Module 4
Instruction Set Architecture (ISA)

REFERENCE:
WILLIAM STALLINGS – COMPUTER ORGANIZATION & ARCHITECTURE
KIP IRVINE – ASSEMBLY LANGUAGE FOR INTEL-BASED COMPUTERS

Content Overview

- Hierarchy of Computer Languages
- General Concepts
- ISA Level
- Elements of Instructions
- Instructions Types
- Instruction Formats
- Number of Addresses
- Registers
- Types of Operands
- Addressing Modes

Content Overview

- **Hierarchy of Computer Languages**
- **General Concepts**
- **ISA Level**
- **Elements of Instructions**
- Instructions Types
- Instruction Formats
- Number of Addresses
- Registers
- Types of Operands
- Addressing Modes

Part 1

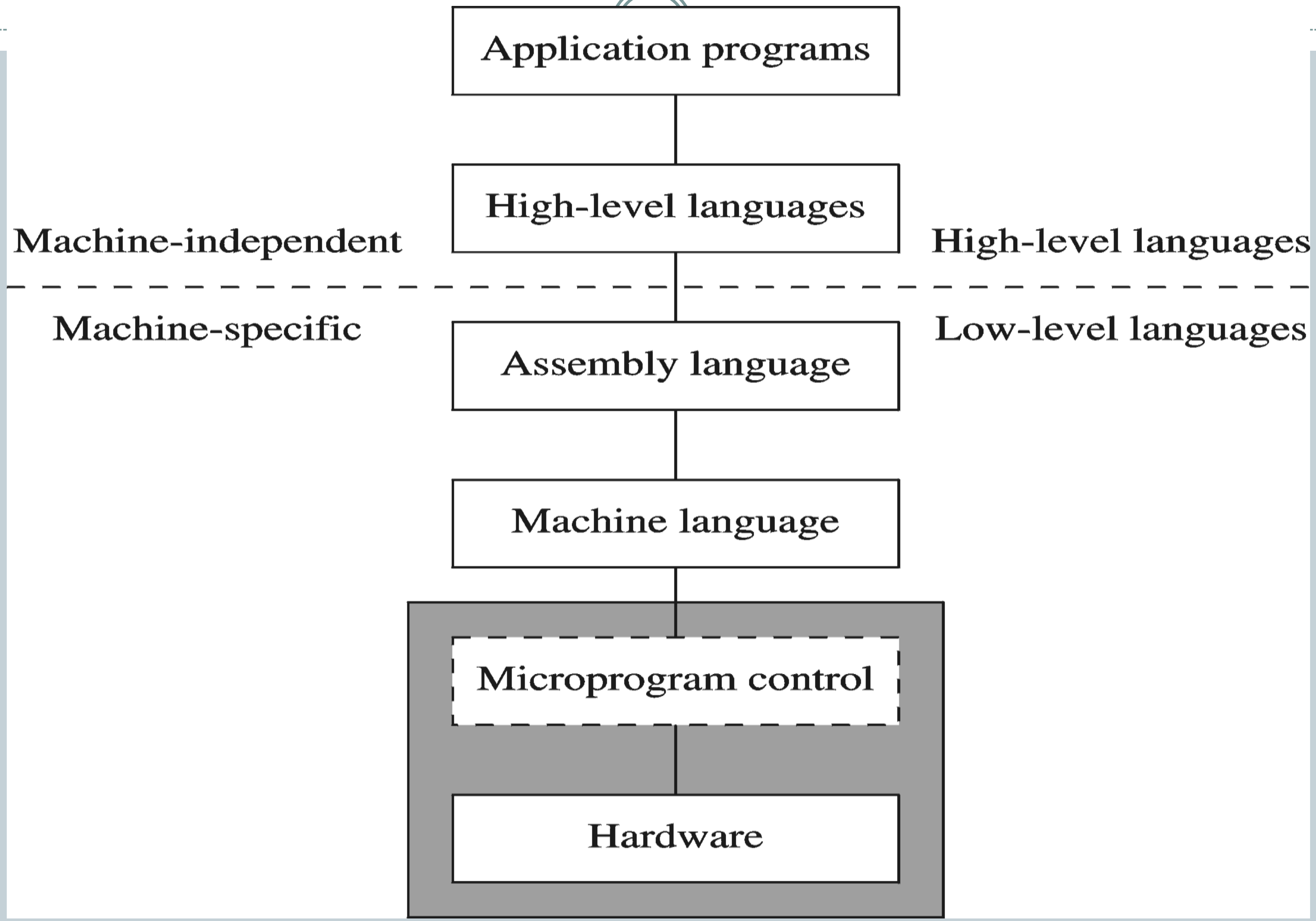
Hierarchy of Computer Languages

MACHINE-, ASSEMBLY-, AND HIGH-LEVEL LANGUAGES

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?

