Computer Systems
Lecture 10

Overview

- Structure of instructions
- Addressing modes
- Register Indirect mode
- Indexed Register Indirect with Displacement mode

Structure of instructions

- The binary codes of almost all instruction contain three pieces of information:
  - The **action** or operation of the instruction.
  - The **operands** involved (where to find the information to operate with).
  - Where the **result** is to go.

Structure of instructions (cont.)

- Machine instructions are encoded with **distinct bit fields in the prefix** to contain information about:
  - The operation required.
  - The location of operands and results.
  - The data type of the operand.
- Pentium instructions can be from 1 to 15 bytes long, depending on:
  - The operation required.
  - The addressing modes employed.

Examples of the coding structure of some instructions

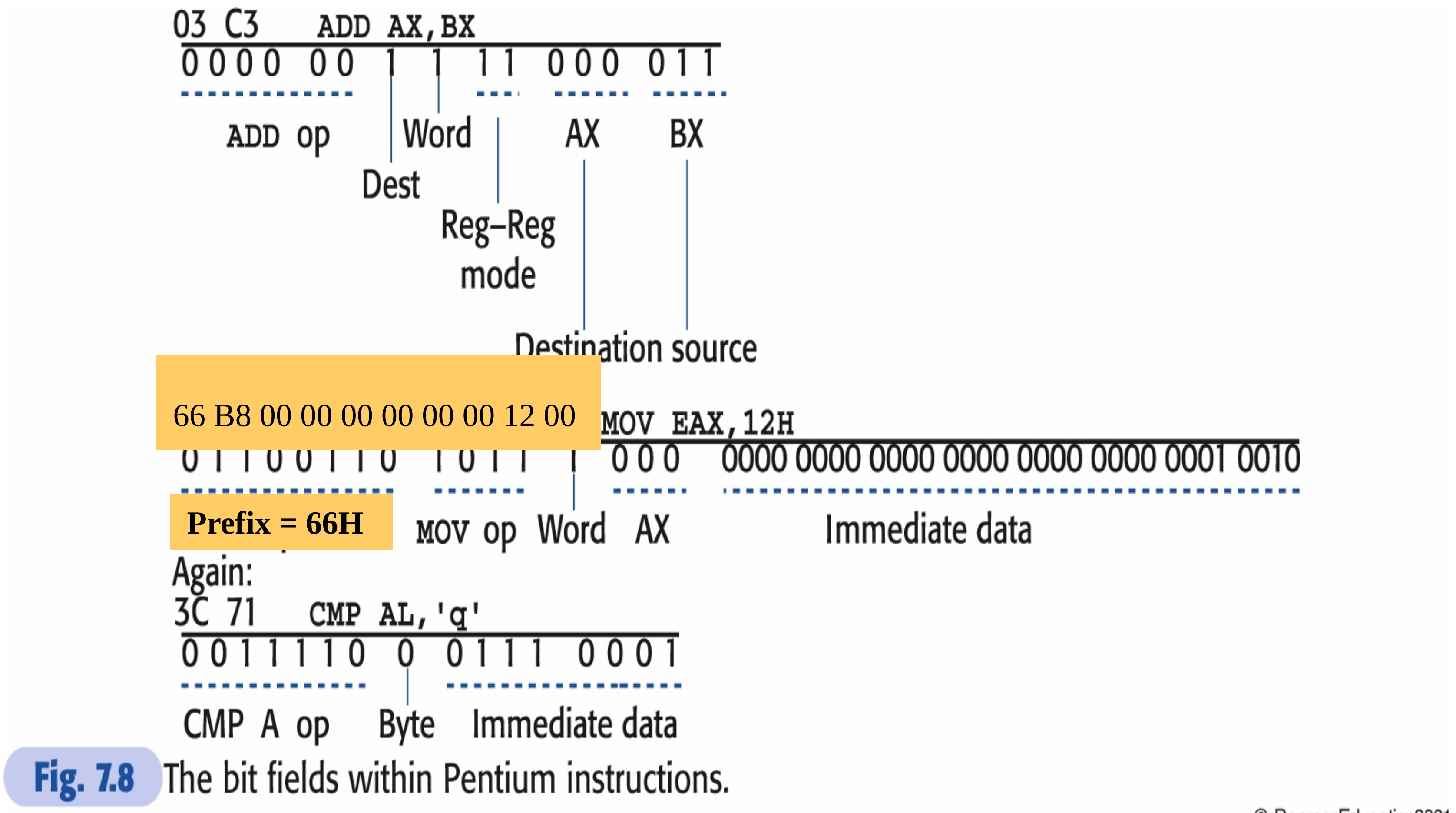

Fig. 7.8 The bit fields within Pentium instructions.

