# Chap 4
# Additional Notes

Operand 0, RPN and ASM prog (*push and pop*)

# (d) No-address instructions

$$Y = \frac{A - B}{C + (D \times E)}$$

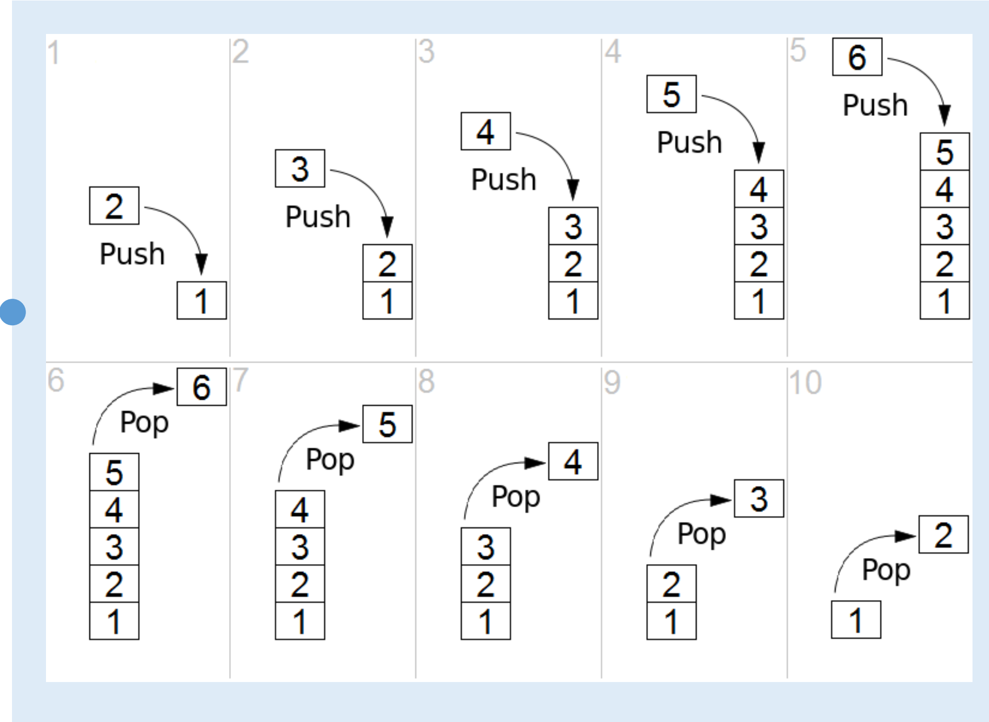
```
PUSH   C
PUSH   D
PUSH   E
MUL
ADD
PUSH   B
PUSH   A
SUB
DIV
POP    Y
```

10 instructions

- 0 (zero) address
- All addresses implicit.
- Usually use a *stack* (a push down stack in CPU).
- There are two *Opcodes* with one *operand*: PUSH op, POP op.

## (d) No-address instructions

### Stack Machine




- A *stack* is an abstract data type and data structure based on the principle of *Last In First Out* (LIFO).
- *Stack machine*: Java Virtual Machine.
- *Call stack* of a program, also known as a function stack, execution stack, control stack, or simply the stack.
- Application: *Reverse Polish Notation (RPN)*, *Depth-First-Search (DFS)*

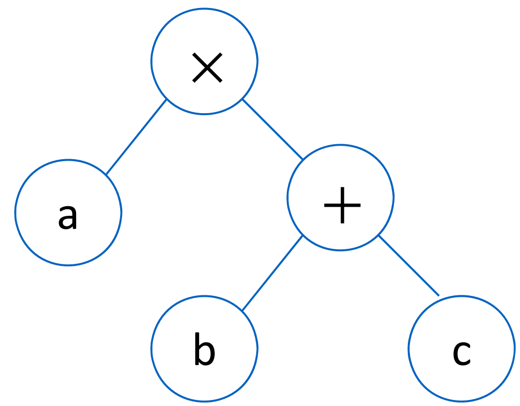
*RPN will be discussed in next example*



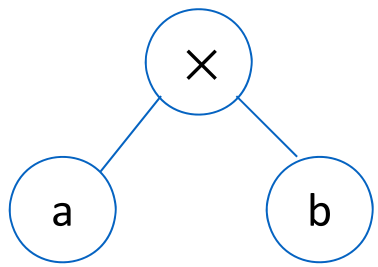
https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