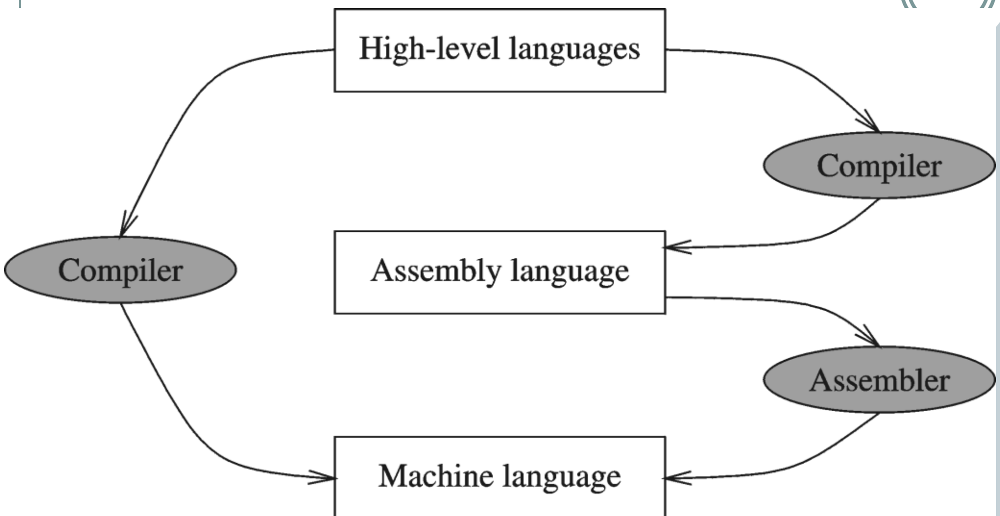
A Hierarchy of Languages



Assembly and Machine Language

- Machine language
  - Native to a processor: executed directly by hardware
  - Instructions consist of binary code: 1 s and 0 s
- 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

Compiler and Assembler



- Assemblers translate assembly to machine code
- Compilers translate high-level programs to machine code
  - Either directly, or
  - Indirectly via an assembler

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*

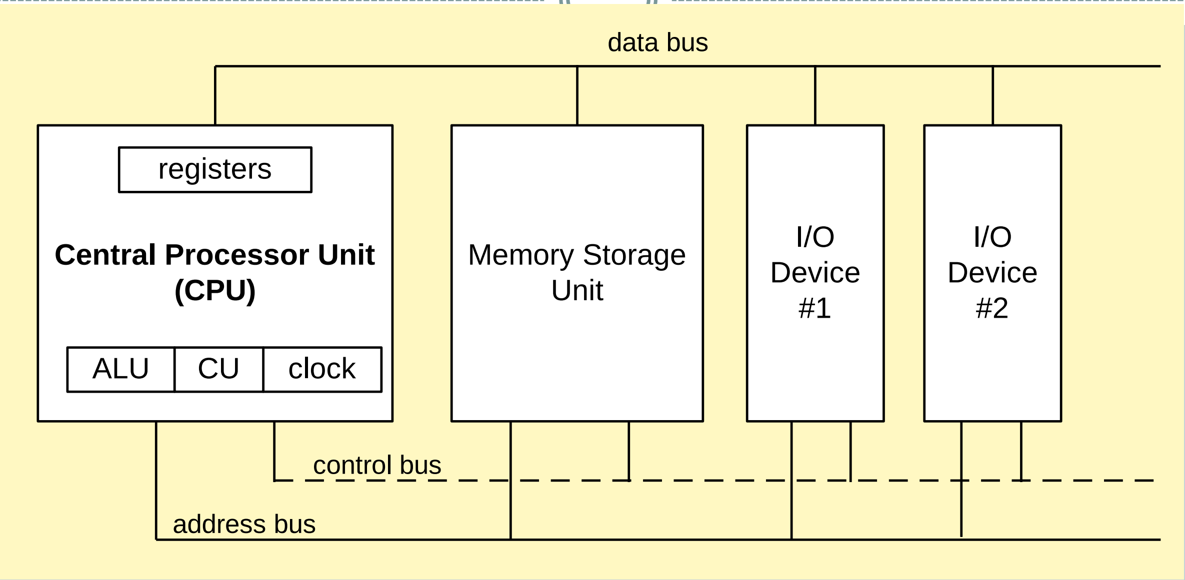
Why Learn Assembly Language?

