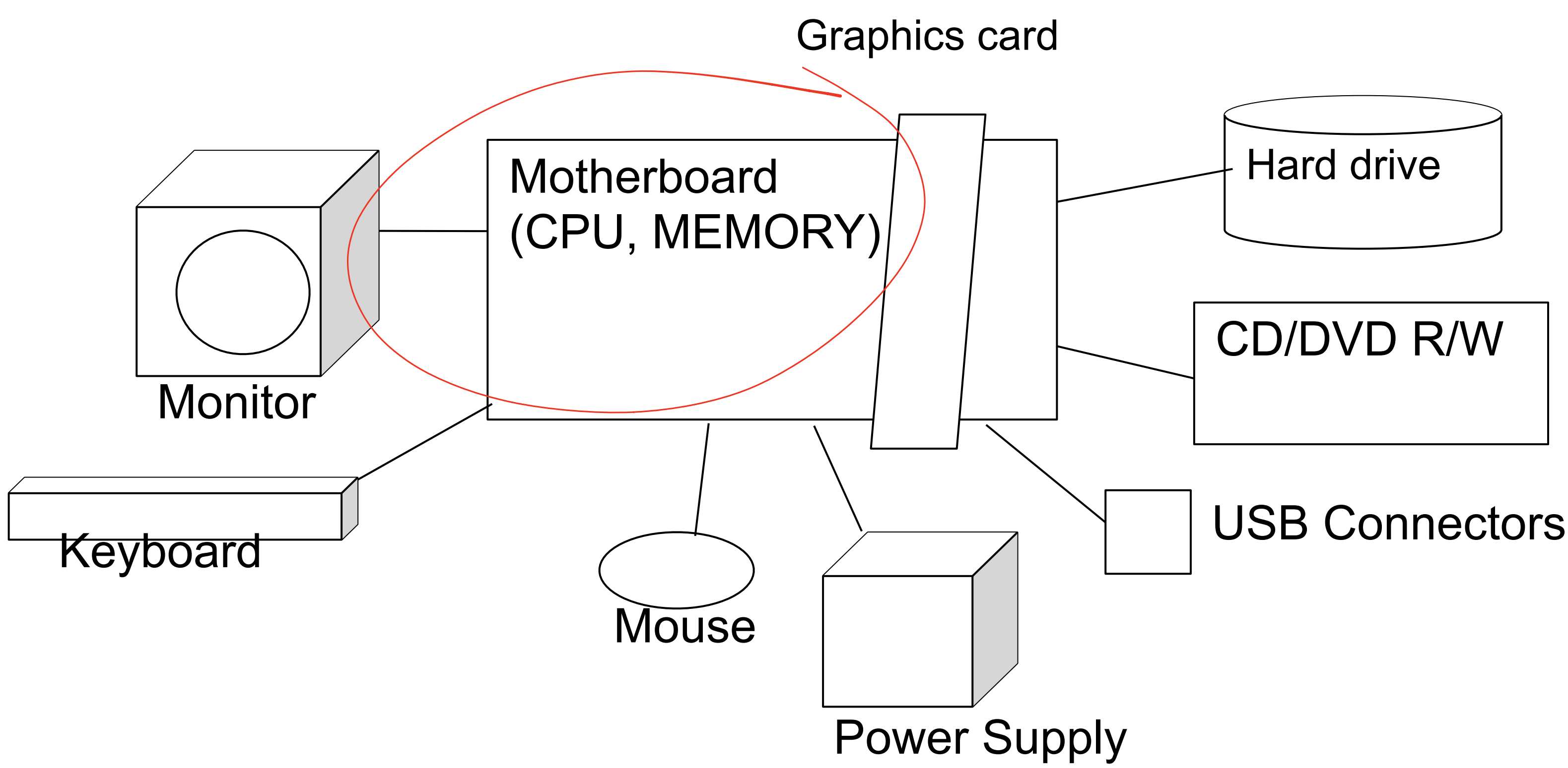
ECE243

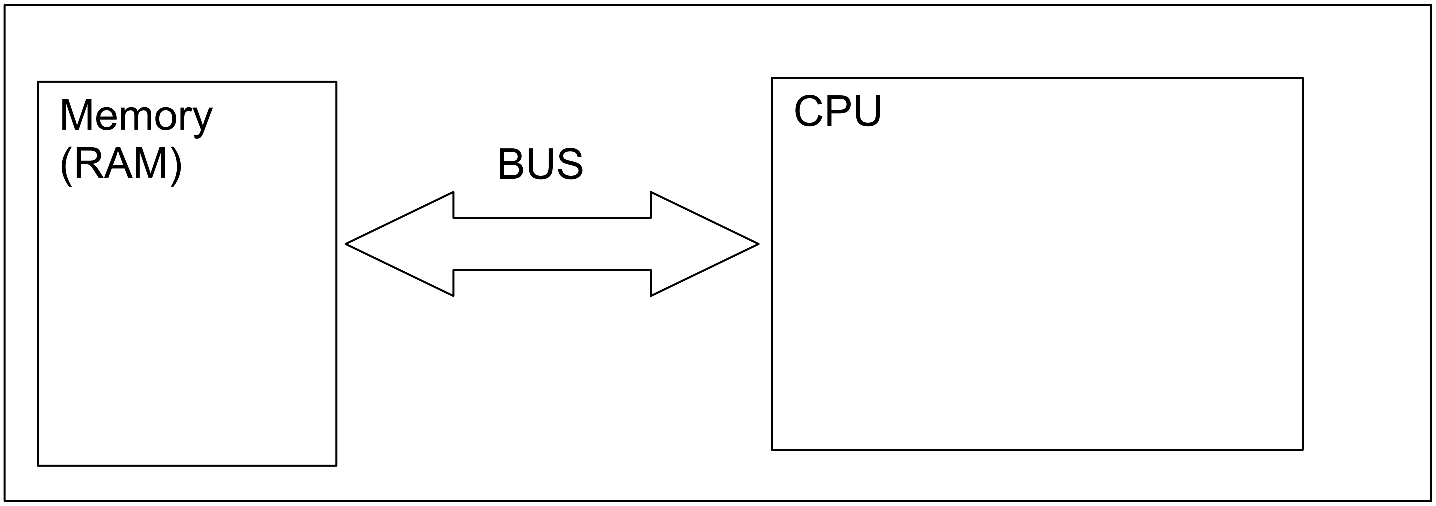
Prof. Enright Jerger

ISA: Instruction Set Architecture

A TYPICAL PC



Simple View of a Motherboard



- Memory:
  - holds bits
  - can be read from or written to
- BUS:
  - A collection of wires connecting two or more things
- CPU:
  - datapath: arithmetic/logic units (add, sub), muxes, etc.
  - control circuitry

GOALS OF A COMPUTER SYSTEM

- To process digital information
  - read data from memory, process by the CPU, write to memory
  - or from/to some other I/O device
- To be programmable
  - can change how the CPU processes
  - CPU “executes” a program
  - Program is a collection of instructions

– Each instruction is encoded as a group of bits, stored in memory

INTERFACE & IMPLEMENTATION

- **Example: Cars**

Interface: Steering wheel, gas & brake pedals, turn signals, etc

Implementation: front vs rear vs all wheel drive, engine size, speed, frame

Separating interface & implementation

Allows drivers to operate any car

Allows car designer to change/ improve implementation - fuel consumption

- **CPUs:**
  - interface:
    - ISA: instruction set architecture:
    - defines "machine instructions": groups of bits
  - implementation:
    - design of the CPU (datapath and control)
    - the logic and wires that execute machine instructions

Real Life ISAs

- Companies invent/license their own ISAs

- Examples:
  - Intel: IA32 (aka x86), IA64; IBM: PowerPC; SUN: SPARC
    - NOTE: x86 designed in 70's for CPUs with 2k transistors!
  - Motorola: 68000 (aka 68k), Altera NIOS II (MSL/243)
- How can the Pentium IV run programs written for the Pentium II?

backwards compatible
Pentium 4 supports ISA supported by Pentium II (interface)
diff implementation

