ECE291

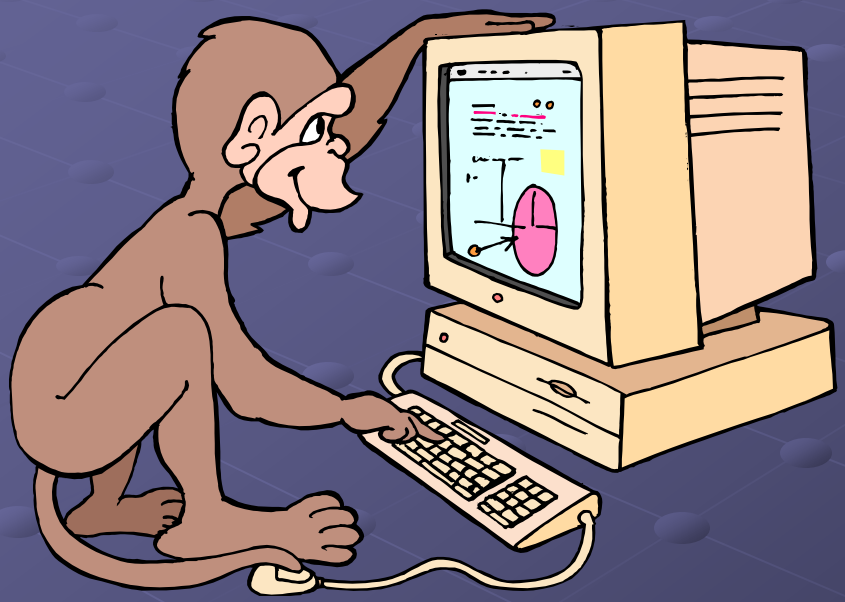
Lecture 1

All about computers

Lecture outline

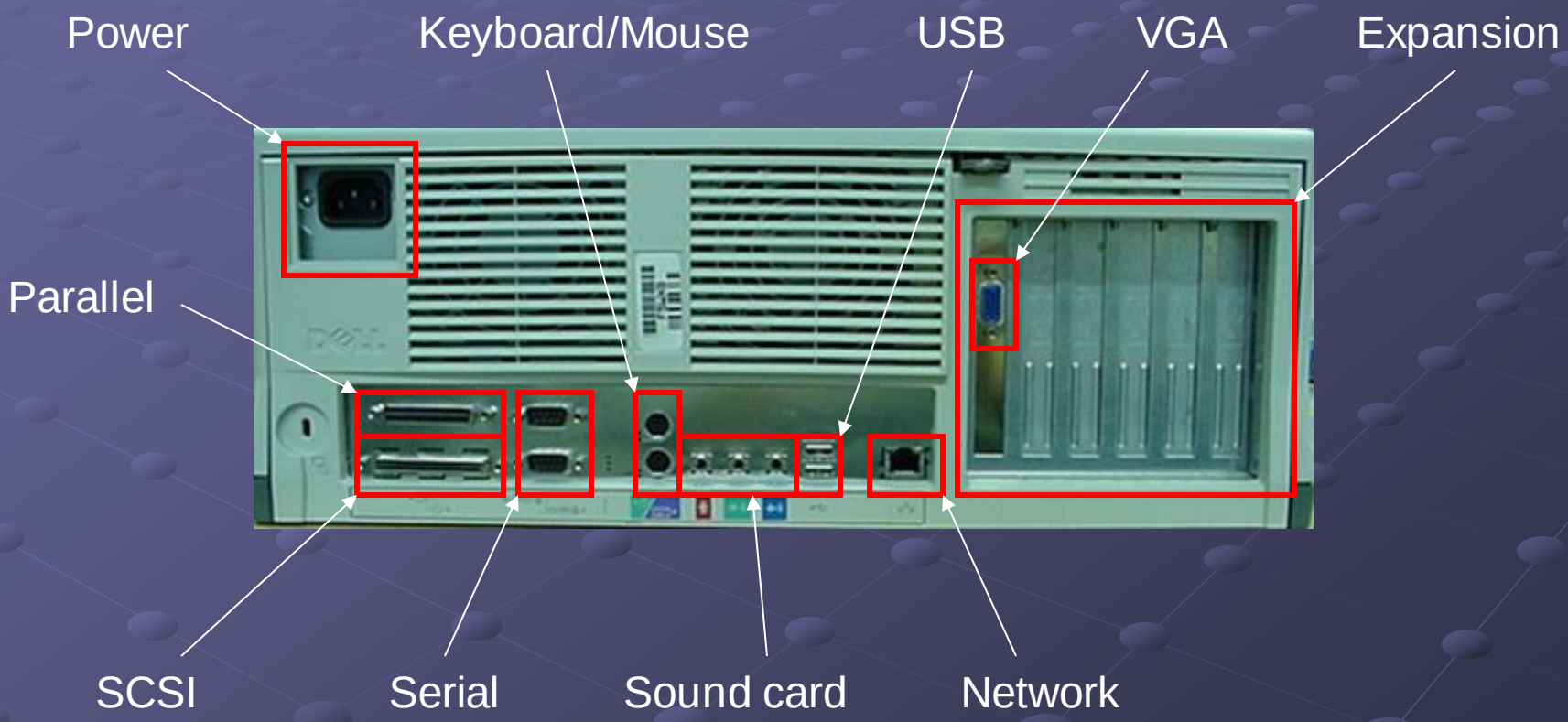
- Top-down view of computers
- Processor architecture
- Registers
- Memory
- Peripherals
- Assembly primer
- Assignments

