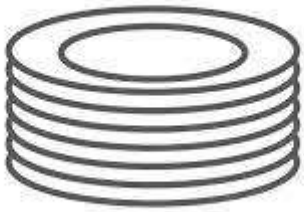
STACK

Definition: -
A stack is an ordered list (i.e., listed in a specific order) in which insertion and deletion are done at one end, called top.

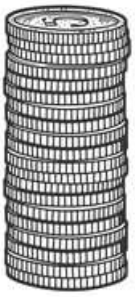
A. Representation of Stack based on Real Physical Entity

On real physical entity we can take example of

1. Stack of Dishes,



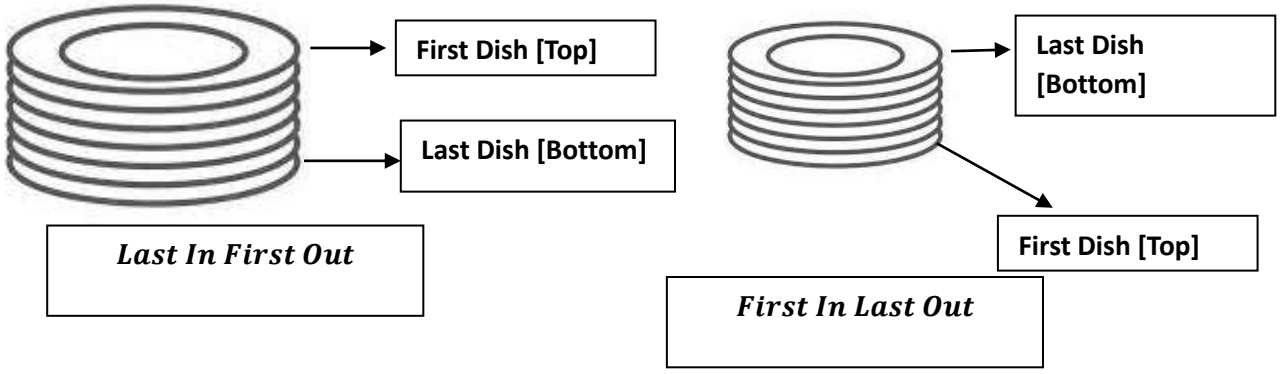
2. Stack of Pennies,



etc.

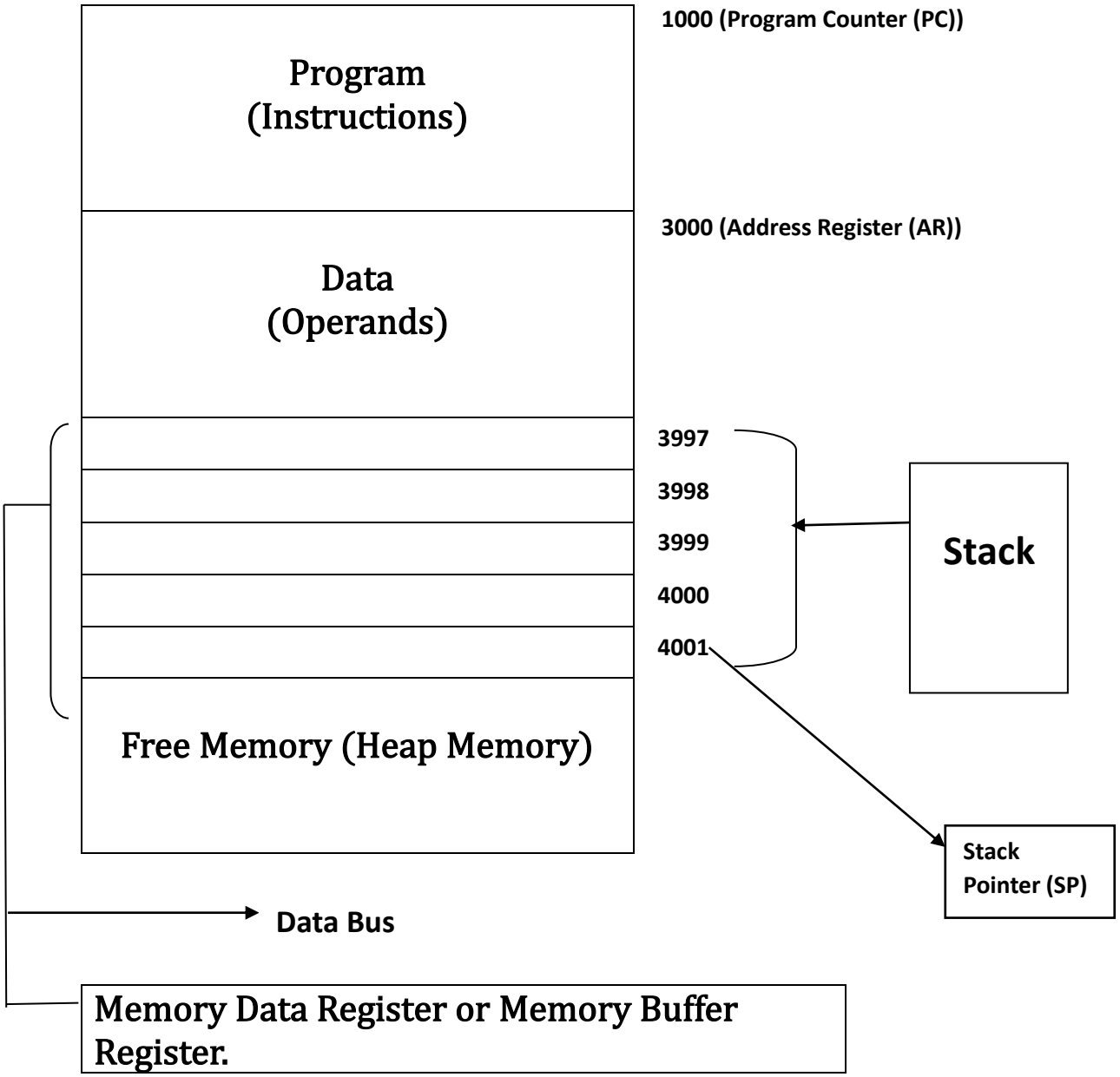
Taking the first examples: Stack of Dishes, we can remove the first dish to keep a new dish on top follows: ***First in Last Out*** mechanism.

Again, while arranging the stack of dishes, we keep one dish upon another i.e., the mechanism called "***Last In***" and now if we want to remove a dish we should remove the first dish, this mechanism is called "***First Out***", hence whole mechanism is called: "***Last in First Out***". Or if we say last dish is First Dish and First Dish as Last Dish, it will become "***First In Last Out***" mechanism.



B. Representation of Stack Based on Computer architecture

B.A. Random Access Memory Stack



1. Program Counter: - It is a register that manages the memory address of the instructions that to be executed next. That is simply a manager which tells CPU that what instruction to be executed.

