CS152 – Computer Architecture and Engineering

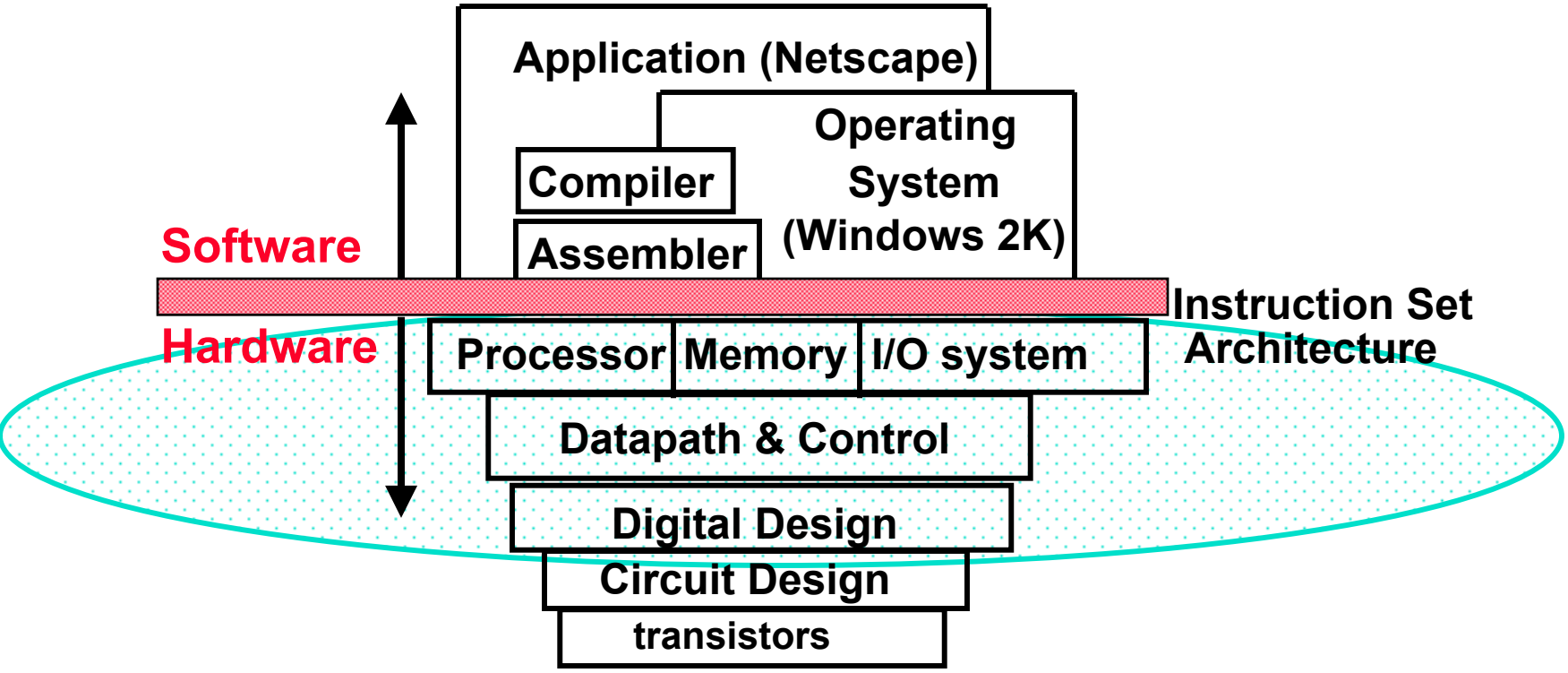
Lecture 1 – Introduction & MIPS Review

2003-08-26

Dave Patterson
(www.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs152/

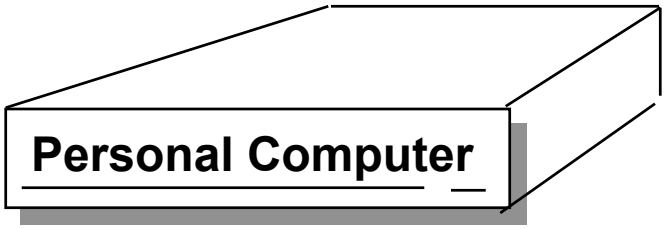
Where is "Computer Architecture and Engineering"?

