PUSH & POP inst^{n}s in the sub prog. should be equal.

The return address 214AH saved in stack mem. during CALL inst^n is transferred back to PC (POP PC)



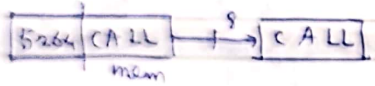
Now, µP will read next inst^n code from addrs. 214AH which is transferred into PC ie µP returns back from subprog to inst^n B of main prog. from where µP has left.

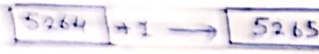
Ex: CALL 2000H and RET



Sequence of operation of CALL 2000H

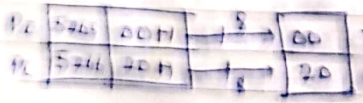
(i) Opcode fetch and decode:





The opcode of CALL inst^n is transferred from IR to ID and call inst^n unit is selected.

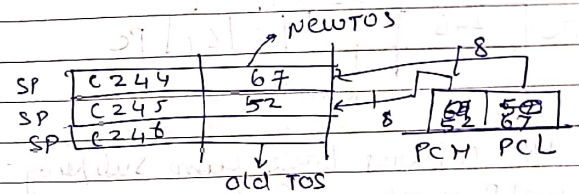
(ii) Operand fetch of 2nd and 3rd byte



(iii) Execut^n operat^n:

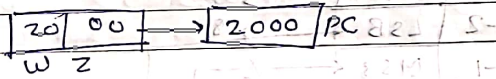
(a) Saving of return address

The return address 5267H of next inst^n Y is first saved from PC to stack mem.



(b) Branching from main prog to sub prog -

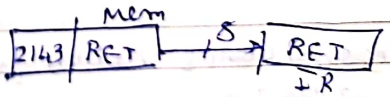
The sub. prog. address 2000H is transferred from reg. pair WZ to PC

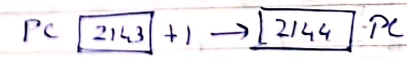


Now µP will read next inst^n code from addr. 2000H which is transferred into PC is µP branches from main prog. to sub prog & execute inst^n P

Sequence of operat^n of RET inst^n

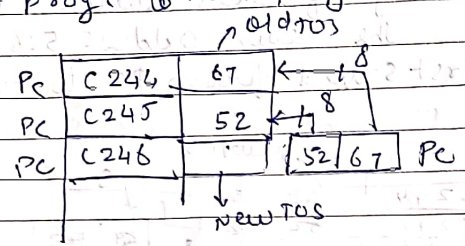
(i) op code fetch & decode





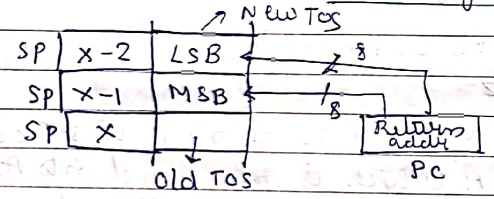
The opcode of RET inst^n is transf. from IR to ID & RET inst^n unit is selected.

(ii) Execution of returning back from sub prog. to main prog.



µp returns back from sub prog. to the next inst^n of main prog. from where µp has left.

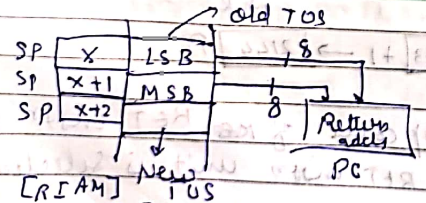
(4) CALL Addr 16 ; [Unconditional call of the sub program]



Addr 16 → Addr 16
PC

[IAM] [3-BI] [NFAC]
SP → [RIAM]

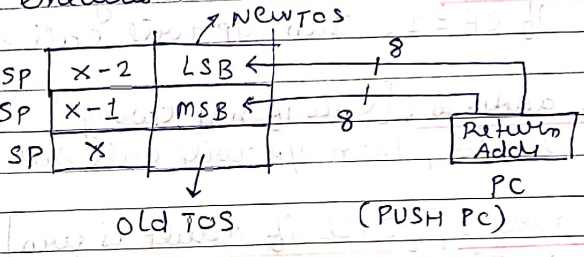
(5) RET ; [Unconditional return from sub-prog.]



PC → [RIAM]
SP → [RIAM] [I-BI] [NFAC]

(6) C condition addr 16 ; [Conditional call of the sub program].