- 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

General Concepts

BASIC MICROCOMPUTER DESIGN
INSTRUCTION EXECUTION CYCLE
READING FROM MEMORY
HOW PROGRAMS RUN

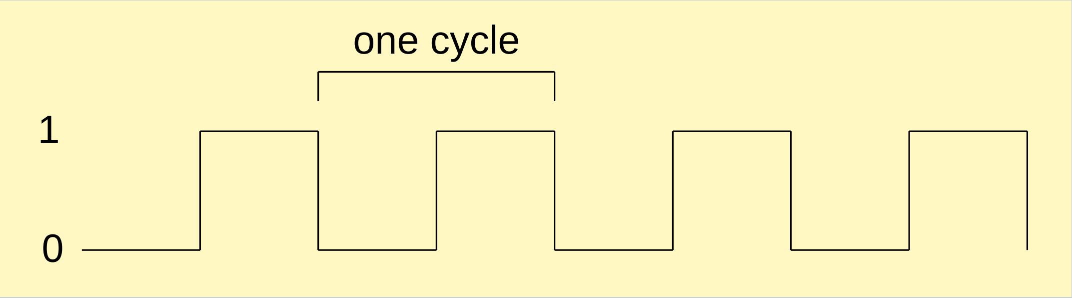
Basic Microcomputer Design



- clock synchronizes CPU operations
- control unit (CU) coordinates sequence of execution steps
- ALU performs arithmetic and bitwise processing

Clock

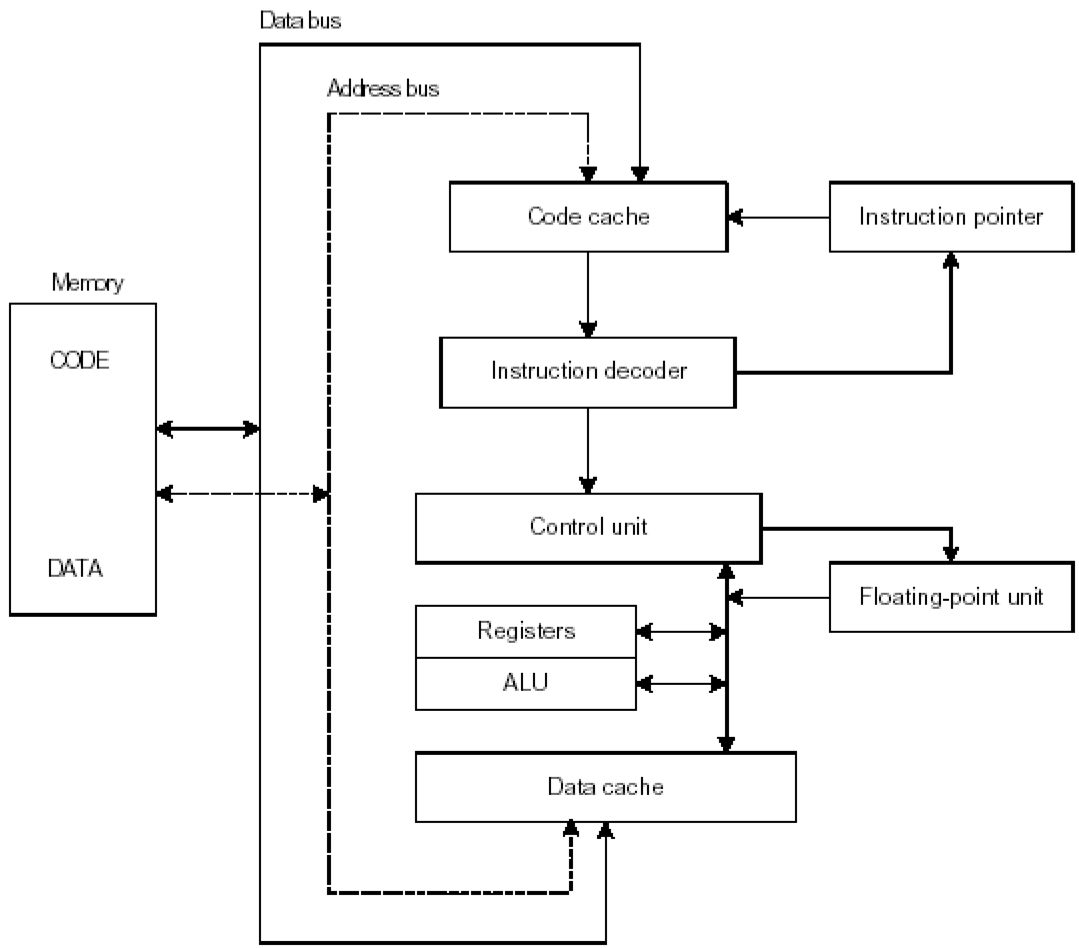
- synchronizes all CPU and BUS operations
- machine (clock) cycle measures time of a single operation
- clock is used to trigger events



