IA-32 Instruction Set Architecture

CS 365 Lecture 4

Prof. Yih Huang

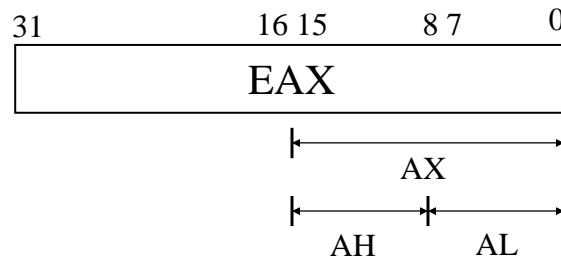
General-Purpose Registers

Assembly Name	Reg #
EAX	000
EBX	001
ECX	010
EDX	011
ESP	100
EDP	101
ESI	110
EDI	111

```
 31              16 15      8 7        0
┌──────────────────────────────────────┐
│                 EAX                   │
└──────────────────────────────────────┘
                    ├──────────────────┤
                            AX
                    ├─────────┼─────────┤
                      AH          AL
```

❑ We also have BX, BH, BL, CX, CH CL,
   DX, DH, DL.

❑ SP, BP, SI, DI are lower-halves of the other
   4 registers.

❑ When operating on 16-bit data, the 8 register
   numbers (000 – 111) refers to AX, BX, CX,
   DX, SP, BP, SI and DI.

❑ When operating on 8-bit data, the 8 register
   numbers (000 – 111) refers to AL, CL, DL,
   BL, AH, CH, DH and BH.

❑ Data width is specified by the opcode.

Instruction Format

Opcode	ModR/M	SIB	displacement	Immediate
1 or 2 bytes	1 byte, If required	1 byte, If required	1,2 or 4 bytes If required	1,2 or 4 bytes If required

- ❑ Opcode: determine the action
- ❑ ModR/M: Addressing modes register/memory
- ❑ SIB: Scale-Index-Base
- ❑ Not all fields are present in all instr.
- ❑ If present, must be in the above order

ModR/M

Mod	Reg #	R/M
2 bits	3 bits	3 bits

❑ **Mod=00**,
- – First operand a register, specified by Reg #
- – Second operand in memory; address stored in a register numbered by R/M.
  - ➢ That is, Memory[Reg[R/M]]
- – Exceptions:
  - ➢ R/M=100 (SP): SIB needed
  - ➢ R/M=101 (BP): disp32 needed

❑ **Mod=01**, same as Mod 00 with 8-bit displacement.

  – Second operand: Memory[disp8+Reg[R/M]]

  – Exception: SIB needed when R/M=100

❑ **Mod=10**, same as Mod 01 with 32-bit displacement

❑ **Mod=11**

  – Second operand is also a register, numbered by R/M.

❑ Do not confuse displacement width with data width.

  – Data width is specified by the opcode.

  – For example, the use of disp8 does not imply 8-bit data.

❑ *For some opcodes, the reg# is used as an extension of the opcode.*

SIB

Scale	Index	Base

2 bits   3 bits   3 bits

❑ Specify how a memory address is calculated

❑ Address=Reg[base] + Reg[Index]*2^{scale}

❑ Exceptions:

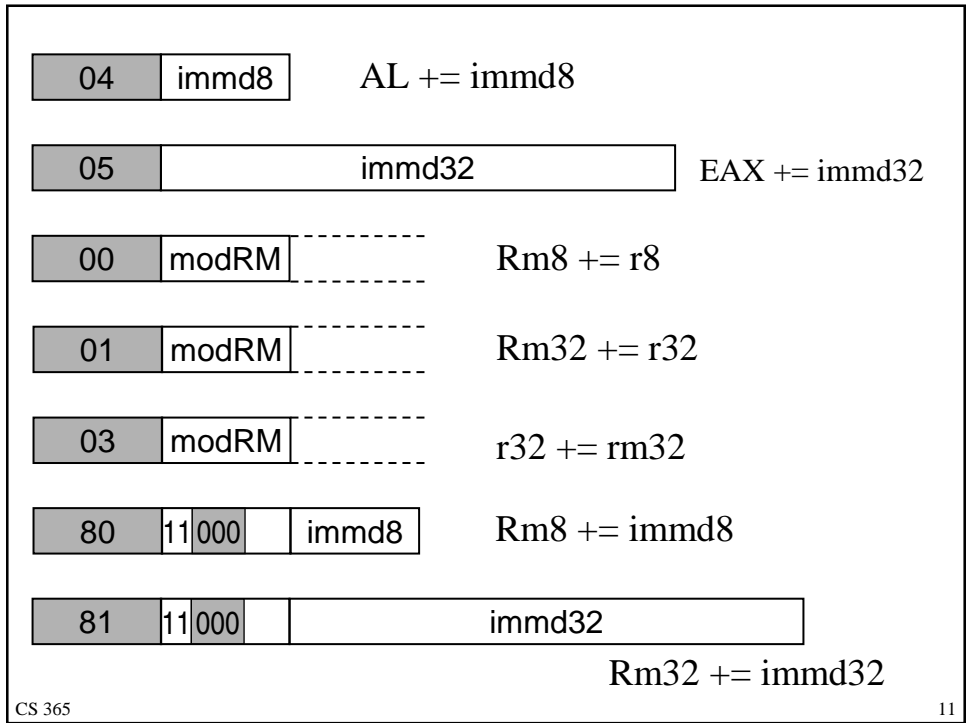   – SP cannot be an index, and

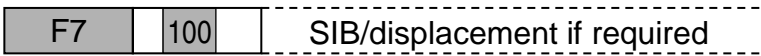   – BP cannot be a base.

Example: Add Instructions

❑ The first operand is the destination.

    – Can be register or memory

❑ The second operand is the source

    – Can be register or memory

❑ The two operands cannot be both memory.

❑ Action: dest += source

| 04 | immd8 | | AL += immd8 |

| 05 | immd32 | | EAX += immd32 |

| 00 | modRM | - - - - - - - - | Rm8 += r8 |

| 01 | modRM | - - - - - - - - | Rm32 += r32 |

| 03 | modRM | - - - - - - - - | r32 += rm32 |

| 80 | 11 000 | | immd8 | Rm8 += immd8 |

| 81 | 11 000 | | immd32 |

Rm32 += immd32

Multiplication

| F7 | | 100 | | SIB/displacement if required |

❑ Action: $EDX{:}EAX \leftarrow EAX \times Rm32$

❑ Notice that the multiplier is fixed. It must be EAX.

❑ The multiplicand can be register or memory.

Special Purpose Instructions

❑ Decimal arithmetic

❑ Strings

❑ MMX

❑ SIMD (single instruction multiple data)

MIPS versus IA32

❑ Fixed instruction formats of MIPS

– Simple decoding logic

– Waste of memory space

– Limited addressing modes

❑ Variable length formats of IA32

– Difficult to decode; sequential decoding

– Compact machine codes

– Versatile addressing modes

❑ Large pool of general purpose registers in MIPS.

– No special considerations for particular opcodes/registers; everything is born equal.

  ➢ Well, there are exceptions. Can you name one?

– Simplify programming and program optimizations

– Good for compilations

❑ Small pool of registers in IA32

– Small amount of data stored inside CPU

  ➢ Recall that moving data between CPU and memory is slow, compared to register operations.

  ➢ Usually lead to inefficient code

– Many registers serve special purposes; making programmer/compiler's job difficult

  ➢ Again could lead to inefficient code

❑ Operand architecture of MIPS

   – Uses three register operands

   – All data must be (explicitly) moved into registers before the CPU and manipulate them.

   – Results have to be explicitly stored back to memory.

   – Creates longer machine codes but reflects the reality.

❑ Operand architecture of IA32

   – One or two operands

   – Operands in some instructions are fixed and implied

      ➢ Compact code but lack flexibilities

      ➢ Makes code optimizations difficult

   – One operand can be memory

      ➢ No explicit load/stores; compact code

      ➢ Data are moved in/out of CPU anyway; no gain in performance

❑ IA32 has to be backward compatible with previous 8/16 bit architectures.

 – This contributes to its complexities, many of which unnecessarily so

 – However, Intel gets to keep its software and customer base. BIG PLUS.

 – Intel commands huge resources to push improvements.

 – The result is IA32 chips are generally on par with other modern ISAs.

❑ MIPS represents a new generation of computer architectures.

 – Called **Reduced Instruction Set Computer** (**RISC**)

 – No corpses to carry; clean designs

 – Design is purposely kept simple.

 – In theory, this shortens design cycles and produces efficient implementations.

 – In reality, you need people and money to compete with Intel. Very difficult.