1. a. Program Instructions: The codes that are stored in memory are stored as instructions in Assembly Language. Assembly Language instructions are called '*machine instructions*.

   Such as: $N = I + J + K, where\ I\ is\ stored\ in\ address\ location\ 201, J\ in\ 202\ , K\ in\ 203\ and\ N\ in\ 204.$

   Instructions:

   1. Load the contents of the location 201 in Accumulator.
   2. Add the contents of location 202 to the Accumulator.
   3. Add the contents of location 203 to the Accumulator.
   4. Store the contents of the Accumulator in the location 204.

   In Assembly Language it looks like:

   ```
   LDA I
   ADD J
   ADD K
   STA N
   ```

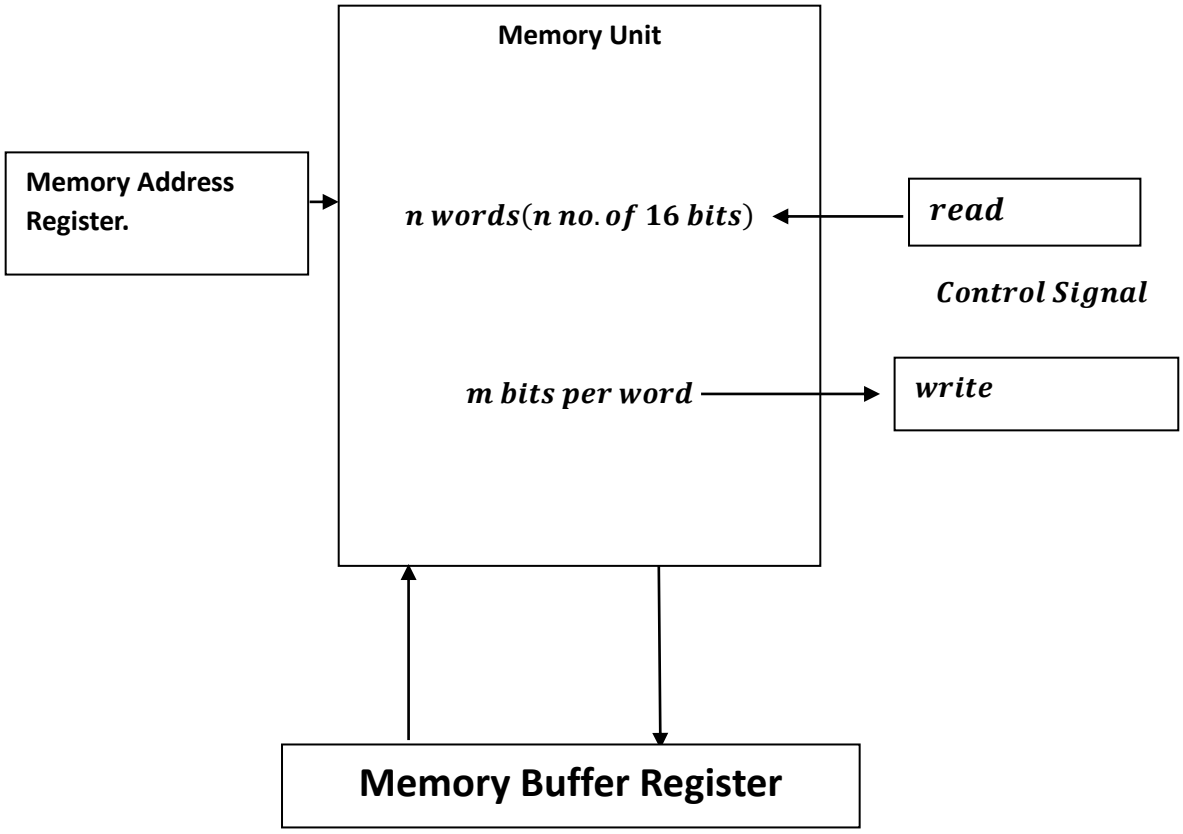
$Where\ , LDA \rightarrow Load\ Accumulator\ and\ STA \rightarrow Store\ Accumulator.$

And there is an assembler that converts the high-level language to the low-level assembly language. Later such low level language are converted to binary language which is better known as "Machine Level Language", when such instructions are executed by the processor.

- Accumulator: An accumulator is a register which acts as intermediate storage of arithmetic and logic data in a computer's central processing unit (CPU).

2. Address Register, better known as Memory Address Register:

Memory Address Register looks like:



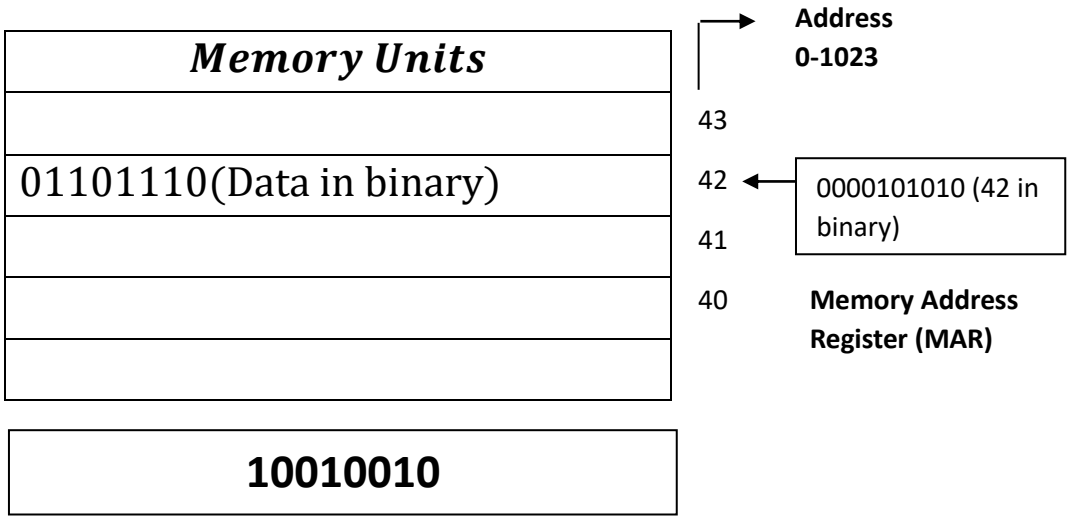
1. The information transfer to and from registers in memory and the external environment is communicated through one common register called the memory buffer register (other names are information register and storage register).

2. When the memory unit receives a write control signal, the, internal control interprets the contents of the buffer register

to be the bit configuration of the word to be stored in a memory register.

3. With a read control signal, the internal control sends the word from a memory register into the buffer register.

4. In each case the contents of the address register specify the particular memory register referenced for writing or reading.



