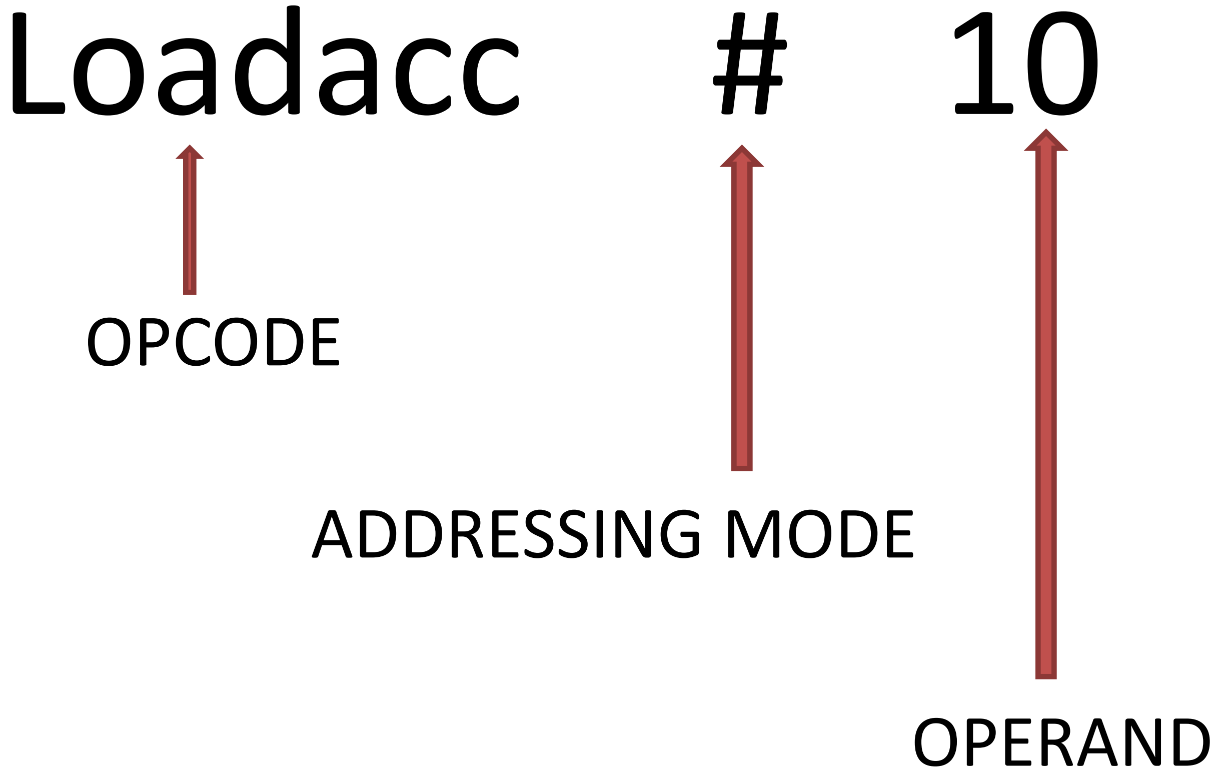
EEX 5563 LAB 2

Programming in Assembly Using SEPSim Simulator

Instruction



Loadacc # 10

OPCODE

ADDRESSING MODE

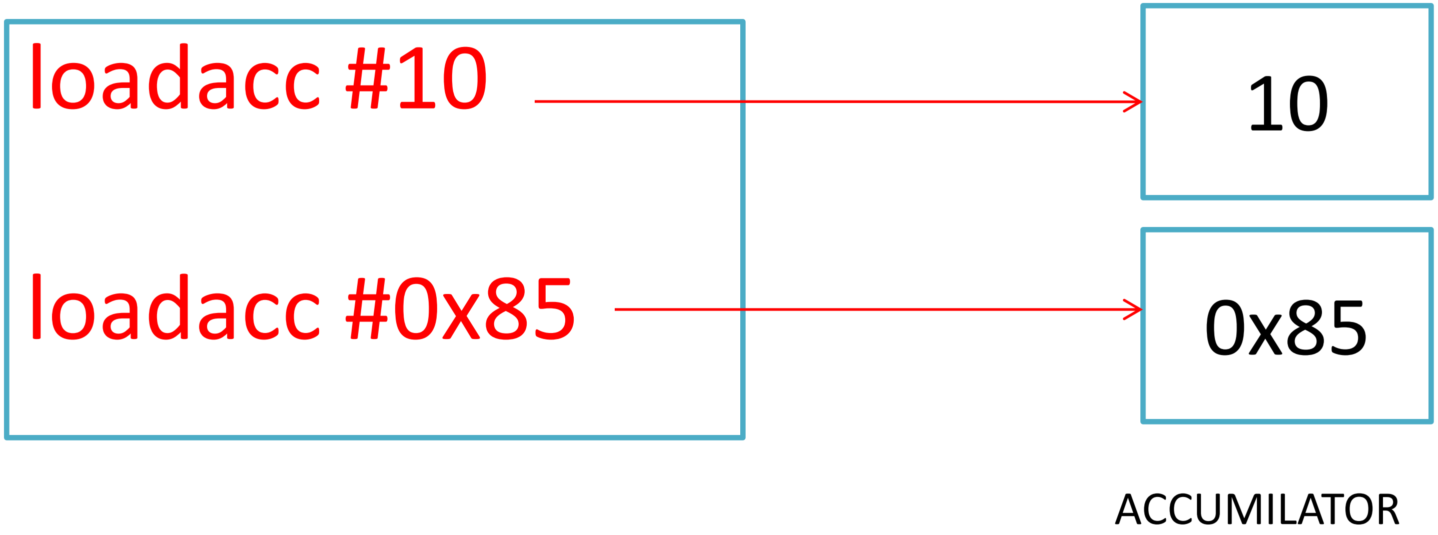
OPERAND

Addressing Modes

- Immediate addressing Mode
- Direct addressing mode
- Indirect addressing mode

Immediate addressing Mode

Immediate addressing mode use # symbol.



