

Berkeley's smart dust. Be very afraid. www.nytimes.com/2005/02/16/technology/16robots.html

Overview – Instruction Representation

- Big idea: stored program · consequences of stored program
- Instructions as numbers
- Instruction encoding

CSEIC L13 Inte

CS61C L13

- MIPS instruction format for Add instructions
- MIPS instruction format for Immediate, Data transfer instructions

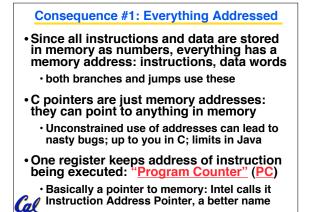
Garcia © UCB

Big Idea: Stored-Program Concept

- Computers built on 2 key principles:
 - 1) Instructions are represented as numbers.
 - 2) Therefore, entire programs can be stored in memory to be read or written just like numbers (data).
- Simplifies SW/HW of computer systems:
 - · Memory technology for data also used for programs

CS61C L13 Introduction to MIPS: Instru

al S61C | 13 |



Consequence #2: Binary Compatibility Programs are distributed in binary form · Programs bound to specific instruction set Different version for Macintoshes and PCs New machines want to run old programs ("binaries") as well as programs compiled to new instructions Leads to instruction set evolving over time Selection of Intel 8086 in 1981 for 1st IBM PC is major reason latest PCs still use 80x86 instruction set (Pentium 4); could

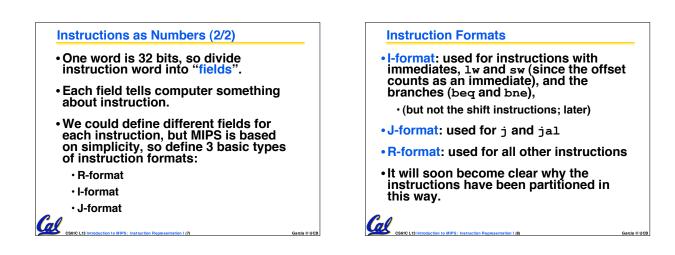
still run program from 1981 PC today

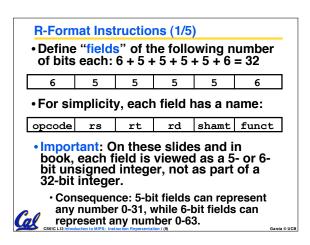
Instructions as Numbers (1/2)

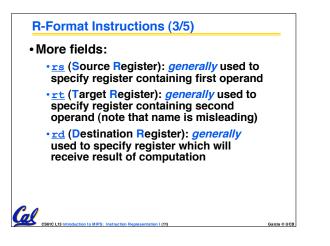
- Currently all data we work with is in words (32-bit blocks):
 - · Each register is a word.

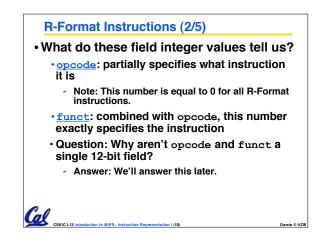
to MIPS: In

- •1w and sw both access memory one word at a time.
- So how do we represent instructions?
 - Remember: Computer only understands 1s and 0s, so "add \$t0,\$0,\$0" is meaningless.
- MIPS wants simplicity: since data is in words, make instructions be words too





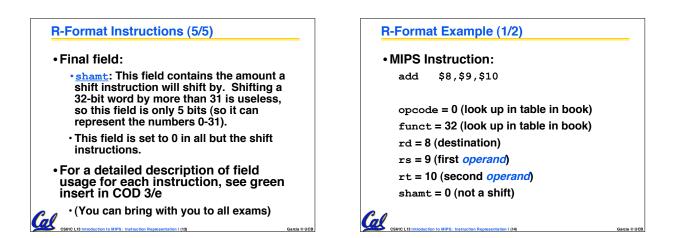


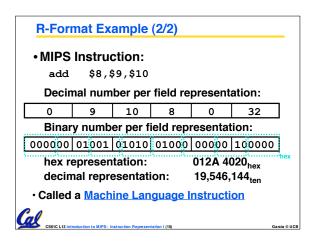


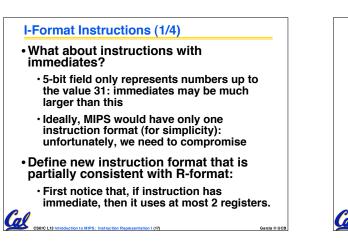
R-Format Instructions (4/5) Notes about register fields: Each register field is exactly 5 bits, which means that it can specify any unsigned integer in the range 0-31. Each of these fields specifies one of the 32 registers by number. The word "generally" was used because there are exceptions that we'll see later. E.g., mult and div have nothing important in the rd field since the dest registers are hi and lo mfhi and mflo have nothing important in the rs and rt fields since the source is

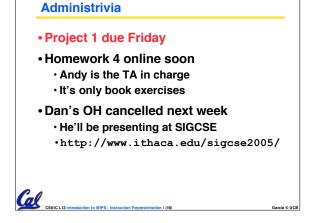


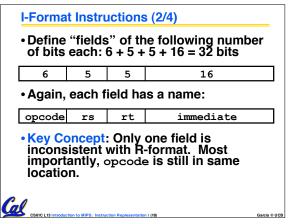
determined by the instruction (p. 264 P&H)











I-Format Instructions (3/4)

What do these fields mean?

I-Format Example (1/2)

- · opcode: same as before except that, since there's no funct field, opcode uniquely specifies an instruction in I-format
- · This also answers question of why R-format has two 6-bit fields to identify instruction instead of a single 12-bit field: in order to be consistent with other formats.
- rs: specifies the only register operand (if there is one)
- •rt: specifies register which will receive result of computation (this is why it's called the target register "rt")

CSSIC L13 Introduction to MIPS: Instruction Representation | (19)

I-Format Instructions (4/4)

• The Immediate Field:

Garcia © UCB

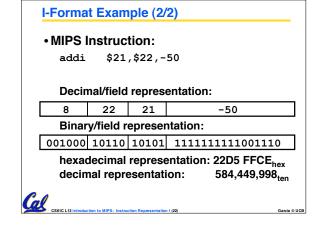
Garcia © UCB

- •addi, slti, sltiu, the immediate is sign-extended to 32 bits. Thus, it's treated as a signed integer.
- 16 bits → can be used to represent immediate up to 216 different values
- This is large enough to handle the offset in a typical 1w or sw, plus a vast majority of values that will be used in the slti instruction.
- · We'll see what to do when the number is too big in our next lecture...

Garcia © UCB

luction to MIPS: Instruction Representation I (20)

• 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) CSSIC L13 Introduction to MIPS: Instruction Representation I (21)



Peer Instruction									
Which instruction has same representation as 35 _{ten} ?									
1. add \$0, \$0, \$0	opcode	rs	rt	rd	shamt	funct			
2. subu \$s0,\$s0,\$s0	opcode	rs	rt	rd	shamt	funct			
3. lw \$0, 0(\$0)	opcode	rs	rt	offset					
4. addi \$0, \$0, 35	opcode	rs	rt	immediate					
5. subu \$0, \$0, \$0	opcode	rs	rt	rd	shamt	funct			
6. 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									
1w: opcode =		tation I (23)				Garcia © UC			

 In conclusion Simplifying MIPS: Define instructions to be same size as data word (one word) so that they can use the same memory (compiler can use 1w 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				
I	opcode	rs	rt	immediate						
Ca	CS61C L13 Introducti	on to MIPS: Instruct	tion Representation	1 (25)		Garcia © UC				