Basic Concepts

COE 205

Computer Organization and Assembly Language

Dr. Aiman El-Maleh

College of Computer Sciences and Engineering

King Fahd University of Petroleum and Minerals

[Adapted from slides of Dr. Kip Irvine: Assembly Language for Intel-Based Computers]

Outline

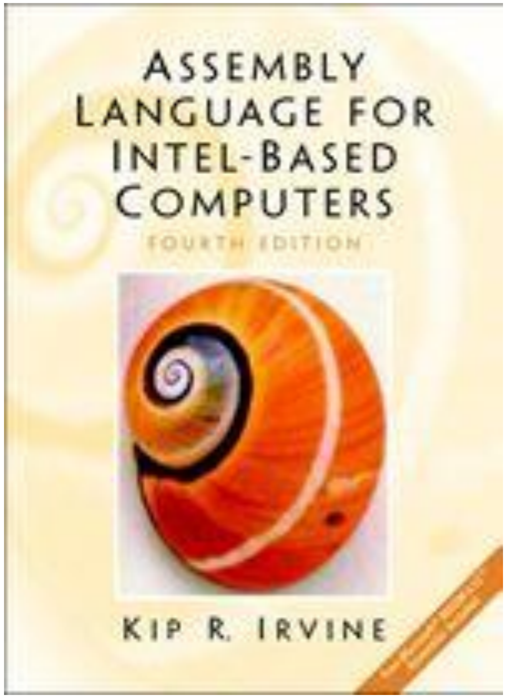
- Welcome to COE 205
- Assembly-, Machine-, and High-Level Languages
- Assembly Language Programming Tools
- Programmer's View of a Computer System
- Basic Computer Organization

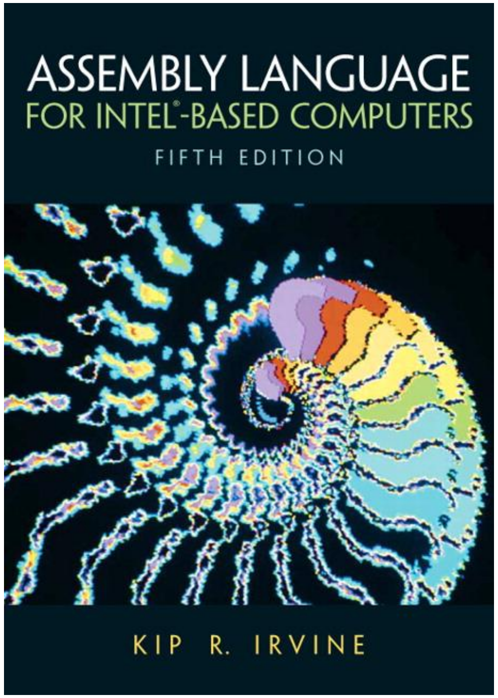
Welcome to COE 205

- Assembly language programming
- Basics of computer organization
- CPU design
- Software Tools
  - Microsoft Macro Assembler (MASM) version 6.15
  - Link Libraries provided by Author (Irvine32.lib and Irivine16.lib)
  - Microsoft Windows debugger
  - ConTEXT Editor

Textbook

- Kip Irvine: Assembly Language for Intel-Based Computers
  - 4th edition (2003)
  - 5th edition (2007)





Course Objectives

After successfully completing the course, students will be able to:

- ❖ Describe the basic components of a computer system, its instruction set architecture and its basic fetch-execute cycle operation.
- ❖ Describe how data is represented in a computer and recognize when overflow occurs.
- ❖ Recognize the basics of assembly language programming including addressing modes.
- ❖ Analyze, design, implement, and test assembly language programs.
- ❖ Recognize, analyze, and design the basic components of a simple CPU including datapath and control unit design alternatives.

Course Learning Outcomes

- ❖ Ability to analyze, design, implement, and test assembly language programs.
- ❖ Ability to use tools and skills in analyzing and debugging assembly language programs.
- ❖ Ability to design the datapath and control unit of a simple CPU.
- ❖ Ability to demonstrate self-learning capability.
- ❖ Ability to work in a team.

Required Background

- The student should already be able to program confidently in at least one high-level programming language, such as Java or C.
- Prerequisite
  - COE 202: Fundamentals of computer engineering
  - ICS 102: Introduction to computing
- Only students with computer engineering major should be registered in this course.

Grading Policy

❖ Discussions & Reflections	5%	
❖ Programming Assignments	10%	
❖ Quizzes	10%	
❖ Exam I	15%	(Sun. March 28, 2010)
❖ Exam II	20%	(Th. May 20, 2010)
❖ Laboratory	20%	
❖ Final	20%	

- ✧ Attendance will be taken regularly.
- ✧ Excuses for officially authorized absences must be presented no later than one week following resumption of class attendance.
- ✧ Late assignments will be accepted but you will be penalized 10% per each late day.
- ✧ A student caught cheating in any of the assignments will get 0 out of 10%.
- ✧ No makeup will be made for missing Quizzes or Exams.

Course Topics

- ***Introduction and Information Representation:*** **7 lectures**
  Introduction to computer organization. Instruction Set Architecture. Computer Components. Fetch-Execute cycle. Signed number representation ranges. Overflow.
- ***Assembly Language Concepts:*** **7 lectures**
  Assembly language format. Directives vs. instructions. Constants and variables. I/O. INT 21H. Addressing modes.
- ***8086 Assembly Language Programming:*** **19 lectures**
  Register set. Memory segmentation. MOV instructions. Arithmetic instructions and flags (ADD, ADC, SUB, SBB, INC, DEC, MUL, IMUL, DIV, IDIV). Compare, Jump and loop (CMP, JMP, Cond. jumps, LOOP). Logic, shift and rotate. Stack operations. Subprograms. Macros. I/O (IN, OUT). String instructions. Interrupts and interrupt processing, INT and IRET.

Course Topics

- ***CPU Design:*** ***12 lectures***

  Register transfer. Data-path design. 1-bus, 2-bus and 3-bus CPU organization. Fetch and execute phases of instruction processing. Performance consideration. Control steps. CPU-Memory interface circuit. Hardwired control unit design. Microprogramming. Horizontal and Vertical microprogramming. Microprogrammed control unit design.