Addressing modes

- Addressing mode
  - The way of forming operand addresses.
  - More technically, the way one can compute an **effective address**.
    - Effective address is the address of operand used by instruction.
  - Offering various addressing modes support better the needs of HLLs when they need to manipulate large data structures.

Addressing modes (cont.)

- **Immediate (operand) mode.**

    Example: **`mov eax,104`**

- Part of the binary code here is the **value (= 104)** of the operand.

Addressing modes (cont.)

- **Data Register Direct:**

  Example: **`mov eax, ebx`**

- The operand is contained in registers and the instruction contains codes of the registers.
- This is the fastest to execute.

Addressing modes (cont.)

- **Memory Direct:**

  Example: **`mov eax, a`**

- Here ‘`a`’ is a variable, stored in memory and the instruction contains the **address** of this variable.
- Decode the address into a temporary register within the CPU, and then go to the memory location to read the data into the CPU.

Addressing modes (cont.)

- **Address Register Direct:**

Example: **`lea eax, message1`**

- The instruction, contains the address of '`message1`' variable, which is loaded into `eax` register after the execution of the instruction.
- Commonly used to initialise pointers which can then reference data arrays.

- What if we say: **mov eax, message1**

Addressing modes (cont.)

- **Register Indirect:**

  Example: **`mov eax, [ebx]`**

- The instruction copies to the `eax` register the content of a memory location with the address stored in `ebx`.

Register Indirect mode in Arrays

```
/* Handling arrays with  Register Indirect addressing mode */
   …
   int myarray[5];        // declaration of an array of integers
   myarray[0] = 1;        //
   myarray[1] = 3;        //
   myarray[2] = 5;        //  initialise the array
   myarray[3] = 7;        //
   myarray[4] = 9;        //
   _asm
   {
     …
     lea ebx,myarray      // address of the array (its 0th element) is saved in ebx
     mov ecx,5            // size of the array is saved in the counter
     mov eax,0            // eax will be used to store the sum, initialise to 0
loop1: add eax,[ebx]      // read an element of the array at the address stored in ebx
     add ebx,4            // int occupies 4 bytes (32 bits),
                          // so to read next element increase the address in ebx by 4
     loop loop1           // end of the loop
    }
```

Addressing modes (cont.)

- **Indexed Register Indirect with displacement:**

  Examples:

```
mov eax,[table+esi]
mov eax,table[esi]
```

- In both cases the effective address is obtained by adding the address ‘`table`’ contained in the operand field of the instruction to the content of a register (registers).
- The symbol ‘`table`’ is the address of the array base.

Alternative array handling

```
/* Handling arrays with  Register Indirect with Displacement addressing mode */
   …
   int myarray[5];        // declaration of an array of integers
   myarray[0] = 1;        //
   myarray[1] = 3;        //
   myarray[2] = 5;        // initialise the array
   myarray[3] = 7;        //
   myarray[4] = 9;        //
   _asm{
      mov ecx,5         // size of the array is saved in the counter
      mov eax,0         // eax will be used to store the sum, initialise to 0
      mov ebx,0
loop1:add eax,myarray[ebx]// read an element of the array at the
                          // address myarray + ebx
      add ebx,4                 // int occupies 4 bytes (32 bits),  so to read
                                // next element increase the address in ebx
                                // by 4
      loop loop1                // end of the loop
    }
```

Q&A

- Q. The Pentium instruction set has a fixed width for all instructions to speed up instruction decoding.

- Q. Which sample program is more efficient? (slide 12 vs slide 14)

Q&A

- What are those 3 major components covered by every instruction?
- Machine instructions are encoded with what to contain information about the operation required?
- Which of the following is the fastest to execute?
  - **`mov eax, b`**
  - **`mov a, ebx`**
  - **`mov eax, ebx`**

- Q. The following instruction will involve the calculation of ‘effective address’.

  (T or F)

- **`mov eax,table[esi]`**

- Q. After the execution of the following instruction, *eax* will contain the content of message1 inside. (T or F)
- **`lea eax, message1`**

Readings

- [Wil06] Sections 7.4, 7.6 for structure of instructions and addressing modes.
- [Wil06] Section 7.3 for basic Pentium instructions.
- Lecture Notes 11, 12