Computer organization and architecture

Lesson 12

x86 architecture

Almost all personal computers today use x86 architecture microprocessors.

x86, also called IA-32, is a 32-bit architecture originally developed by Intel.

AMD also sells x86 compatible microprocessors.

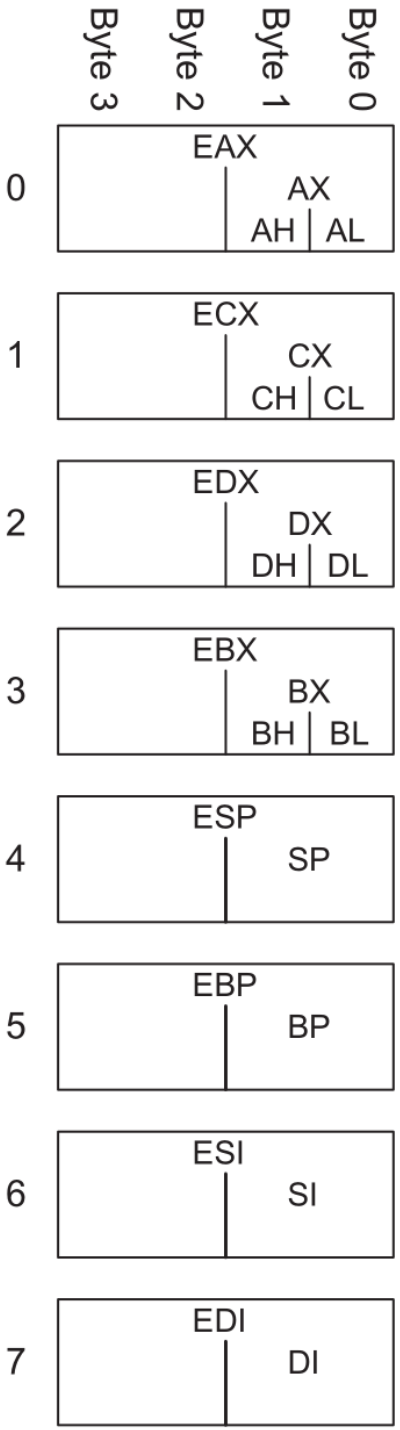
x86 is an example of a Complex Instruction Set Computer (CISC) architecture.

In contrast to RISC architectures such as ARM, each CISC instruction can do more work.

Programs for CISC architectures usually require fewer instructions.

Major differences between ARM and x86

Feature	ARM	x86
# of registers	15 general purpose	8, some restrictions on purpose
# of operands	3–4 (2–3 sources, 1 destination)	2 (1 source, 1 source/destination)
operand location	registers or immediates	registers, immediates, or memory
operand size	32 bits	8, 16, or 32 bits
condition flags	yes	yes
instruction types	simple	simple and complicated
instruction encoding	fixed, 4 bytes	variable, 1–15 bytes



Eight 32-bit registers

The bottom 16 bits and some of the bottom 8-bit portions are also usable

– almost, but not quite, general purpose

Certain instructions cannot use certain registers.

Other instructions always put their results in certain registers.

Like SP in ARM, ESP is normally reserved for the stack pointer.

The x86 program counter is called the EIP (**the extended instruction pointer**).

x86 instructions may operate on registers, immediates, or memory.

x86 instructions specify only two operands.

The first is a source.

The second is both a source and the destination.

Hence, x86 instructions always overwrite one of their sources with the result.

Source/ Dest	Source	Example	Meaning
register	register	`add EAX, EBX`	`EAX <– EAX + EBX`
register	immediate	`add EAX, 42`	`EAX <– EAX + 42`
register	memory	`add EAX, [20]`	`EAX <– EAX + Mem[20]`
memory	register	`add [20], EAX`	`Mem[20] <– Mem[20] + EAX`
memory	immediate	`add [20], 42`	`Mem[20] <– Mem[20] + 42`

Memory addressing modes

Example	Meaning	Comment
`add EAX, [20]`	`EAX <– EAX + Mem[20]`	displacement
`add EAX, [ESP]`	`EAX <– EAX + Mem[ESP]`	base addressing
`add EAX, [EDX+40]`	`EAX <– EAX + Mem[EDX+40]`	base + displacement
`add EAX, [60+EDI*4]`	`EAX <– EAX + Mem[60+EDI*4]`	displacement + scaled index
`add EAX, [EDX+80+EDI*2]`	`EAX <– EAX + Mem[EDX+80+EDI*2]`	base + displacement + scaled index

x86 has a 32-bit memory space that is byte-addressable.

x86 supports a wider variety of memory indexing modes.

Memory locations are specified with any combination of a base register, displacement, and a scaled index register.

The displacement can be an 8-, 16-, or 32-bit value.

The scale multiplying the index register can be 1, 2, 4, 8.

