Stack Organization

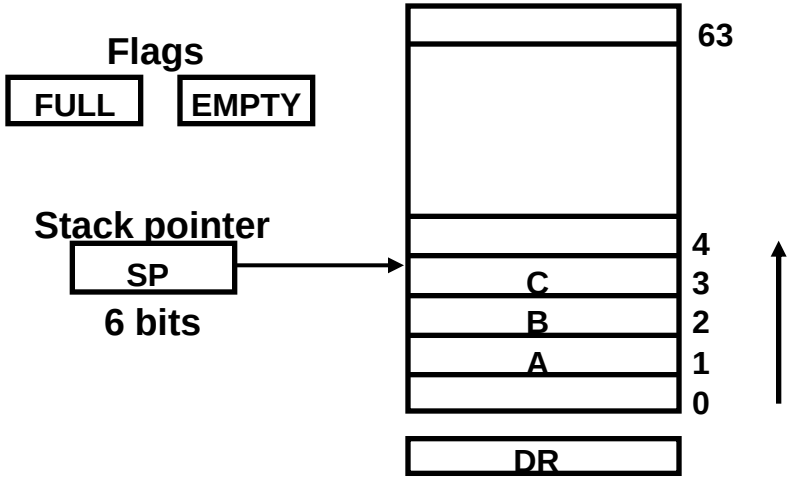
Stack

- Very useful feature for nested subroutines, nested interrupt services
- Also efficient for arithmetic expression evaluation
- Storage which can be accessed in LIFO
- Pointer: SP
- Only PUSH and POP operations are applicable

Stack Organization

- Register Stack Organization
- Memory Stack Organization

Register Stack Organization



Push, Pop operations

/* Initially, SP = 0, EMPTY = 1, FULL = 0 */

PUSH

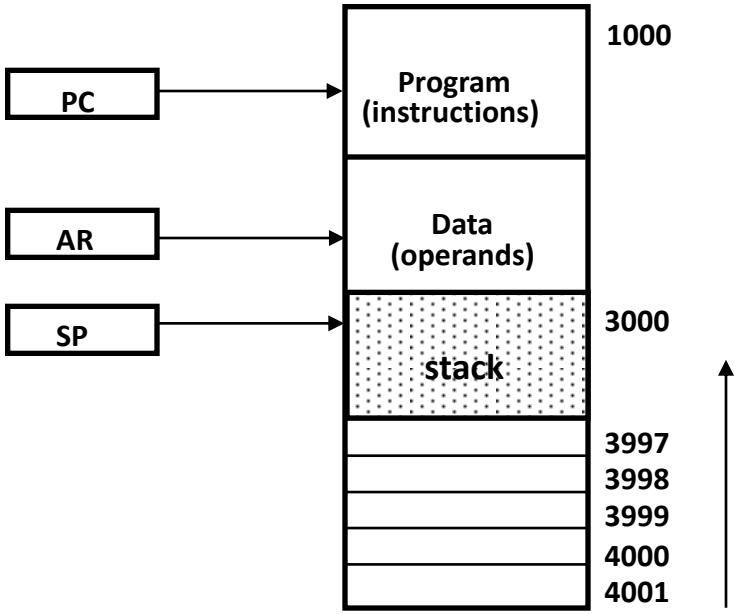
SP ← SP + 1
M[SP] ← DR
If (SP = 0) then (FULL ← 1)
EMPTY ← 0

POP

DR ← M[SP]
SP ← SP – 1
If (SP = 0) then (EMPTY ← 1)
FULL ← 0

Memory Stack Organization

Memory with Program, Data, and Stack Segments



- A portion of memory is used as a stack with a processor register as a stack pointer

- PUSH: SP ← SP - 1
  M[SP] ← DR
- POP: DR ← M[SP]
  SP ← SP + 1

- Most computers do not provide hardware to check stack overflow (full stack) or underflow (empty stack) → must be done in software

Reverse Polish Notation

Stack is very effective in evaluating arithmetic expressions

- **Arithmetic Expressions:**

A * B + C * D

Polish Notation (Prefix) : Place operator before operand

Reverse Polish Notation (Postfix) : Place operator after operand

AB*CD*+

1. (A*B)CD*+
2. (A*B) (C*D) +
3. (A*B) + (C*D)

(A+B) * [C* (D+E)+ F] → AB+DE+C*F+*

Reverse Polish Notation also called as …….