# Expression tree

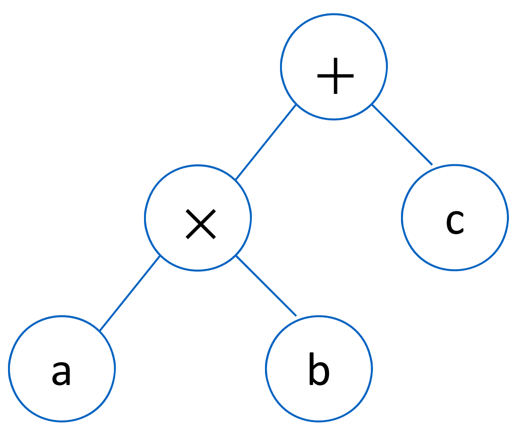
$$a \times (b + c)$$



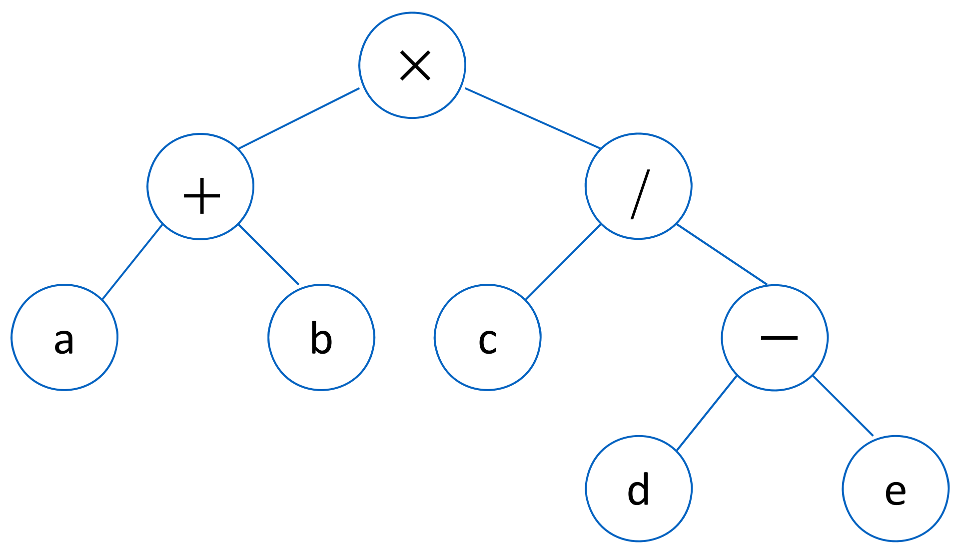

$$a \times b$$




$$a \times b + c$$



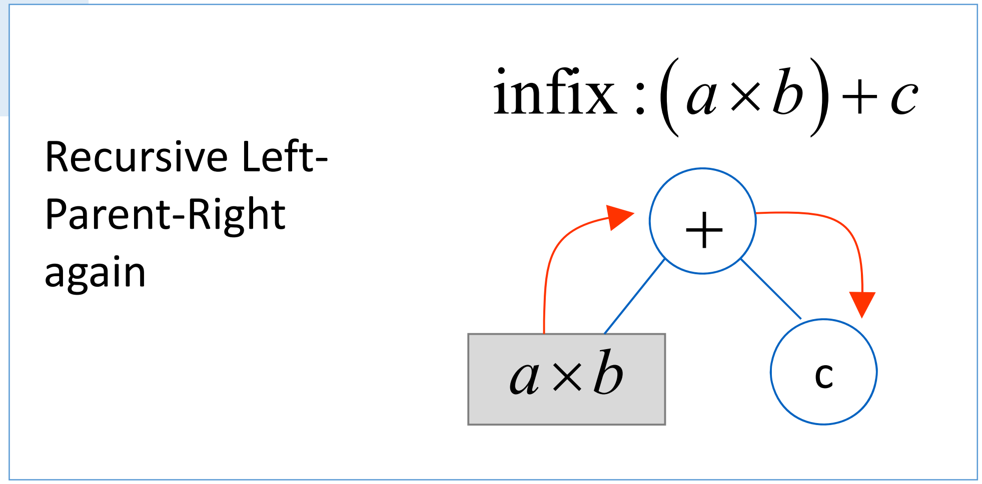

$$(a + b) \times (c / (d - e))$$

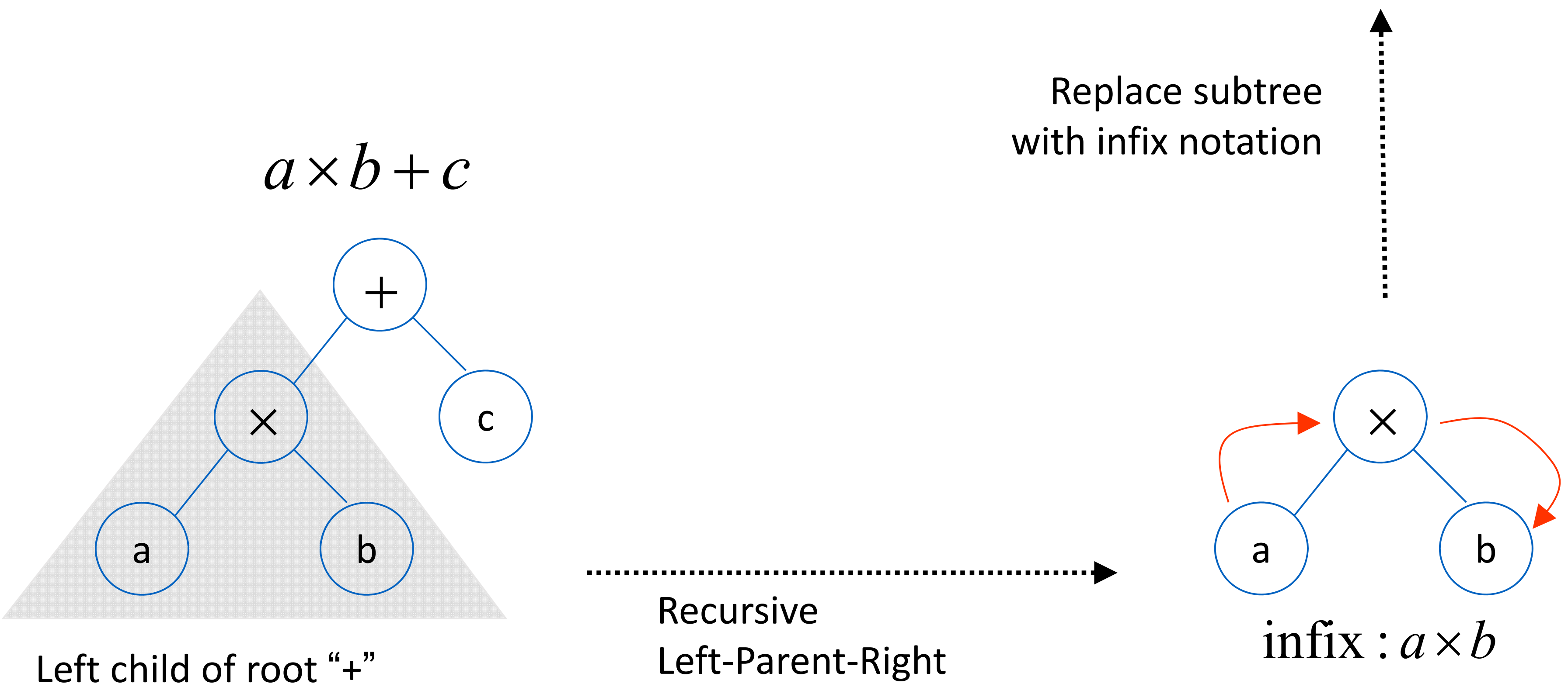


# Expression tree:

**(a) Infix notation**:

Left-Parent-Right order

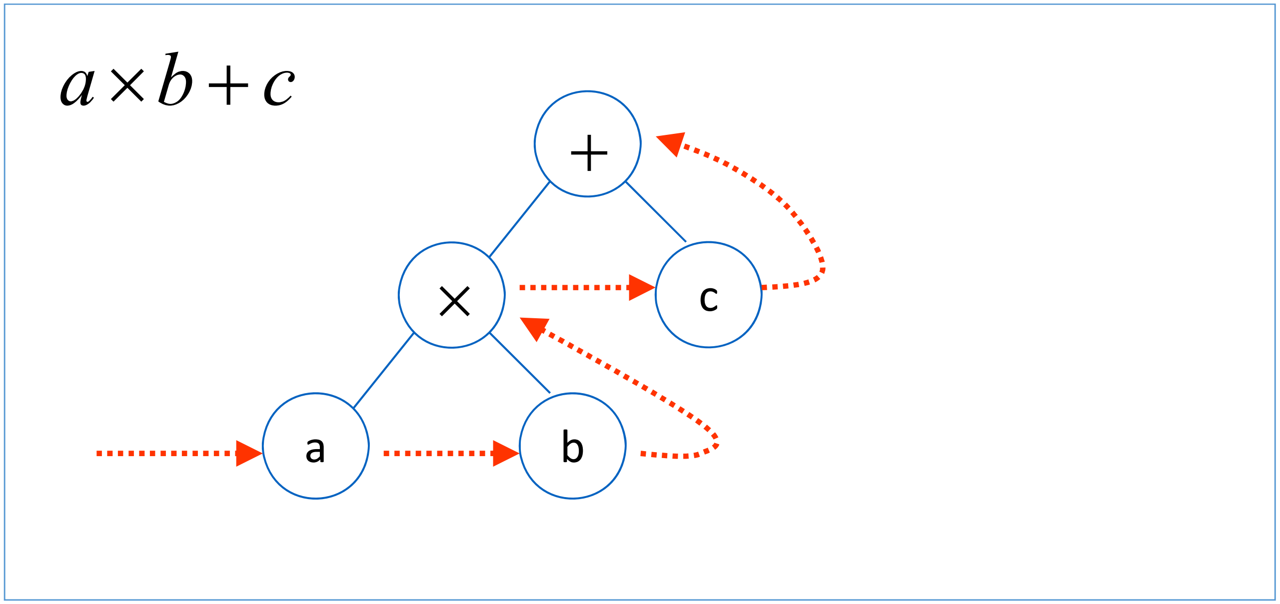


# Expression tree:

**(b) Postfix notation**:
Left-Right-Parent order

$$postfix: ab\times c+$$



