Computer Systems
Lecture 8

Overview

- Execution of instructions
- Assembly language
- Main memory, RAM
- Words, bytes and bits
- Base register
- Instruction pointer

Execution of instructions

- We have already seen the general principles of instructions execution.

- The CPU fetch-execution cycle.
- The fetch phase is the same for different types of instructions, while execute phase depends on the type of instruction.

Execution of instructions

- We will look into:
  - The types of instructions available.
  - What they can do?
  - How they can be combined to perform useful and non-trivial computations?
  - How they are used to implement HLL constructs?

Reminder: HLL and machine instruction

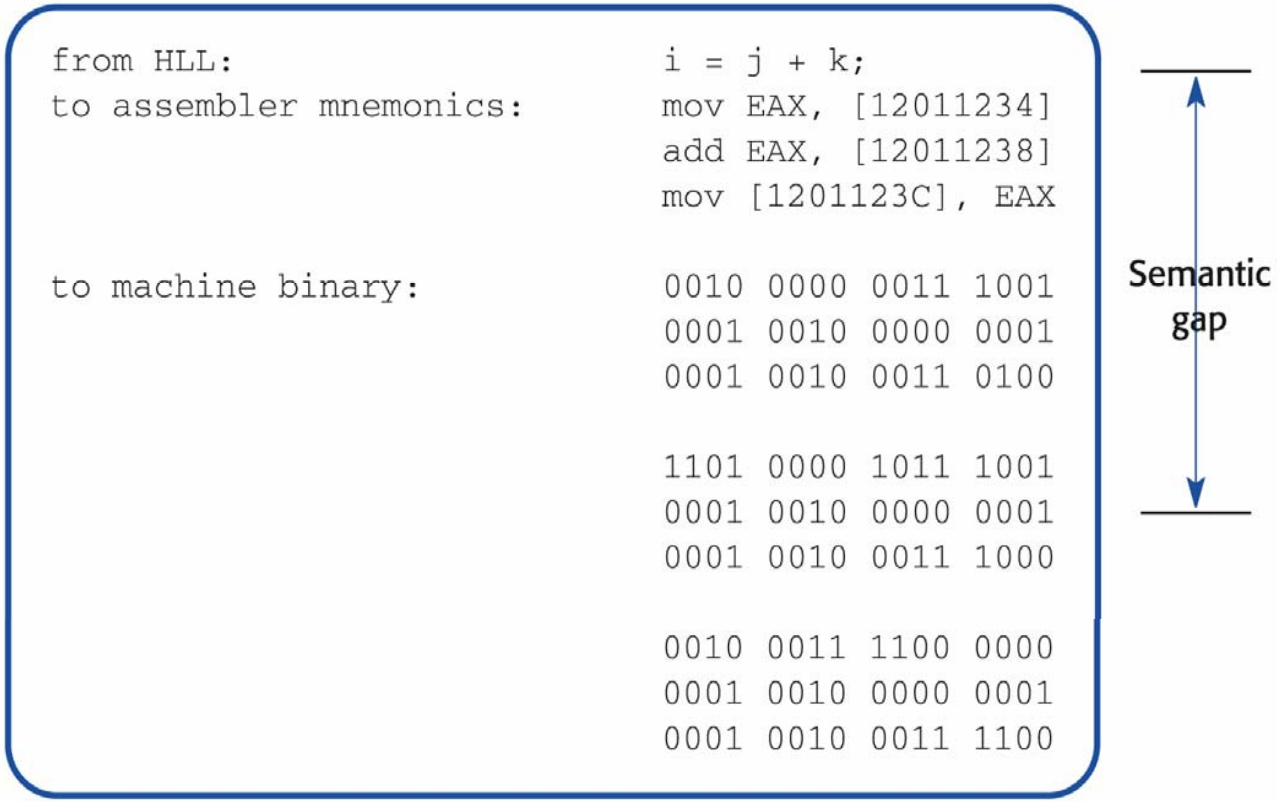


Fig. 2.3 Equivalent HLL, assembler and machine code.

