Assembly Language

Lecture 2 - x86 Processor Architecture

Ahmed Sallam

Outcomes of Lecture 1

- *Introduction to the course*
  - Always check the course website
  - Don‘t forget the deadline rule!!
- *Motivations for studying assembly*
  - Full control
  - Efficiency
- *Different layers of programming languages*
- *Data representation: Base 2 and Base 16*
- *Boolean operations and algebra*

Outline

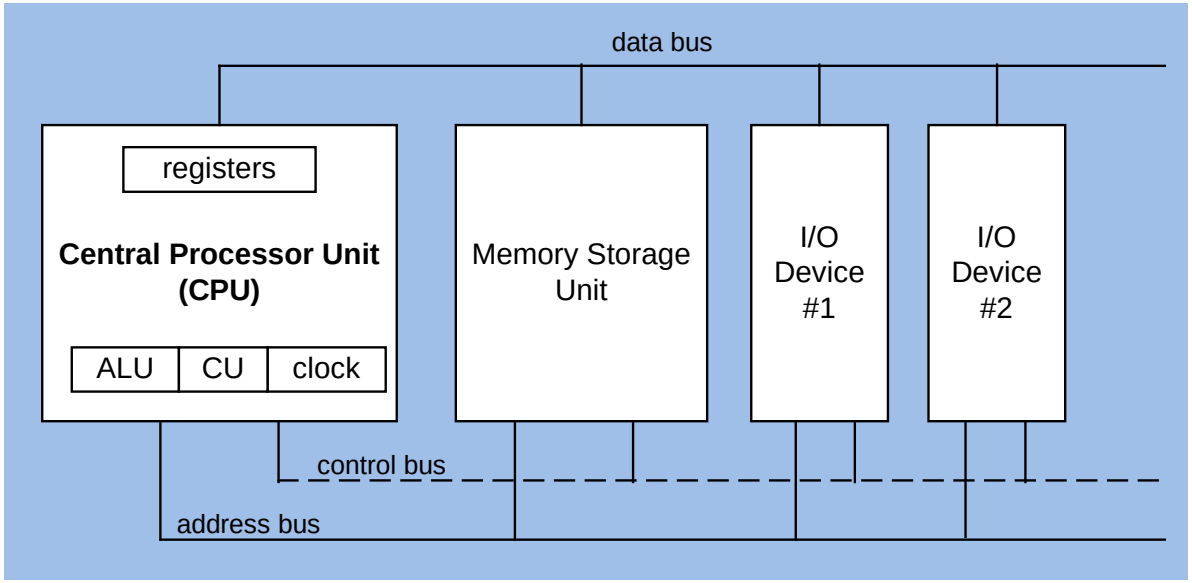
- *General Concepts*
- *IA-32 Processor Architecture*
- *IA-32 Memory Management*
- *Input-Output System*

Outline

- ***General Concepts***
- *IA-32 Processor Architecture*
- *IA-32 Memory Management*
- *Input-Output System*

Basic Computer Design

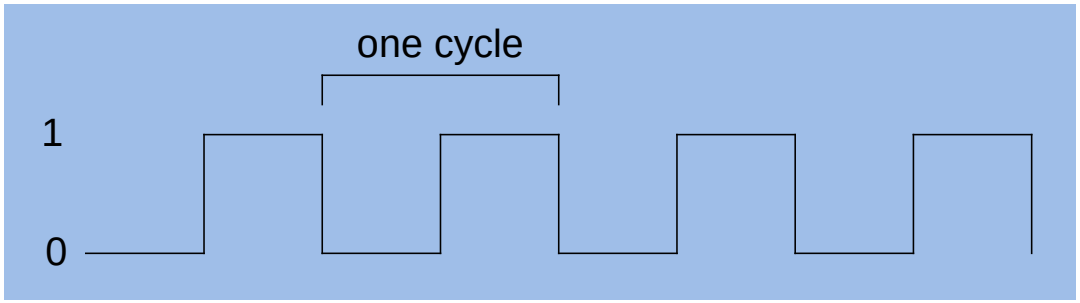
- *Clock synchronizes CPU operations*
- *Control unit (CU) coordinates sequence of execution steps*
- *ALU performs arithmetic and bitwise processing*