Instruction Execution Cycle

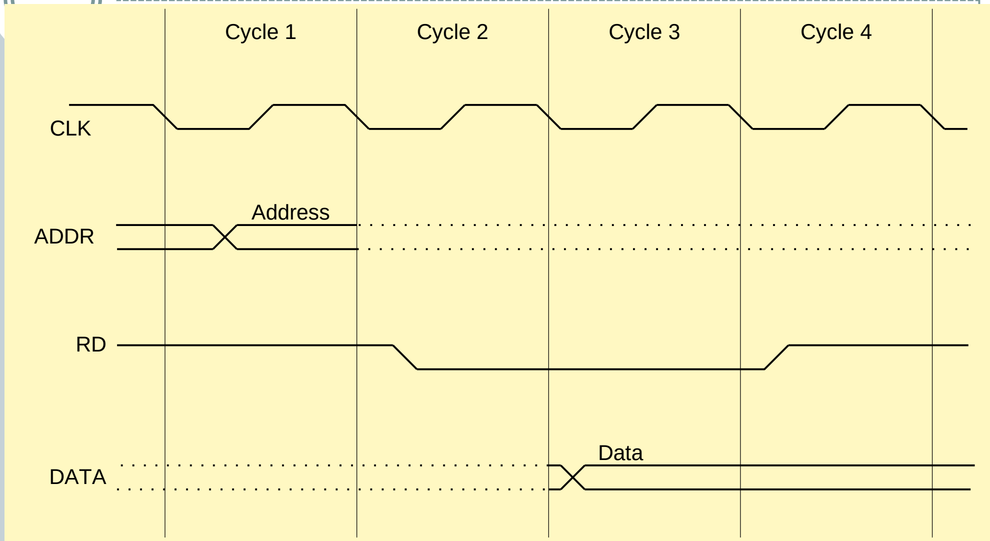
- Fetch
- Decode
- Fetch operands
- Execute
- Store output

Figure 2–2 Simplified Pentium CPU Block Diagram.



Reading from Memory

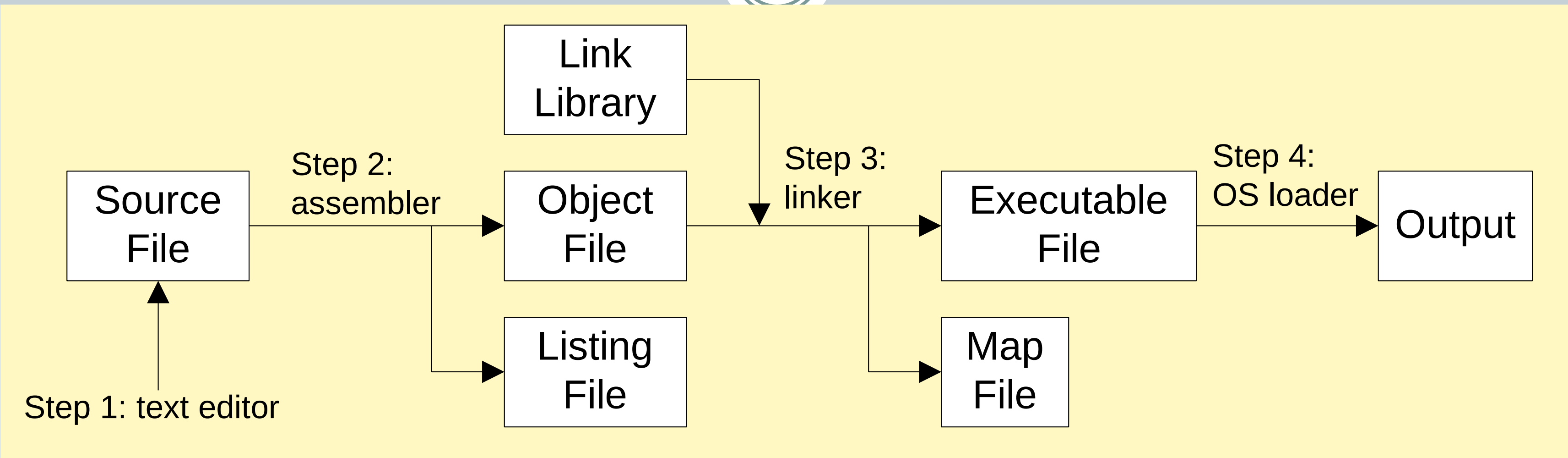
- Multiple machine cycles are required when reading from memory, because it responds much more slowly than the CPU.
- The steps are:
  - Cycle 1:
    - address placed on address bus
  - Cycle 2:
    - Read Line (RD) set low
  - Cycle 3:
    - CPU waits one cycle for memory to respond
  - Cycle 4:
    - Read Line (RD) goes to 1, indicating that the data is on the data bus



Assembling, Linking, and Running Programs

- Assemble-Link-Execute Cycle
- Listing File
- Map File

Assemble-Link Execute Cycle



- The following diagram describes the steps from creating a source program through executing the compiled program.
- If the source code is modified, Steps 2 through 4 must be repeated.