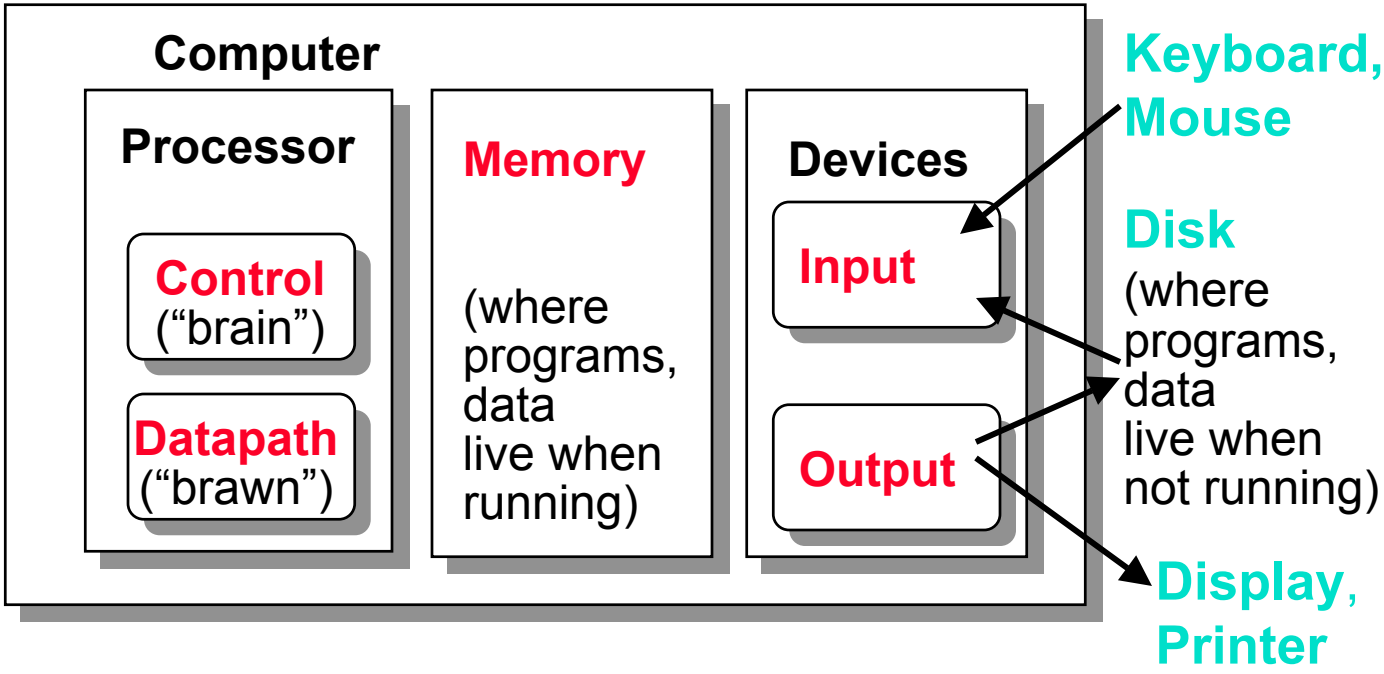


152

* Coordination of many *levels of abstraction*

Anatomy: 5 components of any Computer

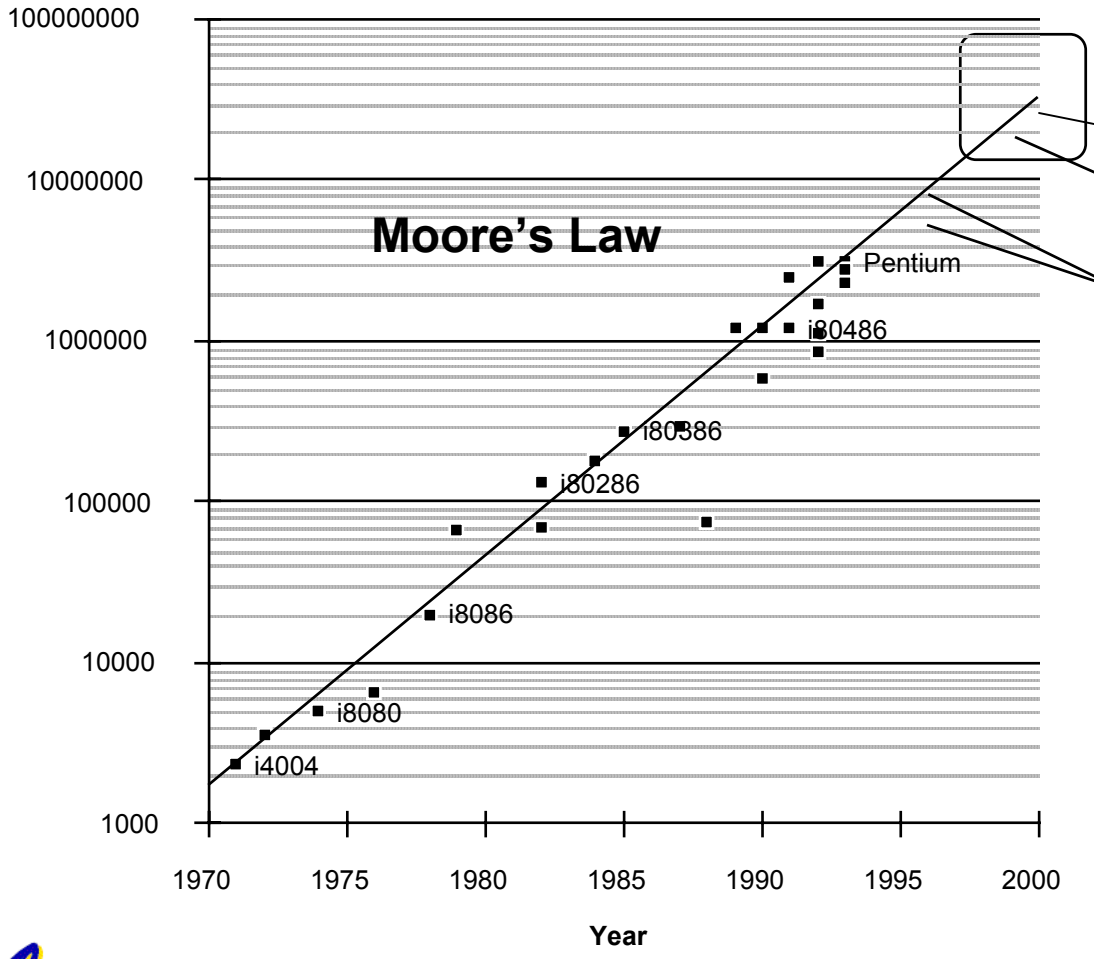




Computer Technology - Dramatic Change!

- Processor
  - 2X in speed every 1.5 years (since ‘85); 100X performance in last decade.
- Memory
  - DRAM capacity: 2x / 2 years (since ‘96); 64x size improvement in last decade.
- Disk
  - Capacity: 2X / 1 year (since ‘97)
  - 250X size in last decade.

Technology Trends: Microprocessor Complexity



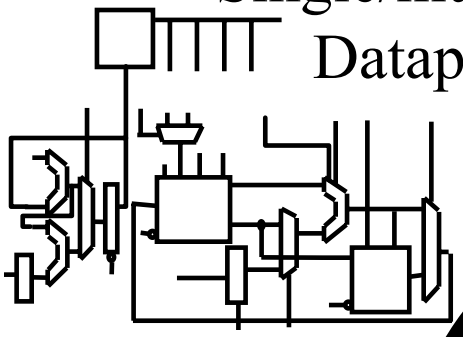
Athlon (K7): 22 Million
Alpha 21264: 15 million
Pentium Pro: 5.5 million
PowerPC 620: 6.9 million
Alpha 21164: 9.3 million
Sparc Ultra: 5.2 million

2X transistors/Chip Every 1.5 years

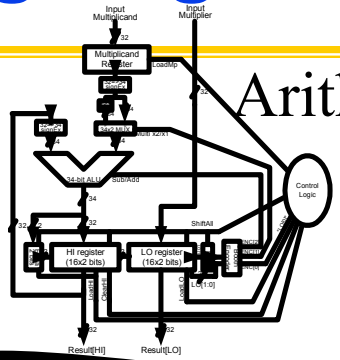
Called "Moore's Law"

Where are we going??