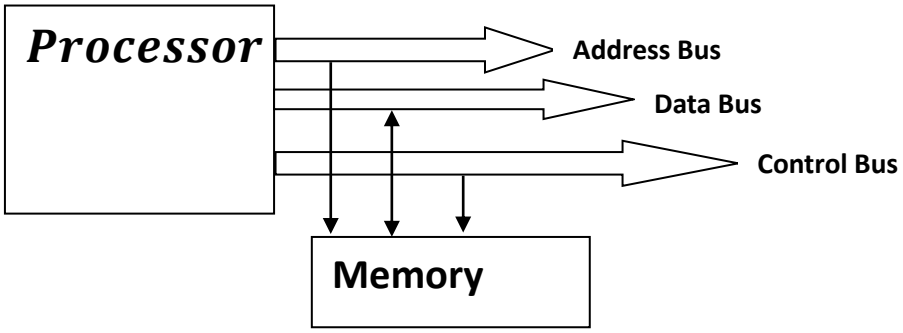
***Memory Buffer Register*(*MBR*)**

Consider a memory unit of 1024 words with 8 bits per word.

To specify 1024 words, we need an address of 10 bits, since $2^{10} = 1024$. Hence the memory has 1024 registers with assigned address numbers from 0 *to* 1023.

- **MAR connects directly to the address bus.**
- **MBR connects directly to the data bus.**

- Registers exchange data with MBR.

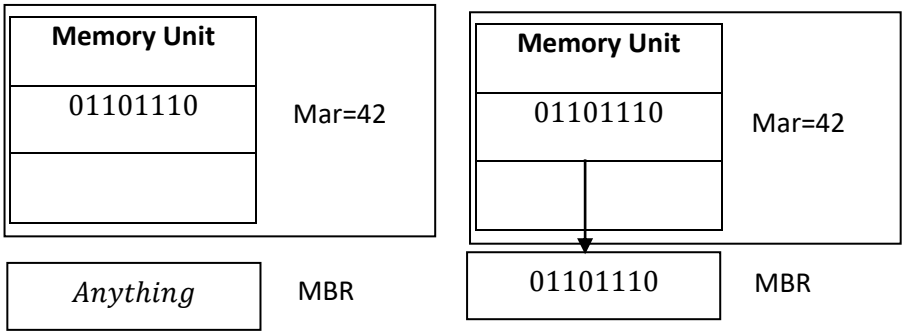


Read Operation

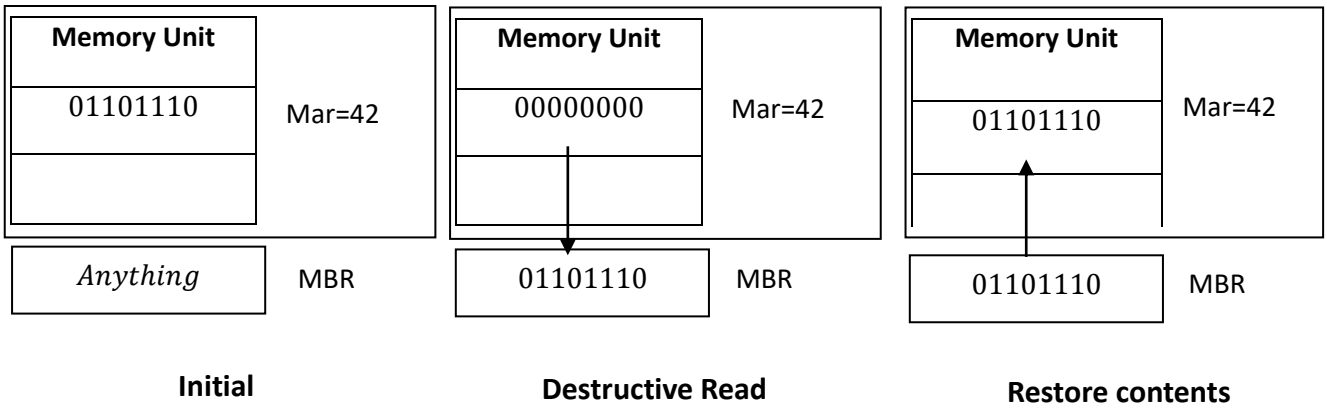
Read Operation in Memories Constructed by Semiconductor ICs

1. ***Transfer the address bits of the selected word into address lines of address bus connected with MAR.***

2. ***Activate the read control input.***



Read Operation in Memories Constructed by Magnetic Core



A destructive read operation transfers the selected word into MBR but leaves the memory register with all 0's. Normal memory operation requires that the content of the selected word remain in memory after a read operation. Therefore, it is necessary to go through a restore operation that writes the value in MBR into the selected memory register. During the restore operation, the contents of MAR and MBR must remain unchanged.

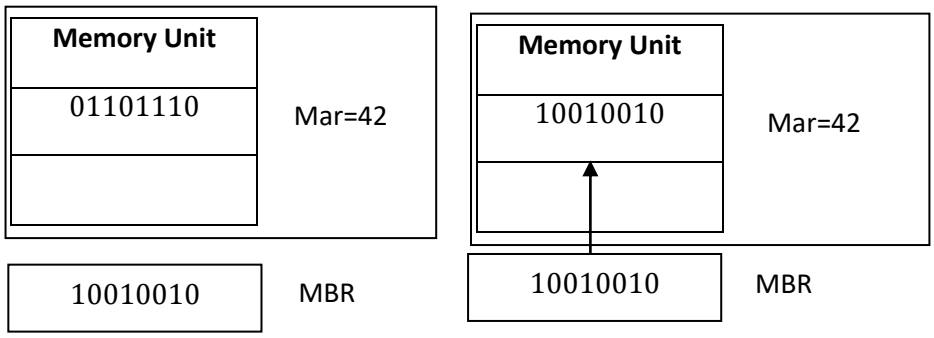
Write Operation

Write Operation in Memories Constructed by Semiconductor ICs

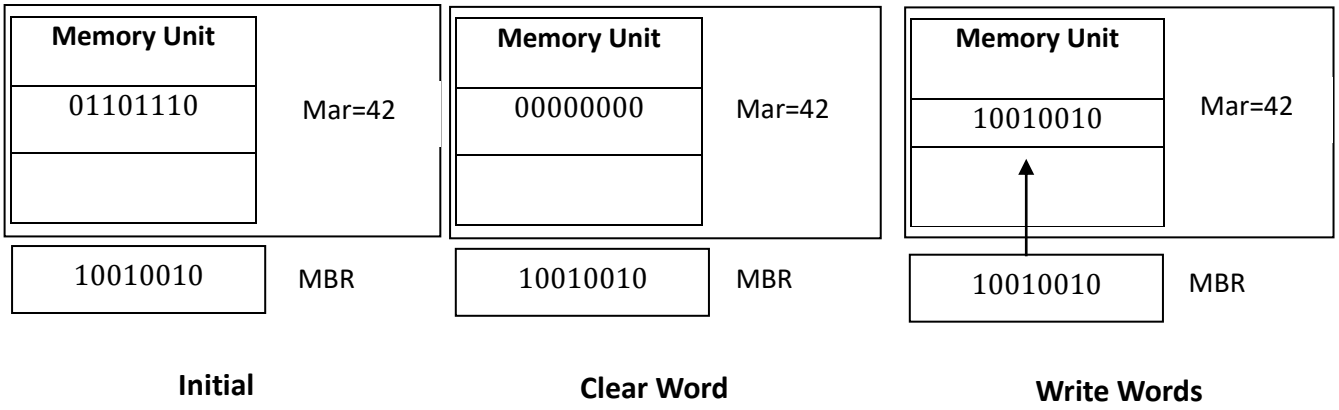
1. ***Transfer the address bits of the selected word into address lines of address bus connected with MAR.***

