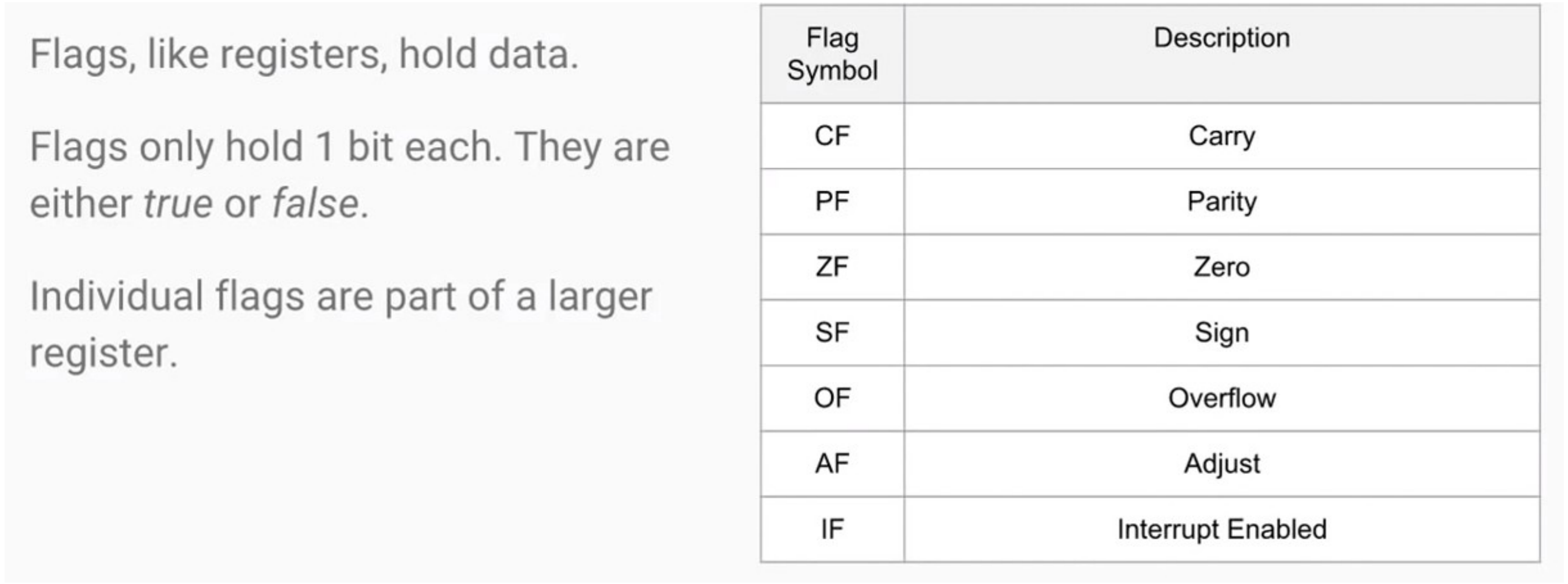
Linux Assembly

Armine Hayrapetyan

Outline

- Flags
- Pointers
- Control Flow
- Jumps
- Comparisons
- Calls

Flags



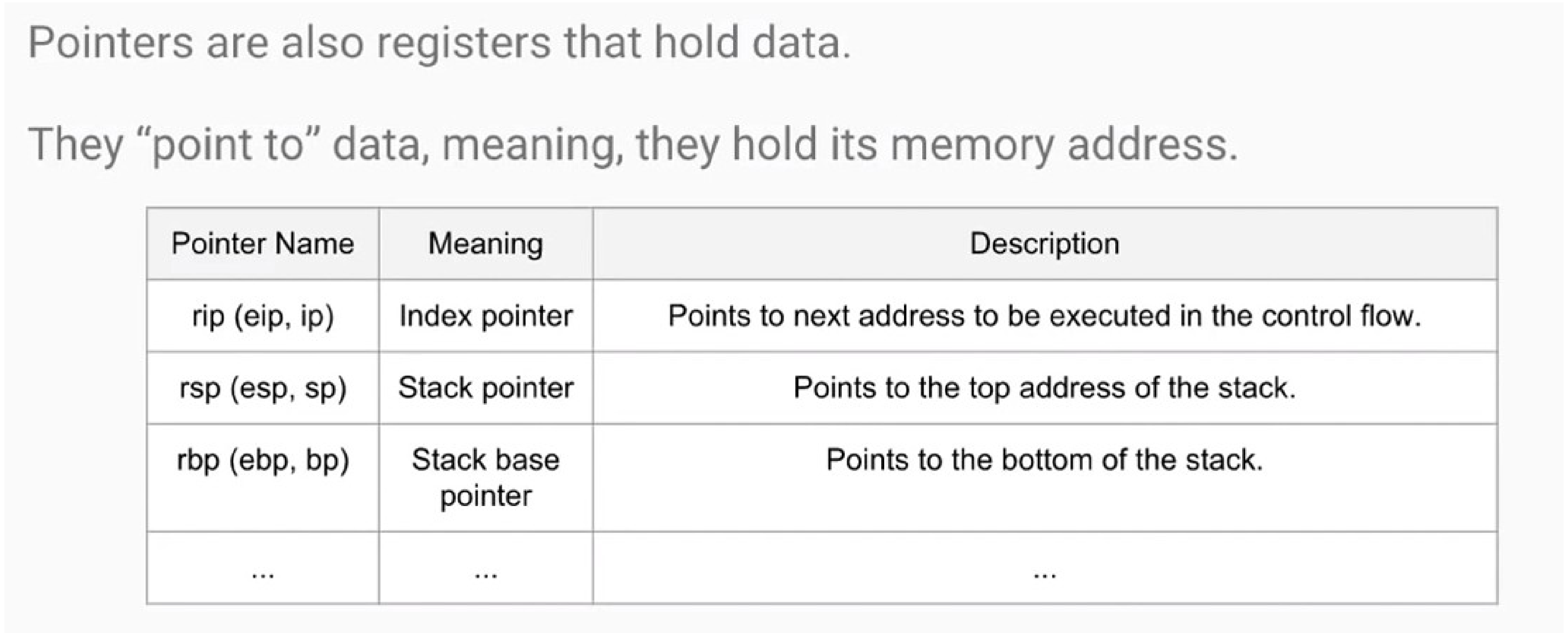
Flags, like registers, hold data.

Flags only hold 1 bit each. They are either *true* or *false*.

Individual flags are part of a larger register.

Flag Symbol	Description
CF	Carry
PF	Parity
ZF	Zero
SF	Sign
OF	Overflow
AF	Adjust
IF	Interrupt Enabled

