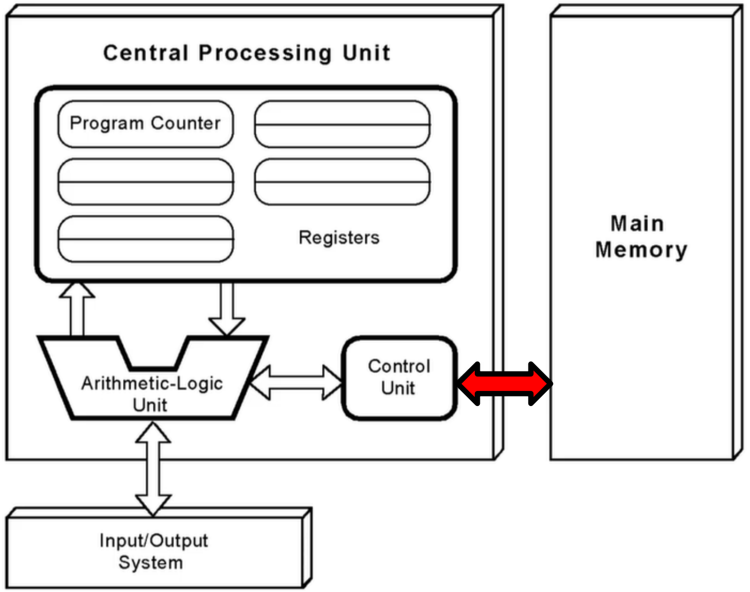
x86 Architecture

CMSC 313
Raphael Elspas

Intel x86-64 architecture

- Based off the von-Neumann
- ALU, Control and Registers are all in CPU
- Note registers are different than memory (RAM)
- Instruction and data memory share the same path between memory and the CPU (labeled in red)