Computers from 50,000 feet

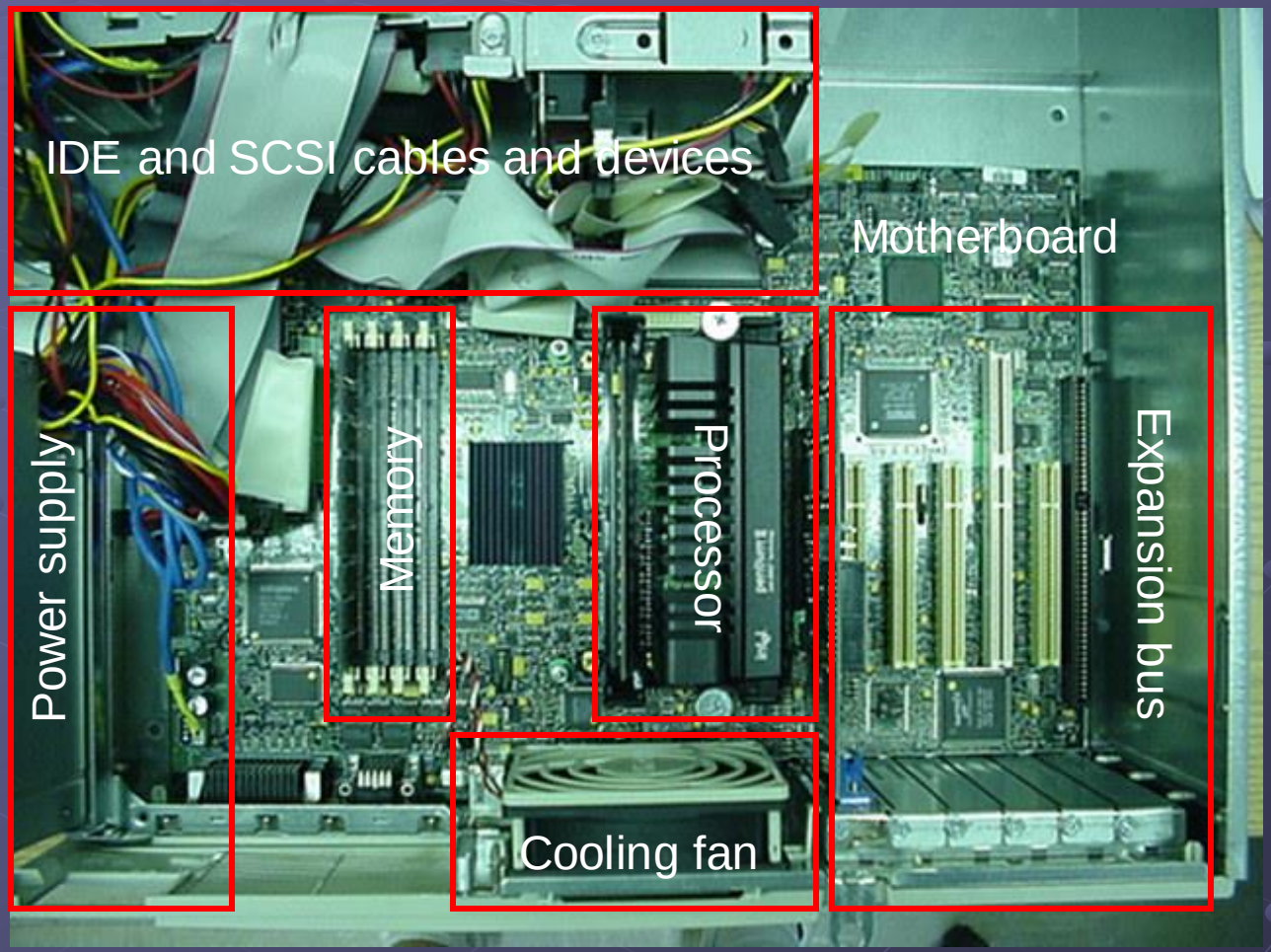


- Basic components
  - Mouse
  - Keyboard
  - Monitor
  - Printer
  - Scanner
  - Joystick
  - “The box”