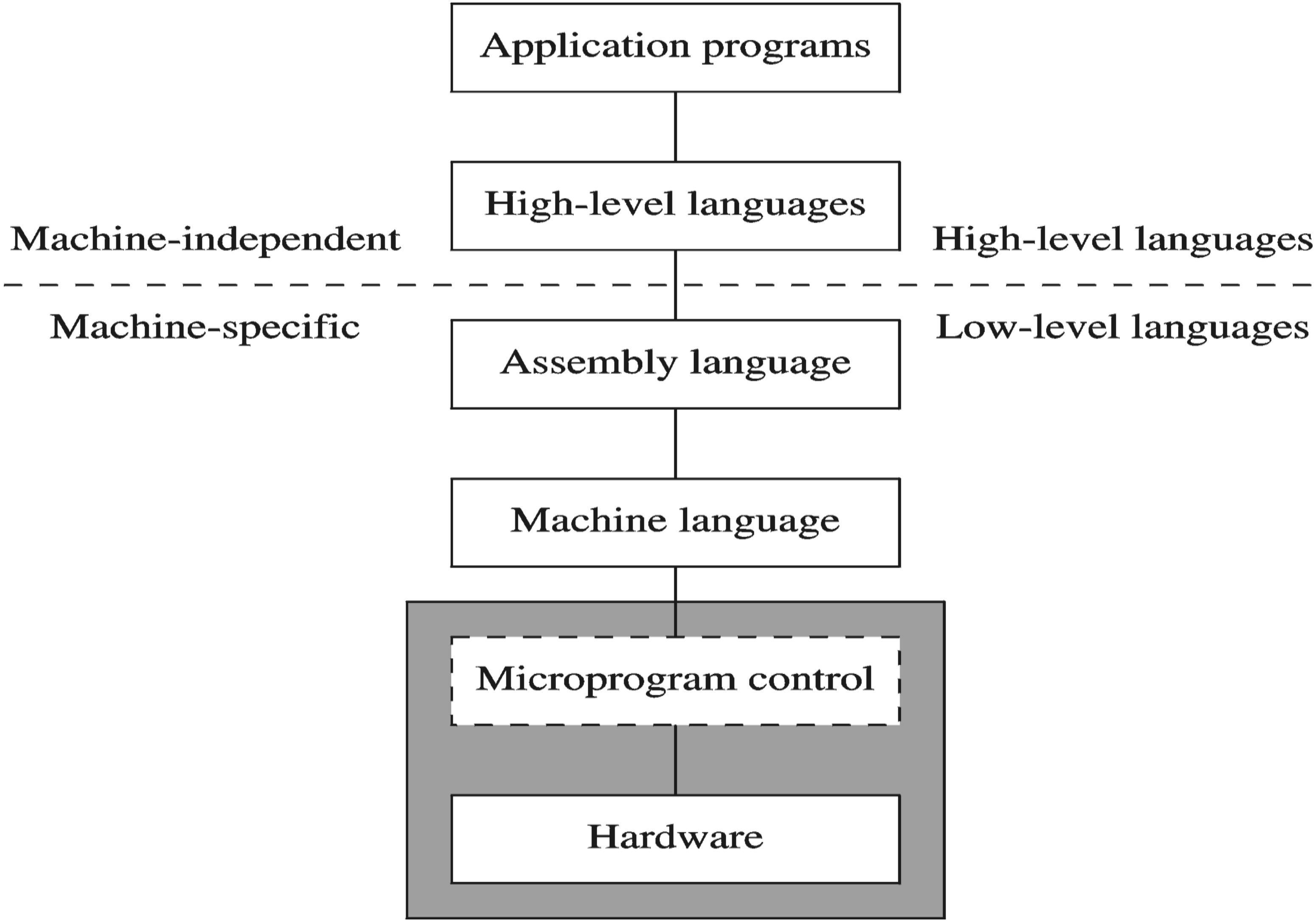
Next …

- ❖ Welcome to COE 205
- ❖ Assembly-, Machine-, and High-Level Languages
- ❖ Assembly Language Programming Tools
- ❖ Programmer's View of a Computer System
- ❖ Basic Computer Organization

Some Important Questions to Ask

- What is Assembly Language?
- Why Learn Assembly Language?
- What is Machine Language?
- How is Assembly related to Machine Language?
- What is an Assembler?
- How is Assembly related to High-Level Language?
- Is Assembly Language portable?

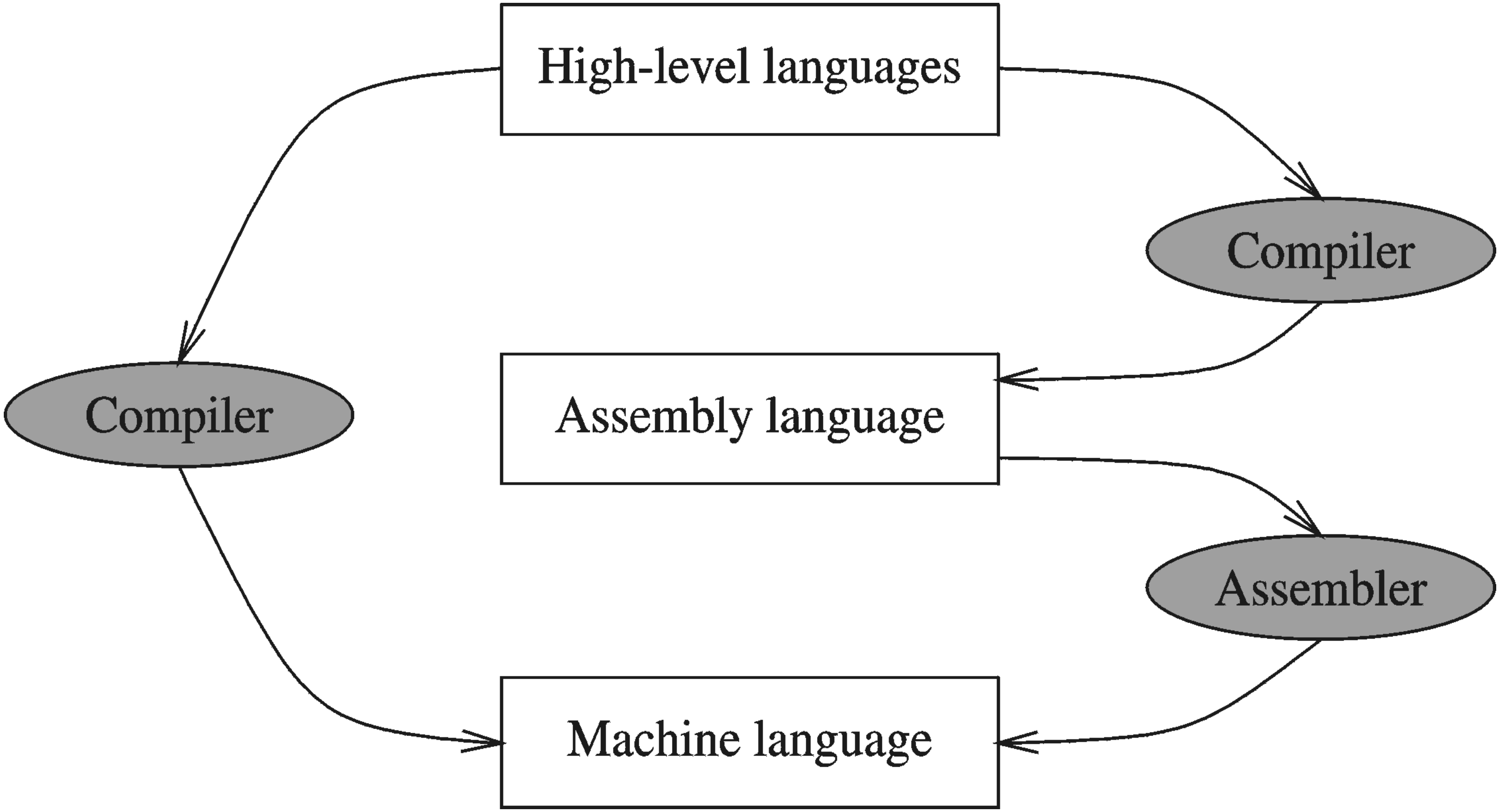
A Hierarchy of Languages



Assembly and Machine Language

- Machine language
  - Native to a processor: executed directly by hardware
  - Instructions consist of binary code: 1s and 0s
- Assembly language
  - A programming language that uses symbolic names to represent operations, registers and memory locations.
  - Slightly higher-level language
  - Readability of instructions is better than machine language
  - One-to-one correspondence with machine language instructions
- Assemblers translate assembly to machine code
- Compilers translate high-level programs to machine code
  - Either directly, or
  - Indirectly via an assembler

Compiler and Assembler



Instructions and Machine Language

- Each command of a program is called an instruction (it instructs the computer what to do).
- Computers only deal with binary data, hence the instructions must be in binary format (0s and 1s) .
- The set of all instructions (in binary form) makes up the computer's machine language. This is also referred to as the instruction set.

Instruction Fields

- Machine language instructions usually are made up of several fields. Each field specifies different information for the computer. The major two fields are:
- Opcode field which stands for operation code and it specifies the particular operation that is to be performed.
  - Each operation has its unique opcode.
- Operands fields which specify where to get the source and destination operands for the operation specified by the opcode.
  - The source/destination of operands can be a constant, the memory or one of the general-purpose registers.

Assembly vs. Machine Code

Instruction Address	Machine Code	Assembly Instruction
0005	B8 0001	MOV AX, 1
0008	B8 0002	MOV AX, 2
000B	B8 0003	MOV AX, 3
000E	B8 0004	MOV AX, 4
0011	BB 0001	MOV BX, 1
0014	B9 0001	MOV CX, 1
0017	BA 0001	MOV DX, 1
001A	8B C3	MOV AX, BX
001C	8B C1	MOV AX, CX
001E	8B C2	MOV AX, DX
0020	83 C0 01	ADD AX, 1
0023	83 C0 02	ADD AX, 2
0026	03 C3	ADD AX, BX
0028	03 C1	ADD AX, CX
002A	03 06 0000	ADD AX, i
002E	83 E8 01	SUB AX, 1
0031	2B C3	SUB AX, BX
0033	05 1234	ADD AX, 1234h

Flash Movie

Translating Languages

English: D is assigned the sum of A times B plus 10.

