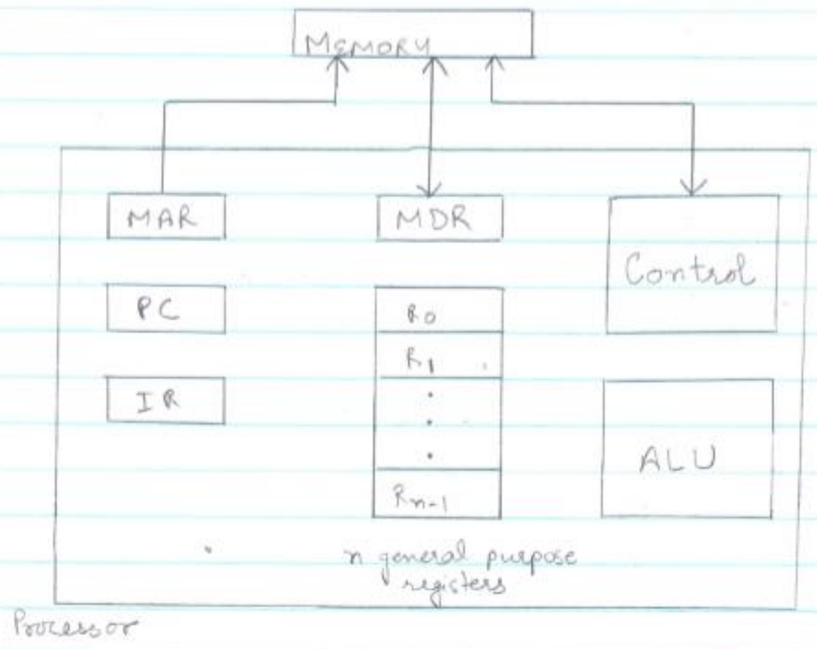
MODULE 1-FREQUENTLY ASKED QUESTIONS

Q1 Connection between memory and processor (operating steps) with diagram?

Now, how contents are transferred between memory and processor?
Address of memory location to be accessed is sent to memory unit. Control signals are issued, and data is transferred to or from memory



Connections b/w Processor & the memory

Memory :- Stores data and instructions.

- Instruction register (IR) - Holds instructions that is currently being executed. Its output is available to the control circuits which generate the timing signals that control various processing elements involved in executing the instruction

- Program counter (PC) - contains memory address of next instruction to be fetched and executed.

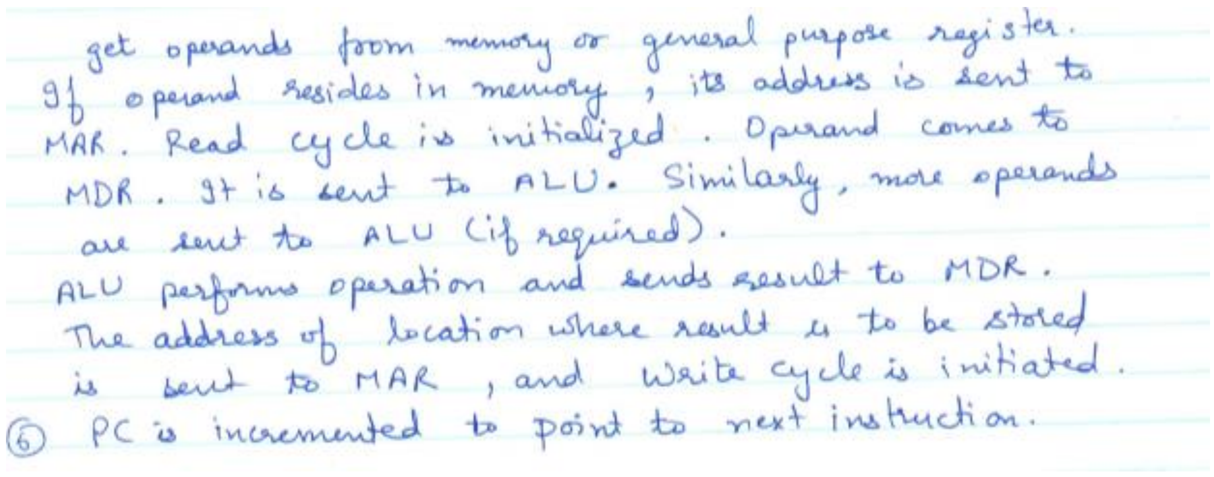
- Memory address register (MAR) - holds address of location to be accessed

- Memory data register (MDR) - contains data to be written into or read out of the addressed location.

<u>Operating steps</u>

❀ Programs (list of instructions) reside in memory (usually ~~stored these things~~ get there through input unit).

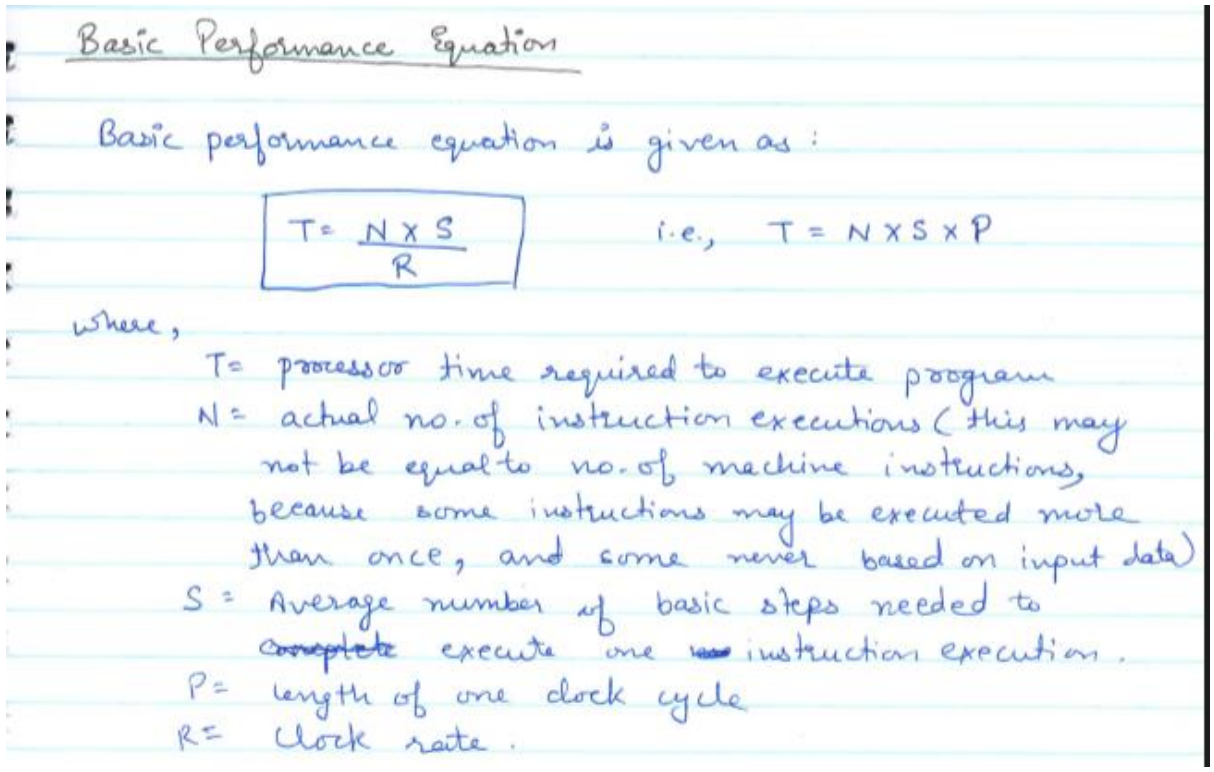
① PC is set to point to first instruction of program.
② This PC contents is transferred to MAR and Read control signal is send to memory
③ After time required to access the memory elapses, addressed word (1st instruction in this case) is read out from memory and loaded in MDR.
④ MDR contents are transferred to IR.
⑤ If instruction involves an operation by ALU:



get operands from memory or general purpose register.
If operand resides in memory, its address is sent to MAR. Read cycle is initialized. Operand comes to MDR. It is sent to ALU. Similarly, more operands are sent to ALU (if required).
ALU performs operation and sends result to MDR.
The address of location where result is to be stored is sent to MAR, and write cycle is initiated.
⑥ PC is incremented to point to next instruction.

