Three address Instructions

ADD R1, A, B $R1 \leftarrow m[A] + m[B]$

ADD R2, C, D $R2 \leftarrow m[C] + m[D]$

MUL X, R_1, R_2 $M[X] \leftarrow R1 * R2$

09/08/17

* Addressing modes :→

* Addressing mode Specify the way operand or Effective address is ~~to~~ determined.

* The mode field in the instruction format ~~ex~~ will be indicating the addressing mode

opcode	Mode	address

* INSTRUCTION format with MODE FIELD*

① * In the implied Code the definition of the instruction will specifying the location of the operand.

* In the Instruction Compliment accumulator (CMA) the operand present in the accumulator is complimented.

↑ Ex for implied mode.

* In CMA we need not Specify the location of the operand

* In the implied mode instruction address field will not be Existing.

② * Immidiate mode :-

* In this mode operand is specefied inthe instruction.

③ * Register mode :-

* Operand is contained in the register specified in the Instruction.

④ * Register indirect mode :-

* The address of the operand is contained in the register in the specified in the Instruction.

*⑤ Auto-increment, Auto-decrement mode :- {comes under ④}

* Register is incremented after the Execution of the Instruction.

* Register is decremented Prior ~~after~~ to the Execution of the Instruction

*⑥ Direct addressing mode;→

* The address part of the Instruction indicates the Effective address.

*⑦ Indirect address mode ;→

* The address part of the Instruction Indicates the address of the Effective address.

*⑧ Relative Address mode;→

* The Effective address is given by adding the ~~ed~~ Content of the program Counter to the address part of the Instruction.

*⑨ Indexedaddressing mode ;→

* The Effective address is given by adding the Content of Index register to the address part of the Instruction.

*⑩ Base register Addressing mode ;→

* The Effective address is given by adding the Content of Base register to the address Part of the Instruction.

*1) A*B + C*D { using reverse polish notation}

Sol) AB* CD* +

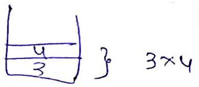
* (A+B) * [C* (D+E) + F]

*Sol) ~~AB+ [C(DE+)*F+] *~~

~~AB+~~ AB+DE+C* F+ *

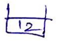
Eg:- (3*4) + (5*6)

34* 56 * 1

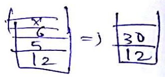


3 & 4 are inserted

as operation is multiplication, the last 2 digits are poped, they are multiplied and result is stored in Stack



⇒ 3x4 = 12



5, 6 are inserted

Illy 5, 6 are multiplied & result is stored

Illy 30, 12 are added (last 2)

∴ Sum = 42.

Instruction formats :— It contain diff fields such
-) address field, mode field, operation code field ...
↳ specifies the memory address or processor register

The no. of address fields in the Instruction formats of a computer depends on organisation of its internal registers

*Stack pointer (sp) is used to locate the top most filled location of the stack.

* Item that is read from the stack or the item that is return to the stack is contained by Data Register "$\boxed{DR}$"

* FULL = set when stack is full.

* EMPTY = SET when Stack is Empty

* In order to address the 64 locations we require 6bit address.

* Initially sp is at "0" & when Ist item is Pushed it is inverted into 1 & after the 63 rd located is filled, the ~~6th~~ next item will be filled in location "0".

* when we are removing the items from stack, the last item will be removed first.

*Operations:→

Push :

```
SP ← SP+1        {increment SP}
M[SP] ← DR       {Transfer data from DR to SP}
If [SP = 0] then [FULL ← 1]
EMPTY ← 0.
```

