EECS150 - Digital Design
Lecture 1 - Introduction

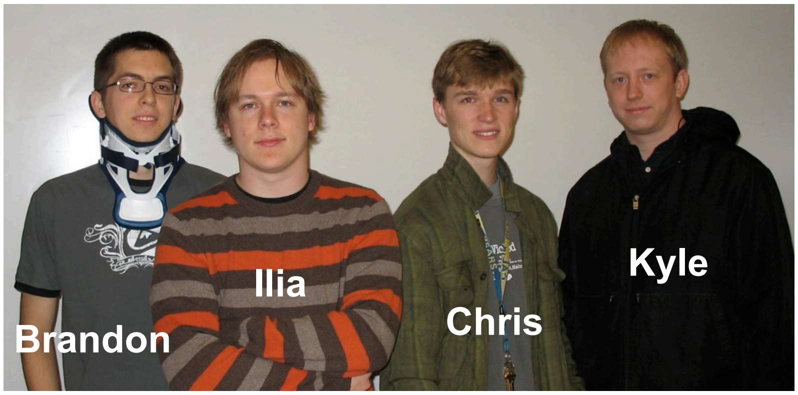
January 19, 2010

John Wawrzynek
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www-inst.eecs.berkeley.edu/~cs150

Teaching Staff

Professor John Wawrzynek
(Warznek)
631 Soda Hall
johnw@cs.berkeley.edu
Office Hours: Tu, Th 1-2pm, & by appointment.



All TA office hours held in 125 Cory. Check website for days and times.

Outline

- Enrollment & Attendance
- Course Materials & Content
- Course Structure & Grading
- A Few Basic Principles of Digital Design
- Begin 61C review

Enrollment

- If you are enrolled and plan to take the course you must attend your lab section next week, if not you will be dropped from the class roster.

- If you are on the waitlist - we have room for you, however, you must:
  1. Have taken the prerequisites - CS61C (required) & EECS40 (recommended).
  2. Attend lectures and do the homework, the first two weeks.
  3. In the second week of classes, go to the lab section in which you wish to enroll. Give the TA your name and student ID. If the lab section is full [the TA will tell you if so], you must find a different section.
  4. Later, we will process the waitlist based on these requests, and lab section openings.
  5. Note: if you are not on the waitlist, you will not be considered for enrollment.

- No lab [or discussion] sections this week. Yes, lab lecture this Friday, 2-3pm, 125 Cory Hall.

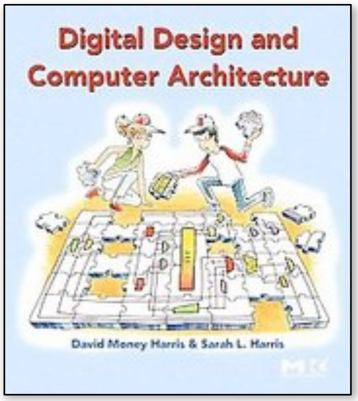
Attendance

- Attend regular lectures and ask questions, offer comments, etc.
- Attend weekly lab lecture (Friday 2pm, 125 Cory).
- Attend your lab section. You must stick with the same lab section all semester.
  - Lab exercises will be done individually; project with a partner.
  - We will put together a lab section exchange in a few weeks to help you move to a different section.
- Attend any discussion section. You may attend any discussion section that you want regardless of which one you are enrolled in.
- The entire teaching staff hold regular office hours (see class webpage). Take advantage of this opportunity!

Lab Lecture

- Friday 2-3pm. Held in the lab - 125 Cory, so we can do demonstrations.
- This is an important part of the course.
  - You get background and practical information regarding the lab exercises and project checkpoints.
- Also, we will have a mandatory short quiz at the beginning of each lab lecture.
  - Quiz will be based on one of the weekly homework problems.
  - Two quiz scores for each student will be dropped at the end of the semester, so you can miss two quizzes (save this option for important dates – like job interview trips, etc.)

Course Materials



Textbook: **Harris** & **Harris**

Publisher: Morgan Kaufmann

- Class notes, homework & lab assignments, solutions, and other documentation will be available on the class webpage linked to the calendar:

  http://www-inst.eecs.berkeley.edu/~cs150

  - Check the class webpage and newsgroup often!
  - You are responsible for checking the class webpage at least once every 24 hours [in case we need to post changes/corrections.]

