

thelede.blogs.nytimes.com/2010/02/19/school-accused-ofusing-webcam-to-photograph-student-at-home/

Review

- Simplifying MIPS: Define instructions to be same size as data word (one word) so that they can use the same memory (compiler can use lw and sw).
- Computer actually stores programs as a series of these 32-bit numbers.
- MIPS Machine Language Instruction: 32 bits representing a single instruction

R	opcode	rs	rt	rd	shamt	funct
1	opcode	rs	rt	iı	mmedia	te
Cal	CS61C L14 : MIPS Instructio	n Representation II (2)				Garcia, Spring 2010 © UCB

I-Format Problems (0/3)

- Problem 0: Unsigned # sign-extended?
 - addiu, sltiu, sign-extends immediates to 32 bits. Thus, # is a "signed" integer.
- Rationale

(al

Cal

- addiu so that can add w/out overflow
- See K&R pp. 230, 305
- sltiu suffers so that we can have easy HW
 Does this mean we'll get wrong answers?
 - Nope, it means assembler has to handle any unsigned immediate $2^{15} \le n < 2^{16}$ (I.e., with a 1 in the 15th bit and 0s in the upper 2 bytes) as it does for numbers that are

too large. \Rightarrow

I-Format Problem (1/3)

Problem:

Cal

- Chances are that addi, lw, sw and slti will use immediates small enough to fit in the immediate field.
-but what if it's too big?
- We need a way to deal with a 32-bit immediate in any I-format instruction.

I-Format Problem (2/3)

- Solution to Problem:
 - Handle it in software + new instruction
 - Don't change the current instructions: instead, add a new instruction to help out
- New instruction:

lui register, immediate

- stands for Load Upper Immediate
- takes 16-bit immediate and puts these bits in the upper half (high order half) of the register

arcia, Spring 2010 © UCB

sets lower half to 0s

I-Format Problems (3/3)

- Solution to Problem (continued):
 - So how does lui help us?
- Example:
 - addiu \$t0,\$t0, 0xABABCDCD
- ...becomes
 - lui \$at OxABAB
 - ori \$at, \$at, 0xCDCD
 - addu \$t0,\$t0,\$at
- Now each I-format instruction has only a 16-bit immediate.
- Wouldn't it be nice if the assembler would this for us automatically? (later)

Garcia, Spring 2010 © UCB

CIGCLUA: MIPS Instruction Representation 11 (6)

			-Relat	ive Addressing (1/5)
•	Use I-F	ormat		
0	pcode	rs	rt	immediate
-	opcod	e speci	fi es beg	versus bne
-	rs and	lrt spe	ecify reg	isters to compare
	What co	an imm	ediate s	pecify?
	□ imme	diate i	s only 16	bits
	instru	ction beir	ounter) h ng execu io memo	
•	So im branc		e cannoi	specify entire address to
al	C\$61C L14 : MIPS Inc	fruction Representation	m 11 (7)	Garcia, Spring 2010 © UCB

Branches: PC-Relative Addressing (2/5)

- How do we typically use branches?
 - Answer: if-else, while, for

Cal CARE LTA : MIPS instruction for

Cal CSGIC L14 : MIPS instruction Report

- Loops are generally small: usually up to 50 instructions
- Function calls and unconditional jumps are done using jump instructions (j and jal), not the branches.
- Conclusion: may want to branch to anywhere in memory, but a branch often changes PC by a small amount

Garcia, Spring 2010 @ UCB

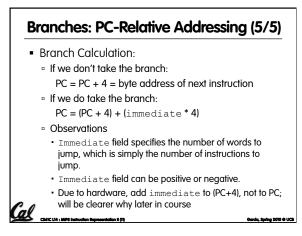
Branches: PC-Relative Addressing (3/5)

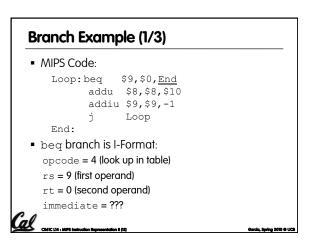
- Solution to branches in a 32-bit instruction: PC-Relative Addressing
- Let the 16-bit immediate field be a signed two's complement integer to be *added* to the PC if we take the branch.
- Now we can branch ± 2¹⁵ bytes from the PC, which should be enough to cover almost any loop.
- Any ideas to further optimize this?