a & stack

@IR, [illegible] should b

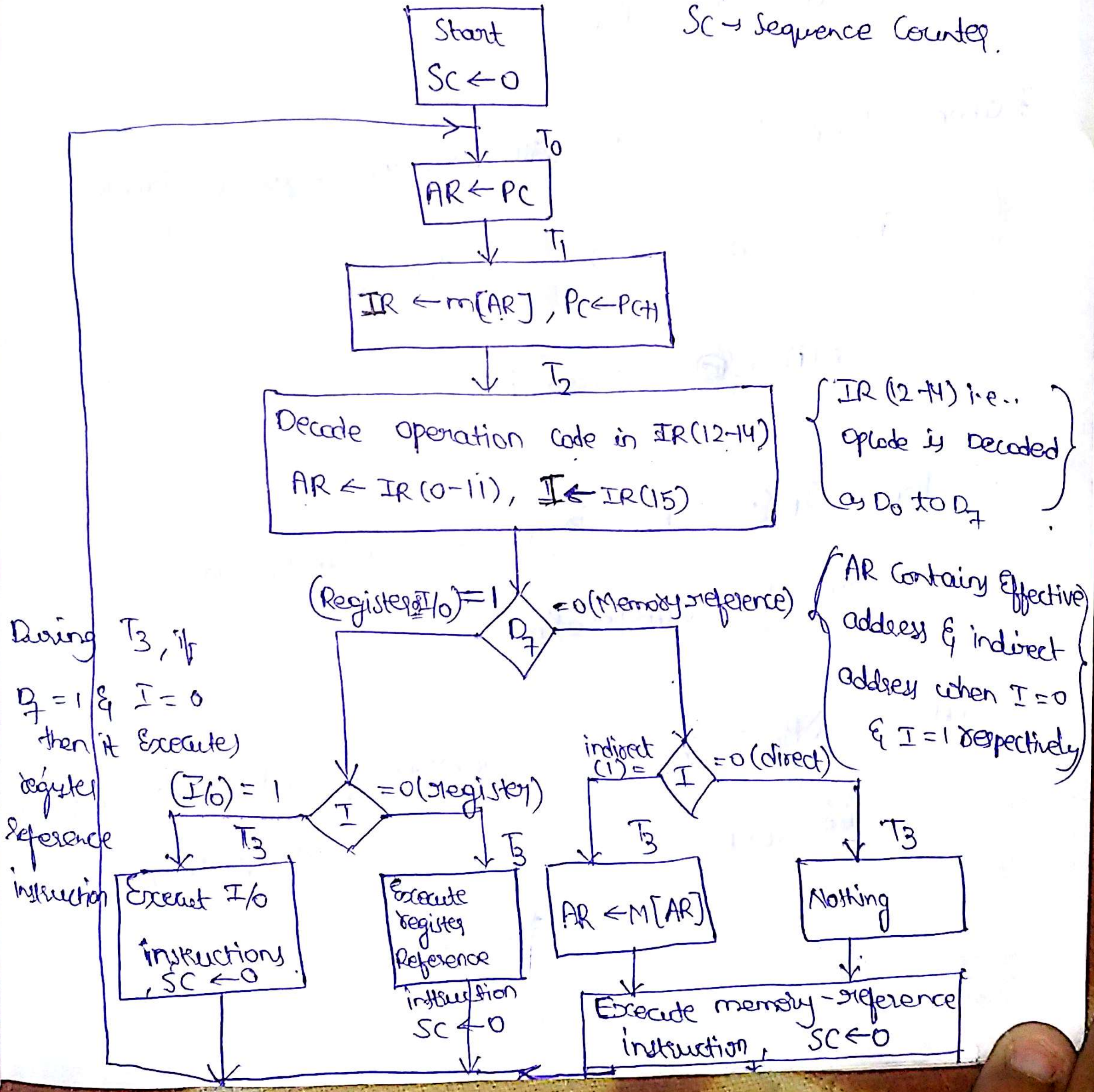
Hence, ~~mea~~ instruction is transfered as PC ← PC+1 at T_1 active

⇒ INR should be enabled when T_1 active

02/08/17

*Flowchart for instruction cycle:→

SC → Sequence Counter.



IR (12-14) i.e.. opcode is decoded as D_0 to D_7.