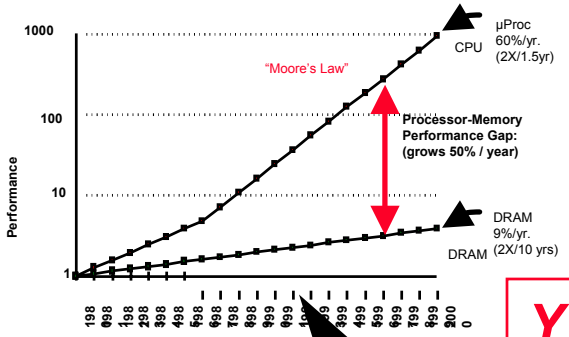
Single/multicycle Datapaths

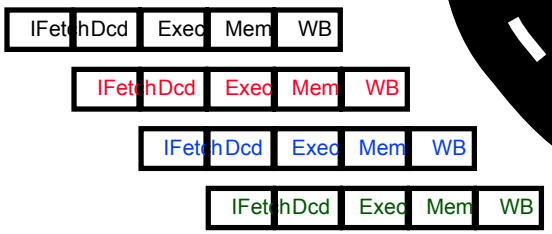


Arithmetic

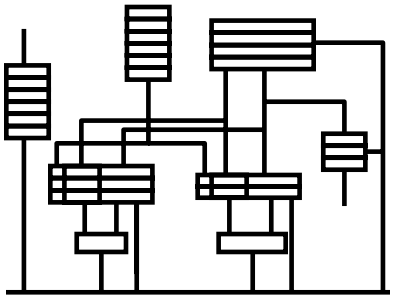


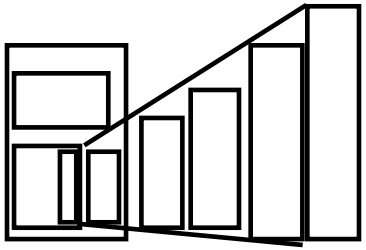
CS152 Fall '03



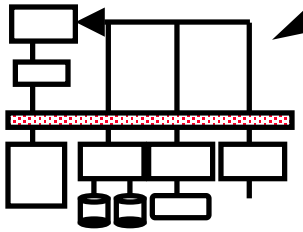


Pipelining





Memory Systems



I/O

YOUR CPU

Project Focus

° Design Intensive Class ---
100 to 200 hours per semester per student
MIPS Instruction Set ---> FPGA implementation

° Modern CAD System:
Schematic capture and Simulation

Design Description

Computer-based "breadboard"
- Behavior over time
- Before construction

Xilinx FPGA board
- Running design at 25 MHz to 50 MHz

(~ state-of-the-art clock rate a decade ago)

Project Simulates Industrial Environment

- Project teams have 4 or 5 members in same discussion section
  - Must work in groups in "the real world"
- Communicate with colleagues (team members)
  - Communication problems are natural
  - What have you done?
  - What answers you need from others?
  - You must document your work!!!
  - Everyone must keep an on-line notebook
- Communicate with supervisor (TAs)
  - How is the team's plan?
  - Short progress reports are required:
    - What is the team's game plan?
    - What is each member's responsibility?

CS152: So what's in it for me?

- In-depth understanding of the inner-workings of computers & trade-offs at HW/SW boundary
  - Insight into fast/slow operations that are easy/hard to implement in hardware (HW)
  - Out of order execution and branch prediction
- Experience with the *design process* in the context of a large complex (hardware) design.
  - Functional Spec --> Control & Datapath --> Physical implementation
  - Modern CAD tools
  - Make 32-bit RISC processor in actual hardware
- Learn to work as team, with manager (TA)
- Designer's "Conceptual" toolbox.

Conceptual tool box?

- Evaluation Techniques
- Levels of translation (e.g., Compilation)
- Levels of Interpretation (e.g., Microprogramming)
- Hierarchy (e.g, registers, cache, mem,disk,tape)
- Pipelining and Parallelism
- Static / Dynamic Scheduling
- Indirection and Address Translation
- Synchronous /Asynchronous Control Transfer
- Timing, Clocking, and Latching
- CAD Programs, Hardware Description Languages, Simulation
- Physical Building Blocks (e.g., Carry Lookahead)
- Understanding Technology Trends / FPGAs

Texts

- Required: *Computer Organization and Design: The Hardware/Software Interface, Beta Version 3rd Edition*, Patterson and Hennessy (COD): Free; hand out in class in 2 or 3 volumes.
  - Just want feedback, learn mistakes
- Reading assignments on web page

http://inst.eecs.berkeley.edu/~cs152

Format: Lecture - Disc - Lecture - Lab

- Mon Labs, Homeworks due
  - Lab 1 due Wed 9/3 since Mon 9/1 is holiday
- Tue Lecture
- Wed (later in semester) Design Doc. Due
- Thu Lecture
- Fri Discussion Section/Lab demo

  There IS discussion this week…;
- Prerequisite Quiz this Friday

TAs

- Jack Kang
  - jackkang@uclink.berkeley.edu
- John Gibson
  - jgibson@uclink.berkeley.edu
- Kurt Meinz
  - kurtm@mail.com