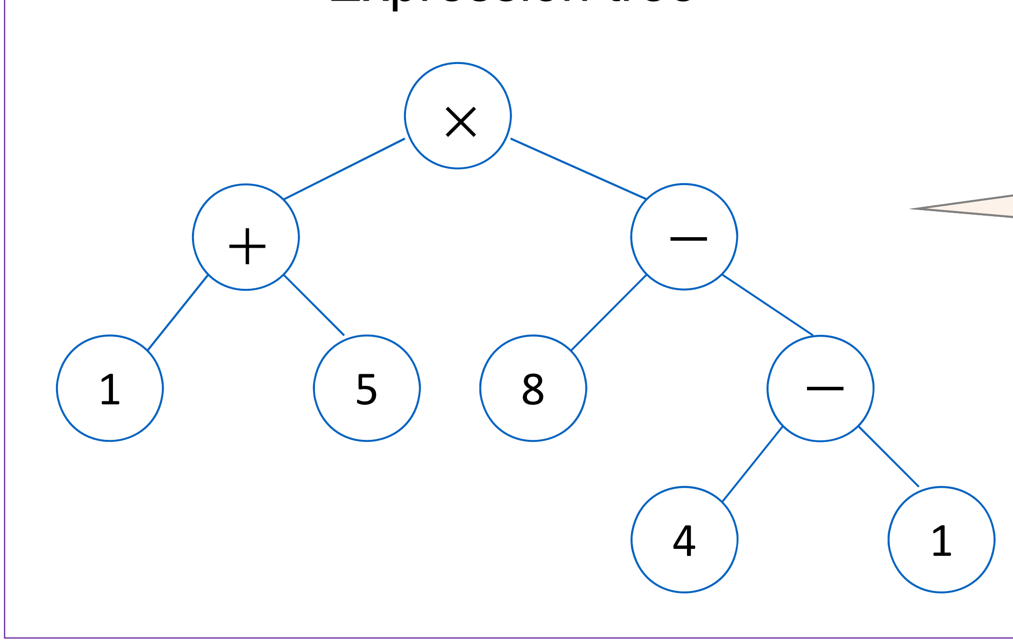
## Reverse Polish Notation (RPN)

- Precedence of multiplication is higher than addition, we need parenthesis to guarantee execution order.
- However in the early 1950s, the Polish logician Jan Lukasiewicz observed that <u>parentheses are not necessary in postfix notation</u>, called *RPN* (*Reverse Polish Notation*).
- The Reverse Polish scheme was proposed by F.L. Bauer and E.W. Dijkstra in the early 1960s to reduce computer memory access and utilize the *stack* to evaluate expressions .