Listing File

- Use it to see how your program is compiled
- Contains
  - source code
  - addresses
  - object code (machine language)
  - segment names
  - symbols (variables, procedures, and constants)
- Example: addSub.lst

Listing File

```
Microsoft (R) Macro Assembler Version 9.00.30729.01    05/07/09
16:43:07
Add and Subtract        (AddSub.asm)         Page 1 - 1

TITLE Add and Subtract                  (AddSub.asm)

; This program adds and subtracts 32-bit integers.

INCLUDE Irvine32.inc
C .NOLIST
C .LIST

00000000                     .code
00000000                     main PROC

00000000  B8 00010000        mov      eax,10000h      ; EAX = 10000h
00000005  05 00040000        add      eax,40000h      ; EAX = 50000h
0000000A  2D 00020000        sub      eax,20000h      ; EAX = 30000h
0000000F  E8 00000000 E      call     DumpRegs

0000001B                     main ENDP
                             END main

Structures and Unions:
             Name                        Size
                                         Offset        Type
```

Listing File

```
Microsoft (R) Macro Assembler V
16:43:07
Add and Subtract        (AddSub.

TITLE Add and Subtract

; This program adds and subtr

INCLUDE Irvine32.inc
C .NOLIST
C .LIST
00000000                       .code
00000000                       main PROC

00000000  B8 00010000          mov     eax,10000h      ; EAX = 10000h
00000005  05 00040000          add     eax,40000h      ; EAX = 50000h
0000000A  2D 00020000          sub     eax,20000h      ; EAX = 30000h
0000000F  E8 00000000 E        call    DumpRegs

0000001B                       main ENDP
                               END main

Structures and Unions:
             Name                         Size
                                          Offset        Type
```

- 32-bit addresses
- indicate the relative byte distance of each statement from the beginning of the program's code area

Listing File

```
Microsoft (R) Macro Assembler Version 9
16:43:07
Add and Subtract      (AddSub.asm)

TITLE Add and Subtract                  (Ad

; This program adds and subtracts 32-bit

INCLUDE Irvine32.inc
C .NOLIST
C .LIST

00000000                       .code
00000000                       main PROC

00000000  B8 00010000          mov      eax,10000h       ; EAX = 10000h
00000005  05 00040000          add      eax,40000h       ; EAX = 50000h
0000000A  2D 00020000          sub      eax,20000h       ; EAX = 30000h
0000000F  E8 00000000 E        call     DumpRegs

0000001B                       main ENDP
                               END main

Structures and Unions:
             Name                         Size
                                          Offset        Type
```

- contain no executable instructions
- ... directives,

Listing File

- assembly language instructions, each 5 bytes long
- the hexadecimal values in the second column, such as **B8 00010000** are the actual instruction bytes.

```
Microsoft (R) Macro Assembler
16:43:07
Add and Subtract          (AddSub

TITLE Add and Subtract

; This program adds and subtra

INCLUDE Irvine32.inc
C .NOLIST
C .LIST

00000000                      .code
00000000                      main PROC

00000000   B8 00010000        mov      eax,10000h      ; EAX = 10000h
00000005   05 00040000        add      eax,40000h      ; EAX = 50000h
0000000A   2D 00020000        sub      eax,20000h      ; EAX = 30000h
0000000F   E8 00000000 E      call     DumpRegs

0000001B                      main ENDP
                              END main

Structures and Unions:
             Name                         Size
                                          Offset         Type
```

Map File

- Information about each program segment:
  - starting address
  - ending address
  - size
  - segment type
- Example:
  - addSub.map

AddSubMap.txt

```
Start  Stop   Length Name                 Class
00000H 006E2H 006E3H _TEXT                CODE
006E4H 008FDH 0021AH _DATA                DATA
00900H 028FFH 02000H STACK                STACK
02900H 02AFFH 00200H _BSS                 BSS

Origin   Group
006E:0   DGROUP

Program entry point at 0000:0000
```

Instruction Set Architecture (ISA)