5

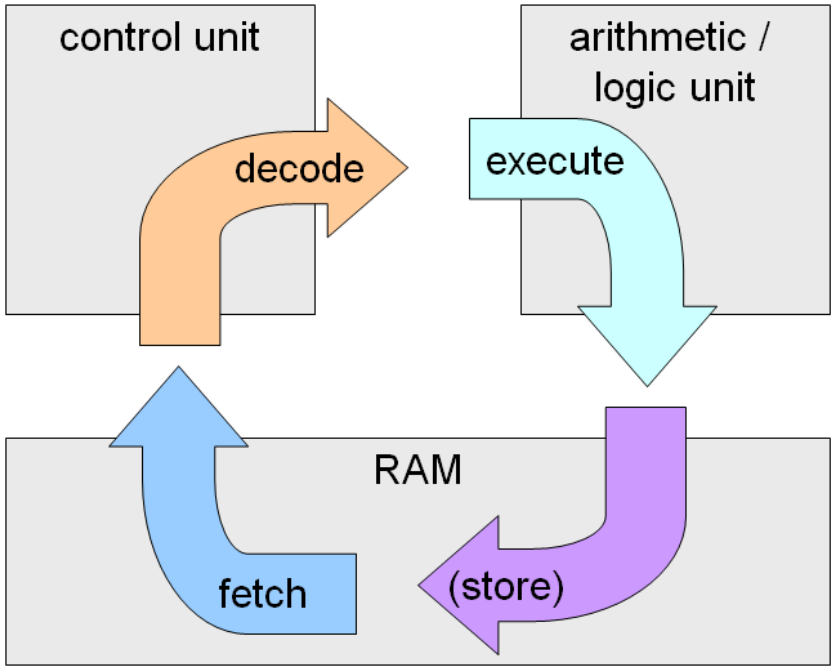
Clock

- *Synchronizes all CPU and bus operations*
- *Clock cycle measures time of a single operation*
- *A machine instruction requires at least one clock cycle to execute,*
  - a few require in excess of 50 clocks (the multiply instruction on the 8088 processor)
- *Instructions requiring memory access often have empty clock cycles (wait states)*
  - differences in speeds of the CPU, the system bus, and memory circuits.



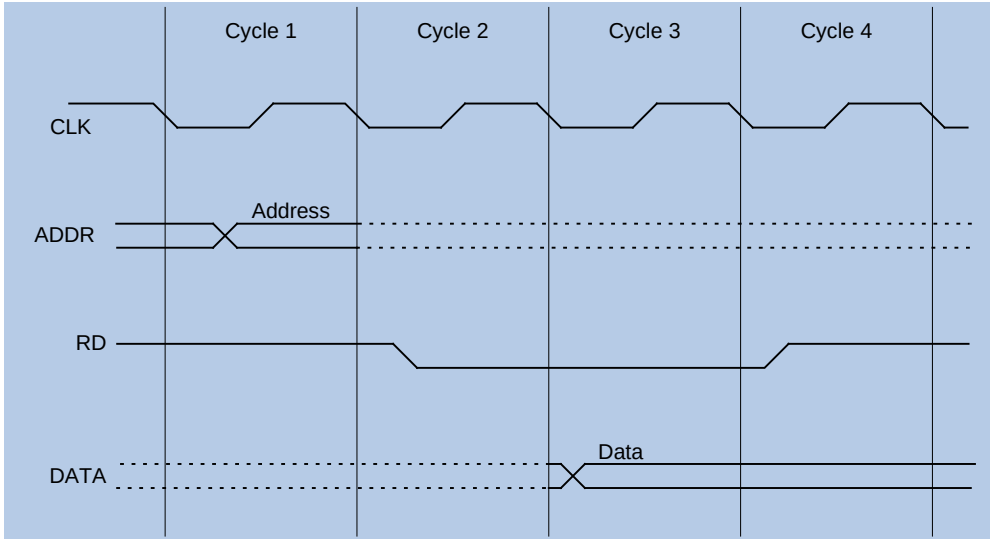
Instruction Execution Cycle

- *Fetch instruction*
- *Decode*
- *Fetch operands*
  - Memory → registers (internal)
- *Execute*
- *Store output*



Reading from Memory

- *Multiple cycles are required when reading from memory, because it responds much more slowly than the CPU*
- *The steps are:*
  - address placed on address bus
  - Read Line (RD) set low (0)
  - CPU waits one cycle for memory to respond
  - Read Line (RD) goes to 1, indicating that the data is on the data bus



Cache Memory

- *High-speed expensive static RAM both inside and outside the CPU*
- *The first time a program reads a block of data, it leaves a copy in the cache.*
  - If the program needs to read the same data a second time, it looks for the data in cache.
- ***Cache hit:*** *when data to be read is already in cache memory*
- ***Cache miss:*** *when data to be read is not in cache memory*

Multitasking

- *OS can run multiple programs at the same time*
- *Multiple threads of execution within the same program*
- *Scheduler utility assigns a given amount of CPU time to each running program*
- *Rapid switching of tasks*
  - Provides illusion that all programs are running at once
  - CPU must support task switching (context switching)

Execution: OS Role

- *When you **executes** a program, the OS does many things for you*
  - Locates the binary executable file on disk
  - Calls the "loader" to move the file from disk to memory and resolve all addresses if needed
  - Locates and links any runtime DLLs
- ***During execution** the OS*
  - Resolves all addresses
  - Allocates runtime (time-slice for multitasking)
  - Manages IO requests with devices
- *Cleans up **after execution** is over*