AR contains Effective address & indirect address when I = 0 & I = 1 respectively

During T_3, if D_7 = 1 & I = 0 then it execute register reference instruction

* R ← ashlR

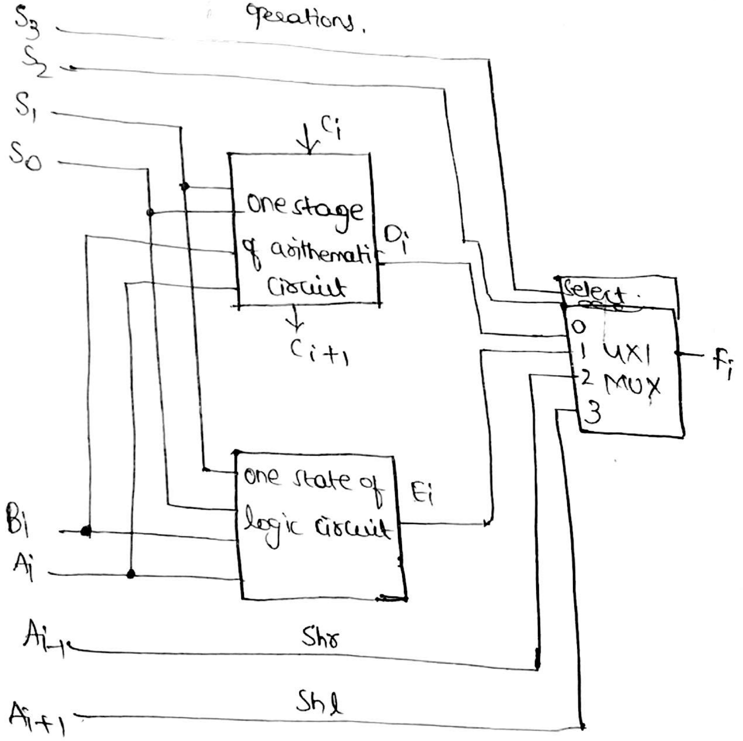
* arithematic shift left Register "R"

*R ← ashrR

* arithematic shift right Register "R"

* Arithematic logic shift unit :-

* we can perform arithematic & logic shift operations.



8 - arithematic

4 → logical

2 → shift {logical}

14 → operation Totally.

S3, S2 determines logical. operations

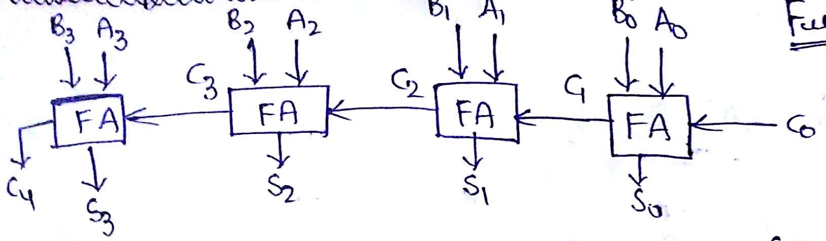
S1, S0, Cin determines type of arithematic operations

Operation Select

S_3	S_2	S_1	S_0	C_{in}		Operation	function.
0	0	0	0	0	arithmatic operations	$F = A$ ~~+B~~	Transfer A.
0	0	0	0	1		$F = A +$ ~~B+~~ 1	Increment.
0	0	0	1	0		$F = A + B$	Addition.
0	0	0	1	1		$F = A + B + 1$	add with Carry.
0	0	1	0	0		$F = A + \bar{B}$	Subraction with borrow
0	0	1	0	1		$F = A + \bar{B} + 1$	& Subraction
0	0	1	1	0		$F = A$ ~~+1~~ -1	decrement
0	0	1	1	1		$F = A$	Transfer A.
0	1	0	0	X	logic ckt independent of Cin	$F = A \wedge B$	AND.
0	1	0	ϕ	X		$F = A \vee B$	OR
0	1	ϕ	0	X		$F = A \oplus B$	XOR
0	1	1	1	X		$F = \bar{A}$.	Complement A
1	0	X	X	X	Shift operations	$F = shr\,A$.	Shift right
ϕ	1	X	X	X		$F = shl\,A$	Shift left.