a) Prefix Notation
b) Postfix Notation
c) Hybrid Notation
d) None of the above

Reverse Polish Notation

- **Arithmetic Expressions: A + B**

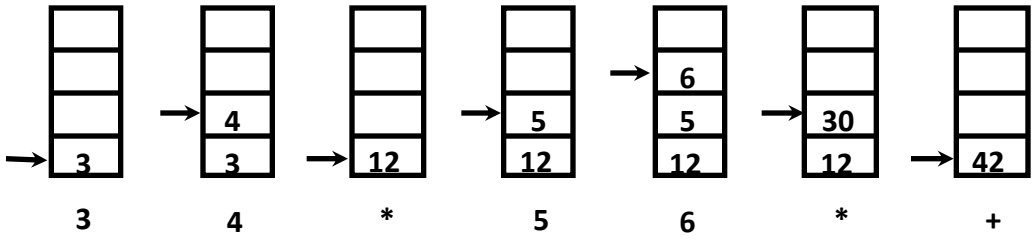
A + B	**Infix notation**
+ A B	**Prefix or Polish notation**
A B +	**Postfix or reverse Polish notation**

- The reverse Polish notation is very suitable for stack manipulation

- **Evaluation of Arithmetic Expressions**

Any arithmetic expression can be expressed in parenthesis-free Polish notation, including reverse Polish notation

(3 * 4) + (5 * 6) ⇒ 3 4 * 5 6 * +



Instruction Format

- **Instruction Fields**

  OP-code field - specifies the operation to be performed

  Address field - designates memory address(es) or a processor register(s)

  Mode field - determines how the address field is to be interpreted (to get effective address or the operand)

- **The number of address fields in the instruction format depends on the internal organization of CPU**
- **The three most common CPU organizations:**

Single accumulator organization:

```
ADD     X              /* AC ← AC + M[X]  */
```

General register organization:

```
ADD     R1, R2, R3        /* R1 ← R2 + R3  */
ADD     R1, R2         /* R1 ← R1 + R2  */
MOV     R1, R2         /* R1 ← R2  */
ADD     R1, X          /* R1 ← R1 + M[X]  */
```

Stack organization:

```
PUSH    X              /* TOS ← M[X]  */
ADD
```

Three & Two Address Instruction

- **Three-Address Instructions**

  Program to evaluate X = (A + B) * (C + D) :

```
ADD   R1, A, B   /*  R1 ← M[A] + M[B] */
ADD   R2, C, D   /*  R2 ← M[C] + M[D] */
MUL   X, R1, R2          /*  M[X] ← R1 * R2          */
```

  - Results in short programs
  - Instruction becomes long (many bits)

- **Two-Address Instructions**

  Program to evaluate X = (A + B) * (C + D) :

```
MOV   R1, A      /* R1 ← M[A]      */
ADD   R1, B      /* R1 ← R1 + M[A] */
MOV   R2, C      /* R2 ← M[C]      */
ADD   R2, D      /* R2 ← R2 + M[D] */
MUL   R1, R2     /* R1 ← R1 * R2   */
MOV   X, R1      /* M[X] ← R1      */
```

  -most common in commercial computer

One Address Instruction

- **One-Address Instructions**
  - **Use an implied AC register for all data manipulation**
  - **Program to evaluate X = (A + B) * (C + D) :**

```
LOAD     A     /*  AC ← M[A]     */
ADD      B     /*  AC ← AC + M[B]  */
STORE    T     /*  M[T] ← AC     */
LOAD     C     /*  AC ← M[C]     */
ADD      D     /*  AC ← AC + M[D]     */
MUL      T     /*  AC ← AC * M[T]     */
STORE    X     /*  M[X] ← AC     */
```

Zero Address Instruction

- **Zero-Address Instructions**

  - Can be found in a stack-organized computer

  - Program to evaluate X = (A + B) * (C + D) :

```
PUSH    A       /*  TOS ← A          */
PUSH    B       /*  TOS ← B          */
ADD             /*  TOS ← (A + B)    */
PUSH    C       /*  TOS ← C          */
PUSH    D       /*  TOS ← D          */
ADD             /*  TOS ← (C + D)    */
MUL             /*  TOS ← (C + D) * (A + B)  */
POP     X       /*  M[X] ← TOS       */
```

In case of Zero address instruction, which instruction is used to place the data at the top of stack?

a) Pop
b) Push
c) Insert
d) None of the above