Pointers

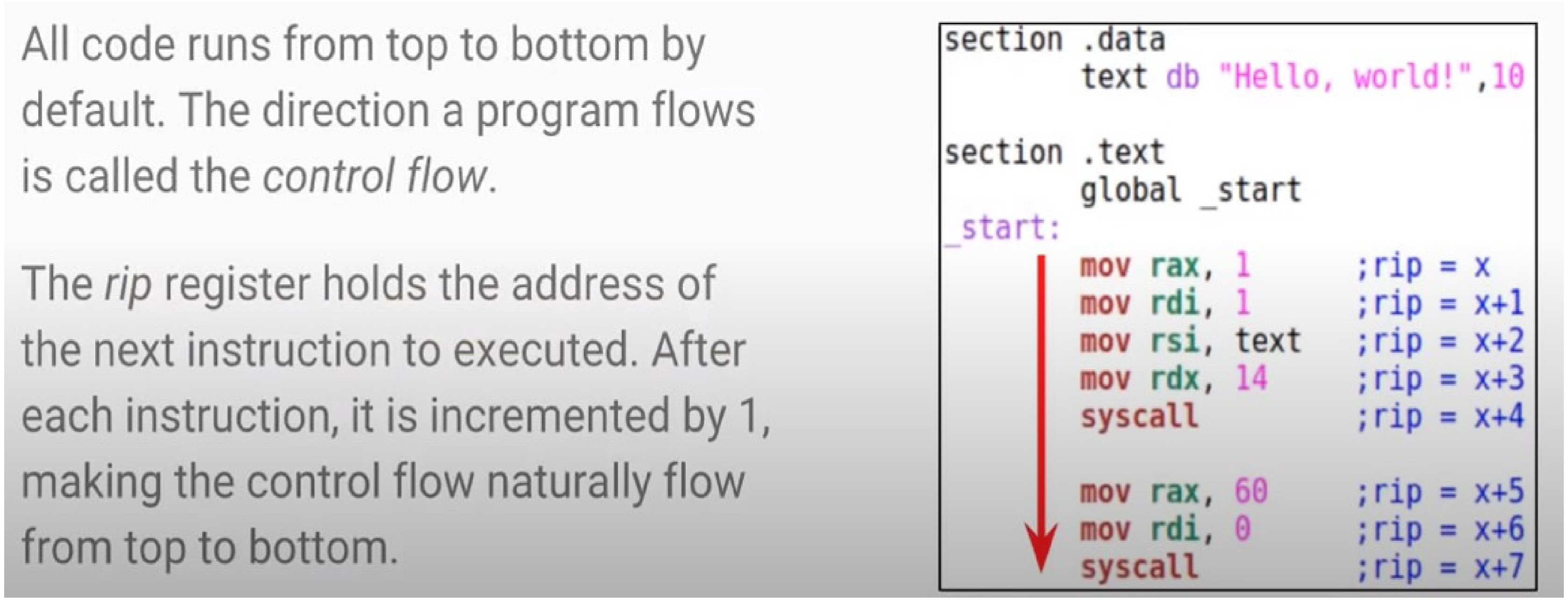


Pointers are also registers that hold data.

They “point to” data, meaning, they hold its memory address.

Pointer Name	Meaning	Description
rip (eip, ip)	Index pointer	Points to next address to be executed in the control flow.
rsp (esp, sp)	Stack pointer	Points to the top address of the stack.
rbp (ebp, bp)	Stack base pointer	Points to the bottom of the stack.
...	...	...

Control Flow



All code runs from top to bottom by default. The direction a program flows is called the *control flow*.

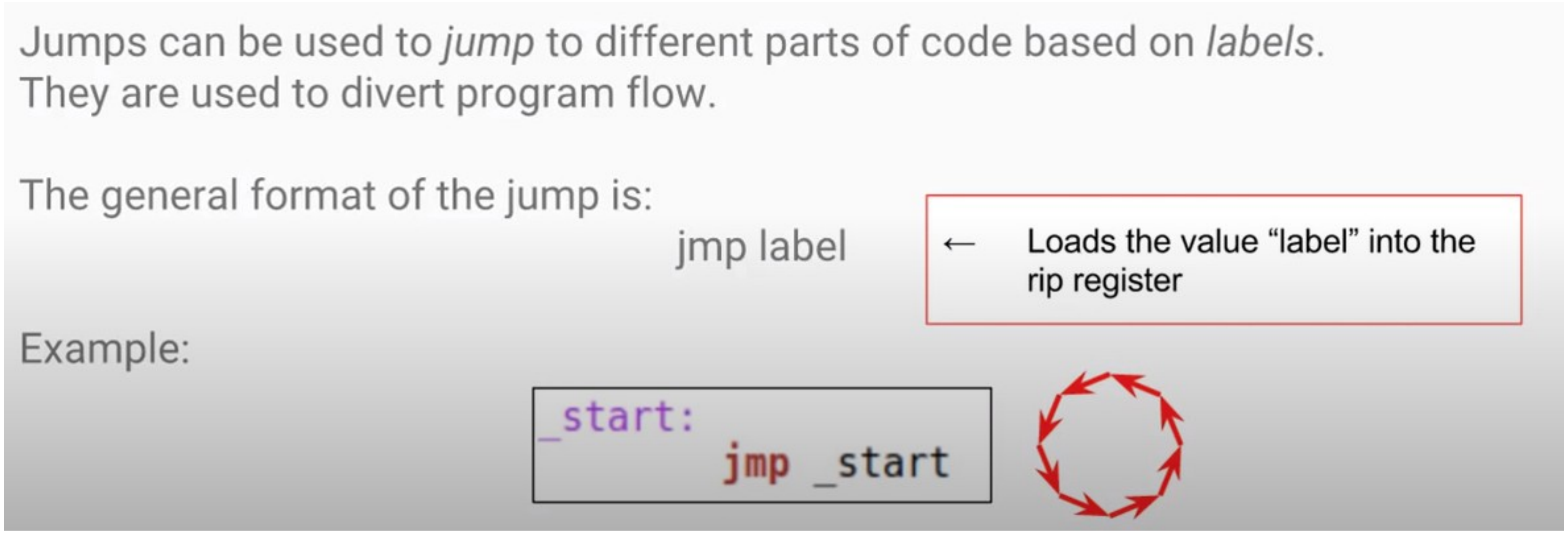
The *rip* register holds the address of the next instruction to executed. After each instruction, it is incremented by 1, making the control flow naturally flow from top to bottom.

```
section .data
        text db "Hello, world!",10

section .text
        global _start
_start:
        mov rax, 1          ;rip = x
        mov rdi, 1          ;rip = x+1
        mov rsi, text       ;rip = x+2
        mov rdx, 14         ;rip = x+3
        syscall             ;rip = x+4

        mov rax, 60         ;rip = x+5
        mov rdi, 0          ;rip = x+6
        syscall             ;rip = x+7
```