Only if given cond^n is satisfied, then µp executes like CALL inst^n as given below:



Addr 16 → Addr 16
PC

[IAM] [3-BI] [NFAC]
SP → [RIAM]

Sequence of Operations :

(i) For cond^n satisfied - If cond^n given in the inst^n is satisfied, then µp will execute like CALL inst^n ie.

(a) The return address of next inst^n is first saved from PC to stack mem. (PUSH PC).

(b) The sub program address given in the inst^n is transfered into PC.

So, finally µp branches from main prog. to the sub program whose address is transfered into PC.

(ii) For cond^n not satisfied - If given cond^n is not satisfied, then µp will not branch from main prog. to sub prog., instead, µp will execute next

inst^n of main prog. in sequence.

The different conditional call inst^{n}s are given below:

(a) CC addr. 16 ; [CALL if Carry]
If CF = 1, then µP will call sub program

(b) CNC addr. 16 ; [CALL if not carry]
If CF=0, then µP will call sub program.

(c) CZ addr 16 ; [CALL if result is zero]
If ZF = 1, µP will call sub program.

(d) CNZ addr 16 ; [CALL if result is not zero]
If ZF=0, µP will call sub program.

(e) CPE addr 16 ; [CALL if Parity even]
If PF = 1, µP will call sub program.

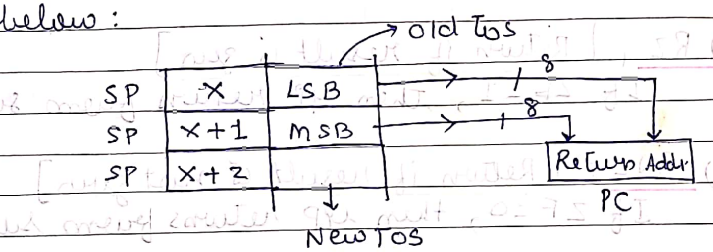
(f) CPO addr 16 ; [CALL if Parity odd]
If PF = 0, µP will call sub prog.

(g) CM addr 16 ; [CALL if minus (-) sign]
If SF = 1, µP will call sub prog.

(h) CP addr 16 ; [CALL if Plus (+) sign]
If SF=0, µP will call sub prog.

(7) Rcondition ; [Conditional Return from the Sub program]

Only if given cond^n is satisfied, then µP will execute like RET inst^n (POP PC) as given below:



S → [RIAM] [I-BI] [NFAC]
D → [RDAM]

Sequence of Operations:

(i) For cond^n satisfied - If given cond^n is satisfied, then µP will execute like RET inst^n, i.e. the return addr. saved in stack mem. during CALL inst^n is transferred back from stack mem. to PC (POP PC). So, µP returns back from sub program to the main program (branching operation).

(ii) For cond^n not satisfied - If given cond^n is not satisfied, then µP will not return from sub prog. to main prog., instead µP will execute next inst^n of sub prog. in sequence (Normal operat^n)

The different conditional Return inst^{n}s are given below:

(a) RC ; [Return if carry]
If CF = 1, then µP returns from sub-prog.

(b) RNC ; [Return if not carry]
If CF = 0, then µP returns from sub-prog.

(c) RZ ; [Return if result is zero]
If ZF = 1, then µP return from sub prog.

(d) RNZ ; [Return if result is not zero]
If ZF = 0, then µP returns from sub prog.

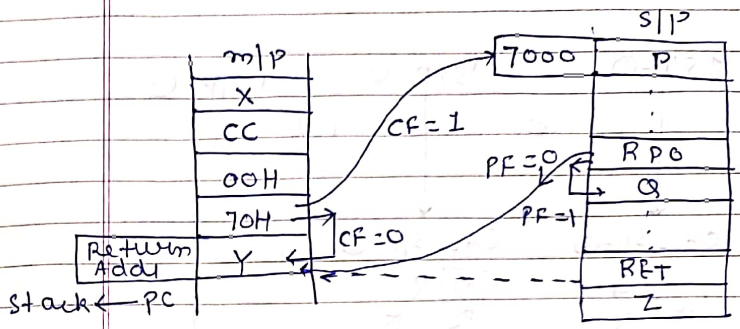
(e) RPE ; [Return if Parity even]
If PF = 1, then µP returns from sub prog.