High-Level Language: D = A * B + 10

A statement in a high-level language is translated typically into several machine-level instructions

Intel Assembly Language:

```
mov   eax, A
mul   B
add   eax, 10
mov   D, eax
```

Intel Machine Language:

```
A1 00404000
F7 25 00404004
83 C0 0A
A3 00404008
```

Mapping Between Assembly Language and HLL

- Translating HLL programs to machine language programs is not a one-to-one mapping
- A HLL instruction (usually called a statement) will be translated to one or more machine language instructions

Mapping between some C instructions and 8086 assembly language

Instruction Class	C	Assembly Language
Data Movement	a = 5	MOV a, 5

Arithmetic/Logic	b = a + 5	MOV ax, a
		ADD ax, 5
		MOV b, ax

Control Flow	goto LBL	JMP LBL

Advantages of High-Level Languages

- Program development is faster
  - High-level statements: fewer instructions to code
- Program maintenance is easier
  - For the same above reasons
- Programs are portable
  - Contain few machine-dependent details
    - Can be used with little or no modifications on different machines
  - Compiler translates to the target machine language
  - However, Assembly language programs are not portable

Why Learn Assembly Language?

- Accessibility to system hardware
  - Assembly Language is useful for implementing system software
  - Also useful for small embedded system applications
- Space and Time efficiency
  - Understanding sources of program inefficiency
  - Tuning program performance
  - Writing compact code
- Writing assembly programs gives the computer designer the needed deep understanding of the instruction set and how to design one
- To be able to write compilers for HLLs, we need to be expert with the machine language. Assembly programming provides this experience

Assembly vs. High-Level Languages

❖Some representative types of applications:

Type of Application	High-Level Languages	Assembly Language
Business application software, written for single platform, medium to large size.	Formal structures make it easy to organize and maintain large sections of code.	Minimal formal structure, so one must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code.
Hardware device driver.	Language may not provide for direct hardware access. Even if it does, awkward coding techniques must often be used, resulting in maintenance difficulties.	Hardware access is straightforward and simple. Easy to maintain when programs are short and well documented.
Business application written for multiple platforms (different operating systems).	Usually very portable. The source code can be recompiled on each target operating system with minimal changes.	Must be recoded separately for each platform, often using an assembler with a different syntax. Difficult to maintain.
Embedded systems and computer games requiring direct hardware access.	Produces too much executable code, and may not run efficiently.	Ideal, because the executable code is small and runs quickly.

Next …

- ❖ Welcome to COE 205
- ❖ Assembly-, Machine-, and High-Level Languages
- ❖ Assembly Language Programming Tools
- ❖ Programmer's View of a Computer System
- ❖ Basic Computer Organization

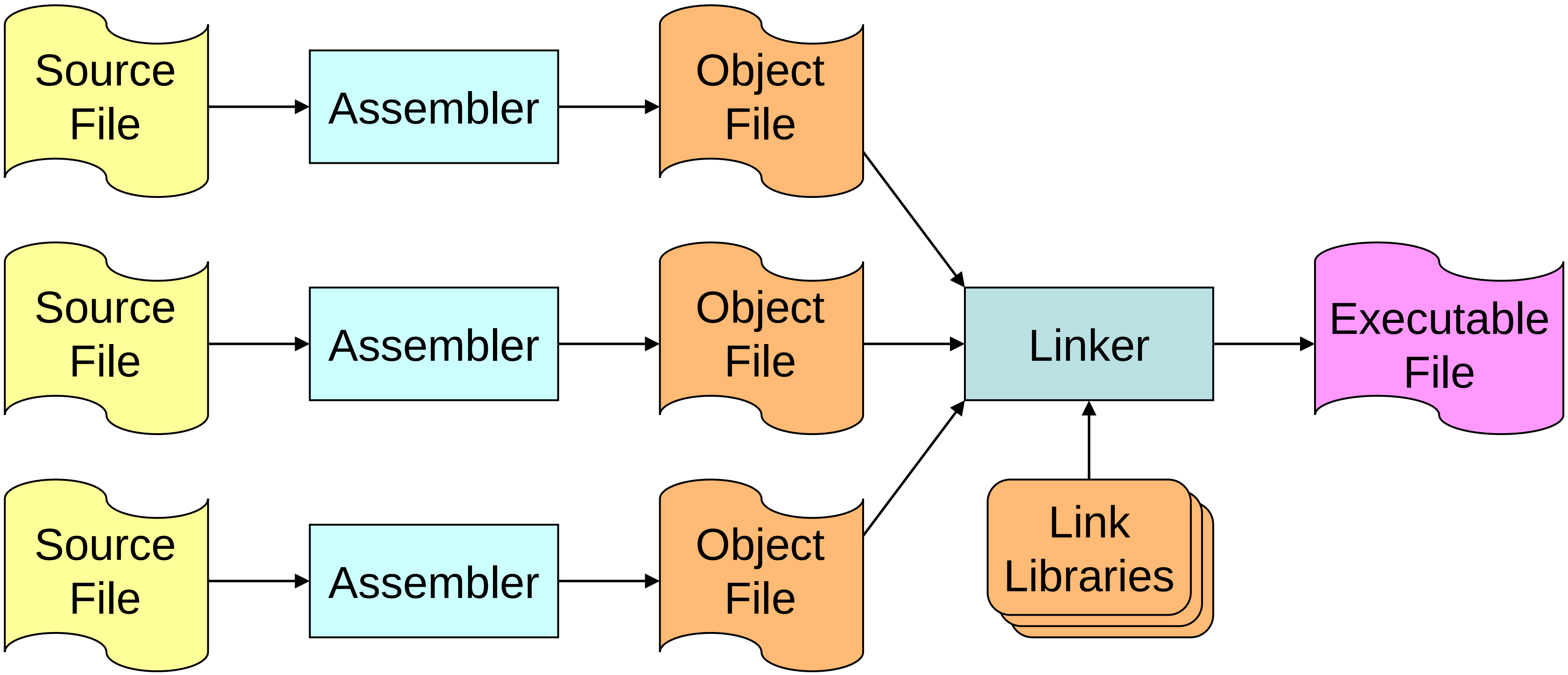
Assembler

- ❖ Software tools are needed for editing, assembling, linking, and debugging assembly language programs
- ❖ An assembler is a program that converts source-code programs written in assembly language into object files in machine language
- ❖ Popular assemblers have emerged over the years for the Intel family of processors. These include …
  - ✧ TASM (Turbo Assembler from Borland)
  - ✧ NASM (Netwide Assembler for both Windows and Linux), and
  - ✧ GNU assembler distributed by the free software foundation
- ❖ We will use MASM (Macro Assembler from Microsoft)

Linker and Link Libraries

- You need a linker program to produce executable files
- It combines your program's object file created by the assembler with other object files and link libraries, and produces a single executable program
- LINK32.EXE is the linker program provided with the MASM distribution for linking 32-bit programs
- We will also use a link library for input and output
- Called Irvine32.lib developed by Kip Irvine
  - Works in Win32 console mode under MS-Windows

