CS 3410: Intro to Computer System Organization and Programming

Kavita Bala

Fall 2008

Computer Science

Cornell University

Information

- Instructor: Kavita Bala (kb@cs.cornell.edu)
- Tu/Th 1:25-2:40
- Hollister B14

© Kavita Bala, Computer Science, Cornell University

Course Objective

- Bridge the gap between hardware and software
  - How a processor works
  - How a computer is organized

- Establish a foundation for building higher-level applications
  - How to understand program performance
  - How to understand where the world is going

© Kavita Bala, Computer Science, Cornell University

Who am I?

- Current life
  - Graphics
  - Parallel processing in graphics

- Previous life
  - Compilers
  - Operating Systems
  - Networks

© Kavita Bala, Computer Science, Cornell University

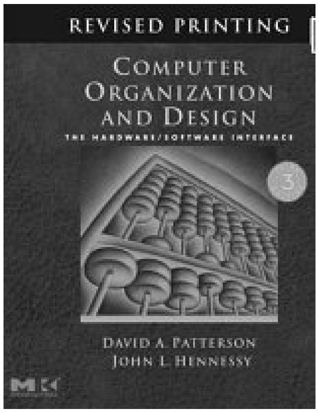
Course Staff

- TAs
  - Adam Arbree (arbree@cs.cornell.edu)
  - Saikat Guha (saikat@cs.cornell.edu)
  - Santosh Selvaraj (ss2346@cornell.edu)
- Undergraduate consultants
  - Steve Milhone
  - Rob Ochshorn
  - Jimmy Qian
- AA: Kelly Patwell (patwell@cs.cornell.edu)
- Sections:
  - Tu/Th 2:55-4:10

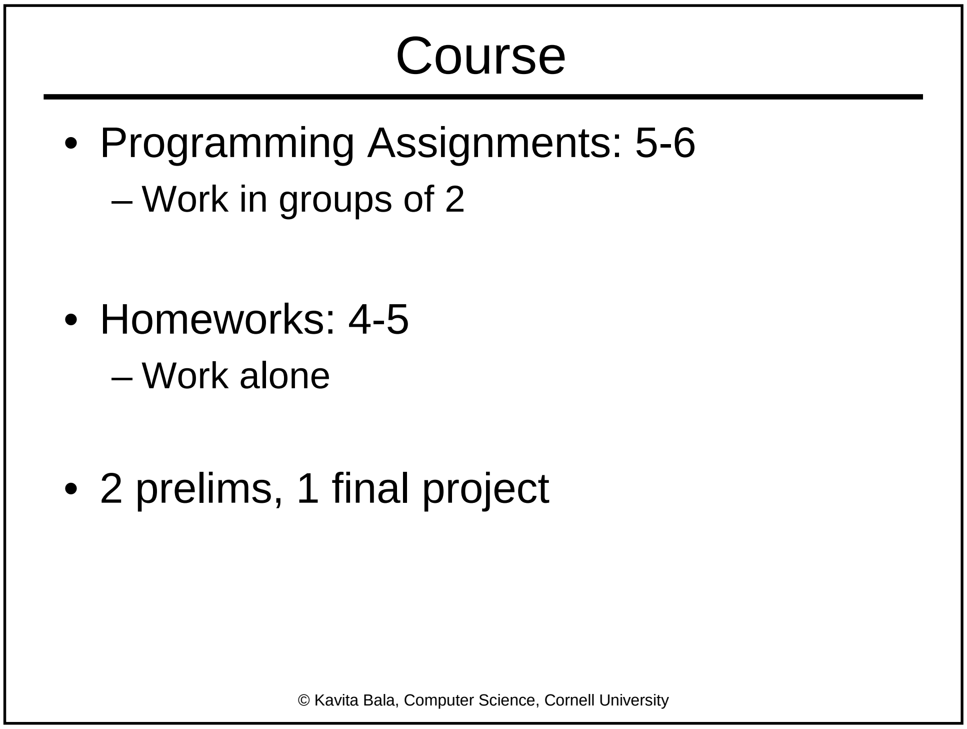
© Kavita Bala, Computer Science, Cornell University

Book

- Computer Organization and Design
  - The Hardware/Software Interface
- David Patterson, John Hennessy
  - Get revised printing from summer 2007


REVISED PRINTING
COMPUTER ORGANIZATION AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
3
DAVID A. PATTERSON
JOHN L. HENNESSY
MK