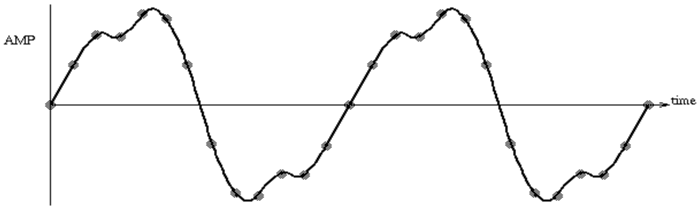
Course Content

Components and Design Techniques for **Digital Systems**

more specifically

Synchronous Digital Hardware Systems

- Synchronous: "Clocked" - all changes in the system are controlled by a global clock and happen at the same time (not asynchronous)
- Digital: All inputs/outputs and internal values (signals) take on discrete values (not analog).
  - Example digital representation: music waveform



  - **A series of numbers is used to represent the waveform, rather than a voltage or current, as in analog systems.**

Course Content – Design Layers

Not a course on computer architecture or the architecture of other systems. Although we will look at these as examples.

High-level Organization : Hardware Architectures

System Building Blocks : Arithmetic units, controllers

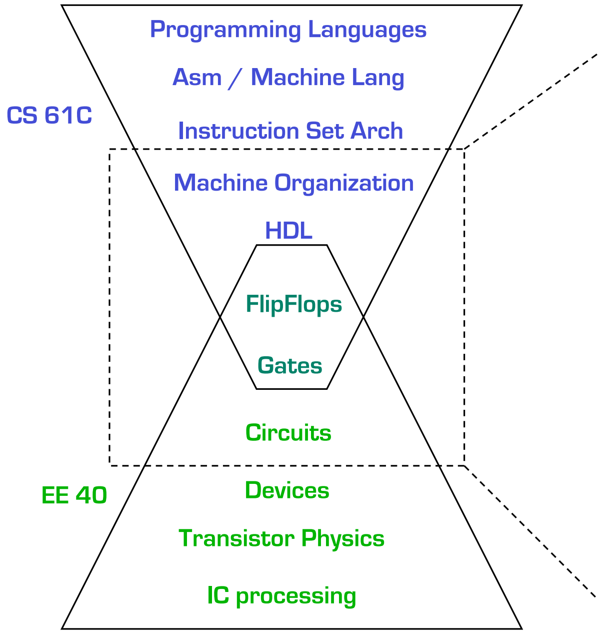
Circuit Elements : Memories, logic blocks

Transistor-level circuit implementations

Circuit primitives : Transistors, wires

Not a course on transistor physics and transistor circuits. Although, we will look at these to better understand the primitive elements for digital circuits.

Course Content

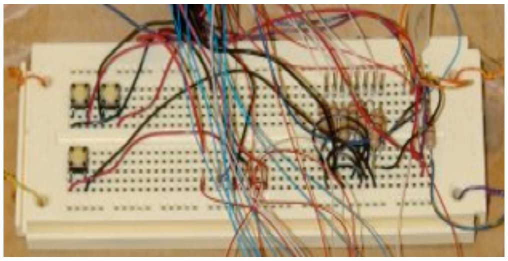


Deep Digital Design Experience

- Fundamentals of Boolean Logic
- Synchronous Circuits
- Finite State Machines
- Timing & Clocking
- Device Technology & Implications
- Controller Design
- Arithmetic Units
- Encoding, Framing
- Testing, Debugging
- Hardware Architecture
- Hardware Design Language (HDL)
- Design Flow (CAD)

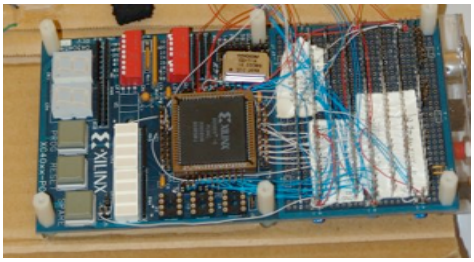
Course Evolution

- Final project circa 1980:
  - Example project: pong game with buttons for paddle and LEDs for output.
  - Few 10's of logic gates
  - Gates hand-wired together on "bread-board" (protoboard).
  - No computer-aided design tools
  - Debugged with oscilloscope and logic analyzer