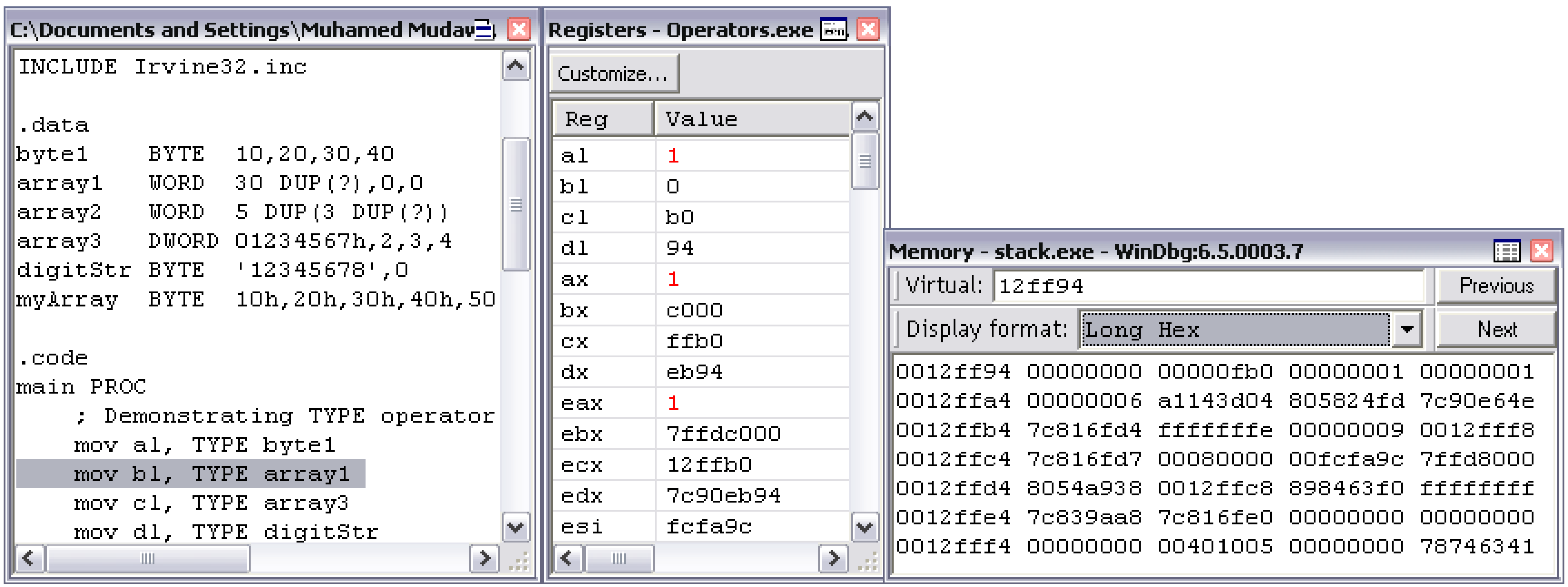
Assemble and Link Process



- A project may consist of multiple source files
- Assembler translates each source file separately into an object file
- Linker links all object files together with link libraries

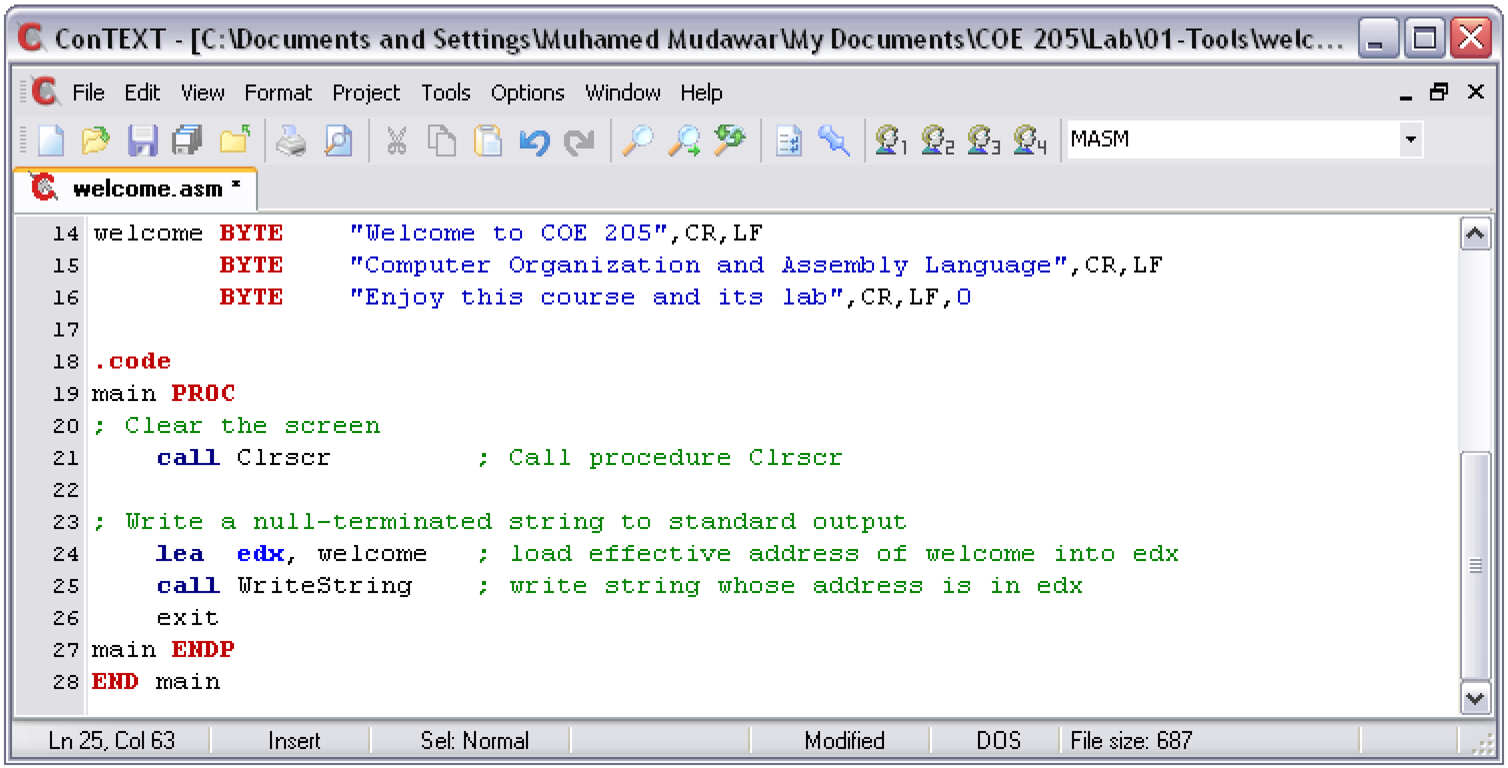
Debugger

- Allows you to trace the execution of a program
- Allows you to view code, memory, registers, etc.
- We will use the 32-bit Windows debugger