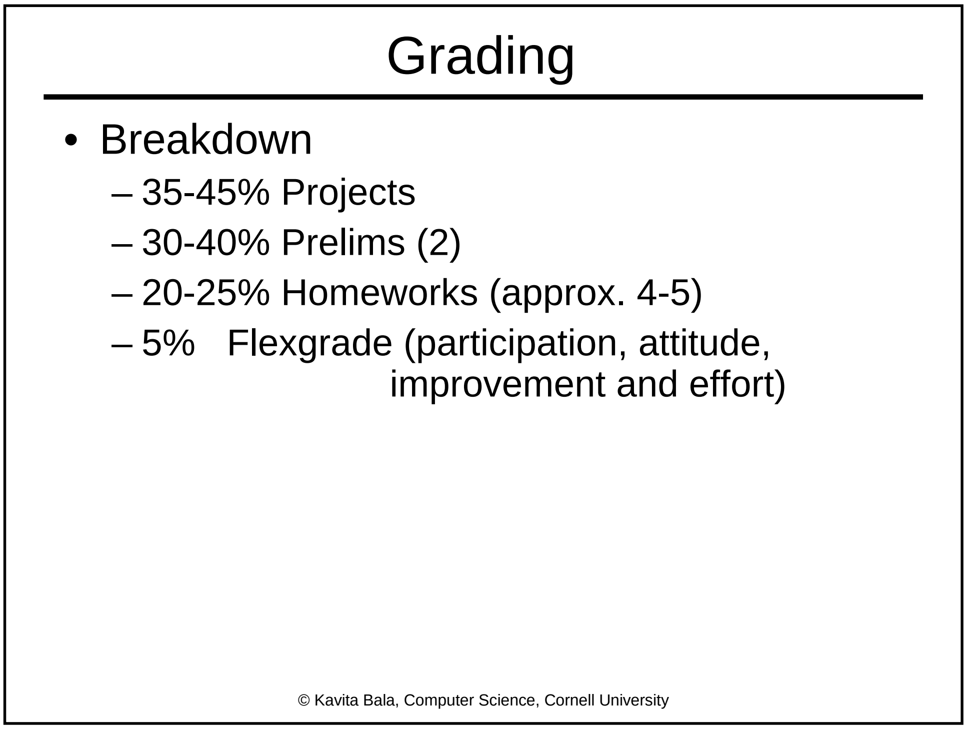
© Kavita Bala, Computer Science, Cornell University



Course

- Programming Assignments: 5-6
  - – Work in groups of 2
- Homeworks: 4-5
  - – Work alone
- 2 prelims, 1 final project

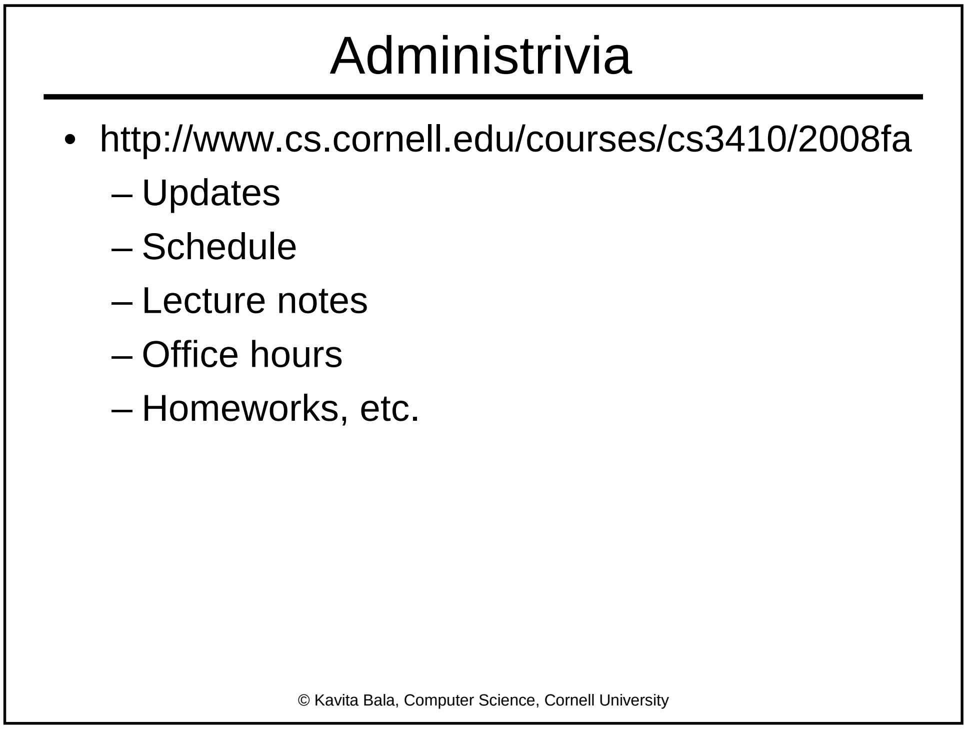
© Kavita Bala, Computer Science, Cornell University



Grading

- Breakdown
  - – 35-45% Projects
  - – 30-40% Prelims (2)
  - – 20-25% Homeworks (approx. 4-5)
  - – 5%   Flexgrade (participation, attitude, improvement and effort)