Jumps



Jumps can be used to *jump* to different parts of code based on *labels*.
They are used to divert program flow.

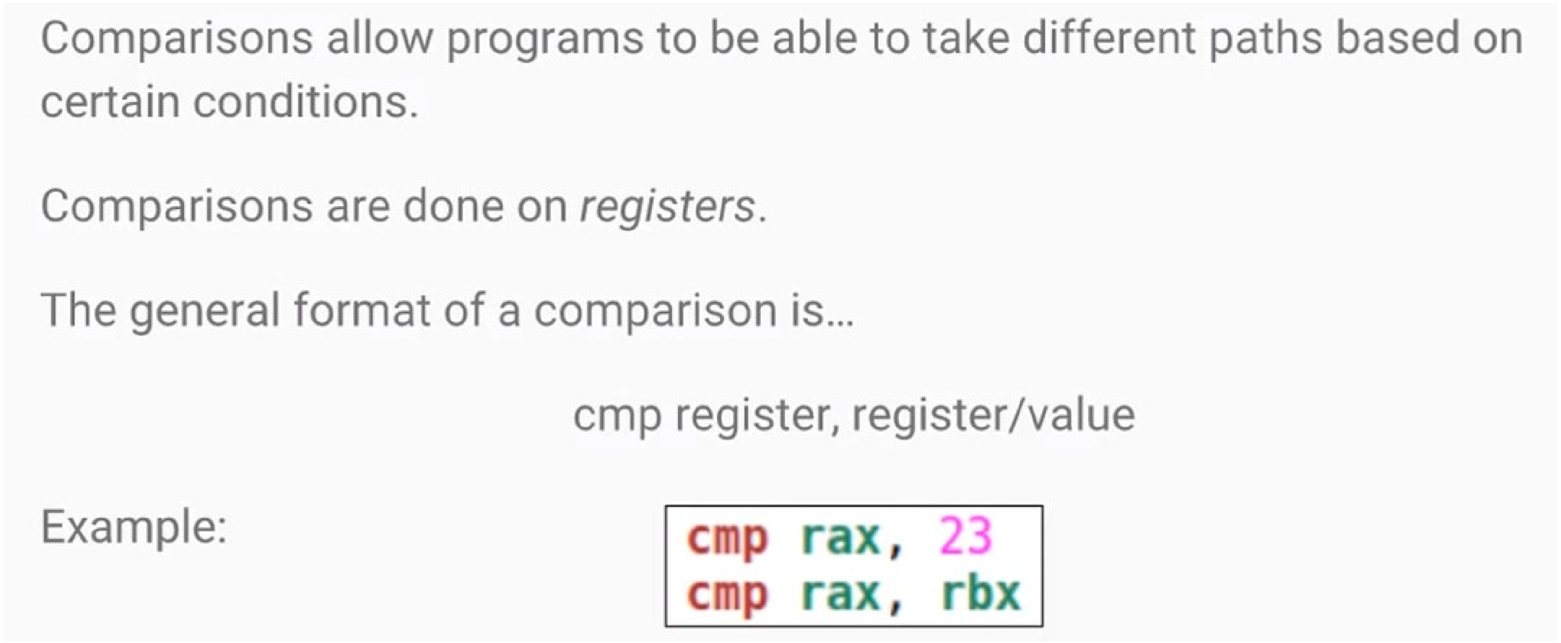
The general format of the jump is:

jmp label ← Loads the value “label” into the rip register

Example:

```
_start:
        jmp _start
```

Comparisons



Comparisons allow programs to be able to take different paths based on certain conditions.

Comparisons are done on *registers*.