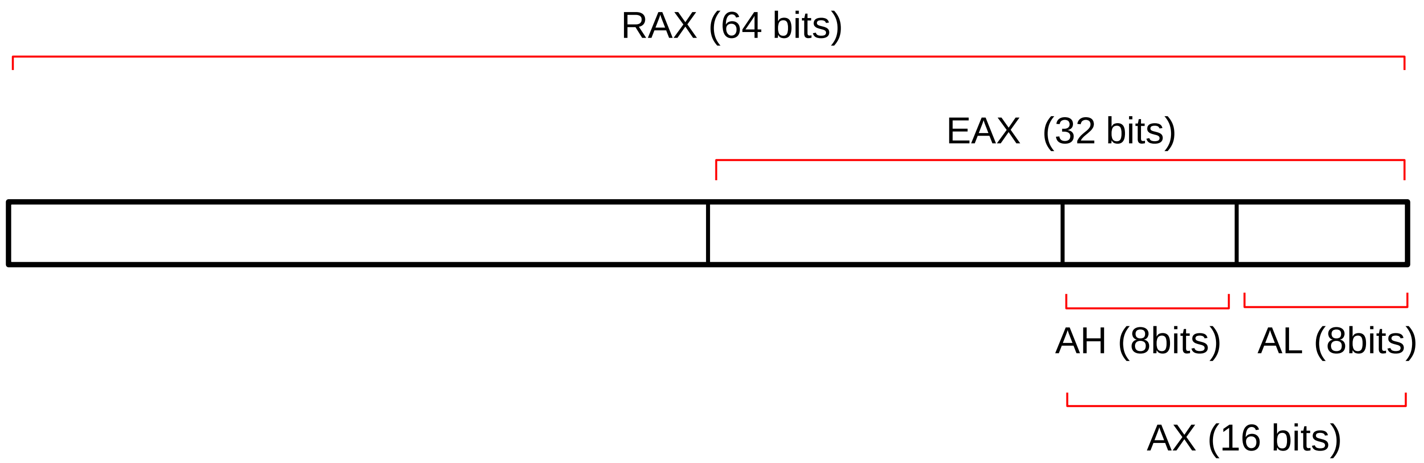


X86-64 general purpose registers

- Intel 32-bit architecture has 8 general purpose registers
- 64-bit architecture has 16 general purpose registers
- Most intel processors these days are 64 bit
- Registers are designed to be “backwards compatible” and most can run on both 32 and 64 bit architectures
  - Standardization is done through register naming conventions
  - EAX refers to a 32 bit register (can be read on 32 and 64 bit)
  - RAX refers to a 64 bit register (can only be read on 64 bit)
- Many of the GP registers have special uses
  - Can or cannot be used by specific instructions
  - Need to look up which register is used by which instruction
  - Depending on architecture (32/64 bit) register for a specific instruction may change

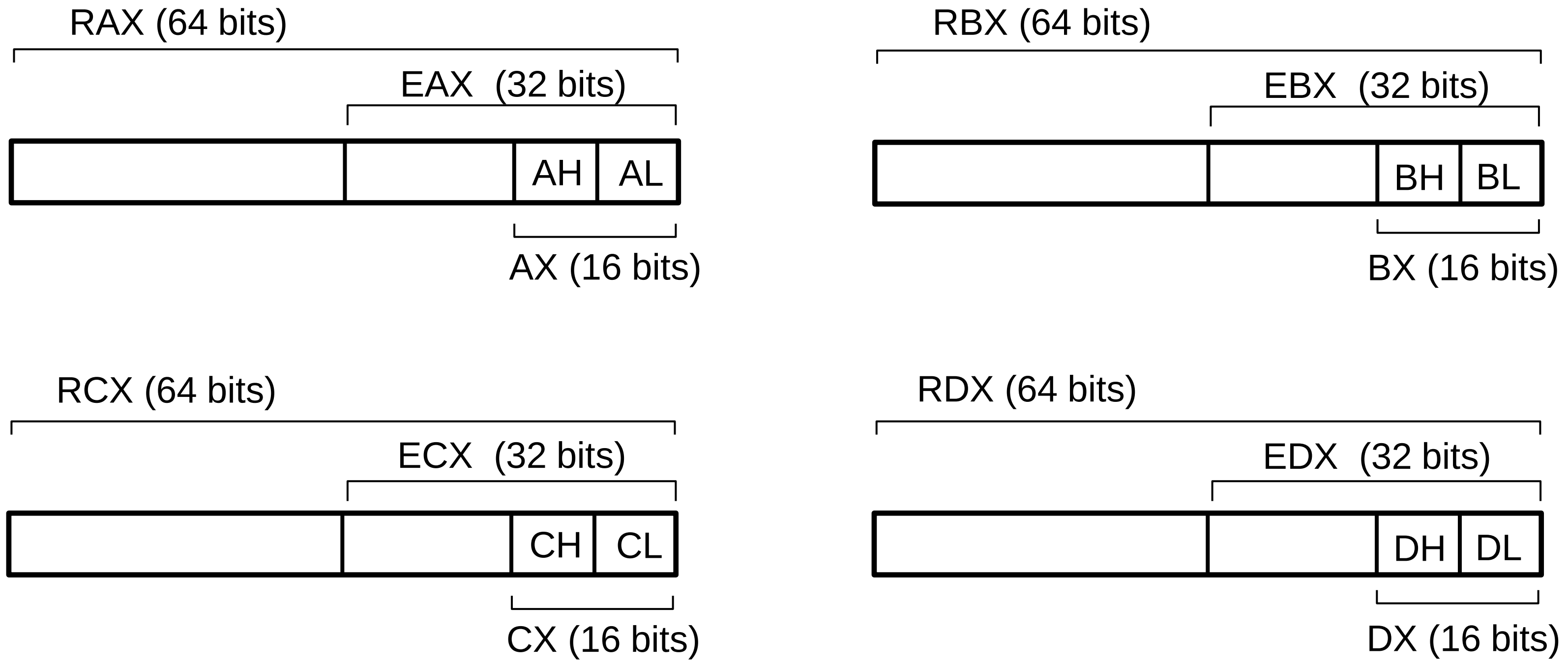
X86-64 registers

- RAX, EAX, AX, AH, AL share the same register
- They have different accesses and are different lengths



X86-64 registers (cont.)

