Linux Assembly

Armine Hayrapetyan

Outline

Math operations are used to mathematically manipulate registers.

The form of a math operation is typically:

operation register, value/register

The first register is the subject of the operation.

For example:

```
add rax, 5
sub rbx, rax
```

Math Operations List

Operation Name	Operation Name (signed)	Description
add a, b	-	a = a+b
sub a, b	-	a = a-b
mul reg	imul reg	rax = rax·reg
div reg	idiv reg	rax = rax/reg
neg reg	-	reg = -reg
inc reg	-	reg = reg + 1
dec reg	-	reg = reg -1
adc a, b	-	a = a+b+CF
sbb a, b	-	a = a-b-CF

ASCII

ASCII is the method which modern computers represent strings of text.

Computers can only store numbers, so ASCII works by mapping numbers to specific characters (letters, numbers, symbols, etc).

ASCII uses 7-bit binary codes to represent characters. However, as the byte, which is 8-bit, is the primary unit of information, modern computers usually support *extended ASCII*, which is 8-bit codes.

It is useful to reference an ASCII table.

Display a Digit

Here is a very simple subroutine for displaying a single digit between 0 and 9.

The digit it displays comes from the *rax* register.

```
section .data
        digit db 0,10

...

_printRAXDigit:

        add rax, 48
        mov [digit], al
        mov rax, 1
        mov rdi, 1
        mov rsi, digit
        mov rdx, 2
        syscall
        ret
```

Display a Digit

The lower byte of the *rax* register is then moved into the the memory address “digit”.

“digit” is actually defined with two bytes, being 0 and 10, a new line character. Since we are only loading the lower byte of the *rax* register into “digit”, it only overwrites the first byte and does not affect the newline character.

```
section .data
        digit db 0,10

...

_printRAXDigit:

        add rax, 48
        mov [digit], al
        mov rax, 1
        mov rdi, 1
        mov rsi, digit
        mov rdx, 2
        syscall
        ret
```

Display a Digit

You can use this subroutine to display a digit between 0-9 by loading that digit into the *rax* register then calling the subroutine.

```
mov rax, 7
call _printRAXDigit
```

This will display 7.

```
section .data
        digit db 0,10

...

_printRAXDigit:

        add rax, 48
        mov [digit], al
        mov rax, 1
        mov rdi, 1
        mov rsi, digit
        mov rdx, 2
        syscall
        ret
```

Testing Math

We can use this to quickly test the math operators.

Code	Output	Reasoning

`mov rax, 6`
`mov rbx, 2`
`div rbx`
`call _printRAXDigit`
	3	rax = 6/2

`mov rax, 1`
`add rax, 4`
`call _printRAXDigit`
	5	rax = 1+4

The Stack

The stack, like registers, is another way to temporarily store data.

It is called the “stack” because you *stack* data onto it.

Imagine a stack of papers. As you stack one paper on top of another, you can only at a given time see the contents of the paper at the top of the stack.

A page within the middle of the stack cannot be removed without removing all the pages on top of it first, as the pages on top of it hold it in place.

Terminology

When you add data onto the top of the stack, you *push* data onto the stack.

When you remove data from the top of the stack, you *pop* data from the stack.

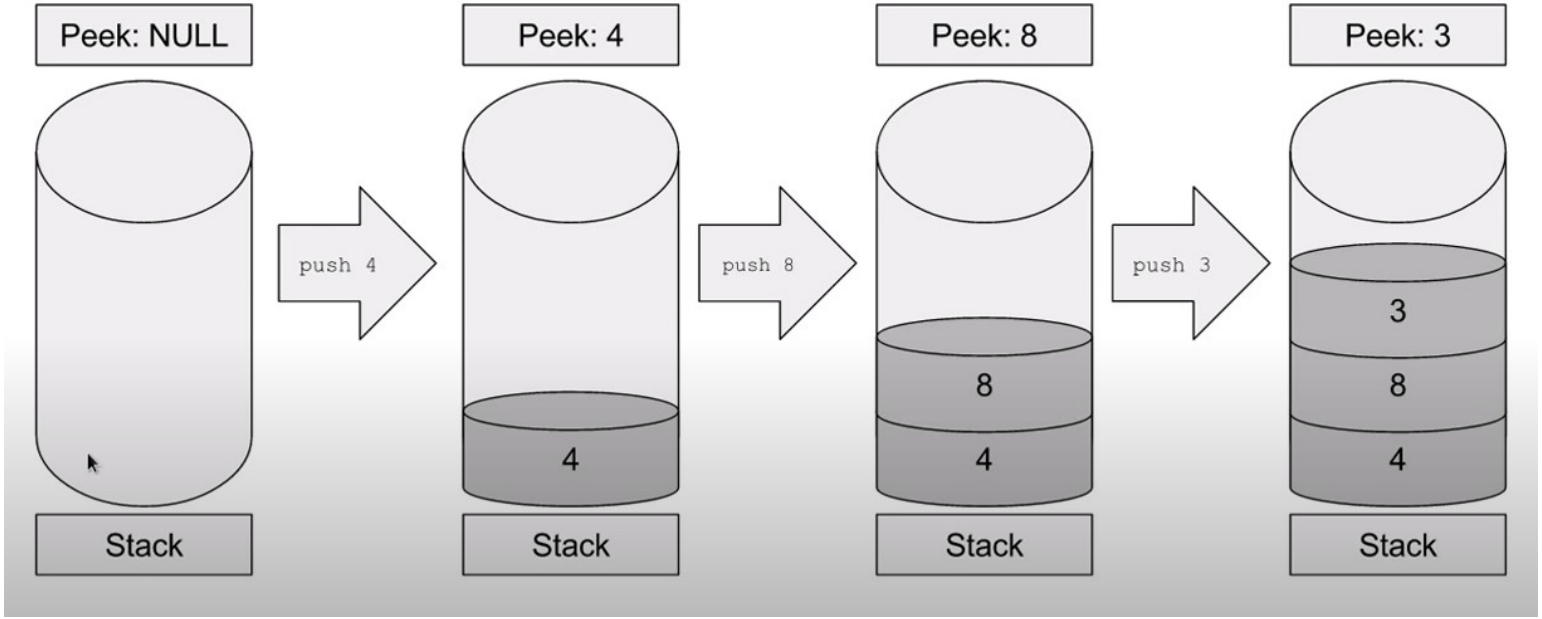
If you look at the top of the stack without removing or adding anything to it, this is called *peeking*.