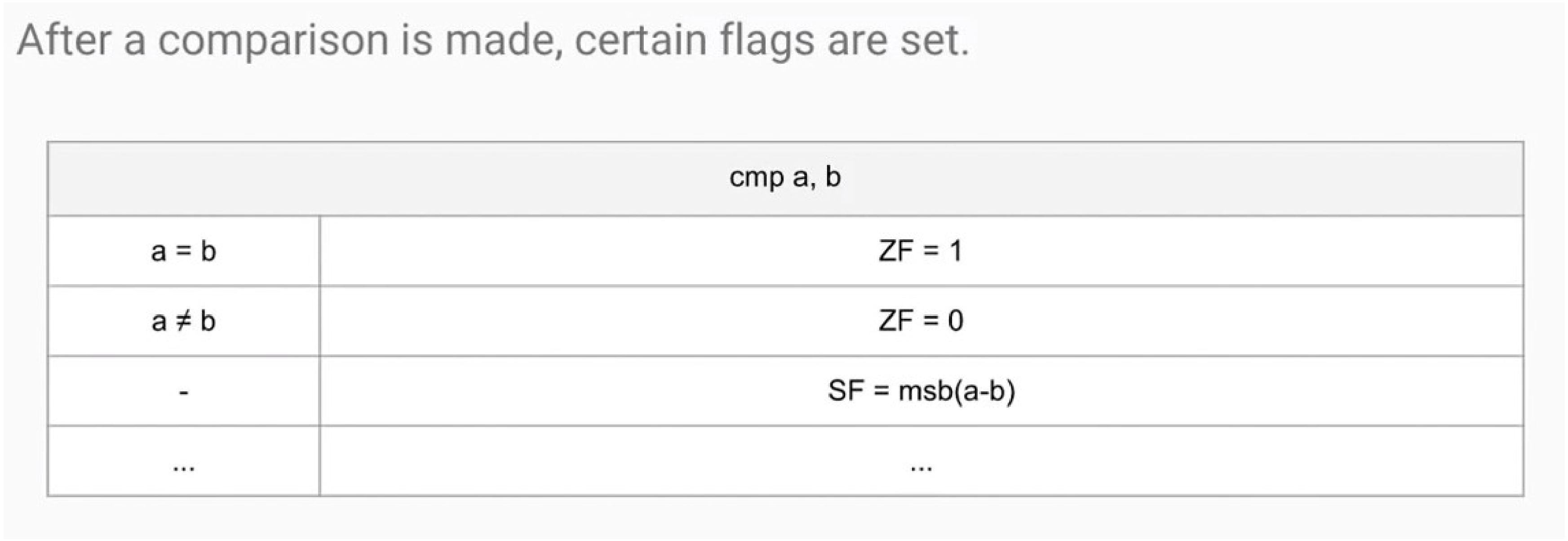
The general format of a comparison is...

cmp register, register/value

Example:

```
cmp rax, 23
cmp rax, rbx
```

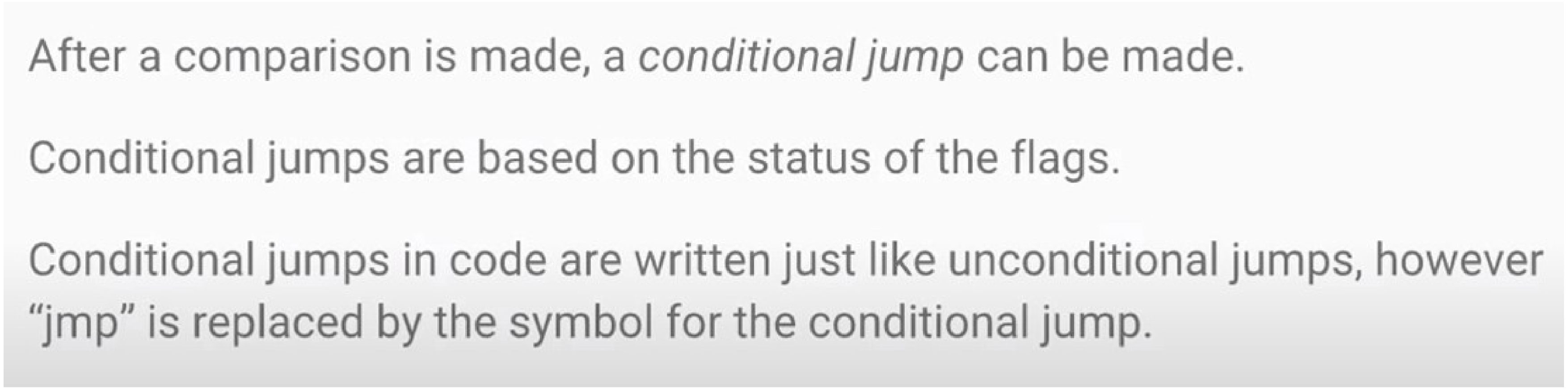
Comparisons with Flags



After a comparison is made, certain flags are set.