Editor

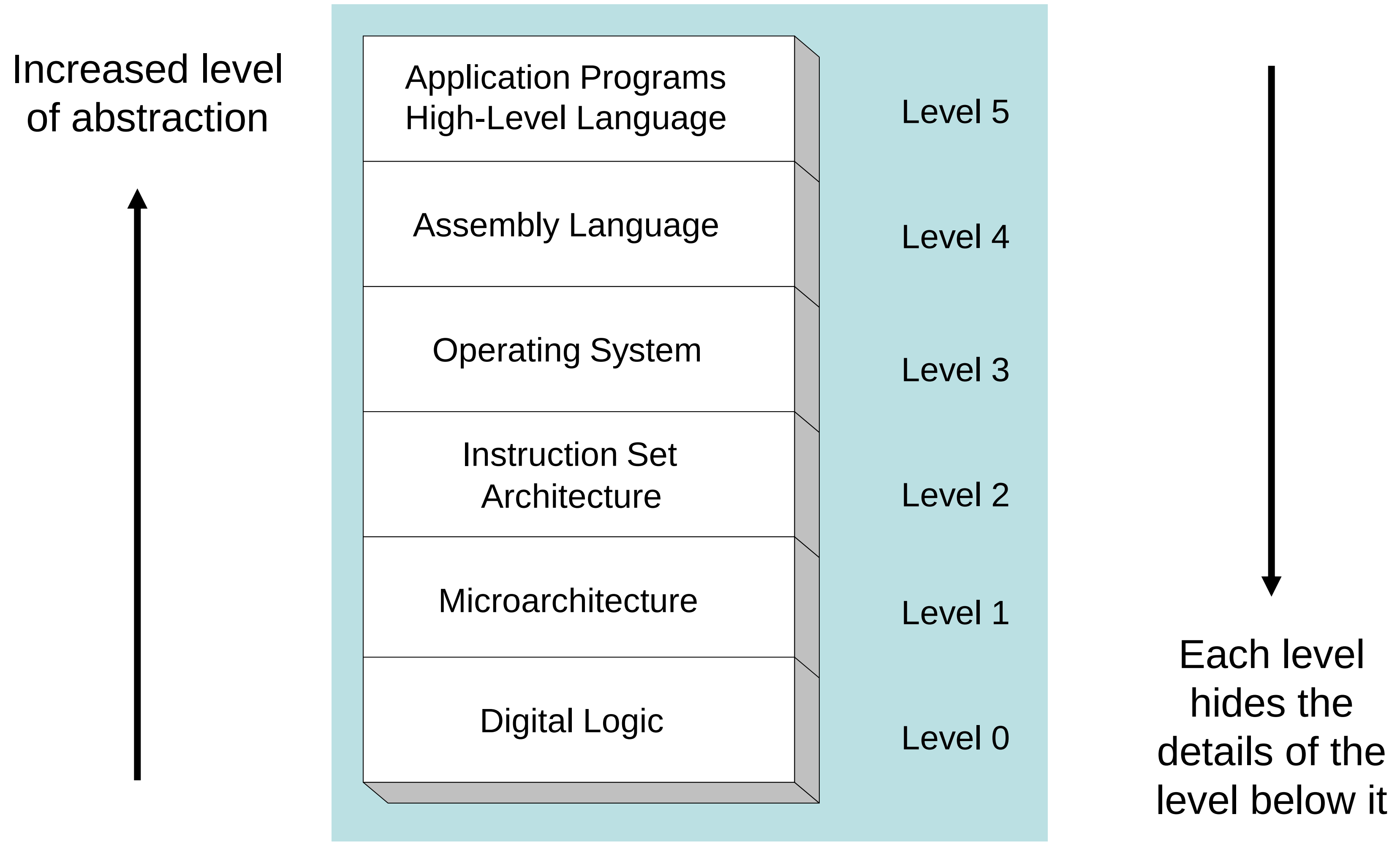
- Allows you to create assembly language source files
- Some editors provide syntax highlighting features and can be customized as a programming environment



Next …

- ❖ Welcome to COE 205
- ❖ Assembly-, Machine-, and High-Level Languages
- ❖ Assembly Language Programming Tools
- ❖ Programmer's View of a Computer System
- ❖ Basic Computer Organization

Programmer's View of a Computer System



Programmer's View – 2

- Application Programs (Level 5)
  - Written in high-level programming languages
  - Such as Java, C++, Pascal, Visual Basic . . .
  - Programs compile into assembly language level (Level 4)
- Assembly Language (Level 4)
  - Instruction mnemonics are used
  - Have one-to-one correspondence to machine language
  - Calls functions written at the operating system level (Level 3)
  - Programs are translated into machine language (Level 2)
- Operating System (Level 3)
  - Provides services to level 4 and 5 programs
  - Translated to run at the machine instruction level (Level 2)

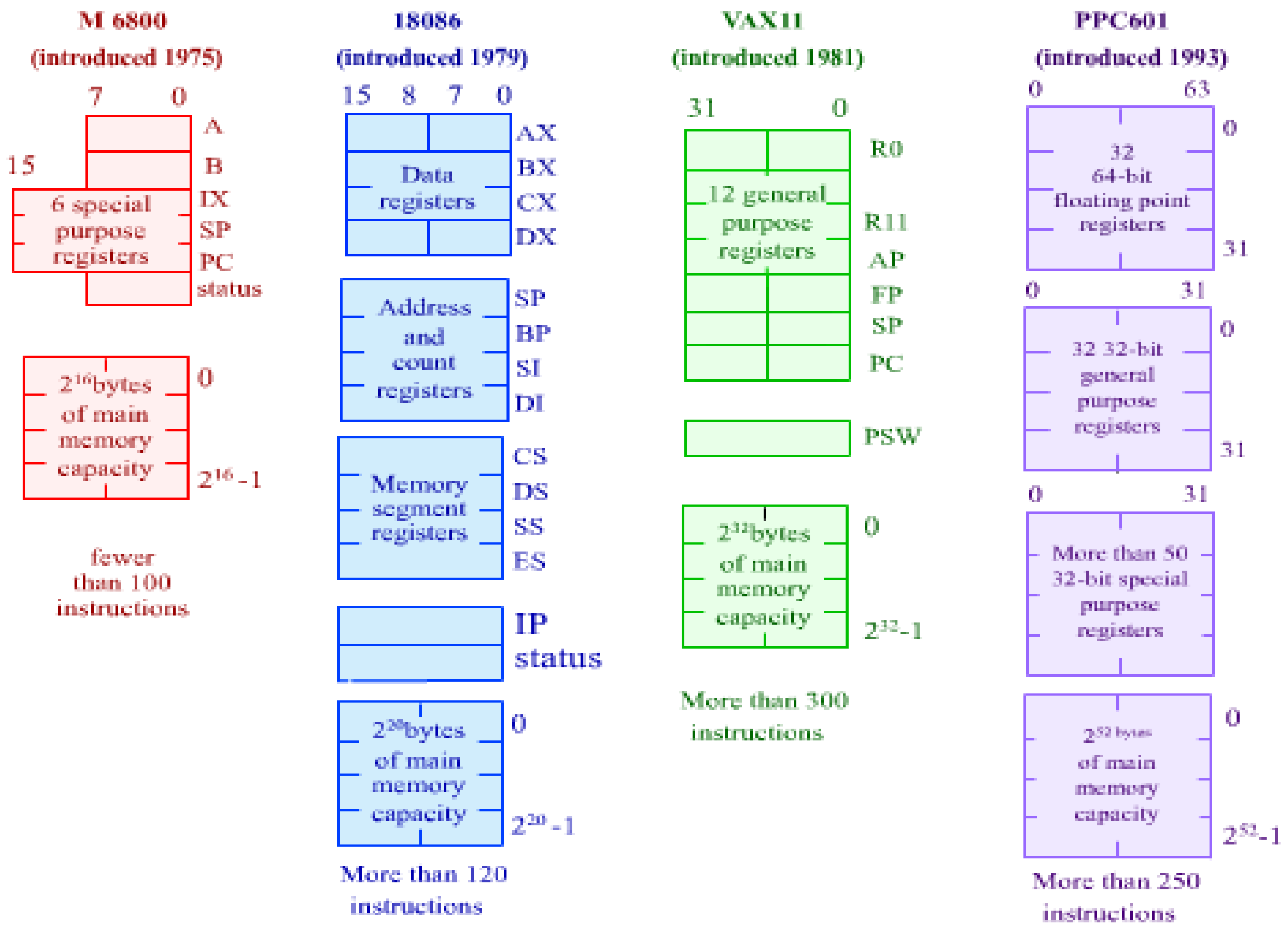
Programmer's View – 3

- Instruction Set Architecture (Level 2)
  - Specifies how a processor functions
  - Machine instructions, registers, and memory are exposed
  - Machine language is executed by Level 1 (microarchitecture)
- Microarchitecture (Level 1)
  - Controls the execution of machine instructions (Level 2)
  - Implemented by digital logic (Level 0)
- Digital Logic (Level 0)
  - Implements the microarchitecture
  - Uses digital logic gates
  - Logic gates are implemented using transistors

Instruction Set Architecture (ISA)

- Collection of assembly/machine instruction set of the machine
- Machine resources that can be managed with these instructions
  - Memory
  - Programmer-accessible registers.
- Provides a hardware/software interface

Instruction Set Architecture (ISA)