Instructions acting on 8-, 16-, or 32-bit data

Example	Meaning	Data Size
`add AH, BL`	`AH <– AH + BL`	8-bit
`add AX, –1`	`AX <– AX + 0xFFFF`	16-bit
`add EAX, EDX`	`EAX <– EAX + EDX`	32-bit

x86, like many CISC architectures, uses condition flags (also called status flags) to make decisions about branches and to keep track of carries and arithmetic overflow.

x86 uses a 32-bit register, called EFLAGS, that stores the status flags.

Some of the bits of the EFLAGS:

Name	Meaning
CF (Carry Flag)	Carry out generated by last arithmetic operation. Indicates overflow in unsigned arithmetic. Also used for propagating the carry between words in multiple-precision arithmetic
ZF (Zero Flag)	Result of last operation was zero
SF (Sign Flag)	Result of last operation was negative (msb = 1)
OF (Overflow Flag)	Overflow of two's complement arithmetic

The architectural state of an x86 processor includes EFLAGS as well as the eight registers and the EIP.

x86 instructions

x86 has a larger set of instructions than ARM.

x86 also has instructions for floating-point arithmetic and for arithmetic on multiple short data elements packed into a longer word.

Some instructions always act on specific registers.

For example, 32×32-bit multiplication always takes one of the sources from EAX and always puts the 64-bit result in EDX and EAX.

`LOOP` always stores the loop counter in ECX.

`PUSH` , `POP` , `CALL` , and `RET` use the stack pointer, ESP

D – destination (a register or memory location)

S – source (a register, memory location, or immediate)

Instruction	Meaning	Function
ADD/SUB	add/subtract	D = D + S / D = D − S
ADDC	add with carry	D = D + S + CF
INC/DEC	increment/decrement	D = D + 1 / D = D − 1
CMP	compare	Set flags based on D − S
NEG	negate	D = − D
AND/OR/XOR	logical AND/OR/XOR	D = D op S
NOT	logical NOT	$D = \overline{D}$

IMUL/MUL	signed/unsigned multiply	EDX:EAX = EAX × D
IDIV/DIV	signed/unsigned divide	EDX:EAX/D EAX = Quotient; EDX = Remainder
SAR/SHR	arithmetic/logical shift right	D = D >>> S / D = D >> S
SAL/SHL	left shift	D = D << S
ROR/ROL	rotate right/left	Rotate D by S
RCR/RCL	rotate right/left with carry	Rotate CF and D by S
BT	bit test	CF = D[S] (the S*th* bit of D)
BTR/BTS	bit test and reset/set	CF = D[S]; D[S] = 0 / 1
TEST	set flags based on masked bits	Set flags based on D AND S

MOV	move	D = S
PUSH	push onto stack	ESP = ESP −4; Mem[ESP] = S
POP	pop off stack	D = MEM[ESP]; ESP = ESP + 4
CLC, STC	clear/set carry flag	CF = 0 / 1
JMP	unconditional jump	relative jump: EIP = EIP + S absolute jump: EIP = S
Jcc	conditional jump	if (flag) EIP = EIP + S
LOOP	loop	ECX = ECX −1 if (ECX ≠ 0) EIP = EIP + imm
CALL	function call	ESP = ESP −4; MEM[ESP] = EIP; EIP = S
RET	function return	EIP = MEM[ESP]; ESP = ESP + 4

Conditional jumps check the flags and branch if the appropriate condition is met.

For example, `JZ` jumps if the zero flag (ZF) is 1.

`JNZ` jumps if the zero flag is 0.

Like ARM, the jumps usually follow an instruction, such as the compare instruction (`CMP`), that sets the flags.