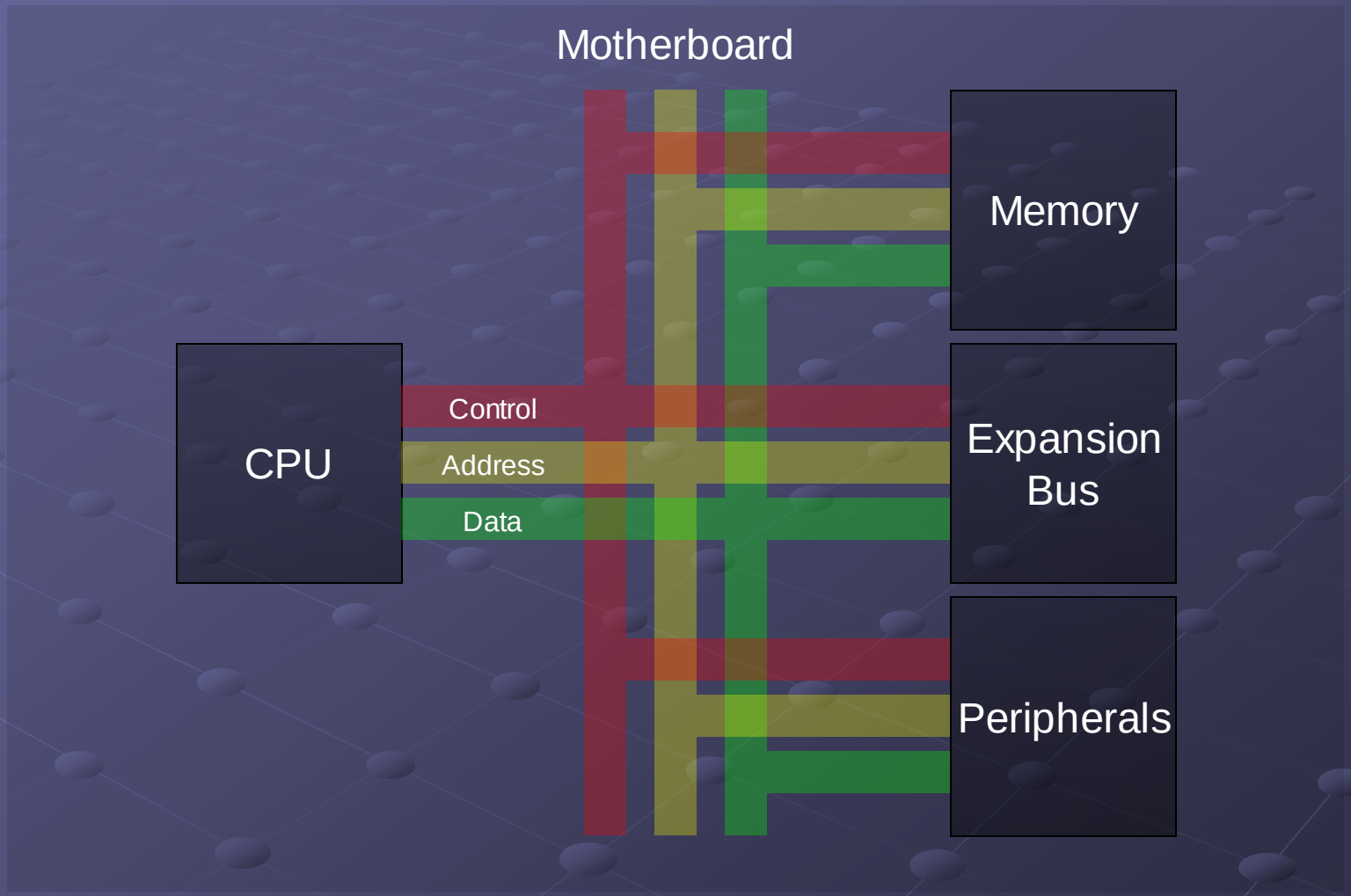
Computers from 10,000 feet



Computers from 100 feet



Computers from 10 feet

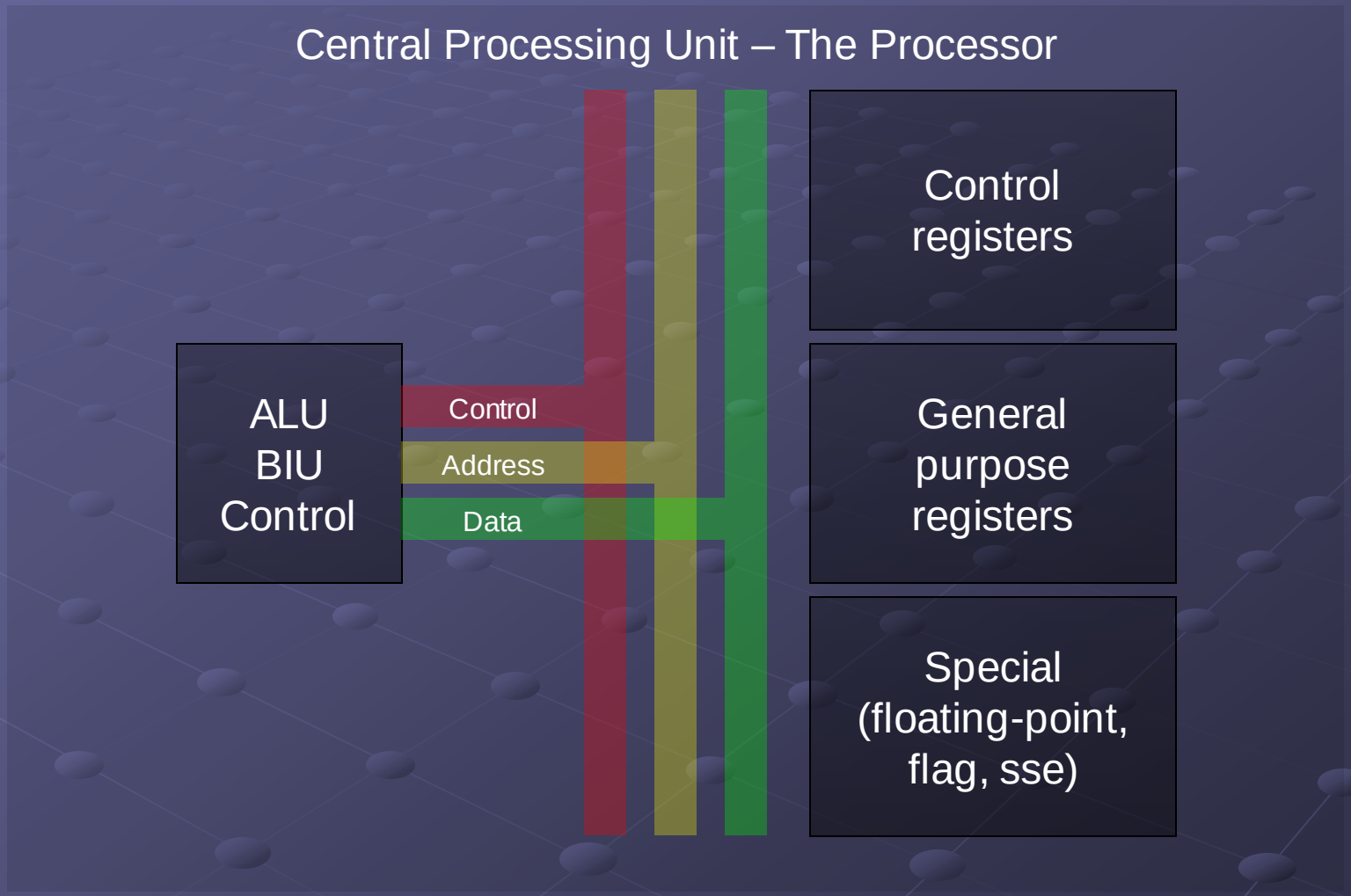


Computers from 10 feet

Motherboard

Processor	Data Bus	Address Bus	Max Addressable Memory	
8088	8	20	1,048,576	(1Mb)
8086	16	20	1,048,576	(1Mb)
80286	16	24	16,777,21	(16Mb)
80386dx	32	32	4,294,976,296	(4Gb)
80486	32	32	4,294,976,296	(4Gb)
80586/Pentium (Pro)	64	32	4,294,976,296	(4Gb)

Computers from 1 micron



The processor

- It's a state machine
- Has a set of hard-coded operations
  - Each operation has an associated code; an "opcode"
  - Some examples are MOV, ADD, SUB, AND
