Intel 8086 Microprocessor

CO 2103 Assembly Language

Topics

- Introduction to Processors
  - Brief History
  - Processor Differences
- Basic Intel 8086 Architecture
  - Specifications
  - Registers
- Memory System
  - Segmentation
  - Memory Access
- Input/Output
  - Accessing/addressing
  - IO Schemes
- Interrupt
  - Hardware
  - Software
- Fetch-Execute Sequence

CPU History - 1

- First CPU: Intel 4004 – 1971, 4-bit data bus, 2,250 transistors
  - 4-bit: 4004 (1971), 4040 (1972)
  - 8-bit: 8008 (1972), 8080 (1974), 6502 (1974), Z80 (1976), 8085 (1976)
- First IBM PC CPU: Intel 8086 – 1978, 16-bit data bus, 20-bit address bus, 29K transistors
  - 16-bit: 8086/8088 (1978), 68000 (1979-1990), 80186/80188 (1982), 80286 (1982), V20/V30 (1984)

CPU History - 2

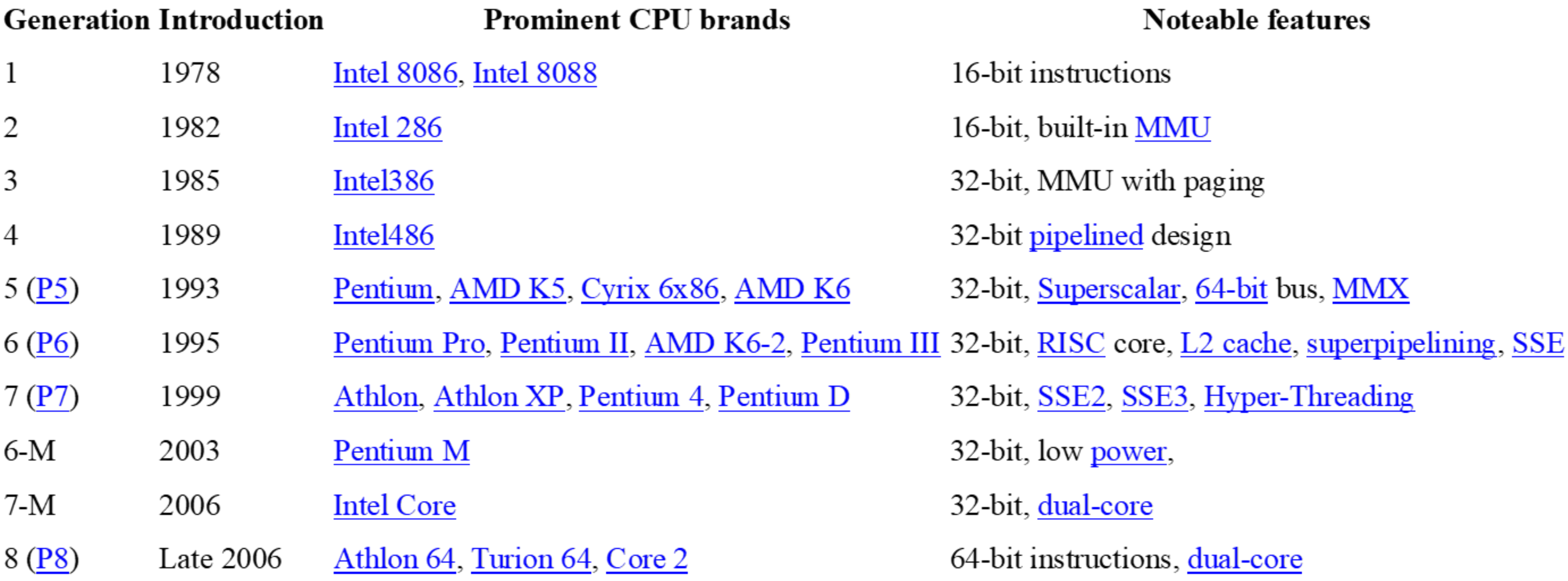
- 3rd Generation: Intel 80386 – 1985, 32-bit data bus, 24-bit/32-bit (DX series) address bus, 275K transistors
  - 32-bit: 80386 (1985), 80486 (1989), Pentium (1993), Cyrix 6x86 (1995), AMD K5 (1996), PII (1997), Celeron (1999), PIII (1999)
  - 64-bit: P4 (2000), Athlon XP (2001), Petium M (2003)
- Latest (by then): Intel Pentium Core 2 Duo/Quad
  - 2006, 64-bit data bus, 2x 32KB L1, 4 MB L2 Cache, >290M transistors
- References:
  - http://redhill.net.au/c/c-1.html
  - http://www.cpu-world.com/
  - http://www.mynikko.com/CPU/index.html

Processor Differences
(some)

- Size
  - data bus – usually dictates size of CPU. It is one major factor effecting speed of CPU.
  - address bus – dictates maximum range of memory
  - register – effect speed of processing
- Clock speed
- Data structure – e.g. the Endian
- Instruction set – available functions
- Architecture – incl. parallelism, caching

Famous x86 Processors

The table below lists brands of famous x86 consumer targeted processors grouped by generations. Note: A definition of CPU generation is not strict.



Generation	Introduction	Prominent CPU brands	Noteable features
1	1978	Intel 8086, Intel 8088	16-bit instructions
2	1982	Intel 286	16-bit, built-in MMU
3	1985	Intel386	32-bit, MMU with paging
4	1989	Intel486	32-bit pipelined design
5 (P5)	1993	Pentium, AMD K5, Cyrix 6x86, AMD K6	32-bit, Superscalar, 64-bit bus, MMX
6 (P6)	1995	Pentium Pro, Pentium II, AMD K6-2, Pentium III	32-bit, RISC core, L2 cache, superpipelining, SSE
7 (P7)	1999	Athlon, Athlon XP, Pentium 4, Pentium D	32-bit, SSE2, SSE3, Hyper-Threading
6-M	2003	Pentium M	32-bit, low power,
7-M	2006	Intel Core	32-bit, dual-core
8 (P8)	Late 2006	Athlon 64, Turion 64, Core 2	64-bit instructions, dual-core