Stack Operations

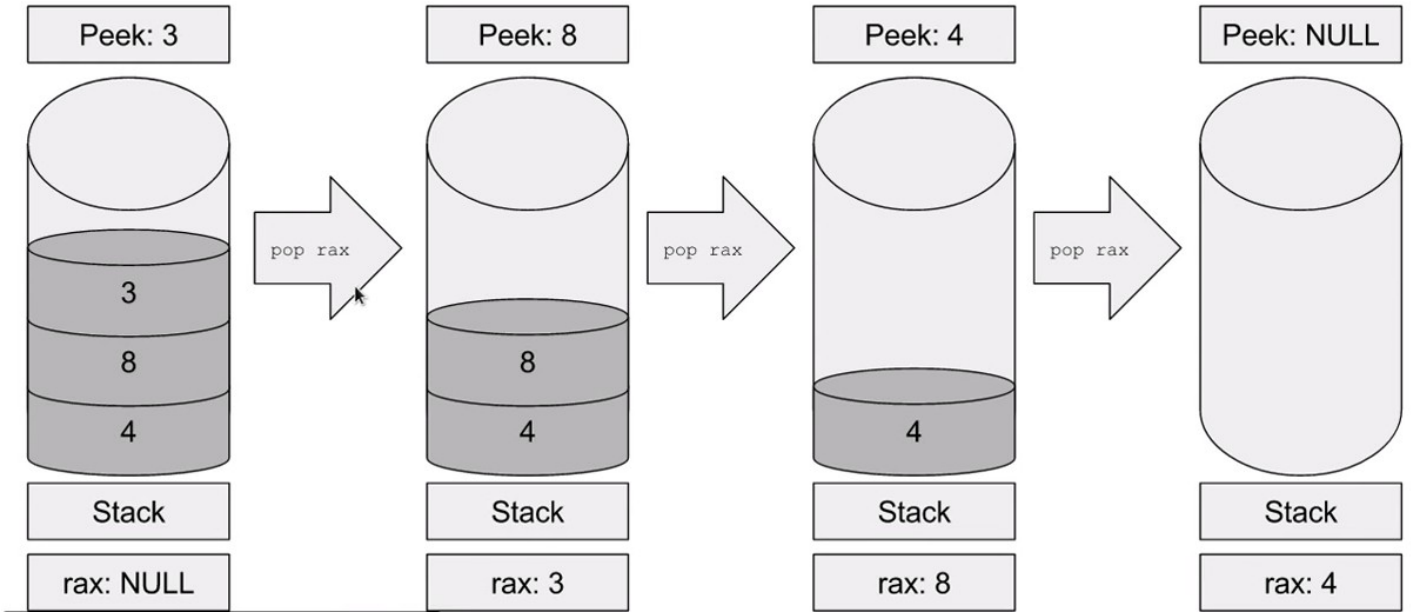
Operation	Effect
`push reg/value`	Pushes a value onto the stack.
`pop reg`	Pops a value off of the stack and stores it in reg.
`mov reg, [rsp]`	Stores the peek value in reg.

Note: Usually in places where you can use registers, you can also use pointers.
Such as, instead of “pop reg”, you can use “pop [label]” to pop a value off the stack directly into a position in memory.

Push


Peek: NULL
Stack
push 4
Peek: 4
4
Stack
push 8
Peek: 8
8
4
Stack
push 3
Peek: 3
3
8
4
Stack

Pop


Peek: 3
3
8
4
Stack
rax: NULL
pop rax
Peek: 8
8
4
Stack
rax: 3
pop rax
Peek: 4
4
Stack
rax: 8
pop rax
Peek: NULL
Stack
rax: 4

Example Code

Code

```
push 4
push 8
push 3

pop rax
call _printRAXDigit
pop rax
call _printRAXDigit
pop rax
call _printRAXDigit
```

Output

```
~$ ./stackexample
3
8
4
```

References

- https://www.youtube.com/watch?v=NFv7l3wQsZ4&list=PLetF-YjXm-sCH6FrTz4AQhfH6INDQvQSn&index=5