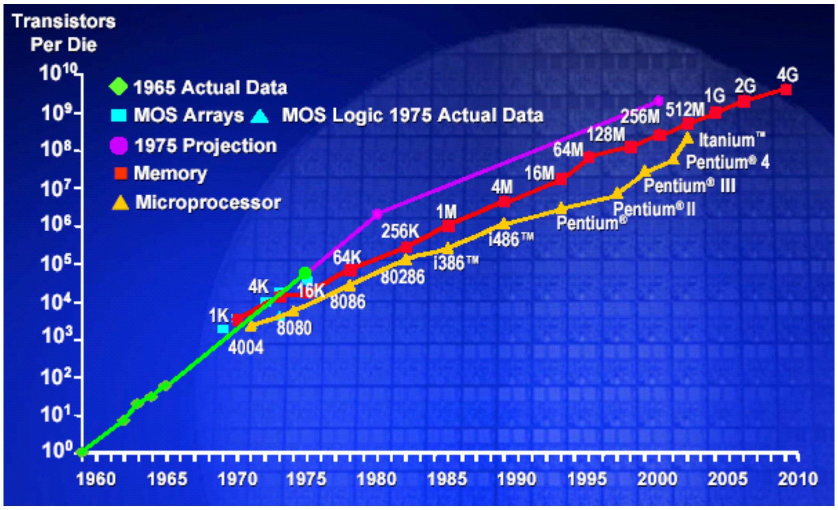


Course Evolution

- Final project circa 1995:
  - Example project: MIDI music synthesizer
  - Few 1000's of logic gates
  - Gates wired together internally on field programmable gate array (FPGA) development board with some external components.
  - Circuit designed "by-hand", computer-aided design tools to help map the design to the hardware.
  - Debugged with circuit simulation, oscilloscope and logic analyzer

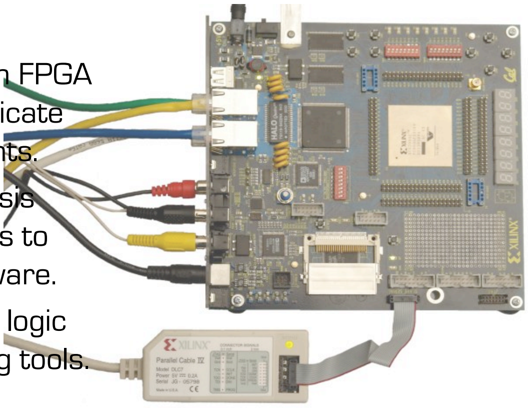


Moore's Law – 2x stuff per 1-2 yr



Course Evolution

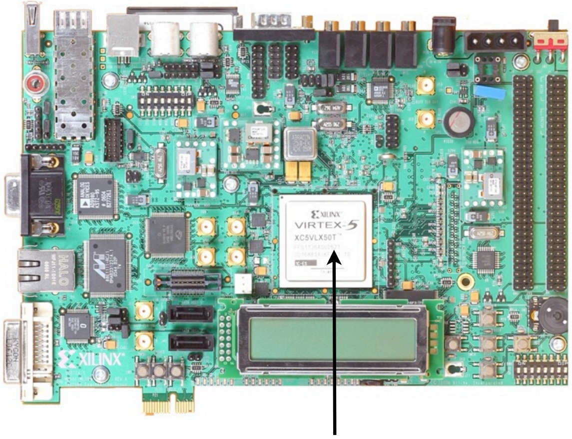
Calinx Board



- Final project circa 2000-2008:
  - Example project: eTV – streaming video broadcast over Ethernet, student project decodes and displays video
  - Few 10,000's of logic gates
  - Gates wired together internally on FPGA development board and communicate with standard external components.
  - Circuit designed with logic-synthesis tools, computer-aided design tools to help map the design to the hardware.
  - Debugged with circuit simulation, logic analyzer, and in-system debugging tools.