## Example: Reverse Polish Notation (RPN) → *Postfix order*

$$Infix: \quad (1+5)\times(8-(4-1))$$

Expression tree



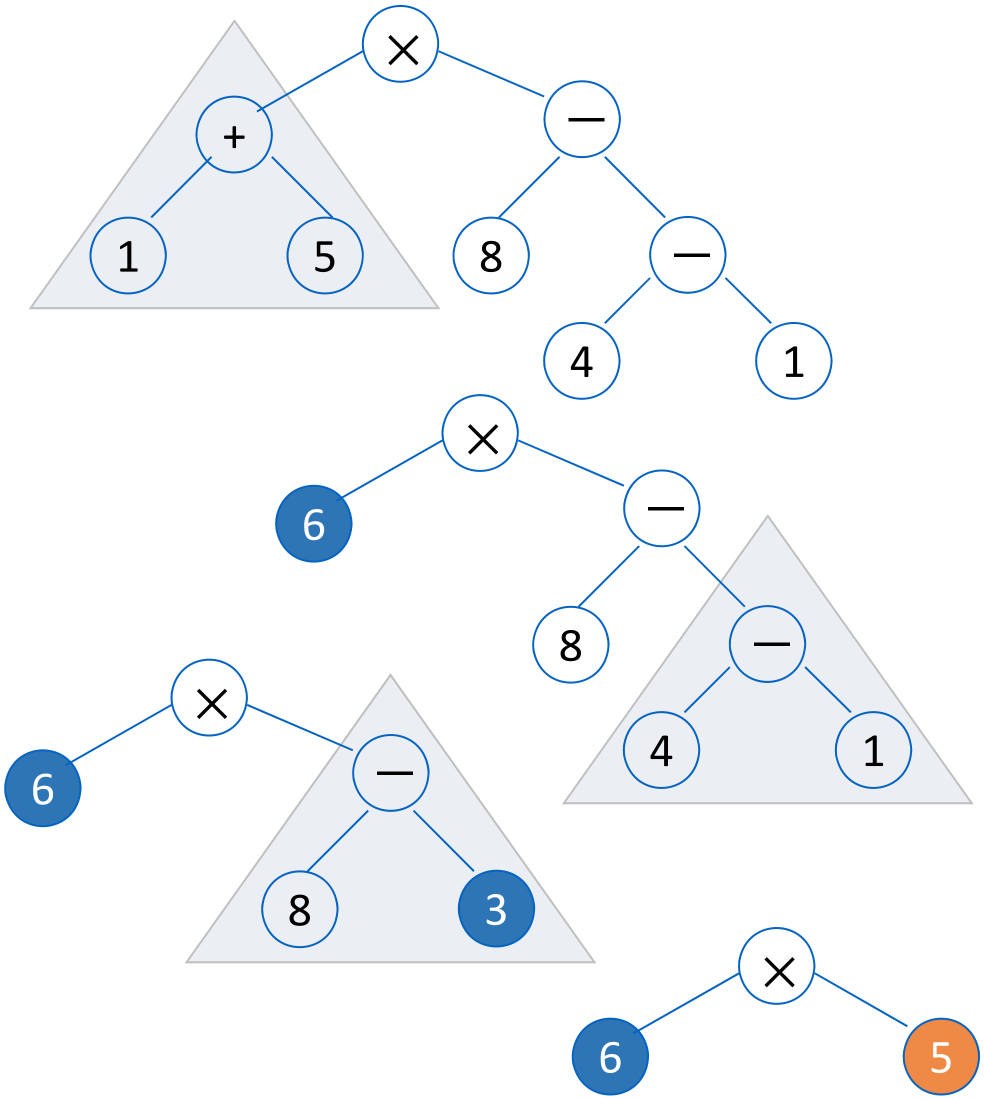

*Postfix:*

$$1\,5 + 8\,4\,1 - - \times$$

*(parenthesis free)*

## Example: Reverse Polish Notation (RPN) → *Postfix order* [1]

$$Infix: \quad (1+5)\times(8-(4-1))$$




$$Postfix: \quad \underline{1\,5\,+}\,8\,4\,1\,-\,-\,\times$$

$$1+5=6$$

$$6\,8\,\underline{4\,1\,-}\,-\,\times$$

$$4-1=3$$

$$6\,\underline{8\,3\,-}\,\times$$