NOTE: If a source of destination is a register (R), MAR, MDR steps are not required as registers are directly accessible to ALU as both reside inside processor. MAR and MDR are required only if we want to access main memory for read or write operation.

Q2 Basic performance equation and SPEC rating



Basic Performance Equation

Basic performance equation is given as :

$$\boxed{T = \frac{N \times S}{R}} \qquad \text{i.e.,} \quad T = N \times S \times P$$

where,

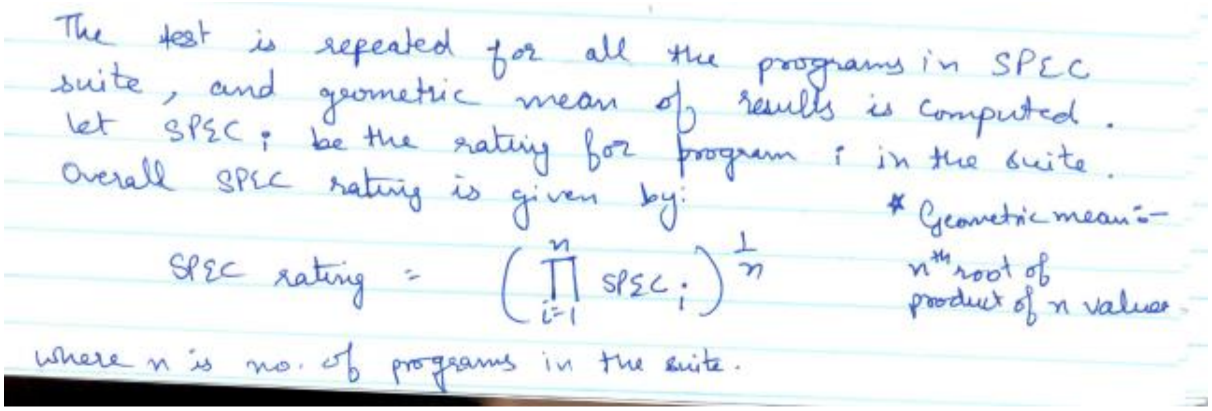
T = processor time required to execute program
N = actual no. of instruction executions (this may not be equal to no. of machine instructions, because some instructions may be executed more than once, and some never based on input data)
S = Average number of basic steps needed to ~~complete~~ execute one [illegible] instruction execution.
P = length of one clock cycle
R = Clock rate.

Performance Measurement

Therefore, now a days, computer performance is measured using benchmark programs. Standardized programs are used for better comparisons.
The performance measure is the time taken by computer to execute a given benchmark.
A non profit organization called System Performance Evaluation Corporation (SPEC) selects and publishes ~~representative~~

representative application programs for different application domains, together with test results for many commercially available computers.
The programs selected range from game playing, compiler and database applications to numerically intensive programs in astrophysics and quantum chemistry.
In each case, the program is compiled for the computer under test, and running time on real computer is measured. Simulation is not allowed. The same program is also compiled and run on one computer selected as a reference.

$$\text{SPEC rating} = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

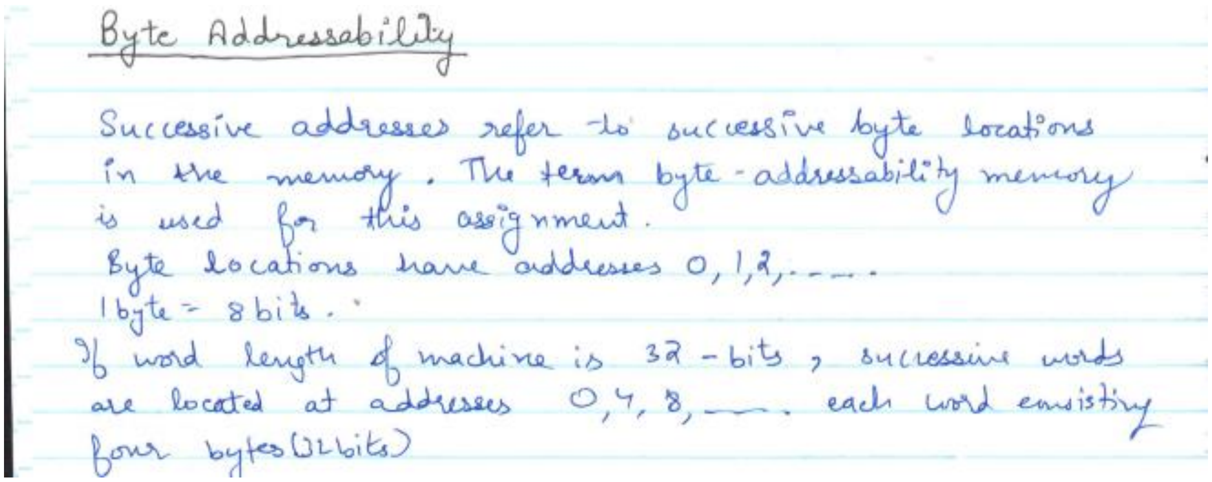

The test is repeated for all the programs in SPEC suite, and geometric mean of results is computed. let $SPEC_i$ be the rating for program i in the suite. Overall SPEC rating is given by:

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} SPEC_i \right)^{\frac{1}{n}}$$

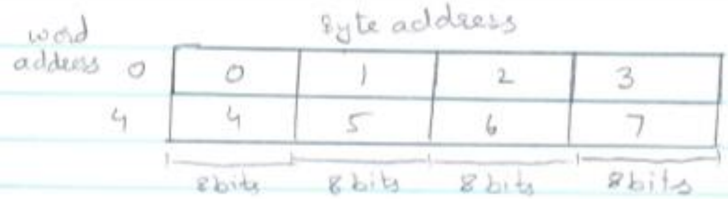
* Geometric mean :- nth root of product of n values.

where n is no. of programs in the suite.

Q3 Byte addressability (Big-endian and Little-endian assignments with diagram)


Byte Addressability

Successive addresses refer to successive byte locations in the memory. The term byte-addressability memory is used for this assignment.
Byte locations have addresses 0, 1, 2, ...
1 byte = 8 bits.
If word length of machine is 32-bits, successive words are located at addresses 0, 4, 8, ... each word consisting four bytes (32 bits)



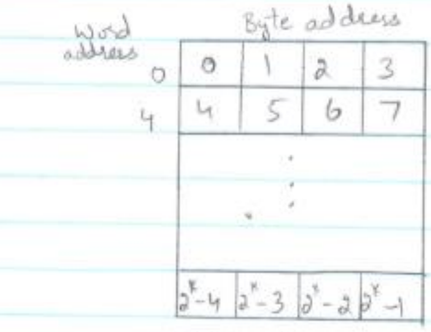
Big-endian and little-endian assignments.