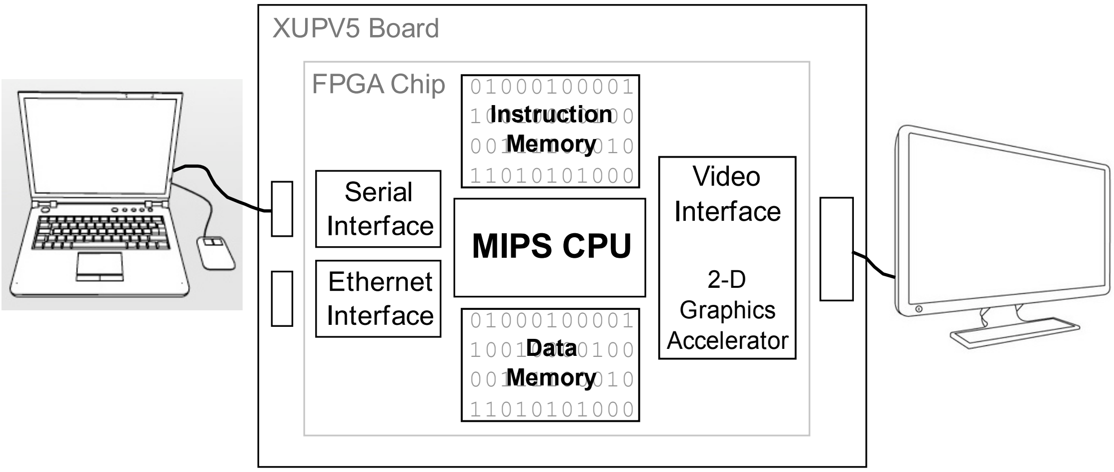
Course Evolution

- Beginning 2009:
  - Xilinx XUPV5 development board (a.k.a ML505)
  - Could enable very aggressive final projects.
  - But, modest use of resources this semester.
  - Project debugging with simulation tools and with in-system hardware debugging tools.



- State-of-the-art LX110T FPGA: ~1M logic gates.
  - Interfaces: Audio in/out, digital video, ethernet, on-board DRAM, PCIe, USB, …

Final Project: Spring 2010



XUPV5 Board

FPGA Chip

Instruction Memory

Serial Interface

Ethernet Interface

MIPS CPU

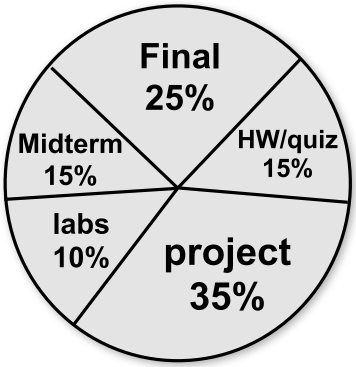
Video Interface

2-D Graphics Accelerator

Data Memory

- Executes most commonly used MIPS instructions.
- Pipelined (high performance) implementation.
- Serial console interface for shell interaction, debugging.
- Ethernet interface for high-speed file transfer.
- Video interface for display with 2-D vector graphics acceleration.
- Supported by a C language compiler.

Course Grading



- Project graded on timeliness, completeness, and optimality.
- Midterm Exam, evening of Wed March 31.
- Comprehensive final exam in exam slot.

- Weekly homework based on reading and lectures.
  - graded on effort only,
  - out at the end of each week, due before next week lab lecture.
- Weekly quiz closely related to one of the homework problems. Given at the beginning of the lab lecture.
  - Most of "HW/quiz" grade points based on quiz grades.
- Lab exercises for weeks 2-5, followed by project checkpoints and final checkoff.
- Labs and checkpoints due at the beginning of your next lab session.

Tips on How to Get a Good Grade

The lecture material is not the most challenging part of the course.

- You should be able to understand everything as we go along.
- Do not fall behind in lecture and tell yourself you "will figure it out later from the notes or book".
- Notes will be online before the lecture (usually the night before). Look at them before class. Do assigned reading (only the required sections).
- Ask questions in class if you don't understand. If you are not getting it then probably others aren't. **Come to office hours to get help.**
- The exams will test your depth of knowledge. You need to understand the material well enough to apply it in new situations (beyond the homework). The homework is a starting point, not the ending point.