source: http://en.wikipedia.org/wiki/X86

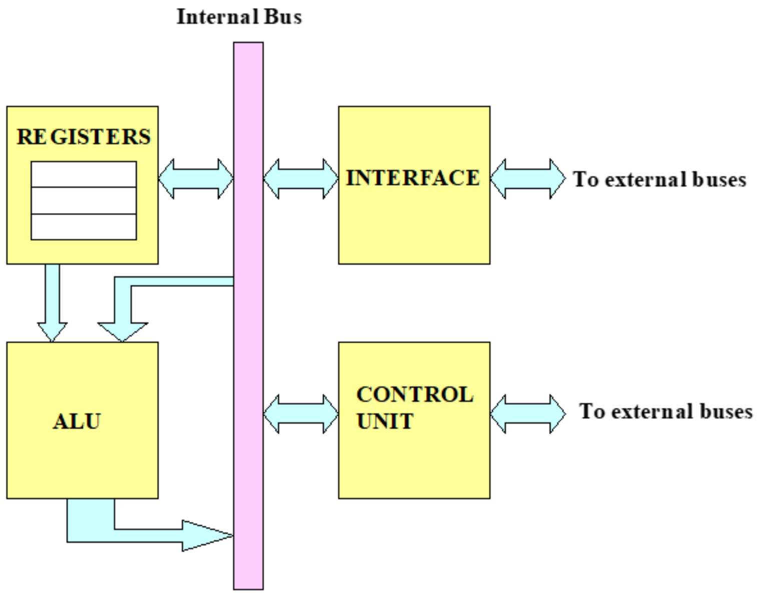
Intel 8086

- Why 8086?
  - the first (hence simplest) chip that powered the original IBM PC, and thousands of other models too
  - all later x86 designs (286, 386, 486, Pentium, and so on) have built on this foundation – hence “upward” compatible Instruction Set
- Brief specifications
  - June 1978, 40 pin DIP, 20-bit address bus, 1 MB max memory, 16-bit data bus, 2^{16} 8-bit IO ports or 2^{15} 16-bit IO ports, 29K transistors, 137 instructions (only), 4.77-10 MHz
  - 4x 8/16-bit data registers, 5x 16-bit pointer/index registers, 4x 16-bit segment registers, 1x 16-bit status register

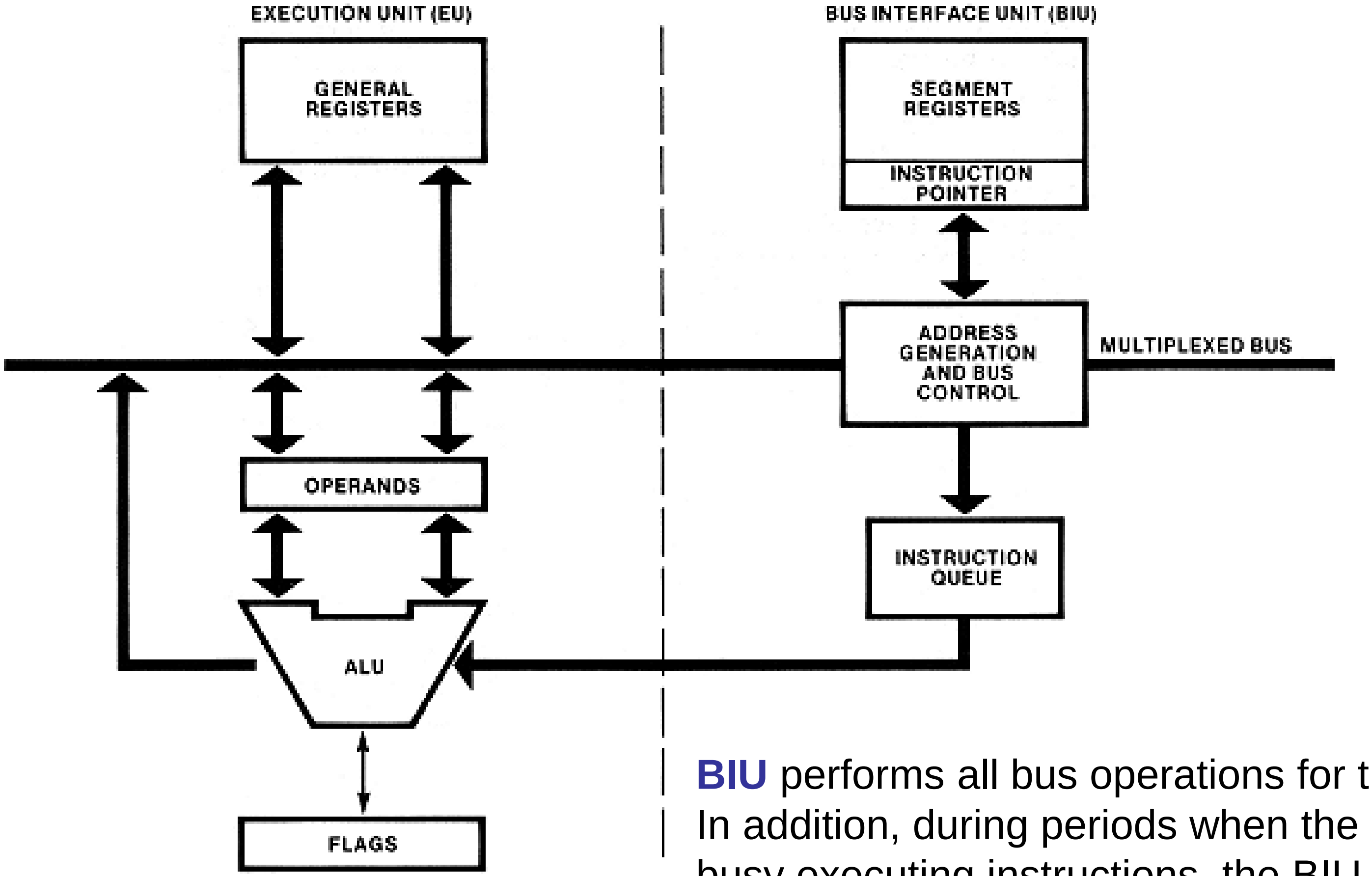
8086 Architecture – 1

(the "how" - how many, arrangement, etc)

- Recall CPU components:
  - Registers, Control Unit, ALU, Bus Interfaces
- Various versions of internal architecture block diagram found, however with similar basic components
- Two versions given in next two slides