- All it can do is move data or perform some calculation on data
- Very simple, right?

The processor - components

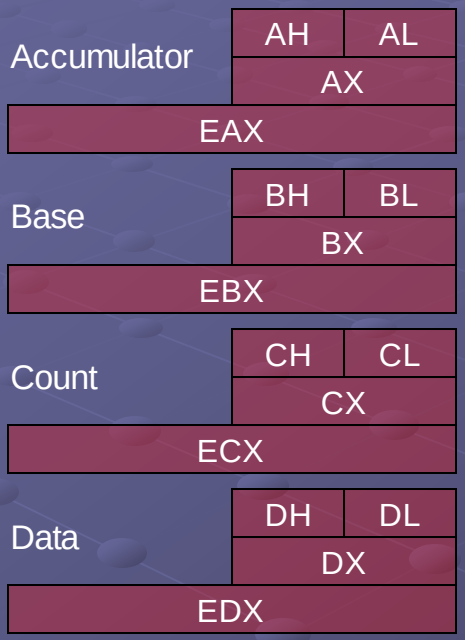
- CPU Registers
  - Special memory locations constructed from flip-flops and implemented on-chip
  - E.g., accumulator, count register, flag register
- Arithmetic and logic Unit (ALU)
  - Where most of the action takes place inside the CPU
- Bus Interface Unit (BIU)
  - Responsible for controlling the address and data busses when accessing main memory and data in the cache
- Control Unit and Instruction Set
  - CPU has a fixed set of instructions
  - E.g. MOV, CMP, ADD, JMP