You need to do well on the project to get a good course grade.

- Take the labs very seriously. They are an integral part of the course.
- Choose your partner carefully. Your best friend may not be the best choice.
- Most important (this comes from 30+ years of hardware design experience):
  - **Be well organized and neat.**
  - **Add complexity a little bit at a time - always have a working design.**
  - **Don't be afraid to throw away your design and start fresh.**

Course Structure

A week in the life of a EECS150 student

Monday (for example):	
Discussion section	1 hours
Tuesday: Lecture	1.5
Wednesday (for example):	
Lab section	3
Thursday: Lecture	1.5
Friday: Lab Lecture	1
Reading book, reviewing notes	3
Homework	4
TOTAL	15 hours/week

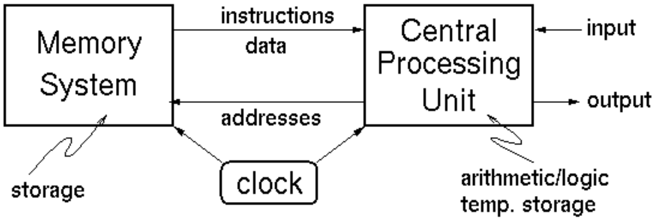
Extra time in lab once project starts.

Cheating

- We have posted the details of my cheating policy on the class web site. Please read it and ask questions.
- If you turn in someone else's work as if it were your own, you are guilty of cheating. This includes homework sets, answers on exams, verilog code, block diagrams, etc.
- Also, if you knowingly aid in cheating, you are guilty.
- We have software that automatically compares your submitted work to others.
- However, it is okay to discuss with others lab exercises and the project. Okay to work together on homework. But everyone must turn in their own work.
- If we catch you cheating, I will give you an F on the assignment. If it is a midterm exam, final exam, or final project, I will give you an F in the class. You will be reported to the office of student conduct. If you have a previous case of cheating on your record, I will push to have you expelled from the University.

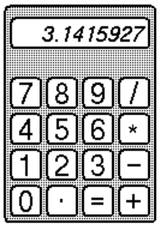
Example Digital Systems

- General Purpose Digital Computer



  - Often designed to maximize performance. "Optimized for speed"

- Handheld Calculator



- Usually designed to minimize cost.
“Optimized for low cost”

- Of course, low cost comes at the expense of speed.

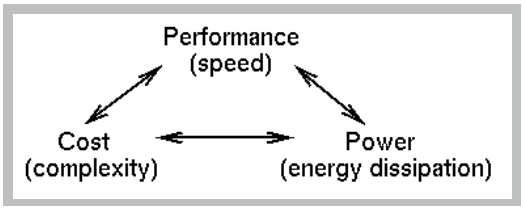
Example Digital Systems

- Digital Watch

Designed to minimize power.
Single battery must last for years.

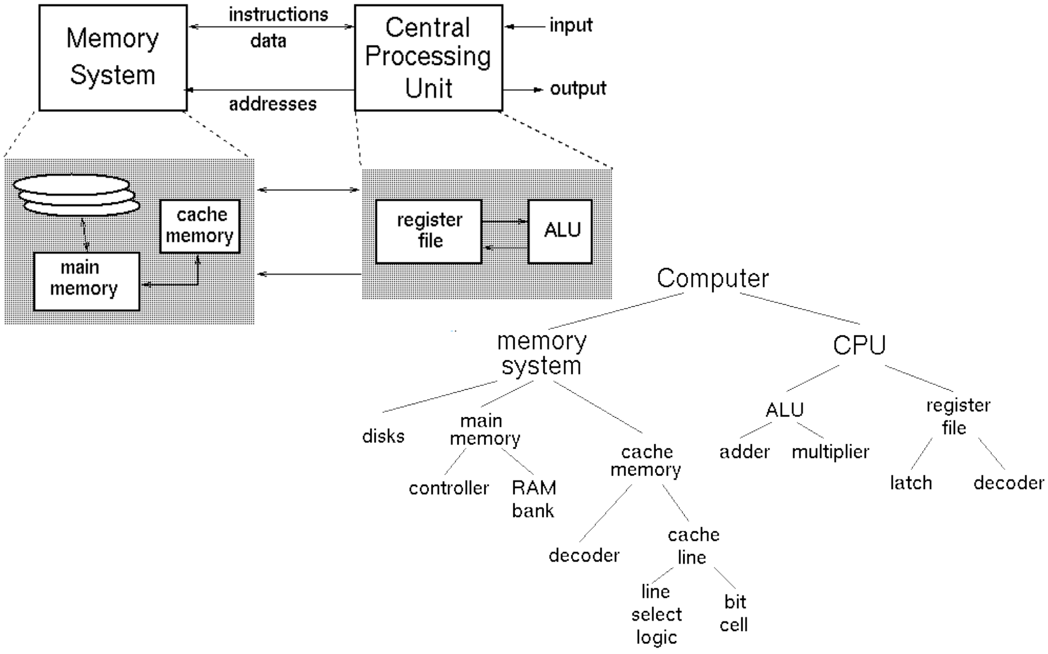
  - Low power operation comes at the expense of:
    - lower speed
    - higher cost

