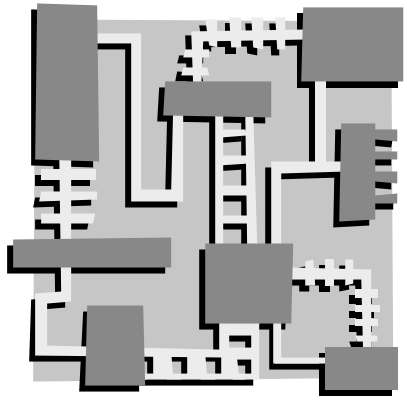
CHAPTER 1

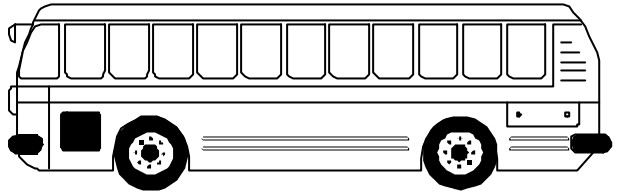
Introduction

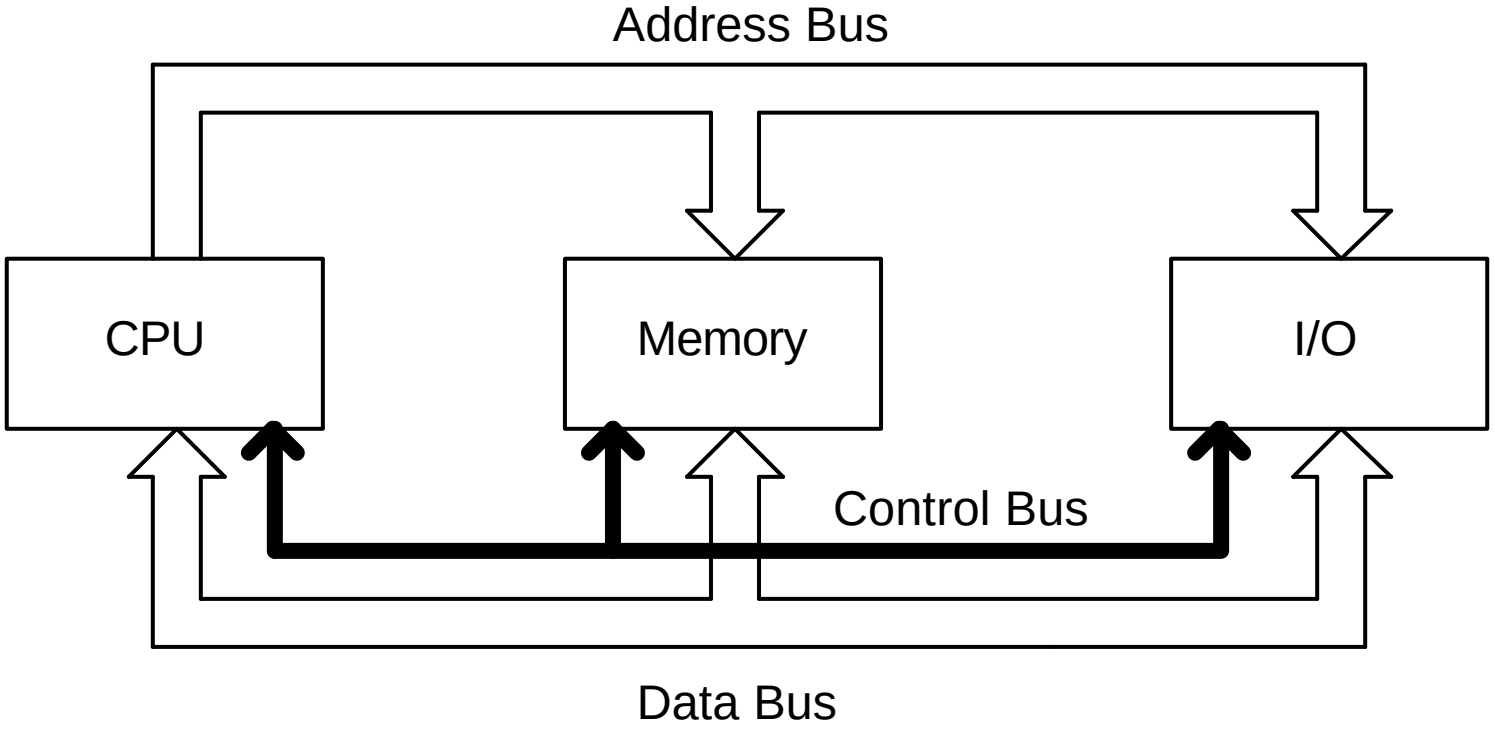
Microprocessor



- Silicon chip that contains a central processing unit (CPU).
- The “Brain” of all personal computers, most workstations, and a great number of digital devices.
- In charge of program execution.
- It can be RISC or CISC.

Bus Connections





Bus Connections (continued)

- A processor communicates with the system's memory and I/O circuits by means of signals that travel through a set of cables or connections known as buses.
  - **Address Bus:** Holds the memory address that will be accessed.
  - **Data Bus:** Holds the piece of data to read or write.
  - **Control Bus:** Indicates the operation to be done (read or write).

CPU Instructions

- Each instruction has:
  - an **opcode** (operation code), that indicates which operation to perform.
  - zero o more **operands**, which may be registers, constants or memory locations.

Fetch-Execute Cycle

- **Fetch:**
  1. Fetch an instruction from memory.
  2. Decode the instruction to determine the operation.
  3. Fetch data from memory if necessary.
- **Execute:**
  4. Perform the operation on the data.
  5. Store the result in memory if needed.

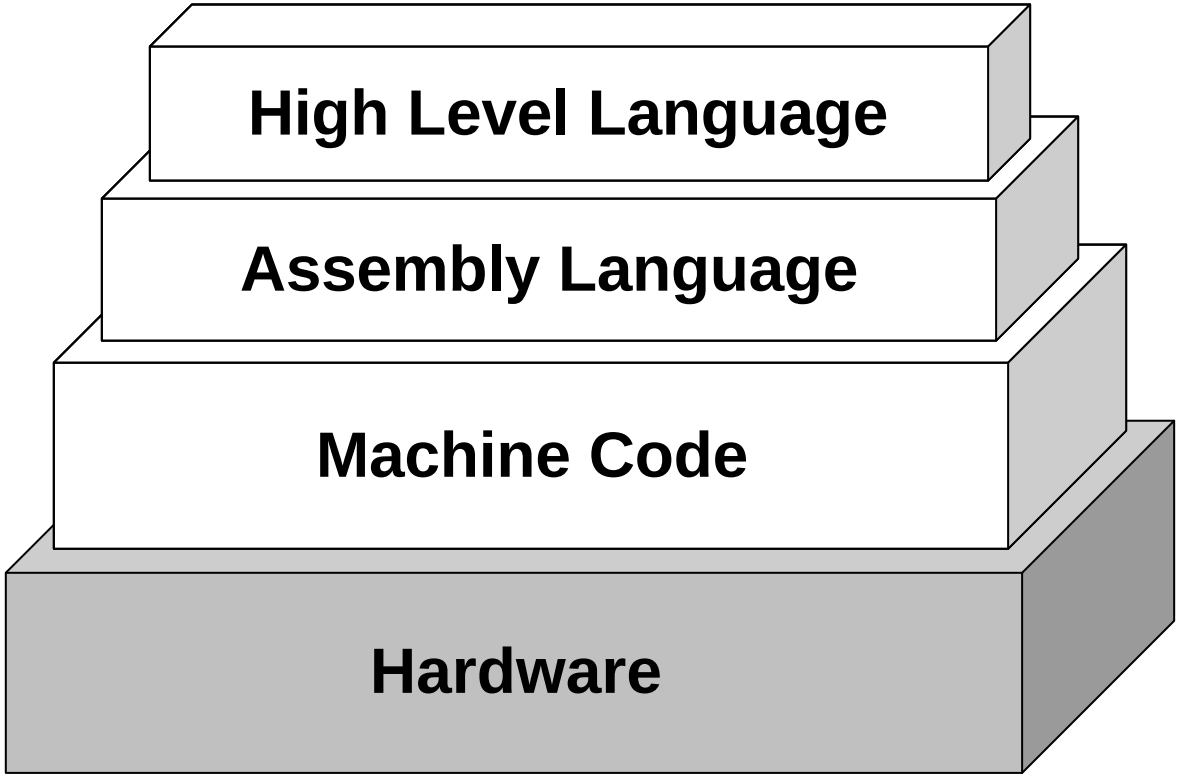
RISC: *Reduced Instruction Set Computer*

- Microprocessor that uses a relatively small number of fast but simple instructions.
- Cheaper to design and produce because they require less transistors.
- Mainly used in workstations.

CISC: *Complex Instruction Set Computer*

- Microprocessor that uses a significantly large amount of complex (specialized) instructions.
- Mainly used for Intel’s x86 architecture.

Programming Languages



Machine Code

- Lowest level programming language.
- Each CPU instruction is represented as an opcode, which is an unsigned integer number.
- Only language that the computer really understands.
- Difficult to understand by human beings.

Machine Code Example

- The *opcode* for adding one to the accumulator in the Intel x86 is:

  `01000000b`

  or

  `0x40`

Assembly Language

- Same instruction set as machine code.
- Each *opcode* is replaced by a symbolic name.
- Less cryptic for human beings.

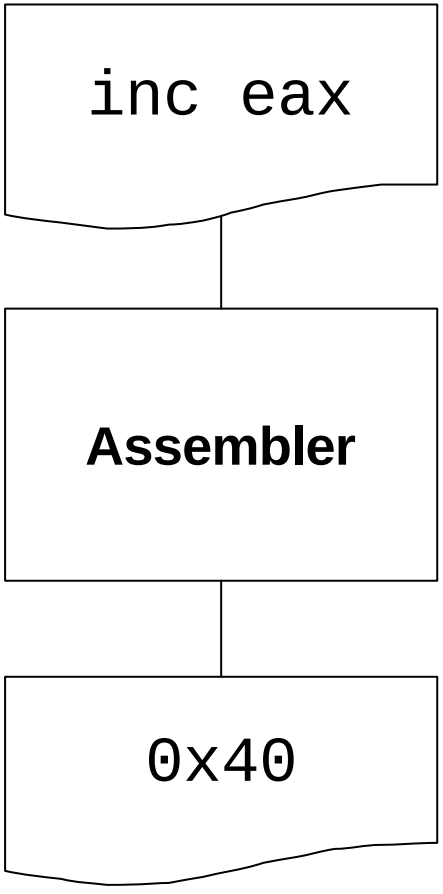
Assembly Language Example

- The Intel x86 assembly language instruction that adds one to the accumulator is:

```
inc eax
```

Assembler

- In order to execute a program written in assembly language, it first has to be translated to machine code using a special program called an **assembler**.