Review Questions

1. *The central processor unit (CPU) contains registers and what other basic elements?*
2. *The central processor unit is connected to the rest of the computer system using what three buses?*
3. *Why does memory access take more machine cycles than register access?*
4. *What are the three basic steps in the instruction execution cycle?*
5. *Define multitasking?*
6. *What is the function of the OS scheduler?*

Outline

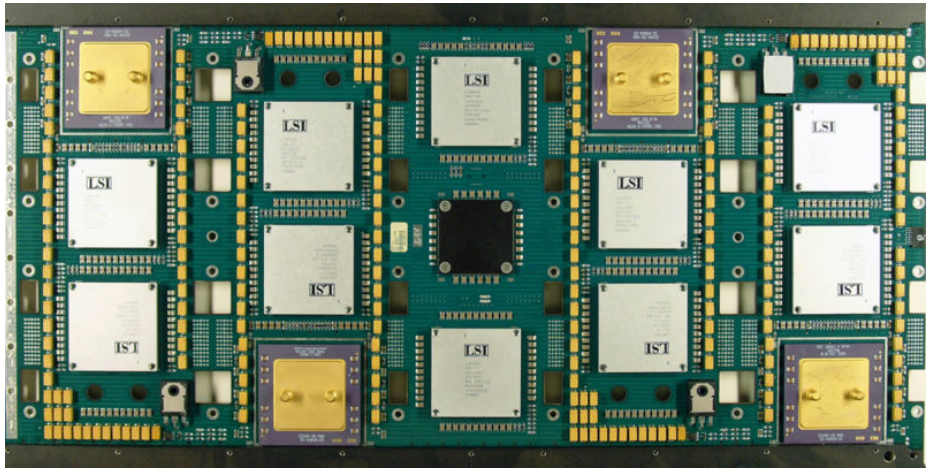
- *General Concepts*
- ***IA-32 Processor Architecture***
- *IA-32 Memory Management*
- *Input-Output System*

Intel CPU History

- *Intel 8086 (1978)/8088 (1979)*
  - First IBM-PC Used 8088 (1980)
  - 1 MB addressable RAM
  - ***16-bit registers***
  - **16-bit data bus** (8-bit for 8088)
  - $3000 - $6000 USD (8088)

History of the Intel IA-32 Family

- *Intel 80386 (1985)*
  - 4 GB addressable RAM, 32-bit registers, paging (virtual memory)
- *Intel 80486 (1989)*
  - instruction pipelining
- *Pentium – P5 (1993)*
  - superscalar (Multiple ALU)



64-bit Processors

- *Intel64 Mode*
  - 64-bit linear address space
  - Intel: Pentium Extreme, Xeon, Celeron D, Pentium D, Core 2, and Core i3, i5, i7
- *IA-32e Mode (2 Sub-modes)*
  - ***Compatibility mode*** for legacy 16- and 32-bit applications
  - ***64-bit Mode*** uses 64-bit addresses and operands

Technologies

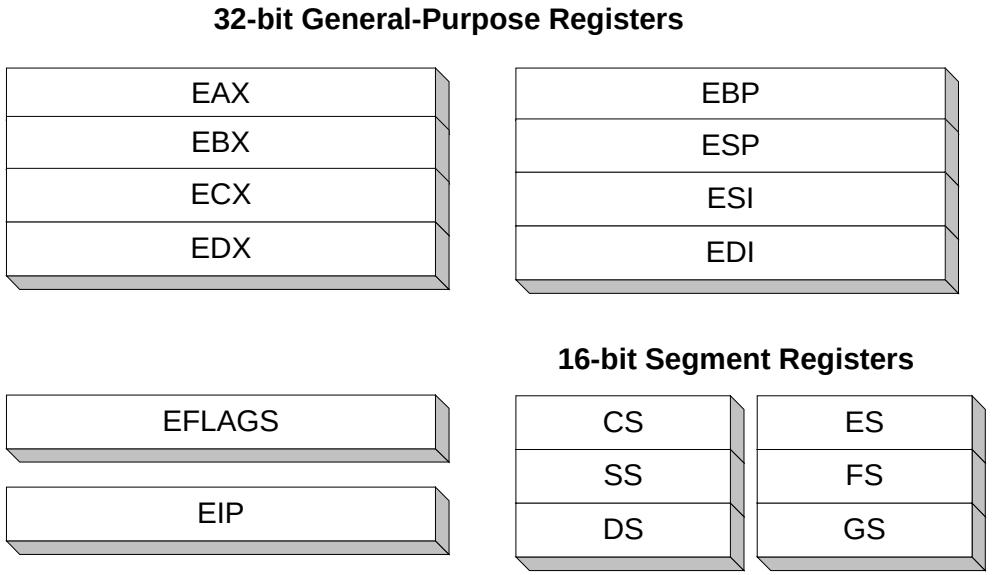
- ***Pipelined***
  - simultaneous staggered execution of instructions
- ***SuperScalar***
  - multiple redundant functional units (e.g. ALUs)
- ***HyperThreading***
  - Two tasks execute on a single processor at the same time
  - Duplicate architectural state components (registers)
- ***Dual Core processing***
  - multiple processor cores in the same IC package
  - each processor has its own resources

