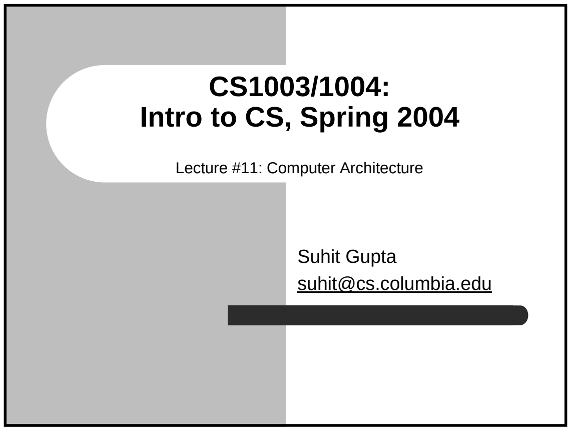
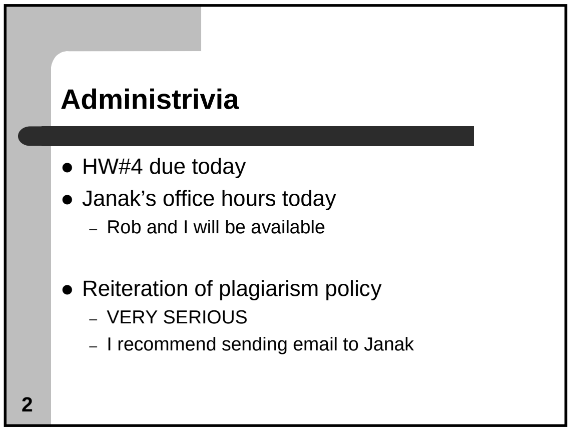


CS1003/1004: Intro to CS, Spring 2004

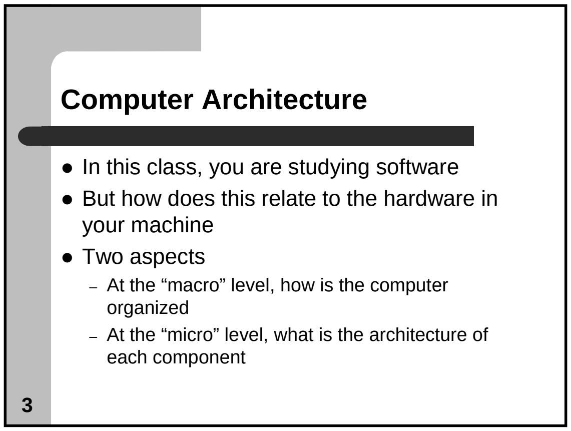
Lecture #11: Computer Architecture

Suhit Gupta
suhit@cs.columbia.edu



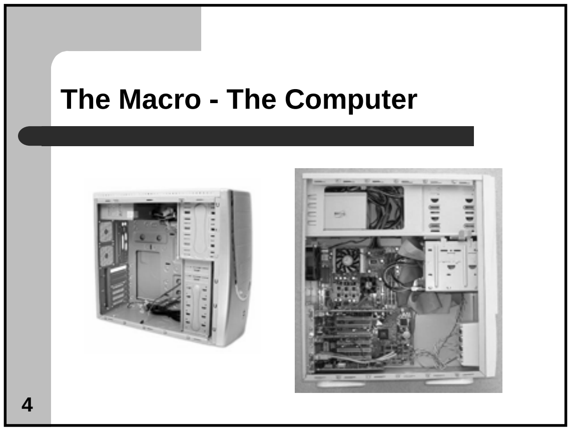
Administrivia

- HW#4 due today
- Janak's office hours today
  - Rob and I will be available
- Reiteration of plagiarism policy
  - VERY SERIOUS
  - I recommend sending email to Janak