Basic Design Tradeoffs

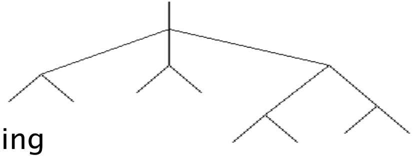


- You can improve on one at the expense of worsening one or both of the others.
- These tradeoffs exist at every level in the system design - every sub-piece and component.
- Design Specification -
  - Functional Description.
  - Performance, cost, power constraints.
- As a designer you must make the tradeoffs necessary to achieve the function within the constraints.

Hierarchy & Design Representation



Hierarchy in Designs



- Helps control complexity -
  - by hiding details and reducing the total number of things to handle at any time.
- Modulalizes the design -
  - divide and conquer
  - simplifies implementation and debugging
- Top-Down Design
  - Starts at the top (root) and works down by successive refinement.
- Bottom-up Design
  - Starts at the leaves & puts pieces together to build up the design.
- Which is better?
  - In practice both are needed & used.
    - Need top-down divide and conquer to handle the complexity.
    - Need bottom-up because in a well designed system, the structure is influence by what primitives are available.

Digital Design: what's it all about?

Given a functional description and performance, cost, & power constraints, come up with an implementation using a set of primitives.

- How do we learn how to do this?
  1. Learn about the primitives and how to use them.
  2. Learn about design representations.
  3. Learn formal methods to optimally manipulate the representations.
  4. Look at design examples.
  5. Use trial and error - CAD tools and prototyping. Practice!
- Digital design is in some ways more an art than a science. The creative spirit is critical in combining primitive elements & other components in new ways to achieve a desired function.
- However, unlike art, we have objective measures of a design:

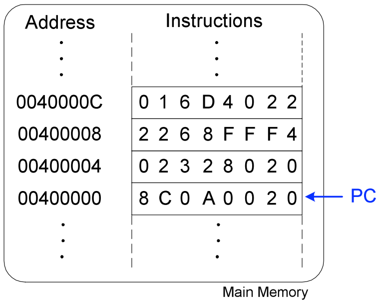
Performance Cost Power

Key 61c Concept: “Stored Program” Computer

- Instructions and data stored in memory.
- Only difference between two applications (for example, a text editor and a video game), is the sequence of instructions.
- To run a new program:
  - No rewiring required
  - Simply store new program in memory
  - The processor hardware executes the program:
  - fetches (reads) the instructions from memory in sequence
  - performs the specified operation
- The program counter (PC) keeps track of the current instruction.

Assembly Code	Machine Code
`lw   $t2, 32($0)`	`0x8C0A0020`
`add  $s0, $s1, $s2`	`0x02328020`
`addi $t0, $s3, -12`	`0x2268FFF4`
`sub  $t0, $t3, $t5`	`0x016D4022`

Stored Program



Address	Instructions
⋮	⋮
0040000C	0 1 6 D 4 0 2 2
00400008	2 2 6 8 F F F 4
00400004	0 2 3 2 8 0 2 0
00400000	8 C 0 A 0 0 2 0
⋮	⋮

Key 61c Concept:
High-level languages help productivity.