- RAX, RBX, RCX, RDX



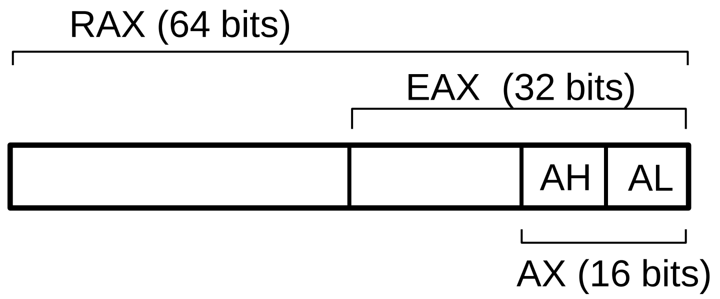
Example

- If I set ah to 5, what value will be read on al, ax, eax, and rax in decimal?

test with:

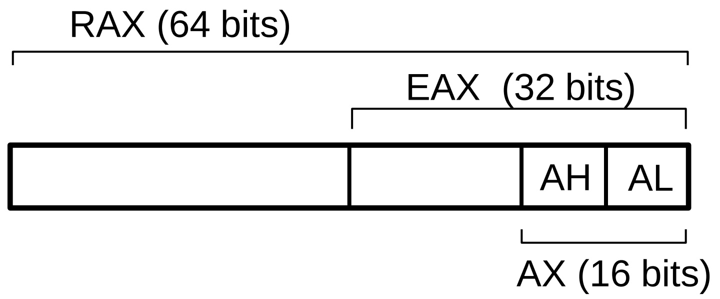
```
mov ah, 5
```

@ http://asmdebugger.com/



- Answer:
- al = 0
- ax = 0x0500 = 1280_{10}
- eax = 1280_{10}
- rax = 1280_{10}

Example



- If I set ah to 0xFF, and then add 2, what will be read on al, ah, ax, eax, and rax in hex?

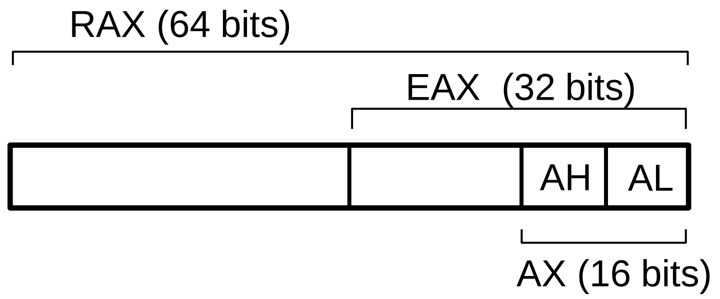
test with:

```
mov ah, 0xFF
add ah, 2
```

@ http://asmdebugger.com/

- Answer: moving 0xff into ah, overwrites the sign bit of ah with a 1, producing ah = -1. Therefore adding 2, ah = 1
- al = 0
- ah = 0x01
- ax = 0x0100
- eax = 0x0000 0100
- rax = 0x0000 0000 0000 0100 = 0x100

Example



- If I set ah to 0xFF, and then add 2, which flags will be affected (OF, SF, CF, ZF)?

test with:

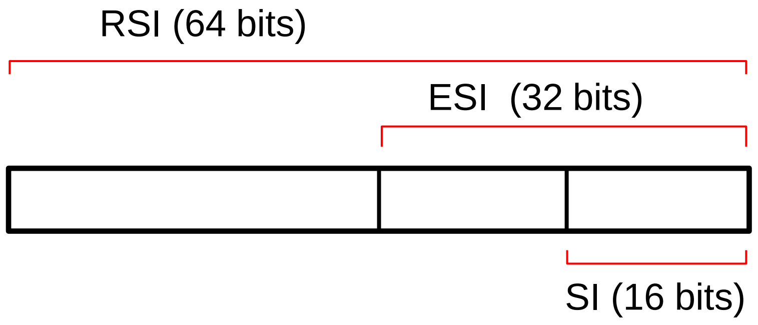
```
mov ah, 0xFF
add ah, 2
```

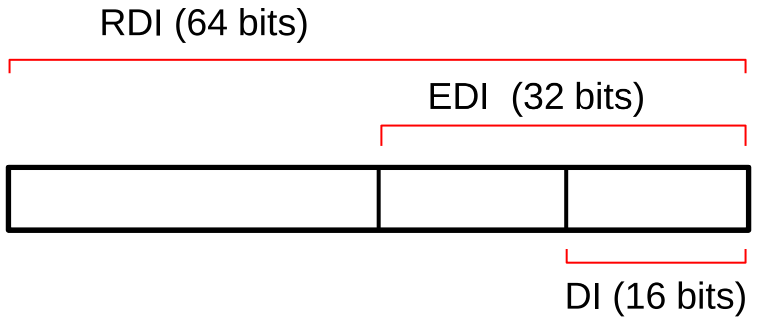
@ http://asmdebugger.com/

- Answer: moving 0xff into ah, overwrites the sign bit of ah with a 1, producing ah= -1. Therefore, after adding 2, ah = 1.
- OF = 0. -1 + 2 = (neg) + (pos) NO OVERFLOW
- SF = 0. -1 + 2 = 1. Result is positive
- CF = 1. -1 + 2 = $1111\ 1111_2$ + $0000\ 0010_2$ produces a carry out bit.
- ZF = 0. -1 + 2 = 1. Result is non-zero

X86-64 registers (cont.)

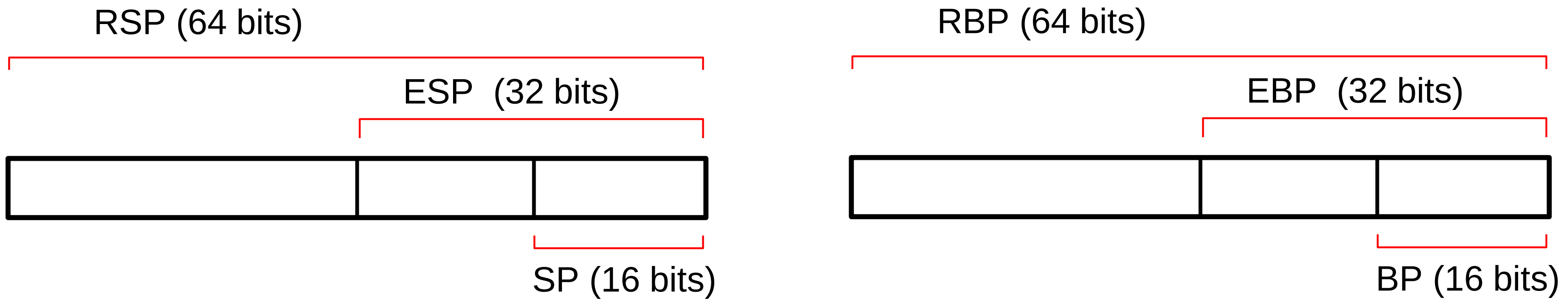
- RSI, ESI, SI share the same register, S stands for source
- RDI, EDI, DI share the same register, D stands for destination
- Not always used as source and destination, these are legacy names.
- These are general purpose registers