(f) ~~RC~~ RPO ; [Return if Parity odd]
If PF = 0, then µP returns from sub prog.

(g) RM ; [Return if - (minus) sign]
If SF = 1, then µP returns from sub prog.

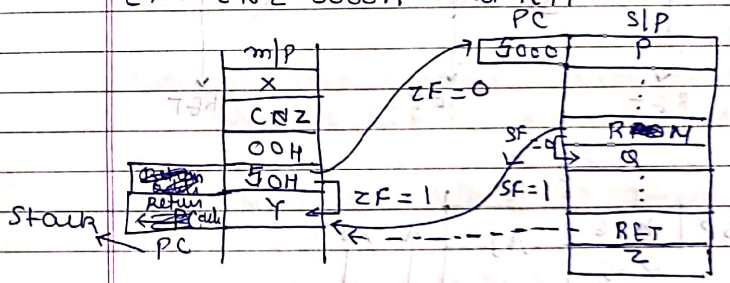
(h) RP ; [Return if + (Plus) Sign]
If SF = 0, then µP returns from sub prog.

Ex: CC 7000H and RPO

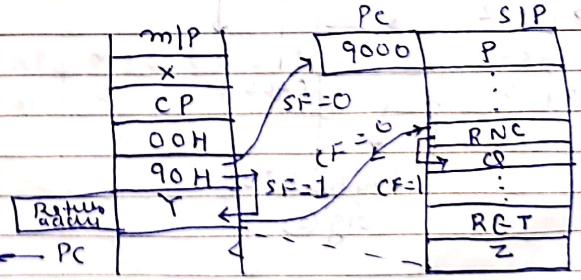


Multiending sub programs.

Ex: CNZ 5000H and RM



Ex: CP 9000H and RNC



Nested Sub programs:

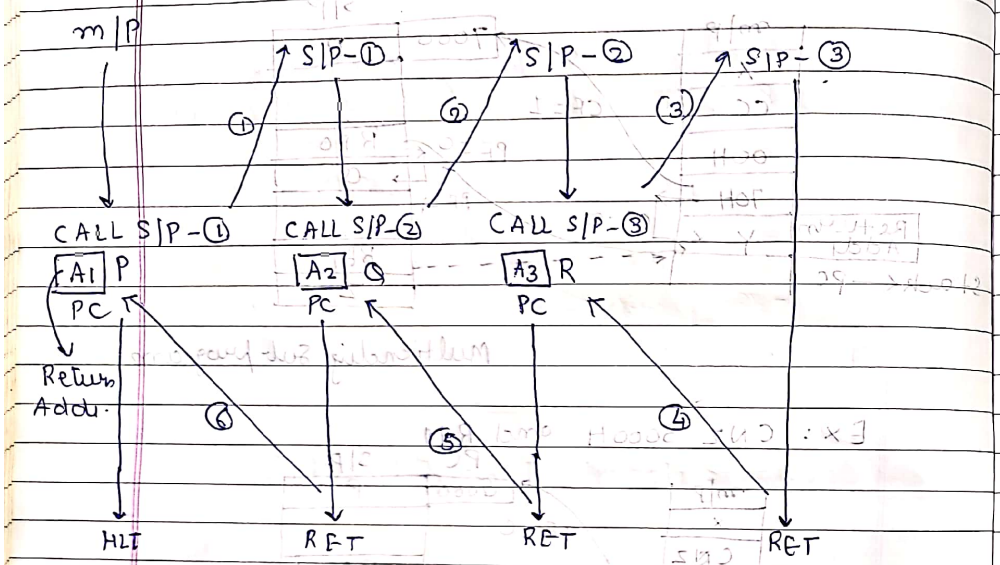


Fig (1)

