X86-64 Registers

- Data are stored only in registers or memory
- Each register is 64bits (lower part vs upper part):
    - The lower part name starts with e (like eax, ebx, ecx, ..)
    - In 1972 with the 8008 CPU, this old process was (8 bits registers): a, b, c and d
    - In 1979 with the 8086 CPU, the next generation was 16 bits registers: ax, bx, cx and dx
    - In 1985 with the 80386 CPU, this was 32 bits registers (extended): eax, ebx, cdx and edx
    - In 2003 with 8086-64, it was 64 bits registers: rax, rbx, rcx, and rdx
- All registers can be used for general purposes except (%rsp and ?%rbp).

Instructions

- Those are the Basic operations performed by circuits

Copying Data:Copying Data:

```
mov <source>, <destination>
mov %rax, %rcx      #copies the value of %rax into %rcx (assignment but in opposite direction)
mov $32, %rdx       # putting constant into a register (immediate value 32)
```

To specify the size of the transfer, use:

- q for 64 bits (q for quad word)
- l for 32 bits. (l for long)

For example:

```
movq $15, %r8           #64bits
movl %rcx, %eax         #32 bits
```

Arithmetic:

```
add <source>, <destination>
addq %rax, %r11     # r11 +=rax

sub <source>, <destination>
subq $30, %rcx      # rcx -=30

imul <source>, <destination>
imulq %rcx, %rdi    # rdi *= rcx    (integer multiplication)

inc <register>
incq %rcx     # rcx +=1

dec <register>
decq %rcx     # rcx -=1
```

Comparison:

```
cmp <second_operand>, <first_operand>
cmpq %rcx, %rdx     # compares %rdx to %rcx
```

- It remembers the result (save it in one of the flags):
  if (first_operand < second_operand)
  if (first_operand > second_operand)
  if (first_operand = second_operand)
- The result of the comparison is stored in flag registers (in zero bit or sign bit)
- In high-level programming langs, multiple comparisons are divided by the compiler usually into simple comparisons.

Jump:

- It transfers the execution to another place (somewhere else). it doesn't move any data but just jumps.
- Conditional jump only applies to the most recent comparison.

```
jmp <LABEL          # jump to LABEL anyway.
jg <LABEL>          # If the result of the previous comparison was "greater than", jump to LABEL
jl <LABEL>          # If the result of the previous comparison was "less than", jump to LABEL
je <LABEL>          # If the result of the previous comparison was "equals to", jump to LABEL
jge <LABEL>         # If the result of the previous comparison was ">=", jump to LABEL
jle <LABEL>         # If the result of the previous comparison was "<=", jump to LABEL
```