loadacc #10

loadacc #0x85

10

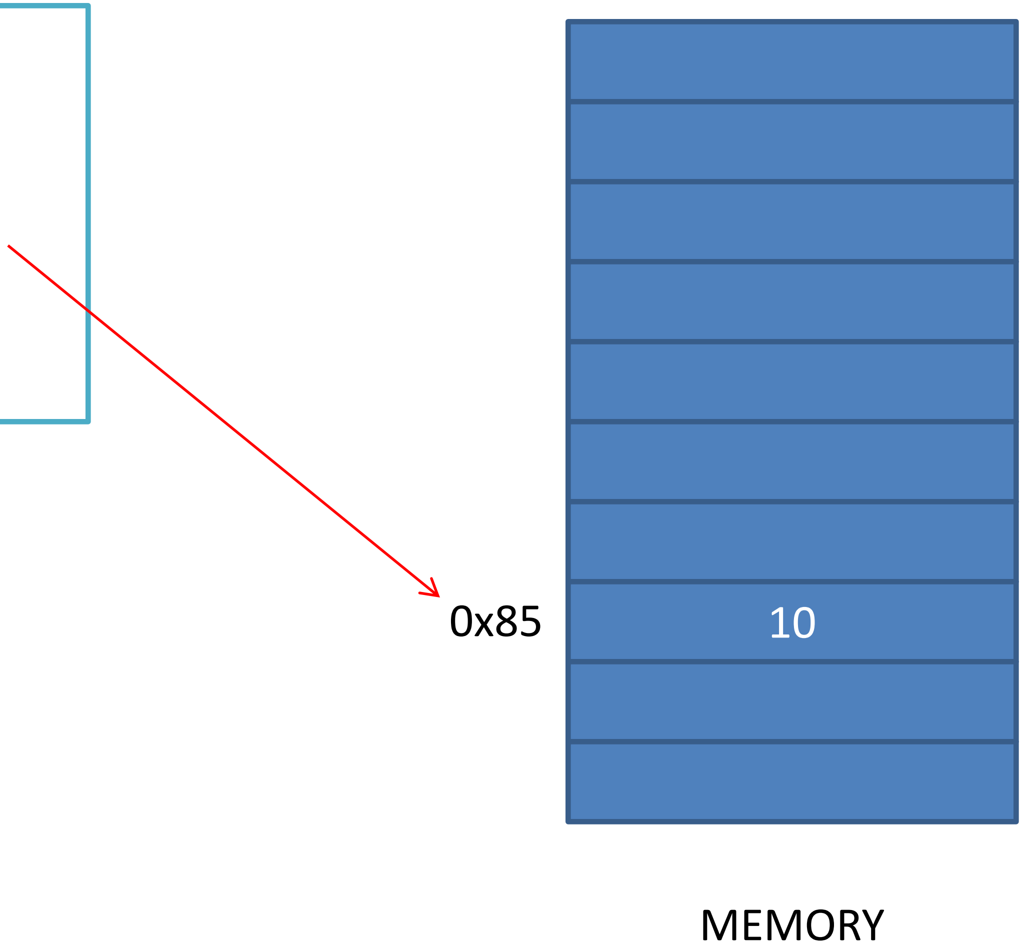
0x85

ACCUMILATOR

Direct addressing mode

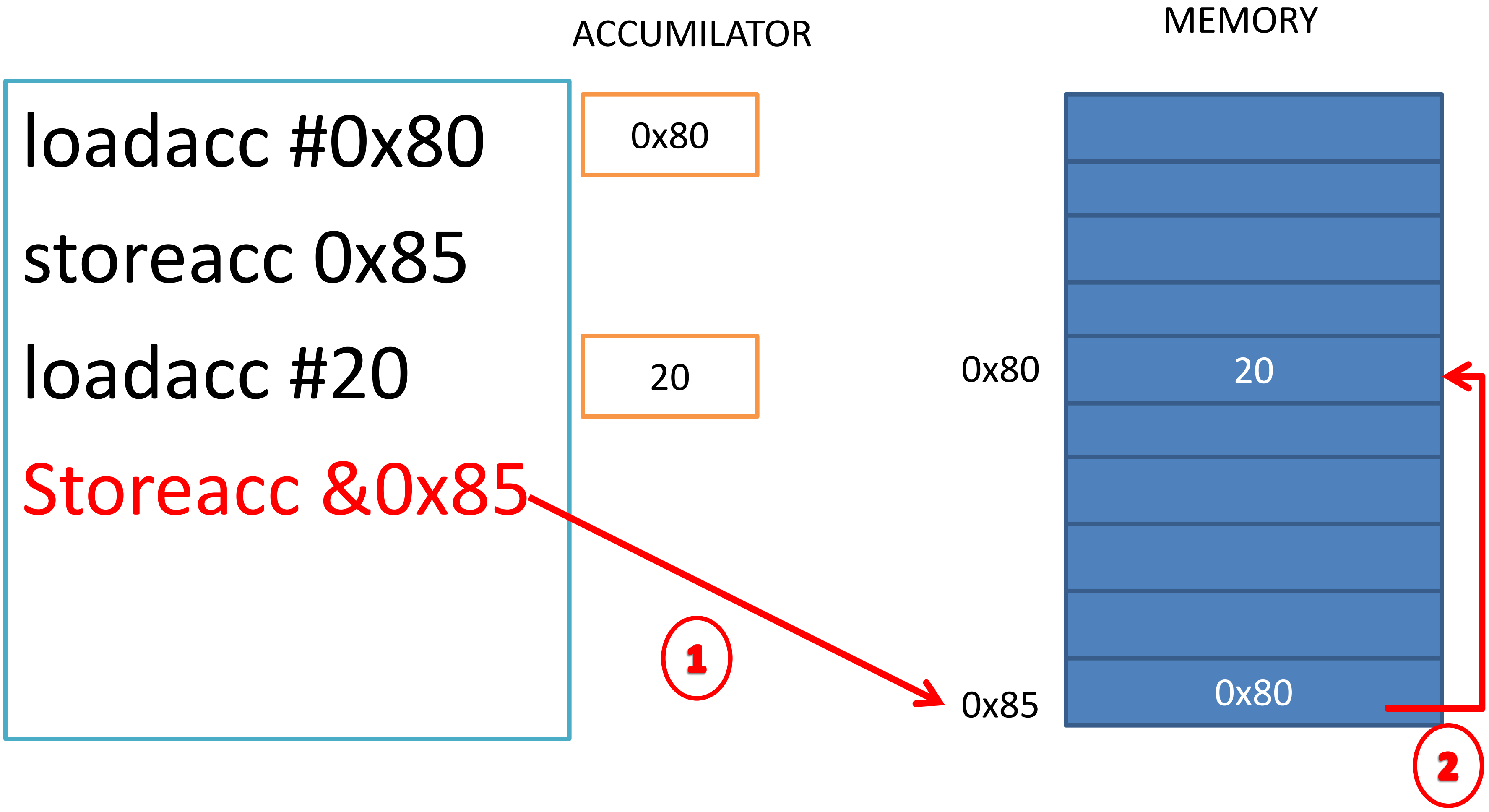
loadacc #10

storeacc 0x85



Indirect addressing mode

Indirect addressing mode use & symbol.



```
loadacc #0x80
storeacc 0x85
loadacc #20
Storeacc &0x85
```