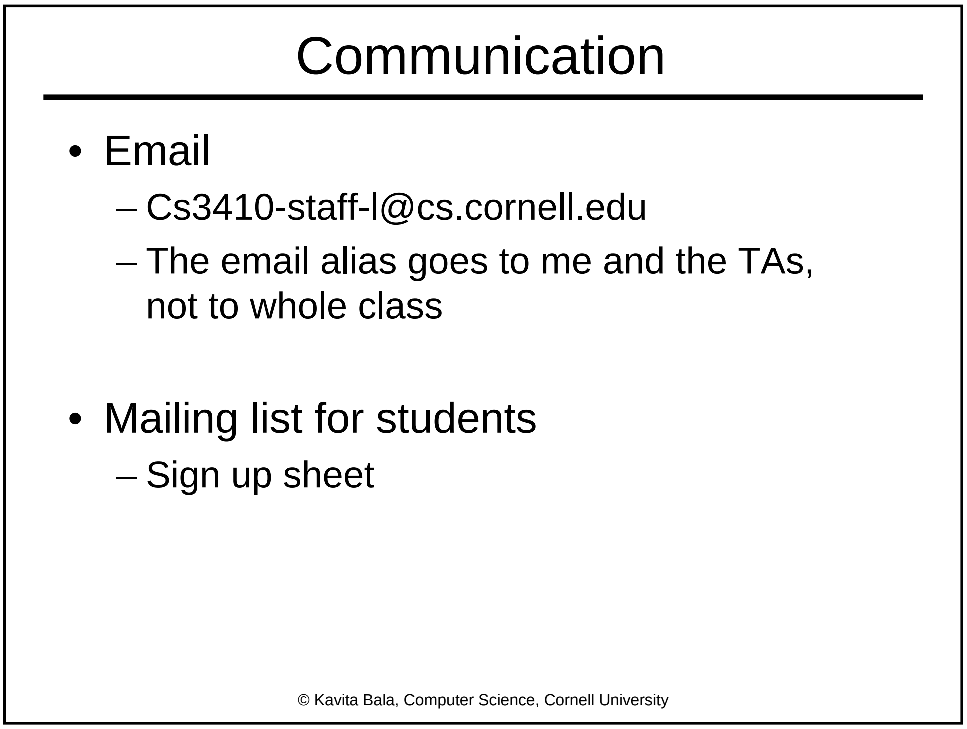
© Kavita Bala, Computer Science, Cornell University



Administrivia

- http://www.cs.cornell.edu/courses/cs3410/2008fa
  – Updates
  – Schedule
  – Lecture notes
  – Office hours
  – Homeworks, etc.

© Kavita Bala, Computer Science, Cornell University



Communication

- Email
  – Cs3410-staff-l@cs.cornell.edu
  – The email alias goes to me and the TAs, not to whole class

- Mailing list for students
  – Sign up sheet

© Kavita Bala, Computer Science, Cornell University

Sections & Projects

- Sections start next week

- Projects will be done in two-person teams
  - We will pair you up if you don’t have a preferred partner
  - Start early, time management is key
  - Manage the team effort

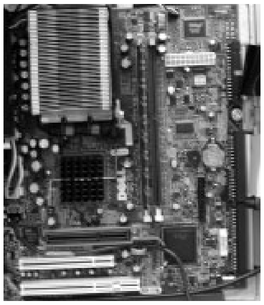
© Kavita Bala, Computer Science, Cornell University

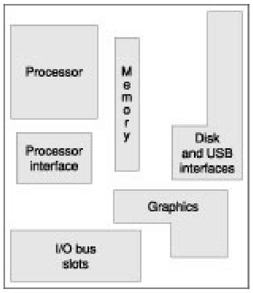
Academic Integrity

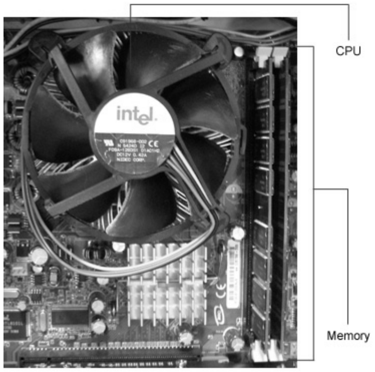
- All submitted work must be your own
  - OK to study together
  - Cannot share solutions however
- Project groups submit joint work
  - Same restrictions apply to projects at the group level
  - Cannot be in possession of someone else’s solution
- Closed-book exams, no calculators