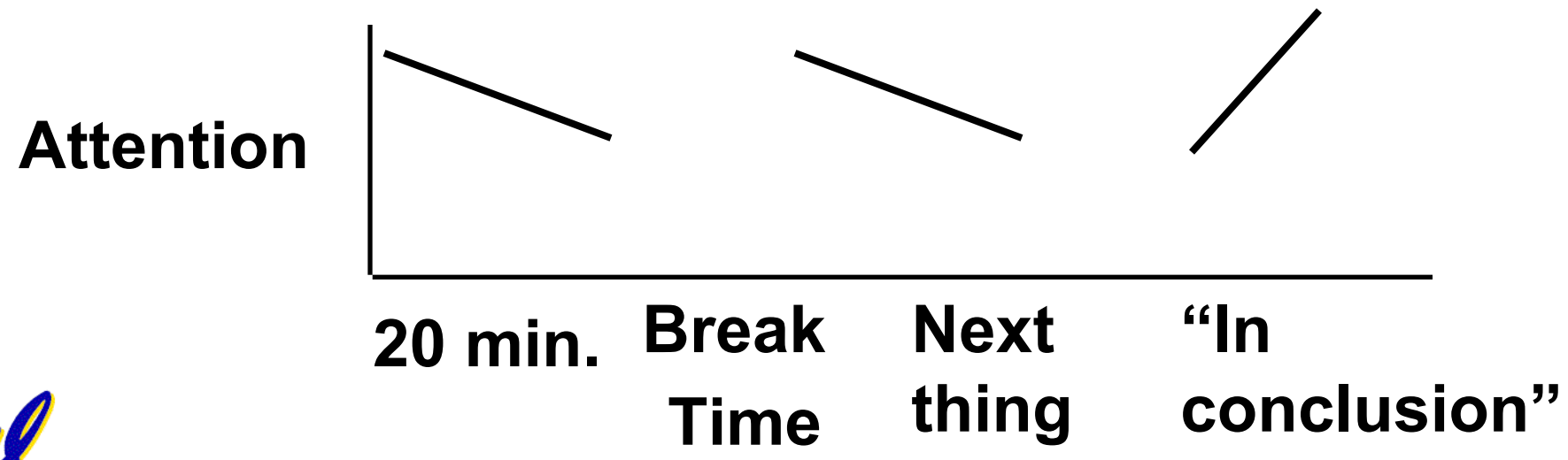
3 Discussion Sections

1. 11 AM - 1 PM 320 Soda (John)
2. 2 PM - 4 PM 4 Evans (Kurt)
3. 3 PM - 5 PM 81 Evans (Jack)

- 2-hour discussion section for later in term. Early sections may end in 1 hour. Make sure that you are free for both hours however!
- Project team must be in same section!

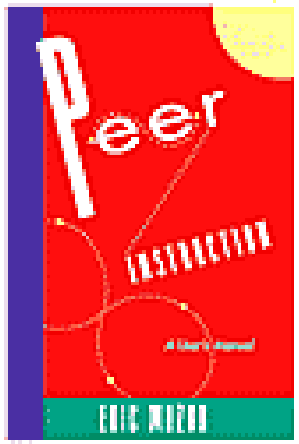
Typical 80-minute Lecture Format

- 18-Minute Lecture + 2-Min admin break
- 20-Minute Lecture + 10-Min Peer instruct.
- 25-Minute Lecture + 5-Min wrap-up
- We'll come to class early & try to stay after to answer questions

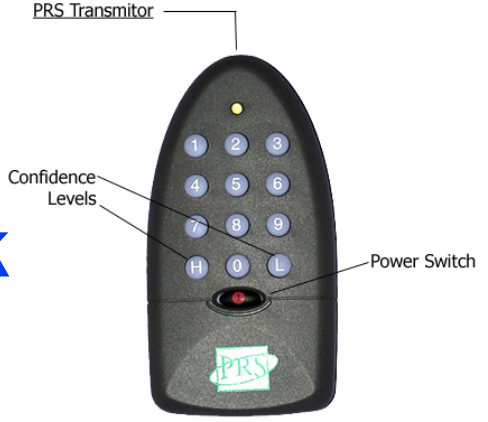


Tried-and-True Technique: Peer Instruction

- Increase real-time learning in lecture, test understanding of concepts vs. details



- As complete a "segment" ask multiple choice question
  - 1-2 minutes to decide yourself
  - 3-4 minutes in pairs/triples to reach consensus. Teach each other!
  - 2-3 minute discussion of answers, questions, clarifications



- By 9/2: buy $37 PRS Transmitor from behind ASUC textbook desk (Chem 1A, CS 61ABC, 160)

Peer Instruction

- Read textbook, review lectures (new or old) before class
  - Reduces examples have to do in class
  - Get more from lecture
    - also good advice in general
- Fill out 3 question Web Form on reading (deadline 11am before lecture)
  - Graded for correctness
  - Will start next week, since Prerequisite Quiz this week

Homeworks and Labs/Projects

- **Homework exercises** (every 2 weeks)
- **Lab Projects** (every ~2 weeks)
  - Lab 1 Write diagnostics to debug bad SPIM
  - Lab 2 Multiplier Design + Intro to hardware synthesis on FPGA board
  - Lab 3 Single Cycle Processor Design
  - Lab 4 Pipelined Processor Design
  - Lab 5/6 Cache (3 weeks; Read Only 1st)
  - Lab 7 Advanced pipelined modules
- All exercises, reading, homeworks, projects on course web page

Project/Lab Summary

- Tool Flow runs on PCs in 119 and 125 Cory, but 119 Cory is primary CS152 lab
- Get instructional UNIX/PC account now ("name account"); get in discussion
- Get card-key access to Cory now (3rd floor)
- End of semester Project finale:
  - Demo
  - Oral Presentation
  - Head-to-head Race
  - Final Report

Course Exams

- Reduce the pressure of taking exams
  - Midterms: Wed October 8th and Wed Nov. 19th in 1 LeConte
  - 3 hrs to take 1.5-hr test (5:30-8:30 PM)
  - Our goal: test knowledge vs. speed writing
  - Review meetings: Sunday/Tues before?
  - Both mid-terms can bring summary sheets
- Students/Staff meet over pizza after exam at LaVals!
  - Allow me to meet you
  - I'll buy!

Grading

- Grade breakdown
  - Two Midterm Exams: 35% (combined)
  - Labs and Design Project: 45%
  - Homework Reading Quiz: 5%
  - Project Group Participation: 5%
  - Class Participation (PRS): 10%
- No late homeworks or labs: our goal grade, return in 1 week
- Grades posted on home page/glookup?
  - Written/email request for changes to grades
- EECS GPA guideline upper div. class: 2.7 to 3.1
  - average 152 grade = B/B+; set expectations accordingly

Course Problems…Cheating

- What is cheating?
  - Studying together in groups is encouraged
  - Work must be your own
  - Common examples of cheating: work together on wording of answer to homework, running out of time on a assignment and then pick up output, take homework from box and copy, person asks to borrow solution "just to take a look", copying an exam question, …
- Homeworks/labs/projects
  # points varies; -10 cheat
- Inform Office of Student Conduct

My Goal

- Show you how to understand modern computer architecture in its rapidly changing form.
- Show you how to design by leading you through the process on challenging design problems
- Learn how to test things.
- NOT just to talk at you. So ...
  - ask questions
  - come to office hours