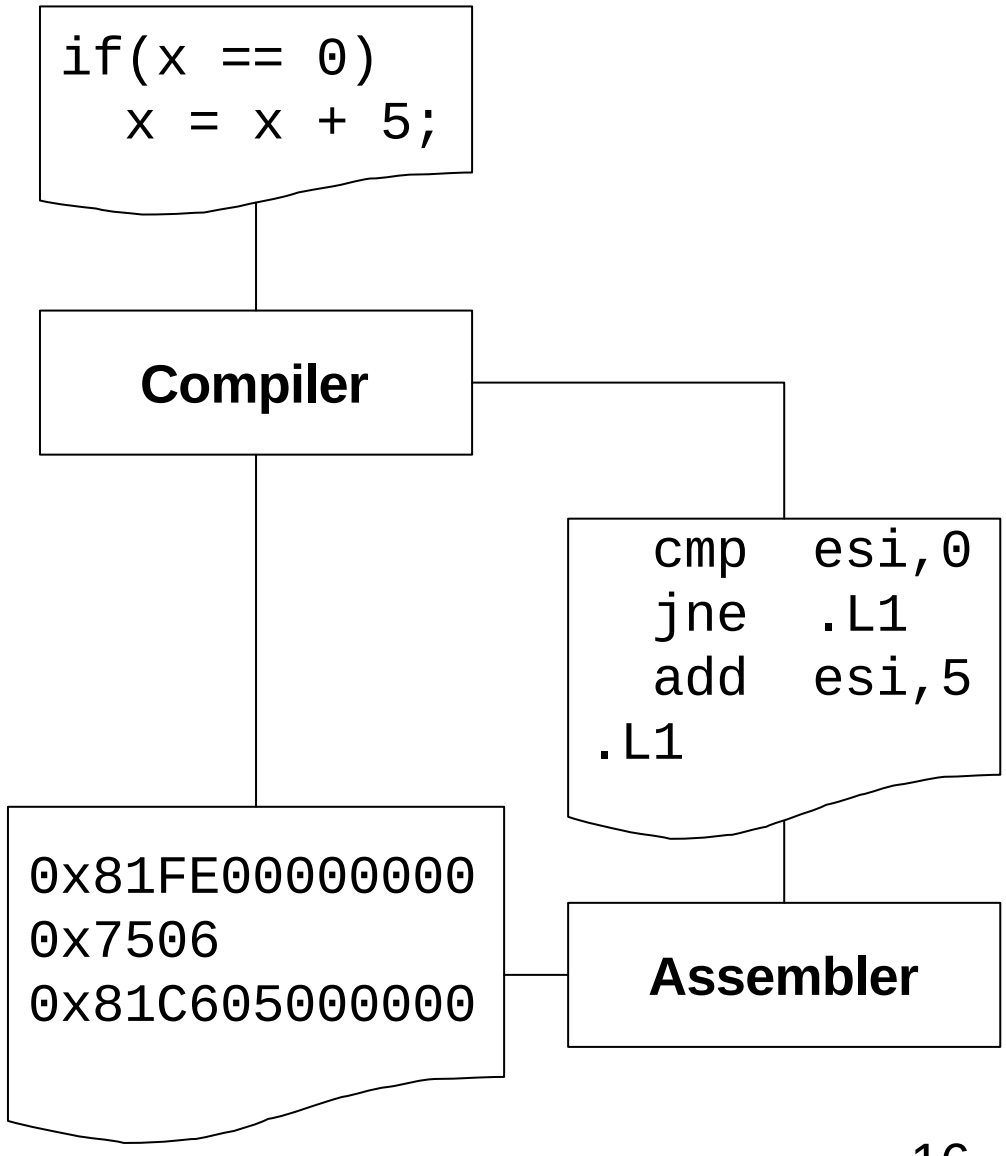


High Level Language

- Has less primitive instructions than assembly language and machine code.
- Program text is much more like natural language.
- Easier to understand by human beings.
- Examples: FORTRAN, LISP, COBOL, BASIC and C.

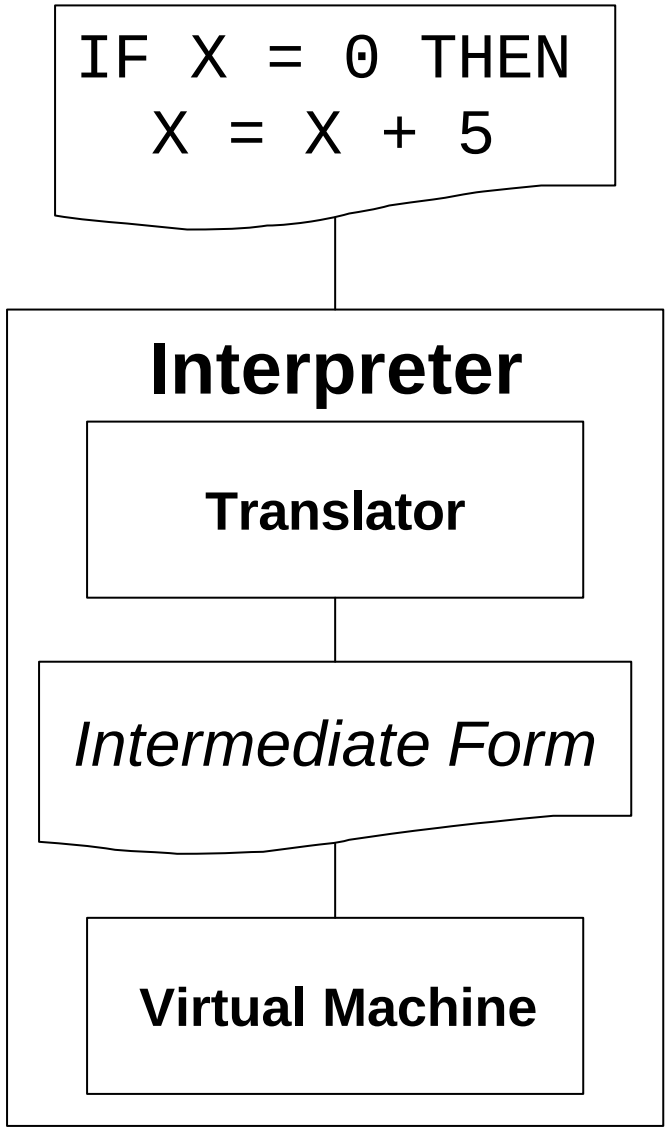
Compiler

- A program written in a high level language may be translated to machine code using a **compiler**.

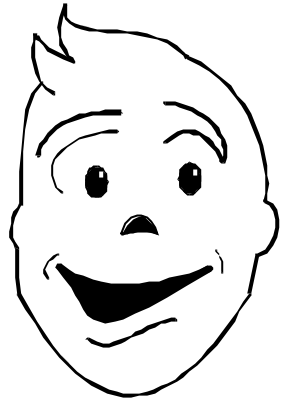


Interpreter

- An **interpreter** translates a high level language program to an intermediate form that is subsequently executed by a *virtual machine*.

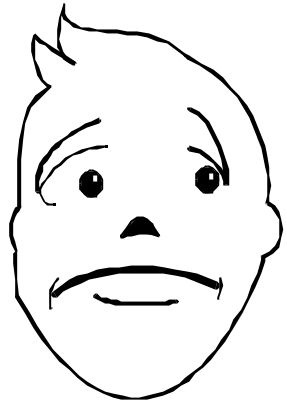


Assembly Language Advantages



- Program execution speed.
- Executable code size.
- “Bare bones” programming:
  - special instructions (FPU, MMX)
  - I/O ports
  - special CPU modes of operation

Assembly Language Disadvantages



- Error prone.
- Long and tedious to write.
- Difficult to understand and modify.
- Strongly tied to a specific computer architecture.

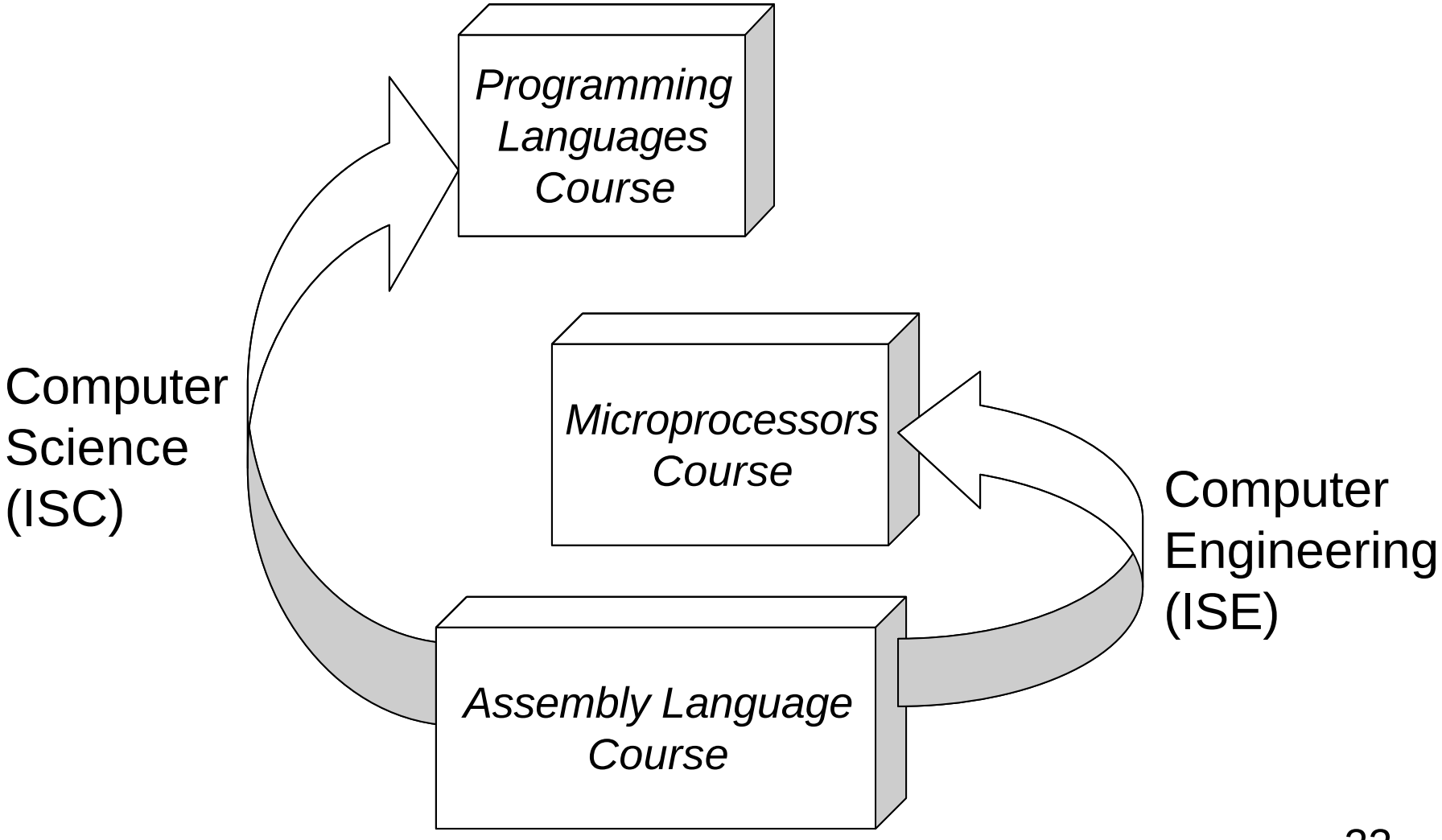
Commonly Used Assembly Language Applications

- Operating Systems
- Device Drivers
- Communication Software
- Real Time Systems
- Embedded Systems
- Graphics

Reasons for Studying Assembly Language

- To understand some of the low level details of how a real computer operates.
- To get to know some technologies that can only be adequately understood using assembly language.
- To obtain a better appreciation of the inner-workings of a compiler.

What's next?

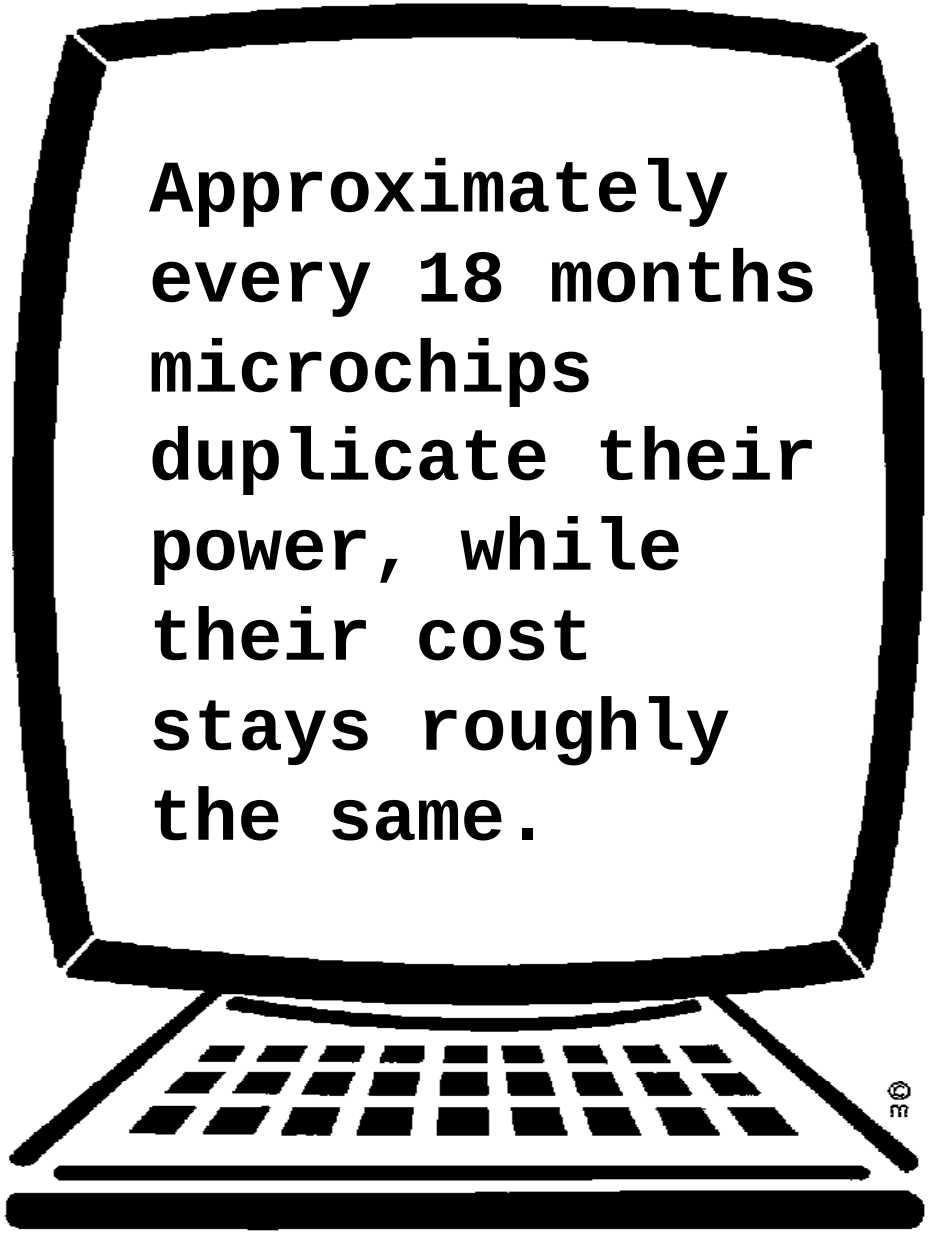

Programming Languages Course
Microprocessors Course
Assembly Language Course
Computer Science (ISC)
Computer Engineering (ISE)