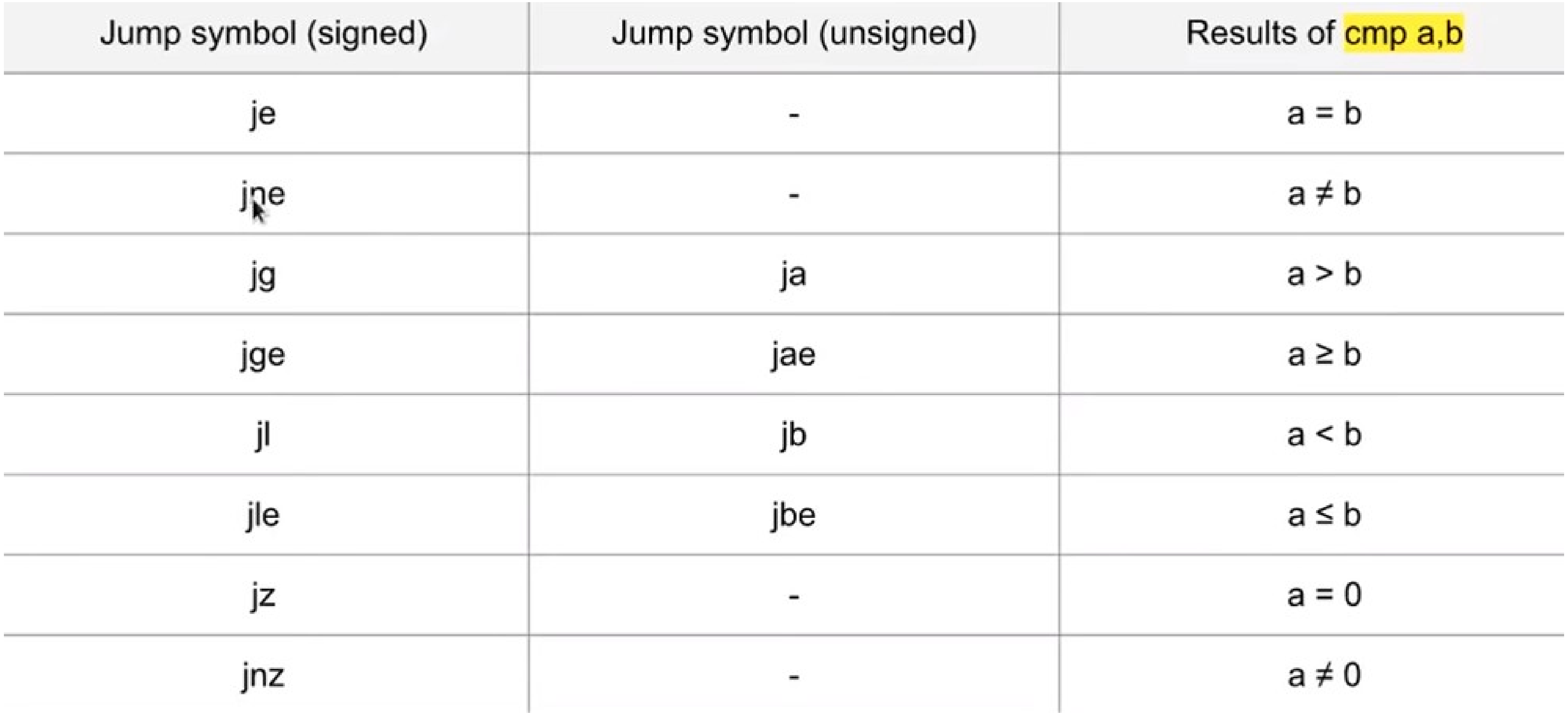
cmp a, b	
a = b	ZF = 1
a ≠ b	ZF = 0
-	SF = msb(a-b)
...	...

Conditional Jumps


After a comparison is made, a *conditional jump* can be made.