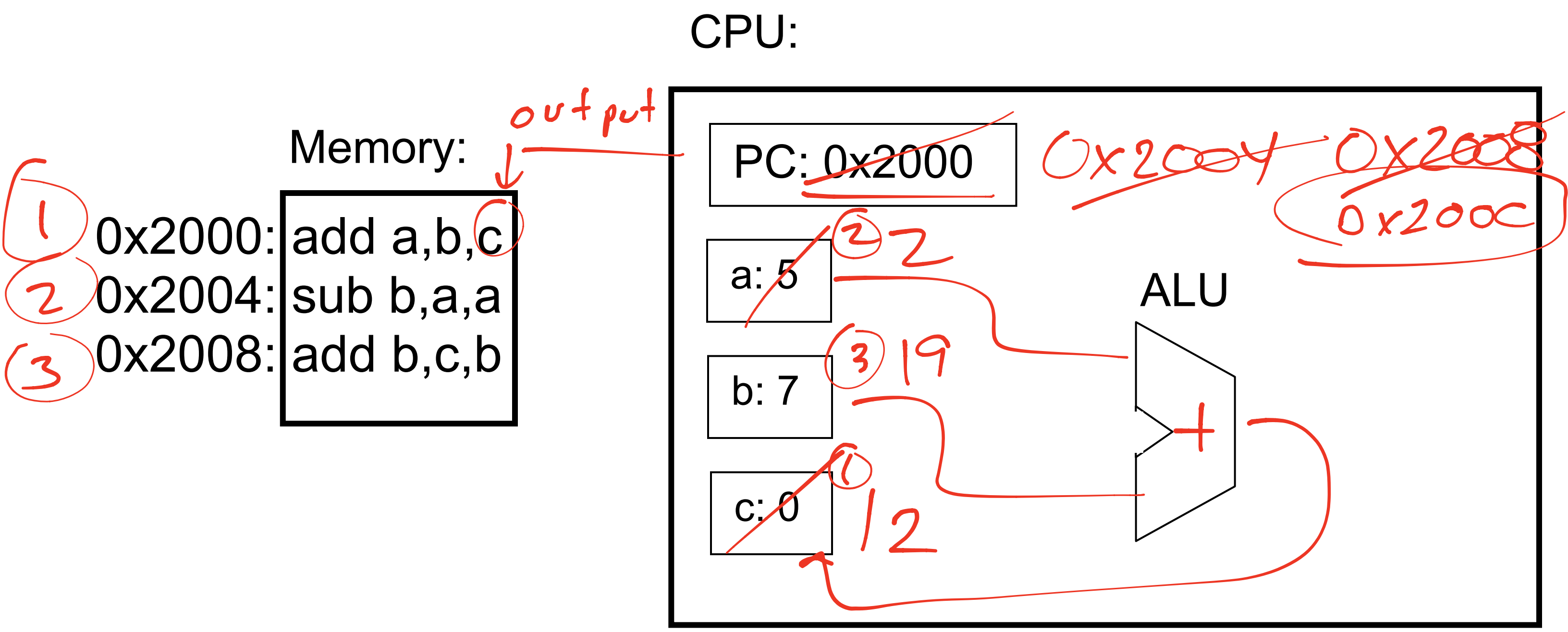
THE BASIC CPU CYCLE

- Forever:

  1. Fetch Instruction from Memory
  2. Read Input Operands
  3. Execute (calculate)
  4. Write Result
  5. Determine Next Instruction
- How does CPU know where the next inst is?
  - Program counter:
  - Called the PC
  - Internal to the CPU
  - Holds the memory address of the next instruction

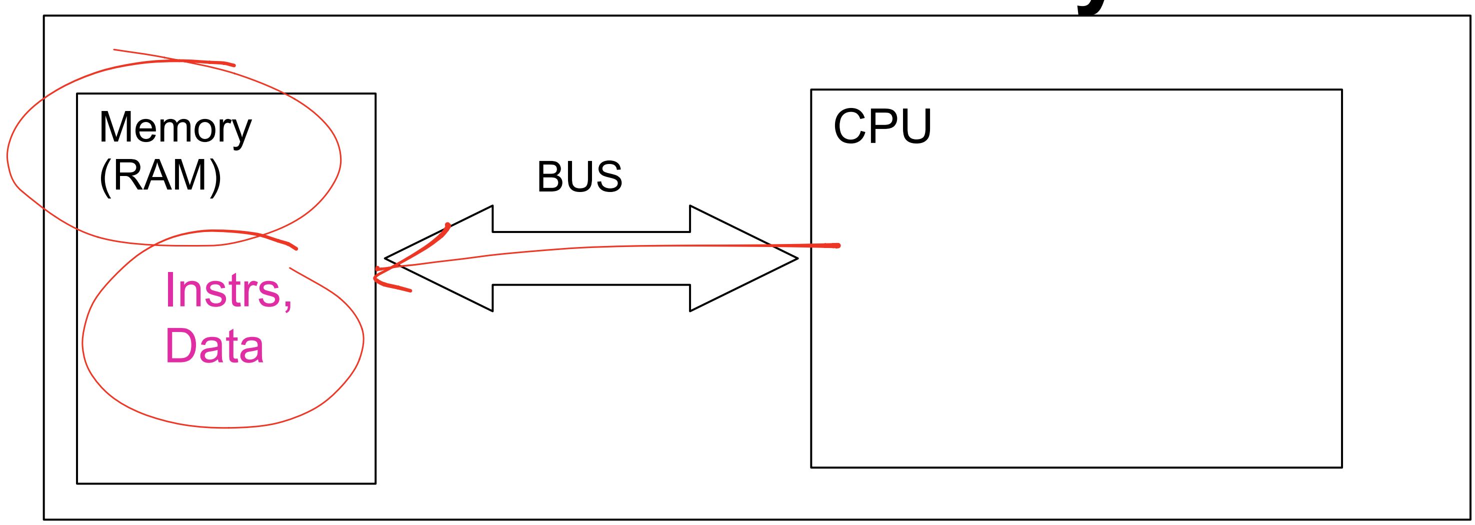
Simplified Example

1. Fetch Instruction from Memory
2. Read Input Operands
3. Execute (calculate)
4. Write Result
5. Determine Next Instruction (PC = PC + 4) = sizeof (instr)



