X86 history

- Lots of other ISAs
- RISC vs. CISC

- 8086 \rightarrow \text{one of the first single chip microprocessors}

\text{l7 80286} \rightarrow \text{80386 (i386)}
\rightarrow \text{486 ... X86}

Pentium 3 (’98) (PC)
\downarrow
Pentium 4
\vdots
Core 2 arch.
- Subset of x86 instructions
  - Data movement
  - Arithmetic & logical
  - Control flow

---

```
movl %eax, %ebx
```

- Only covering AT&T syntax (GAS)
- Intel syntax => drops suffixes
  -> switches operand ordering
- Most common operand is registers

Registers: place hold temp. data

Most hardware/instr. directly interacts w/ registers

\[ \text{CPU} \quad \text{registers (load, store)} \quad \text{Memory} \]

\[ \text{mov instr.} \]

\[ \text{x86} \rightarrow 8 \text{ registers} \rightarrow i386 \]
- general purpose registers
  - can be used for anything
  - except when they can't

-x86 → "word" 16-bits (2-bytes)
(double) → "long" 32-bits (4-bytes)
        → "quad" 64-bits (8-bytes)
operand specifiers

\text{instr op1, op2}

$\rightarrow$ what's allowed here

- Registers $\rightarrow$ %eax (value in the register)

- Immediate $\rightarrow$ $0x1000$

\text{mov}$ $12$, %eax

$12$ (this value)

$0xFF$ $-15$

---

Addressing memory (value in addr $0x1000$)

- Absolute addressing: $0x1000$

\text{mov}$ $0x1000$, %eax

- Register direct: $(%eax) \rightarrow$ Value in address held in %eax

$(%esi)$

$(%esp)$
base-displacement: 8(%eax)

Imm (%R) \rightarrow \text{value in address held in } %eax + 8

-16(%eax)

0(%esp)

Scaled index: most general memory addressing

$\text{Imm (}\%R, %R, \text{Imm}) \rightarrow \text{size}$

0(%eax, %ebx)I

$\rightarrow \text{value in address} \%eax + (\%ebx \cdot 1)$

= 0(%ebx, %edx, 4)

$\rightarrow \text{value in address} \%ebx + \%edx \cdot 4$

8(%esp, %edx, 82)

$\rightarrow \text{value in address} \%esp + \%edx \cdot 2 + 8$
mem (Abs. Addressing)

0x1000 $0xABC0

$0x1000 $0x1000

L$Imm.