$$8-3=5$$

$$\underline{6\,5\,\times}$$

$$6\times5=30$$

$$30$$

## Example: Evaluate RPN expression [1]

Expression

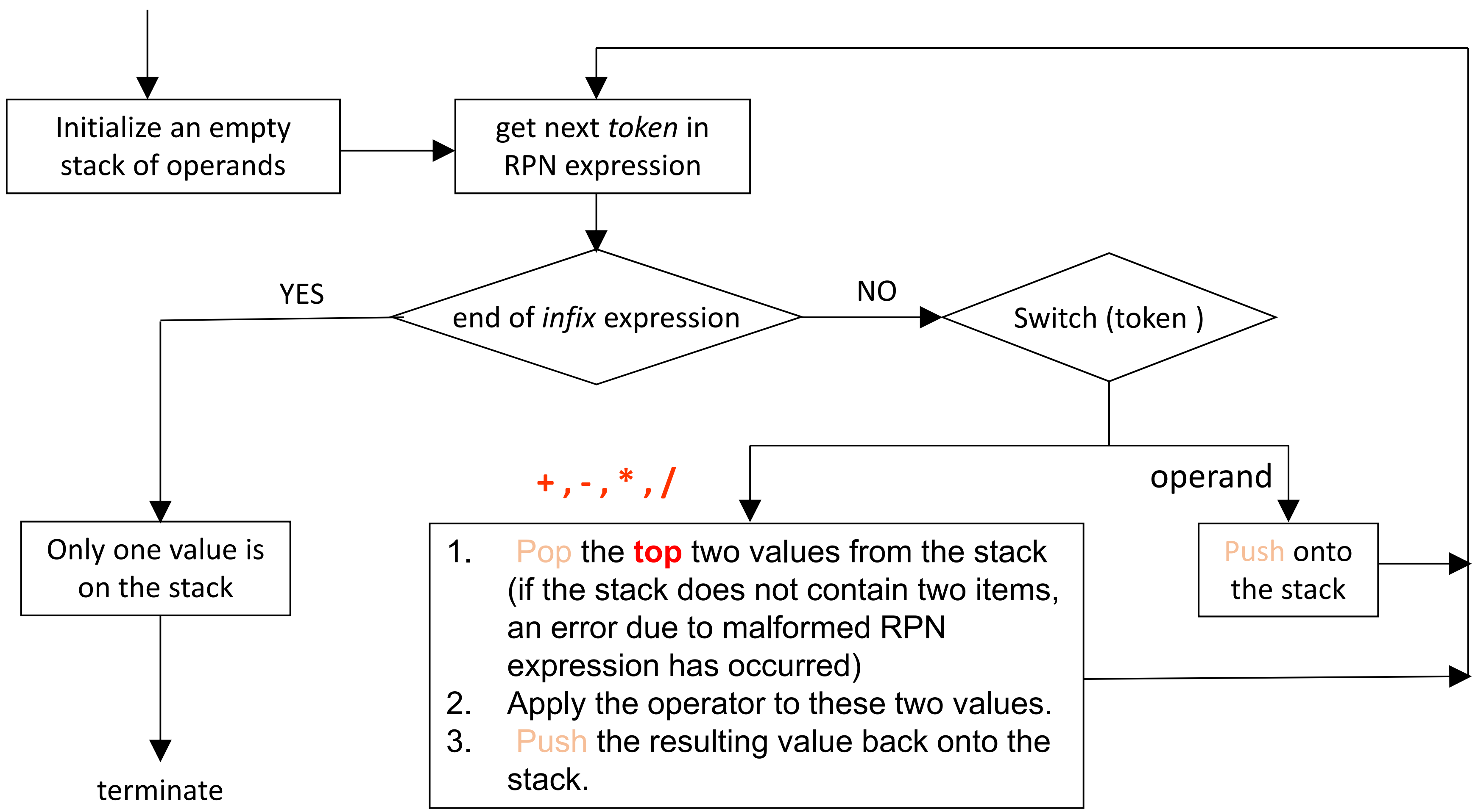
30

*Infix* : $(1+5)\times(8-(4-1))$

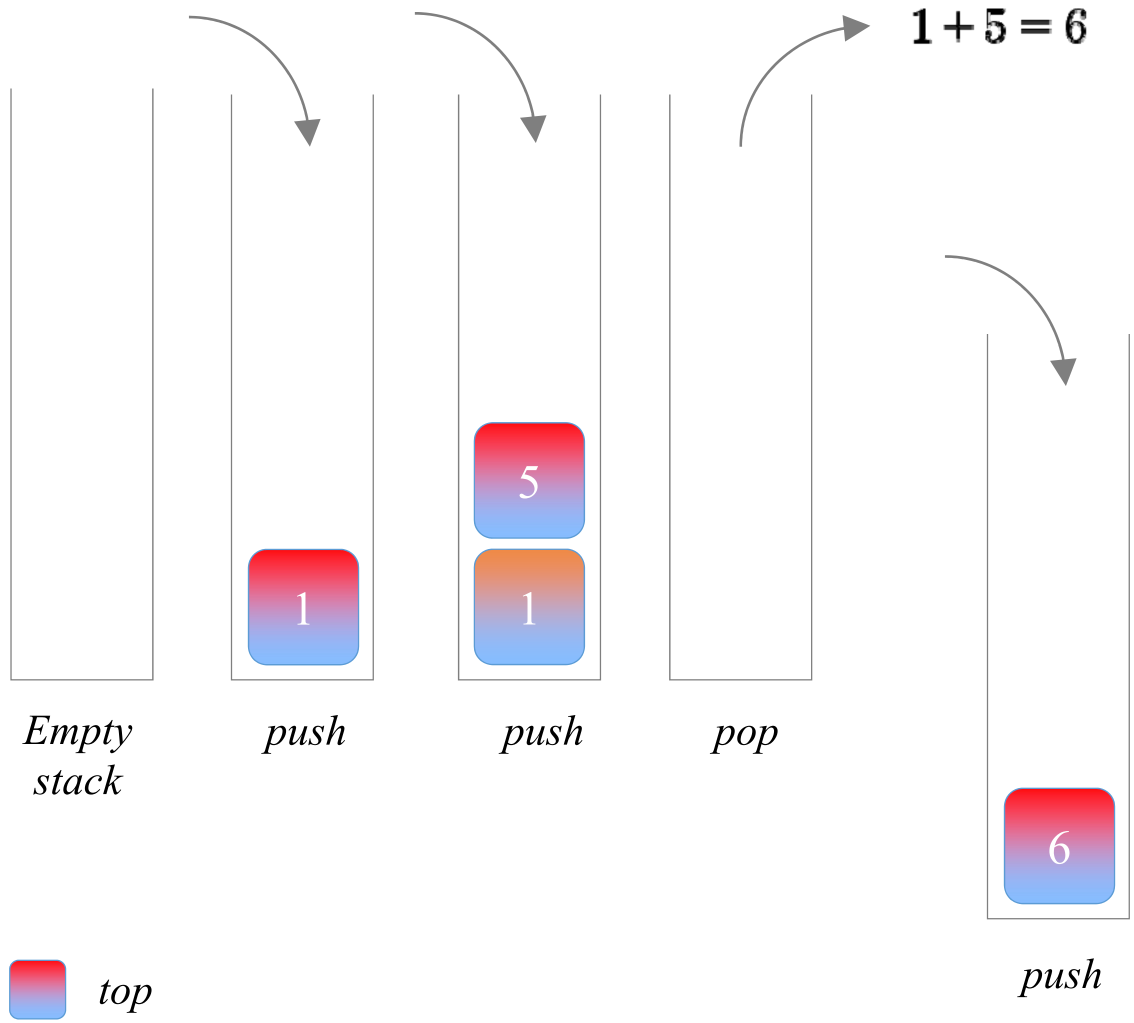
*Postfix* : $1\,5+8\,4\,1--\times$

- **Scanned** from left to right until an operator is found, then the last two operands must be retrieved and combined.
- Order of operands satisfy LIFO, so we can use **stack** to store operands and then evaluate RPN expression.