2. ***Transfer the data bits of the word into the data lines of the data bus into MBR.***

3. ***Activate the write control unit.***



Write Operation in Memories Constructed by Magnetic Core



A write control input applied to a magnetic-core memory causes a transfer of information. To transfer new information into a selected register, the old information must first be erased by clearing all the bits of the word to 0. After this done, the content of the MBR can be transferred to the selected word. MAR must not change during the operation to ensure that the same selected word that is cleared is the one that receives the new information.

Words, Bytes and Bits

$8\ bits = 1\ byte.$

$16\ bits\ or\ 2\ bytes \rightarrow Half\ Word.$

$32\ bits\ or\ 4\ bytes \rightarrow Single\ Word\ /Full\ Word.$

$64\ bits\ or\ 8\ bytes \rightarrow Double\ Word.$

$128\ bits\ or\ 16\ bytes \rightarrow Quad\ Word.$

Normally, 1 word or a word represents 16 bits in representation.

Data Operand

Suppose we have:

$$\boldsymbol{int\ a\ =\ 10};$$

$$\boldsymbol{int\ b = 20};$$

$$\boldsymbol{int\ c = b};$$

Here, a, b , c are all operands that will be further used for operation such as int sum = a + b.

The operands are those variables that are worked by operator to produce a result and such addresses of operands are stored in data operands section of memory.

Hence memory address register and memory buffer register work with those data operands while RAM is in operation.

Memory Units	MAR
10 (A)	0x19
20 (B)	0x2A

And MBR will hold the contents of the memory in word read from or written to memory while in we process.

STEP BY STEP PROCESS

1. High level programming language converted to programming instructions by an assembler.

2. These instructions are provided address by Program Counter.

3. The operand data gets stored in Data Operand section as shown above and MAR will hold the address and MBR will hold the data i.e., content of the memory in word while we process the data.

4. There is another register associated with the CPU that is Instruction Register (IR) which receives these operation code of instructions communicated through address bus connected with Program Counter (PC) which holds addresses of operation code instructions and the operations can be defined as: ADD (Addition), SUB (Subtraction), MUL (Multiplication), etc. and the operation which will be done with operands such addresses is fetched by MAR (Memory Address Register) and a decoder associated with

Instruction Register (IR) supplies each output to the instruction .

Stack

Now, Suppose we need to ADD B. Here ADD is OPCODE, B is the variable or the operand stored in Data Operand section. Now the instruction is set in STACK through PUSH and POP operation .

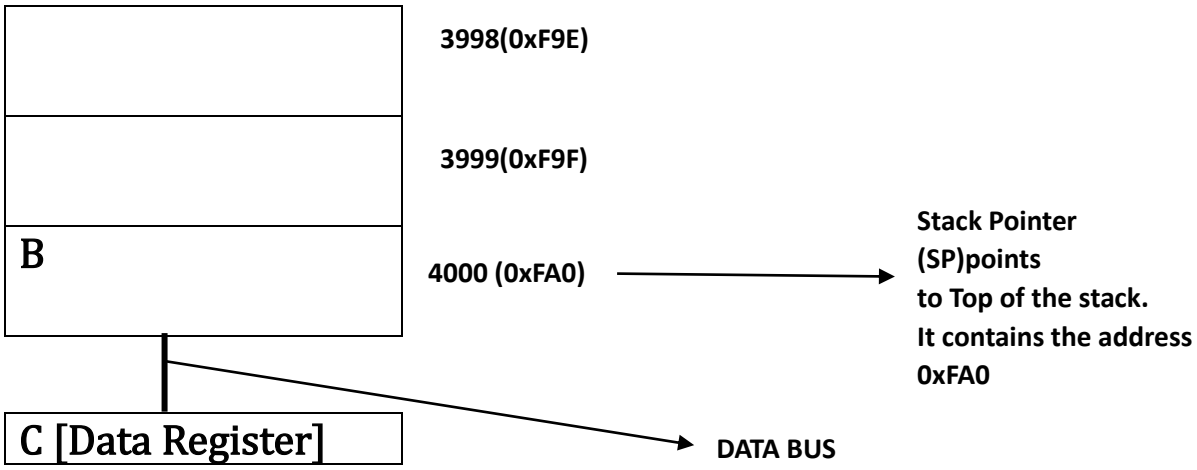
Suppose we have:

Opcodes: ADD, MUL stored in address 0x19 and 0x23.

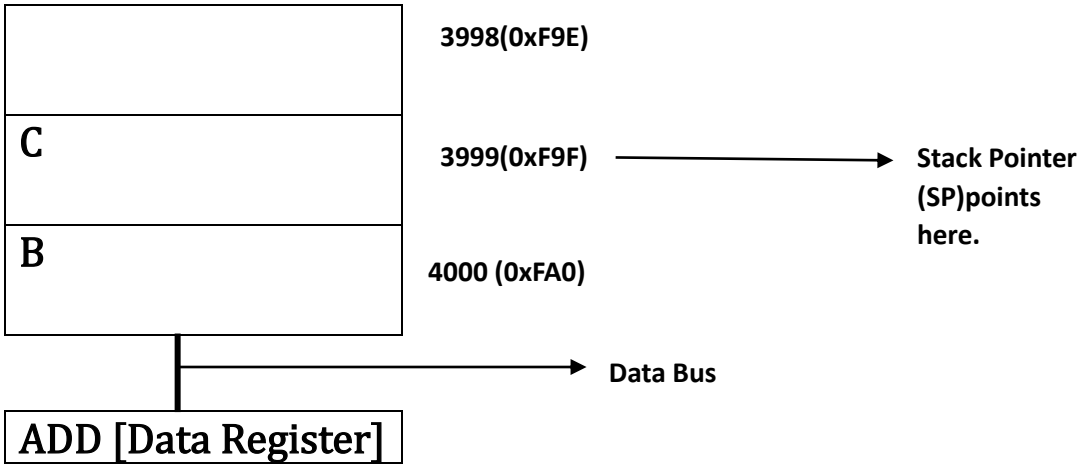
Data Operands: D=10, B=12, C=13 stored in address 0x21, 0x24 and 0x49.