These 2 methods are used for byte addressing. Any one method is selected out of these.

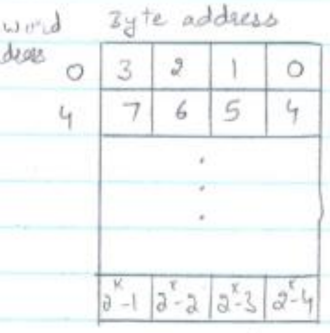
Big-endian assignment - Lower byte addresses are used for more significant bytes (leftmost bytes) of the word.

Little-endian assignment - Lower byte addresses are used for the less significant bytes (rightmost bytes) of the word.

The words 'more significant' and 'less significant' are used in relation to the weights (power of 2) assigned to bits when word represents a number.



(a) Big-endian assignment



(b) Little-endian assignment

BYTE & WORD ADDRESSING

Words are said to be aligned in memory if they begin at a byte address that is a multiple of the number of bytes in a word.

Q4 Instruction types (one-address, two-address, three-address instructions)

Basic Instruction Types

$C \leftarrow [A] + [B]$

Add contents of A & B, place sum in C. A, B contents are unchanged

Consider three-address instruction

Operation Source1, Source2, Destination

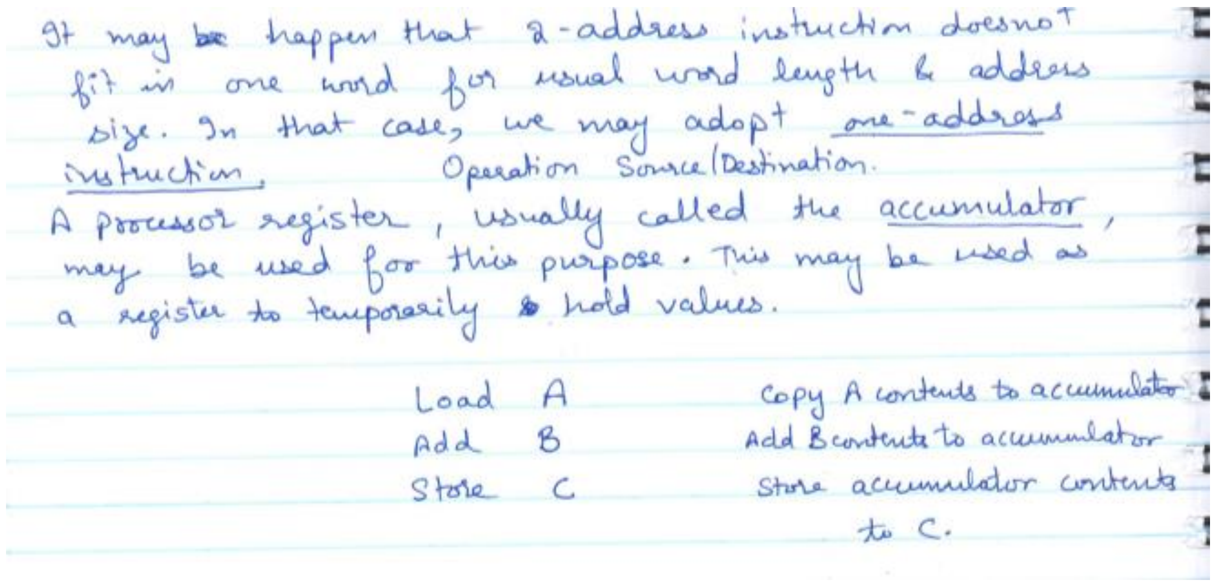
Add A, B, C

Two-address instruction

Operation Source, Destination

Move B, C

Add A, C



It may ~~be~~ happen that 2-address instruction doesnot fit in one word for usual word length & address size. In that case, we may adopt one-address instruction, Operation Source/Destination.
A processor register, usually called the accumulator, may be used for this purpose. This may be used as a register to temporarily ~~s~~ hold values.

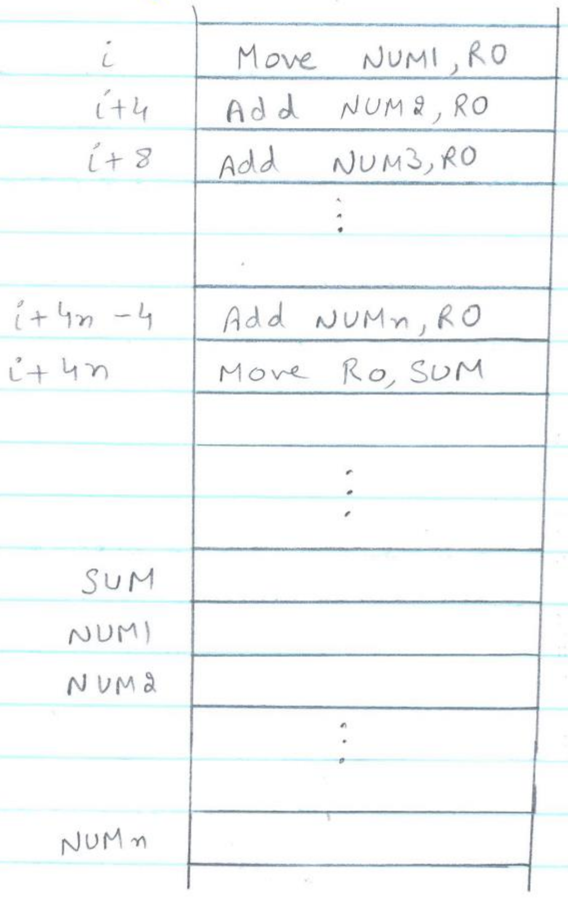
Load	A	Copy A contents to accumulator
Add	B	Add B contents to accumulator
Store	C	Store accumulator contents to C.

Q5 Explain Branching and example?

Branching

Consider the task of adding a list of n numbers.
It can be done in straight line sequencing or by using a loop. lets consider both, one by one.

Straight line sequencing - Add n numbers

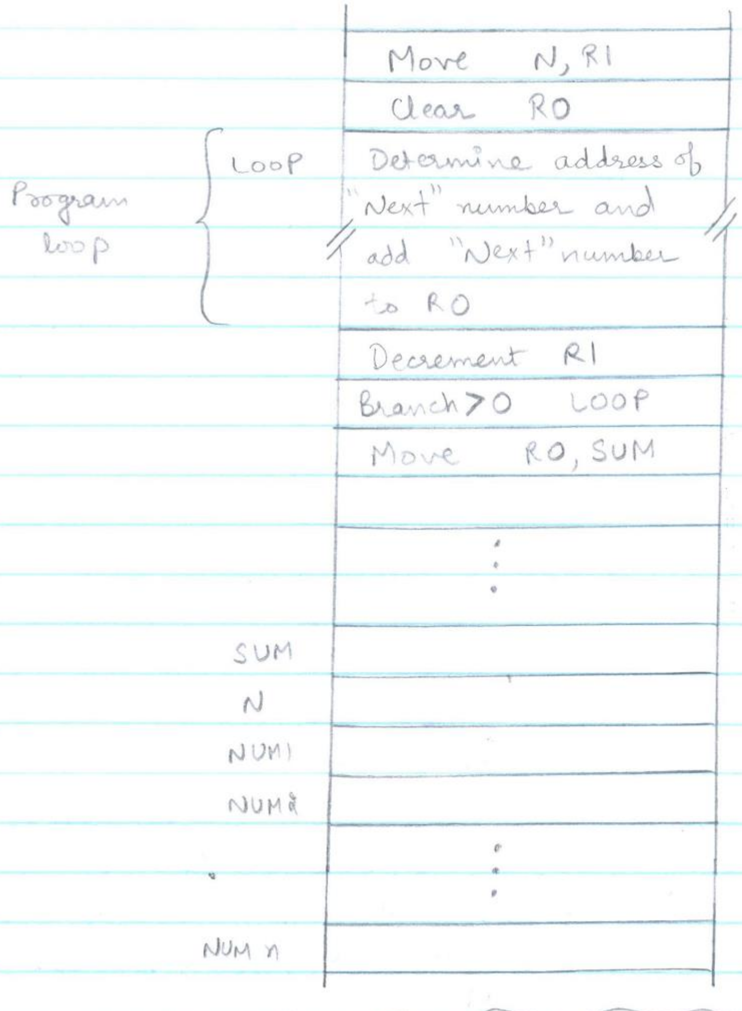


Straight-line program for adding n numbers

Addresses of memory locations containing n numbers are symbolically given as NUM1, NUM2, ..., NUMn.
Add instruction is used to add each number to the contents