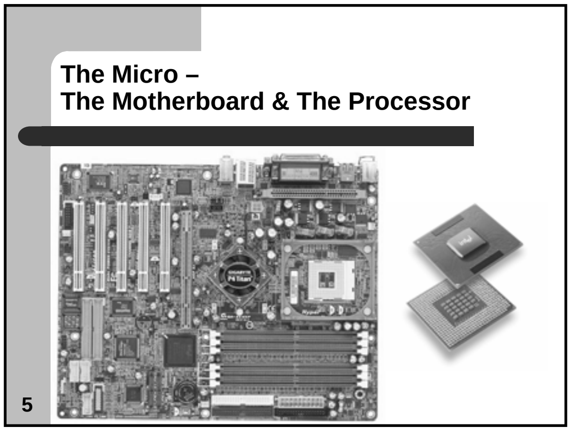
Computer Architecture

- In this class, you are studying software
- But how does this relate to the hardware in your machine
- Two aspects
  - At the "macro" level, how is the computer organized
  - At the "micro" level, what is the architecture of each component



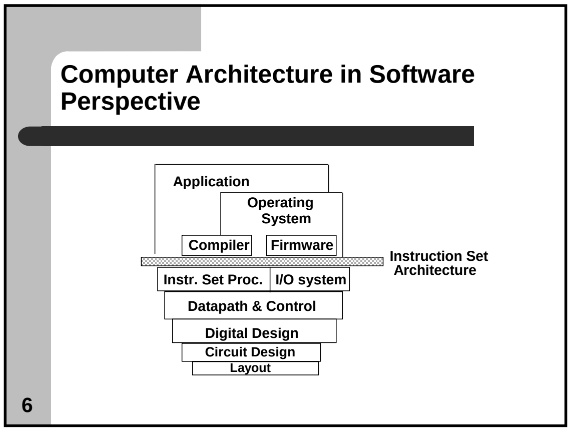
The Macro - The Computer

4



The Micro – The Motherboard & The Processor

5



Computer Architecture in Software Perspective

Application

Operating System

Compiler

Firmware

Instruction Set Architecture

Instr. Set Proc.

I/O system

Datapath & Control

Digital Design

Circuit Design

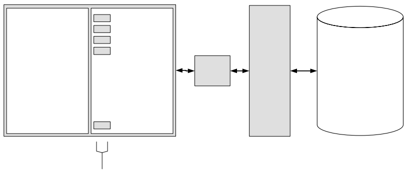
Layout

6

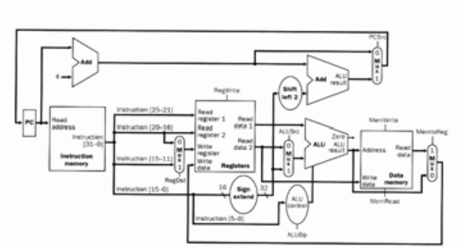
The CPU

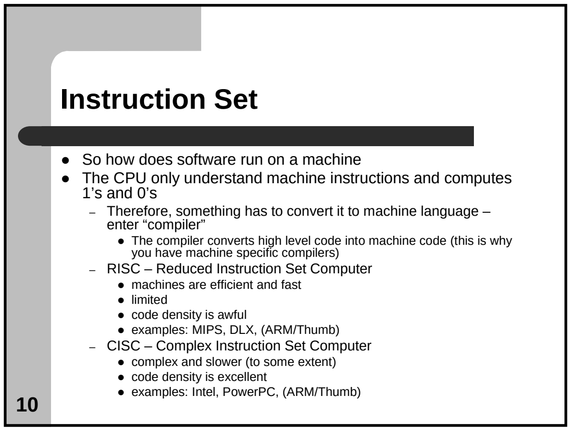
- CPU = Central Processing Unit
  - consists of two parts
    - ALU – Arithmetic Logic Unit
    - Control Unit
- The CPU contains talks to the machine memory (RAM) and the system cache, but it also has internal memory called registers

The CPU-Memory Relationship & Hierarchy



Chip Architecture (MIPS)