## Example: Evaluate RPN expression → *Flow Chart*



## Example: Evaluate RPN expression [2]




$$1\ 5 + 8\ 4\ 1 - - \times$$

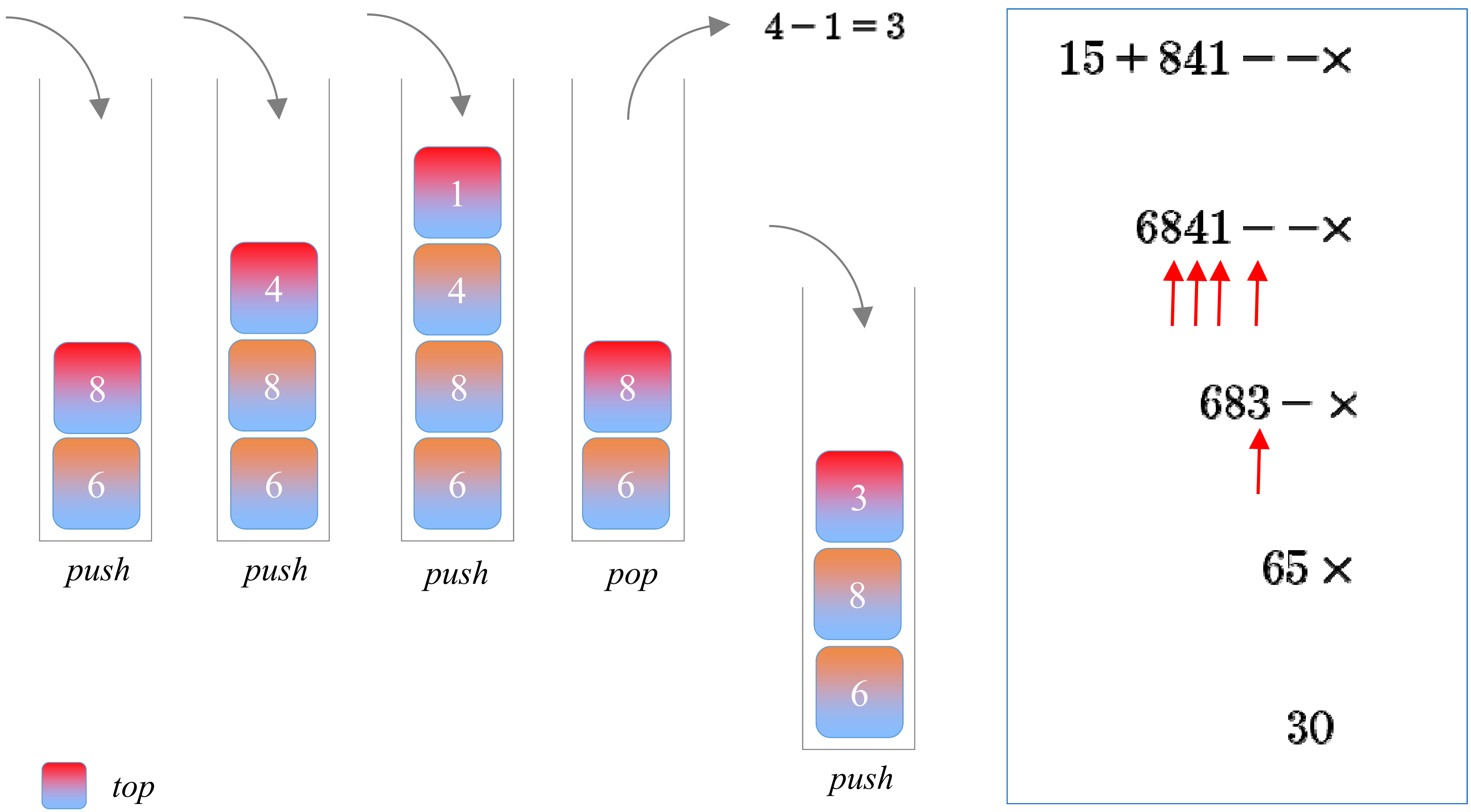
$$6\ 8\ 4\ 1 - - \times$$

$$6\ 8\ 3 - \times$$

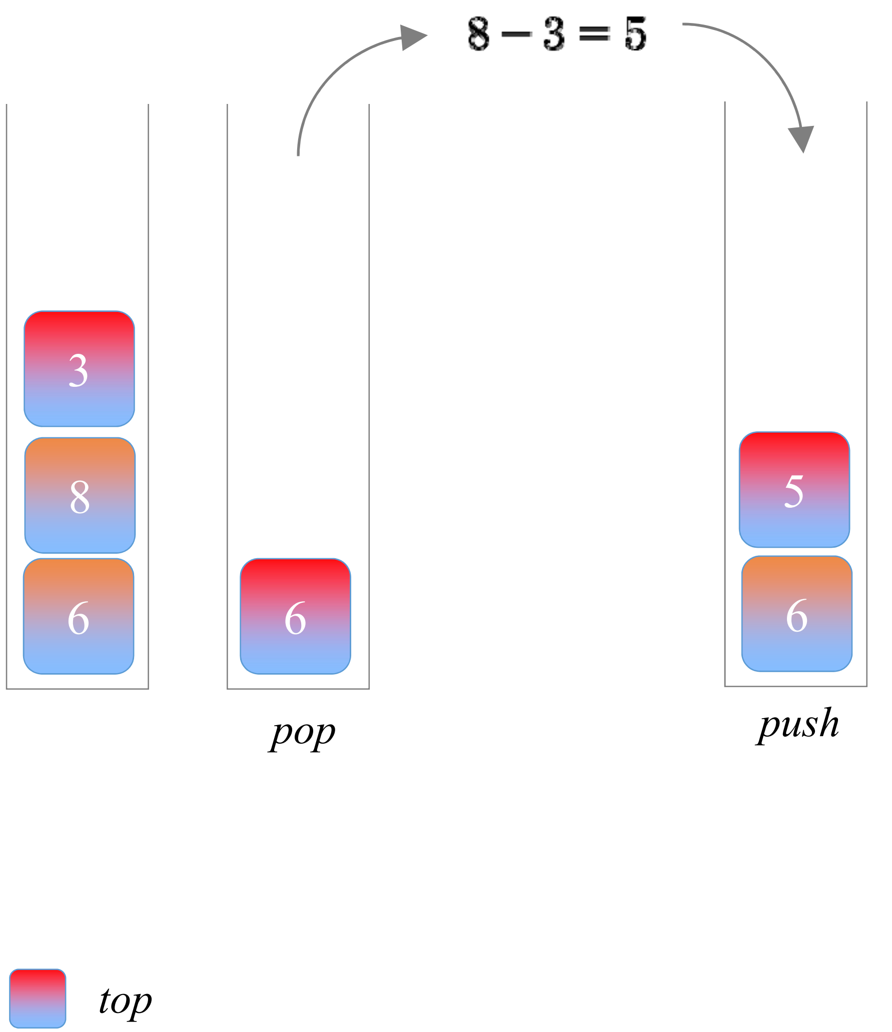
$$6\ 5 \times$$

$$30$$

## Example: Evaluate RPN expression [2]



## Example: Evaluate RPN expression [2]




$$15 + 841 - - \times$$