CPU and instructions

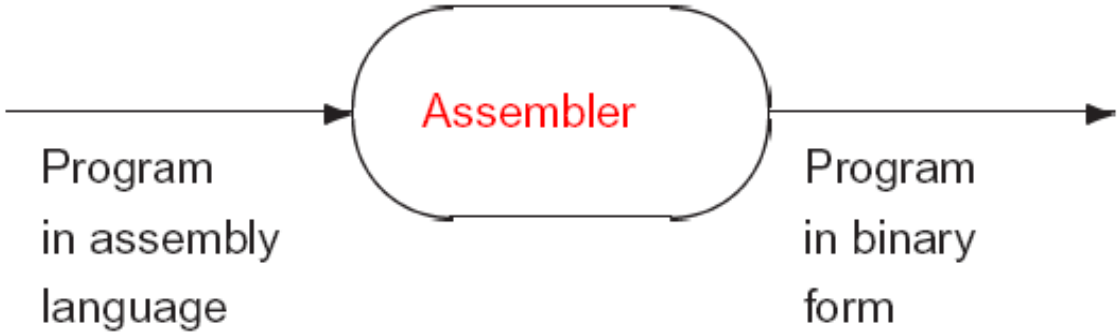
- Each different kind of CPU will be different in the set of instructions that it can perform, and in the way these are represented in binary machine code.
- General principles are similar.

- We will look into some details of **Pentium** family of CPUs.

Mnemonic form and assembly language

- In order to study more closely the way in which programs are performed, we will write some simple programs for the **Pentium** processor.
- We will write this in a mnemonic form of the Pentium machine code.
- This form is called **assembly language**.

Assembly language and assembler



- The assembler program takes as input a program written in assembly language, and:
  - Translate instructions into their binary **machine-code** form.
  - Associates labels with memory **addresses**.
  - Produces a binary **machine object code program**.

Assembly language and assembler (cont.)

- Associates **labels** with memory addresses.

```
L1:    JZ     L2
       ...
       CALL   doD
       JMP    L1
L2:    ...
```

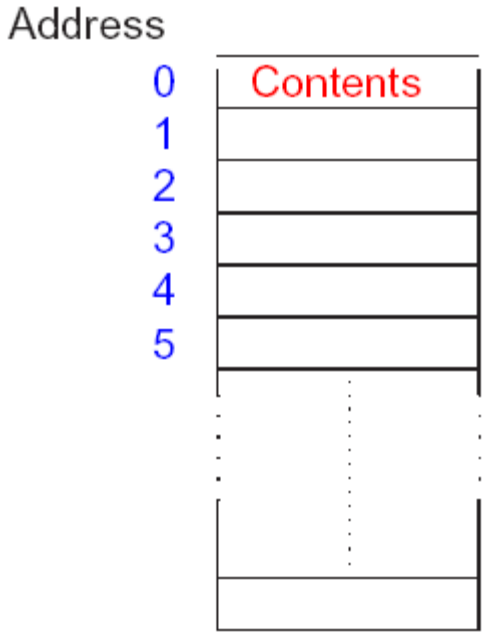
- Labels, for example, signify the beginning of a loop, the beginning of a block of memory cells, etc.
- A program has to be loaded into somewhere in main memory in order to be executed.
  - Programs must be **relocatable**.
  - What does this imply?

Assembly language and assembler (cont.)

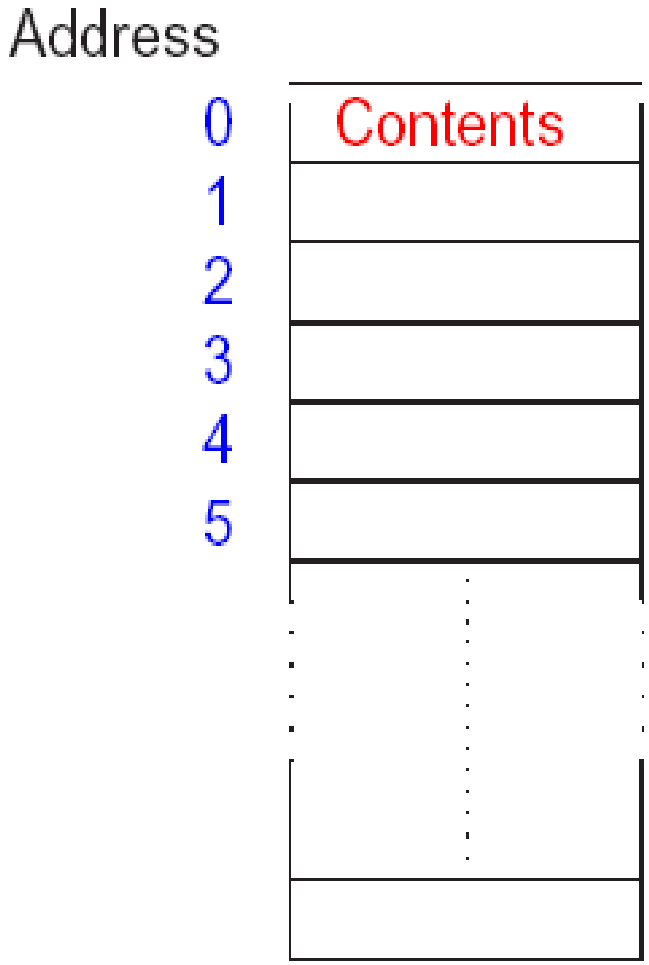
- Binary machine object code program.
  - Also called **object file**.
  - Contains binary machine code.
  - May still need a linker to generate an executable or library by linking object files together.