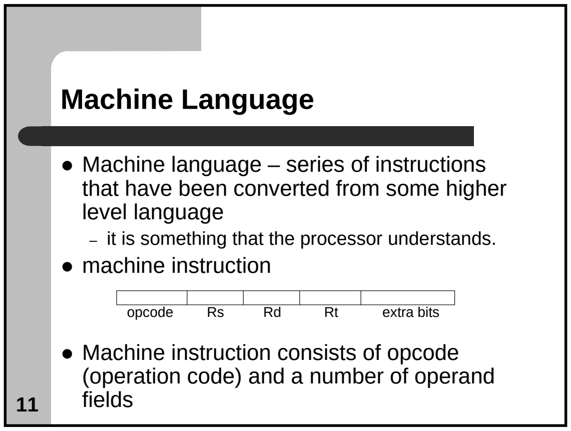




Instruction Set

- So how does software run on a machine
- The CPU only understand machine instructions and computes 1's and 0's
  - Therefore, something has to convert it to machine language – enter "compiler"
    - The compiler converts high level code into machine code (this is why you have machine specific compilers)
  - RISC – Reduced Instruction Set Computer
    - machines are efficient and fast
    - limited
    - code density is awful
    - examples: MIPS, DLX, (ARM/Thumb)
  - CISC – Complex Instruction Set Computer
    - complex and slower (to some extent)
    - code density is excellent
    - examples: Intel, PowerPC, (ARM/Thumb)

10



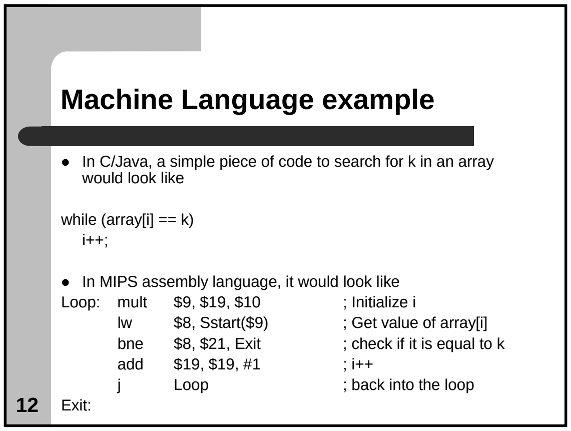
Machine Language

- Machine language – series of instructions that have been converted from some higher level language
  - it is something that the processor understands.
- machine instruction

opcode	Rs	Rd	Rt	extra bits

- Machine instruction consists of opcode (operation code) and a number of operand fields

11



Machine Language example

- In C/Java, a simple piece of code to search for k in an array would look like

```
while (array[i] == k)
    i++;
```

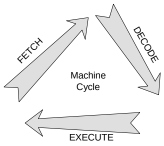
- In MIPS assembly language, it would look like

```
Loop:  mult   $9, $19, $10        ; Initialize i
       lw     $8, Sstart($9)      ; Get value of array[i]
       bne    $8, $21, Exit       ; check if it is equal to k
       add    $19, $19, #1        ; i++
       j      Loop                ; back into the loop
Exit:
```

12

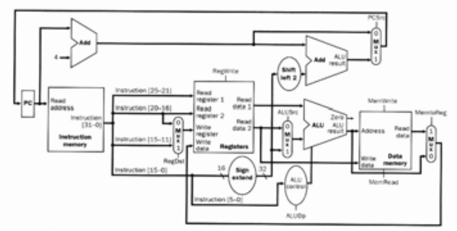
Also included in the architecture

- Program counter
  - contains the address of the next instruction
- The machine cycle



13

Back to the Chip Architecture



14

Pipelining (using DLX assembly)

- Blocks of code are typically large
  - One cannot execute each instruction, one at a time
  - Therefore, execute them together?
  - Pipeline them

```
LOOP: LW      R8,   0(R2)
      ADD     R10, R6,   R8
      ADDI    R2,   R10, #4
      SW       R10, 0(R2)
      ADDI    R3,   R3,   #4
      LW      R1,   100(R3)
      LW      R12, 100(R1)
      BGTZ   R12, LOOP
LOOP: LW      R8,   0(R2)
      ADD     R10, R6,   R8
      ADDI    R2,   R10, #4
```