The registers

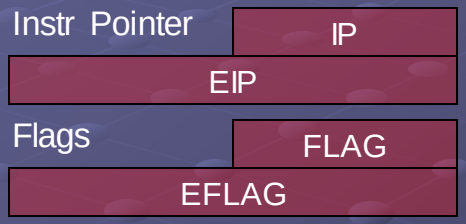
- Small memory locations that are quickly accessible by the processor
- Hold data and results for operations
- Point to memory locations
- Contain status information about the results of operations

80x86 registers

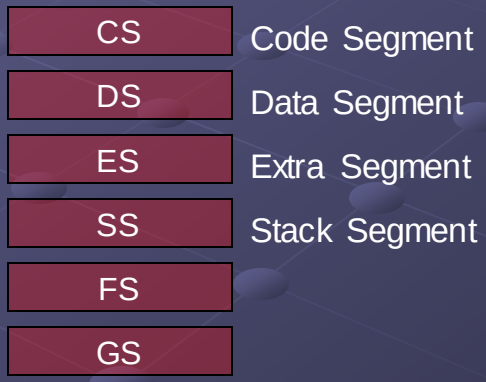
General Purpose



Special Registers



Segment Registers



Index Registers

Register specifics

- Accumulator (AL, AH, AX, EAX)
  - Usually stores results from the ALU
  - It “accumulates” totals from math operations
  - General purpose
- Base (BL, BH, BX, EBX)
  - Usually holds addresses – points to memory locations