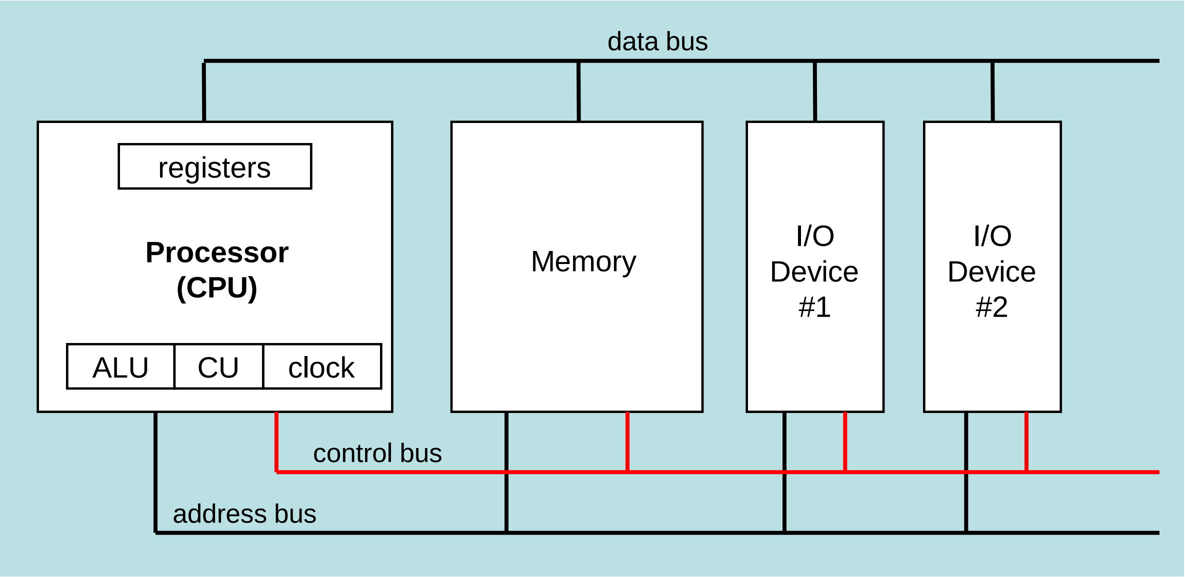


Next …

- ❖ Welcome to COE 205
- ❖ Assembly-, Machine-, and High-Level Languages
- ❖ Assembly Language Programming Tools
- ❖ Programmer's View of a Computer System
- ❖ Basic Computer Organization

Basic Computer Organization

- Since the 1940's, computers have 3 classic components:
  - Processor, called also the CPU (Central Processing Unit)
  - Memory and Storage Devices
  - I/O Devices
- Interconnected with one or more buses
- Bus consists of
  - Data Bus
  - Address Bus
  - Control Bus



Processor (CPU)

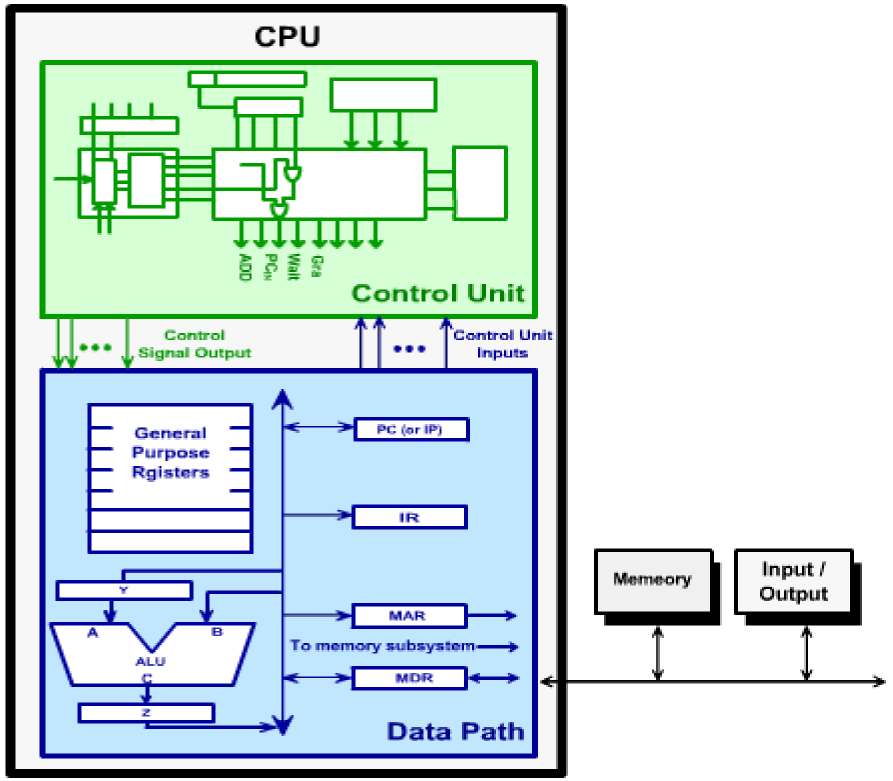
- Processor consists of
  - Datapath
    - ALU
    - Registers
  - Control unit

Flash Movie

- ALU
  - Performs arithmetic and logic instructions

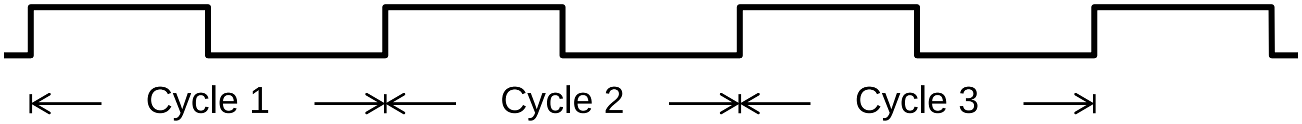
Flash Movie

- Control unit (CU)
  - Generates the control signals required to execute instructions
- Implementation varies from one processor to another



Clock

- ❖ Synchronizes Processor and Bus operations
- ❖ Clock cycle = Clock period = 1 / Clock rate

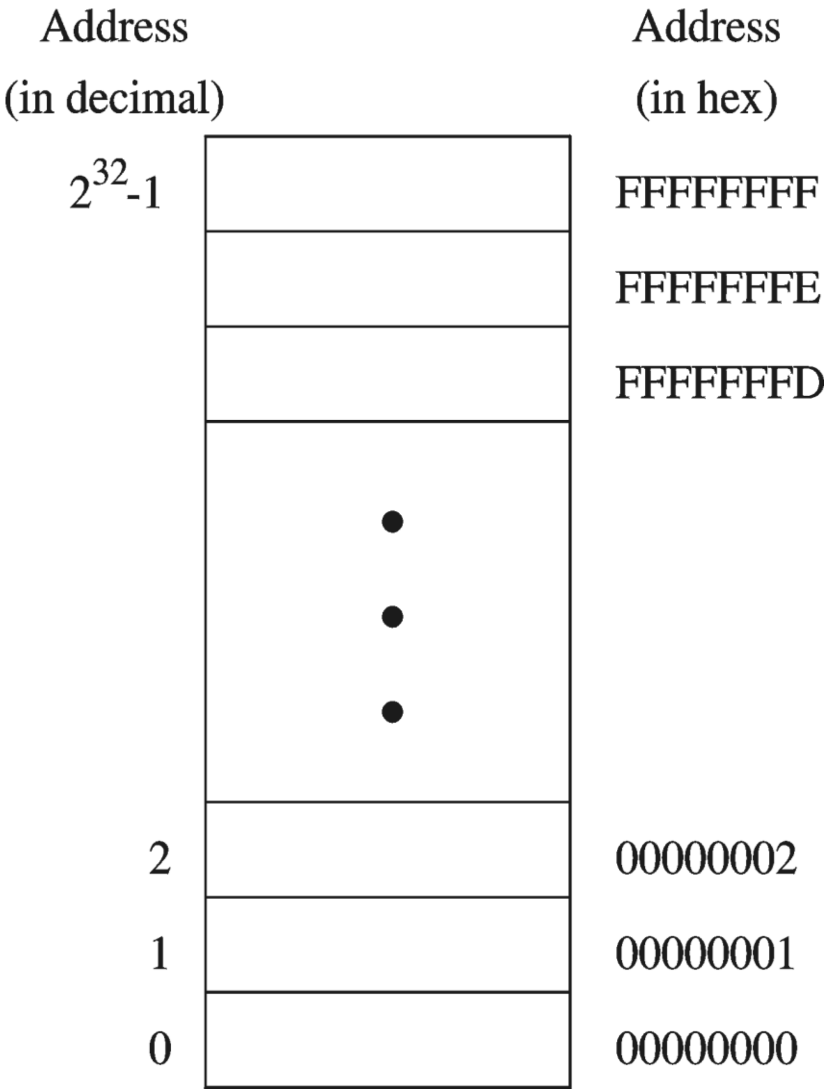


- ❖ Clock rate = Clock frequency = Cycles per second
  - ✧ 1 Hz = 1 cycle/sec　　1 KHz = 10^3 cycles/sec
  - ✧ 1 MHz = 10^6 cycles/sec　　1 GHz = 10^9 cycles/sec
  - ✧ 2 GHz clock has a cycle time = $1/(2\times10^9)$ = 0.5 nanosecond (ns)
- ❖ Clock cycles measure the execution of instructions