15

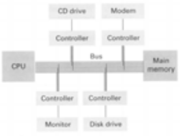
LOOP: LW R8, 0(R2)	IF	ID	EX	MEM	WB									
ADD R10, R6, R8						IF	ID	EX	MEM	WB				

LOOP: LW R8, 0(R2)	IF	ID	EX	MEM	WB									
ADD R10, R6, R8		IF	st	ID	EX	MEM	WB							
ADDI R2, R10, #4			st	IF	ID	EX	MEM	WB						
SW R10, 0(R2)					IF	ID	EX	MEM	WB					
ADDI R3, R3, #4						IF	ID	EX	MEM	WB				
LW R1, 100(R3)							IF	ID	EX	MEM	WB			
LW R12, 100(R1)								IF	st	ID	EX	MEM	WB	
BGTZ R12, LOOP									st	IF	st	st	ID	EX
Fall Through											st	st	flush	
LOOP: LW R8, 0(R2)												st	IF	ID
ADD R10, R6, R8														IF
ADDI R2, R10, #4														

16

Communication via controllers

- Communication between a computer and other devices is typically handled through an intermediary device called a *controller*
- A controller converts messages and data back and forth for compatibility
- Each controller is assigned unique addresses
  - Set of addresses assigned is called a port
- Memory mapped I/O
- Direct Memory Access (DMA)
  - wonderful for performance
- von Neumann bottleneck
  - CPU and controllers, both trying to access the machine bus

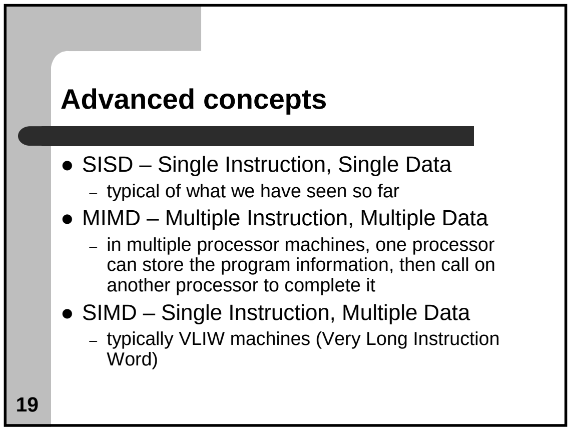


17

Multiprocessor machines

- Pipelining can be viewed as the first step towards supporting multiple processors (parallel processing)
- Common pitfall: mutiple processors is different from multiple processes
- Common design pitfall: throw lots of workers at a task and it will get done faster
  - Works with extreme delicacy in Software Engineering
  - Works better in hardware but makes design much harder

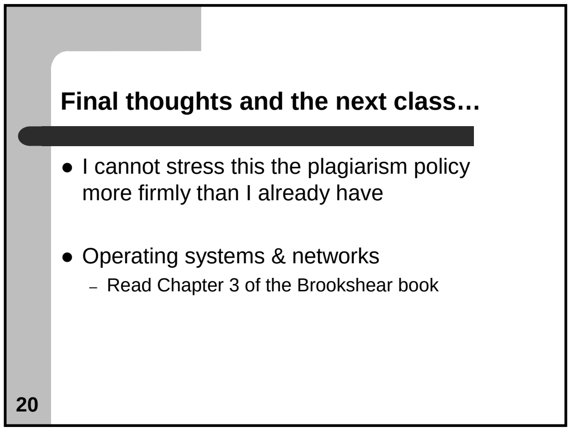
18



Advanced concepts

- SISD – Single Instruction, Single Data
  - typical of what we have seen so far
- MIMD – Multiple Instruction, Multiple Data
  - in multiple processor machines, one processor can store the program information, then call on another processor to complete it
- SIMD – Single Instruction, Multiple Data
  - typically VLIW machines (Very Long Instruction Word)

19



Final thoughts and the next class…

- I cannot stress this the plagiarism policy more firmly than I already have
- Operating systems & networks
  - Read Chapter 3 of the Brookshear book

20