CISC and RISC

- *CISC – complex instruction set computing (Intel)*
  - large instruction set
  - high-level complex operations (fewer instructions)
  - requires microcode interpreter
- *RISC – reduced instruction set computing*
  - simple, atomic instructions (directly executed by hardware)
  - small instruction set
  - examples: ARM (Advanced RISC Machines)

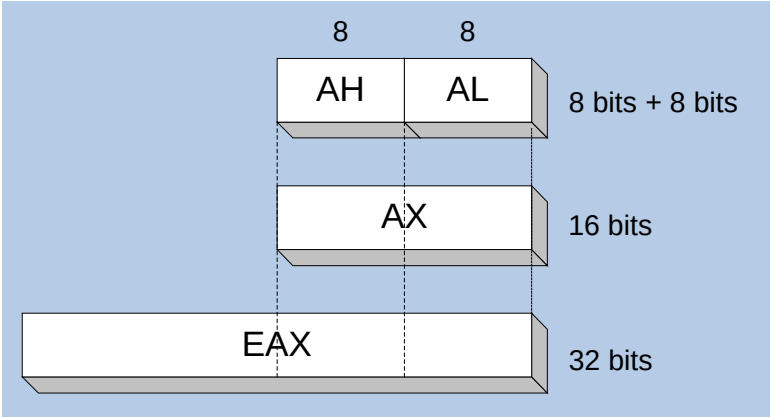
Basic Program Execution Registers

- *Registers are high-speed storage locations directly inside the CPU*
  - designed to be accessed at much higher speed than conventional memory
- *Types of registers*
  - General purpose registers
  - Segment registers
  - Processor status flags register
  - Instruction pointer



General Purpose Registers

- *These registers are primarily used for arithmetic and data movement.*
- *Use 8-bit name, 16-bit name, or 32-bit name*
- *Applies to EAX, EBX, ECX, and EDX*



32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

Index & Base Registers

- *Remaining general purpose registers have only a 16-bit name for their lower half:*

32-bit	16-bit	
ESI	SI	Source Index
EDI	DI	Destination Index
EBP	BP	Stack Base Pointer
ESP	SP	Stack Pointer

Intended Register Use

- *General-Purpose*

▪ EAX – accumulator	▪ ECX – loop counter
▪ ESP – stack pointer (TOS)	▪ ESI, EDI – index registers
▪ EBP – extended (stack) frame (base) pointer	

- *Segment*

▪ CS – code segment	▪ DS – data segment
▪ SS – stack segment	

Specialized Registers

- *EIP – instruction pointer*
  - the address of the next instruction to be executed
- *EFLAGS*
  - status and control flags
  - each flag is a single binary bit

Status Flags

- *Carry: unsigned arithmetic out of range*
- *Overflow: signed arithmetic out of range*
- *Sign: result is negative*
- *Zero: result is zero*

Example: Zero Flag

```
                        ; initially, assume ZF = 0
mov   AL,55H            ; ZF is still zero
sub   AL,55H            ; result is 0
                        ; ZF is set (ZF = 1)
mov   CX,0              ; ZF remains 1
inc   CX                ; result is 1
                         ; ZF is cleared (ZF = 0)
```

Summary: Registers

32-Bit registers	Name	16- and 8-bit sub-registers	Brief description and/or primary use
eax	Accumulator	ax,ah,al	Arithmetic and logic
ebx	Base	bx,bh,bl	Arrays
ecx	Counter	cx,ch,cl	Loops
edx	Data	dx,dh,dl	Arithmetic
esi	Source index	si	Strings and arrays
edi	Destination index	di	Strings and arrays
esp	Stack pointer	sp	Top of stack
ebp	Base pointer	bp	Stack base
eip	Instruction pointer	ip	Points to next instruction
eflags	Flag	flags	Status and control flags

Review Questions

1. *What are the x86 processor's three basic modes of operation?*
2. *What special purpose does the ECX register serve?*
3. *Besides the stack pointer (ESP), what other register points to variables on the stack?*
4. *Which flag is set when the result of an unsigned arithmetic operation is too large to fit into the destination?*
5. *Which Intel processor first introduced superscalar execution?*
6. *Describe the RISC design approach?*

Outline

- *General Concepts*
- *IA-32 Processor Architecture*
- ***IA-32 Memory Management***
- *Input-Output System*

x86 Memory Management

- ***Protected*** *mode*
  - native mode → Windows, Linux
  - Programs are given separate memory areas named ***segments***
  - processor prevents programs from referencing memory outside their assigned segments.
- ***Real-address*** *mode*
  - native mode → MS-DOS
  - direct access to system memory and hardware devices: can cause the OS to crash.
- ***System management*** *mode*
  - power management, system security, diagnostics

- Virtual-8086 mode
  - Special case of Protected (each program has its own 8086 computer)
  - In short, virtual 8086 mode is whereby the CPU (in protected mode) is running a "Emulated" (real mode) machine, e.g. cmd command.