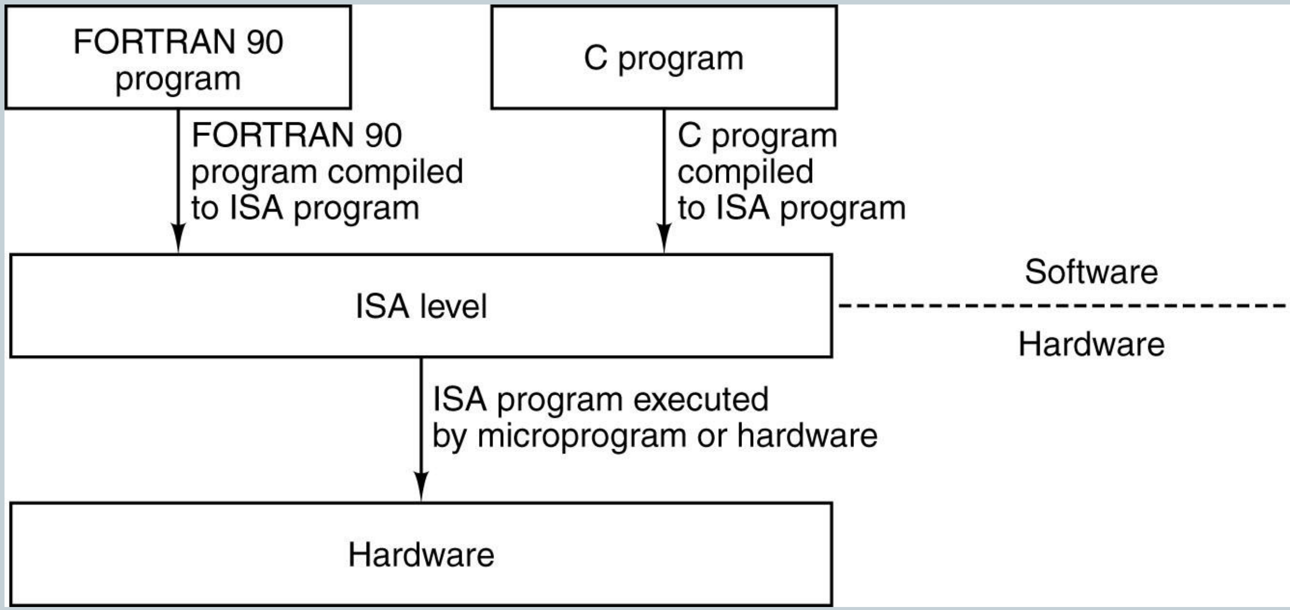
ISA LEVEL

ELEMENTS OF INSTRUCTIONS

INSTRUCTIONS TYPES

INSTRUCTIONS FORMAT

Instruction Set Architecture (ISA) Level



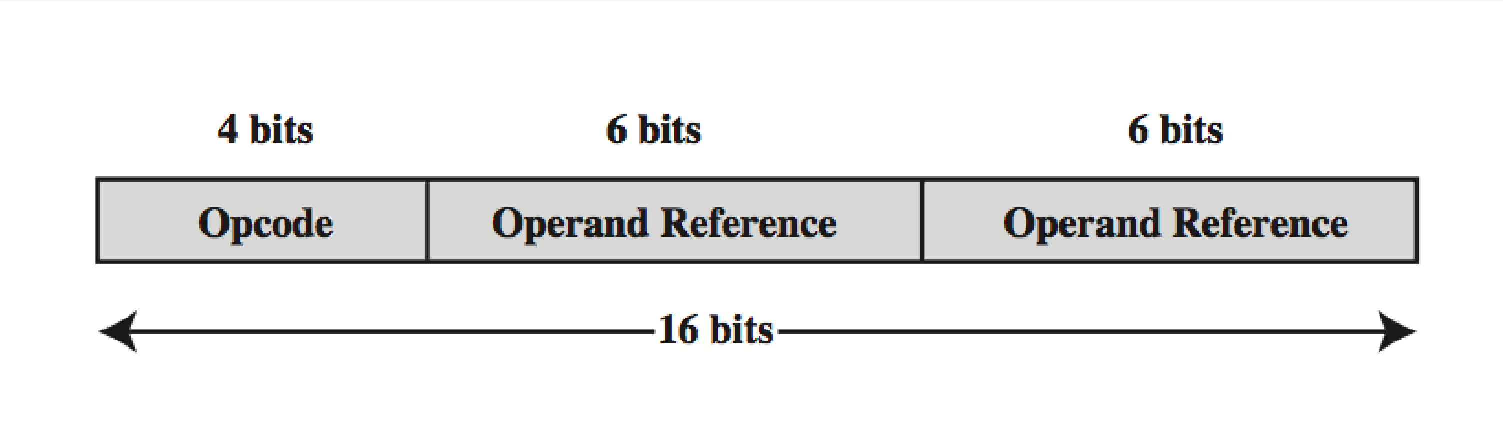
- ISA Level defines the interface between the compilers (high level language) and the hardware. It is the language that both them understand

What is an Instruction Set?

- The complete collection of instructions that are understood by a CPU
- Known also as Machine Code/Machine Instruction
- Binary representation
- Usually represented by assembly codes
- ***Programmer*** becomes aware of *registers*, *memory structure*, *data types* supported by machine and the *functioning of ALU*