IL and RTL

- IL – Instruction Level
- RTL – Register Transfer Level

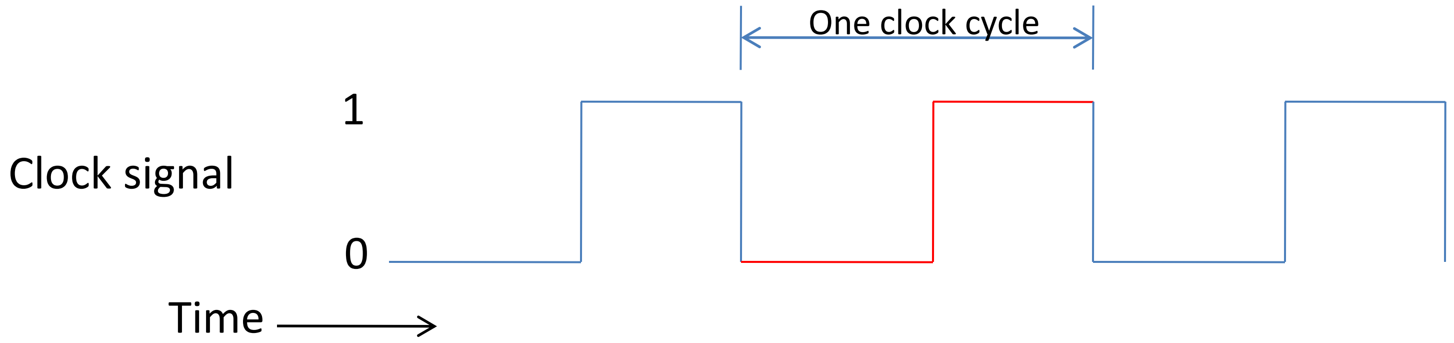
CLOCK CYCLE

- Clock signal typically consists of a square wave that oscillates between a high and low voltage level.
- A clock cycle is a single electronic pulse generated by the CPU's clock
- During each cycle, the CPU performs basic operations such as,

  -Fetching an instruction from memory.

  -Accessing memory to read or write data.

  -Executing simple commands.



MAR AND MDR

- MAR

  The memory address register holds the address of the current instruction that is to be fetch from memory, or the address in memory to which data is to be transferred.

- MDR

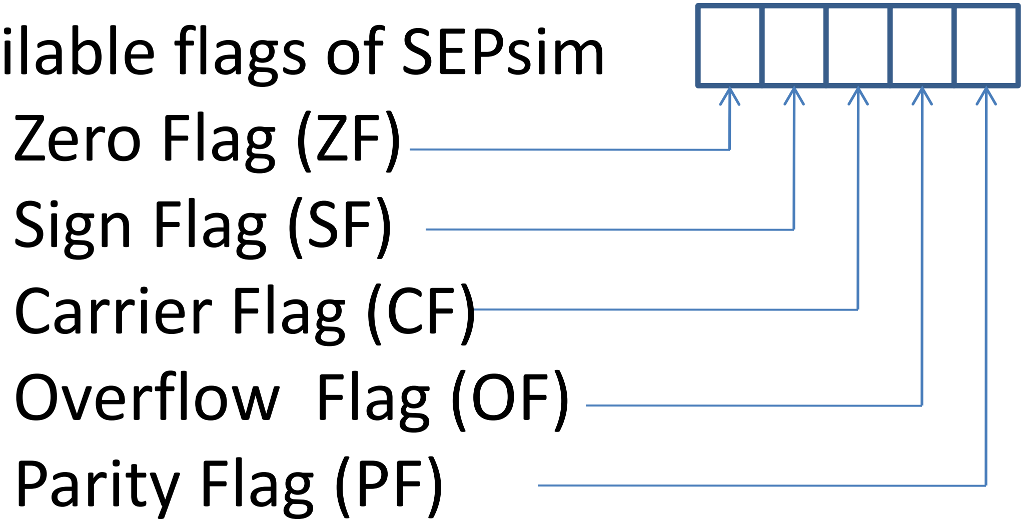
  Memory Data Register holds the contents found at the address held in the MAR, or data which is to be transferred to primary storage.

Simple IF condition using SEPsim

- There is no if instruction in Accumulator base architecture.
- For that purpose we can use conditional branch operations such as JZ,JS,JC,JP,JOF

STATUS / FLAG REGISTER

- Hardware register that contains information about the state of the processor.
- It reflect the result after any arithmetic and logic operation
- Available flags of SEPsim



Zero Flag (ZF)

Sign Flag (SF)

Carrier Flag (CF)

Overflow Flag (OF)

Parity Flag (PF)

- Flags are deal with conditional branch operations like JZ,JS,JC,JP,JOF

STATUS / FLAG REGISTER

Sign Flag (SF)

Indicates that the result of a mathematical operation is negative.

Example:

```
loadacc #5
sub #6    // final value is (5-6=-1)
```

then the sign flag will set.

1

SF

STATUS / FLAG REGISTER

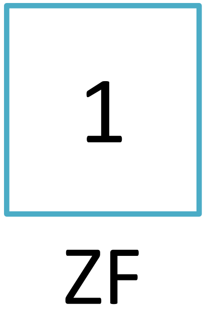
Zero Flag (ZF)

Indicated that the result of an arithmetic operation was zero.

Example:

```
loadacc #5
sub #5    // final value is (5-5=0)
```

then the zero flag will set.