Memory

- Ordered sequence of bytes
  - The sequence number is called the memory address
- Byte addressable memory
  - Each byte has a unique address
  - Supported by almost all processors
- Physical address space
  - Determined by the address bus width
  - Pentium has a 32-bit address bus
    - Physical address space = **4GB = 2^{32} bytes**
  - Itanium with a 64-bit address bus can support
    - Up to **2^{64} bytes** of physical address space

Address Space



Address Space is the set of memory locations (bytes) that can be addressed

CPU Memory Interface

❖ Address Bus

✧ Memory address is put on address bus

✧ If memory address = m bits then 2^m locations are addressed

❖ Data Bus: b-bit bi-directional bus

✧ Data can be transferred in both directions on the data bus

✧ Note that b is not necessary equal to w or s. So data transfers might take more than a single cycle (if w > b) .

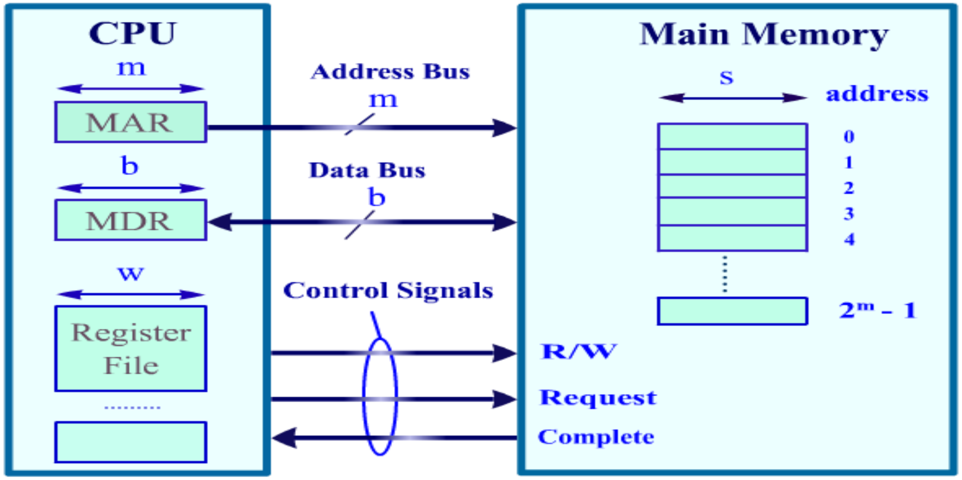
❖ Control Bus

✧ Signals control transfer of data

✧ Read request

✧ Write request

✧ Complete transfer



Memory Devices

- Random-Access Memory (RAM)
  - Usually called the main memory
  - It can be read and written to
  - It does not store information permanently (Volatile , when it is powered off, the stored information are gone)
  - Information stored in it can be accessed in any order at equal time periods (hence the name random access)
  - Information is accessed by an address that specifies the exact location of the piece of information in the RAM.
  - DRAM = Dynamic RAM
    - 1-Transistor cell + trench capacitor
    - Dense but slow, must be refreshed
    - Typical choice for main memory

  - SRAM: Static RAM
    - 6-Transistor cell, faster but less dense than DRAM
    - Typical choice for cache memory

Memory Devices

❖ ROM (Read-Only-Memory)

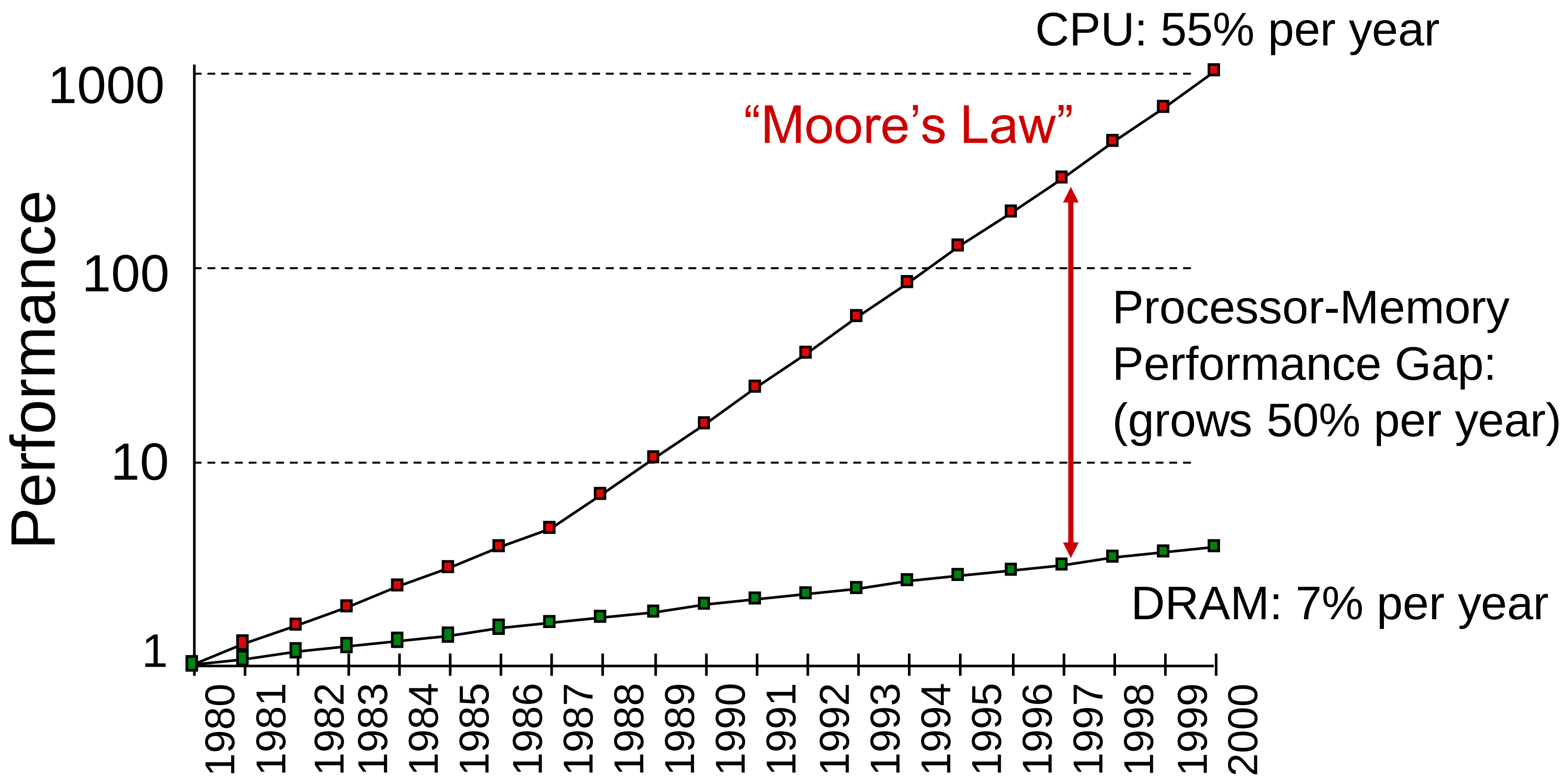
- A read-only-memory, non-volatile i.e. stores information permanently
- Has random access of stored information

- Used to store the information required to startup the computer
- Many types: ROM, EPROM, EEPROM, and FLASH
- FLASH memory can be erased electrically in blocks

❖ Cache

- A very fast type of RAM that is used to store information that is most frequently or recently used by the computer
- Recent computers have 2-levels of cache; the first level is faster but smaller in size (usually called internal cache), and the second level is slower but larger in size (external cache).