Now we have to perform: $(C + B) * D$

Stack Operation: PUSH B



Stack Operation: PUSH C stored in Data Register



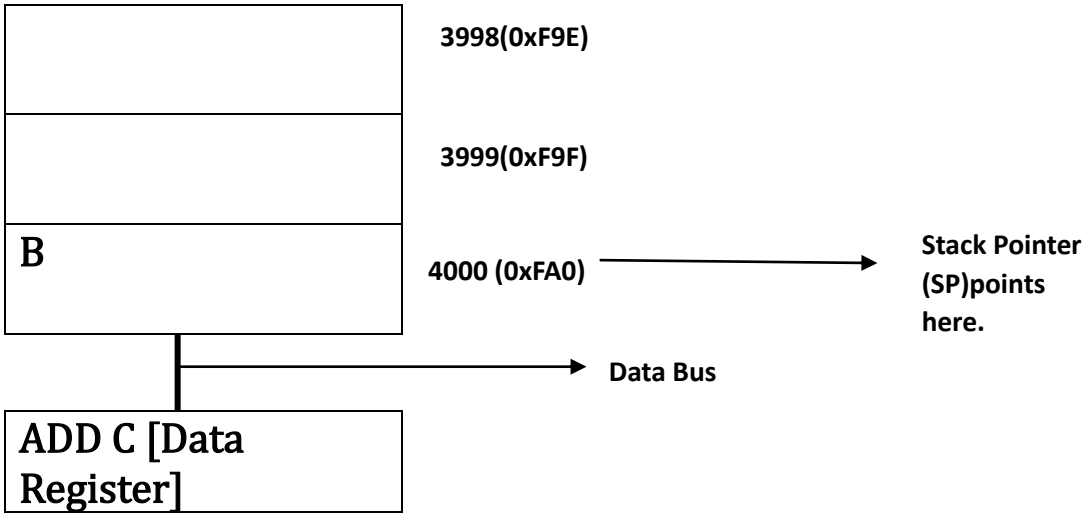
Therefore, Stack Pointer gets decremented (SP-1 = (4000-1) = 3999 or 0xF9F).

Now it will execute Zero Address Instruction:

As Add is an opcode (operation code) and then it will generate:

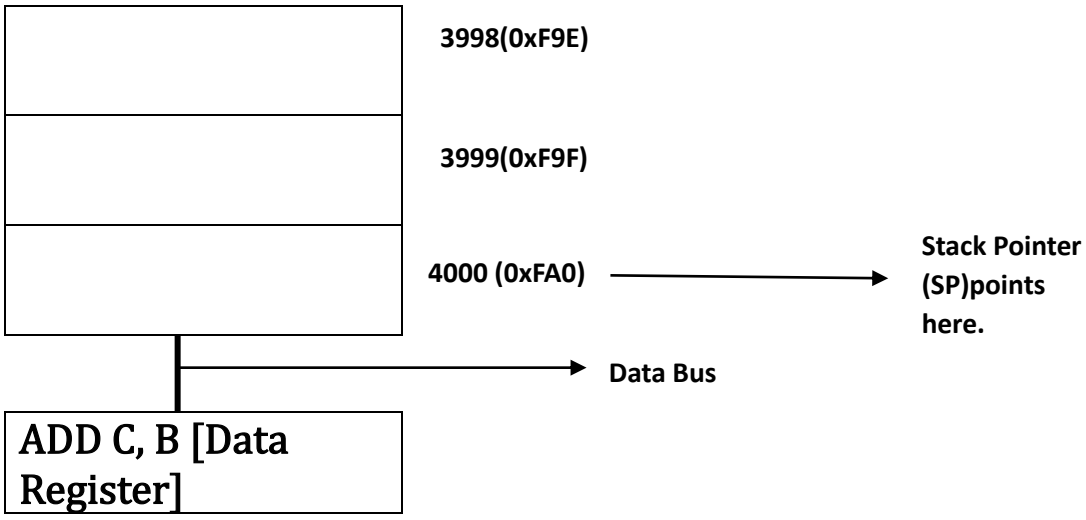
Add C, B and the instruction will get minimized(Zero Address), i.e. without PUSH and POP.

Stack Operation: POP C stored in Memory

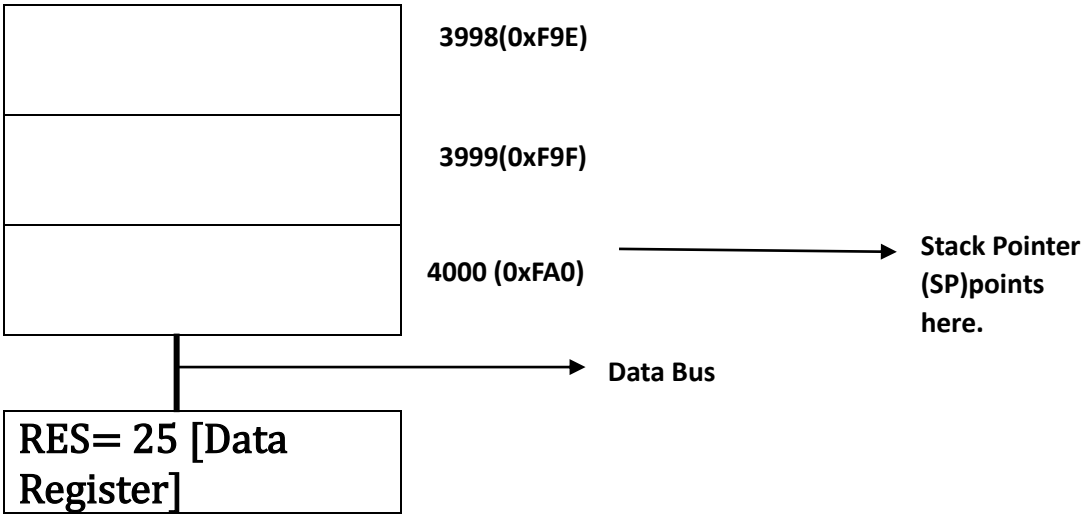


Therefore, Stack Pointer gets incremented (SP+1 = (3999+1) = 4000 or 0xFA0).