Register specifics

- Count (CL, CH, CX, ECX)
  - Counting and looping
  - Certain instructions automatically decrement the count register
- Data (DL, DH, DX, EDX)
  - Usually contains data for instructions
  - Contain the most significant bytes of 16-bit Mul/div's

Register specifics

- Stack pointer (SP, ESP)
  - Used by stack operations
  - Usually you don't want to mess with this one
- Base pointer (BP, EBP)
  - Addresses stack memory
  - Can be used to read subroutine arguments

Register specifics

- Source index (SI, ESI)
  - Often used to address a source array or string data type
- Destination index (DI, EDI)
  - Often used to address a destination array or string data type

Register specifics

- Code segment (CS)
  - Points to the area in memory where instructions are stored
- Instruction pointer (IP)
  - An index into the code segment that points to the next instruction to be executed
- Data segment (DS)
  - Points to the area in memory where data needed by a program is stored

Register specifics

- Stack segment (SS)
  - Points to the area in memory containing the system's FIFO stack. We'll have a whole lecture on this.
- ES, FS, GS
  - Extra segment registers available to point to additional data segments should that be necessary.

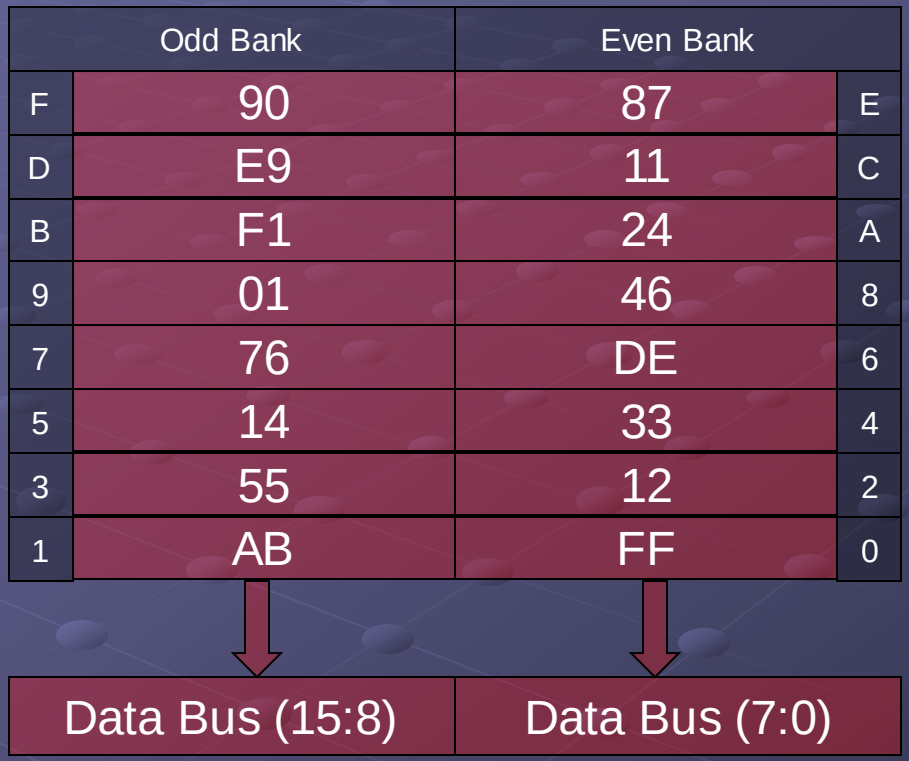
Memory

- System memory or RAM holds data and instruction opcodes that are being used by the CPU
- Organized in banks, usually even and odd
- Always byte addressable
- More on all this tomorrow

Memory access example

CPU wants to read memory at address 12345h

Memory organization



Memory organization

- In Real Mode:
  - The address bus is 20-bits for memory operations.
  - 2^{20} = 1MB = how much memory an x86 processor can address in real mode
  - The data bus is 16-bits which means it can transfer two bytes in one operation
  - Limited to 16-bit registers