CHAPTER 2

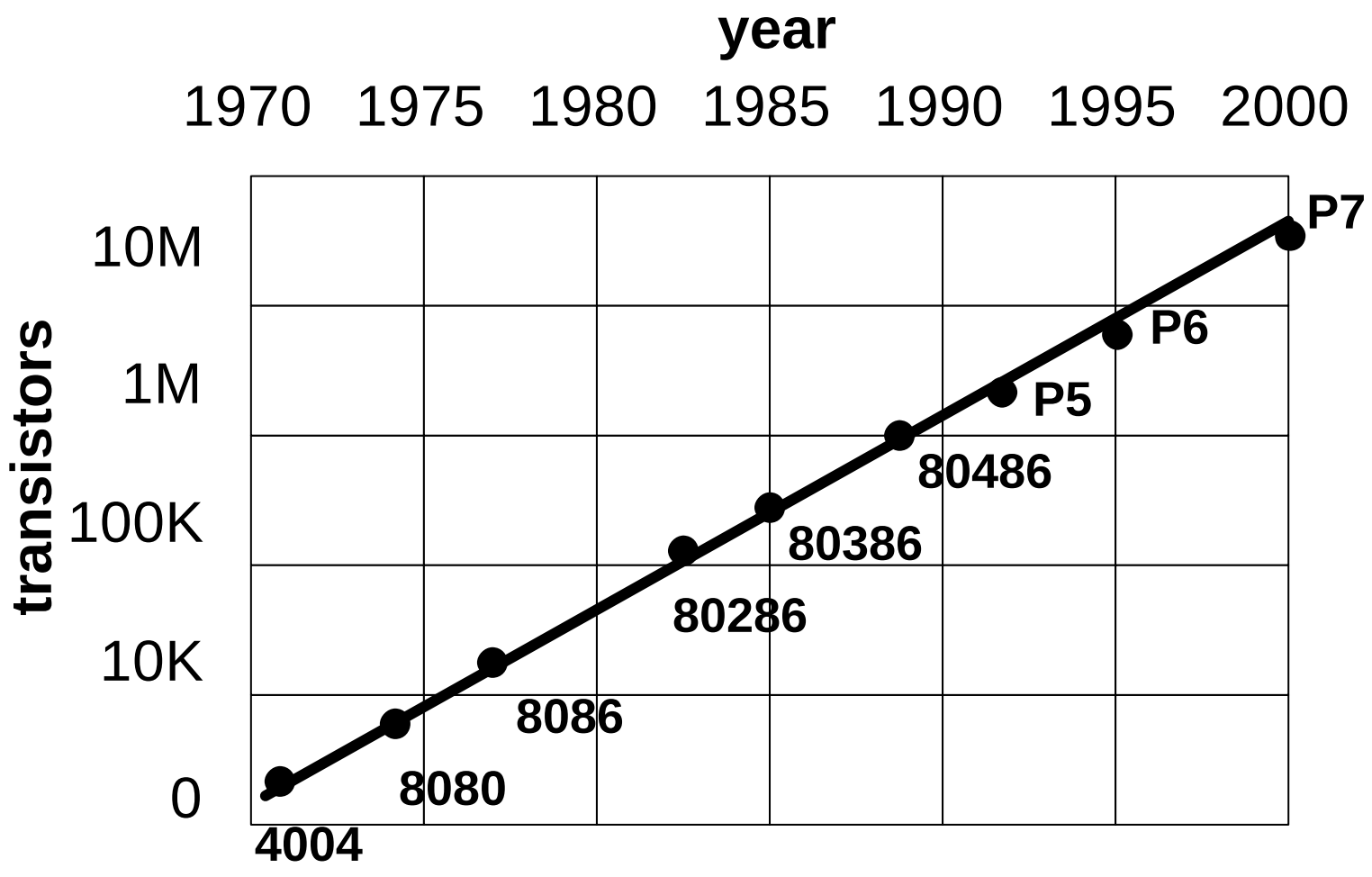
The Intel x86 Architecture

Moore's Law

- In 1965, Intel's co-founder Gordon Moore, made the following observation:

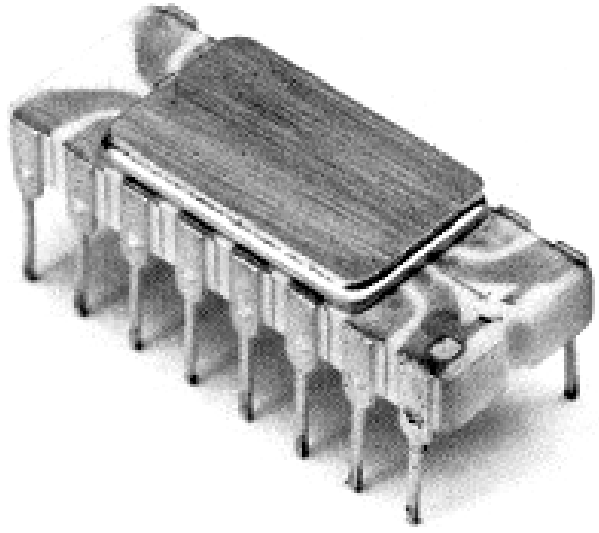


Intel Processors



4004 (1971)

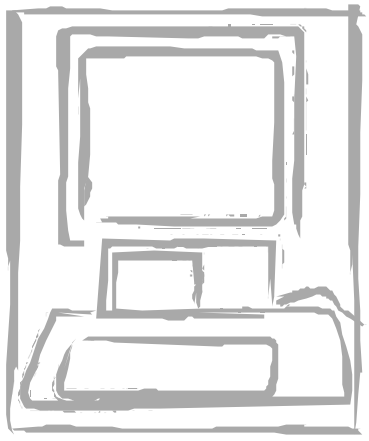
- First microprocessor.
- Built by Intel for Busicom calculators.
- 4-bit registers.
- 108 kHz.
- 2,300 transistors.



8080 (1974)

- Used in the MITS Altair 8800, the first commercial personal computer.
- 8-bit registers.
- 16-bit address bus.
- 2 MHz.
- 6,000 transistors.

8086/8088 (1978)



- Used in the original IBM PC.
- First 16-bit microprocessor.
- 20-bit address bus.
- 16-bit (8086) and 8-bit (8088) data bus.
- 4.77+ MHz.
- 29,000 transistors.

80286 (1982)

- Used in the original IBM PC/AT.
- 24-bit address bus.
- 16-bit data bus.
- 6+ MHz.
- 134,000 transistors.
- Multitasking, protected mode and virtual memory.

80386 (1985)

- 32-bit registers.
- 32-bit address bus.
- 32-bit data bus.
- Pipelining.
- 16+ MHz.
- 275,000 transistors.

P4: 80486 (1989)

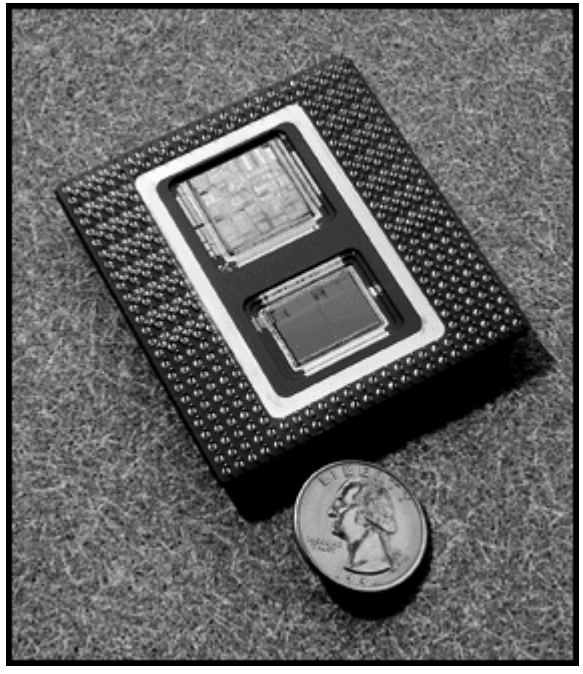
- Better execution speed.
- Integrated floating point unit (FPU).
- 8 KB L1 cache.
- 25+ MHz.
- 1’200,000 transistors.

P5: Pentium (1993)

- 64-bit data bus.
- 8 KB L1 cache for data and 8 KB for code.
- Dual pipeline for integer operations.
- 60+ MHz.
- 3’100,000 transistors.

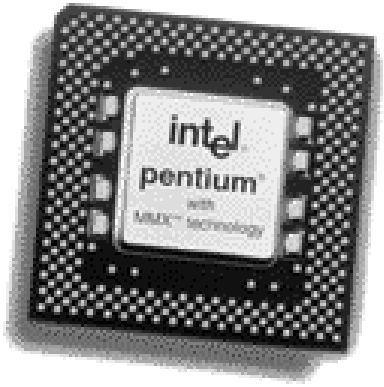
P6: Pentium Pro (1995)

- 36-bit address bus.
- 256 KB L2 cache.
- Superpipelining.
- Speculative and out of order execution.
- 150+ MHz.
- 5’500,000 transistors.



P55C: Pentium MMX (1997)

- Classic Pentium with MMX technology: 64-bit SIMD multimedia and communication extensions.
- 16 KB L1 cache for data and 16 KB for code.
- 166+ MHz.
- 4’500,000 transistors.



Klamath: Pentium II (1997)

- Pentium Pro with MMX technology.
- 16 KB L1 cache for data and 16 KB for code.
- 512 KB L2 cache.
- 233+ MHz.
- 7’500,000 transistors.

New P6 processors

- Pentium II Xeon (“Pentium II on steroids”)
  - L2 cache runs at full processor speed.
  - Designed for the computer server market.
- Celeron (“the Castrated One”)
  - Pentium II with no L2 cache.
  - Designed for the sub-$1,000 PC market.