1

ZF

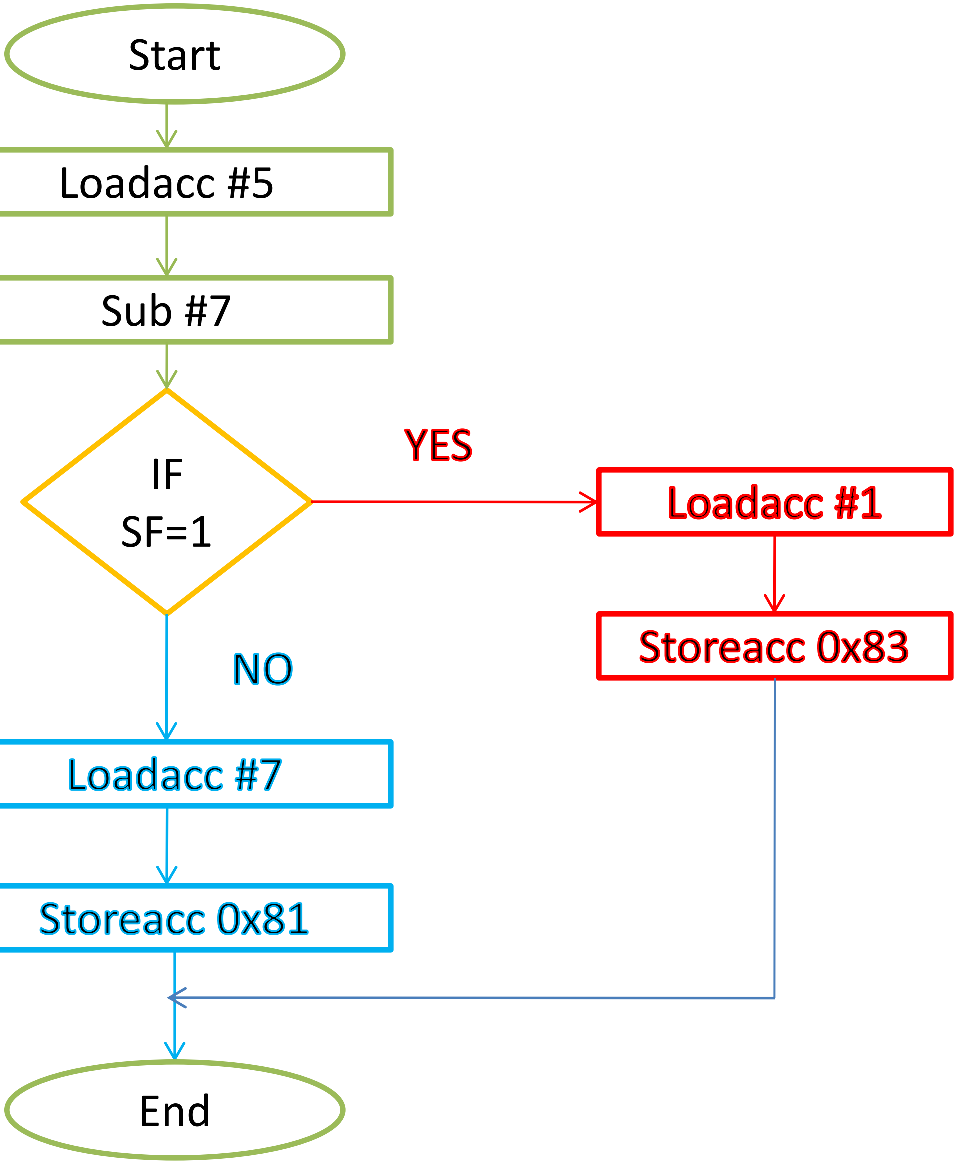
Simple program with IF

Draw a flowchart to demonstrate the process of comparing two values. Write an assembly language program for the above algorithm.

