1. Stack Organization:

- A stack or last-in first-out (LIFO) is useful feature that is included in the CPU of most computers.
- Stack:
    - A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.
- The operation of a stack can be compared to a stack of trays. The last tray placed on top of the stack is the first to be taken off.
- In the computer stack is a memory unit with an address register that can count the address only.
- The register that holds the address for the stack is called a stack pointer (SP). It always points at the top item in the stack.
- The two operations that are performed on stack are the insertion and deletion.
- The operation of insertion is called *PUSH.*
- The operation of deletion is called *POP.*
- These operations are simulated by incrementing and decrementing the stack pointer register (SP).

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.
- The below figure shows the organization of a 64-word register stack.

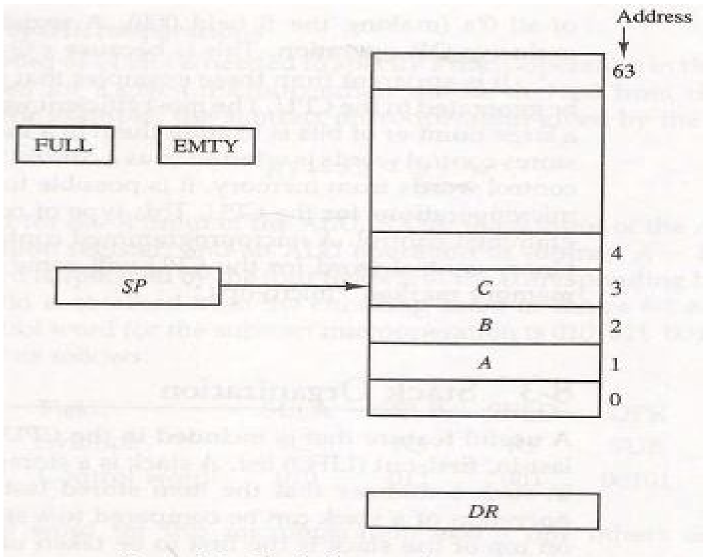


Figure 8-3 Block diagram of a 64-word stack.

- The stack pointer register SP contains a binary number whose value is equal to the address of the word is currently on top of the stack. Three items are placed in the stack: A, *B, C, in* that order.
- In above figure C is on top of the stack so that the content of *SP* is 3.
- For removing the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of stack SP.
- Now the top of the stack is B, so that the content of SP is 2.
- Similarly for inserting the 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.
- Then the one-bit register FULL is set to 1, when the stack is full.
- Similarly when 000000 is decremented by 1, the result is 111111, and then the one-bit register EMTY is set 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 microoperations:
- The stack pointer is incremented so that it points to the address of next-higher word.
- A memory write operation inserts the word from DR the top of the stack.

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

- 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 to 1.
- This condition is reached if the top item prior to the last push way 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, so the 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 min operations:

$DR \leftarrow M[SP]$	Read item from the top of stack
$SP \leftarrow SP - 1$	Decrement stack pointer
If $(SP = 0)$ then $(EMTY \leftarrow 1)$	Check if stack is empty
$FULL \leftarrow 0$	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 1.
- This condition is reached if the item read was in location 1. Once this it is read out, SP is decremented and reaches the value 0, which is the initial value of SP.
- If a pop operation reads the item from location 0 and then is decremented, *SP* changes to 111111, which is equivalent to decimal 63 in above configuration, the word in address 0 receives the last item in the stack.

Memory Stack:

- In the above discussion a stack can exist as a stand-alone unit. But in the CPU implementation of a stack is done by assigning a portion of memory to a stack operation and using a processor register as stack pointer.
- The below figure shows a portion computer memory partitioned into three segments: program, data, and stack.

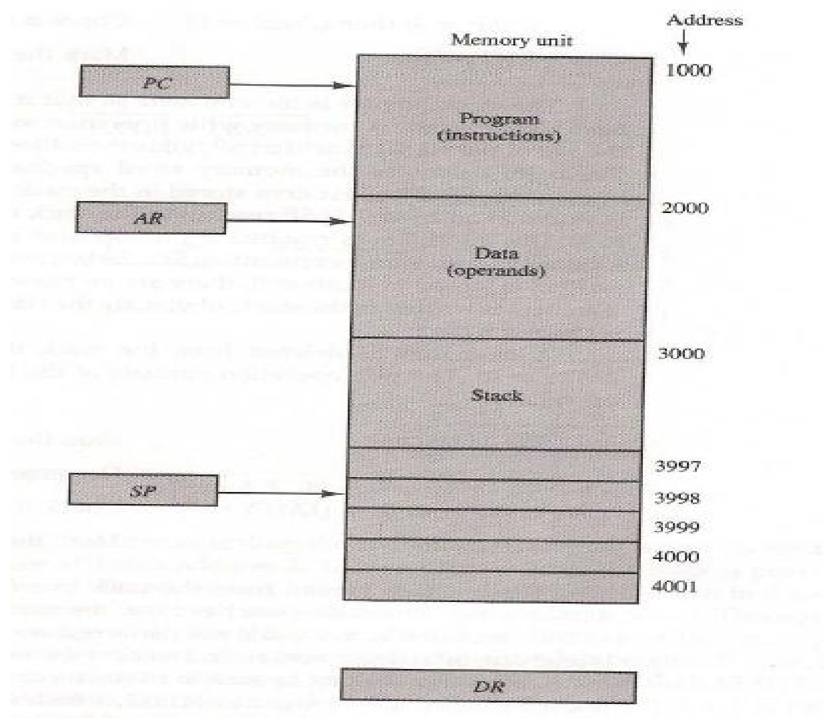


Figure 8-4 Computer memory with program, data, and stack segments.