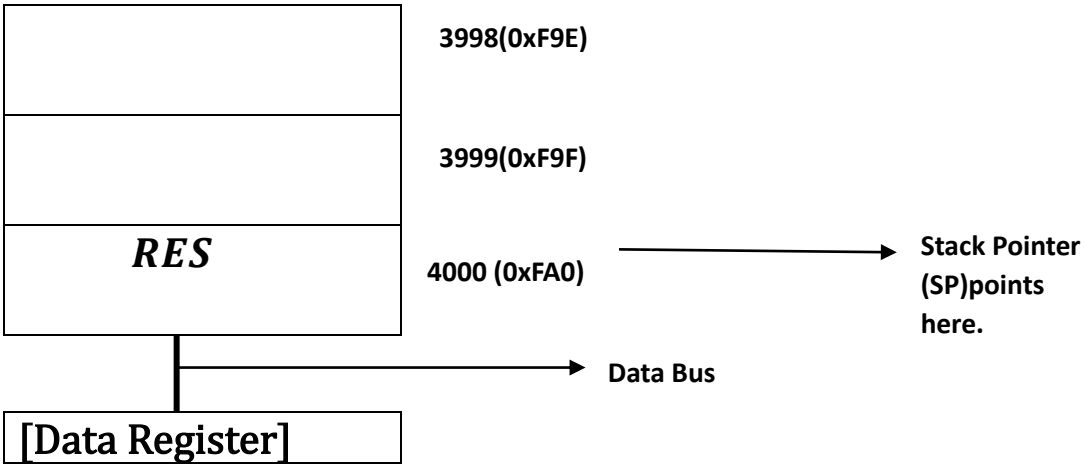
Stack Operation: POP B stored in Memory



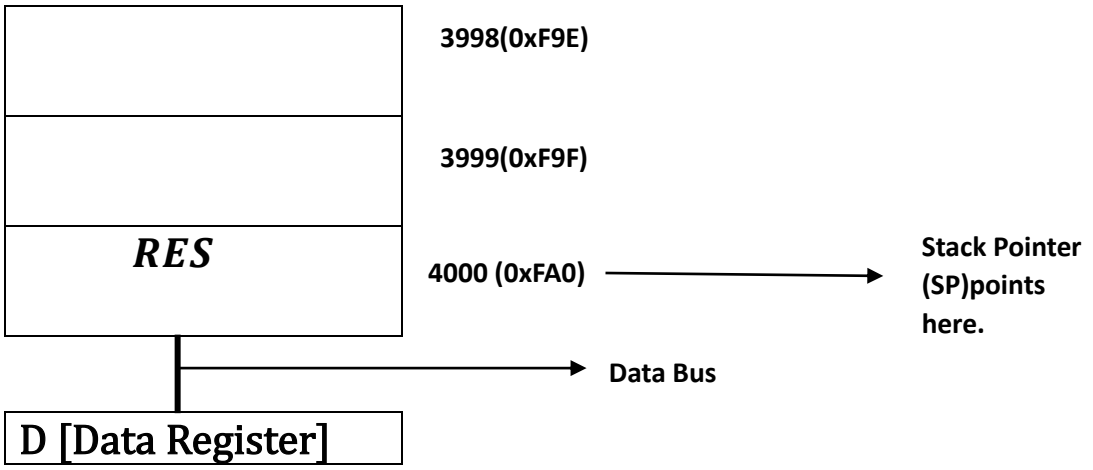
Therefore, Stack Pointer points at the last address **4000**, While the instruction gets completed i.e. ADD C,B (=13+12)and it goes to Instruction Register associated with its address and decoder associated with Instruction Register of CPU provides the out put , say output = RES(=25) . The value gets again stored at Data Register.



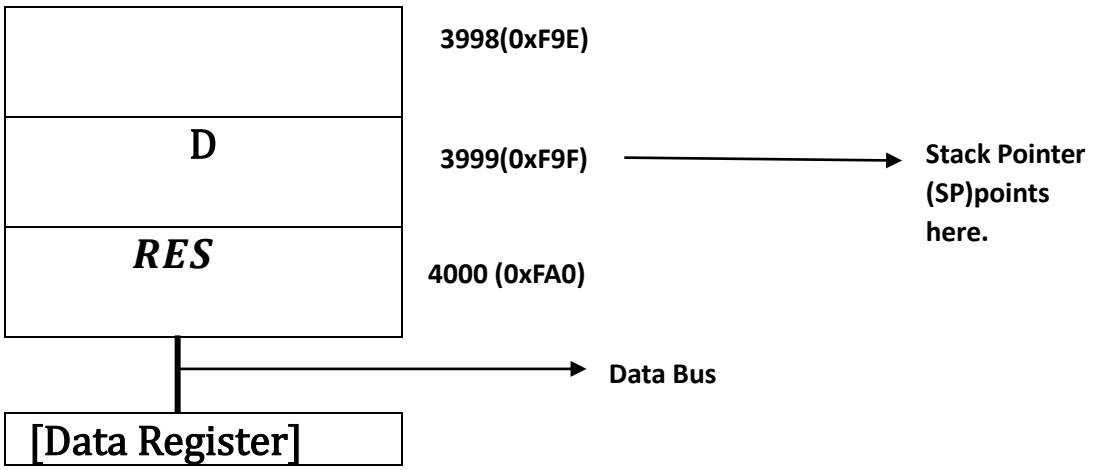
And from Data register it again Pushed into the stack for further process.



Now Data Register contains D.

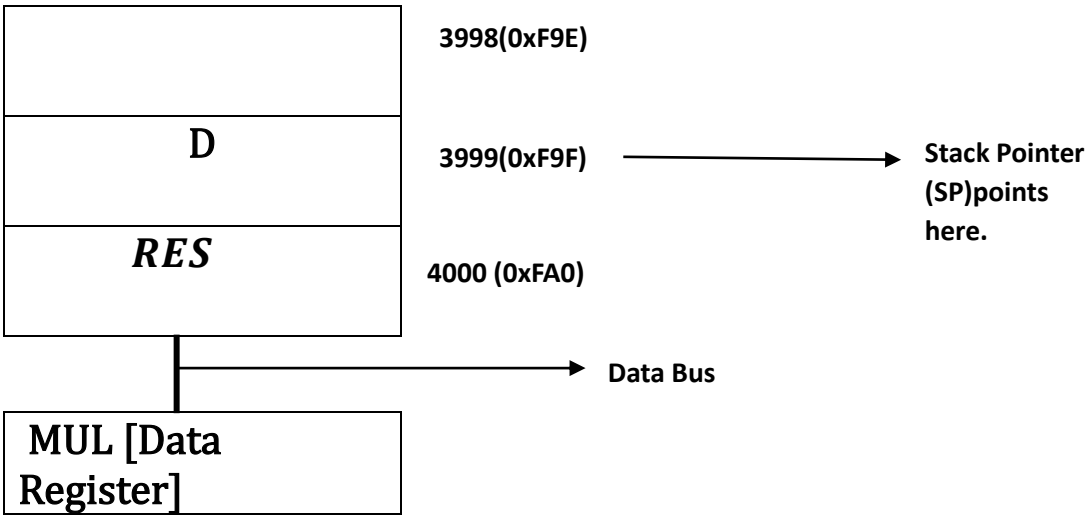


STACK OPERATION PUSH D STORED IN DATA REGISTER.