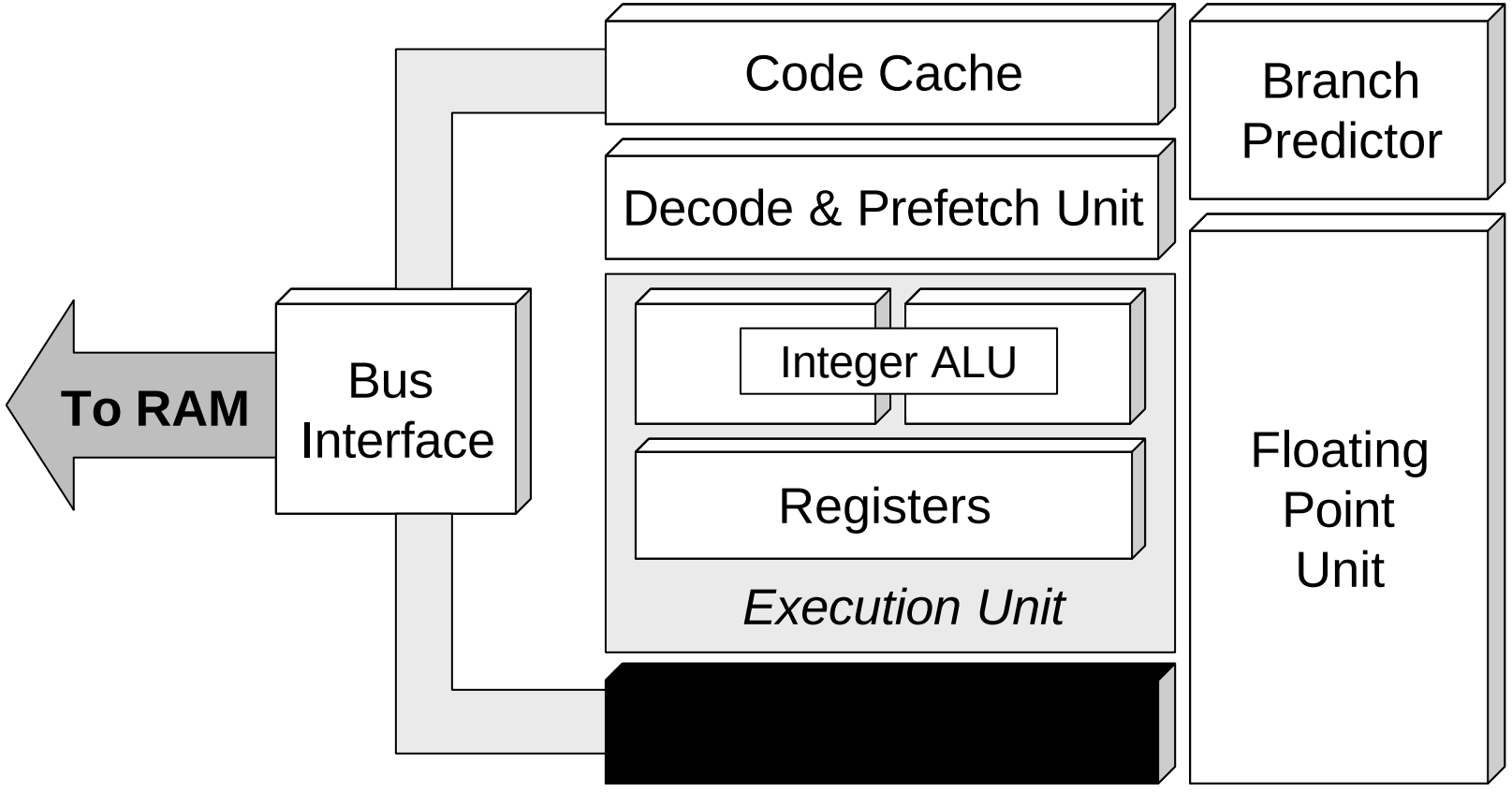
Katmai: Pentium III (1999)

- Pentium II with 128-bit SIMD floating point oriented extension to the MMX technology.
- Processor serial number in order to “enhance security”.
- 450+ MHz.

Merced: Itanium (2000)

- Intel Architecture-64 (IA-64).
- Developed jointly by Intel and Hewlett-Packard.
- Hardware x86 emulation.
- Not RISC or CISC, but EPIC (*Explicitly Parallel Instruction Computing*).
- 600 MHz and 1,000 MHz.
- Tens of millions of transistors.

x86 Basic Structure



x86 Basic Structure (continued)

- **Execution unit:** two parallel integer pipelines enable the CPU to read, interpret, execute and dispatch two instructions simultaneously.
- **Branch Predictor:** The branch prediction unit tries to guess which sequence will be executed each time the program contains a conditional jump, so that the Prefetch and Decode Unit can get the instructions ready in advance.

x86 Basic Structure (continued)

- **Floating Point Unit:** Third execution unit, where non-integer calculations are performed.
- **Primary Cache:** Two on-chip caches, one for code and one for data, are far quicker than the external memory.
- **Bus Interface:** This brings a mixture of code and data into the CPU, separates the two ready for use, and then recombines them and sends them back out.

x86 Modes of Operation

- The **operating mode** determines which instructions and architectural features are accessible.
- The Intel Architecture supports three operating modes:
  - Real Mode
  - Protected Mode
  - Virtual-8086 Mode

Real Mode

- Mode in which all x86 processors boot.
- The CPU works like a very fast 8086.
- Can only access up to 1 MB of memory.
- Only one task is executed at a time.

Protected Mode

- Allows multitasking.
- Each program has its own memory protected from other programs.
- Extended memory: more than 1 MB of memory available.
- Supports virtual memory.

Virtual-8086 Mode

- Allows simultaneous execution of two or more programs designed to work in real mode, each program having up to 1 MB of independent memory.

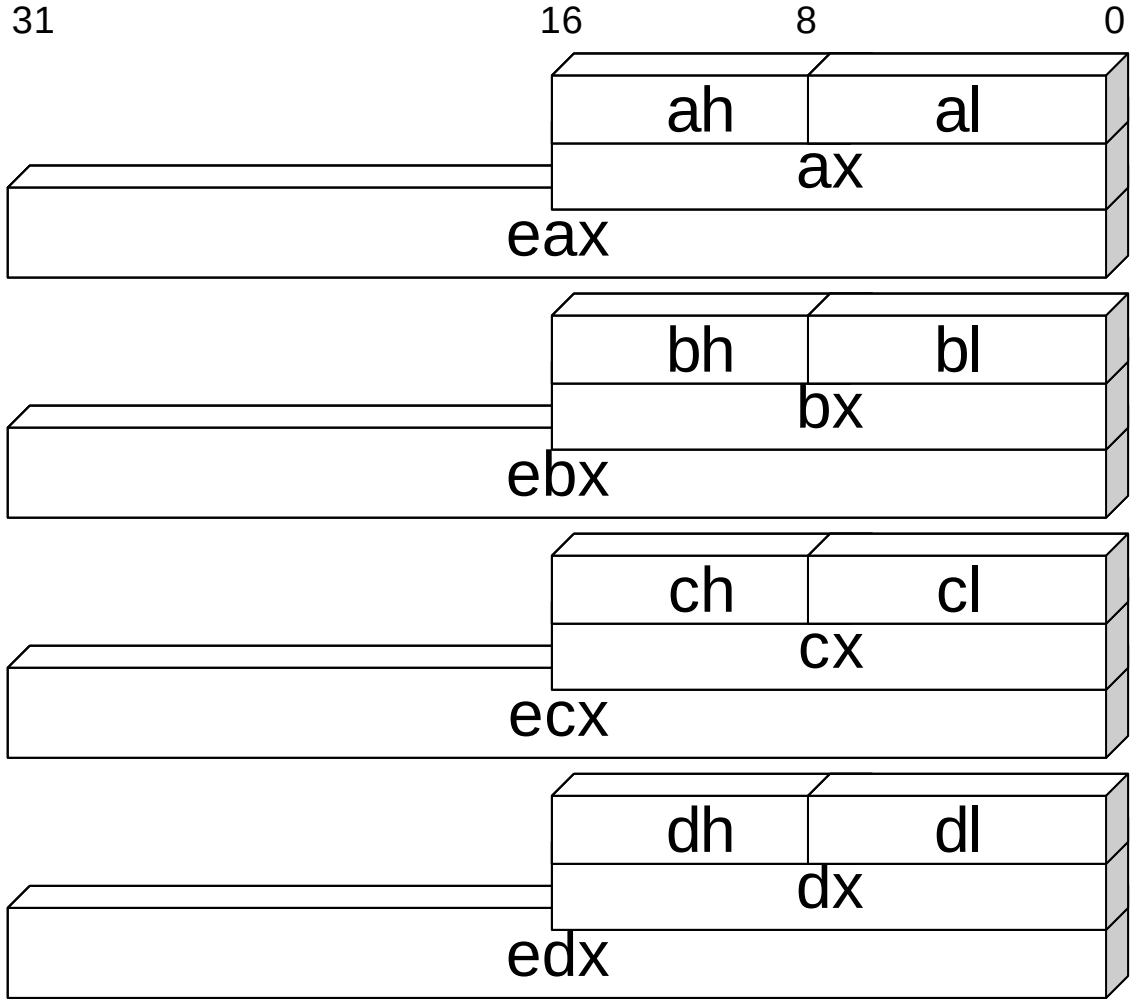
Registers

- A **register** is a special high-speed storage area within the CPU.
- The x86 processors have several registers available for the application programmer, grouped as follows:
  - General-purpose data registers.
  - Segment registers.
  - Status and control registers (EIP and EFLAGS registers).

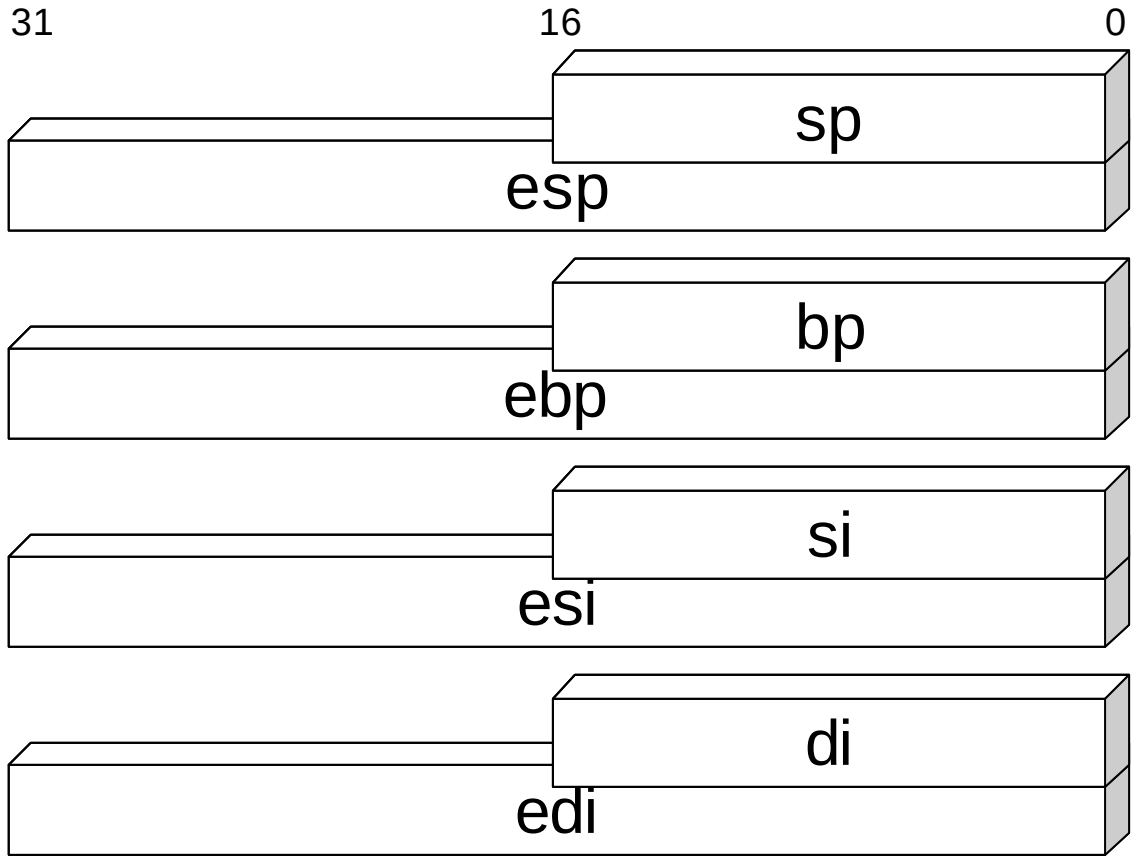
General-Purpose Data Registers

- These eight 32-bit registers are available for holding the following data items:
  - Integer operands for logical and arithmetic operations.
  - Pointers (memory addresses).

