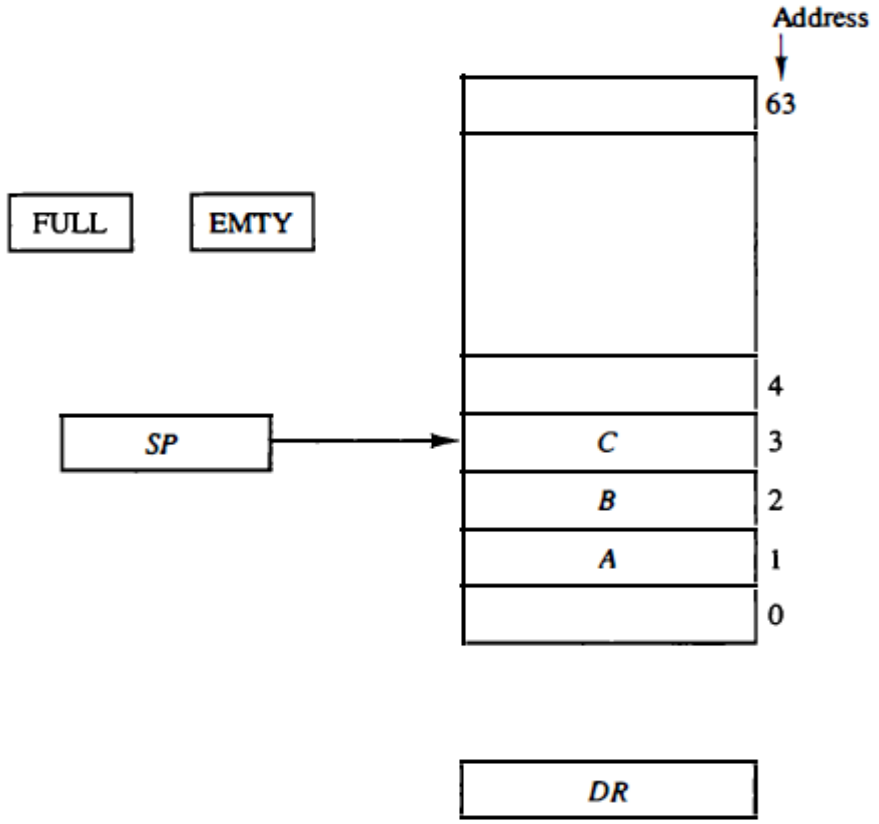
Stack Organization

- A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved, so that stack is known as Last-in-First-Out (LIFO).
- The stack in digital computers is essentially a memory unit with an address register that can count only (after an initial value is loaded into it).
- The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.
- The two operations of a stack are the insertion and deletion of items.
- The operation of insertion is called Push (or push-down) because it can be thought of as the result of pushing a new item on top.
- The operation of deletion is called Pop (or pop-up) because it can be thought of as the result of removing one item so that the stack pops up.
- However, nothing is pushed or popped in a computer stack. These operations are simulated by incrementing or decrementing the stack pointer register.

Register Stack

- A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers.



Block Diagram of a 64 word stack

- As shows the organization of a 64-word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack.
- Three items are placed in the stack: A, B, and C, in that order. Item C is on top of the stack so that the content of SP is now 3.
- To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address 2.
- To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack.
- In a 64-word stack, the stack pointer contains 6 bits because 2^6 = 64. Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary).
- When 63 is incremented by 1, the result is 0 since 111111 + 1 = 1000000 in binary, but SP can accommodate only the six least significant bits.
- Similarly, when 000000 is decremented by 1, the result is 111111 .
- The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items.
- DR is the data register that holds the binary data to be written into or read out of the stack.

- Push: Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full.
- If the stack is not full (if FULL = 0), a new item is inserted with a push operation.
- The push operation is implemented with the following sequence of micro-operations

```
SP <- SP + 1                          //Increment stack pointer
M [SP] <- DR                          //Write item on top of the stack
If (SP = 0) then (FULL <--1)          //Check if stack is full
EMTY <--0                             //Mark the stack not empty
```

- The stack pointer is incremented so that it points to the address of the next-higher word.
- A memory write operation inserts the word from DR into the top of the stack. Note that SP holds the address of the top of the stack and that M [SP] denotes the memory word specified by the address presently available in SP.
- The first item stored in the stack is at address 1 The last item is stored at address 0. If SP reaches 0, the stack is full of items, so FULL is set to 1.
- This condition is reached if the top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in location 0. Once an item is stored in location 0, there are no more empty registers in the stack. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 0.