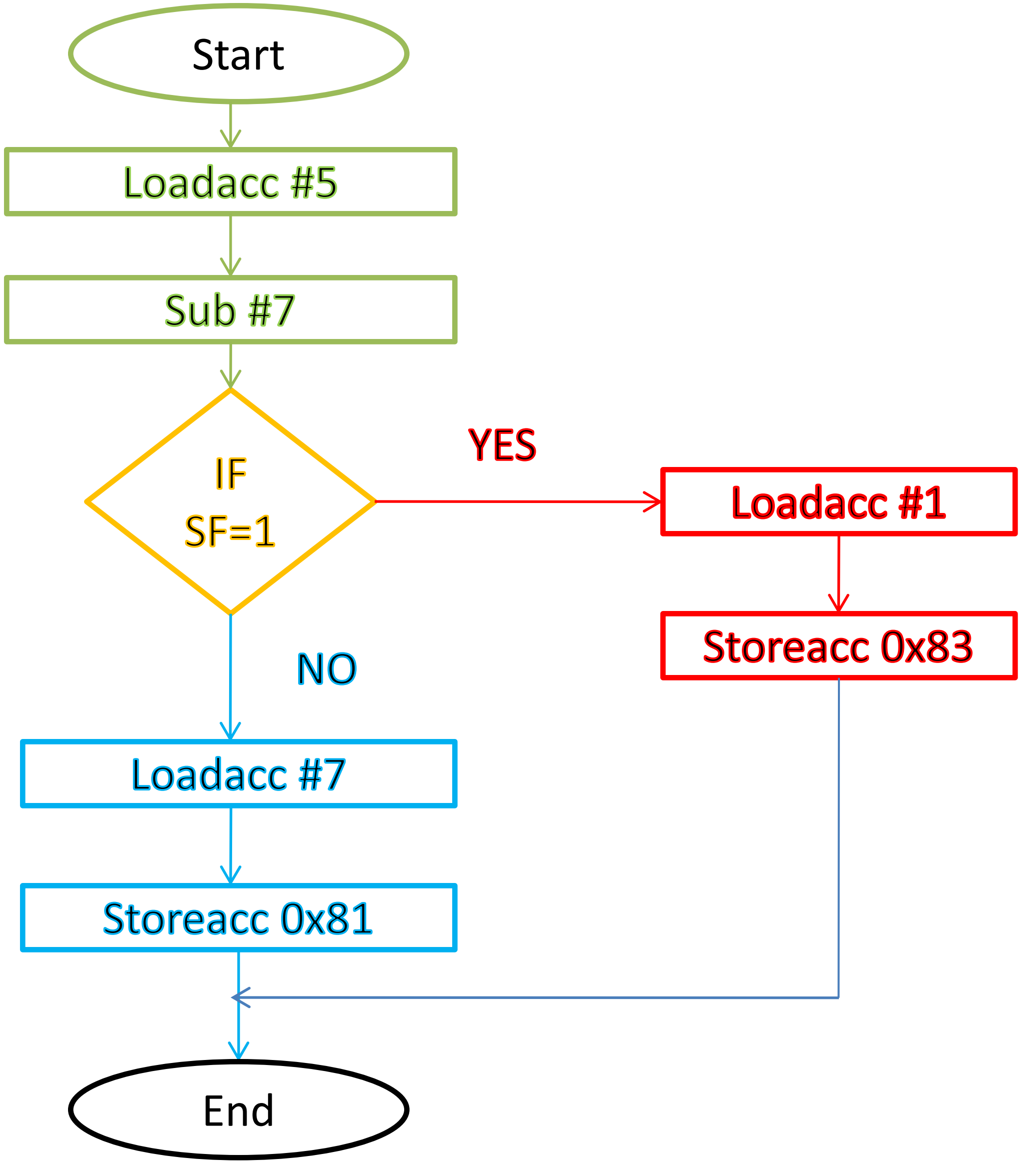
After comparing two digits if the result is ,

- Negative store the value 1 in 0x83 memory location.

- Positive store the value 7 in 0x81 memory location.


Start
Loadacc #5
Sub #7
IF
SF=1
YES
Loadacc #1
Storeacc 0x83
NO
Loadacc #7
Storeacc 0x81
End

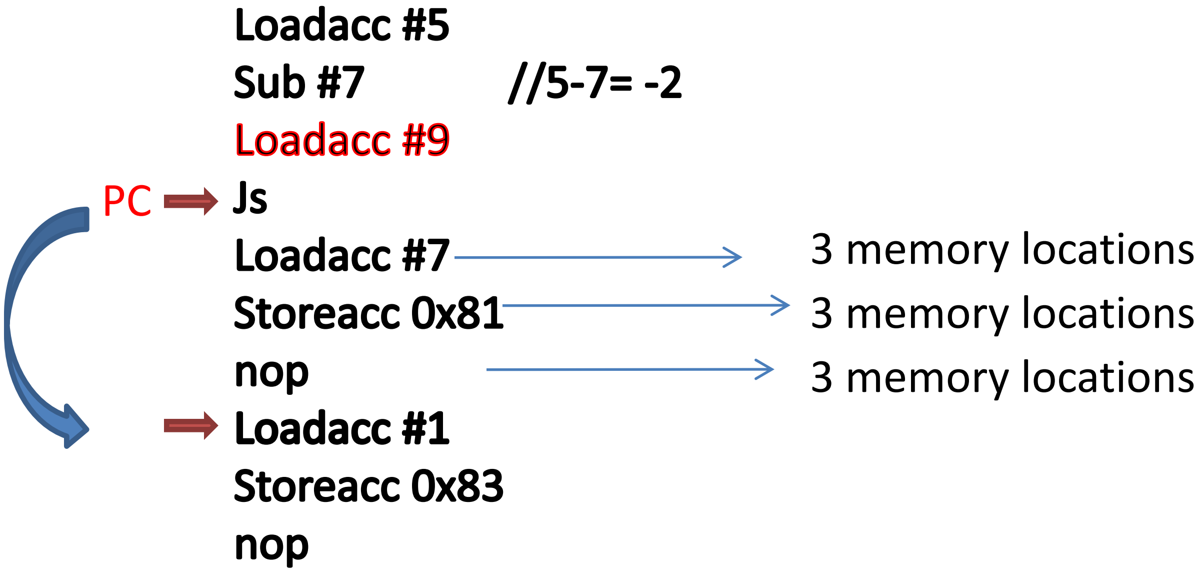
```
Loadacc #5
Sub #7
(to set jump location)
Js
Loadacc #7
Storeacc 0x81
nop
Loadacc #1
Storeacc 0x83
nop
```