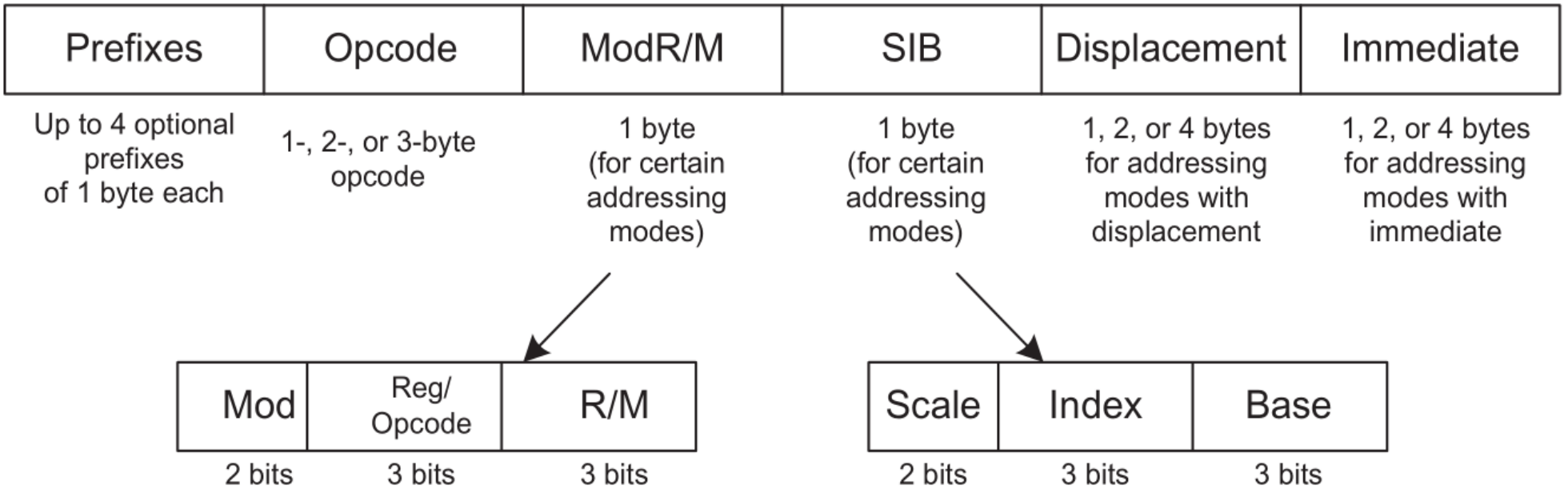
Instruction	Meaning	Function after CMP D, S
JZ/JE	jump if ZF = 1	jump if D = S
JNZ/JNE	jump if ZF = 0	jump if D ≠ S
JGE	jump if SF = OF	jump if D ≥ S
JG	jump if SF = OF and ZF = 0	jump if D >S
JLE	jump if SF ≠ OF or ZF = 1	jump if D ≤ S
JL	jump if SF ≠ OF	jump if D <S
JC/JB	jump if CF = 1	
JNC	jump if CF = 0	
JO	jump if OF = 1	
JNO	jump if OF = 0	
JS	jump if SF = 1	
JNS	jump if SF = 0	

More branch conditions exist

x86 instruction encoding

The x86 instruction encodings are truly messy, a legacy of decades of piece-meal changes.

Unlike ARMv4, whose instructions are uniformly 32 bits, x86 instructions vary from 1 to 15 bytes.



ModR/M, SIB, Displacement, Immediate - optional fields

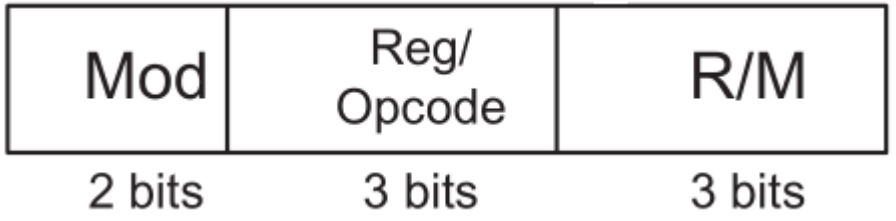
ModR/M specifies an addressing mode.

SIB specifies the scale, index, and base registers in certain addressing modes.

Displacement indicates a 1-, 2-, or 4-byte displacement in certain addressing modes.

And Immediate is a 1-, 2-, or 4-byte constant for instructions using an immediate as the source operand.

An instruction can be preceded by up to four optional byte-long prefixes that modify its behavior.



The ModR/M byte uses the 2-bit Mod and 3-bit R/M field to specify the addressing mode for one of the operands.

The operand can come from one of the eight registers, or from one of 24 memory addressing modes.

Due to artifacts in the encodings, the ESP and EBP registers are not available for use as the base or index register in certain addressing modes.

The Reg field specifies the register used as the other operand.

For certain instructions that do not require a second operand, the Reg field is used to specify three more bits of the opcode.

In addressing modes using a scaled index register, the SIB byte specifies the index register and the scale (1, 2, 4, 8).

If both a base and index are used, the SIB byte also specifies the base register.

ARM fully specifies the instruction in the cond, op, and funct fields of the instruction.

x86 uses a variable number of bits to specify different instructions.

Some instructions even have multiple opcodes.

Example:

```
add AL, imm8
```

performs an 8-bit add of an immediate to AL.

It is represented with the 1-byte opcode, 0x04, followed by a 1-byte immediate.

The A register (AL, AX, or EAX) is called the **accumulator**.

On the other hand,

```
add D, imm8
```

performs an 8-bit add of an immediate to an arbitrary destination, D (memory or a register).

It is represented with the 1-byte opcode 0x80 followed by one or more bytes specifying D, followed by a 1-byte immediate.

Many instructions have shortened encodings when the destination is the accumulator.

x86 contains string instructions that act on entire strings of bytes or words.

The operations include moving, comparing, or scanning for a specific value.

In modern processors, these instructions are usually slower than performing the equivalent operation with a series of simpler instructions, so they are best avoided.

With 32-bit addresses, x86 can access 4 GB of memory.

Intel and Hewlett-Packard jointly developed a new 64-bit architecture called IA-64 in the mid 1990's.

It was designed from a clean slate, bypassing the convoluted history of x86, taking advantage of 20 years of new research in computer architecture, and providing a 64-bit address space.

However, IA-64 has not become popular.

Most computers needing the large address space now use the 64-bit extensions of x86 called x86-64 developed by AMD and later accepted by Intel.