Addressable Memory

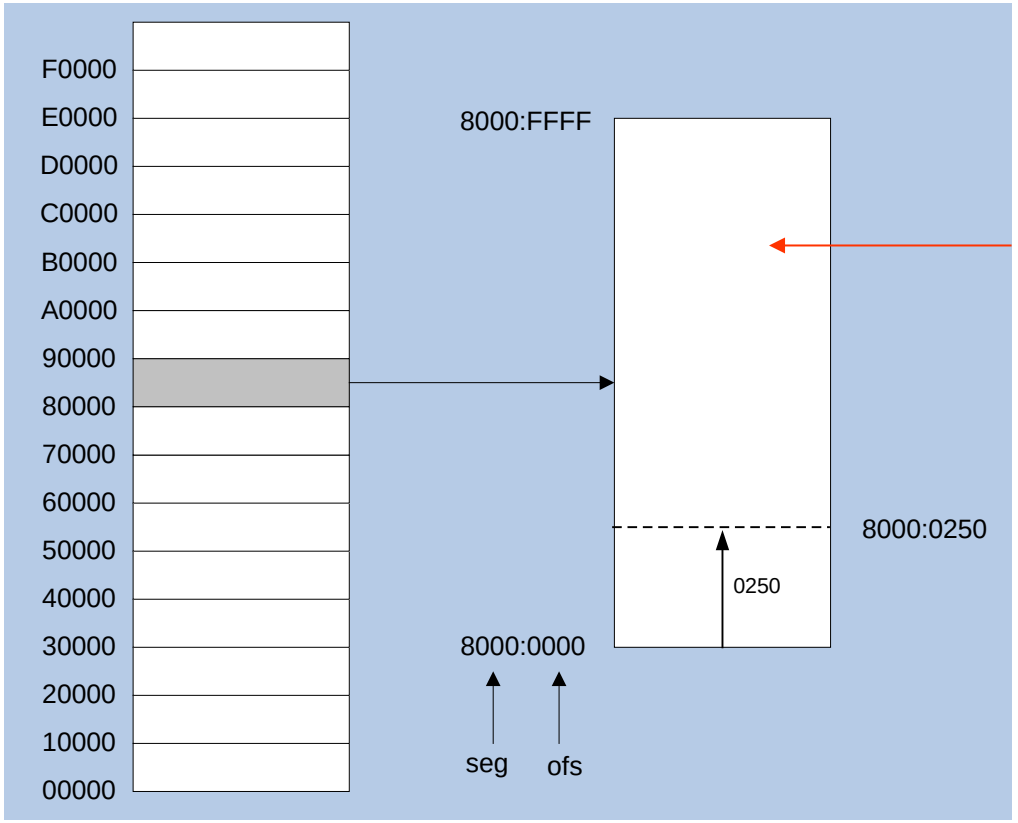
- *Protected mode*
  - Memory segment up to 4 GB
  - 32-bit address
- *Real-address and Virtual-8086 modes*
  - 1 MB space
  - 20-bit address

Real-Address Mode

- ***1 MB*** *RAM maximum addressable*
- *Applications can access* ***any area*** *of memory*
- ***Single*** *tasking (one program at a time)*
- *Supported by (obsolete) MS-DOS operating system*
- *1 Mb =* 2^{20} *thus* ***20 bit address***
- ***16 bit registers****, thus requiring* ***segmented*** *memory*

Segmented Memory

- ***Segmented** memory addressing:*
  - 16 64-kilobyte segments
  - absolute (linear) address is a combination of a 16-bit segment value added to a 16-bit offset



Calculating Linear Addresses

- *Given a segment address:*
  1. Multiply it by 16 (add a hexadecimal zero), and
  2. add it to the offset
- *Example: convert 08F1:0100 to a linear address*

```
Adjusted Segment value: 0 8 F 1 0
Add the offset:           0 1 0 0
Linear address:         0 9 0 1 0
```