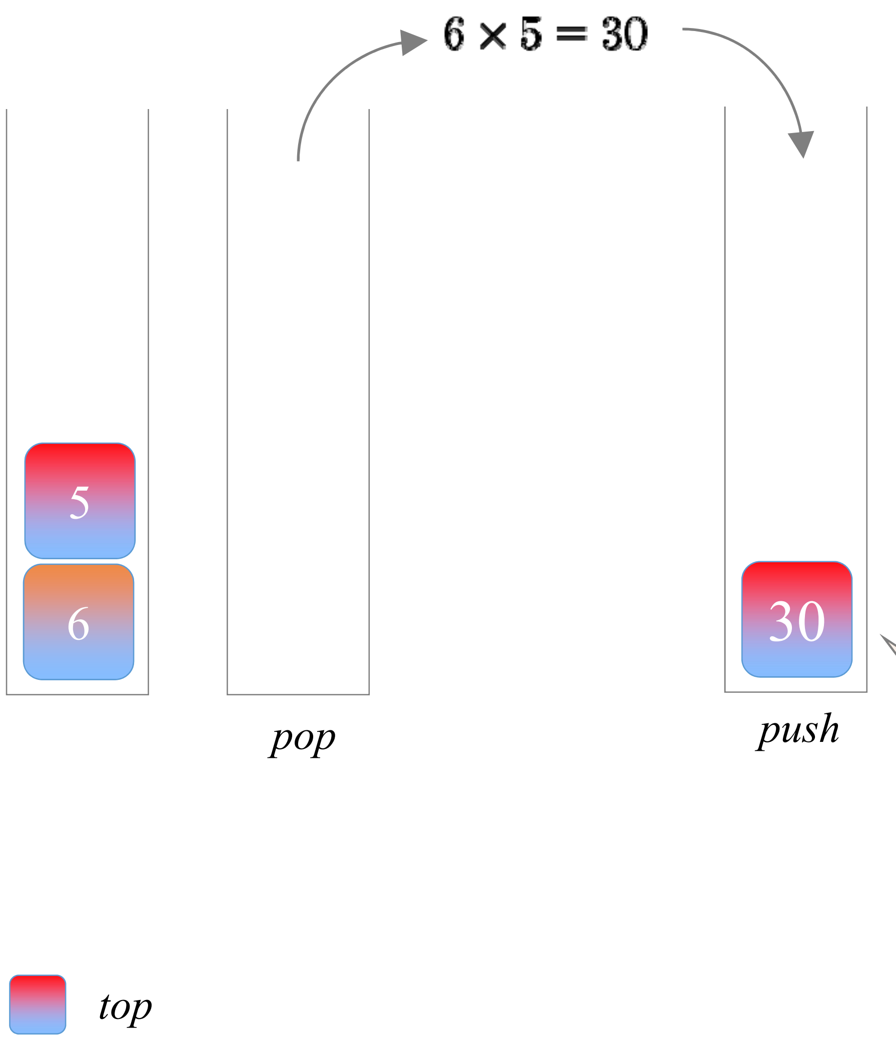
$$6841 - - \times$$

$$683 - \times$$

$$65 \times$$

$$30$$

## Example: Evaluate RPN expression [2]




$$15 + 841 - - \times$$

$$6841 - - \times$$

$$683 - \times$$

$$65 \times$$

$$30$$

*Only one value on the stack, and this is the final result.*

## Exercise 4.1a:

Given an expression as

$$A + B * C - (D/E + F) * G$$

(a) Construct the expression tree.