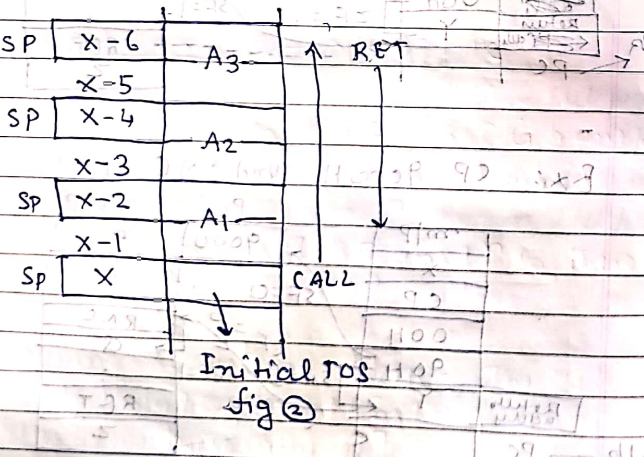


fig ②

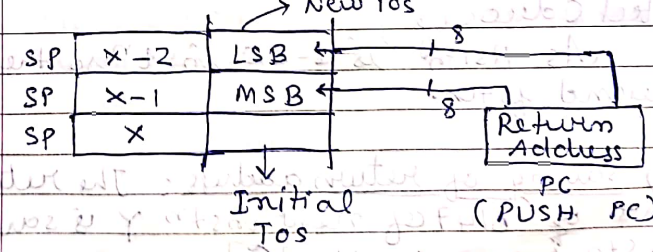
µp starts executing the program from main program. Then µp executes CALL S/P-1 instⁿ in the main prog., then µp performs two steps:
(a) The return address A_1 of next instⁿ P is first saved from PC to stack mem. (PUSH PC).
(b) Now Sub program 1 address is transferred into PC ie µp branches from m/p to S/P-1.

When µp executes CALL S/P-2 instⁿ in S/P-1 is, then µp performs two steps:
(a) The return address A_2 of next instⁿ Q is first saved from PC to stack mem (PUSH PC)
(b) Now, S/P-2 addr. is transfered into PC ie µp branches from S/P-1 to S/P-2 and so on..

If S/P-2 is called from S/P-1, then these two S/Ps are called nested sub programs.

Due to LIFO sequence of stack memory, µp will return back in sequence from SP/2 to S/P-2, from S/P-2 to S/P-1 and from S/P-1 to main program.

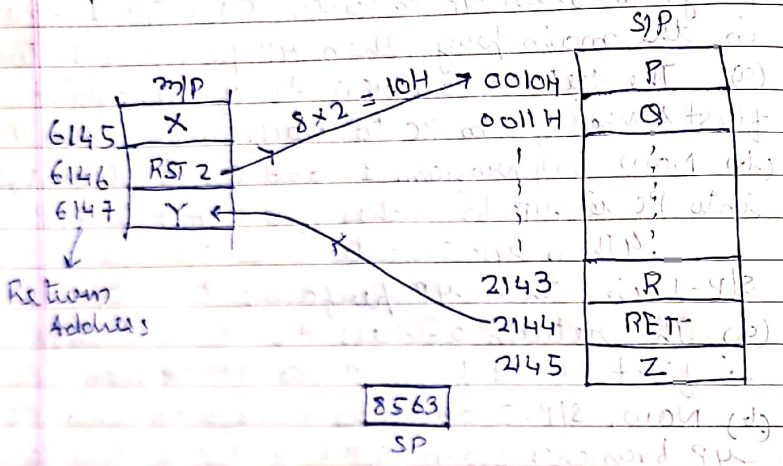
(8) RST n ; [Restart from address 8×n]
(n = 0 to 7) [Software interrupt instⁿ]



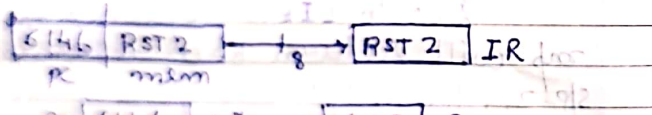
S/P Addr = 8×n → 8×n PC

SP ⟶ [RIAM] [1-BI] [NFAC]
PC ⟶ [RDAM]

Sequence of operation of RST 2 :



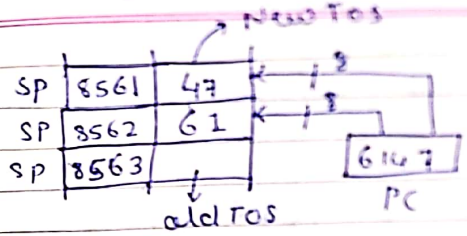
(a) Opcode fetch and decode :



The opcode of RST 2 instⁿ is transfered from IR to ID and RST 2 instⁿ unit is selected (decoding)

As RST 2 is 1-BI instⁿ so there is no operand fetch.