Therefore, Stack Pointer gets decremented (SP-1 = (4000-1) = 3999 or 0xF9F).

Now Data Register will contain MUL opcode.

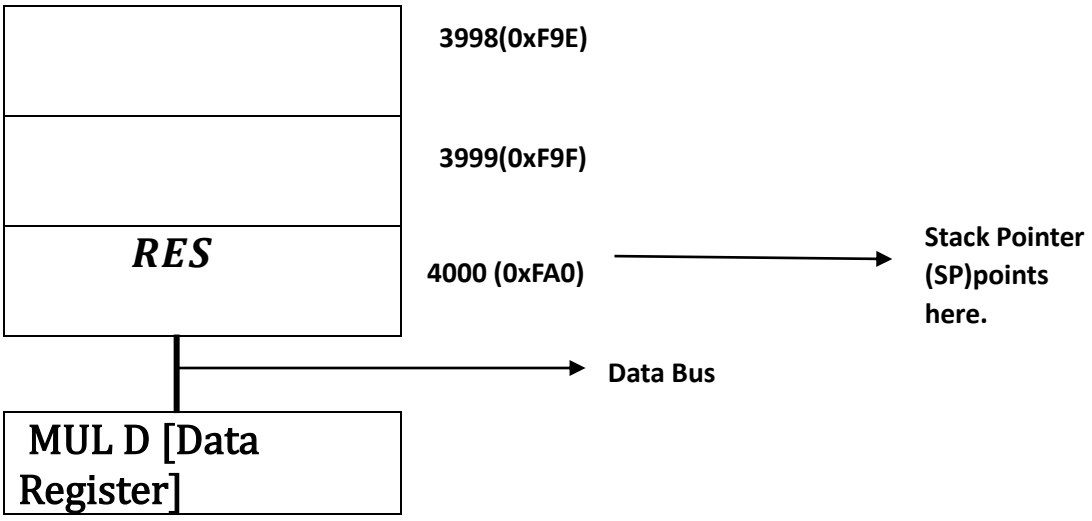


Now it will execute Zero Address Instruction:

As MUL is an opcode (operation code) and then it will generate:

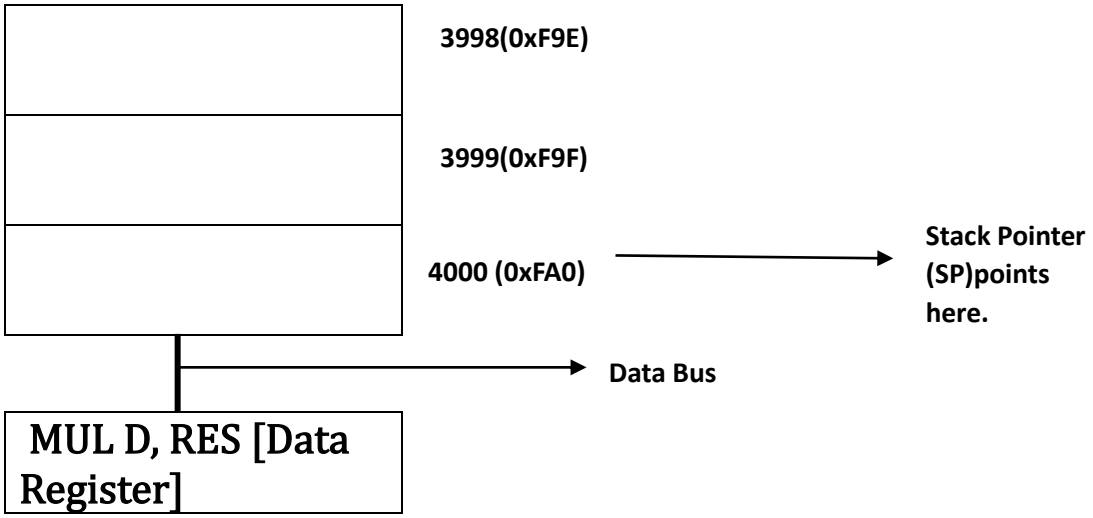
MUL D, RES and the instruction will get minimized(Zero Address), i.e. without PUSH and POP.

Stack Operation: POP D stored in Memory



Therefore, Stack Pointer gets incremented (SP+1 = (3999+1) = 4000 or 0xFA0).

Stack Operation: POP RES stored in Memory



Therefore, Stack Pointer points at the last address **4000**, While the instruction gets completed i.e., MUL D, RES (**=10 × 25 = 250**)and it goes to Instruction Register associated with its address and decoder associated with Instruction Register of CPU provides the output. The value gets again stored at Data Register and then it gets interpreted to user.

Q) What will happen when stack gets Empty or UNDERFLOW?

ANS: There are two flags working together:

1. FULL
2. EMPTY

While stack gets empty, stack pointer will become 0.

Now, as soon as Stack Pointer becomes 0, FULL flag will become 0 i.e., false, and EMPTY flag will become 1 i.e., true.

It can also be represented as:

$$If\ (SP\ ==\ 0)\{$$

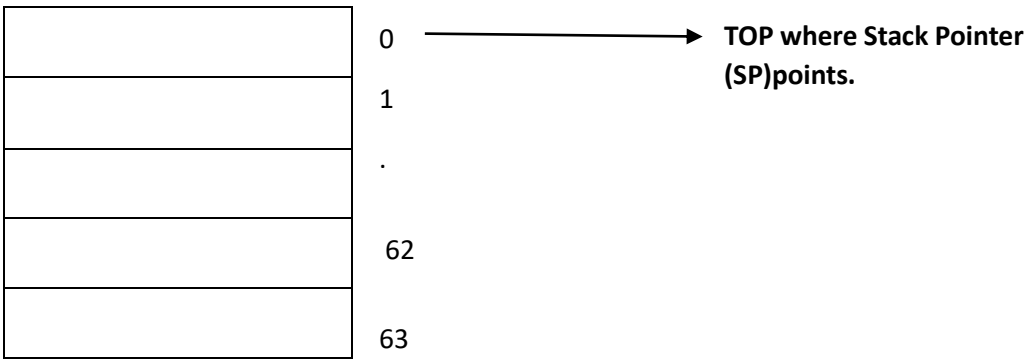
$$EMPTY\ \leftarrow 1$$

$$FULL\ \leftarrow 0$$

$$\}$$

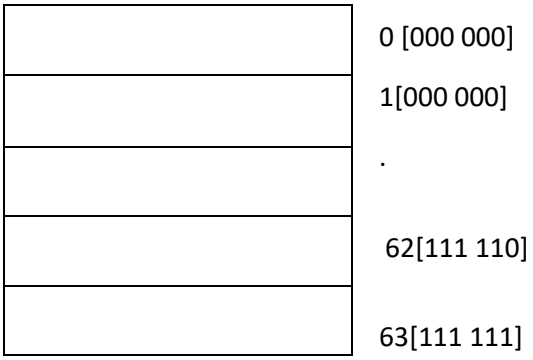
Q) What will happen when stack gets Full or Overflow?