Program Segments

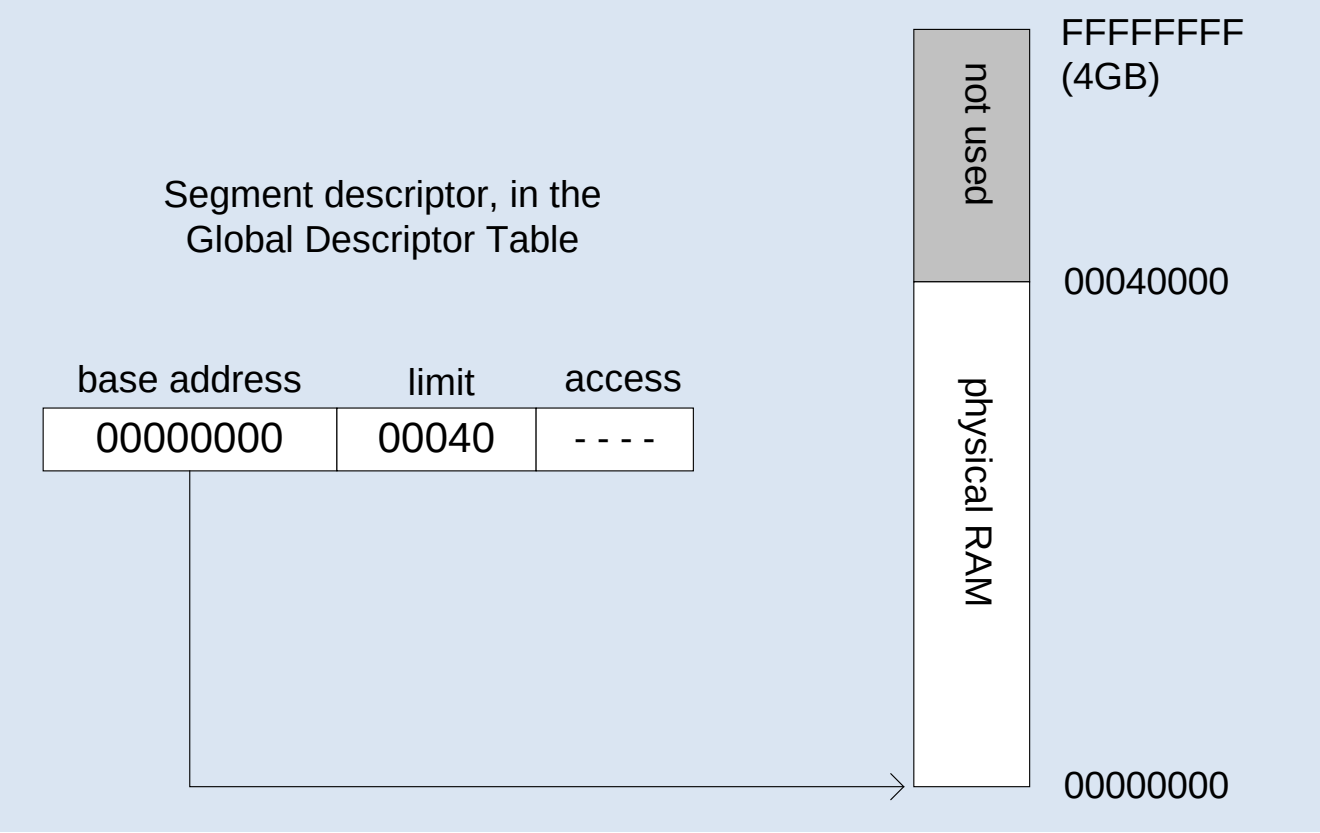
- *Always use hexadecimal notation for addresses*
- *A program has three segments: code, data, and stack*
  - Three segment registers, CS, DS, and SS, contain the segments' base locations.
  - ES, FS, and GS can point to alternate data segments supplement the default data segment

Protected Mode

- *4 GB addressable RAM (00000000h to FFFFFFFFh)*
- *Each program assigned a protected memory partition*
- *Designed for multitasking*
  - Supported by Linux & MS-Windows
- *Memory models*
  - Flat Segment Model
  - Multi Segment Model
  - Paging
- *MASM Programs use the Microsoft **flat memory model***