Memory organization

- In Protected Mode:
  - The memory address bus is 32-bits
  - 2^{32} = 4GB = the maximum addressable amount of memory in modern x86 processors
  - Data bus is also 32-bits meaning four bytes can be transferred in a single operation
  - Gain access to the 32-bit extended registers

Peripherals

- In our context, things like printers, joysticks, and scanners aren't peripherals
- Some examples are parallel, serial and USB ports, the internal timers, interrupt controllers, network interface cards
- Many helper devices inside the processor, on the mother board, or in expansion slots

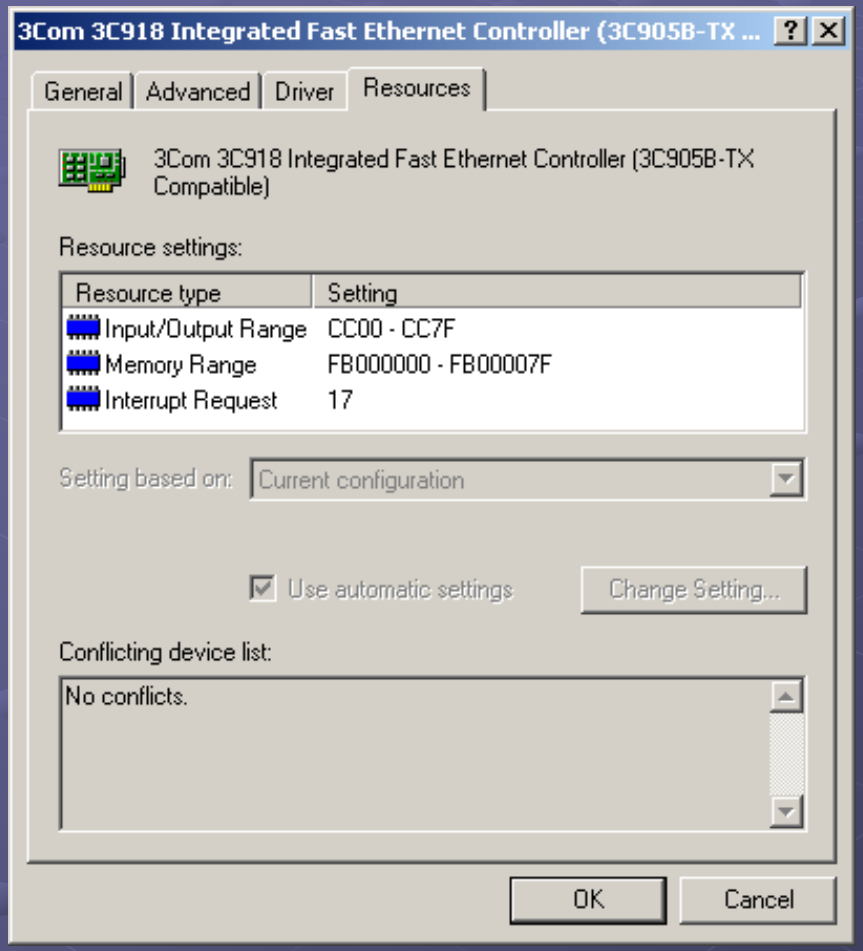
Peripheral I/O bus

- There is a separate set of address/data busses commonly used for communication with peripheral devices
- In reality, they're the same busses, just different signals in the control bus cause different things to happen
- The I/O address bus is 16-bits wide, meaning you can address 65,535 different I/O ports.
- The I/O data bus is 16-bits

Peripheral I/O

- Peripheral devices don't *have* to use the I/O bus. They can be memory mapped.
- Video cards are an example
- They have a large amount of memory (VRAM) and that is typically mapped to a range of addresses in the memory space