Accessing Memory

"Von Neumann" Memory Model



- Instrs and data both reside in a memory
  - Note: instrs and data are both just bits
    - but interpreted differently

MEMORY OPERATIONS

- Load:
  - CPU provides an address
  - MEM returns value from that location
- Store:
  - CPU supplies address and value
  - MEM updates that location with the value
- A C-code analogy
  - char MEM[size];   // byte-sized elements
  - A load: val = mem[address];
  - A store: mem[address] = val;

MEMORY ADDRESSES

- A number
  - an index into the giant memory array
  - enough bits in number to index every memory location
- Address space:
  - the space of possible addresses for a memory
  - b = #bits to represent the address space,

- size = 2^b

- EX: how big is a 32-bit address space?

32 addr bits

∴ 2^{32} memory locations

4 billion locations - if each holds 1 byte → 4GB!

MEMORY GRANULARITY

- How much data is in each memory location?
  - usually one byte per location
  - such a machine is called "byte-addressable"
- EXAMPLE (assuming byte addressable)
  - Loadbyte A,0x20

A = 5

  - Storebyte 0x23,A

mem[0x23] ← 5

0x20	5
0x21	6
0x22	7
0x23	~~8~~ 5

Bits and Bytes Terminology

MS-bit ↓ ... LS-bit ↓

1	0	0	1	0	1

- LS-Bit: least-significant bit
- MS-Bit: most-significant bit

MSB			LSB
10101001	01001010	10101011	10001100

- LSB: least-significant byte
- MSB: most-significant byte

ENDIAN

- In what order do we load/store the bytes of a multi-byte value?
  - Depends whether processor is:
    - “big endian” or “little endian”
- Big Endian:
  - load/store the MSB first
    - i.e., in the lowest address location
- Little Endian:
  - load/store the LSB first
    - i.e., in the lowest address location

Endian Matters When:

1) You store a multi-byte value to memory

2) Then you load a subset of those bytes

Different endian will give you different results!

Different processors support different endian

- Big endian: motorola 68k, PowerPC (by default)
- Little endian: intel x86/IA-32, NIOS
- Supports both: PowerPC (via a mode)

Eg: must account for this if you send data from big-E machine to a little-E machine

EXAMPLE: ENDIAN

unsigned int a = 0x00000000;
unsigned int b = 0x11223344;
unsigned int c = 0x55667788;

- Assume: 'a' starts at addr 0x20000
- **Conceptual View:** (same for little or big endian)

Addr	Value	
0x20000	0x00000000	a
0x20004	0x11223344	b
0x20008	0x55667788	c

EXAMPLE: LITTLE ENDIAN

unsigned int a = 0x00000000;
unsigned int b = 0x11223344;
unsigned int c = 0x55667788;

Detailed View:

Addr	Value	
0x20000	00	
0x20001	00	a
0x20002	00	
0x20003	00	
0x20004	44	
0x20005	33	
0x20006	22	b
0x20007	11	
0x20008	88	
0x20009	77	c
0x2000a	66	
0x2000b	55	

EXAMPLE: BIG ENDIAN

unsigned int a = 0x00000000;
unsigned int b = 0x11223344;
unsigned int c = 0x55667788;

Addr	Value
0x20000	00
0x20001	00
0x20002	00
0x20003	00
0x20004	11
0x20005	22
0x20006	33
0x20007	44
0x20008	55
0x20009	66
0x2000a	77
0x2000b	88

a

b

c

EX: store 0xA1B2C3D4 to addr 0x20

Big Endian

Addr	Value
0x20	A1
0x21	B2
0x22	C3
0x23	D4

Little Endian

Addr	Value
0x20	D4
0x21	C3
0x22	B2
0x23	A1

Load 4B at 0x20:

0xA1B2C3D4

0xA1B2C3D4

Load 2B at 0x22:

0xC3D4

0xA1B2

Load 1B at 0x21:

0x B2

0x C3

Endian: Punchline

- Endian: a tricky detail of computer systems
  - can cause big headaches if you forget about it
- In labs and exams:
  - be careful not to forget about endian!
- Recall: endian matters when:
  1) You store a multi-byte value to memory
  2) Then you load a subset of those bytes