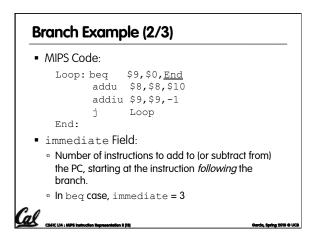
Call CASTC 114 : MIPS instruction Representation II (9)

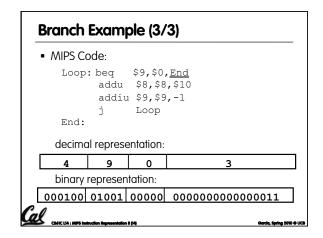
Branches: PC-Relative Addressing (4/5)

- Note: Instructions are words, so they're word aligned (byte address is always a multiple of 4, which means it ends with 00 in binary).
 - So the number of bytes to add to the PC will always be a multiple of 4.
 - So specify the immediate in words.
- Now, we can branch $\pm 2^{15}$ words from the PC (or $\pm 2^{17}$ bytes), so we can handle loops 4 times as large.









Questions on PC-addressing • Does the value in branch field change if we move the code? • What do we do if destination is $> 2^{15}$ instructions away from branch? Why do we need different addressing modes (different ways of forming a memory address)? Why not just one? Cal CHECHAINM

Week # Mon Wed Thu Lab Fri #6 MIPS Inst Format II Floating Pt I Floating Pt I Floating Pt II Floating Pt II #7 MIPS Inst Week Running Format III Running Program Running Program Running Program #8 SDS I SDS III (TA) SDS III (TA) SDS III (TA) SDS IIII (TA)	Upco	ming Cale	endar		
MIPS Inst Format II Floating Pt I Floating Pt I Floating Pt II #7 MIPS Inst Format III Running Program Running Program Running Program #8 SDS I SDS II SDS III Midterm Week Midterm I III SDS III	Week #	Mon	Wed	Thu Lab	Fri
Next week MIPS Inst Format III Running Program Running Program Running Program Running Program #8 SDS I SDS II SDS III SDS III SDS III Midterm week Midterm In they give us SDS III SDS III SDS III SDS III	This				Pt II Č
SDS I SDS II SDS SDS III Midterm (TA) (TA) (TA) Tonight (if they give us	Next				
	Midterm	Midterm Tonight (if they give us		SDS	

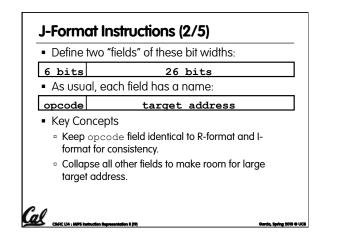
Project 2 has been converted to a homework thanks to the feedback of past classes Cal CANCUAR

Administrivia

HW4 due Wed

J-Format Instructions (1/5) For branches, we assumed that we won't want to branch too far, so we can specify change in PC. • For general jumps (j and jal), we may jump to anywhere in memory. Ideally, we could specify a 32-bit memory address to jump to. • Unfortunately, we can't fit both a 6-bit opcode and a 32-bit address into a single 32-bit word, so we compromise. CASIC L14 : MPS Instruction Representation II (14)

Garcia, Spring 2010 © UCB



J-Format Instructions (3/5)

- For now, we can specify 26 bits of the 32-bit bit address.
- Optimization:
 - Note that, just like with branches, jumps will only jump to word aligned addresses, so last two bits are always 00 (in binary).
 - $\circ\,$ So let's just take this for granted and not even specify them.

J-Format Instructions (4/5)

- Now specify 28 bits of a 32-bit address
- Where do we get the other 4 bits?

Cal

- $^\circ\,$ By definition, take the 4 highest order bits from the PC.
- Technically, this means that we cannot jump to anywhere in memory, but it's adequate 99.9999...
 % of the time, since programs aren't that long
 only if straddle a 256 MB boundary
- If we absolutely need to specify a 32-bit address, we can always put it in a register and use the jr instruction.

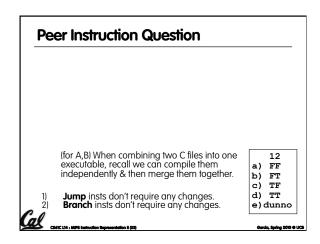


• Summary:

Cal Cal

Cal CARLES

- New PC = { PC[31..28], target address, 00 }
- Understand where each part came from!
- Note: { , , } means concatenation
- $\{4 \text{ bits}, 26 \text{ bits}, 2 \text{ bits}\} = 32 \text{ bit address}$
- { 1010, 11111111111111111111111, 00 } =
 101011111111111111111111100
- Note: Book uses II



opcode rs rt rd shamt funct opcode rs rt immediate opcode target address Branches use PC-relative addressing, Jumps use absolute addressing. Disassembly is simple and starts by decoding		 MIPS Mo 32 bits r 		5 5			
 opcode target address Branches use PC-relative addressing, Jumps use absolute addressing. Disassembly is simple and starts by decoding 	2	opcode	rs	rt	rd	shamt	funct
 Branches use PC-relative addressing, Jumps use absolute addressing. Disassembly is simple and starts by decoding 		opcode	rs rt immediate				
use absolute addressing.Disassembly is simple and starts by decoding		opcode	opcode target address				
opcode field. (more in a week)		Branche				ssing, Ju	mps