Main memory, RAM

- A CPU may execute only instructions loaded in the main memory (why?).
  - RAM: **Random Access Memory**.
  - The name is to set it apart from **sequential access memory** (e.g. serial tape storage) with which you cannot access data stored in the last blocks without going through the first blocks of data.



RAM



- The memory can be seen as a set of numbered storage elements, called ***words***, each of which can contain some information.
- Each word is numbered with its **address**.
- Any word of memory can be accessed “without touching” the preceding words (Random Access).
- **Access time** is the same for all the stored items.

Words, Bytes and Bits

- A **word** may be visualised as a set of more elementary storage elements (memory cells), called **bits**, arranged in a row.
- The number of bits in the word may vary between different computers; 32 bits in word is common.
- A standard unit of 8 bits called a **byte**.

CPU Registers in Pentium

- Not including the floating point registers, or the MMX (Multi-Media Extension) group.

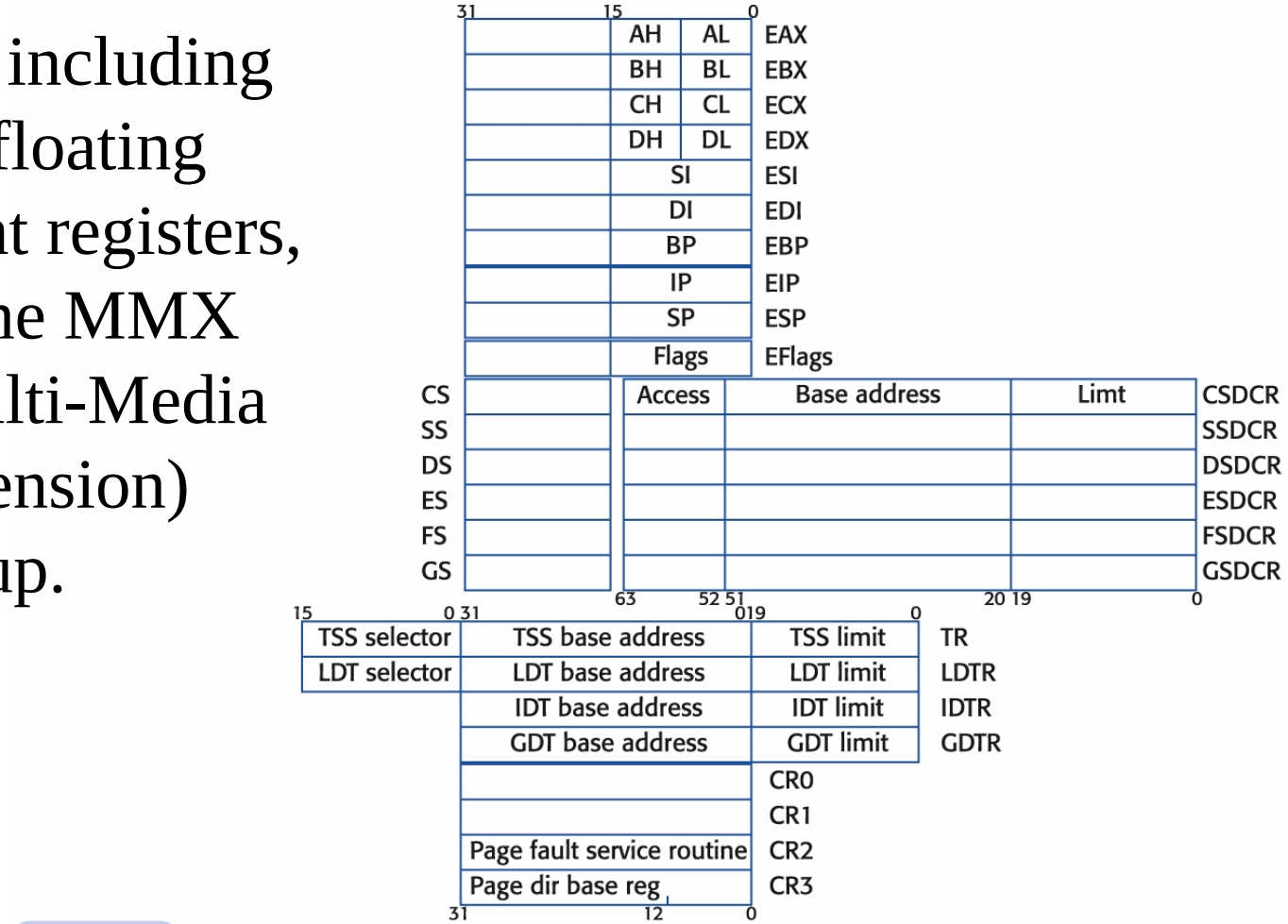


Fig. 7.5 Intel80x86/Pentium CPU register set.

EAX register, or **A**ccumulator

- Used as a general-purpose data register during arithmetic and logical operations.
- Some instructions are optimised to work faster with the accumulator.
- In some instructions (`MUL` and `DIV`) The accumulator is assumed. No need to write it explicitly.
- It can be accessed in various ways as 8,16, or 32 bit chunks, referred to as **AL, AH, AX, EAX**.

Examples of instructions using EAX

`MOV EAX,1234H`	Load constant value 4660 into 32 bit accumulator.
`INC EAX`	Add 1 to accumulator value.
`MOV maxval,EAX`	Store accumulator value to memory variable ‘maxval’.
`DIV CX`	Divide accumulator by value in 16 bit register CX.

EBX: the Base register