Peripheral example



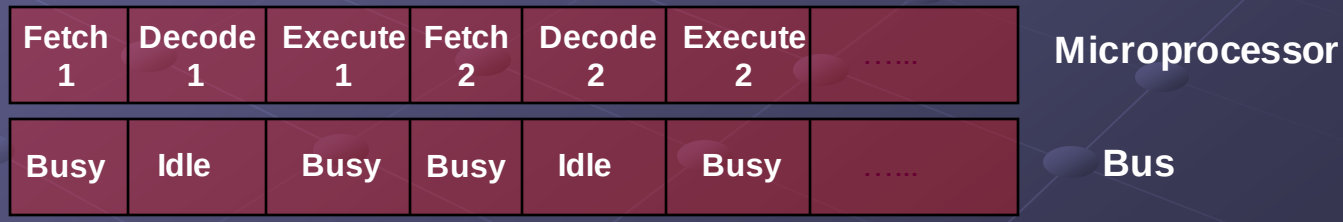
- Network card
- Input/Output Range CC00-CC7F
- Memory Range FB000000-FB00007F
- This one is both memory mapped *and* I/O mapped

Peripherals

- All this really means to you is that sometimes you'll use one set of instructions to access a device, and sometimes you'll use a different set.
- We'll talk about this again when we cover serial/parallel ports, timers, interrupts, and the video graphics adapter.

Instruction processing

- Processing of an instruction by the microprocessor consists of three basic steps
  1. fetch instruction from the memory
  2. decode the instruction
  3. execute (usually involves accessing the memory for getting operands and storing results)
- Operation of an early processor like the Intel 8085



Instruction processing

- Modern microprocessors can process several instructions simultaneously at various stages of execution
  - this ability is called pipelining
- Operation of a pipelined microprocessor like the Intel 80486

Assembly primer

- Everything in your assembly file is one of the following:
  - A comment
  - A label
  - An instruction or directive
  - Data

Assembly primer - comments

```
; Comments are denoted by semi-colons.
; These are comments!!!  Yeah!!!.
; Please comment your MP's thoroughly.
; It helps us figure out what you were doing
; It also helps you figure out what you were
;   doing when you look back at code you
;   wrote more than two minutes ago.
```

Assembly primer

- You might be asking, “What about variables? Why aren’t they on the bulleted list?”
- Variables are nothing more than labels that mark specific memory locations
- Labels can also mark the beginning of procedures or “jump to” locations in your code

Assembly primer - labels

```
; Labels can be global
MyGlobalLabel:
MyOtherGlobalLabel

; They can be local too
.MyLocalLabel

; Local labels are good only back to the
  previous un-dotted label
```

Assembly primer - labels

```
MyBigGlobalLabel
   .local_label1
   .local_label2

MyNextBigGlobalLabel
   .local_label1       ; these are distinct
   .local_label2       ; to the ones above
```

Assembly primer - variables

```
; Let's do something with labels now
VAR1    DB    255
VAR2    DB    0FFh
VAR3    DW    1234h
ARR1    DB    12, 34, 56, 78, 90
ARR2    DW    12, 34, 56, 78, 90
STR1    DB    '291 is awesome!!!', 0
BUF1    RESB 80
```

Assembly primer - directives

- EXTERN – allows you to declare external procedures so you can use them in your program.
- SEGMENT – lets you declare the beginning of a memory segment. We'll talk in detail about memory segments tomorrow.
- EQU – lets you define pre-processor constants.

Assembly primer - directives

```
; Let's declare some external functions
EXTERN kbdine, dspout, dspmsg, dosxit

; Let's begin our code segment
SEGMENT code

; I'd normally put my variables right here

..start      ; This is a special label that
             ; denotes the execution starting
             ; point.  The next instruction
             ; after ..start will be the first
             ; to run when you execute your program
```

Assembly primer - instructions

```
; Here are some simple instructions
mov     ax, [VAR1] ; notice the brackets
mov     dx, STR1   ; notice the not brackets
call    dspmsg
jmp     done
mov     bx, [VAR2] ; this will never happen

done
```

Example

```
SEGMENT code
  var1 db  55h                ; 55
  var2 db  0AAh               ; AA
  var3 db  12h                ; 12
  str1 db  "Hello", 0         ; 48 65 6C 6C 6F 00

..start
  mov  al, [var1]             ; A0 00 00
  mov  bx, var3               ; BB 02 00
  inc  bx                     ; 43
  mov  al, [bx]               ; 8A 07
```

Assignments

- Keep working on MP0 and HW0