of register R0. After all the numbers have been added, the result is placed in memory location SUM.

But, if you observe, you can see that a long list of ~~Add~~ Add instructions is listed. Instead of using a long list of Add instructions, it is possible to place a single Add instruction in a program loop. The loop is a straight-line sequence of instructions executed as many times as needed.



using a loop to add n numbers

Q6 Addressing modes (definitions with examples) – mention all 8 addressing modes no matter how many have been asked

ADDRESSING MODES

The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.

① Immediate mode

The operand is given explicitly in the instruction.

Eg. Move #200, R0

Above instruction places value 200 in register R0.
Generally, this mode is used to represent constants.

② Register mode

The operand is the contents of a processor register; the name of the register is given in the instruction.

Eg. - Move R1, R2. R1 Operand

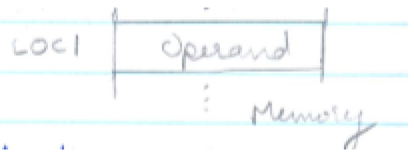
Above instruction moves the contents of register R1 to R2.
Generally, this mode is used to access variables.

③ Absolute (Direct) mode

The address of the memory location where operand is located is given explicitly in the instruction.

Eg- Move LOC1, LOC2 Me

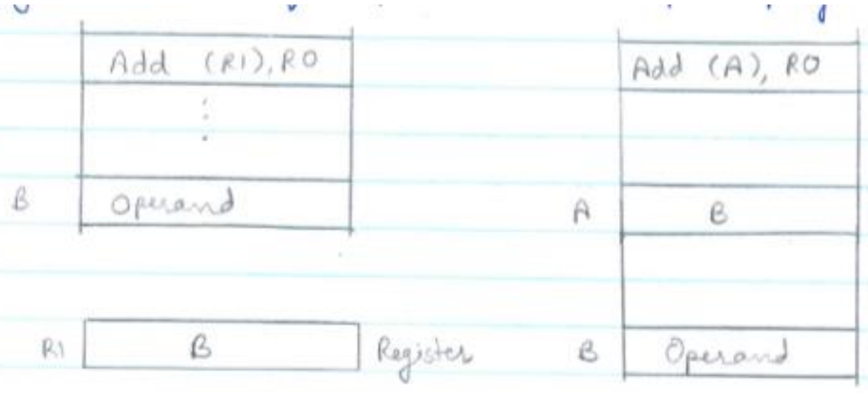
Above instruction moves the operand at location LOC1 to location LOC2.
It is generally used for representing global variables.



④ <u>Indirect mode</u>

The contents of register or memory location given in the instruction gives the effective address of the operand.

Eg. -



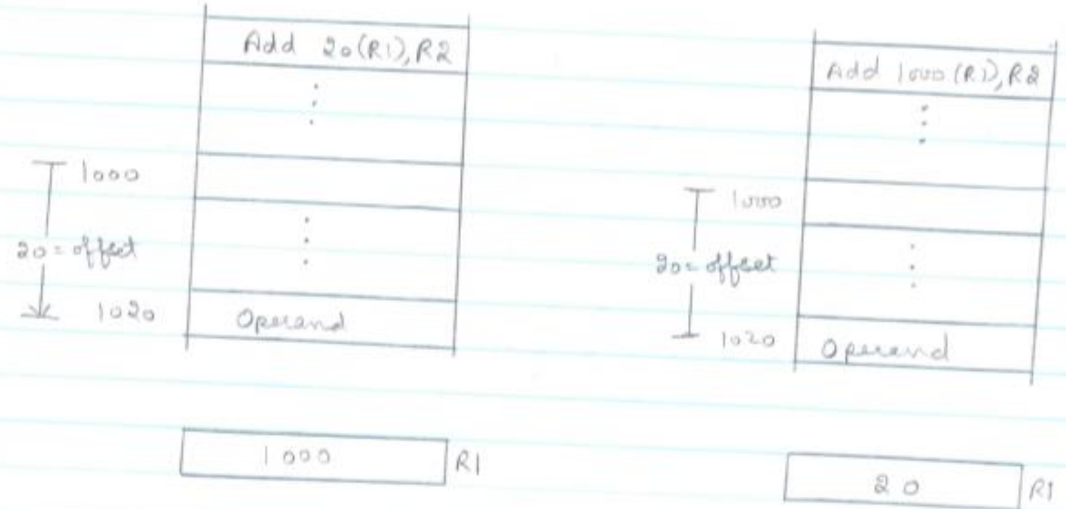
(a) Through a general purpose register (b) Through a memory location

Indirect addressing

⑤ <u>Index mode</u>

The effective address of the operand is generated by adding a constant value to the contents of a register. The register used may be a special register or a general-purpose register and is known as an index register.

Eg.



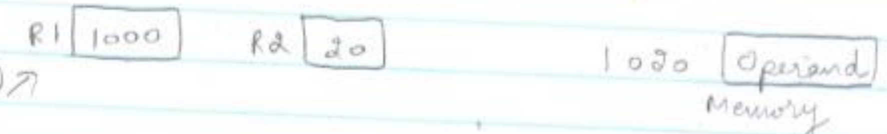
(a) offset is given as a constant (b) offset is in the index register

Indexed addressing

Base with index mode

The effective address is the sum of the contents of of multiple registers.

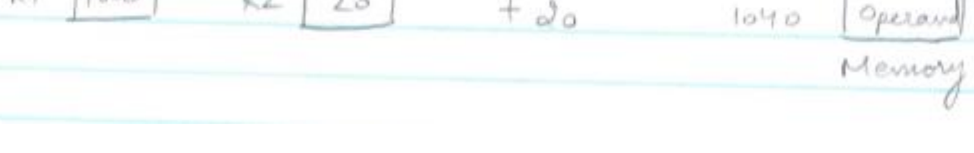
Eg - (R1, R2)