High-level code

```
// add the numbers from 0 to 9
int sum = 0;
int i;

for (i=0; i!=10; i = i+1) {
  sum = sum + i;
}
```

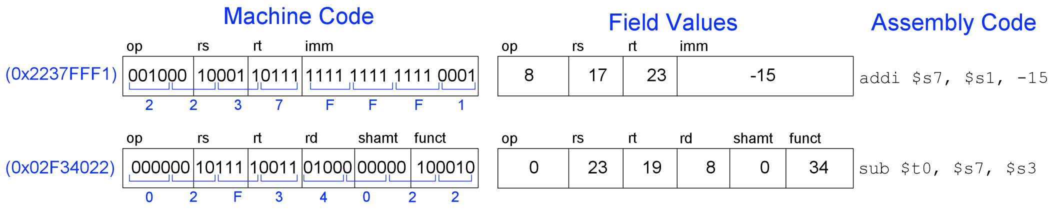
MIPS assembly code

```
# $s0 = i, $s1 = sum
       addi $s1, $0, 0
       add  $s0, $0, $0
       addi $t0, $0, 10
for:   beq  $s0, $t0, done
       add  $s1, $s1, $s0
       addi $s0, $s0, 1
       j    for
done:
```

Therefore with the help of a compiler (and assembler), to run applications all we need is a means to interpret (or “execute”) machine instructions. Usually the application calls on the operating system and libraries to provide special functions.

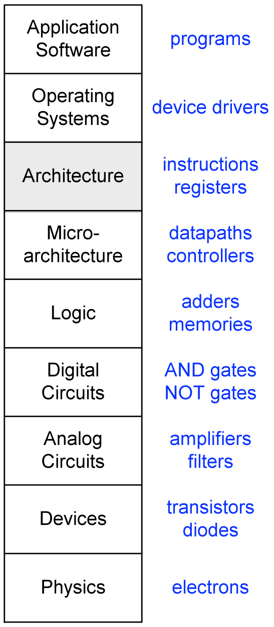
Interpreting Machine Code

- Start with opcode
- Opcode tells how to parse the remaining bits
- If opcode is all 0's
  - R-type instruction
  - Function bits tell what instruction it is
- Otherwise
  - opcode tells what instruction it is



A processor is a machine code interpreter build in hardware!

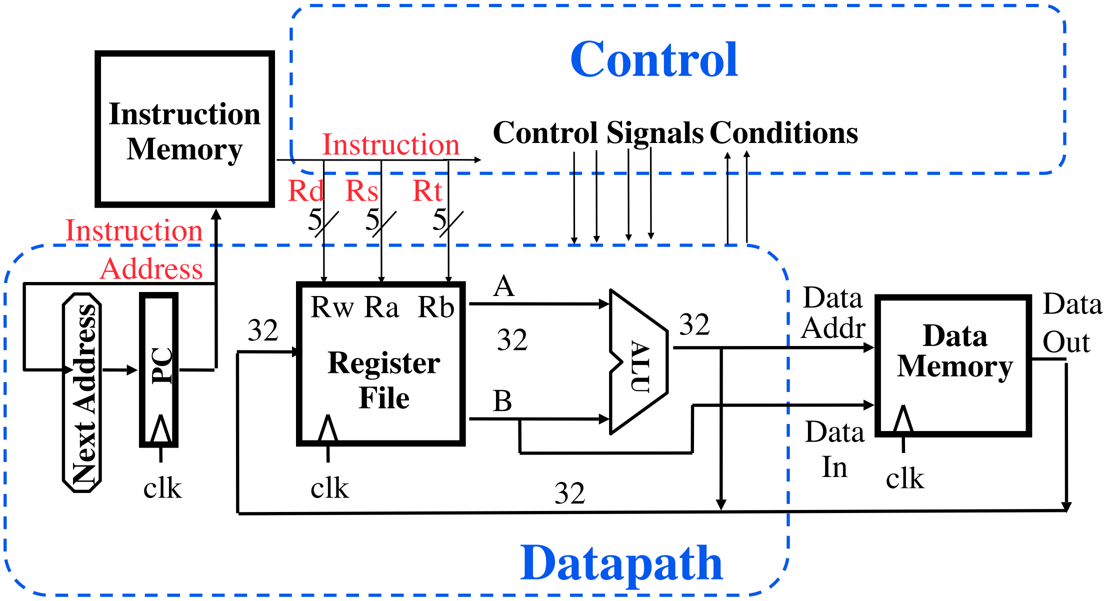
Abstraction Layers



Application Software	programs
Operating Systems	device drivers
Architecture	instructions registers
Micro-architecture	datapaths controllers
Logic	adders memories
Digital Circuits	AND gates NOT gates
Analog Circuits	amplifiers filters
Devices	transistors diodes
Physics	electrons