(b) Convert into PRN postfix evaluation.

## Exercise 4.1b:

Given an expression as

$$(A+B)*((C-D)/(E+F))*G$$

(a) Construct the expression tree.

(b) Convert into PRN postfix evaluation.

## Exercise 4.2:

Get the infix expression for the following postfix:

(a) $AB + C -$

(b) $AB + CD - *$

(c) $AB \wedge C * D - EF / GH + / +$

(d) $AB + C * DE - - FG + \wedge$

(e) $ABCDE \wedge * / -$

# Example:
# RPN and zero operand

Use x86 family

# Post fix : RPN

Question:

Express 15 - (3*2)/2 in RPN and write asm code use push and pop (zero operand)

Answer:

To express the expression $15-(3\times2)/2$ in Reverse Polish Notation (RPN), it would be: $15\ 3\ 2 * 2\ /\ -$

# Convert to ASM code

- use instruction with zero operand:
  - *push* (push the values onto the stack)
  - Pop (Pop the value from stack into example register (eax))

- Now, let's convert this RPN expression into assembly code using only push and pop (zero-operand instructions):

- Post fix (RPN), it would be: $15\ 3\ 2 * 2 / -$

Post fix (RPN), it would be: 15 3 2 ∗ 2 / −

Evaluate value in RPN

15 3 2 ∗ 2 / −
15 6 2 / −
15 6 2 / −
15 3 −
15 3 −
12

Answer = 12

```
section .text
   global _start

_start:

   ; Push the values onto the stack
   push dword 15   ; Push 15
   push dword 3    ; Push 3
   push dword 2    ; Push 2

   ; Multiply 3 * 2
   pop eax         ; Pop 2 into eax
   pop ebx         ; Pop 3 into ebx
   imul ebx, eax   ; ebx = 3 * 2

   ; Push the result of multiplication onto the
stack
   push ebx        ; Push (3 * 2) onto the stack

   ; Push 2 onto the stack
   push dword 2    ; Push 2

   ; Divide the result by 2
   pop ebx         ; Pop 2 into ebx
   pop eax         ; Pop (3 * 2) into eax
   idiv ebx        ; eax = (3 * 2) / 2

   ; Push the result of division onto the stack
   push eax        ; Push ((3 * 2) / 2) onto the stack

   ; Push 15 onto the stack
   push dword 15   ; Push 15

   ; Perform subtraction
   pop eax         ; Pop 15 into eax
   pop ebx         ; Pop ((3 * 2) / 2) into ebx
   sub eax, ebx    ; eax = 15 - ((3 * 2) / 2)

   ; eax now contains the final result
```

This code:

- Pushes 15, 3, and 2 onto the stack.
- Multiplies 3 and 2, then divides the result by 2.
- Performs subtraction of 15 and the result of the division.

# Other example

- 2*3 + (4*5)/(2+3) express in RPN and write asm code use push and pop

To express the expression (2*3) + (4*5) / (2+3) in Reverse Polish Notation (RPN),

it would be:
2 3 ∗ 4 5 ∗ 2 3 + / +

evaluate

2 3 ∗ 4 5 ∗ 2 3 + / +
6 4 5 ∗ 2 3 + / +
6 4 5 ∗ 2 3 + / +
6 20 2 3 + / +
6 20 2 3 + / +
6 20 5 / +
6 20 5 / +
6 4 +
6 4 + Final answer = 10

Now, let's convert this RPN expression into assembly code using only push and pop (zero-operand instructions):

```
section .text
   global _start

_start:
   ; Push the values onto the stack
   push dword 2    ; Push 2
   push dword 3    ; Push 3

   ; Multiply 2 * 3
   pop eax         ; Pop 3 into eax
   pop ebx         ; Pop 2 into ebx
   imul ebx, eax   ; ebx = 2 * 3

   ; Push the result of multiplication onto the stack
   push ebx        ; Push (2 * 3) onto the stack

   ; Push 4 and 5 onto the stack
   push dword 4    ; Push 4
   push dword 5    ; Push 5

   ; Multiply 4 * 5
   pop eax         ; Pop 5 into eax
   pop ebx         ; Pop 4 into ebx
   imul ebx, eax   ; ebx = 4 * 5

   ; Push the result of multiplication onto the stack
   push ebx        ; Push (4 * 5) onto the stack
```

```
; Push 2 and 3 onto the stack
   push dword 2    ; Push 2
   push dword 3    ; Push 3

   ; Add 2 + 3
   pop eax         ; Pop 3 into eax
   pop ebx         ; Pop 2 into ebx
   add ebx, eax    ; ebx = 2 + 3

   ; Push the result of addition onto the stack
   push ebx        ; Push (2 + 3) onto the stack

   ; Divide (4 * 5) / (2 + 3)
   pop eax         ; Pop (2 + 3) into eax
   pop ebx         ; Pop (4 * 5) into ebx
   idiv ebx        ; eax = (4 * 5) / (2 + 3)

   ; Push the result of division onto the stack
   push eax        ; Push ((4 * 5) / (2 + 3)) onto the stack

   ; Perform addition (2 * 3) + ((4 * 5) / (2 + 3))
   pop eax         ; Pop ((4 * 5) / (2 + 3)) into eax
   pop ebx         ; Pop (2 * 3) into ebx
   add eax, ebx    ; eax = (2 * 3) + ((4 * 5) / (2 + 3))

   ; eax now contains the final result
```

This code:

- Pushes 2 and 3 onto the stack, then multiplies them
.
- Pushes 4 and 5 onto the stack, then multiplies them.
- Pushes 2 and 3 onto the stack, then adds them.
- Divides the result of the multiplication by the result of the addition.
- Performs the final addition of the two results.