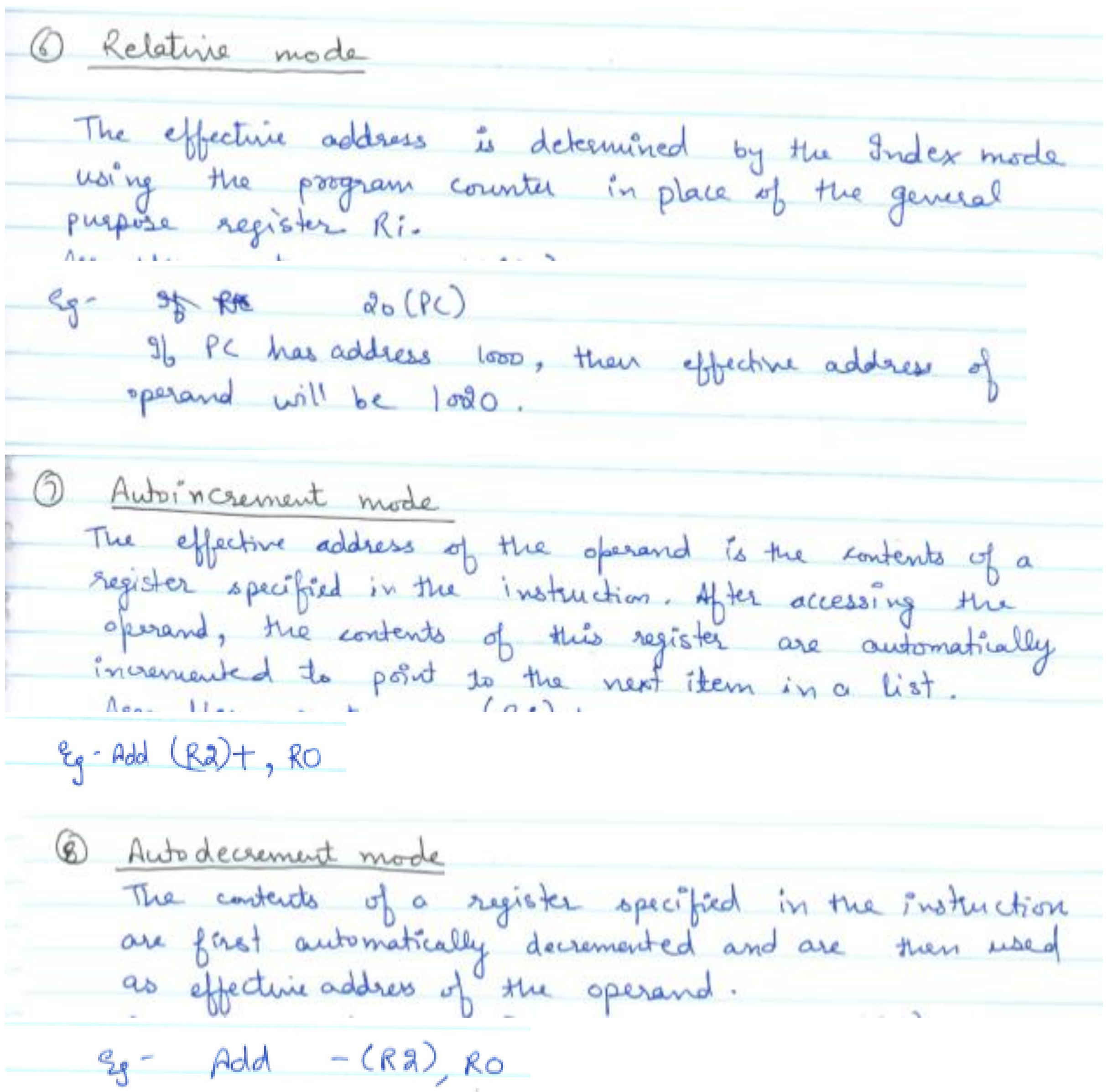


Base with index and offset

2 registers and a constant is used to calculate effective ~~off~~ address of the operand.

Eg - 20(R1, R2)




⑥ Relative mode

The effective address is determined by the Index mode using the program counter in place of the general purpose register Ri.

Eg- ~~If Ri~~ 20(PC)
If PC has address 1000, then effective address of operand will be 1020.

⑦ Autoincrement mode

The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.

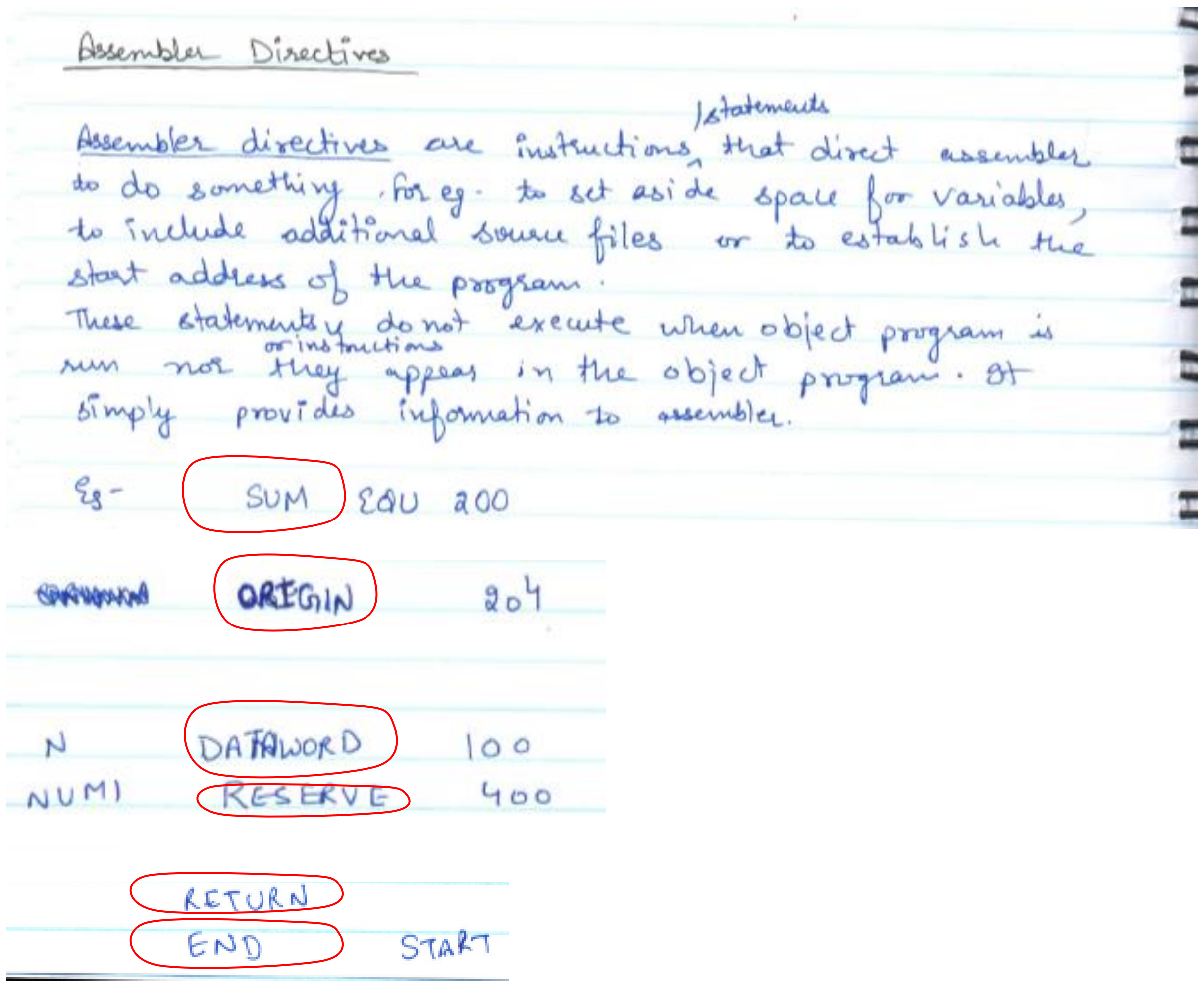
Eg- Add (R2)+, R0

⑧ Autodecrement mode

The contents of a register specified in the instruction are first automatically decremented and are then used as effective address of the operand.