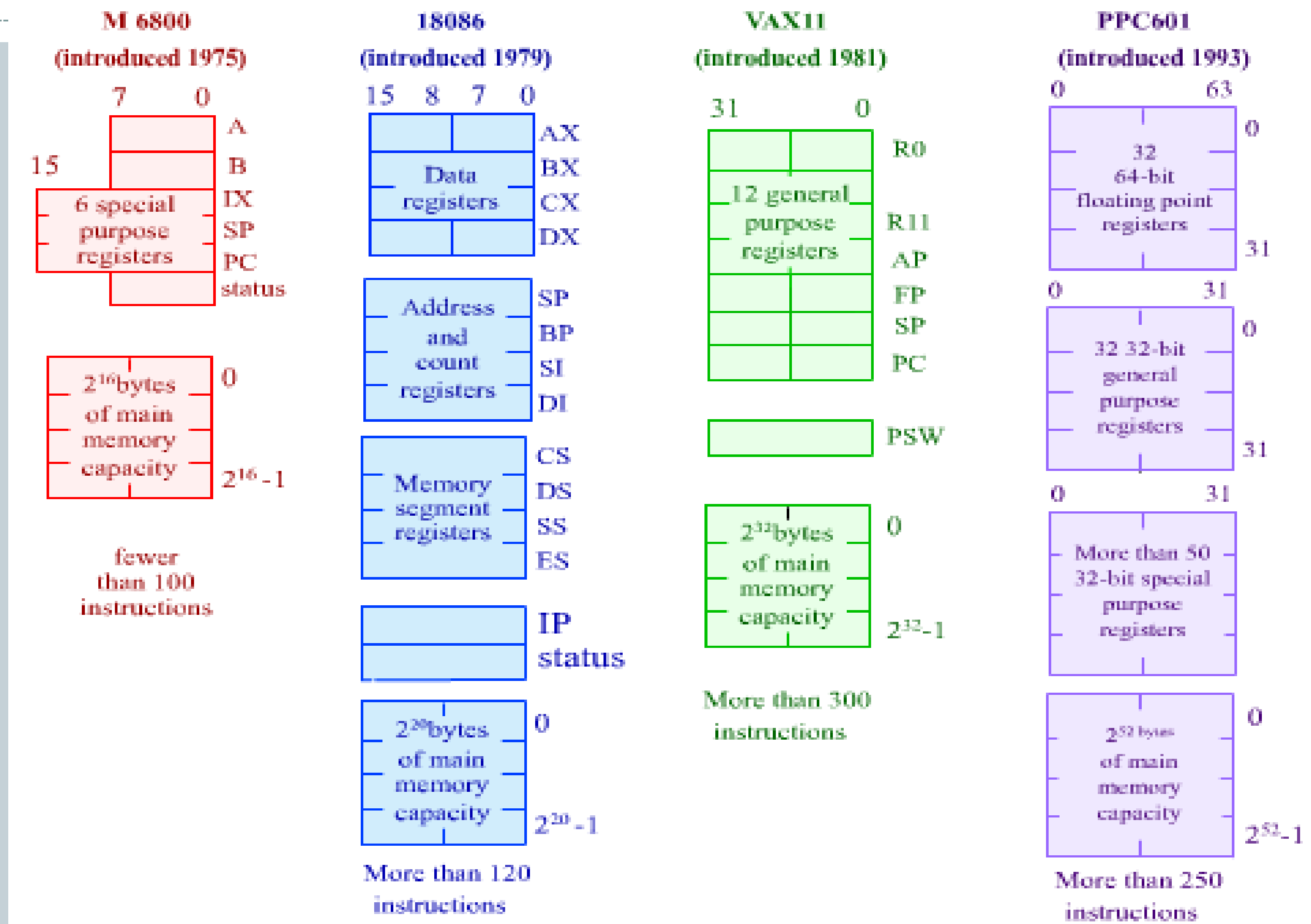
What is an Instruction Set?

- An example of instruction format (size 16-bit)



- Above instruction must fit with **register** of 16-bit size

Example of some ISA



Instruction Set Architecture (ISA)

ISA LEVEL

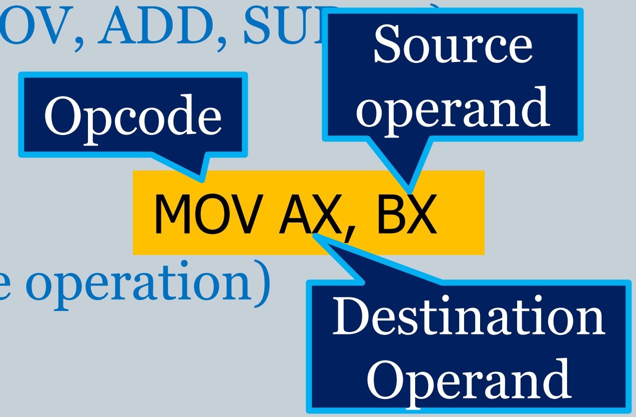
ELEMENTS OF INSTRUCTIONS

INSTRUCTIONS TYPES

INSTRUCTIONS FORMAT

Elements of an Instruction

- Operation code (Opcode)
  - Specifies the operation to be performed (MOV, ADD, SUB, …)
  - Specified as binary code know as OPCODE



- Source Operand reference
  - One or more source operands (input for the operation)
- Result (Destination) Operand reference
  - Operation produce a result (output for the operation)
  - Sometimes the result is an action, like JMP target
- Next Instruction Reference
  - Tells processor where to fetch the next instruction after the execution of current instruction is completed

.... invisible !!!