General-Purpose Data Registers (continued)



General-Purpose Data Registers (continued)



Stack Pointer

Base Pointer

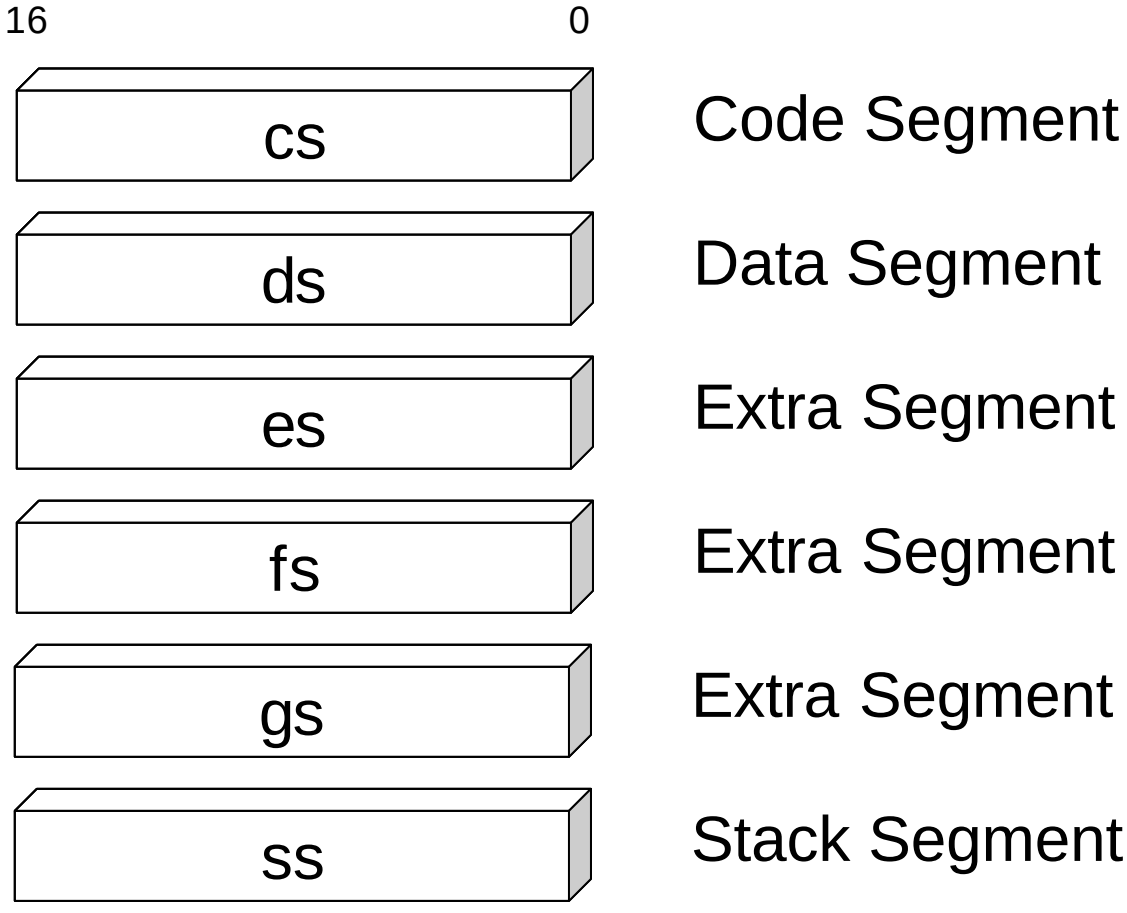
Source Index

Destination Index

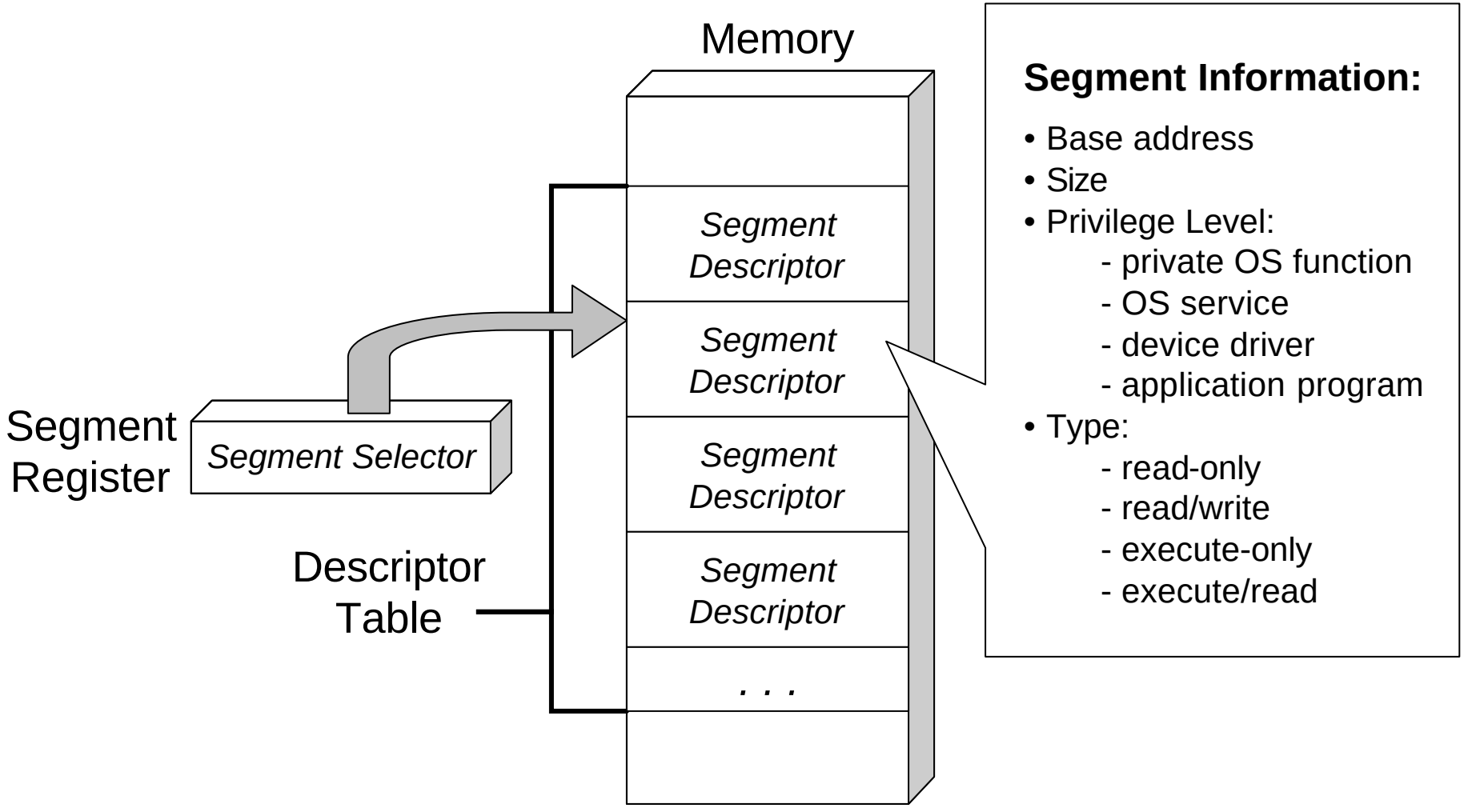
Segment Registers

- The six segment registers hold 16-bit segment selectors.
- A **segment selector** points to a special structure in memory called a segment descriptor. Several segment descriptors are grouped together into a **descriptor table**.
- A **segment descriptor** contains addressing and control information which is used to control how a 32-bit linear address is generated.

Segment Registers (continued)

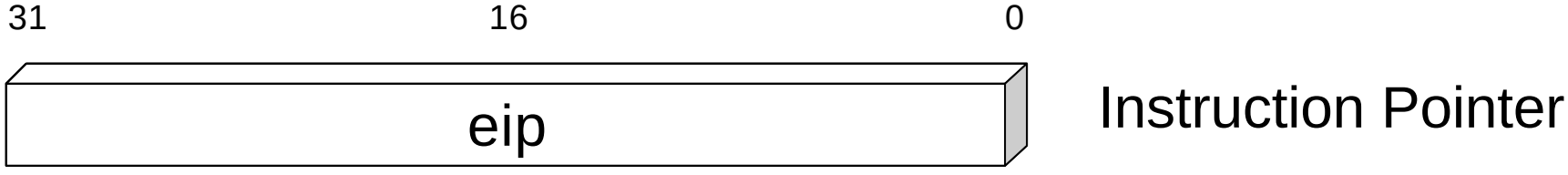


Segment Registers (continued)



Instruction Pointer Register

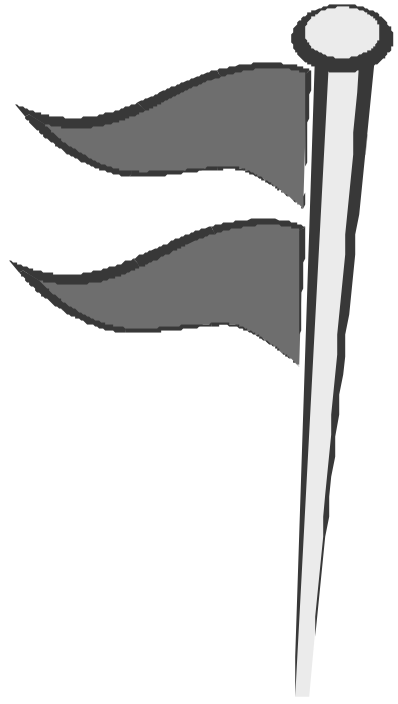
- The instruction pointer (EIP) is a 32-bit register that contains the offset in the current code segment for the next instruction to be executed.



Instruction Pointer Register (continued)

- It is advanced from one instruction boundary to the next in straight-line code or it is moved ahead or backwards by a number of instructions when executing flow control instructions such as jumps or subroutine calls.
- It cannot be accessed directly by software.

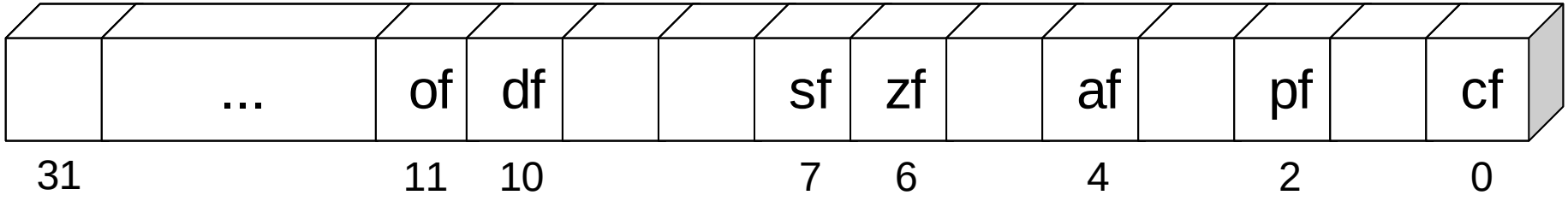
Flags Register



- This 32-bit register is a collection of individual status and control bits called *flags*.
- Each flag is usually manipulated independently and not as a set.

Flags Register (continued)

eflags



- CF carry flag
- PF parity flag
- AF auxiliary flag
- ZF Zero Flag
- SF sign flag
- DF direction flag
- OF overflow flag

Flags Register (continued)

- **Carry Flag** Is set if the result of an arithmetic operation involving unsigned numbers overflows.
- **Overflow Flag** Is set if the result of an arithmetic operation involving signed numbers overflows.
- **Sign Flag** Is set if the result of an arithmetic or logical operation is negative.
- **Zero Flag** Is set if the result of an arithmetic or logical operation is zero.

Flags Register (continued)

- **Parity Flag** Is set if the result of an arithmetic or logical operation has an even number of 1 bits in its 8 least significant bits.
- **Auxiliary Flag** Is set if the result of an arithmetic operation has a carry out from the low-order nibble. Used in binary-coded decimal (BCD) operations.
- **Direction Flag** Is explicitly set or cleared by the programmer in order to modify the behavior of some special string operations.

Memory Organization

- The memory that the processor addresses on its bus is called **physical memory**.
- Physical memory is organized as a sequence of 8-bit bytes. Each byte is assigned a unique address, called a **physical address**.

Memory Organization (continued)

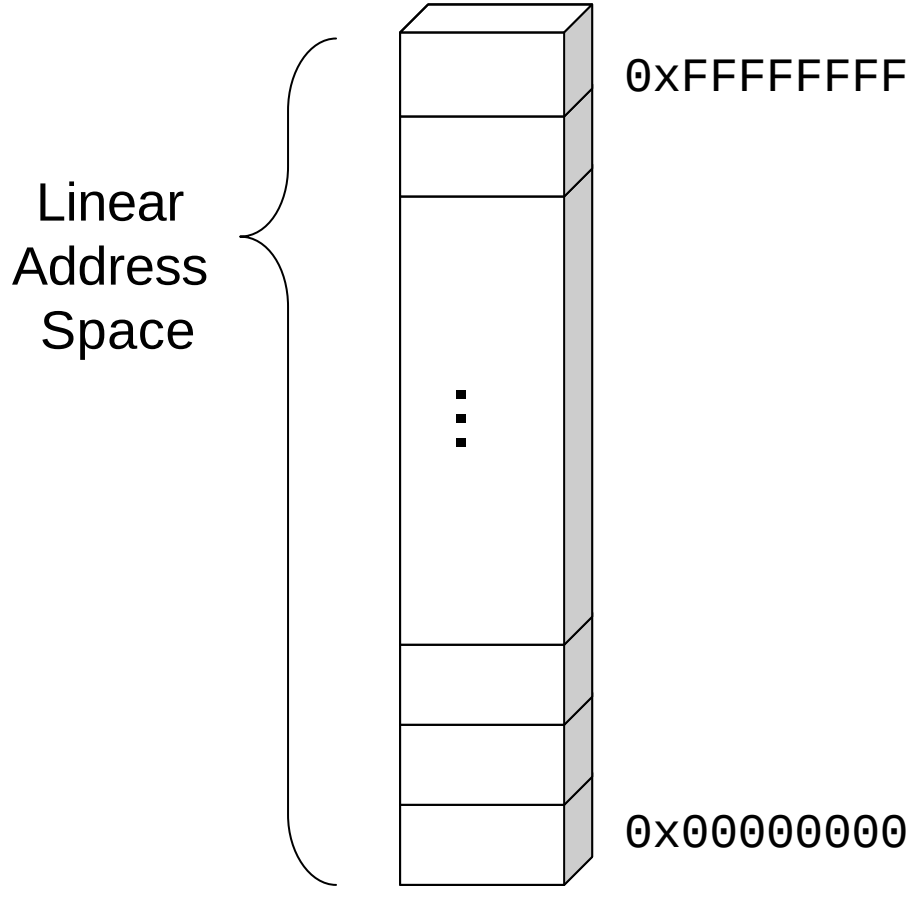
- The physical address space ranges from zero to a maximum of $2^{32} - 1$ (4 GB).
- When employing the processor's memory management facilities, programs **DO NOT** directly address physical memory. Instead, they access memory using a **memory model**.