8086 Architecture - 2



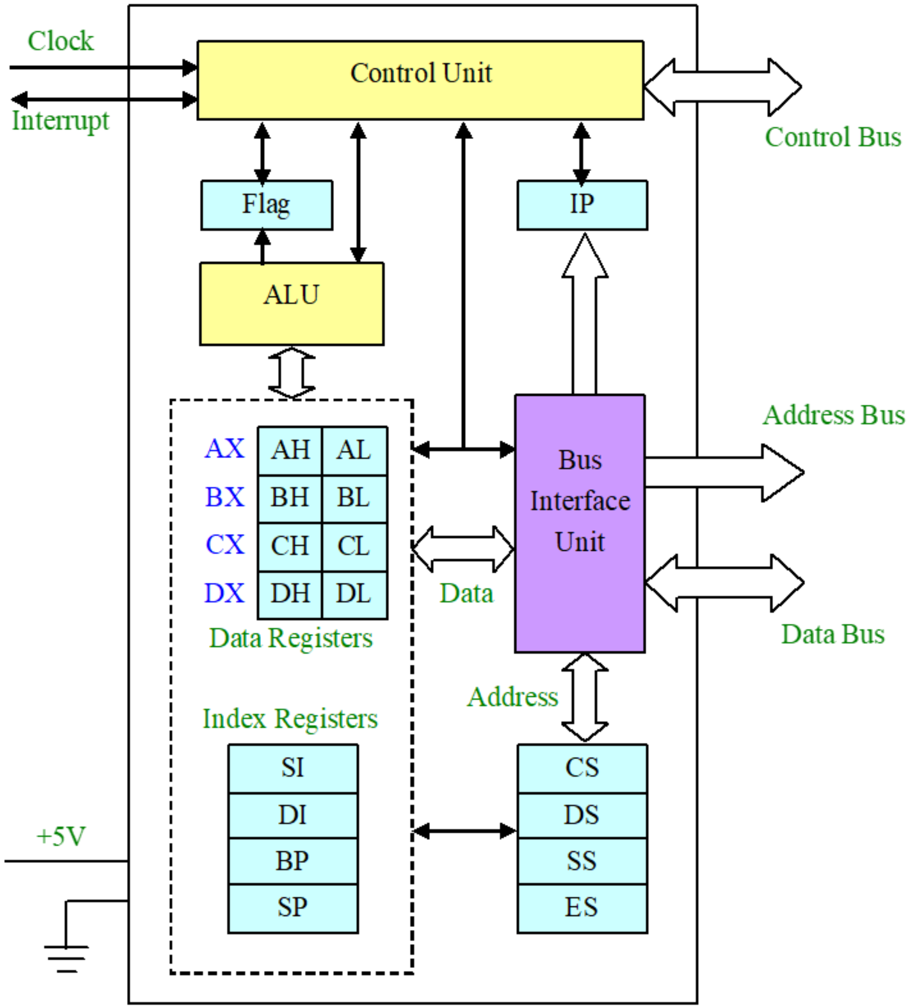
EU contains all components that are responsible for the execution of instructions.

BIU performs all bus operations for the EU. In addition, during periods when the EU is busy executing instructions, the BIU "looks ahead" and fetches more instructions from memory.

8086 Architecture - 3

Intel 8086

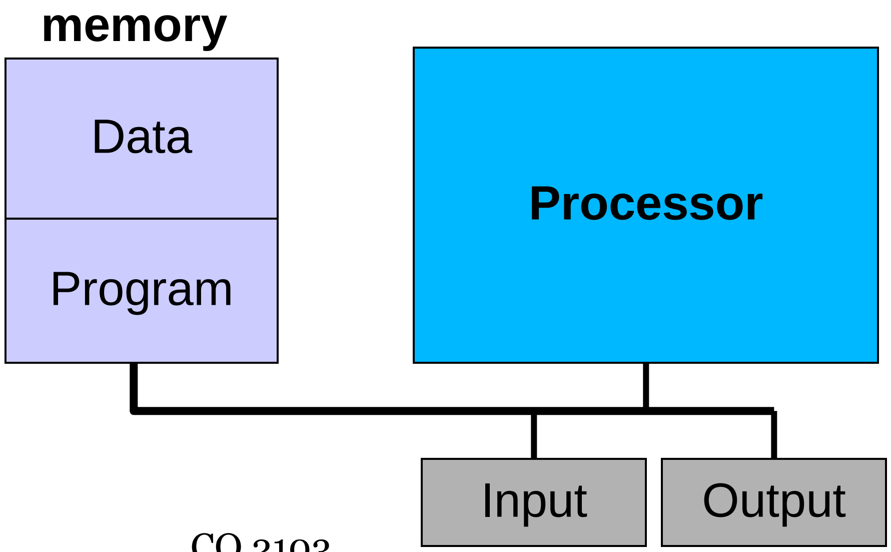
Internal Block Diagram



This is the preferred representation in this course

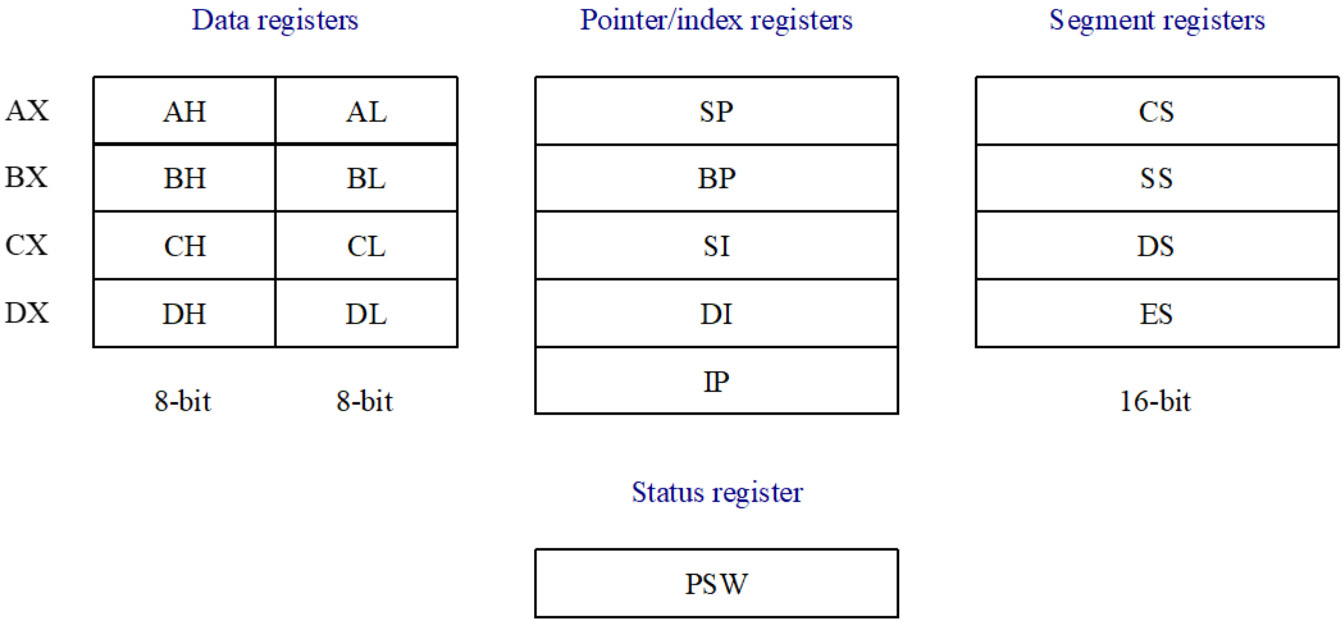
Von Neumann

- Intel 8086 uses Von Neumann architecture
- The von Neumann architecture is a design model for a stored-program digital computer that uses a processing unit and a single separate storage structure to hold both instructions and data.