Eg- Add -(R2), R0

Q7 Explain Assembler directives?



Assembler Directives

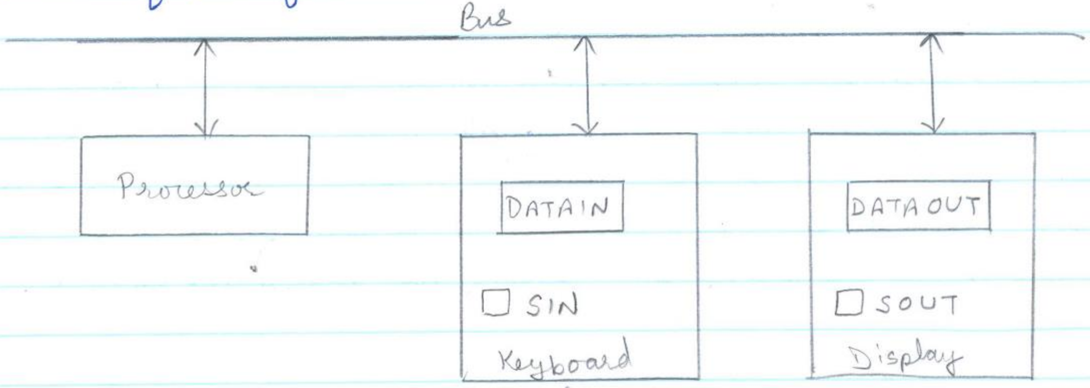
Assembler directives are instructions/statements that direct assembler to do something, for eg. to set aside space for variables, to include additional source files or to establish the start address of the program.
These statements or instructions do not execute when object program is run nor they appear in the object program. It simply provides information to assembler.

```
Eg-     SUM       EQU   200

        ORIGIN          204

N       DATAWORD        100
NUM1    RESERVE         400

        RETURN
        END             START
```

Q8 Basic I/O operations- bus connection for processor, keyboard and display?

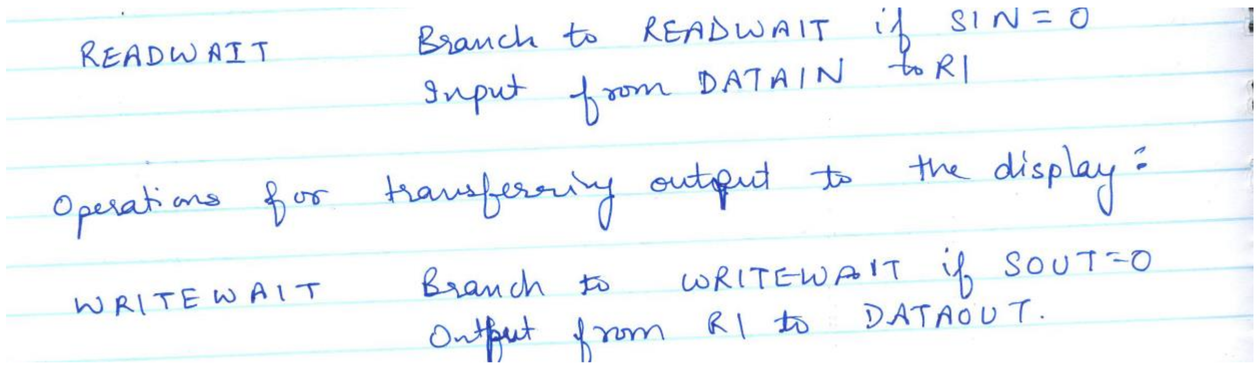
BASIC INPUT/OUTPUT OPERATIONS

Data on which instructions operate might not always be in memory. I/O devices are used for this purpose, when data needs to be transferred to/from I/O devices. The difference in speed between the processor and I/O devices creates the need for mechanisms to synchronize the transfer of data between them.



Bus connection for processor, keyboard, & display

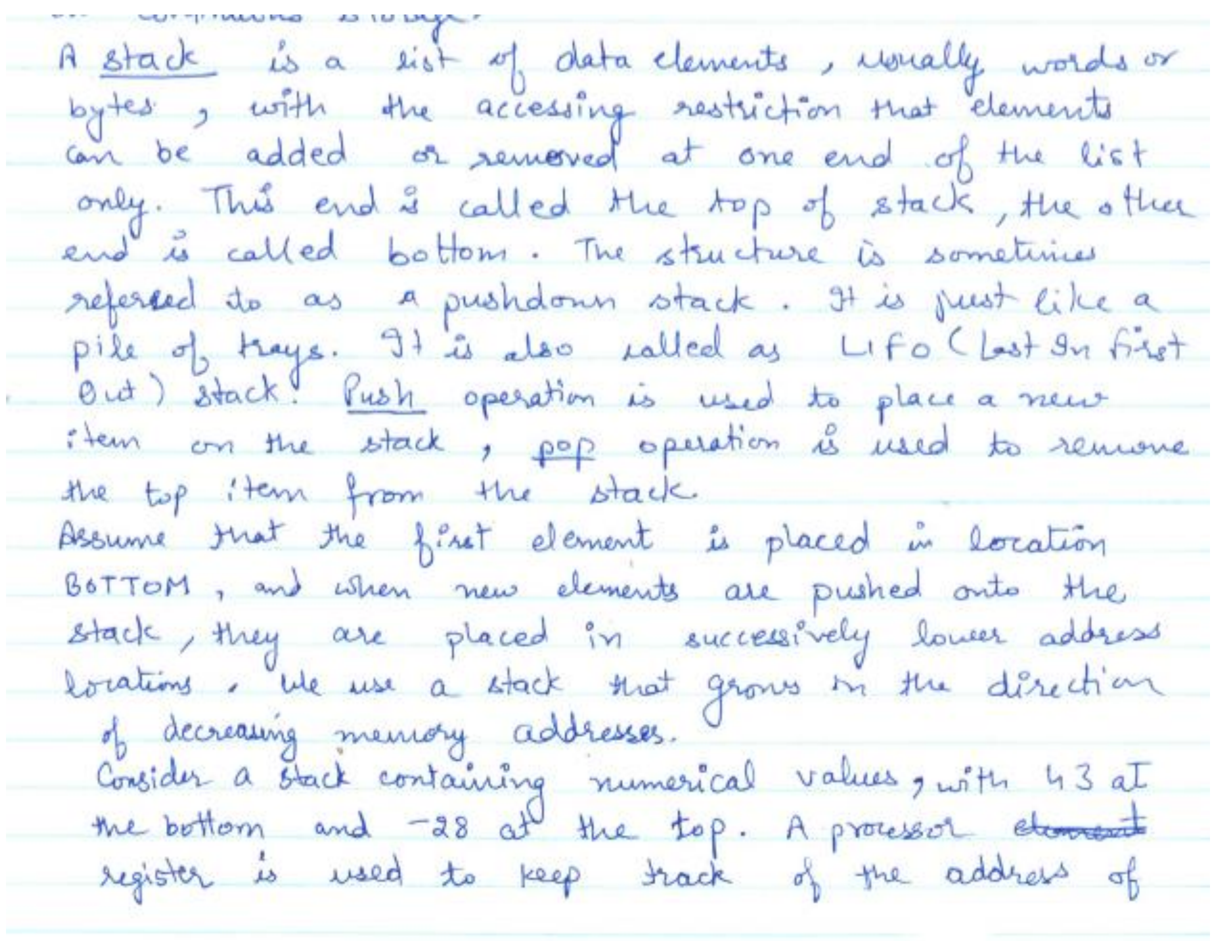
Consider moving a character code from the keyboard to the processor. Striking a key stores the corresponde character code in an 8-bit buffer register associate with the keyboard. This buffer register may be called DATAIN. To inform processor that a valid characte is in DATAIN, a status control flag, SIN, is set to 1. value of SIN is monitored. When SIN is set to 1, the processor reads the contents of DATAIN. When the charact is transferred to the processor, SIN is automatically clear to 0. If a second character is entered at the keyboar SIN is again set to 1 and process repeats.
For displaying character code on display, a buffer register, DATAOUT, and a status control flag, SOUT are used. When SOUT equals 1, the display is ready to receive a character, when SOUT is set to 1, processor transfers a character code to DATAOUT. SOUT is cleared to 0 ~~after~~ during the transfer. ~~No~~, when display device is ready to receive a second character, SOUT is again set to 1.
The buffer registers DATAIN and DATAOUT and Status flags SIN and SOUT are a part of circuitry of each device commonly known as a device interface. This circuitry is connected to processor via bus.
If a program residing in CPU monitors the status of buffer registers for I/O transfers, it is known as program-controlled I/O. But it wastes lots of processor time.
This can be implemented in 2 ways:
) Port mapped IO - It uses a separate, dedicated address space and is accessed via a dedicated set of microprocessor instructions.