Set jump location

JS- Jump if sign – if SF 1 then

PC ------- > PC + Operand



```
Loadacc #5
Sub #7          //5-7= -2
Loadacc #9
Js
Loadacc #7
Storeacc 0x81
nop
Loadacc #1
Storeacc 0x83
nop
```

PC ------- > PC + Operand

PC ------- > PC + (instructions* memory locations)

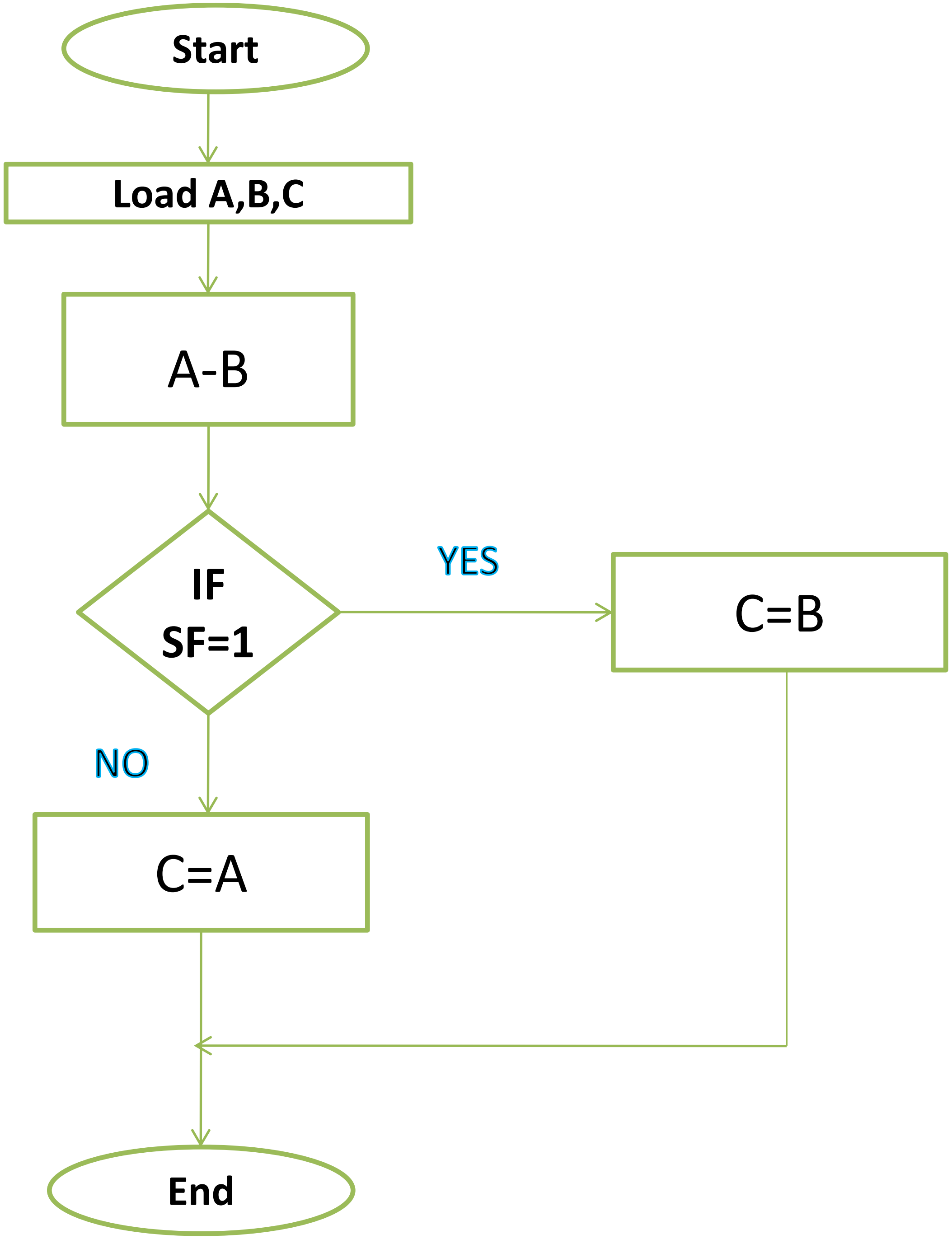
PC ------- > PC + (3* 3)