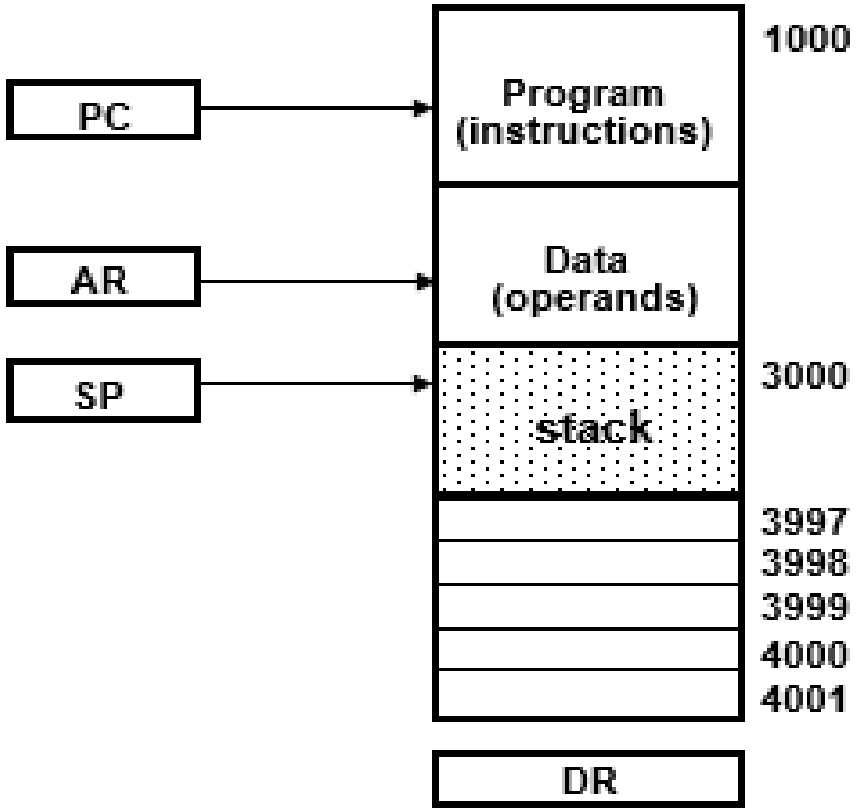
- Pop: A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequence of micro-operations:

```
DR <--M [SP]                    //Read item from the top of stack
SP <--SP – 1                    // Decrement stack pointer
If (SP = 0) then (EMTY <--1)    //Check if stack is empty
FULL <--0                       // Mark the stack not full
```

- The top item is read from the stack into DR .
- The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1. This condition is reached if the item read was in location 1. Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP.

Memory Stack

- A stack can exist as a stand-alone unit as in Fig. or can be implemented in a random-access memory attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer.
- shows in the fig. a portion of computer memory partitioned into three segments: program, data, and stack.



- The program counter PC points at the address of the next instruction in the program.
- The address register AR points at an array of data. The stack pointer SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory.
- PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or pop items into or from the stack.
- The initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000 , the second item is stored at address 3999, and the last address that can be used for the stack Is 3000. No provisions are available for stack limit checks.

➢ Stack limits: Most computers do not provide hardware to check for stack overflow (full stack) or underflow (empty stack). The stack limits can be checked by using two processor registers: one to hold the upper limit (3000 in this case), and the other to hold the lower limit (4001 in this case). After a push operation, SP is compared with the upper-limit register and after a pop operation, SP is compared with the lower-limit register.

Reverse Polish Notation

- A stack organization is very effective for evaluating arithmetic expressions. The common mathematical method of writing arithmetic expressions imposes difficulties when evaluated by a computer. The common arithmetic expressions are written in infix notation, with each operator written between the operands. Consider the simple arithmetic expression

$$A \times B + C \times D$$

- The star (denoting multiplication) is placed between two operands A and B or C and D. The plus is between the two products.
- The Polish mathematician Lukasiewicz showed that arithmetic expressions can be represented in prefix notation . This representation, often referred to as Polish notation, places the operator before the operands. The postfix notation, referred to as reverse Polish notation (RPN), places the operator after the operands. The following examples demonstrate the three representations

A + B	Infix notation
+ AB	Prefix or Polish notation
AB +	Postfix or reverse Polish notation

- The reverse Polish notation is in a form suitable for stack manipulation. The expression
- A × B + C × D is written in reverse Polish notation as

$$AB \times CD \times +$$