READWAIT Branch to READWAIT if SIN=0
Input from DATAIN to R1

Operations for transferring output to the display:

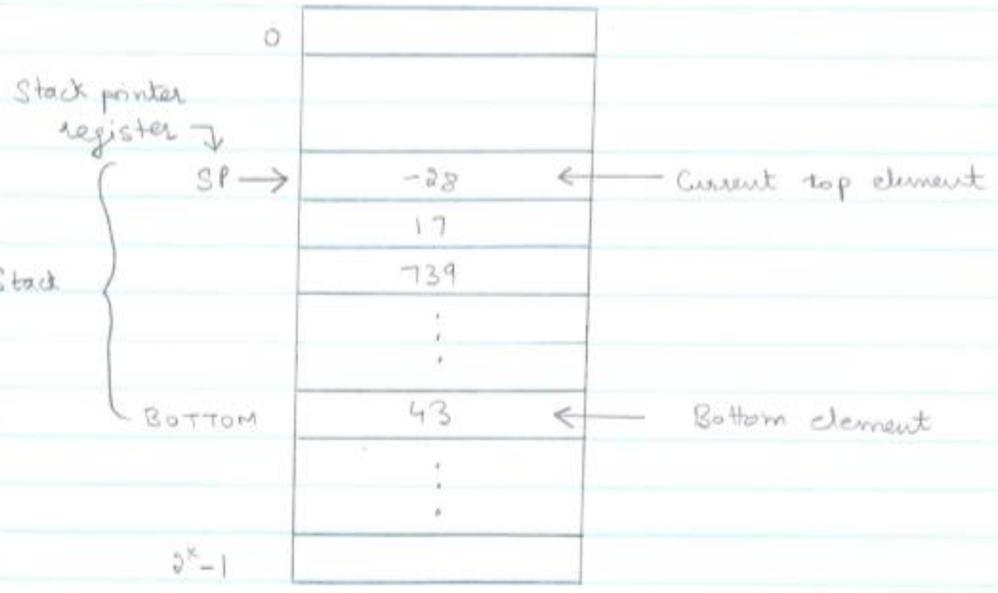
WRITEWAIT Branch to WRITEWAIT if SOUT=0
Output from R1 to DATAOUT.

Q9 Explain the concept of Stack ?


[illegible]

A stack is a list of data elements, usually words or bytes, with the accessing restriction that elements can be added or removed at one end of the list only. This end is called the top of stack, the other end is called bottom. The structure is sometimes referred to as a pushdown stack. It is just like a pile of trays. It is also called as LIFO (Last In first Out) stack. Push operation is used to place a new item on the stack, pop operation is used to remove the top item from the stack.
Assume that the first element is placed in location BOTTOM, and when new elements are pushed onto the stack, they are placed in successively lower address locations. We use a stack that grows in the direction of decreasing memory addresses.
Consider a stack containing numerical values, with 43 at the bottom and -28 at the top. A processor ~~element~~ register is used to keep track of the address of

the element of the stack that is at the Top at any given time. This register is called the stack pointer (SP)



A Stack of words in the memory

Assume a byte-addressable memory with a 32-bit word length.

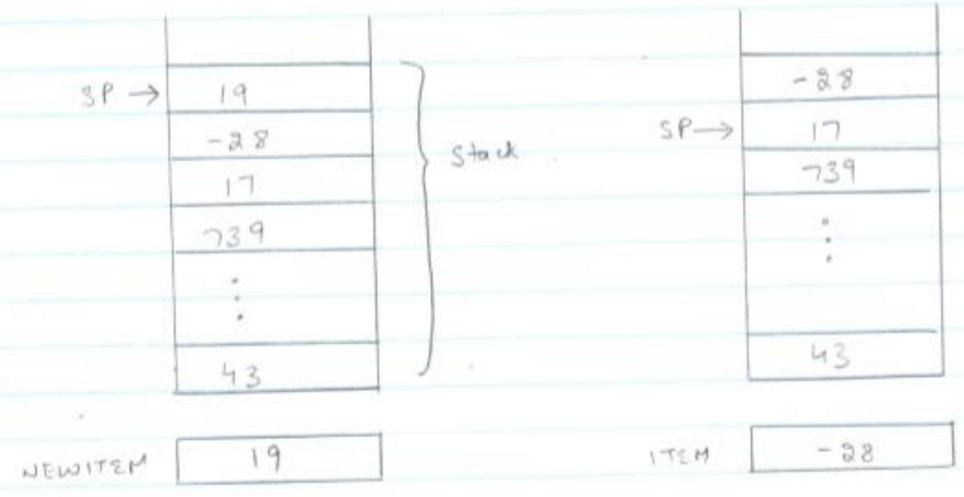
Push :
```
Subtract  #4,SP
Move      NEWITEM,(SP)
```

Pop :
```
Move  (SP),ITEM
Add   #4,SP
```

If processor has Autoincrement and Autodecrement addressing modes, ~~we~~ push and pop can be performed using single instructions :-