[illegible]y Adder* {used for addition of two bits along with carries}.

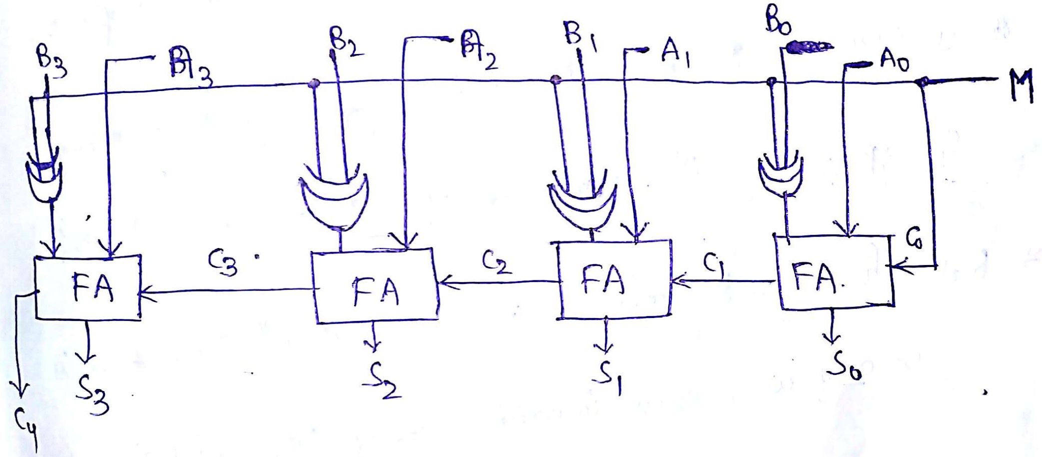
Full Adder (FA)



* fig :→ 4-bit binary adder.*

{C0 → input carry
C4 → o/p carry. }

* Binary adder - Subractor :→



* fig :→ 4-bit adder & subractor

if $M = 0 \Rightarrow$ B_0 XOR with $0 = B_0$. & i/p carry

B_1 " " $0 = B_1$, $C_0 = 0$

$0 = B_2$ if $M=0$

B_2 " " $0 = B_3$

$\Rightarrow S_0 = A_0 + B_0 + C_0$ { B_3 " "

$S_1 = A_1 + B_1 + C_1$ { - - -

If $M = 1$; B_0 XOR with $1 = \bar{B_0}$ & carry i/p

B_1 " " $1 = \bar{B_1}$ → $C_0 = 1$

B_2 " " $1 = \bar{B_2}$ if $M = 1$

B_3 " " $1 = \bar{B_3}$

$S_0 = A_0 + \bar{B_0} + C_0.$

$\Rightarrow S_0 = A_0 + \bar{B_0} + 1$

$\Rightarrow S_0 = A_0 - B_0$ } for unsigned numbers when $A \geq B$.

i.e.: $\Rightarrow S = A - B.$

$\bar{B}+1 = 2's$ of B

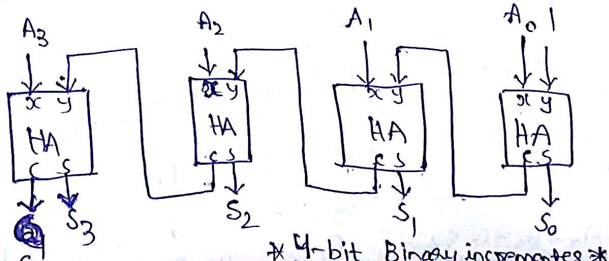
$\bar{B}+1$ →

$\Rightarrow A + \bar{B} + 1$

∴ is giving $A - B$ when $A \geq B$ for unsigned numbers & gives 2's compliment of "B − A" for "A < B".

* But for signed numbers it gives "A − B" when theirs is no overflow → {result exceeding size of the register}

* Binary Incrementer:→



{Result = A+1}

{A → A3A2A1A0}

* 4-bit Binary incrementer *

* 06!

* Probler

*1) Perfor by t by

*sol)

*2)

*sol)