- The program counter *PC* points at the address of the next instruction in 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 exec phase to read an operand.
    - *SP* is used to push or pop items into or from stack.
- As shown in Fig. 8-4, 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.
- The items in the stack communicate with a data register *DR.* A new item is inserted with the push operation as follows:

  SP← SP-1

  M [SP] ← DR

- The stack pointer is decremented so that it points at the address of the next word.
- A memory write operation inserts the word from DR into the top of stack. A new item is deleted with a pop operation as follows:

  DR ← M [SP]
  SP← SP+1
- The top item is read from the stack into DR. The stack pointer is then decremented to point at the next item in the stack.
- 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 processor registers:
    - one to hold the upper limit (3000 in this case)
    - Other to hold the lower limit (4001 in this case).
- After a push operation, *SP* compared with the upper-limit register and after a pop operation, *SP* is a compared with the lower-limit register.
- The two microoperations needed for either the push or pop are

  (1) An access to memory through SP

  (2) Updating SP.

- The advantage of a memory stack is that the CPU can refer to it without having specify an address, since the address is always available and automatically updated in the stack pointer.

Reverse Polish Notation:

- A stack organization is very effective for evaluating arithmetic expressions.
- The common arithmetic expressions are written in *infix notation,* with each operator written *between* the operands.
- Consider the simple arithmetic expression.

  A*B+C*D
- For evaluating the above expression it is necessary to compute the product A*B, store this product result while computing C*D, and then sum the two products.
- For doing this type of infix notation, it is necessary to scan back and forth along the expression to determine the next operation to be performed.
- The Polish mathematician Lukasiewicz showed that arithmetic expression can be represented in *prefix notation.*
- This representation, often referred to as *Polish notation,* places the operator before the operands. So it is also called as *prefix notation*.
- The *Postfix notation,* referred to as *reverse Polish notation (RPN),* places the operator after the operands.
- The following examples demonstrate the three representations

  Eg: A+B   > Infix notation

+AB > Prefix or Polish notation

AB+ > Post 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* CD* +

  And it is evaluated as follows

  - ✓ Scan the expression from left to right.
  - ✓ When operator is reached, perform the operation with the two operands found on the left side of the operator.
  - ✓ Remove the two operands and the operator and replace them by the number obtained from the result of the operation.
  - ✓ Continue to scan the expression and repeat the procedure for every operation encountered until there are no more operators.
- For the expression above it find the operator * after A and B. So it perform the operation A*B and replace A, B and * with the result.
- The next operator is a * and it previous two operands are C and D, so it perform the operation C*D and places the result in places C, D and *.
- The next operator is + and the two operands to be added are the two products, so we add the two quantities to obtain the result.
- The conversion from infix notation to reverse Polish notation must take into consideration the operational hierarchy adopted for infix notation.
- This hierarchy dictates that we first perform all arithmetic inside inner parentheses, then inside outer parentheses, and do multiplication and division operations before addition and subtraction operations.

Evaluation of Arithmetic Expressions:

- Reverse Polish notation, combined with a stack arrangement of registers, is the most efficient way known for evaluating arithmetic expressions.
- This procedure is employed in some electronic calculators and also in some computer.
- The following numerical example may clarify this procedure. Consider the arithmetic expression

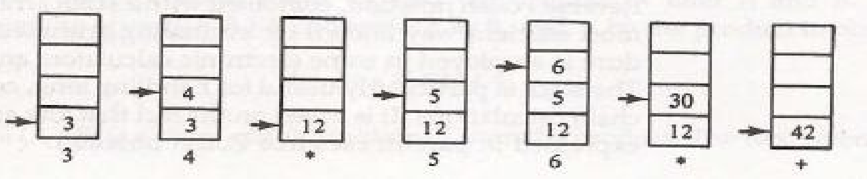
  (3*4) + (5*6)

  In reverse polish notation, it is expressed as

  34 * 56* +

- Now consider the stack operations shown in Fig. 8-5.

Figure 8-5 Stack operations to evaluate 3 • 4 + 5 • 6.



- Each box represents one stack operation and the arrow always points to the top of the stack.
- Scanning the expression from left to right, we encounter two operands.
- First the number 3 is pushed into the stack, then the number 4.
- The next symbol is the multiplication operator *.
- This causes a multiplication of the two top most items the stack.
- The stack is then popped and the product is placed on top of the stack, replacing the two original operands.
- Next we encounter the two operands 5 and 6, so they are pushed into the stack.
- The stack operation results from the next * replaces these two numbers by their product.
- The last operation causes an arithmetic addition of the two topmost numbers in the stack to produce the final result of 42.