Registers - 1

- Out of all CPU components, of interest to programmers are the Registers – storage for immediate processing



Data registers

AX AH AL

BX BH BL

CX CH CL

DX DH DL

8-bit 8-bit

Pointer/index registers

SP

BP

SI

DI

IP

Segment registers

CS

SS

DS

ES

16-bit

Status register

PSW

Registers - 2

- Data registers , also known as General Purpose registers:
  - used for arithmetic operations and data movement
  - can be addressed as 16 bit or 8 bit values
- Pointer and Index registers contain the offsets to a base memory address
  - offset refers to the distance of a variable, label, or instruction from its base segment
  - used when processing strings, arrays, and other data structures
  - pointer registers are sometimes called as Index registers
- Segment registers are used as base locations for program instructions, data, and the stack
  - all references to memory involve a segment register as the base location

Registers - 3

- Data registers
  - AX – Accumulator – involves in most ALU operations. Store most ALU results.
  - BX – Base – can be used to hold base address in indirect addressing, e.g. accessing array
  - CX – Counter – can be used when programming iteration to count the loops
  - DX – Double/Data – used in double-word multiply/divide. Used to hold IO port address in IO instructions.
  - all above can be used as two 8-bit registers (high & low)

Registers - 4

- Pointer registers
  - IP (Instruction Pointer)
    - store address of next instruction
    - specifies an offset into the CS segment
    - not the operand of any instruction
  - SP (Stack Pointer)
    - points to the top item on the stack
    - specifies an offset into the SS segment
    - can be used as an operand in some instructions
  - BP (Base Pointer)
    - specifies an offset into any segment, but most commonly the SS segment
    - can be used as an operand in some instructions

Registers - 5

- Index registers
  - SI (Source Index)
    - specifies an offset into the DS segment, although they can be used as offsets into any segment (using BX)
    - use with BP to index into SS segment
    - points to source string in some string instructions
  - DI (Destination Index)
    - specifies an offset into the DS segment, although they can be used as offsets into any segment (using BX)
    - use with BP to index into SS segment
    - points to destination string in some string instructions

Registers - 6

- Segment registers
  - points to 64K segments in main memory
    - more details in segmented memory later
  - segments:
    - CS (Code Segment) – where program goes; works with IP
    - SS (Stack Segment) –where temporary storage goes; works with SP
    - DS (Data Segment) – where data (variables, etc) go
    - ES (Extra Segment) – used as a segment override to read a few bytes from far memory or for additional data

Registers - 7

- Status register
  - PSW (Processor Status Word) or Flag register
    - individual bits store the status of the CPU
    - bits are set or cleared as the result of many operations
    - bits may be affected indirectly (by the execution of an instruction) or directly by an instruction designed to access the status word
  - flags in PSW:

				O	D	I	T	S	Z		A		P		C

Registers - 8

- PSW flags (1 ≡ Set/Yes; 0 ≡ Clear/No → 0)
  - C (Carry) – is there carry in previous operation?
  - P (Parity) – is the parity even?
  - A (Auxiliary) – is there half-carry (nibble) in previous operation?
  - Z (Zero) – is the previous result zero?
  - S (Sign) – is the previous result negative?
  - T (Trap) – set to put processor in single-step mode for debugging
  - I (Interrupt) – is the interrupt enabled?
  - O (Overflow) – is there overflow in previous operation?
  - D (Direction) – set to process block data transfer or string from high order to low order memory; reset for reverse direction

Registers - 9

- Other registers available, example:
  - IR – Instruction Register
  - and other temporary registers
  - not for use by programmers

Memory System - 1

- Recall microcomputer (or any microprocessor-based system) components:
  - CPU, Memory, IO, Buses
- Of interest to programmers: CPU, Memory, IO
  - memory as important as registers
  - note register is a form of memory
- Memory external to CPU, while registers are in CPU
  - memory slower than registers
  - memory cheaper than registers
  - memory more space than registers
- Two types: Random Access (RAM) and Read Only (ROM)

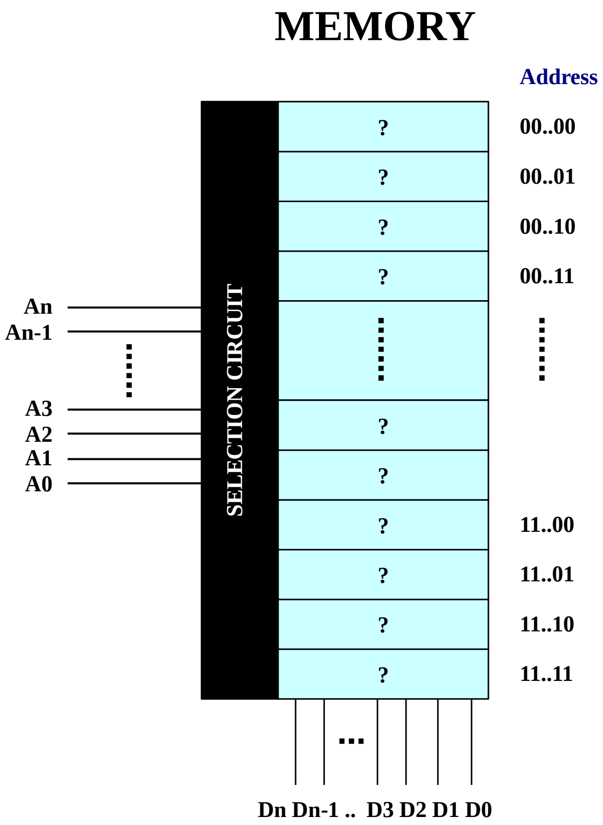
Memory System - 2

- Intel 8086 memory system:
  - 20-bit address bus giving 2^{20} locations, i.e. 1 MB (each location stores a byte)
  - each memory location is accessed by specifying a 20-bit address (00000_h – $FFFFF_h$)
- However, internal registers are only 16-bit wide, i.e. can't hold full address
- Solution: Use Memory Segmentation

Memory Access

(brief info)