© Kavita Bala, Computer Science, Cornell University

Computer System Organization

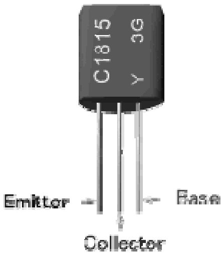


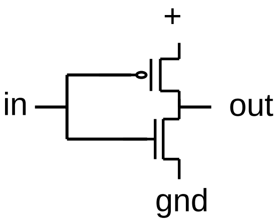




© Kavita Bala, Computer Science, Cornell University

Transistors and Gates



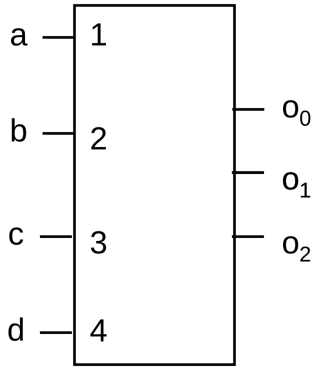


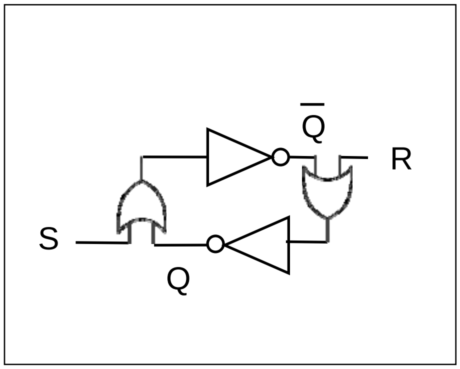
In	Out
0	1
1	0

Truth table

© Kavita Bala, Computer Science, Cornell University

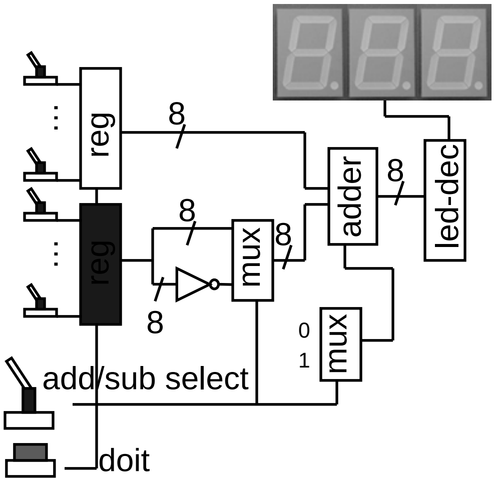
Logic and State





© Kavita Bala, Computer Science, Cornell University

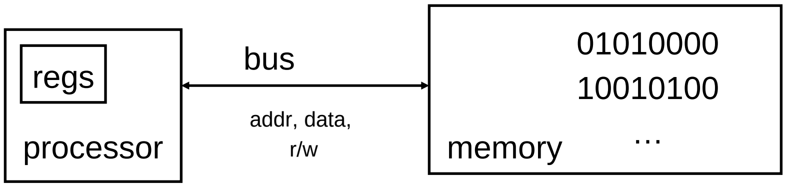
A Calculator



© Kavita Bala, Computer Science, Cornell University

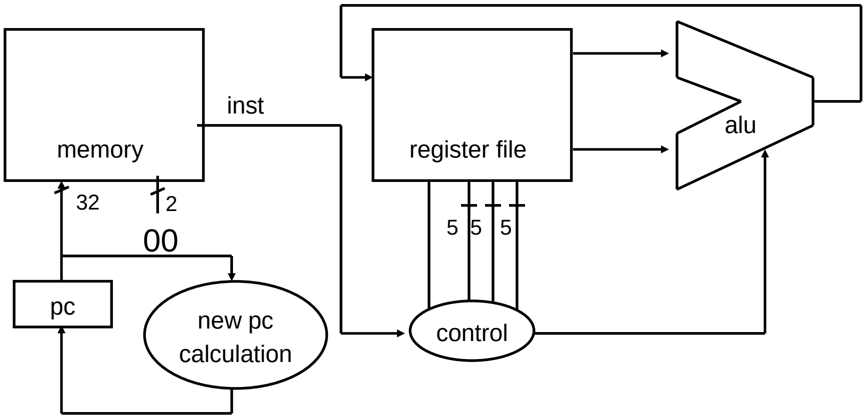
Basic Computer System

- A processor executes instructions
  - Processor has some internal state in storage elements (registers)
- A memory holds instructions and data
  - von Neumann architecture: combined inst and data
- A bus connects the two



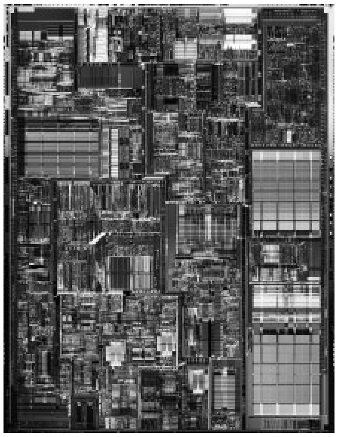
© Kavita Bala, Computer Science, Cornell University

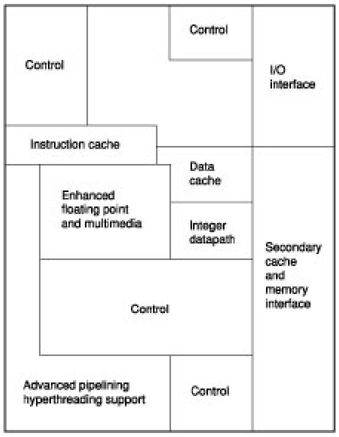
Simple Processor



© Kavita Bala, Computer Science, Cornell University

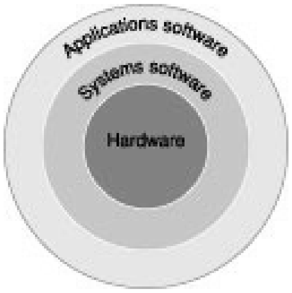
Computer System Organization

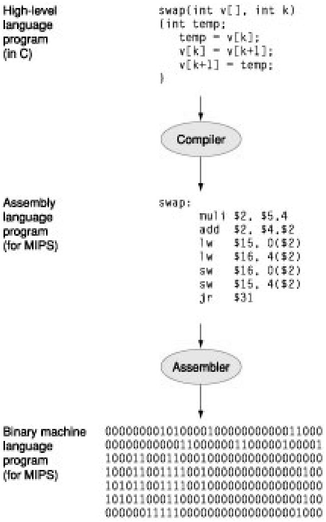



Control
Control
I/O interface
Instruction cache
Data cache
Enhanced floating point and multimedia
Integer datapath
Secondary cache and memory interface
Control
Advanced pipelining hyperthreading support
Control

© Kavita Bala, Computer Science, Cornell University

Computer System Programming




High-level language program (in C)
Compiler
Assembly language program (for MIPS)
Assembler
Binary machine language program (for MIPS)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