- The Base register can hold addresses to point to the base of data structures, such as arrays in memory.

`LEA EBX, marks`	Initialise EBX with address of the variable ‘marks’.
`MOV AL,[EBX]`	Get 1-byte value into AL using EBX as the memory pointer.

ECX: the Count register

- It is used as a counter in loops.

`MOV ECX,100`	Initialise ECX as the FOR loop index.
……	
`for1:`	Symbolic address label.
……	
`LOOP for1`	Decrement ECX, test for zero, JMP back if non-zero.

EIP: the Instruction Pointer

- The Instruction Pointer (Program Counter) holds the address of the next instruction.

`JMP mylabel`	Forces a new address into EIP.

Further registers: ESI, EDI

- ESI: the Source Index register is used as a pointer for string or array operations.

```
MOV AX,[EBX+ESI]
```

Get one 16-bit word using Base address and Index register.

- EDI: The Destination Index register is used as a pointer for string or array operations.

```
MOV EAX,[ESI]
MOV [EDI],EAX
```

Moves one 32-bit word from source to destination.

Further registers: ESP, EBP

- EBP: The Stack Base Pointer.
- ESP: The Stack Pointer.

(more details later on)

Q&A

- The ‘execute’ phase is the same for different types of instructions. (T or F?)
- The assembler program takes as input a program written in assembly language, and translate instructions into ? form.
- Assembler associates labels with ?
- A CPU may execute only instructions loaded in the main memory (T or F ?).
- Access time is the same for all the RAM cells (T or F ?).

Q&A

- Q. What is the meaning of
  MOV EBX, [EBX] ?

  What is the side effect of this instruction?

Q&A

- Q. JNZ L2 means ?
- When will there be compile time error for the above instruction?

Q&A

- Is this valid assembly code?
- MOV maxval, loc1

Readings

- [Wil06] Section 3.6 for memory.
- [Wil06] Sections 7.1, 7.2 for the registers.