- The desired address is asserted on the address lines (address bus), by the CPU: An … Ao
- The selection circuit of the memory will select the desired memory location to connect to the data lines (data bus) for read/write: Dn … Do
- n address lines can select upto 2^n different addresses (memory locations) – these are physical addresses
- Intel 8086 has 20-bit physical address – but cannot be stored in single 16-bit registers (for programming)



Memory Segmentation - 1

- Divide the memory space into logical segments of 64K, i.e. accessible by 16-bit address (2^{16} = 64K)
- Enable two 16-bit registers to store the address into two parts:
  - segment address – point to the start of segment
  - offset address – point within the segment
  - logical address – written in the form of **segment:offset**
- Physical Address (PA) computed from Segment Address (SA) and Offset Address (OA)
  - **$PA = (SA \times 16) + OA = (SA \times 10_h) + OA$**
    - $(SA \text{ x } 10_h)$ is the base address of the segment
    - x 10_h (shift 4 bits or 1 hex digit to left) effectively convert 16-bit into 20-bit

Memory Segmentation - 2

- Example: Given CS = 010Ch, IP = 14D2h, determine the physical address of the next instruction
  - PA $= (010C_h \times 10_h) + 14D2_h = 010C0_h + 14D2_h$
    $= 02592_h$
  - logical address 010C:14D2 ≡ physical address 02592

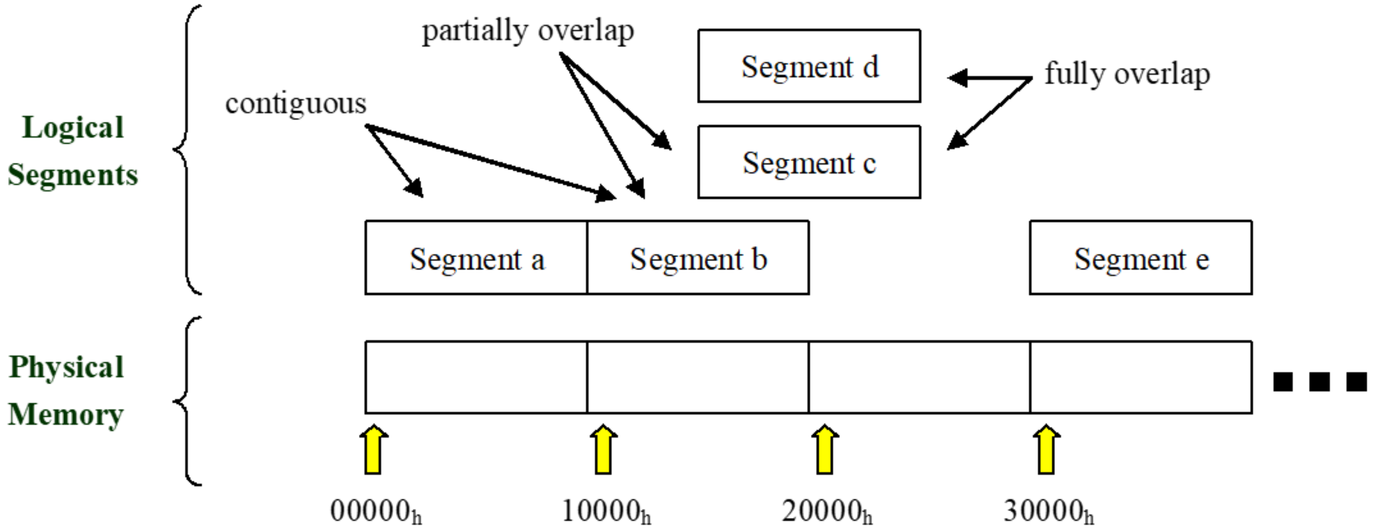
16-bit	CS		0	1	0	C	0
16-bit	IP	+		1	4	D	2
20-bit	Address		0	2	5	9	2

  - in general for: xxxx:yyyy ≡ aaaaa

16-bit	CS		x	x	x	x	0
16-bit	IP	+		y	y	y	y
20-bit	Address		a	a	a	a	a

Memory Segmentation - 3

- Segments can overlap



  - Two logical addresses can point to the same physical address – this is called Aliasing (or overlapping)
    - e.g. 1234:4321 ≡ 16661_h and 1665:0011 ≡ 16661_h

Memory Segmentation - 4

- Programmers use Logical Address, while CPU use Physical Address
- 80286 and above have Real and Protected Mode
  - simple calculation of PA only for Real Mode (compatible with 8088, 8086, 80186)
  - Protected Mode uses different algorithm to map logical address into physical address
- Segment base address is always physically at $xxxx0_h$
  - segment must begin at address that is multiple of 10_h (16) – called Paragraph boundary
  - each 10_h block is called a Paragraph

Physical Address	Memory
00000h	Paragraph 1
00010h	Paragraph 2
00020h	Paragraph 3

Memory Segmentation - 5

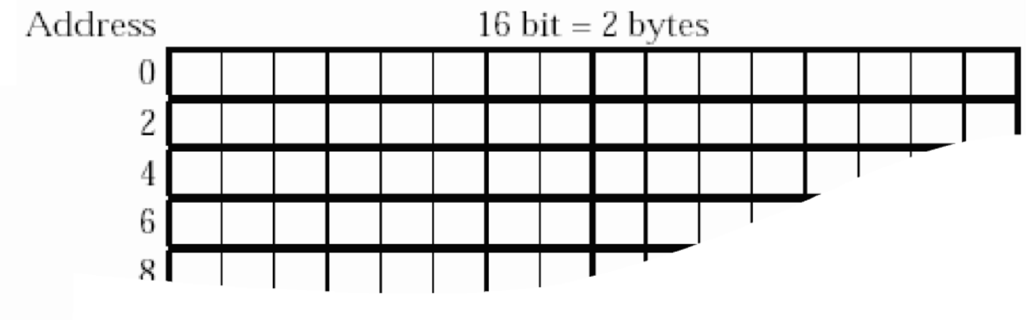
- Segments in 8086 (repeat)
  - CS (Code Segment) – where program goes; IP stores the offset
  - SS (Stack Segment) –where temporary storage goes; SP stores the offset
  - DS (Data Segment) – where data (variables, etc) go
  - ES (Extra Segment) – used as a segment override to move from one segment to another
  - segment base address stored in CS, SS, DS and ES registers accordingly
- Reserved segments
  - 0000_h - $03FF_h$ are reserved for interrupt vectors
  - $FFFF0_h$ - $FFFFF_h$ - after RESET the processor always starts program execution at the FFFF0h address, i.e. $CS=FFFF_h$, $IP=0000_h$