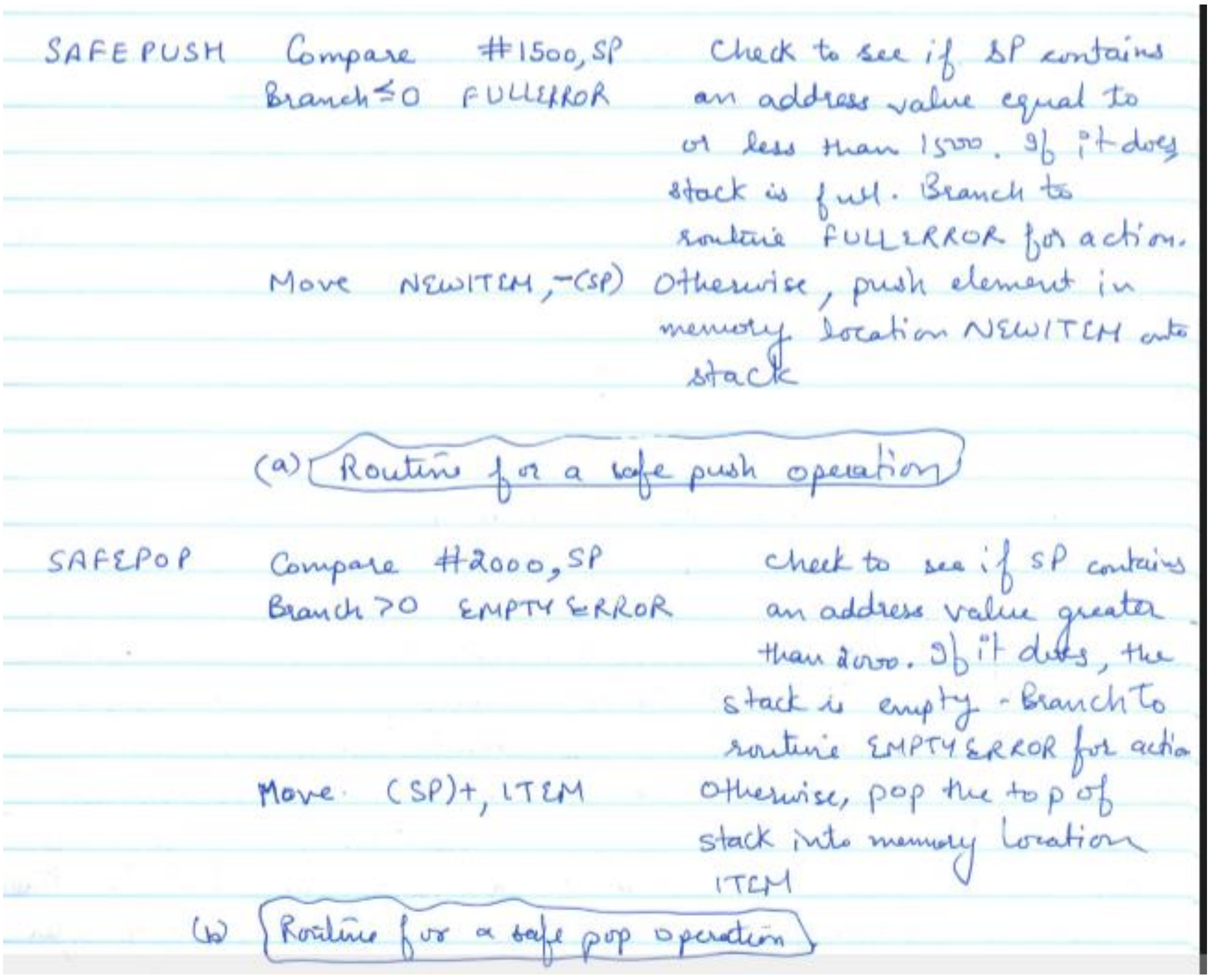
Push :- Move NEWITEM, -(SP)

Pop :- Move (SP)+, ITEM



a) After push from NEWITEM b) After pop into ITEM

* Consider stack siz from $\xrightarrow{1500 \text{ to }}$ 2000 memory location addresses.



```
SAFEPUSH   Compare    #1500,SP       Check to see if SP contains
           Branch≤0   FULLERROR      an address value equal to
                                     or less than 1500. If it does
                                     stack is full. Branch to
                                     routine FULLERROR for action.
           Move       NEWITEM,-(SP)  Otherwise, push element in
                                     memory location NEWITEM onto
                                     stack
```

(a) Routine for a safe push operation

```
SAFEPOP    Compare    #2000,SP       check to see if SP contains
           Branch>0   EMPTYERROR     an address value greater
                                     than 2000. If it does, the
                                     stack is empty - Branch to
                                     routine EMPTYERROR for action
           Move       (SP)+,ITEM     Otherwise, pop the top of
                                     stack into memory location
                                     ITEM
```

(b) Routine for a safe pop operation

Q10 Explain Subroutine?

SUBROUTINES

In a program, if a particular subtask is performed many times on different data values, such subtask is usually called a subroutine. Eg subroutine to evaluate the sine function. To save space, only one copy of the instructions that constitute the subroutine is placed in the memory, and any program that requires the use of the subroutine simply branches to its starting location. This branching to a subroutine is called as calling the subroutine. The instruction that performs this branch operation is named a Call instruction. The subroutine is said to return (resume execution, continuing immediately after the call instruction) to the program that called it by executing a Return instruction. Contents of PC must be saved by the Call instruction to enable correct return to the calling program.

The method followed to call and return from subroutines is referred to as subroutine linkage method.

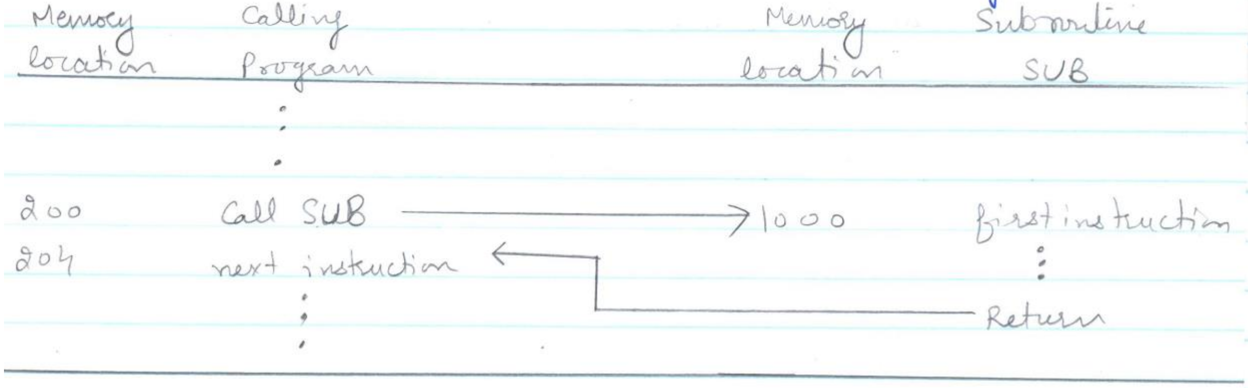
Eg- save a return address in a specific location like a link register. When subroutine completes its task, the Return instruction returns to the calling program by branching indirectly through the link register.

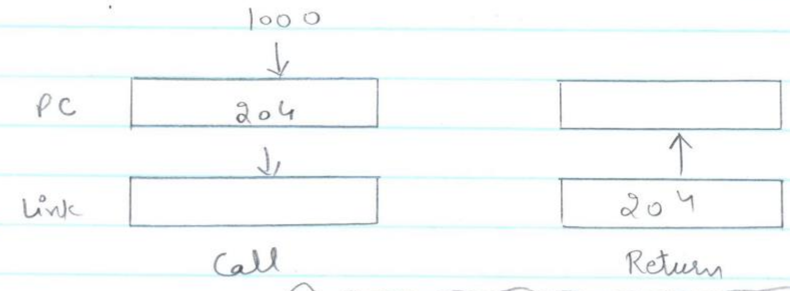
Call instruction

- Store contents of PC in link register
- Branch to the target address specied by the instruction.

Return instruction

- Branch to the address contained in the link register.





Subroutine linkage using a link register

Subroutine nesting and the processor stack.

Subroutine nesting - when one subroutine calls another.
The return address of second call is also stored in link register after saving contents of link register at other location.
Return addresses are generated and used in a LIFO order. A particular register, is designated as stack pointer, SP, The SP points to a stack called processor stack. The Call instruction pushes the contents of PC onto processor stack & loads subroutine address into PC. The Return instruction pops the return address from the processor stack into PC.