- **Architecture**: the programmer's view of the computer
  - Defined by instructions (operations) and operand locations
- **Microarchitecture**: how to implement an architecture in hardware (covered in great detail later)
- The microarchitecture is built out of "logic" circuits and memory elements (this semester).
- All logic circuits and memory elements are implemented in the physical world with transistors.
- This semester we will implement our projects using circuits on FPGAs (field programmable gate arrays).

Abstract View of MIPS Implementation



How do we implement these various pieces?

Extra Slides

MIPS Register Definitions

Name	Register Number	Usage
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2-3	procedure return values
$a0-$a3	4-7	procedure arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved variables
$t8-$t9	24-25	more temporaries
$k0-$k1	26-27	OS temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	procedure return address

Instruction Format Review

R-Type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

J-Type

op	addr
6 bits	26 bits

R-Type Instructions

- *Register-type*
- 3 register operands:
  - `rs`, `rt`: source registers
  - `rd`: destination register
- Other fields:
  - `op`: the *operation code* or *opcode* (0 for R-type instructions)
  - `funct`: the *function*
    together, the opcode and function tell the computer what operation to perform
  - `shamt`: the *shift amount* for shift instructions, otherwise it's 0

R-Type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

R-Type Examples

Assembly Code

```
add $s0, $s1, $s2
sub $t0, $t3, $t5
```

Field Values

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32
0	11	13	8	0	34
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Machine Code

op	rs	rt	rd	shamt	funct	
000000	10001	10010	10000	00000	100000	(0x02328020)
000000	01011	01101	01000	00000	100010	(0x016D4022)
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Note the order of registers in the assembly code:

```
add rd, rs, rt
```

I-Type Instructions

- *Immediate-type*
- 3 operands:
  - `rs`, `rt`: register operands
  - `imm`: 16-bit two's complement immediate
- Other fields:
  - `op`: the opcode
  - Simplicity favors regularity: all instructions have opcode
  - Operation is completely determined by the opcode

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

I-Type Examples

Assembly Code	Field Values			
	op	rs	rt	imm
`addi $s0, $s1, 5`	8	17	16	5
`addi $t0, $s3, -12`	8	19	8	-12
`lw   $t2, 32($0)`	35	0	10	32
`sw   $s1,  4($t1)`	43	9	17	4
	6 bits	5 bits	5 bits	16 bits

Note the differing order of registers in the assembly and machine codes:

```
addi rt, rs, imm
lw   rt, imm(rs)
sw   rt, imm(rs)
```

Machine Code

op	rs	rt	imm	
001000	10001	10000	0000 0000 0000 0101	(0x22300005)
001000	10011	01000	1111 1111 1111 0100	(0x2268FFF4)
100011	00000	01010	0000 0000 0010 0000	(0x8C0A0020)
101011	01001	10001	0000 0000 0000 0100	(0xAD310004)
6 bits	5 bits	5 bits	16 bits	

J-Type

- *Jump-type*
- 26-bit address operand (`addr`)
- Used for jump instructions (`j`)

J-Type

op	addr
6 bits	26 bits

MIPS150 Project Instruction Summary (SP09)

mnemonic	*description*	*type*
`lw`	load word	I
`sw`	store word	I
`beq`	branch if equal	I
`bne`	branch if not equal	I
`addu`	add	R
`subu`	subtract	R
`or`	bitwise or	R
`slt`	set less than	R
`sll`	shift left logical	R
`sra`	shift right arithmetic	R
`addiu`	add immediate	I
`andi`	and immediate	I
`ori`	or immediate	I
`jr`	jump register	R
`jal`	jump and link	J