Memory Segmentation - 6

- Max value for segment = FFFF and max value for offset = FFFF
  - max memory that can be addressed:
    FFFF:FFFF = FFFF0 + FFFF
    = FFFF0 + (FFF0 + F) = FFFFF + FFF0
    = 1MB + FFF0
    - FFFFF equals to 1MB
    - FFF0 equals to 64Kb minus 16 bytes
- In Real mode a program can actually refer to (1MB + 64KB - 16) bytes of memory
  - however, Intel 8086 has 20-bit address bus and can only access up to 1MB, and this resulted in wrapping around of the addresses
  - e.g. if a code is referring to 1Mb + 1, this will get wrapped around and point to zeroth location in memory, likewise 1MB+2 will wrap around to address 1 (or 0000:0001)

Memory Access - 1

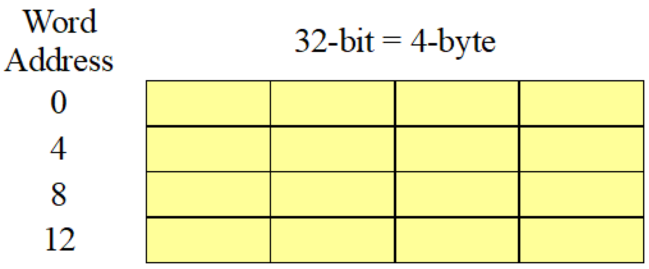
- Main memories generally store and recall rows, which are multi-byte in length (e.g. 16-bit word = 2 bytes, 32-bit word = 4 bytes)



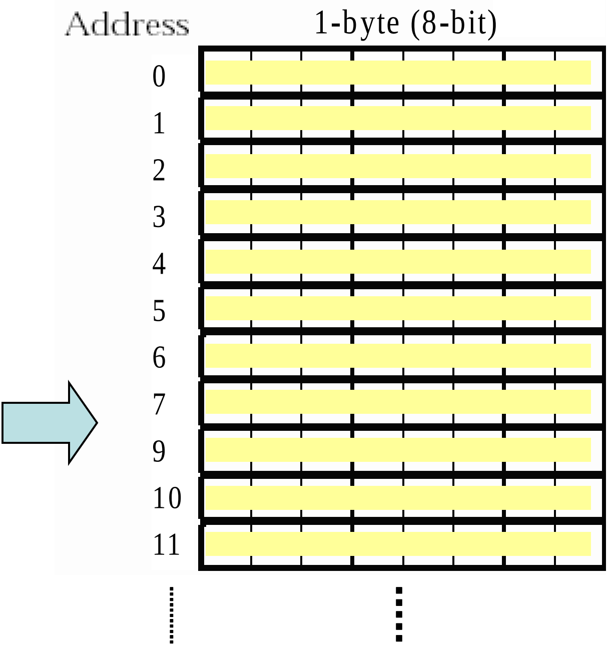
- 16-bit data bus implies that the CPU can access a 16-bit data at once
  - for 8086, 16-bit data bus → 16-bit per row of memory
- However, most architectures, make main memory byte-addressable rather than word addressable (read/write)
  - a memory address is assigned to each byte within the memory

Memory Access - 2

- The diagrams below show a memory system with 32-bit memory rows
  - the memory locations (rows of quad-words) have even addresses
  - often represented logically in rows of bytes, which have consecutive logical addresses (byte addressing)