Suppose we have stack contains 64-word address looks like:



Lets say 0 is top then Push operation occurs: SP = SP+1 (Increment of Stack Pointer) and POP operation occurs: SP = SP-1 (Decrement of Stack Pointer). 63 is the last bit represented with

Binary digits $\mathbf{111111}$ $(\boldsymbol{Six\ Ones = Six\ Bits})$. Then the top $\mathbf{0}$ $\boldsymbol{will\ be\ represented\ in}$ $\mathbf{6}$ $\boldsymbol{bits\ or}$ $\mathbf{6}$ $\boldsymbol{zeroes}$ $[\mathbf{000\ 000}]$.



Therefore at 64 it will be $\mathbf{1\ 000\ 000}$ $\boldsymbol{i.e.}$ $\mathbf{1}$ $\boldsymbol{and\ six\ zeroes}$.

Hence $\mathbf{1}$ $\boldsymbol{will\ be\ discarded\ and\ six\ zeroes\ will\ be\ accepted}$.

Six zeroes represent zero th location i.e., Top of the stack, where the last element will be inserted when the stack gets filled.

This tells Stack Pointer will become zero again and if stack pointer becomes zero here:

$\boldsymbol{Full} \rightarrow \mathbf{1}(\boldsymbol{true})$

$\boldsymbol{Empty} \rightarrow \mathbf{0}\ (\boldsymbol{False})$.

i.e.,

$If\ (SP\ ==\ 0)\{$

$FULL\ \leftarrow 1$

$EMPTY\ \leftarrow 0$

}

<u>SUPPOSE ITS OPPOSITE IF TOP POINTS TO 63 THEN:</u>

Push = Stack Pointer (SP) -1(Decrement)

Pop = Stack Pointer (SP) +1(Increment)

Therefore when it reaches 0, next decrement will cause :

-1 , Representation of 6 bit of 1 = 000 001 , One's complement of 1 = 111 110 and Two's complement (111 110 +1) = 111 111. And we know 111 111 is 63 , hence again stack will point at 63.

Again, it satisfies:

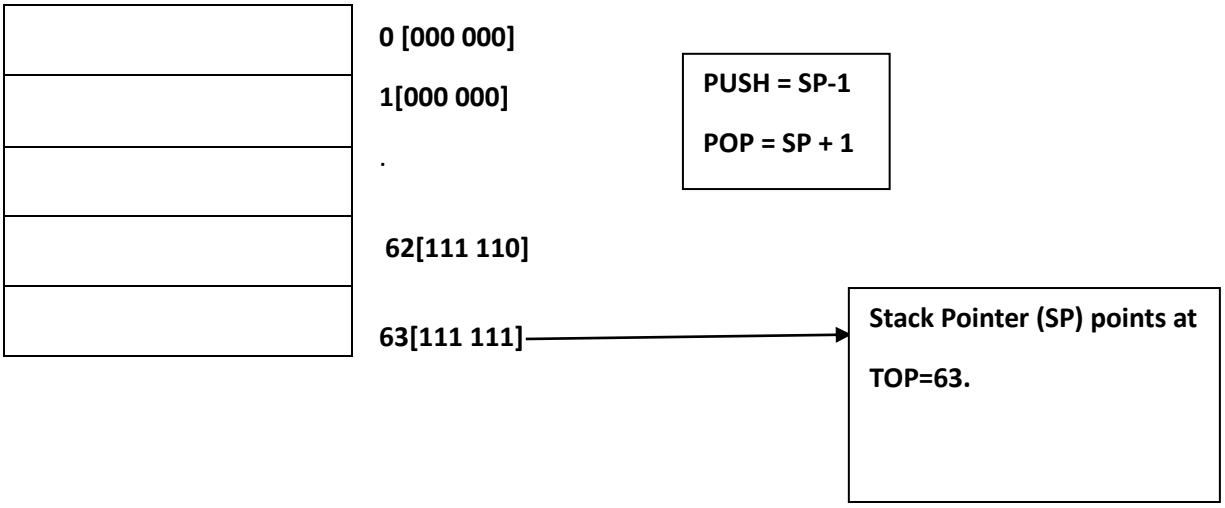
$If\ (SP\ ==\ 0)\{$

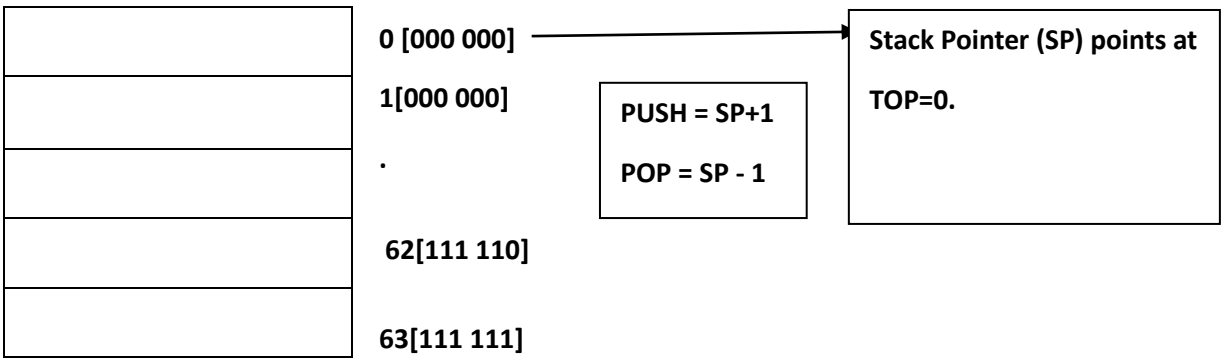
$FULL\ \leftarrow 1$

$EMPTY\ \leftarrow 0$

}

Note: - It does not matter that Stack Pointer will always decrement or always increment, it depends upon our representation of our Memory, whether we are pointing stack at first or last index address.





Again, if we notice, Same way A REGISTER STACK works like MEMORY STACK as Registers are also a memory (smallest data storage unit).

C. Stack Representation Based on Applications:

1. Balancing of Symbols
2. Infix to post fix conversion.
3. Evaluation of post-fix expression.
4. Implementing function call (including recursion).
5. Finding of spans in Stock Market.
6. Page Visited history in a Web Browser [History Section of Web Browser].
7. Back Buttons in HTML. That is all pages in web site are stored like Stacks and back button helps the pages to pop out the Page visited and Push in the Last Page visited [Last in First Out].
8. Undo sequence in a Text editor.
9. Matching Tags in HTML and XHTML.
10. In cross platform application we can set windows or pages in Stack Format for navigation and when we navigate to different pages in cross platform applications (android, iOS), the push and pop are being implemented.