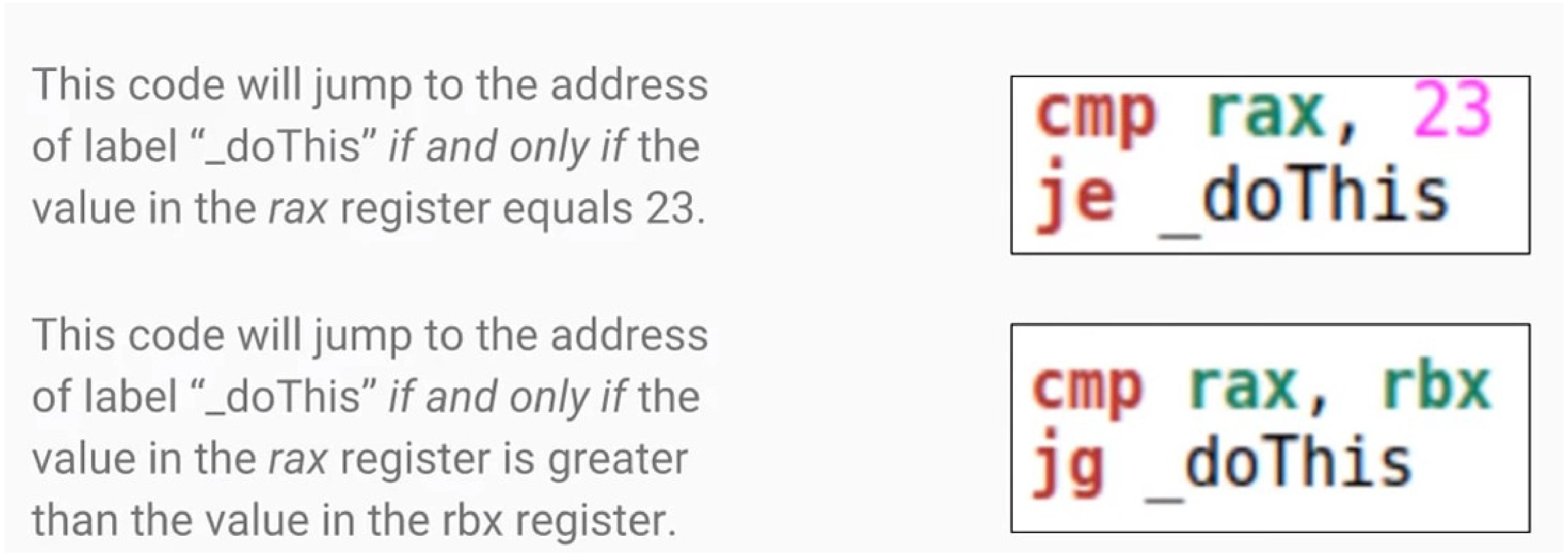
Conditional jumps are based on the status of the flags.

Conditional jumps in code are written just like unconditional jumps, however “jmp” is replaced by the symbol for the conditional jump.



Jump symbol (signed)	Jump symbol (unsigned)	Results of cmp a,b
je	-	a = b
jne	-	a ≠ b
jg	ja	a > b
jge	jae	a ≥ b
jl	jb	a < b
jle	jbe	a ≤ b
jz	-	a = 0
jnz	-	a ≠ 0