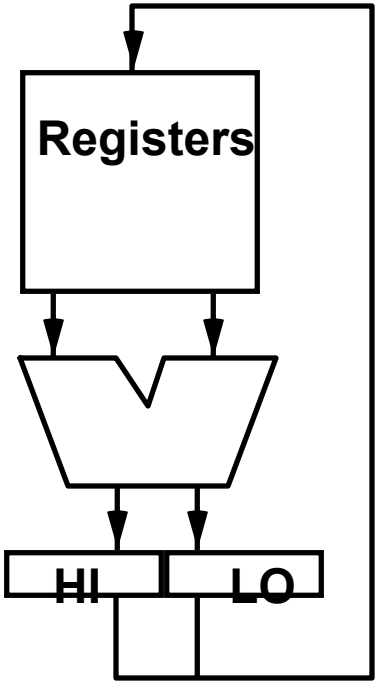
MIPS I
Instruction set

MIPS I Operation Overview

- **Arithmetic Logical:**
  - Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU
  - AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, LUI
  - SLL, SRL, SRA, SLLV, SRLV, SRAV
- **Memory Access:**
  - LB, LBU, LH, LHU, LW, LWL,LWR
  - SB, SH, SW, SWL, SWR

Multiply / Divide

- Start multiply, divide
  - MULT rs, rt
  - MULTU rs, rt
  - DIV rs, rt
  - DIVU rs, rt
- Move result from multiply, divide
  - MFHI rd
  - MFLO rd
- Move to HI or LO
  - MTHI rd
  - MTLO rd



Q: Why not Third field for destination?

Data Types

Bit: 0, 1

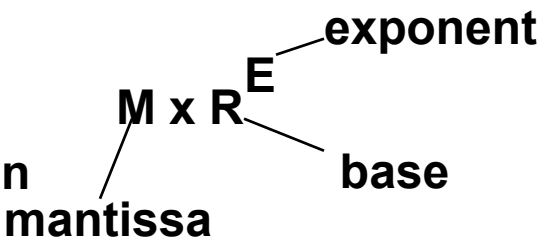
Bit String: sequence of bits of a particular length
- 4 bits is a nibble
- 8 bits is a byte
- 16 bits is a half-word
- 32 bits is a word
- 64 bits is a double-word

Character:
- ASCII 7 bit code
- UNICODE 16 bit code

Decimal:
- digits 0-9 encoded as 0000b thru 1001b
- two decimal digits packed per 8 bit byte

Integers:
- 2's Complement

Floating Point:
- Single Precision
- Double Precision
- Extended Precision



How many +/- #'s?
Where is decimal pt?
How are +/- exponents represented?

MIPS arithmetic instructions

Instruction	*Example*	*Meaning*	*Comments*
add	add $1,$2,$3	$1 = $2 + $3	3 operands; exception possible
subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands; exception possible
add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant; exception possible
add unsigned	addu $1,$2,$3	$1 = $2 + $3	3 operands; no exceptions
subtract unsigned	subu $1,$2,$3	$1 = $2 – $3	3 operands; no exceptions
add imm. unsign.	addiu $1,$2,100	$1 = $2 + 100	+ constant; no exceptions
multiply	mult $2,$3	Hi, Lo = $2 x $3	64-bit signed product
multiply unsigned	multu$2,$3	Hi, Lo = $2 x $3	64-bit unsigned product
divide	div $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Lo = quotient, Hi = remainder
divide unsigned	divu $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Unsigned quotient & remainder
Move from Hi	mfhi $1	$1 = Hi	Used to get copy of Hi
Move from Lo	mflo $1	$1 = Lo	Used to get copy of Lo

Q: Which add for address arithmetic? Which add for integers?

MIPS logical instructions

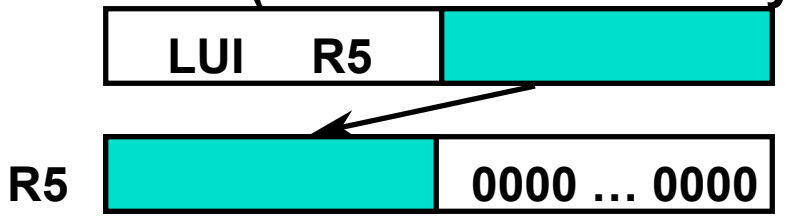
Instruction	*Example*	*Meaning*	*Comment*
and	and $1,$2,$3	$1 = $2 & $3	3 reg. operands; Logical AND
or	or $1,$2,$3	$1 = $2 | $3	3 reg. operands; Logical OR
xor	xor $1,$2,$3	$1 = $2 ^ $3	3 reg. operands; Logical XOR
nor	nor $1,$2,$3	$1 = ~($2 |$3)	3 reg. operands; Logical NOR
and immediate	andi $1,$2,10	$1 = $2 & 10	Logical AND reg, constant
or immediate	ori $1,$2,10	$1 = $2 | 10	Logical OR reg, constant
xor immediate	xori $1, $2,10	$1 = ~$2 &~10	Logical XOR reg, constant
shift left logical	sll $1,$2,10	$1 = $2 << 10	Shift left by constant
shift right logical	srl $1,$2,10	$1 = $2 >> 10	Shift right by constant
shift right arithm.	sra $1,$2,10	$1 = $2 >> 10	Shift right (sign extend)
shift left logical	sllv $1,$2,$3	$1 = $2 << $3	Shift left by variable
shift right logical	srlv $1,$2, $3	$1 = $2 >> $3	Shift right by variable
shift right arithm.	srav $1,$2, $3	$1 = $2 >> $3	Shift right arith. by variable

Q: Can some multiply by 2^i ? Divide by 2^i ? Invert?

MIPS data transfer instructions

Instruction	*Comment*
sw 500($4), $3	Store word
sh 502($2), $3	Store half
sb 41($3), $2	Store byte
lw $1, 30($2)	Load word
lh $1, 40($3)	Load halfword
lhu $1, 40($3)	Load halfword unsigned
lb $1, 40($3)	Load byte
lbu $1, 40($3)	Load byte unsigned
lui $1, 40	Load Upper Immediate (16 bits shifted left by 16)

Q: Why need lui?



When does MIPS sign extend?