Processor-Memory Performance Gap



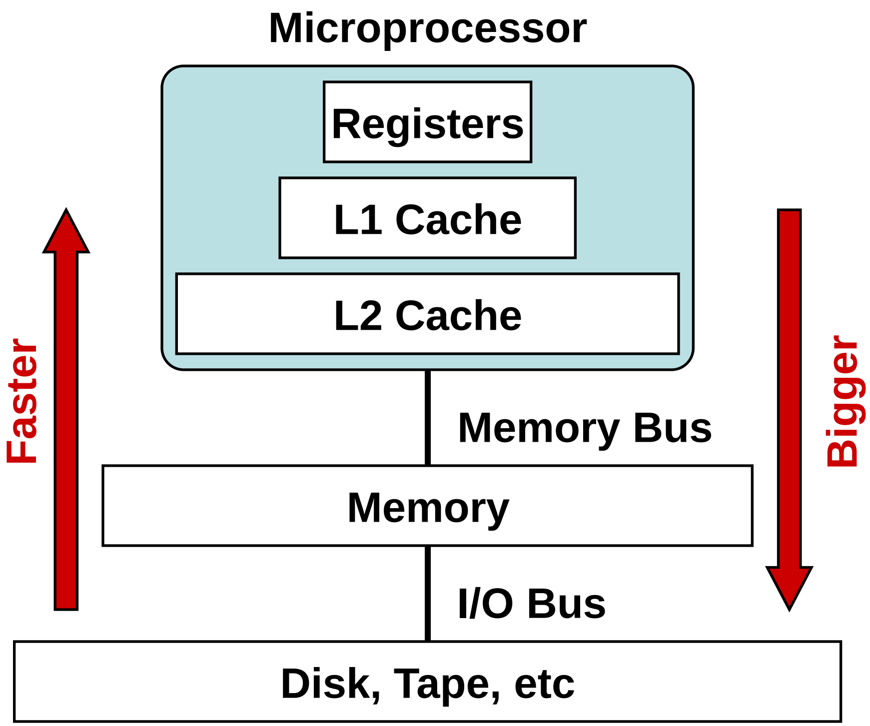
- ❖ 1980 – No cache in microprocessor
- ❖ 1995 – Two-level cache on microprocessor

The Need for a Memory Hierarchy

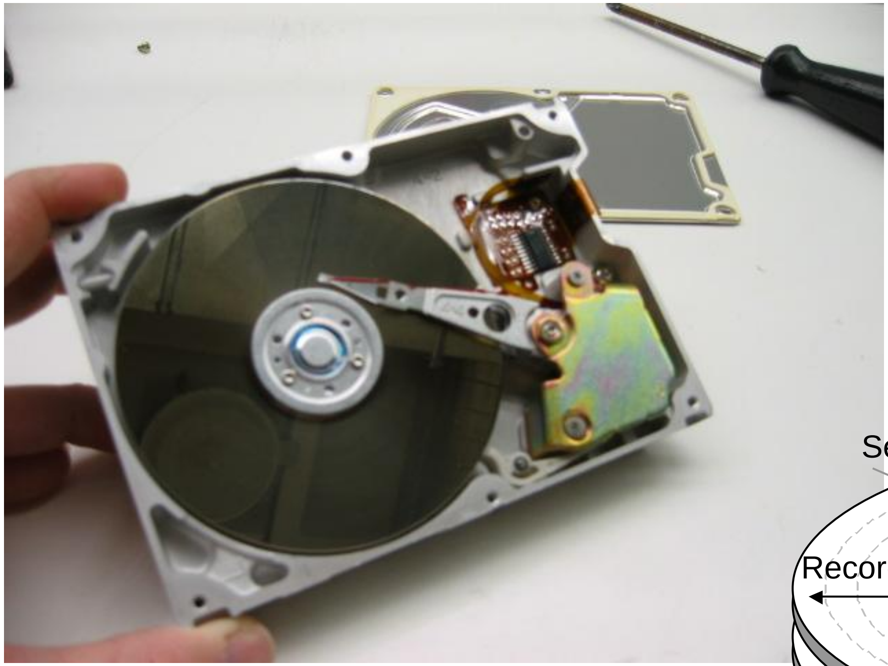
- Widening speed gap between CPU and main memory
  - Processor operation takes less than 1 ns
  - Main memory requires more than 50 ns to access
- Each instruction involves at least one memory access
  - One memory access to fetch the instruction
  - Additional memory accesses for instructions involving memory data access
- Memory bandwidth limits the instruction execution rate
- Cache memory can help bridge the CPU-memory gap
- Cache memory is small in size but fast

Typical Memory Hierarchy

- Registers are at the top of the hierarchy
  - Typical size < 1 KB
  - Access time < 0.5 ns
- Level 1 Cache (8 – 64 KB)
  - Access time: 0.5 – 1 ns
- L2 Cache (512KB – 8MB)
  - Access time: 2 – 10 ns
- Main Memory (1 – 2 GB)
  - Access time: 50 – 70 ns
- Disk Storage (> 200 GB)
  - Access time: milliseconds



Magnetic Disk Storage

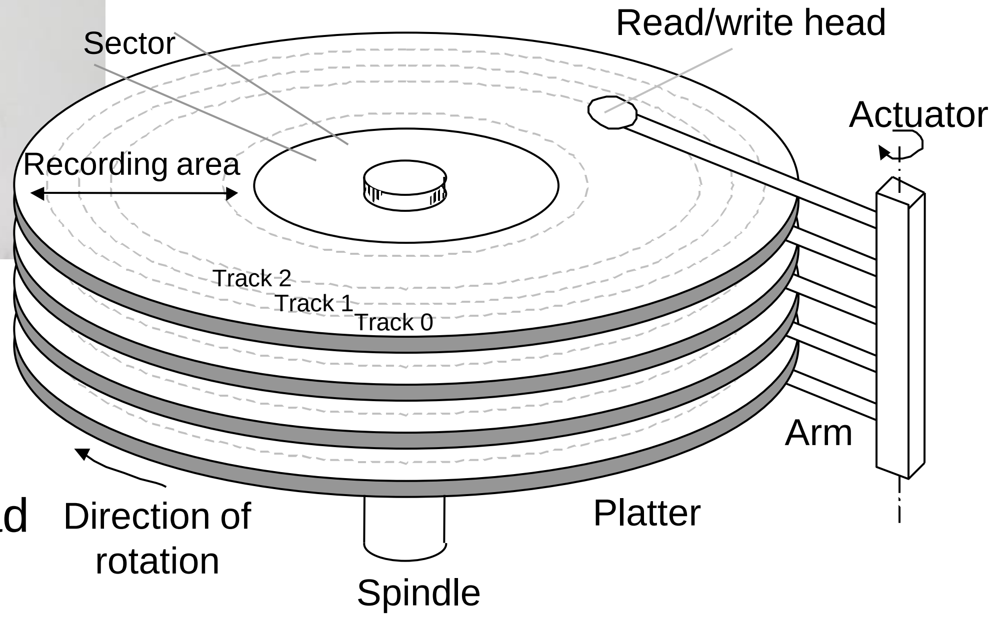


Disk Access Time =

Seek Time +

Rotation Latency +

Transfer Time



Seek Time: head movement to the desired track (milliseconds)

Rotation Latency: disk rotation until desired sector arrives under the head

Transfer Time: to transfer data

Example on Disk Access Time

- Given a magnetic disk with the following properties
  - Rotation speed = 7200 RPM (rotations per minute)
  - Average seek = 8 ms, Sector = 512 bytes, Track = 200 sectors
- Calculate
  - Time of one rotation (in milliseconds)
  - Average time to access a block of 32 consecutive sectors
- Answer
  - Rotations per second = 7200/60 = 120 RPS
  - Rotation time in milliseconds = 1000/120 = 8.33 ms
  - Average rotational latency = time of half rotation = 4.17 ms
  - Time to transfer 32 sectors = (32/200) * 8.33 = 1.33 ms
  - Average access time = 8 + 4.17 + 1.33 = 13.5 ms