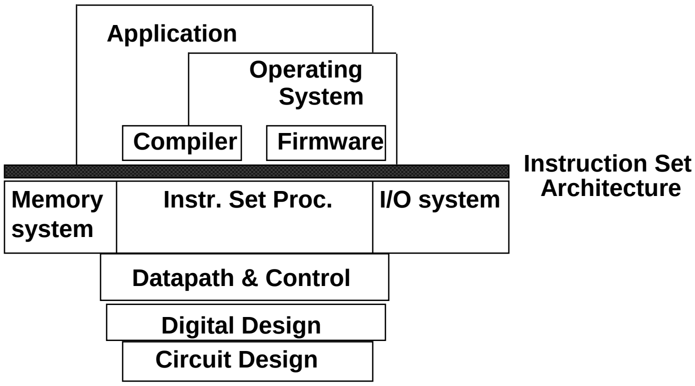
© Kavita Bala, Computer Science, Cornell University

Instruction Set Architecture

- ISA
  - abstract interface between hardware and the lowest level software
  - user portion of the instruction set plus the operating system interfaces used by application programmers

© Kavita Bala, Computer Science, Cornell University

Overview



Application

Operating System

Compiler

Firmware

Instruction Set Architecture

Memory system

Instr. Set Proc.

I/O system

Datapath & Control

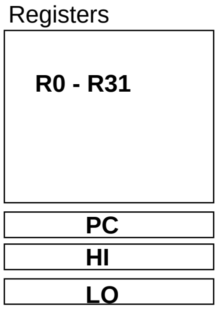
Digital Design

Circuit Design

© Kavita Bala, Computer Science, Cornell University

MIPS R3000 ISA

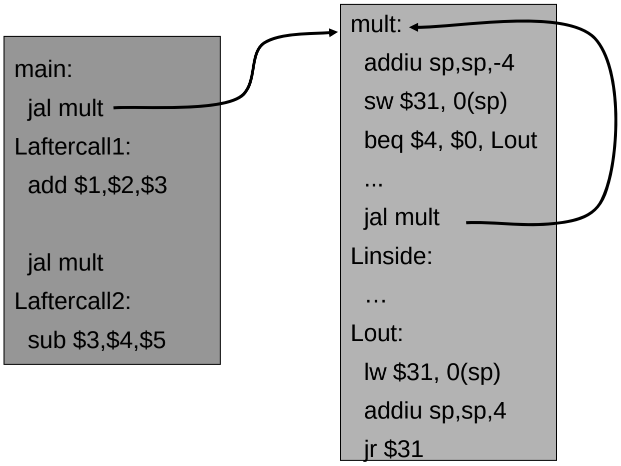
- Instruction Categories
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management



OP	rs	rt	rd	sa	funct
OP	rs	rt	immediate		
OP	jump target				

© Kavita Bala, Computer Science, Cornell University

Calling Conventions



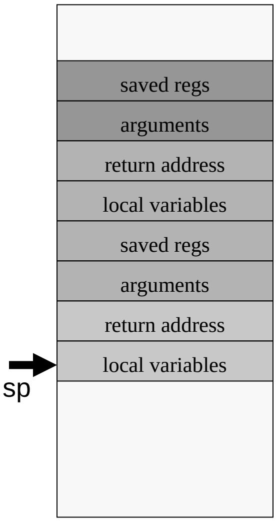
```
main:
  jal mult
Laftercall1:
  add $1,$2,$3

  jal mult
Laftercall2:
  sub $3,$4,$5
```

```
mult:
  addiu sp,sp,-4
  sw $31, 0(sp)
  beq $4, $0, Lout
  ...
  jal mult
Linside:
  …
Lout:
  lw $31, 0(sp)
  addiu sp,sp,4
  jr $31
```

© Kavita Bala, Computer Science, Cornell University

Data Layout



saved regs

arguments

return address

local variables

saved regs

arguments

return address

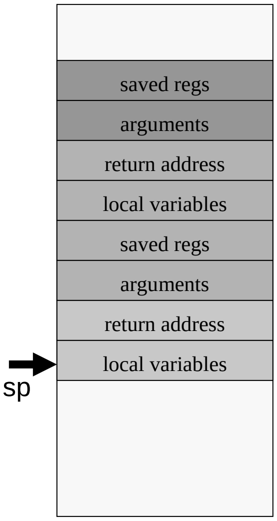
local variables

sp

```
blue() {
   pink(0,1,2,3,4,5);
}
pink() {
    orange(10,11,12,13,14);
}
```