° When value is sign extended, copy upper bit to full value:

Examples of sign extending 8 bits to 16 bits:

0**0001010** ⇒ **00000000 00001010**
1**0001100** ⇒ **11111111 10001100**

° When is an immediate operand sign extended?

- Arithmetic instructions (add, sub, etc.) always sign extend immediates *even for the unsigned versions of the instructions!*
- Logical instructions do not sign extend immediates (They are zero extended)
- Load/Store address computations always sign extend immediates

° Multiply/Divide have no immediate operands however:

- "unsigned" ⇒ treat operands as unsigned

° The data loaded by the instructions lb and lh are extended as follows ("unsigned" ⇒ don't extend):

- lbu, lhu are zero extended
- lb, lh are sign extended

Q: Then what is does add unsigned (addu) mean since not immediate?

MIPS Compare and Branch

- Compare and Branch
  - BEQ rs, rt, offset if R[rs] == R[rt] then PC-relative branch
  - BNE rs, rt, offset <>
- Compare to zero and Branch
  - BLEZ rs, offset if R[rs] <= 0 then PC-relative branch
  - BGTZ rs, offset >
  - BLT <
  - BGEZ >=
  - BLTZAL rs, offset if R[rs] < 0 then branch and link (into R 31)
  - BGEZAL >=!
- Remaining set of compare and branch ops take two instructions
- Almost all comparisons are against zero!

MIPS jump, branch, compare instructions

Instruction	*Example*	*Meaning*
branch on equal	beq $1,$2,100	if ($1 == $2) go to PC+4+100
	Equal test; PC relative branch	
branch on not eq.	bne $1,$2,100	if ($1!= $2) go to PC+4+100
	Not equal test; PC relative	
set on less than	slt $1,$2,$3	if ($2 < $3) $1=1; else $1=0
	Compare less than; 2's comp.	
set less than imm.	slti $1,$2,100	if ($2 < 100) $1=1; else $1=0
	Compare < constant; 2's comp.	
set less than uns.	sltu $1,$2,$3	if ($2 < $3) $1=1; else $1=0
	Compare less than; natural numbers	
set l. t. imm. uns.	sltiu $1,$2,100	if ($2 < 100) $1=1; else $1=0
	Compare < constant; natural numbers	
jump	j 10000	go to 10000
	Jump to target address	
jump register	jr $31	go to $31
	For switch, procedure return	
jump and link	jal 10000	$31 = PC + 4; go to 10000
	For procedure call	

Signed vs. Unsigned Comparison

$1= 0…00 0000 0000 0000 0001 two

$2= 0…00 0000 0000 0000 0010 two

$3= 1…11 1111 1111 1111 1111 two

° After executing these instructions:

```
slt  $4,$2,$1 ;
slt  $5,$3,$1 ;
sltu $6,$2,$1 ;
sltu $7,$3,$1 ;
```

if ($2 < $1) $4=1; else $4=0

if ($3 < $1) $5=1; else $5=0

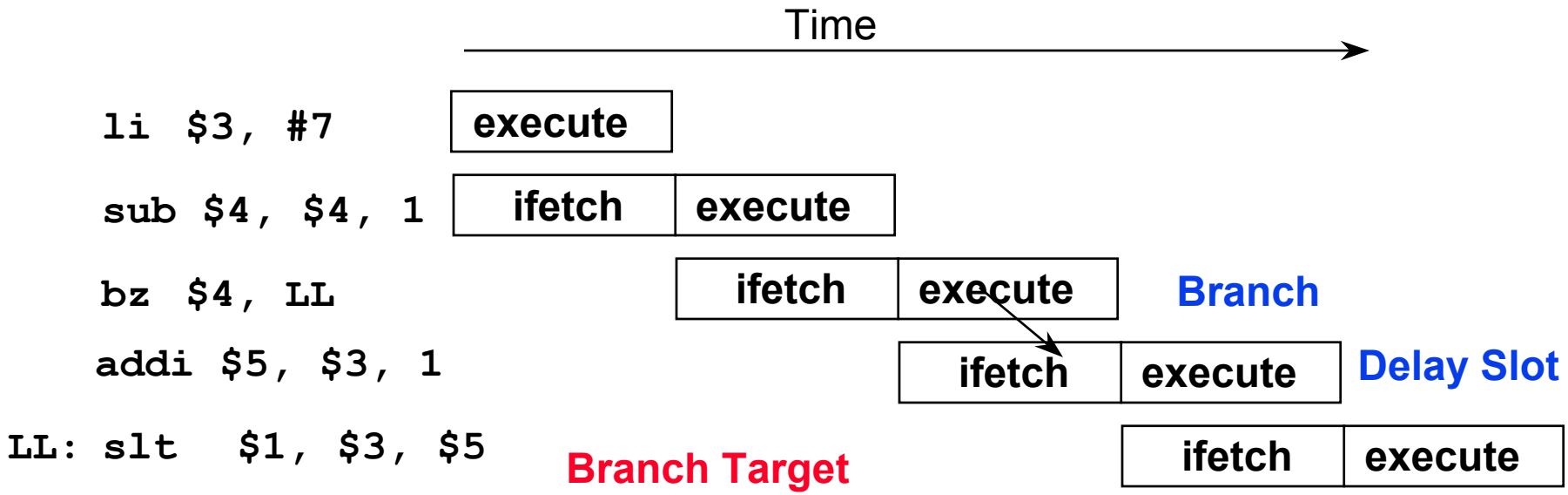
if ($2 < $1) $6=1; else $6=0

if ($3 < $1) $7=1; else $7=0

° What are values of registers $4 - $7? Why?

$4 = ; $5 = ; $6 = ; $7 = ;

Branch & Pipelines



By the end of Branch instruction, the CPU knows whether or not the branch will take place.

However, it will have fetched the next instruction by then, regardless of whether or not a branch will be taken.