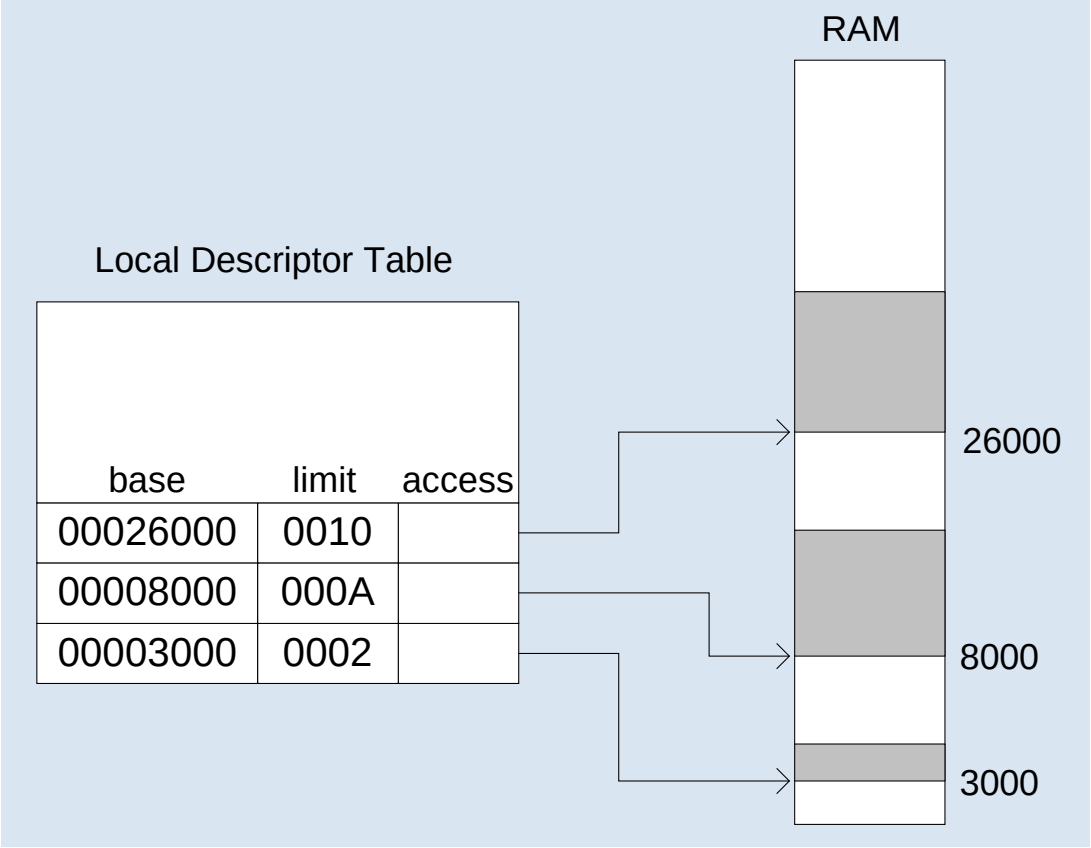
Flat Segment Model

- *Single global descriptor table (GDT) managed by OS*
- *All segments mapped to entire 32-bit address space*
- *Modern operating systems based on x86 use the flat model.*



Multi-Segment Model

- *Each program has a local descriptor table (LDT) to hold descriptor for each segment used by the program*
- *We won't use this, just need to know it exists*



Paging

- *Supported directly by the CPU*
- *Divides each segment into 4096-byte (4 Kb) blocks called pages*
- *Sum of all programs can be larger than physical memory*
- *Part of running program is in memory, part is on disk*
- ***Virtual memory manager** (VMM) – OS utility that manages the loading and unloading of pages*
- ***Page fault** – issued by CPU when a page must be loaded from disk*

Review Questions

1. *What is the range of addressable memory in protected mode and real-address mode?*
2. *In flat segmentation model, how many bits hold the address of an instruction or variable?*
3. *In the flat segmentation model, which table contains pointers to at least two segments?*
4. *In protected mode, which register references the descriptor for the stack segment?*

Outline

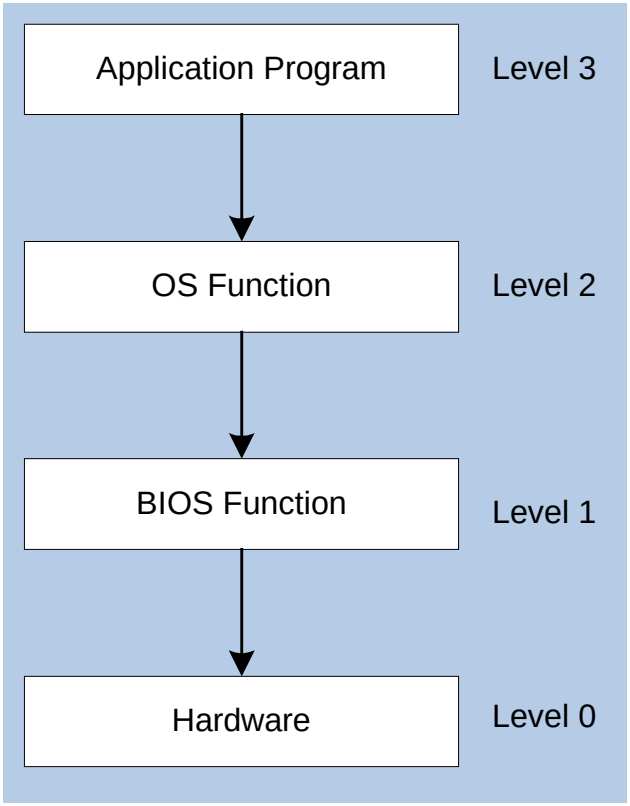
- *General Concepts*
- *IA-32 Processor Architecture*
- *IA-32 Memory Management*
- *Basic components (self reading)*
- ***Input-Output System***