© Kavita Bala, Computer Science, Cornell University

Buffer Overflows



saved regs

arguments

return address

local variables

saved regs

arguments

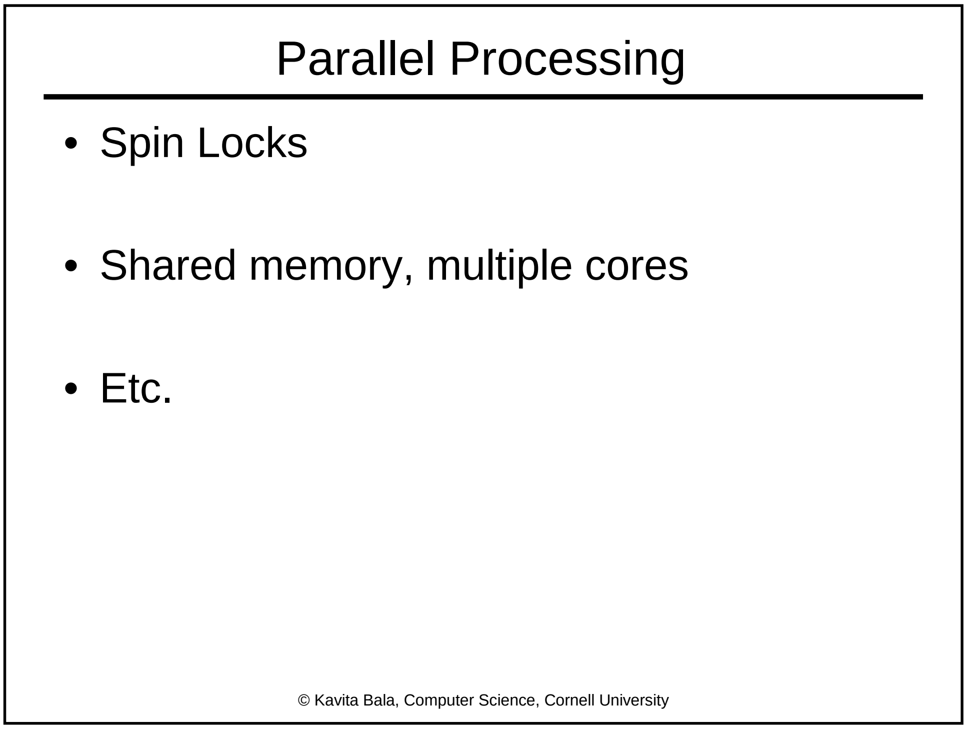
return address

local variables

sp

```
blue() {
   pink(0,1,2,3,4,5);
}
pink() {
    orange(10,11,12,13,14);
}
orange() {
        char buf[100];
        gets(buf); // read string, no check
}
```

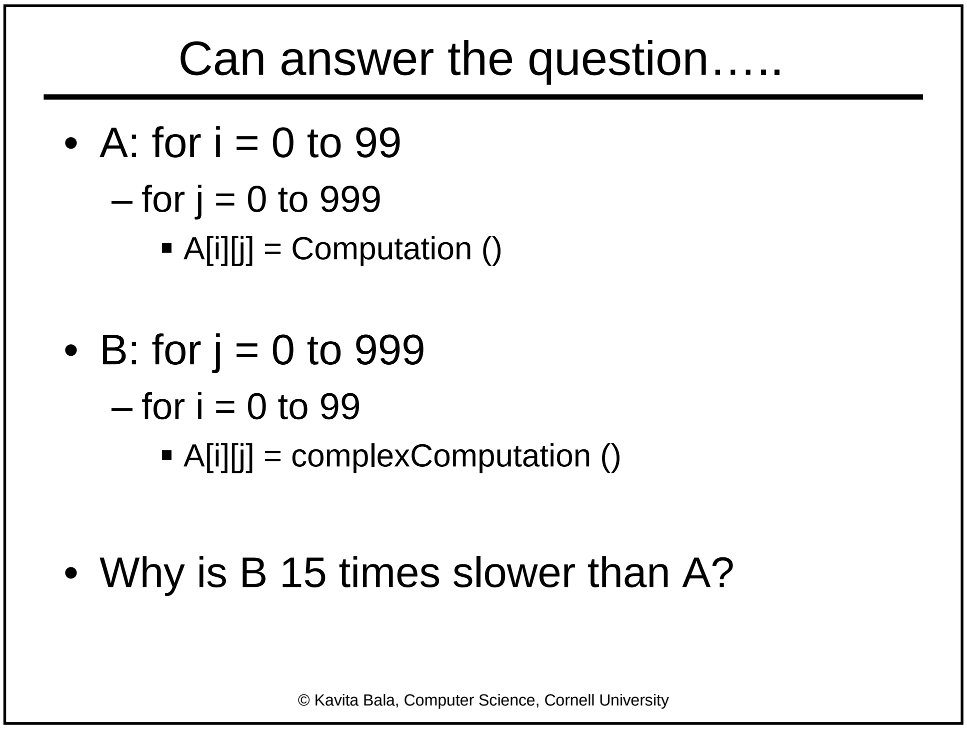
© Kavita Bala, Computer Science, Cornell University



Parallel Processing

- Spin Locks
- Shared memory, multiple cores
- Etc.

© Kavita Bala, Computer Science, Cornell University



Can answer the question…..

- A: for i = 0 to 99
  - for j = 0 to 999
    - A[i][j] = Computation ()
- B: for j = 0 to 999
  - for i = 0 to 99
    - A[i][j] = complexComputation ()
- Why is B 15 times slower than A?

© Kavita Bala, Computer Science, Cornell University

Applications

- Distributed ray tracer
  - Multiple cores running highly parallel application
  - Great images!
- Core war
  - Corrupt your neighbors context!

© Kavita Bala, Computer Science, Cornell University

Why should you care?