Conditional Jump Examples



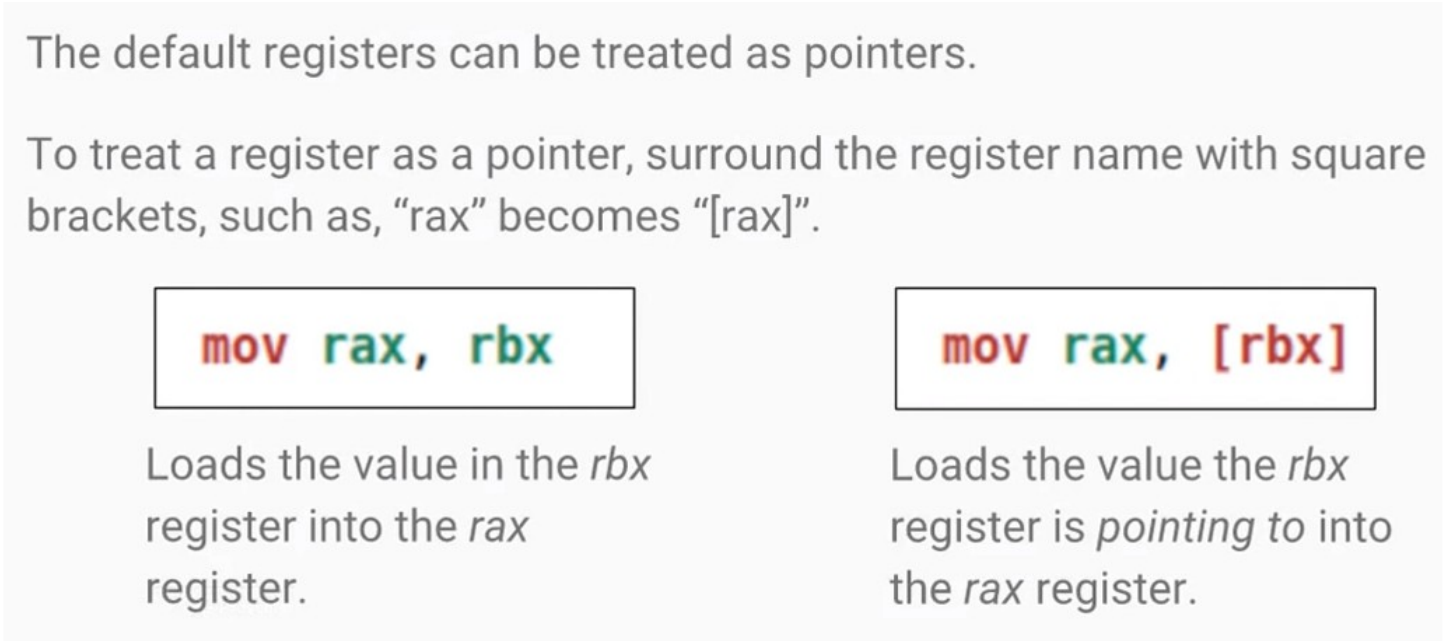
This code will jump to the address of label "_doThis" *if and only if* the value in the *rax* register equals 23.

```
cmp rax, 23
je _doThis
```

This code will jump to the address of label "_doThis" *if and only if* the value in the *rax* register is greater than the value in the rbx register.

```
cmp rax, rbx
jg _doThis
```

Registers as Pointers



The default registers can be treated as pointers.

To treat a register as a pointer, surround the register name with square brackets, such as, “rax” becomes “[rax]”.

```
mov rax, rbx
```

Loads the value in the *rbx* register into the *rax* register.

```
mov rax, [rbx]
```

Loads the value the *rbx* register is *pointing to* into the *rax* register.

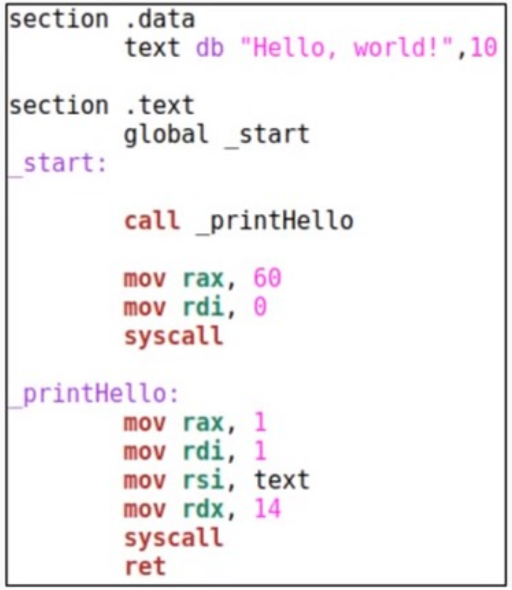
Calls

Calls and jumps are essentially the same.

However, when “call” is used, the original position the call was made can be returned to using “ret”.

In this modification of the “Hello, World!” code, the part of code that prints “Hello, World!” was moved into its own section, and that section was *called*.

This is called a *subroutine*.



```
section .data
        text db "Hello, world!",10

section .text
        global _start
_start:

        call _printHello

        mov rax, 60
        mov rdi, 0
        syscall

_printHello:
        mov rax, 1
        mov rdi, 1
        mov rsi, text
        mov rdx, 14
        syscall
        ret
```

References

- https://www.youtube.com/watch?v=busHtSyx2-w&list=PLetF-YjXm-sCH6FrTz4AQhfH6INDQvQSn&index=3