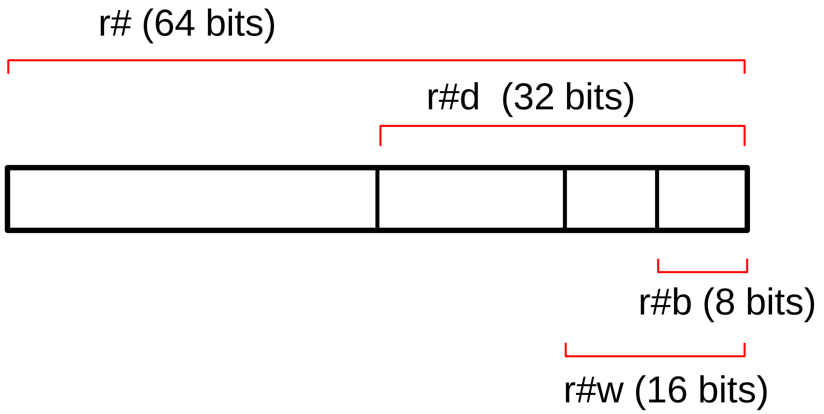
Stack registers

- The Stack pointer has accesses: RSP, ESP, SP share the same register
- Another register called RBP, EBP, and BP identifies the base of the stack.
- Is technically are general purpose registers, but shouldn't be used for storing temporary values
- Editing these should only be done when modifying the stack



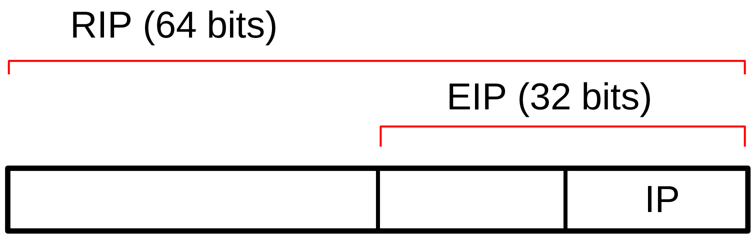
X86-64 registers (cont.)

- R8 – R15 are registers only available on a 64 bit architecture
  - B stands for byte (1 byte)
  - W stands for word (2 bytes)
  - D stands for double word (4 bytes)
  - Q stands for quad word (8 bytes) (we leave out q when identifying the entire register



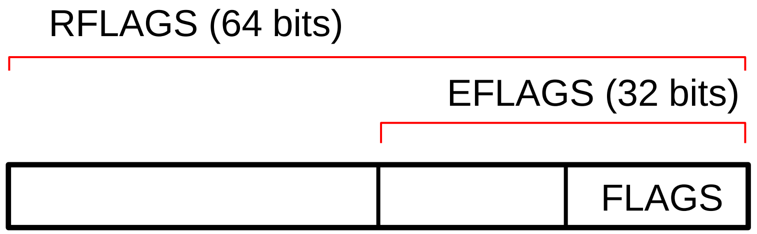
X86-64 Instruction Pointer register

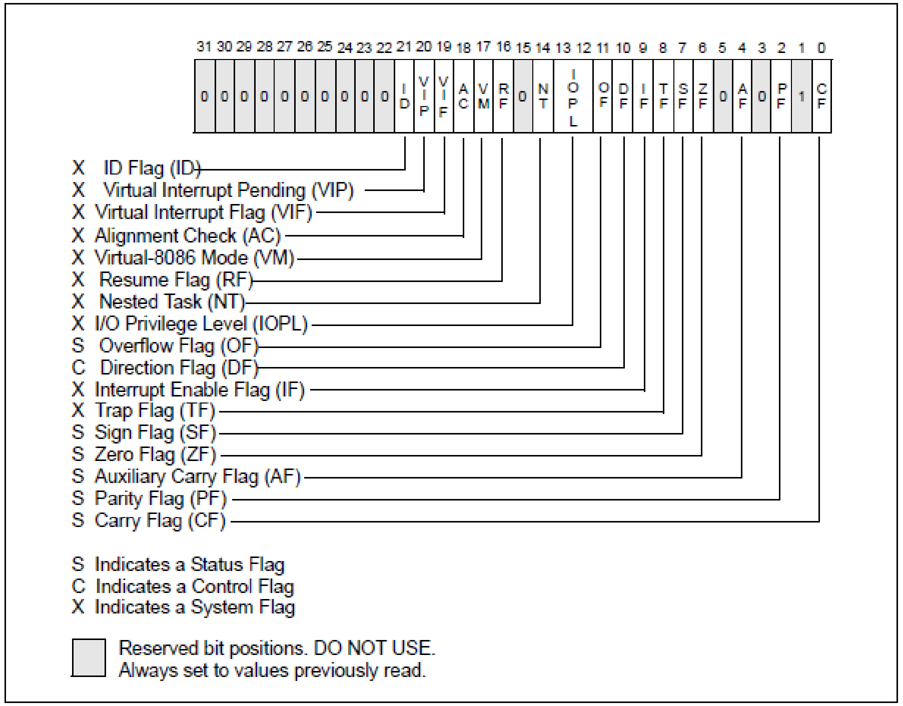
- RIP, EIP, IP identify the register that points to which instruction the CPU will execute next.
- Same thing as the PC (Program Counter)
- Modifying this register will likely seg-fault your program