- Bridge the gap between hardware and software
  - How a processor works
  - How a computer is organized
- Establish a foundation for building higher-level applications
  - How to understand program performance
  - How to understand where the world is going

© Kavita Bala, Computer Science, Cornell University

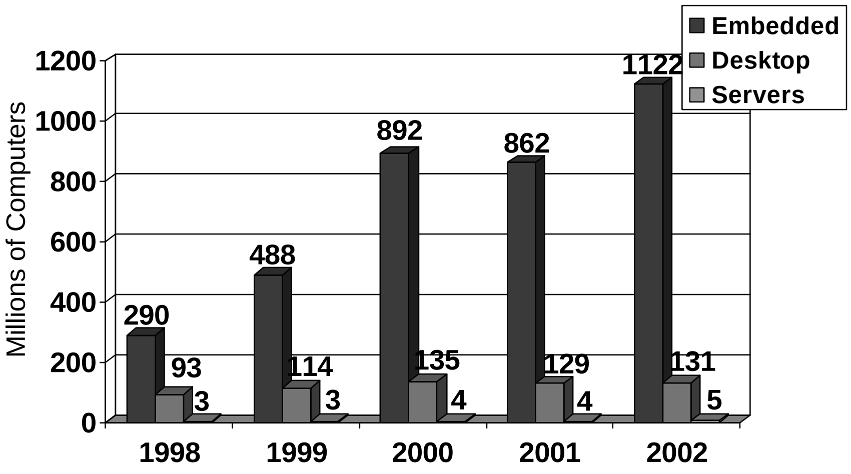
Moore's Law

- 1965
  - number of transistors that can be integrated on a die would double every 18 to 24 months (i.e., grow exponentially with time)

- Amazingly visionary
  - 2300 transistors, 1 MHz clock (Intel 4004) - 1971
  - 16 Million transistors (Ultra Sparc III)
  - 42 Million transistors, 2 GHz clock (Intel Xeon) – 2001
  - 55 Million transistors, 3 GHz, 130nm technology, 250mm^2 die (Intel Pentium 4) – 2004
  - 290+ Million transistors, 3 GHz (Intel Core 2 Duo) – 2007

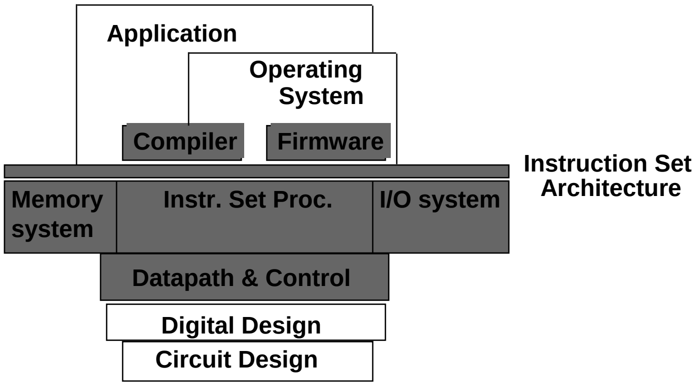
© Kavita Bala, Computer Science, Cornell University

Where is the Market?



© Kavita Bala, Computer Science, Cornell University

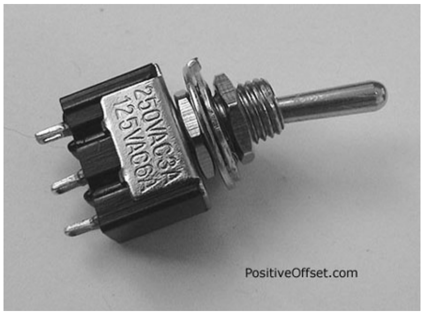
Covered in this course

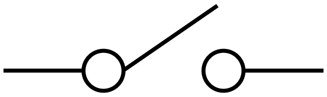


© Kavita Bala, Computer Science, Cornell University

Nuts and Bolts: Switches, Transistors, Gates

A switch

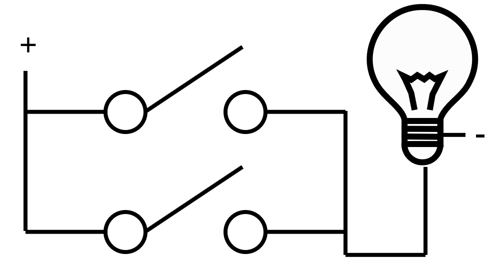




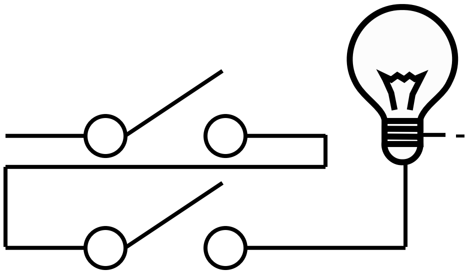
- A switch is a simple device that can act as a conductor or isolator
- Can be used for amazing things…

© Kavita Bala, Computer Science, Cornell University

Switches



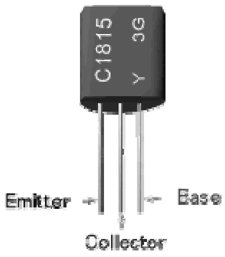
- Either (OR)



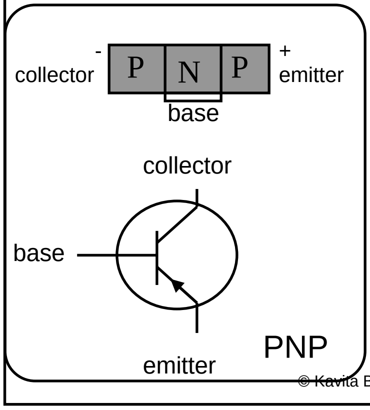
- Both (AND)
- But requires mechanical force

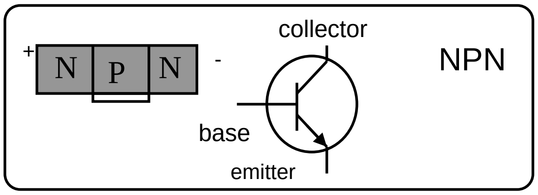
© Kavita Bala, Computer Science, Cornell University