(b) Saving of return address : The return address 6147 of next instⁿ Y is saved from PC to stack memory [PUSH PC]



Note : TOS after RST 2 instⁿ is 8561H. So, TOS before RST instⁿ should also be 8561H

(b) Calculation of sub prog. address and branching :

When µP executes PUSH PC operation, then at the same time, µP will calculate sub prog. address using the eqⁿ 8×n. This address 8×n is saved into reg. pair WZ.

$8 \times n = 8 \times 2 = (16)_H = 0010H \rightarrow$ W: 00, Z: 10H

W − Z = 0000 0000 00 | 010 | 000 ($D_5 D_4 D_3$)

$n = 2 = (010)_B$

= 0010H

After completing the operation of PUSH PC, this sub prog. address 0010H is transfered into PC from reg. pair WZ to PC.

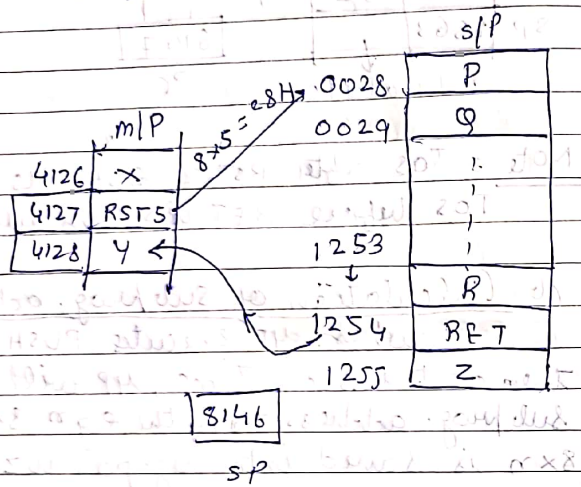
W: 00, Z: 10H → PC: 0010H

So, finally µP branches from main prog. to sub prog. at address 0010H

After executing the sub prog., when µP executes RST instⁿ, then µP performs POP PC and µP return back from sub prog to the next instⁿ Y

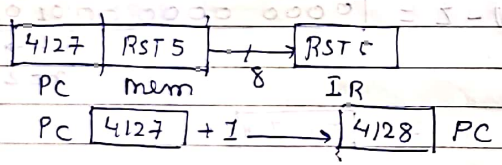
of main prog. from where µp has left.

Ex: RST 5



Sequence of Operation:

(i) Opcode fetch and decode

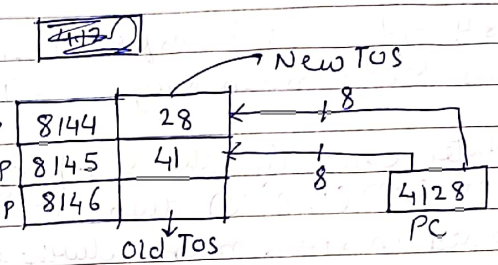


The opcode of RST 5 inst^n is transfered from IR to ID and RST 5 inst^n unit is selected (decoding)

Note: As RST 5 is 1byte inst^n, so there is no operand fetch.

(ii) Execution:

(a) Saving of return address: The return address 4128 H of next inst^n Y is first saved from R to stack mem (PUSH PC)

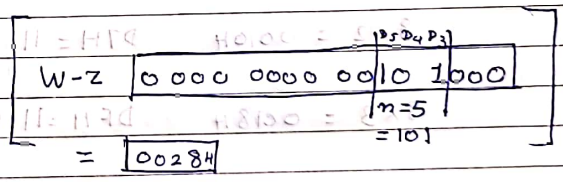


TOS after RST 5 = 8144 H, so TOS before RET inst^n should also be 8144 H

(b) Calculation of subprogram address and Branching:

When µp execute PUSH PC operat^n, then at the same time, µp will calculate subprogram address using the eq^n 8×n as given below and this subprog. address is stored in reg. pair WZ.

$8 \times n = 8 \times 5 = (40)_D = 0028\,H \rightarrow$ W-Z: 00 28



After Completing the operation of PUSH PC, µp will transfer this subprog. address 0028 H from WZ to PC.

Now µp will read new inst^n code from

addr. 0028H which is transferred into PC. ie µP branches from main prog. to S/P at addr. 0028H

After executing the sub prog., when µP execute RET instⁿ (POP PC) then the return addr. 4128H saved in stack mem. during RST 5 instⁿ is transfered back from stack mem. to PC. So after RET instⁿ, µP execute instⁿ 1 of main prog.