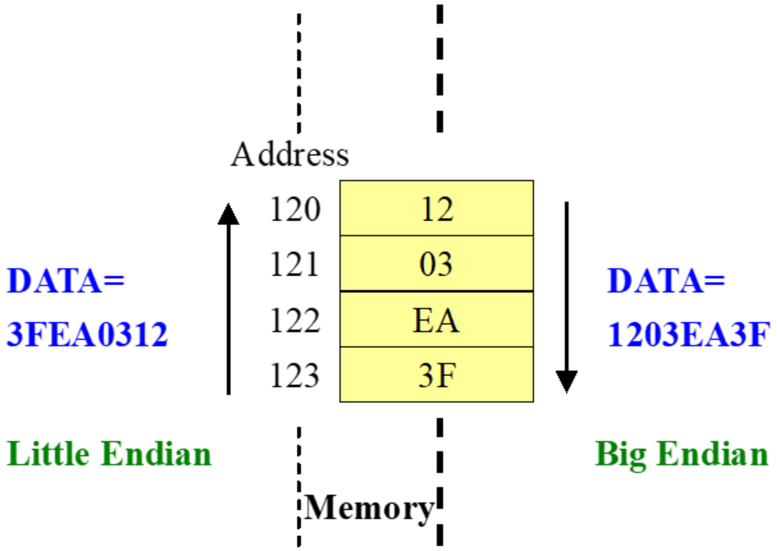
Physical arrangement



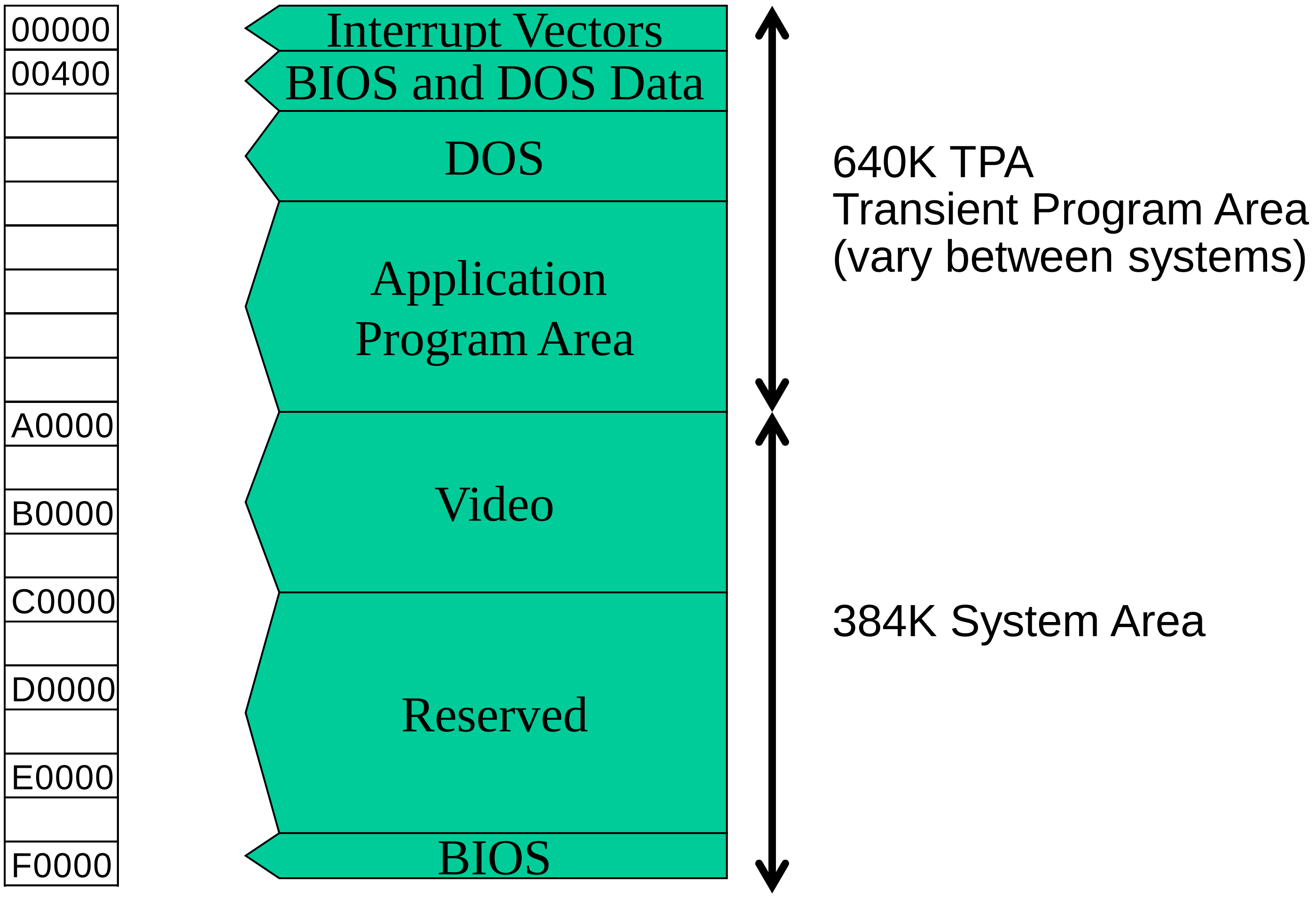
Logical representation

Memory Access - 3

- Memory access (read/write) of multi-byte data can be either of the two orders (schemes) :
  - Big Endian - the most significant byte of a multi-byte data item has the lowest address
  - Little Endian - the least significant byte of a multi-byte data item has the lowest address
  - Example on right: read/write a quad-word
  - Intel 8086 uses Little Endian system



Memory Map of Intel PC



00000

00400

A0000

B0000

C0000

D0000

E0000

F0000

Interrupt Vectors

BIOS and DOS Data

DOS

Application
Program Area

Video

Reserved

BIOS

640K TPA
Transient Program Area
(vary between systems)

384K System Area

System Startup

- Reset state
  - CS = FFFFh
  - IP = 0000h
- Executes an instruction in ROM that transfers to a BIOS routine

- System memory check
- Initialize interrupt vectors and BIOS data
- Load operating system from disk
  - boot sector

Input / Output - 1

- Input and Output are essential
  - imagine PC without keyboard, mouse, screen
  - imagine calculator without keypad and display
- I/O devices are interfaced to CPU through I/O Controllers and simplest I/O Controller has two I/O Ports (or addresses)
  - Data Port – passing data to/from CPU and the I/O device
  - Control Port – issue I/O COMMANDS (configuration) and to check the STATUS of the device
    - Many I/O Controllers have Status Port for checking its status
- In 8086, an I/O Port can use either of two address spaces
  - Dedicated I/O Space
  - Memory-Mapped I/O

Dedicated I/O Space

- 64K 8-bit I/O ports addresses or 32K 16-bit I/O ports
  - two consecutive bytes can be treated as a 16-bit port
  - four consecutive bytes can be treated as a 32-bit port (386 onwards?)
- Used in the IN and OUT instructions
- Not segmented, i.e. 8-bit physical address used in programming
- Locations $F8_h$ through FF_h (eight of the 64k locations) are reserved by Intel Corporation for use by future Intel hardware and software products
  - using these locations for any other purpose may inhibit compatibility with future Intel products.

Memory-Mapped I/O

- I/O devices may be placed in the memory space
- As long as the devices respond like memory components, the CPU does not know the difference
- Pros and Cons of Memory-mapped I/O
  - provides additional programming flexibility
  - brings the power of the full instruction set and addressing modes to I/O processing
  - reduces the number of addresses available for memory
  - memory reference instructions also take longer to execute
  - somewhat less compact than the simpler IN and OUT instructions

I/O Schemes

- Four I/O Schemes (ways to manage IO operations)
  - Programmed I/O
    - Continually "poll" a device's control port until the device is ready, then initiate transfer.
  - Interrupt-Driven I/O
    - CPU initiate transfer and then do something else. Device will "interrupt" CPU when transfer is complete.
  - DMA I/O
    - CPU initiate large data (block) transfer and then do something else. DMA controller will transfer block to/from main memory and then "interrupt" CPU after block is transferred.
  - I/O Processor
    - Delegate complex I/O processing tasks to a dedicated processor.

Interrupt - 1

- An interrupt
  - is signal from an event that alters the sequence in which the processor executes instructions
  - might be planned (specifically requested by the currently running program) or unplanned (caused by an event that might or might not be related to the currently running program)
- Types of Interrupt (sources of interrupt signal)
  - Hardware Interrupt
  - Software Interrupt (a.k.a Trap or Exception)

Interrupt - 2

- When a program receives an interrupt signal, it takes a specified action (which can be to ignore the signal)
  - hardware interrupt causes the processor to save its state of execution via a context switch, and begin execution of an interrupt handler
  - software interrupts are usually implemented as instructions in the instruction set, which cause a context switch to an interrupt handler similar to a hardware interrupt
- PCs support 256 types of software interrupts and 15 hardware interrupts
- Both processed the same way – use of interrupt handlers (interrupt service routines)

Hardware Interrupt - 1

- A way to avoid wasting the processor's valuable time in polling loops, waiting for external events
- Two types
  - INTR is a maskable hardware interrupt - can be enabled/disabled in programming
  - NMI is a non-maskable interrupt - cannot be ignored and has higher priority than INTR
- Initiated through interrupt input lines
  - often identified by an index with the format of IRQ followed by a number
  - the combined set of lines are referred to as IRQ0 through IRQ15

Hardware Interrupt - 2

- IRQ Numbers
  - IRQ 0 - System timer. Reserved for the system. Cannot be changed by a user.
  - IRQ 1 - Keyboard. Reserved for the system. Cannot be altered even if no keyboard is present or needed.
  - IRQ 2 - Cascaded to 9
  - IRQ 3 - COM 2(Default) COM 4(User)
  - IRQ 4 - COM 1(Default) COM 3(User)
  - IRQ 5 - Sound card (Sound Blaster Pro or later) or LPT2(User)
  - IRQ 6 - Floppy disk controller
  - IRQ 7 - LPT1(Parallel port) or sound card (8-bit Sound Blaster and compatibles)