Flat Memory Model

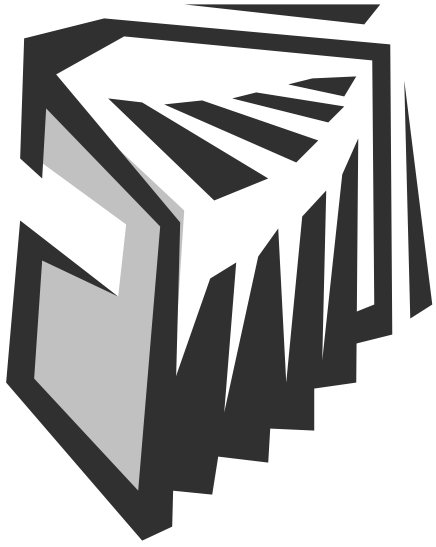
- Memory appears to a program as a single, continuous address space, called a **linear address space**. All code and data are contained in this address space.

Flat Memory Model (continued)

- The linear address space is byte addressable, with addresses running contiguously from 0 to $2^{32} - 1$.

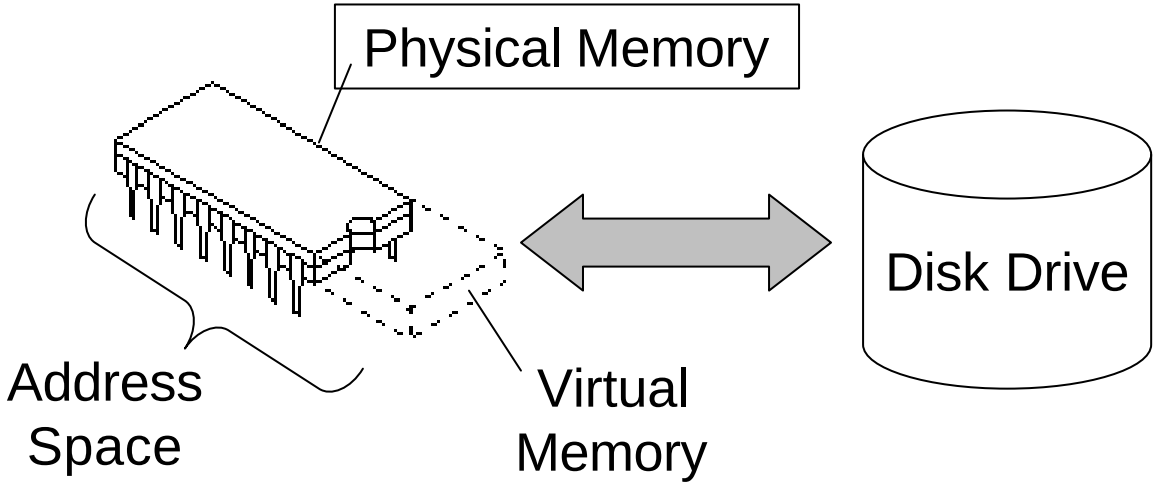


Paging



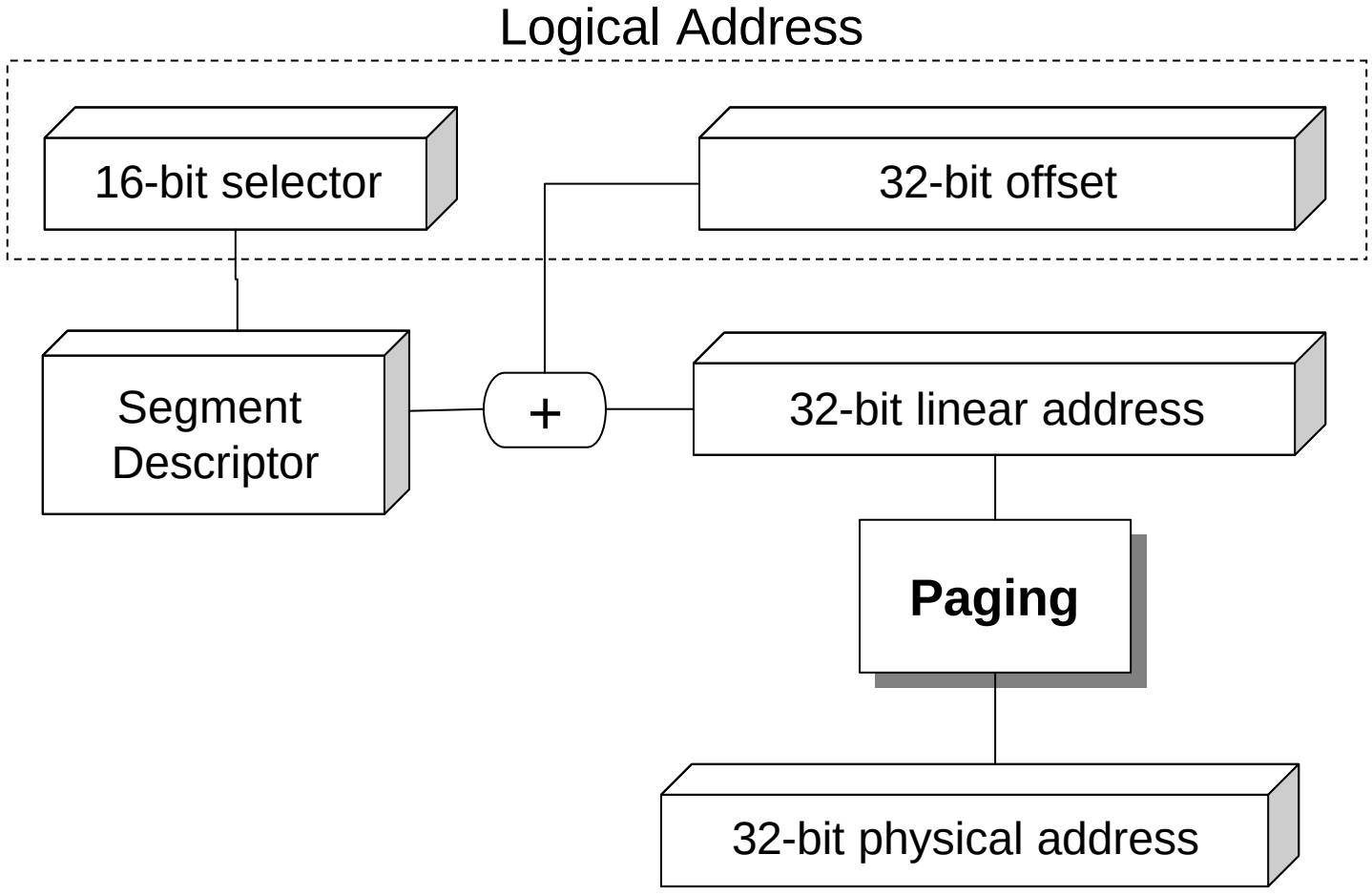
- The x86 supports translation of **linear (virtual) addresses** into physical addresses through **paging**.
- Special tables map portions of the virtual addresses into physical memory locations.
- Physical memory is divided into **page frames**, each 4 KB in size.
- The operating system copies a certain number of pages from your storage device to main memory.

Paging (continued)

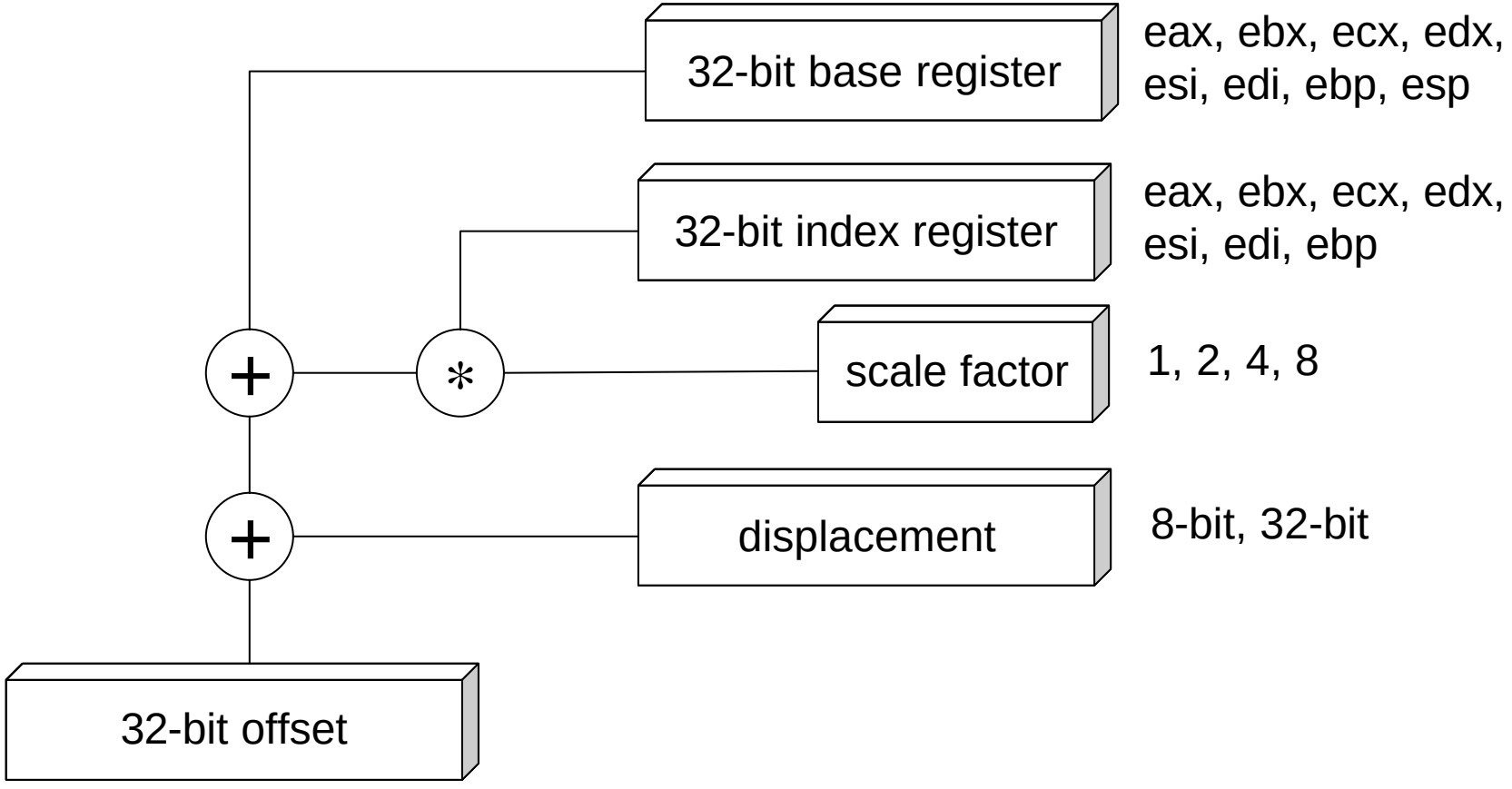


- When a program needs a page that is not in main memory, the operating system copies the required page into memory and copies another page back to the disk.
- Each time a page is needed that is not currently in memory, a **page fault** occurs.