The table showing 8 different RST n instⁿ, their corres. sub prog. address and opcodes is given below:

RST n	S/P Addr	OPcodes
RST 0	8×0 = 0000H	C7H = 11 000 111
RST 1	8×1 = 0008H	CFH = 11 001 111
RST 2	8×2 = 0010H	D7H = 11 010 111
RST 3	8×3 = 0018H	DFH = 11 011 111
RST 4	8×4 = 0020H	E7H = 11 100 111
RST 5	8×5 = 0028H	EFH = 11 101 111
RST 6	8×6 = 0030H	F7H = 11 110 111
RST 7	8×7 = 0038H	FFH = 11 111 111

Jmp Difference betⁿ JMP, CALL, RST N

JMP addr 16	CALL addr 16	RST n
1. 3-BI	3-BI	1-BI
2. There are two operands fetch	There are two operands fetch	No operand fetch.
3. The branching addr. of target instⁿ is given.	The S/P address is given.	S/P addr is obtained using eqⁿ 8×n.
4. The branching addr. is variable from 0000H to FFFF H.	The S/P address is variable from 0000H to FFFF H.	The S/P address is fixed ie 8×n.
5. The Return addr. is not saved.	The return address is saved from PC to stack mem.	The return addr. is saved from PC to stack mem.

addr. 0028H which is transferred into PC i.e. µP branches from main prog. to S/P at addr. 0028H.

After executing the sub prog., when µP execute RET inst^n (POP PC) then the return addr. 4128H saved in stack mem. during RST 5 inst^n is transfered back from stack mem. to PC. So after RET inst^n, µP execute inst^n of main prog.

The table showing 8 different RST n inst^{ns} their corres. sub prog. address and opcodes is given below:

RST n	S/P Addr	Opcodes
RST 0	8×0 = 0000H	C7H = 11 000 111
RST 1	8×1 = 0008H	CFH = 11 001 111
RST 2	8×2 = 0010H	D7H = 11 010 111
RST 3	8×3 = 0018H	DFH = 11 011 111
RST 4	8×4 = 0020H	E7H = 11 100 111
RST 5	8×5 = 0028H	EFH = 11 101 111
RST 6	8×6 = 0030H	F7H = 11 110 111
RST 7	8×7 = 0038H	FFH = 11 111 111

Jmp Difference bet^n JMP, CALL, RST N

# JMP addr 16	CALL addr 16	RST n
1. 3-BI	3-BI	1-BI
2. There are two operands fetch	There are two operands fetch	No operand fetch.
3. The branching addr. of target inst^n is given.	The S/P address is given	S/P addr. is obtained using eq^n 8×n.
4. The branching addr. is variable from 0000H to FFFF H.	The S/P address is variable from 0000H to FFFF H.	The S/P address is fixed i.e 8×n.
5. The Return addr. is not saved.	The return address is saved from PC to stack mem.	The return addr. is saved from PC to stack mem.

Summary of Branching Instructions:

JMP Instn

1. JMP Addr 16
   Addr 16 ⟶ PC

3. JC addr 16
   If CF = 1, then
   Addr 16 ⟶ PC

5. JNC Addr 16
   (CF = 0)

7. JZ addr 16
   (CF = 1)

9. JNZ addr 16
   (ZF = 0)

11. JPE addr 16
    (PF = 1)

13. JPO addr 16
    (PF = 0)

15. JM addr 16
    (SF = 1)

17. JP addr 16
    (SF = 0)

CALL instn

2. CALL addr 16
   PUSH PC &
   Addr 16 ⟶ PC

4. CC Addr 16
   If CF = 1, then
   PUSH PC & Addr 16 ⟶ PC

6. CNC addr 16
   (CF = 0)

8. CZ addr 16
   (ZF = 1)

10. CNZ addr 16
    (ZF = 0)

12. CPE addr 16
    (PF = 1)

14. CPO addr 16
    (PF = 0)

16. CM addr 16
    (SF = 1)

18. CP addr 16
    (SF = 0)

19. PCHL
    HL ⟶ PC
    (1-BI jump instn)

21. RET
    (POP PC)

22. RC
    If CF = 1, then
    (POP PC)

24. RZ
    (ZF = 1)

26. RPE
    (PF = 1)

28. RM
    (SF = 1)

23. RNC
    (CF = 0)

25. RNZ ; ZF = 0

27. RPO
    (PF = 0)

29. RP
    (SF = 0)