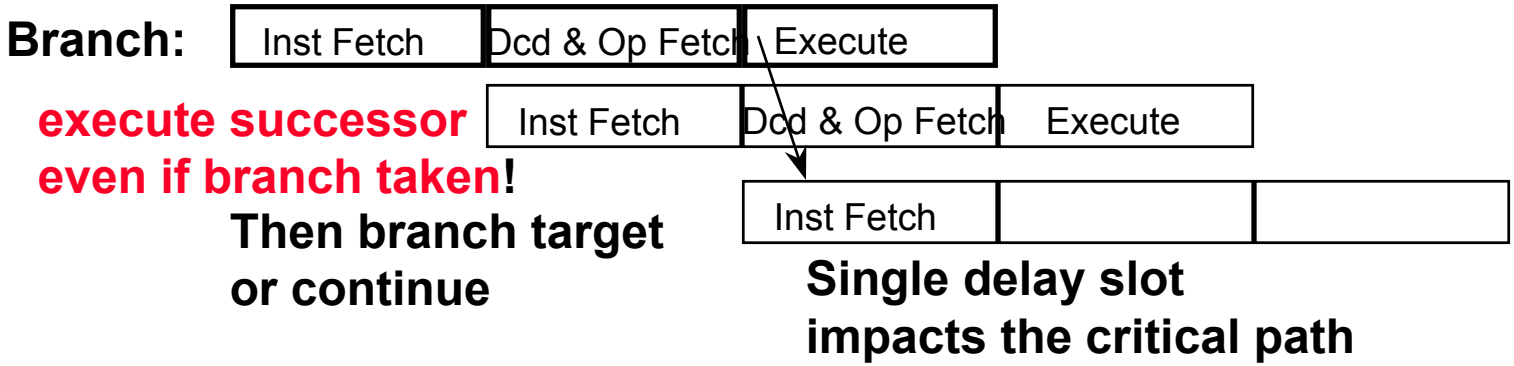
Why not execute it?

Delayed Branches

```
      li   $3, #7
      sub  $4, $4, 1
      bz   $4, LL
      addi $5, $3, 1     ⇐ Delay Slot Instruction
      subi $6, $6, 2
LL:   slt  $1, $3, $5
```

- In the "Raw" MIPS, the instruction after the branch is executed even when the branch is taken
  - This is hidden by the assembler for the MIPS "virtual machine"
  - allows the compiler to better utilize the instruction pipeline (???)
- Jump and link (jal inst):
  - Put the return addr. Into link register ($31):
    - PC+4 (logical architecture)
    - PC+8 physical ("Raw") architecture ⇒ delay slot executed
  - Then jump to destination address

Filling Delayed Branches



- Compiler can fill a single delay slot with a useful instruction 50% of the time.
  - try to move down from above jump
  - move up from target, if safe

```
        add $3, $1, $2
        sub $4, $4, 1
        bz  $4, LL
        NOP
        ...
LL:     add rd, ...
```

Is this violating the ISA abstraction?

Miscellaneous MIPS I instructions

- **break** — A breakpoint trap occurs, transfers control to exception handler
- **syscall** — A system trap occurs, transfers control to exception handler
- **coprocessor instrs.** — Support for floating point
- **TLB instructions** — Support for virtual memory: discussed later
- **restore from exception kernel/user** — Restores previous interrupt mask & mode bits into status register
- **load word left/right** — Supports misaligned word loads
- **store word left/right** — Supports misaligned word stores

MIPS assembler register convention

Name	Number	Usage	Preserved on call?
$zero	0	the value 0	n/a
$v0-$v1	2-3	return values	no
$a0-$a3	4-7	arguments	no
$t0-$t7	8-15	temporaries	no
$s0-$s7	16-23	saved	yes
$t18-$t19	24-25	temporaries	no
$sp	29	stack pointer	yes
$ra	31	return address	yes

- "caller saved"
- "callee saved"

Summary: Salient features of MIPS I

- **32-bit fixed format inst** (3 formats)
- **32 32-bit GPR** (R0 contains zero) and 32 FP registers (and HI LO)
  - partitioned by software convention
- **3-address, reg-reg arithmetic instr.**
- **Single address mode for load/store:** base+displacement
  - no indirection, scaled
- **16-bit immediate plus LUI**
- **Simple branch conditions**
  - compare against zero or two registers for =,≠
  - no integer condition codes
- **Delayed branch**
  - execute instruction after a branch (or jump) even if the branch is taken

  (Compiler can fill a delayed branch with useful work about 50% of the time)

Peer Instruction: $s3=i, $s4=j, $s5=@A

```
Loop: addiu  $s4,$s4,1      # j = j + 1
      sll    $t1,$s3,2      # $t1 = 4 * i
      addu   $t1,$t1,$s5    # $t1 = @ A[i]
      lw     $t0,0($t1)     # $t0 = A[i]
      addiu  $s3,$s3,1      # i = i + 1
      slti   $t1,$t0,10     # $t1 = $t0 < 10
      beq    $t1,$0, Loop   # goto Loop
      slti   $t1,$t0, 0     # $t1 = $t0 <  0
      bne    $t1,$0, Loop   # goto Loop
```

```
do j = j + 1
  while (_____);
```

What C code properly fills in the blank in loop on right?

1: `A[i++] >= 10`

2: `A[i++] >= 10 |  A[i]   < 0`

3: `A[i]   >= 10 || A[i++] < 0`

4: `A[i++] >= 10 || A[i]   < 0`

5: `A[i]   >= 10 && A[i++] < 0`

6 None of the above

Peer Instruction: $s3=i, $s4=j, $s5=@A

```
Loop: addiu  $s4,$s4,1      # j = j + 1
      sll    $t1,$s3,2      # $t1 = 4 * i
      addu   $t1,$t1,$s5    # $t1 = @ A[i]
      lw     $t0,0($t1)     # $t0 = A[i]
      addiu  $s3,$s3,1      # i = i + 1
      slti   $t1,$t0,10     # $t1 = $t0 < 10
      beq    $t1,$0, Loop   # goto Loop if $t1 == 0 ($t0 >= 10)
      slti   $t1,$t0, 0     # $t1 = $t0 <  0
      bne    $t1,$0, Loop   # goto Loop if $t1 != 0 ($t0 < 0)
```

```
do j = j + 1
  while (_____);
```

What C code properly fills in the blank in loop on right?

```
1: A[i++] >= 10
2: A[i++] >= 10 |  A[i]   < 0
3: A[i]   >= 10 || A[i++] < 0
4: A[i++] >= 10 || A[i]   < 0
5: A[i]   >= 10 && A[i++] < 0
6: None of the above
```

Instruction Formats

- **I-format**: used for instructions with immediates, `lw` and `sw` (since the offset counts as an immediate), and the branches (`beq` and `bne`),
  - (but not the shift instructions; later)
- **J-format**: used for `j` and `jal`
- **R-format**: used for all other instructions
- It will soon become clear why the instructions have been partitioned in this way.