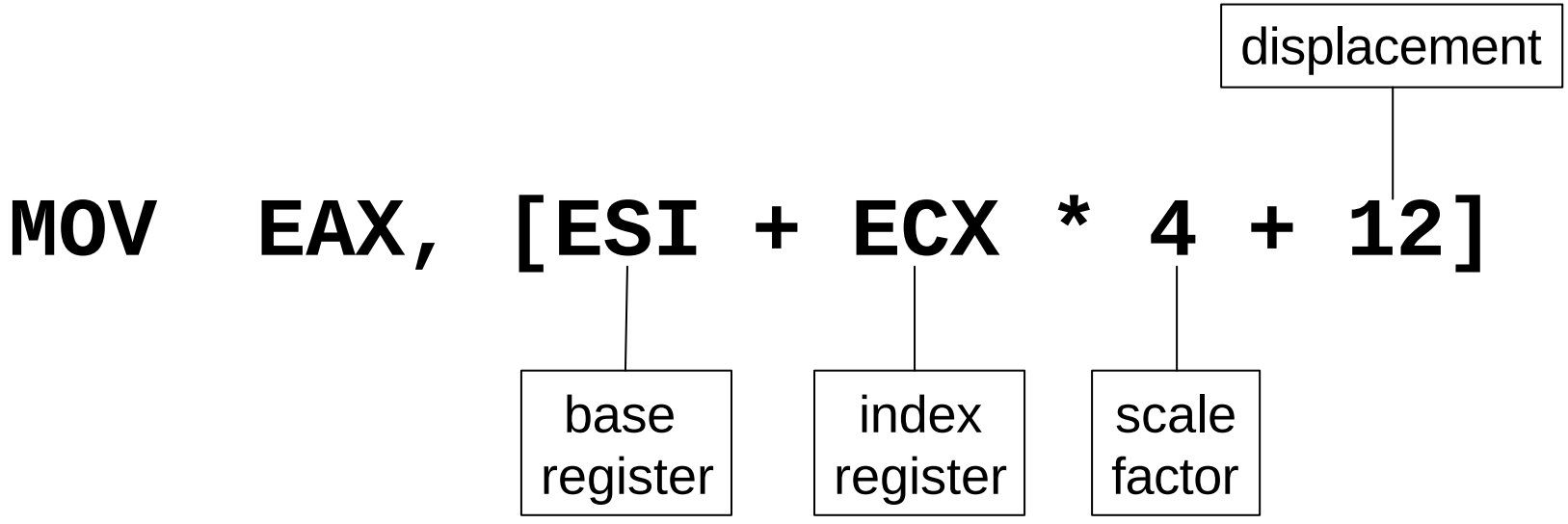
Generating a Physical Address



32-bit Offset

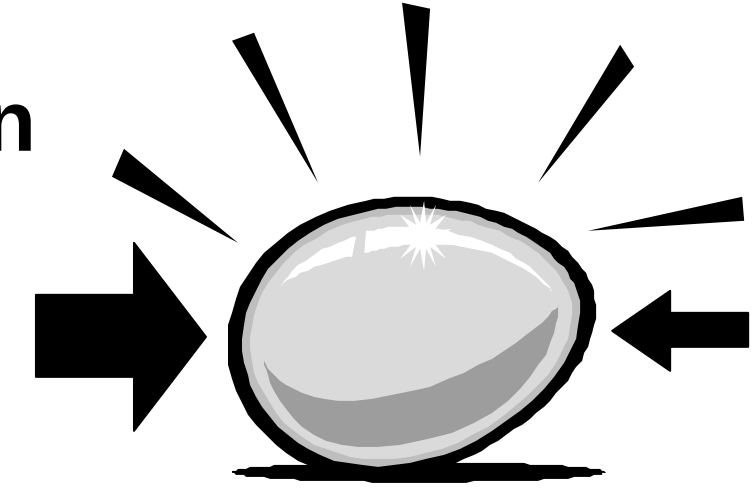


32-bit Offset Example



Byte Order

- When a value is stored in memory in multiple bytes, two distinct **byte orders** may be used:
  - **Big-Endian**
  - **Little-Endian**



Byte Order (continued)

- In big-endian architectures, the leftmost bytes (those with a lower address) are most significant. In little-endian architectures, the rightmost bytes are most significant.
- The terms *big-endian* and *little-endian* are derived from the Lilliputians of Jonathan Swift's *Gulliver's Travels*, whose major political issue was whether soft-boiled eggs should be opened on the big side or the little side.

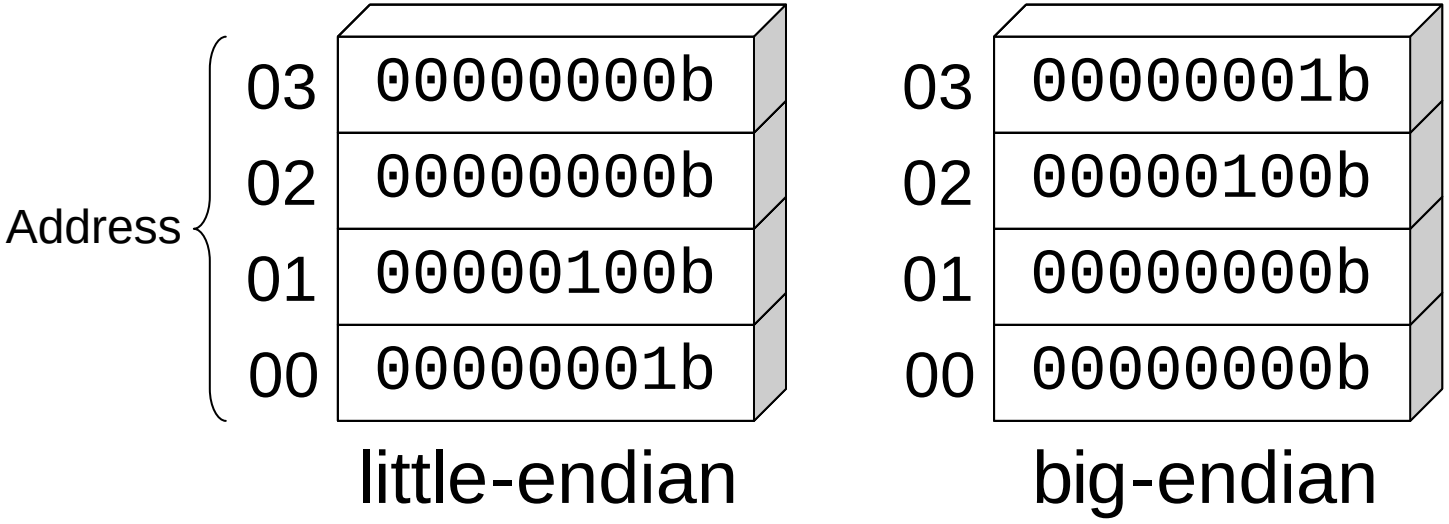
Byte Order (continued)

- Intel x86 and DEC VAX systems store multibyte values in little-endian order.
- HP, IBM and Motorola 68K systems store multibyte values in big-endian order.
- The Power PC is a bi-endian processor: it supports both big and little-endian byte ordering.

Byte Order Example

- The byte ordering for the number 1025 stored in 4 bytes is:

1025 = 00000000 00000000 00000100 00000001b

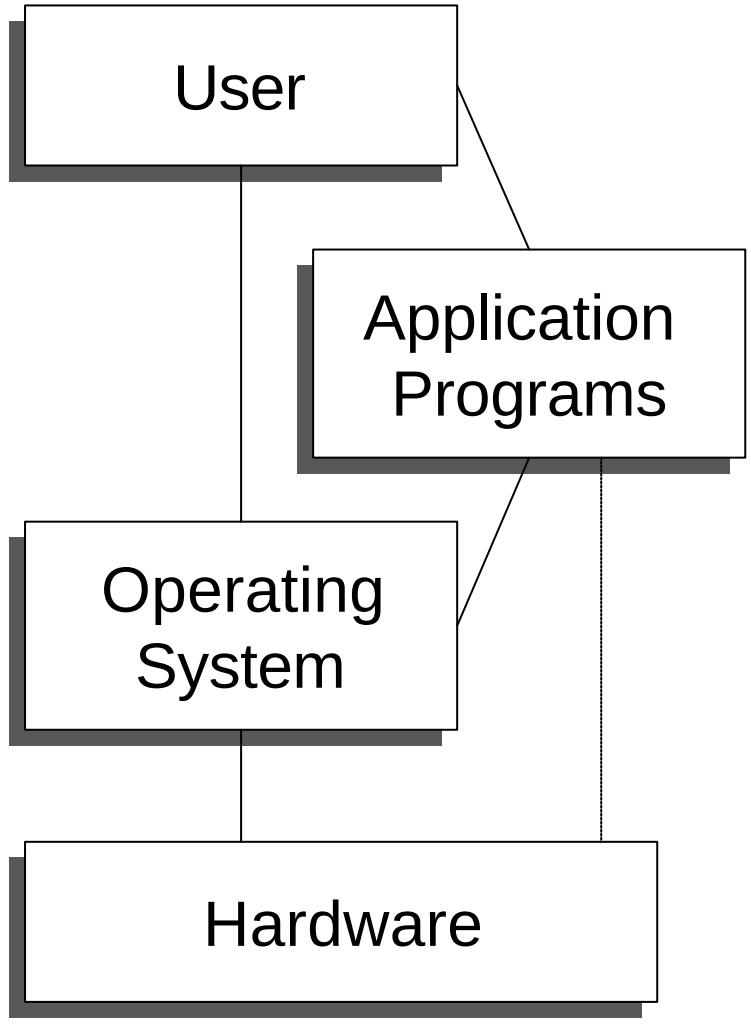


CHAPTER 3

The Linux Operating System

Operating System

- Software that makes hardware usable.
- Manages such things as: memory, screen display, keyboard input, disk files and printer output.



UNIX

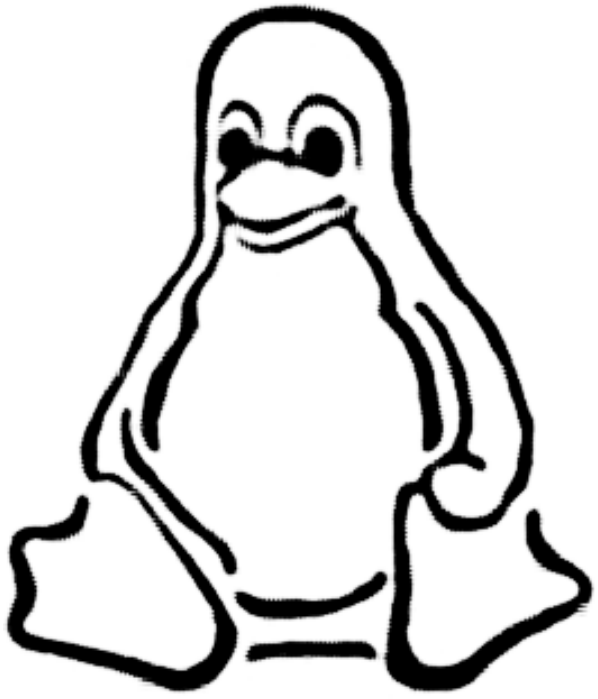
- Operating system developed at Bell Labs in the early 1970s by Ken Thompson and Dennis Ritchie.
- First operating system to be written in a high-level programming language, namely C.

UNIX (continued)

- The name UNIX was intended as a pun on a previous OS called MULTICS (and was written UNICS at first: ***UN****iplexed* ***I****nformation and* ***C****omputing* ***S****ystem*).
- Leading operating system for workstations

Linux

- Free UNIX-type operating system originally created by Linus Torvalds at the University of Helsinki in Finland.
- Developed under the GNU General Public License, the source code for Linux is freely available to everyone.



Linux (continued)

- Linux is an independent POSIX implementation and includes: multitasking, multi-user, multiprocessing, virtual memory, shared libraries and TCP/IP networking.
- Currently implemented in a wide range of platforms, including: x86, Alpha, SPARC, 68K and PowerPC.

GNU Project

- Short for GNU's Not UNIX.
- A UNIX-compatible software system developed by the Free Software Foundation (FSF).
- The philosophy behind GNU is to produce software that is non-proprietary. Anyone can download, modify and redistribute GNU software. The only restriction is that they cannot limit further redistribution.
- The GNU project was started in 1983 by Richard Stallman at the MIT.

POSIX

- Acronym for ***P****ortable* ***O****perating* ***S****ystem* ***I****nterface for UNI****X***.
- Set of IEEE and ISO standards that define an interface between programs and operating systems.
- Supported by most UNIX systems and Windows NT.

Multitasking

- The ability to execute more than one **task** (program) at the same time.
- The CPU switches from one program to another so quickly that it gives the appearance of executing all of the programs at the same time.