Elements of an Instruction

- Source and result operands could be:
  - **Main memory or virtual memory** – addresses is supplied for instruction references
  - **CPU registers (processor registers)** – One or more registers that can be referenced by instructions
  - **Immediate** – the value of the operand is contained in the field in the instruction executed.
  - **I/O device** – instruction specifies the I/O module and device for the operation

Elements of an Instruction

Go to the address location that holds TOTAL and get the value

Operand: memory

Current Ins.: 0000
Next Ins.: 0001

0000	MOV AX, TOTAL
0001	MOV BX, AX
0002	ADD AX, 2
0003	TARGET:
0004	CALL READINT
0005	MOV VAL, EAX
0006	ADD EBX, VAL
0007	JMP TARGET

Operand: register

Operand: immediate value

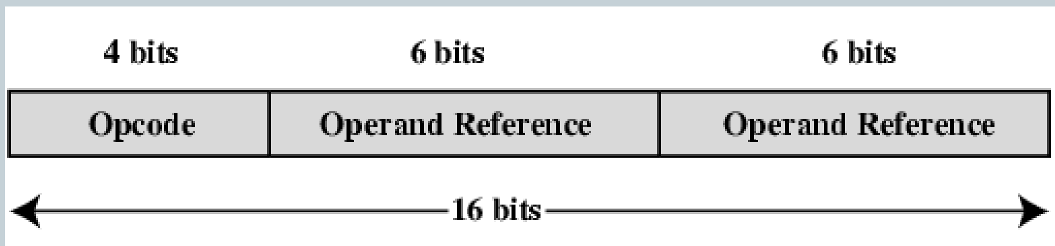
Operand: from I/O

Current Ins.: 0007
Next Ins.: 0003

Next instruction is where TARGET is located = 0003

Instruction Representation

- In machine code:
  - each instruction has a **unique bit pattern** (a sequence of bits)
  - An instruction is divided into fields and with multiple formats, e.g.;



- During instruction execution:
  - An instruction is **read into the Instruction Register (IR)** in the processor
  - The processor then extract the data and perform the required operation

Instruction Representation

- For better understanding, a *symbolic representation* is used
  - **Opcodes** represented as mnemonics, indicates the operations
    - e.g. **ADD**, **SUB**, **LOAD**
  - Difficult to deal in binary representation of machine instructions

Instruction type	Opcode	Symbolic representation	Description
Data transfer	00001010	LOAD MQ,M(X)	Transfer contents of memory location X to register MQ

What the processor see

What the programmer see

Translating Languages

- A single instructions in a High Level Language like C may require more than 1 instruction in Assembly language.
- Example : Total = Total + stuff; *// in C lang*.
  - add the value stored in Total to the value stored in stuff and put result in Total.
- In assembly language (assuming Total and stuff has been declared):
  - Load a register with the contents of memory (for Total)
  - Add the contents of memory (for stuff) to the register
  - Store the content of the register to memory location (for Total)

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 Code:

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

Mapping Between Assembly Language and High Level Language (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
- Example of mapping between some C instructions and x86 assembly language:

C	Assembly Language
a = 5	MOV a, 5

b = a + 5	MOV ax, a
	ADD ax, 5
	MOV b, ax

goto LBL	JMP LBL

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

e.g. MOV actual format

- actual MOV instructions defined in 80x86 consists of **32** different machine instructions (or 'opcode')
- next slide will show just part of it.

e.g. MOV actual format

Mnemonic	Operand(s)	Flags affected	Opcode	Number of Bytes	Timing 386	Timing 486	Timing Pentium
mov	AL, imm8	none	B0	2	2	1	1
mov	CL, imm8	none	B1	2	2	1	1
mov	DL, imm8	none	B2	2	2	1	1
mov	BL, imm8	none	B3	2	2	1	1
mov	AH, imm8	none	B4	2	2	1	1
mov	CH, imm8	none	B5	2	2	1	1
mov	DH, imm8	none	B6	2	2	1	1
mov	BH, imm8	none	B7	2	2	1	1

e.g. MOV actual format

Mnemonic	Operand(s)	Flags affected	Opcode	Number of Bytes	Timing 386	Timing 486	Timing Pentium
mov	AX, imm16 EAX, imm32	none	B8	3 5	2	1	1
mov	CX, imm16 ECX, imm32	none	B9	3 5	2	1	1
mov	DX, imm16 EDX, imm32	none	BA	3 5	2	1	1
mov	BX, imm16 EBX, imm32	none	BB	3 5	2	1	1
mov	SP, imm16 ESP, imm32	none	BC	3 5	2	1	1
mov	BP, imm16 EPB, imm32	none	BD	3 5	2	1	1

e.g. MOV actual format

Mnemonic	Operand(s)	Flags affected	Opcode	Number of Bytes	Timing 386	Timing 486	Timing Pentium
mov	AL, direct	none	A0	5	4	1	1
mov	AX, direct EAX, direct	none	A1	5	4	1	1
mov	reg8,mem8	none	8A	2+	4	1	1
mov	reg16,mem16 reg32,mem32	none	8B	2+	4	1	1
mov	mem8,reg8	none	88	2+	2	1	1
mov	mem16,reg16 mem32,reg32	none	89	2+	2	1	1
mov	direct ,AL	none	A2	5	2	1	1
mov	direct, AX direct, EAX	none	A3	5	2	1	1

… end of Part 1

- **Hierarchy of Computer Languages**
- **General Concepts**
- **ISA Level**
- **Elements of Instructions**
- Instructions Types
- Instruction Formats
- Number of Addresses
- Registers
- Types of Operands
- Addressing Modes

Part 1