Hardware Interrupt - 3

- IRQ Numbers (continued)
  - IRQ 8 - Real time clock
  - IRQ 9 - Cascaded to 2
  - IRQ 10 - Free / Open interrupt / Available / SCSI
  - IRQ 11 - Free / Open interrupt / Available / SCSI
  - IRQ 12 - PS/2 connector Mouse / If no PS/2 connector mouse is used, this can be used for other peripherals
  - IRQ 13 - ISA / Math Co-Processor
  - IRQ 14 - Primary IDE. If no Primary IDE this can be changed
  - IRQ 15 - Secondary IDE

Software Interrupt

- Each software interrupt is associated with an interrupt handler – a routine that takes control when the interrupt occurs
- Interrupt Vector Table
  - stores complete list of 256 interrupts and associated interrupt handlers
  - sides in the first 1 K of addressable memory
- Software interrupts can be caused by:
  - INT instruction - breakpoint interrupt
  - INT <interrupt number> instruction - any one interrupt from available 256 interrupts
  - INTO instruction - interrupt on overflow
  - Single-step interrupt - generated if the TF flag is set
  - Processor exceptions - due to error

INT – brief info

- INT n are instructions to invoke BIOS or DOS function calls
  - allow AL to access system hardware conveniently (ready-made routines)
  - standard list available as reference - which n for what function, how to call, etc
  - more on this in laboratories; few quick examples:

Return to DOS:

```
mov ax, 4c00h      ;function number
INT 21h
```

Red a character from keyboard:

```
mov ah, 01h  ;function number
INT 21h      ;AL=character read
```

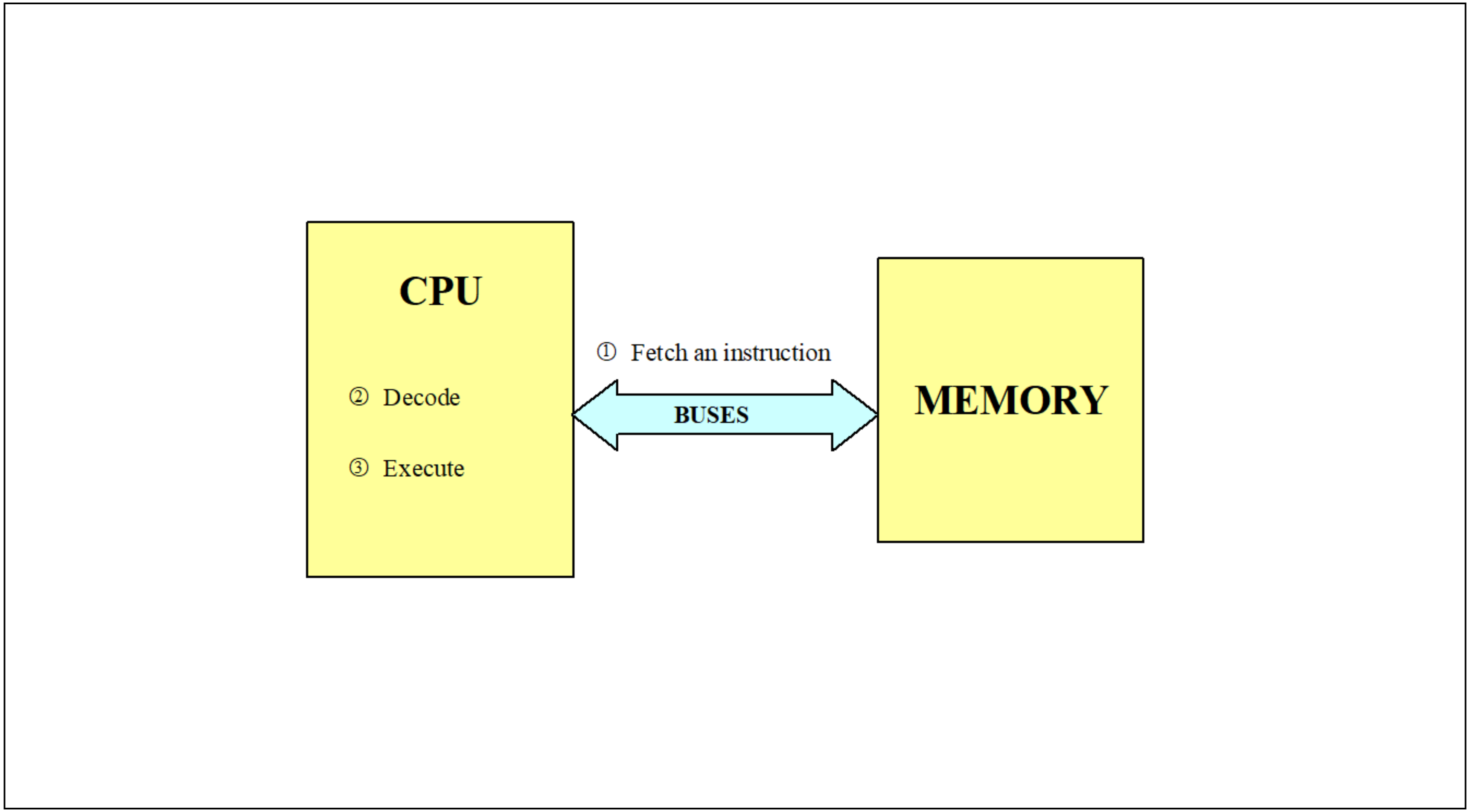
Print string on screen:

```
mov ah, 09h  ;function number
mov dx, 200h      ;offset to string
INT 21h
```

Fetch-Execute Sequence - 1

- Only one thing CPU does: fetch-decode-execute
  - regulated by clock signal – execution of instruction not within one clock cycle
  - step-by-step, clock-by-clock
    - fetch instruction byte from memory into IR
    - update the IP to point at the next byte
    - decode the instruction to see what operation
    - if required, fetch further operands and update IP after each fetch
      - compute the address of the operand, if necessary
    - execute the instruction
    - if any, store the result into the destination register

Fetch-Execute Sequence - 2



Summary

- Intel 8086 – good starting point and compatible with later Intel CPUs
- Important component of CPU (for programmers) – registers
- Important component of PC (for programmers) – memory and I/O
  - Memory segmentation
  - Memory Access – Big Endian and Little Endian
  - Memory mapped and dedicated IO
  - 4 IO Schemes
- Other stuff
  - Interrupt –Hardware and Software