R-Format Instructions (1/4)

- Define "fields" of the following number of bits each: 6 + 5 + 5 + 5 + 5 + 6 = 32

6	5	5	5	5	6

- For simplicity, each field has a name:

opcode	rs	rt	rd	shamt	funct

R-Format Instructions (2/4)

- What do these field integer values tell us?
  - `opcode`: partially specifies what instruction it is
    - Note: This number is equal to 0 for all R-Format instructions.
  - `funct`: combined with `opcode`, this number exactly specifies the instruction

R-Format Instructions (3/4)

- More fields:
  - `rs` (Source Register): *generally* used to specify register containing first operand
  - `rt` (Target Register): *generally* used to specify register containing second operand (note that name is misleading)
  - `rd` (Destination Register): *generally* used to specify register which will receive result of computation

R-Format Instructions (4/4)

- Final field:
  - `shamt`: This field contains the amount a shift instruction will shift by. Shifting a 32-bit word by more than 31 is useless, so this field is only 5 bits (so it can represent the numbers 0-31).
  - This field is set to 0 in all but the shift instructions.

R-Format Example

- **MIPS Instruction:**

```
add    $8,$9,$10
```

Decimal number per field representation:

0	9	10	8	0	32

Binary number per field representation:

000000	01001	01010	01000	00000	100000

hex

hex representation: $012A\ 4020_{hex}$

decimal representation: $19{,}546{,}144_{ten}$

I-Format Example (1/2)

- MIPS Instruction:

```
addi    $21,$22,-50
```

`opcode` = 8 (look up in table in book)

`rs` = 22 (register containing operand)

`rt` = 21 (target register)

`immediate` = -50 (by default, this is decimal)

I-Format Example (2/2)

- **MIPS Instruction:**

```
addi    $21,$22,-50
```

Decimal/field representation:

8	22	21	-50

Binary/field representation:

001000	10110	10101	1111111111001110

hexadecimal representation: 22D5 FFCE$_{hex}$

decimal representation: 584,449,998$_{ten}$

J-Format Instructions (1/2)

- Define "fields" of the following number of bits each:

6 bits	26 bits

- As usual, each field has a name:

opcode	target address

- Key Concepts
  - Keep `opcode` field identical to R-format and I-format for consistency.
  - Combine all other fields to make room for large target address.

J-Format Instructions (2/2)

- Summary:
  - New PC = { PC[31..28], target address, 00 }
- Understand where each part came from!
- Note: { , , } means concatenation
  { 4 bits , 26 bits , 2 bits } = 32 bit address
  - { 1010, 11111111111111111111111111, 00 }
    = 10101111111111111111111111111100
  - Note: Book uses ||, Verilog uses { , , }
  - We will use Verilog in this class

Peer Instruction

Which instruction has same representation as 35_{ten}?

A. add $0, $0, $0

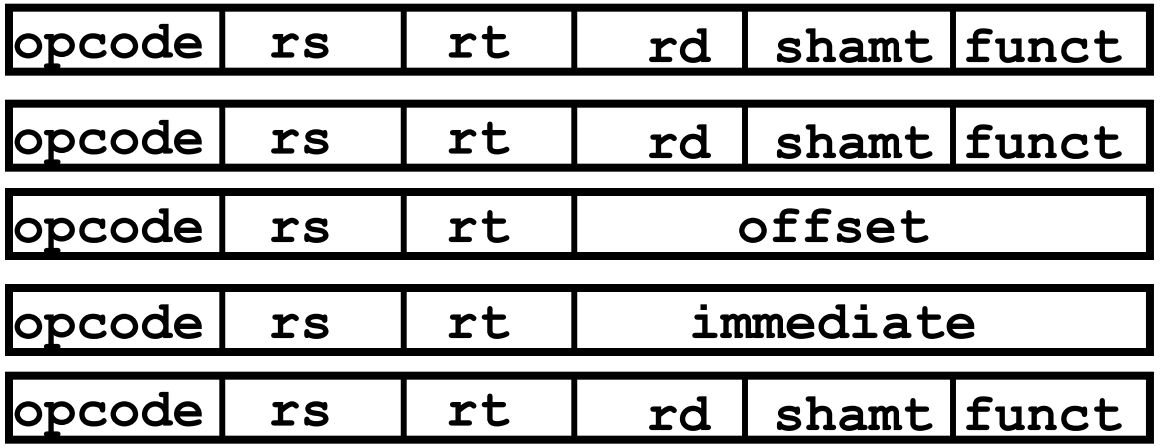
B. subu $s0,$s0,$s0

C. lw $0, 0($0)

D. addi $0, $0, 35

E. subu $0, $0, $0

F. Trick question! Instructions are not numbers



Registers numbers and names:
0: $0, 8: $t0, 9:$t1, ..15: $t7, 16: $s0, 17: $s1, .. 23: $s7

Opcodes and function fields (if necessary)

`add`: opcode = 0, funct = 32

`subu`: opcode = 0, funct = 35

`addi`: opcode = 8

`lw`: opcode = 35

Peer Instruction

Which instruction bit pattern = number 35?

A. add $0, $0, $0

0	0	0	0	0	32

B. subu $s0,$s0,$s0

0	16	16	16	0	35

C. lw $0, 0($0)

35	0	0			0

D. addi $0, $0, 35

8	0	0			35

E. subu $0, $0, $0

0	0	0	0	0	35

F. Trick question!
Instructions != numbers

Registers numbers and names:
0: $0, 8: $t0, 9:$t1, …,16: $s0, 17: $s1, …,

Opcodes and function fields

`add`: opcode = 0, function field = 32

`subu`: opcode = 0, function field = 35

`addi`: opcode = 8

`lw`: opcode = 35

And in conclusion...

- Continued rapid improvement in Computing
  - 2X every 1.5 years in processor speed; every 2.0 years in memory size; every 1.0 year in disk capacity; Moore's Law enables processor, memory (2X transistors/chip/ ~1.5 yrs)
- 5 classic components of all computers

Control Datapath Memory Input Output

Processor