Experiment 1

Implement the following using the Instruction set of the Accumulator of the SEP.

```
If A< B then
    C=B
    else
    C= A
```

value of the three variables A,B and C are stored in the memory.


Start
Load A,B,C
A-B
IF
SF=1
YES
C=B
NO
C=A
End

Experiment 2

The array X has n elements. Find the sum of all elements of x. Use the loop instruction to do this task.

1	2	3			3	6	8
1	2					n-1	n

LOOZ

Looz instruction is use to create a loop.

Have to consider two things.

1. Looz counter
2. Jumping location

Looz counter

Looz counter memory address is 0x95 (149)

Number of loops =n

- Load accumulator with n

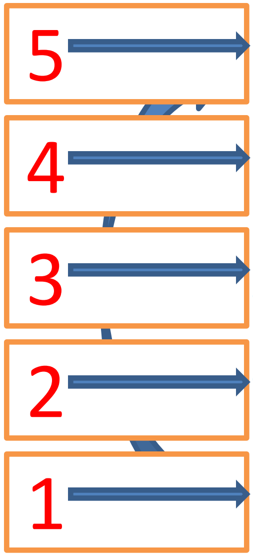
-Store that value in 0x95 memory location.

Example:

```
Loadacc #n
Storeacc 0x95
```

Jumping location

```
1 loadacc #4
2 storeacc 0x95
3
4
5
6
7
8
9 loadacc #(Jumping location)
10 looz
```



Jump location

No of Jumping locations → 3*5= 15

15 in binary → 1111

As 19 bit value → 000 0000 0000 0000 1111

1’s compliment → 111 1111 1111 1111 0000

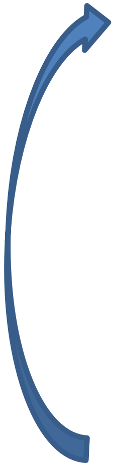
2’s compliment → 111 1111 1111 1111 0000

```
  111 1111 1111 1111 0000
                      + 1
  -----------------------
  111 1111 1111 1111 0001
```

As a hex value → 111 | 1111 | 1111 | 1111 | 0001

111	1111	1111	1111	0001
7	F	F	F	1

Jump location

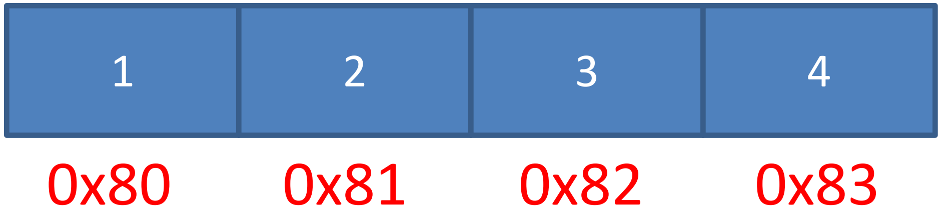


```
1 loadacc #4  // number of loops
2 storeacc 0x95 // looz counter
3
4
5
6
7
8
9 loadacc #0x7FFF1 //Jump location
10 looz
```

Program with Looz

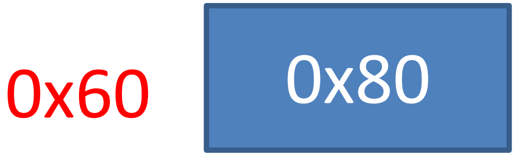
```
Loadacc #4 // no of loops
Storeacc 0x95 //looz counter
Loadacc #2 //jump location
loadacc #23
loadacc #16
Loadacc #0x7FFF1
looz
```

Hint



1	2	3	4
0x80	0x81	0x82	0x83

- Store 0x80 to another location

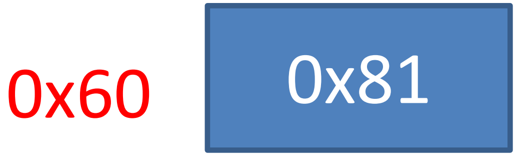


0x60 0x80

- Use indirect addressing to add values

add &0x60

- Increment the 0x60 value



0x60 0x81