Transistors



- Solid-state switch
  - The most amazing invention of the 1900s
- PNP and NPN

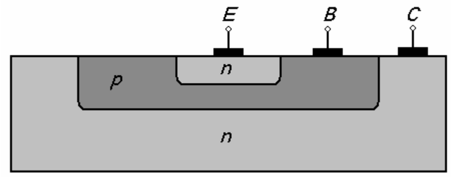


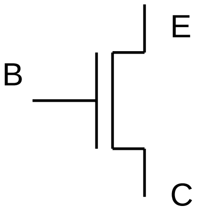


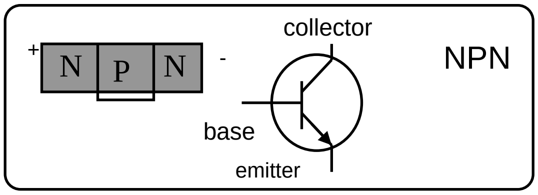
© Kavita Bala, Computer Science, Cornell University

NPN Transistors

- Semi-conductor





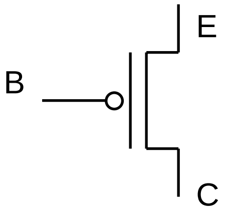


- Connect E to C when base = 1

© Kavita Bala, Computer Science, Cornell University

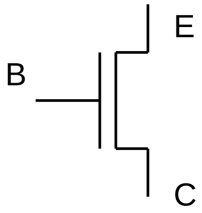
P and N Transistors

- PNP Transistor



- Connect E to C when base = 0

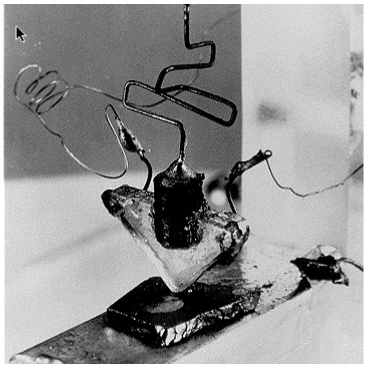
- NPN Transistor



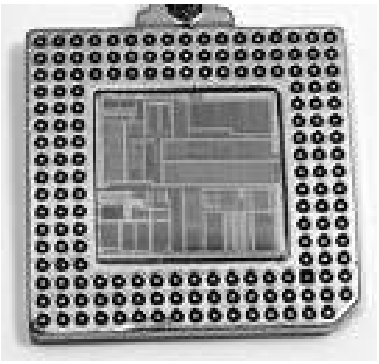
- Connect E to C when base = 1

© Kavita Bala, Computer Science, Cornell University

Then and Now



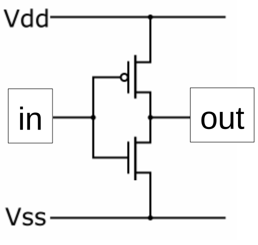
- The first transistor
  - on a workbench at AT&T Bell Labs in 1947



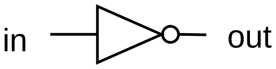
- An Intel Pentium
  - 125 million transistors

© Kavita Bala, Computer Science, Cornell University

Inverter



- Function: NOT
- Called an inverter
- Symbol:



In	Out
0	1
1	0

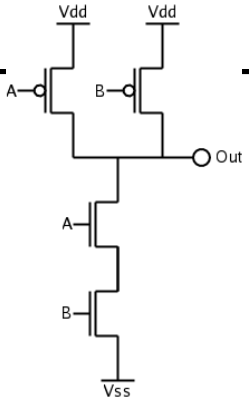
Truth table

- Useful for taking the inverse of an input

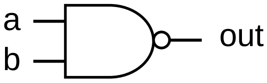
- CMOS: complementary-symmetry metal**–oxide–semiconductor**

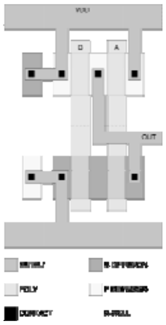
© Kavita Bala, Computer Science, Cornell University

NAND Gate



- Function: NAND
- Symbol:

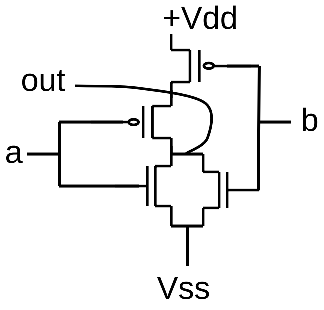




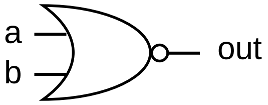
A	B	out
0	0	1
1	0	1
0	1	1
1	1	0

© Kavita Bala, Computer Science, Cornell University

NOR Gate



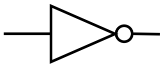
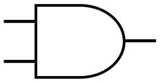
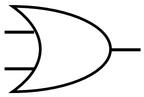
- Function: NOR
- Symbol:



A	B	out
0	0	1
1	0	0
0	1	0
1	1	0

© Kavita Bala, Computer Science, Cornell University

Building Functions

- NOT: 
- AND: 
- OR: 
- NAND and NOR are universal
  - Can implement any function with NAND or just NOR gates
  - useful for manufacturing

© Kavita Bala, Computer Science, Cornell University