Levels of Input-Output

- *Level 3: High-level language function*
  - examples: C++, Java
  - portable, convenient, not always the fastest
- *Level 2: Operating system*
  - Application Programming Interface (API)
  - extended capabilities, lots of details to master
- *Level 1: BIOS (Basic Input-Output System)*
  - drivers that communicate directly with devices
  - OS security may prevent application-level code from working at this level

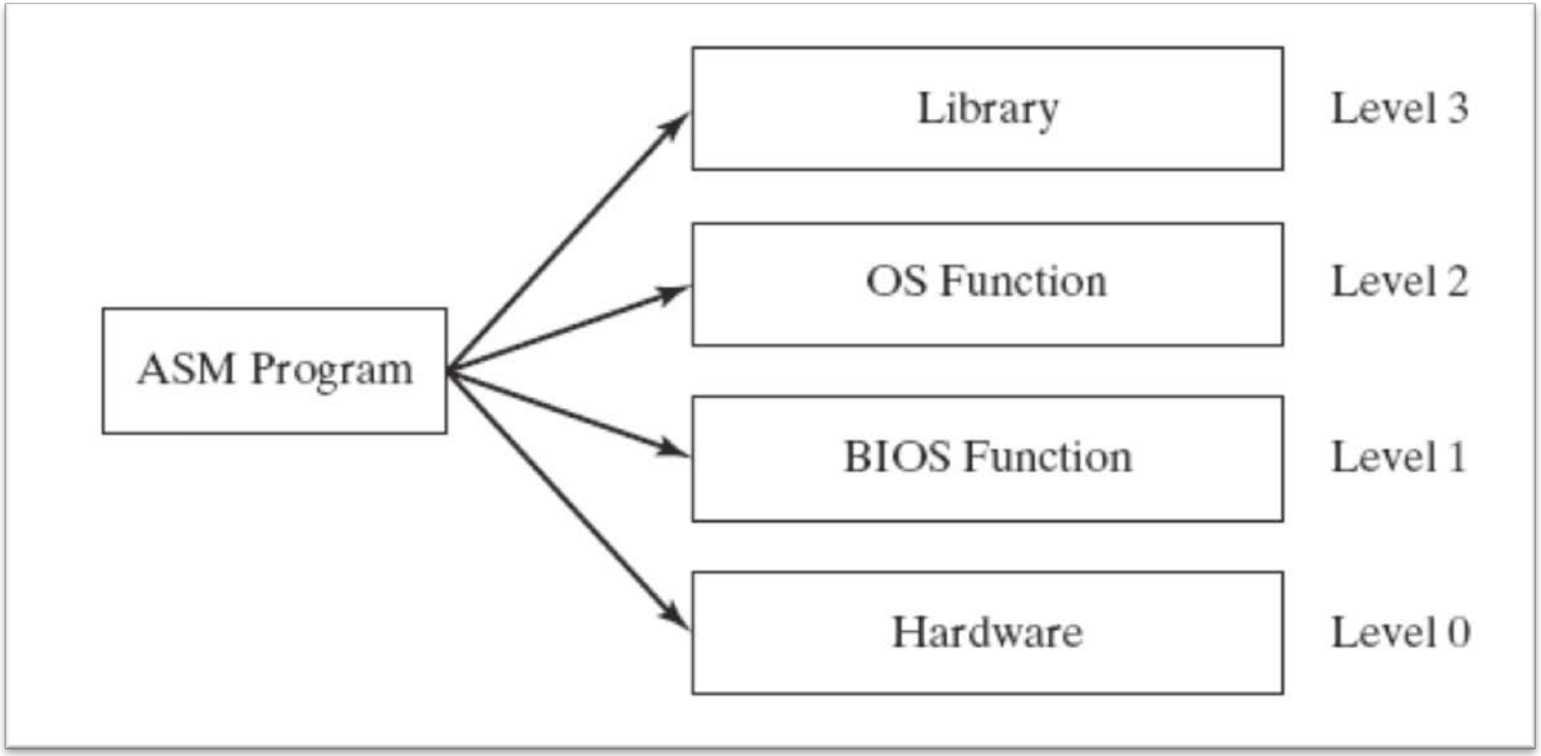
Displaying a String of Characters

- *When a HLL program displays a string of characters, the following steps take place:*



Programming levels

- *Assembly language programs can perform input-output at each of the following levels:*



Review Questions

1. *Of the four levels of input/output in a computer system, which is the most universal and portable?*
2. *What characteristics distinguish BIOS-level input/output?*
3. *Why are device drivers necessary, given that the BIOS already has code that communicates with the computer's hardware?*
4. *Is it likely that the BIOS for a computer running MS-Windows would be different from that used by a computer running Linux?*

Summary

- *General Concepts*
  - CPU Design, Instruction execution cycle
- *IA-32 Processor Architecture*
  - Modes of operations, CPU Registers & Flags, Intel CPU History
- *IA-32 Memory Management*
  - Real address mode, segmented memory, paging
- *Input-Output System*
  - Levels of input / output system