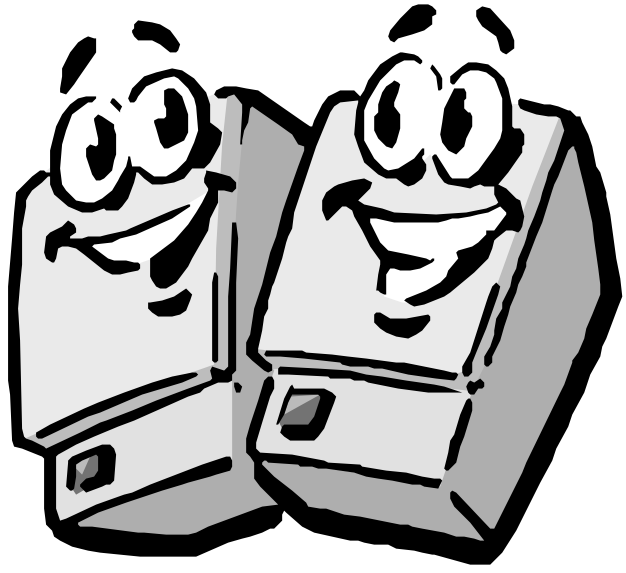
Multitasking (continued)

- There are two basic types of multitasking:
  - **Preemptive multitasking:** the operating system assigns CPU time slices to each program.
  - **Cooperative multitasking:** each program can control the CPU for as long as it needs it. If a program is not using the CPU, however, it can allow another program to use it temporarily.
- Linux supports preemptive multitasking.

Multi-user

- Computer systems that support two or more simultaneous users.
- All mainframes and minicomputers and most workstations are multi-user systems.

Multiprocessing



- Since version 2.0, Linux has the ability to run in multiprocessor architectures.
- The OS can distribute several applications in true parallel fashion across several CPUs.

Virtual Memory

If it’s there and you can see it – it’s real

If it’s not there and you can see it – it’s virtual

If it’s there and you can’t see it – it’s transparent

If it’s not there and you can’t see it – you erased it!

IBM poster explaining virtual memory,
circa 1978.

Virtual Memory (continued)

- Technique that allows to increases the amount of apparent memory available on a system.
- A **swap space** is an area on disk in which the OS stores images of running programs when memory is tight.
- The Linux virtual memory system uses a swap space to implement paging.

Shared Libraries

- A **library** is a collection of precompiled routines that a program can use.
- In a **static library**, all library functions that a program requires are made part of an executable, which can make it rather large.
- In a **shared library**, function code is not directly included in an executable file. Instead, the OS **dynamically links** a running program to the required routines contained in the shared library.

Shared Libraries (continued)

- Shared libraries have two important advantages:
  - Small executable files.
  - Several programs running at the same time can share a single copy of the library code.

TCP/IP Networking

- Acronym for ***T****ransmission* ***C****ontrol* ***P****rotocol/****I****nternet* ***P****rotocol*.
- Consists of a suite of communications protocols used to connect hosts on the Internet.
- Allows services such as: *e-mail*, *telnet*, *ftp* and *http*.

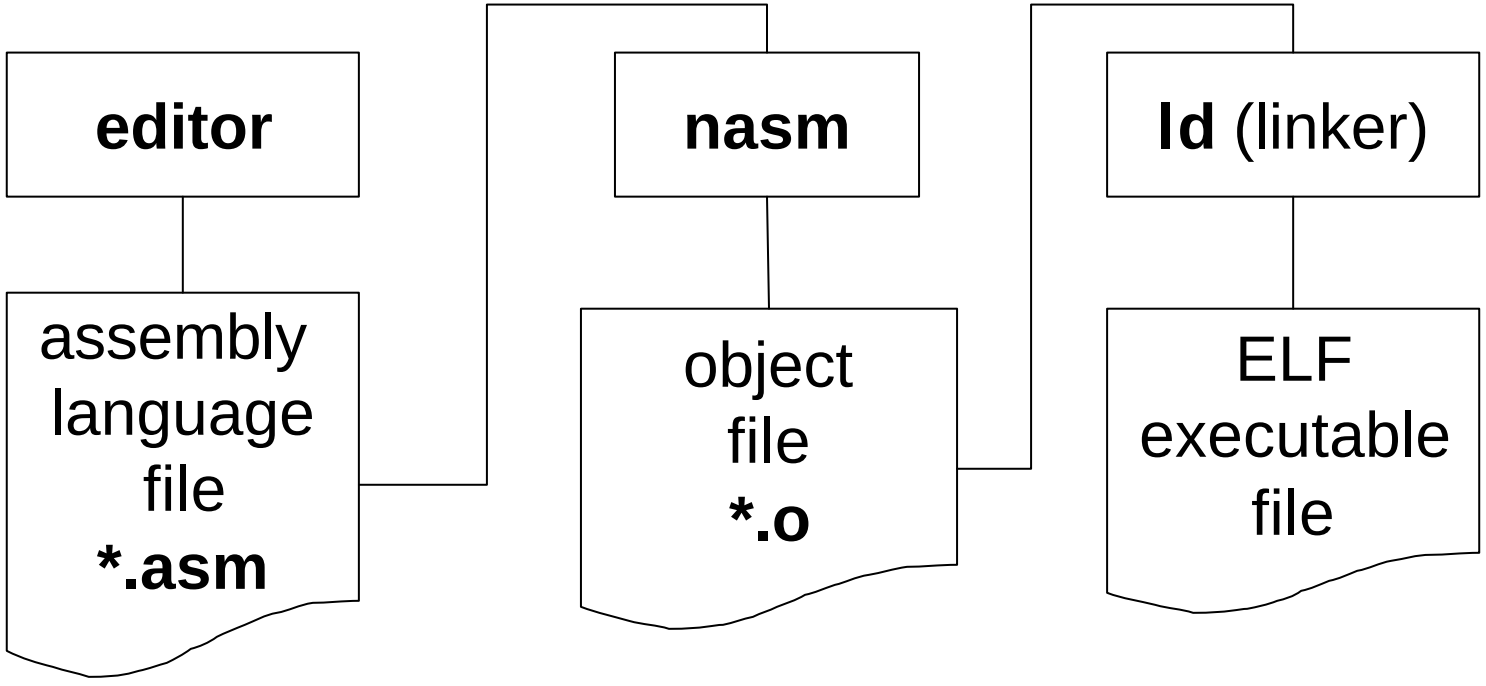
CHAPTER 4

The Netwide Assembly Language

nasm: The Netwide Assembler

- Free and portable x86 assembler originally developed by Simon Tatham and Julian Hall.
- It supports a range of object file formats, including Linux ELF, NetBSD/FreeBSD, COFF, Microsoft 16-bit OBJ and Win32.

Development Cycle



ld: The Linker

- An object file isn't directly executable; it first needs to be fed into a **linker** (also known as **link-loader** or **link-editor**).
- The linker does the following tasks:
  - identifies the initial program entry point (**`_start`** label)
  - binds symbolic references to memory addresses
  - unites all the object and library files
  - produces an executable ELF file

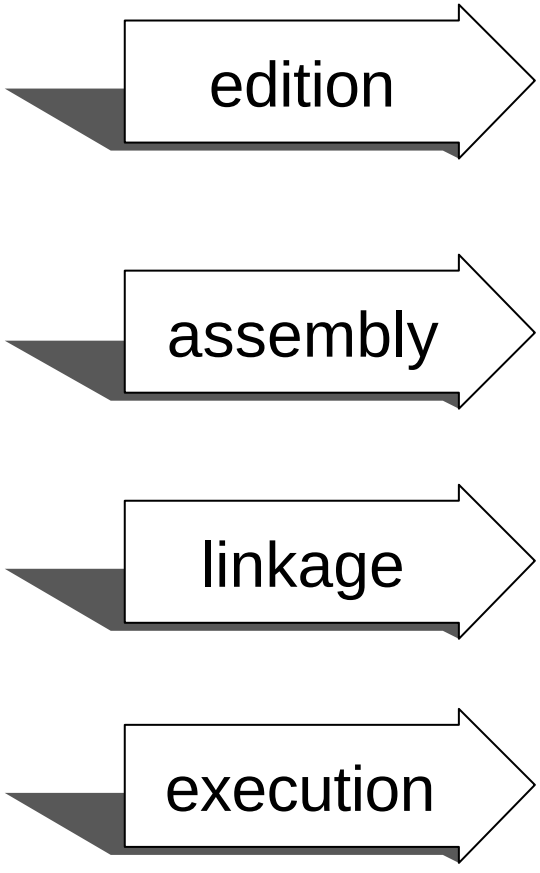
ELF File

- The *Executable and Linkable Format* was designed by the UNIX System Laboratories.
- Used by contemporary Linux implementations as its standard executable file format.
- Supports shared libraries (dynamic linking).

a.out File

- **a.out** is the default file name given to executable files by UNIX linkers.
- It means “assembly output”, in spite of being linker output!
- On the PDP-7 computer, there was no linker. Executable programs were created directly by the assembler. The name stuck, even when the linkers started to appear in newer machines.

Building a Program

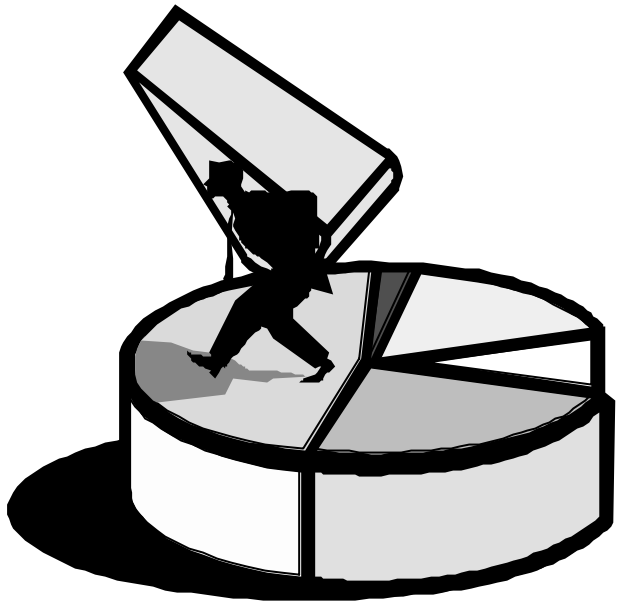



```
$ vi test.asm
$ ls
test.asm
$ nasm -f elf test.asm
$ ls
test.asm     test.o
$ ld -s -o test test.o
$ ls
test   test.asm   test.o
$ test
```

Linux-NASM Program Skeleton

```
bits 32                 ; -- 32 bit program
section .data           ; -- Start data segment
    ; put initialized data here
section .bss            ; -- Start bss segment
    ; put non-initialized data here
section .text           ; -- Start code segment
    global _start       ; -- Export “_start” label
_start                  ; -- Define “_start” label
    ; put program code here
    mov eax, 1          ; -- Exit system call
    mov ebx, 0          ;    exit code #0
    int 0x80
```

Segments



- A **segment** on UNIX is *a section of related stuff in a binary*.
- ELF files have three segments:
  - **TEXT** for storing code
  - **DATA** for storing initialized data
  - **BSS** for non-initialized data

NASM Source Code

- Every NASM program source line has the following four fields:

label*: *instruction operands* ; *comment

- Every field is optional.
- The number of operands depend of the instruction.

Instructions

Pseudo-Instructions

- **Mnemonics** that represent x86 *opcodes*.
- Generate code that produce actions at run time.

- Not real x86 instructions (they don't produce any actions at run time).
- Are used in the instruction field because that's the most convenient place to put them.

Directives

- Statements that allow us to control how a program is assembled.
- They only work at assembly time (they don't directly produce any machine code).

`bits` Directive

- Specifies if NASM must produce code that will run in 16 or 32-bit mode.
- ELF files only support 32-bit mode:

```
bits 32
```

- May be omitted for ELF files.

`section .data` Directive

- States the beginning of the initialized data segment.
- An image of this segment's data is physically stored in the executable file.
- This segment contains read/write data.

Pseudo-Instructions for the Data Segment

Pseudo-Instruction	***Meaning***	***Size (bits)***
`db`	Define byte	8
`dw`	Define word	16
`dd`	Define double word	32
`dq`	Define quadword	64
`dt`	Define ten bytes	80

`section .bss` Directive

- States the beginning of the non-initialized data segment.
- Only the size of the data is stored in the executable file. Once the program is loaded into memory, all the data in this section is set to zero.
- This segment contains read/write data.
- BSS means “Block Started by Symbol”, a pseudo-instruction from the old IBM 704 assembler, carried over into UNIX.