Flags register

- RFLAGS, EFLAGS, FLAGS
- Stores carry flag, zero flag, sign flag, overflow flag, many other flags





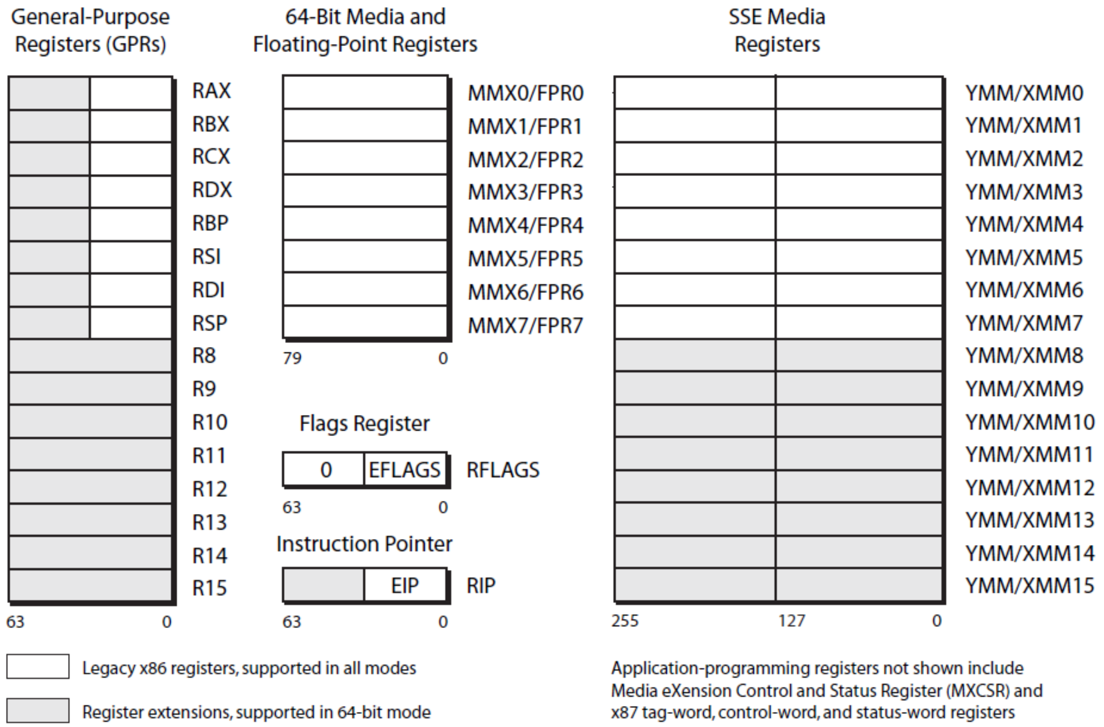
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 ID VIP VIF AC VM RF 0 NT IOPL OF DF IF TF SF ZF 0 AF 0 PF 1 CF
X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)
S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag
Reserved bit positions. DO NOT USE. Always set to values previously read.

EFLAGS Register

Summary of registers

Notes:

- MMX# and XMM# registers we won't really use.
- MMX are for floating point numbers
- XMM are for longer length values



References

- Ivan Sekyonda’s slides
- https://en.wikipedia.org/wiki/FLAGS_register
- http://asmdebugger.com/
- https://kobzol.github.io/davis/