

# Instruction Representation

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# Review (1/2)

- Logical and Shift Instructions

- Operate on bits individually, unlike arithmetic, which operate on entire word
- Use to isolate fields, either by masking or by shifting back and forth
- Shift left logical (`sll`) multiplies by powers of 2
- Shift right arithmetic (`sra`) divides by powers of 2  
close but strange rounding for negative numbers  
(e.g., `-5 sra 2 bits = -2` while  $-5 / 2^2 = -5 / 4 = -1$ )

- New Instructions:

**`and andi or ori nor sll srl sra`**

# Review (2/2)

- MIPS Signed versus Unsigned is an "overloaded" term

- Do/Don't sign extend (lb, lbu)
- Don't overflow (addu, addiu, subu)
- Compute the correct answer (multu, divu)
- Do signed/unsigned compare (slt, slti/sltu, sltiu)

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# MULT vs MULTU

- In 2s complement, addition and subtraction are the same, as is the case in the low-half of a multiply.  
A full multiply, however, is not!

- In 32-bit twos-complement, -1 has the same representation as the unsigned quantity  $2^{32} - 1$ . However:

$$(-1)(-1) = +1$$

$$(2^{32} - 1)(2^{32} - 1) = 2^{64} - 2^{33} + 1$$

# Overview

- Big idea: stored program
  - consequences of stored program
- MIPS instruction format for Add instructions
- MIPS instruction format for Immediate, Data transfer instructions

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# 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

# Consequence #1: Everything Addressed

- Since all instructions and data are stored in memory as numbers, everything has a memory address
  - Instruction words and data words
  - Both branches and jumps use these instruction addresses
- C pointers are just memory addresses: can point to anything in memory
  - Unconstrained use of addresses can lead to nasty bugs!
  - Up to you in MIPS; up to you in C; limits in Java
- One register keeps address of instruction being executed:  
“Program Counter” (PC)
  - Just a pointer to memory: Intel calls it Instruction Address Pointer, a better name

# Consequence #2: Binary Compatibility

- Programs are distributed in binary form
  - Programs bound to specific instruction set
  - Different version for Macintosh and IBM PC
- New machines want to run old programs (“binaries”) as well as programs compiled to new instructions
- Leads to instruction set evolving over time
- Intel 8086 was selected in 1981 for 1st IBM PC
  - Latest PCs still use 80x86 instruction set...
  - Can (more or less) still run program from 1981 PC today!

Assignment Project Exam Help

<https://powcoder.com>


Add WeChat powcoder



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



0x28800001  
0x1100fffe  
0x20020001  
0x23bd0008  
0x03e00008

# Instructions as Numbers (1/2)

- All data we work with is in words (32-bit blocks):
  - Each register is a word
  - **lw** 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, let the instructions be words too

# Instructions as Numbers (2/2)

- Divide the 32-bit instruction word into “fields”
- Each field tells something about the instruction
- We could define different fields for each instruction, but MIPS is based on simplicity, so define 3 basic types of instruction formats:
  - R-format
  - I-format
  - J-format (next lecture)

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# Instruction Formats

- I-format: used for instructions with *immediates*,
  - **lw** and **sw** (since the offset counts as an immediate),
  - **beq** and **bne** (branches use offsets as we will see later)
  - But not the shift instructions (more on this later)
- J-format: *jump* format used for **j** and **jal**
- R-format: used for all other instructions
  - R stands for *register* format
- It will soon become clear why the instructions have been partitioned in this way!

# R-Format Instructions (1/5)

6	5	5	5	5	6
opcode	rs	rt	rd	shamt	funct

- Assignment Project Exam Help  
<https://powcoder.com>  
Add WeChat powcoder
- Break 32 bit “instruction” word into fields
    - For simplicity, each field has a name
  - **Important:** On these slides and in book, each field is viewed as a 5 or 6 bit unsigned integer, not as part of a 32 bit integer
    - 5 bit fields can represent any number 0-31,
    - 6 bit fields can represent any number 0-63.

# R-Format Instructions (2/5)

6	5	5	5	5	6
<b>opcode</b>	<b>rs</b>	<b>rt</b>	<b>rd</b>	<b>shamt</b>	<b>funct</b>

Assignment Project Exam Help

- 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

- Question:
  - Why aren't **opcode** and **funct** a single 12-bit field?
  - Think about it... We'll see the answer this later.



# R-Format Instructions (3/5)

6	5	5	5	5	6
opcode	rs	rt	rd	shamt	funct

Assignment Project Exam Help

- 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 (4/5)

6	5	5	5	5	6
opcode	rs	rt	rd	shamt	funct

- Assignment Project Exam Help  
<https://powcoder.com>  
Add WeChat powcoder
- Notes about register fields
    - Each register field is exactly 5 bits
    - It can specify any unsigned integer in the range 0-31
    - It specifies one of the 32 registers by number
  - “**generally**” on previous slide because there are exceptions that we’ll discuss more later...

*mult and div have nothing important in the rd field since the dest registers are hi and lo*

*mflhi and mfllo have nothing important in the rs and rt fields since the source is determined by the instruction*



# R-Format Instructions (5/5)

6	5	5	5	5	6
opcode	rs	rt	rd	shamt	funct

## Assignment Project Exam Help

- One more field we have not yet discussed

shamt contains the amount a shift instruction will shift

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
- For a detailed description of field usage for each instruction, see green reference card in the textbook

The image shows two pages from a MIPS reference manual. The left page is titled 'MIPS Reference Data' and contains a table of MIPS instructions with their opcodes, formats, and descriptions. The right page is titled 'MIPS Reference Data Card (Green Card)' and contains a table of MIPS instructions with their opcodes, formats, and descriptions. The card is a condensed version of the reference data, designed for quick reference during exams or assignments.

# R-Format Example (1/2)

- MIPS Instruction:

**add      \$8 \$9 \$10**

**opcode** = 0 (look up in table)

**funct** = 32 (look up in table)

**rs** = 9 (first *operand*)

**rt** = 10 (second *operand*)


**rd** = 8 (destination)

**shamt** = 0 (not a shift)

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

MIPS Reference Data				①	
NAME, MNEMONIC	FOR- MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)		
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>		
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>		
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>		
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>		
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>		
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>		
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>		
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>		
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>		
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>		
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>		
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2) 24 <sub>hex</sub>		
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2) 25 <sub>hex</sub>		
Load Linked	ll	I R[rt] = M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>		
Load Upper Imm.	lui	I R[rt] = {imm, 16'b0}	f <sub>hex</sub>		
Load Word	lw	I R[rt] = M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>		
Nor	nor	R R[rd] = ~(R[rs]   R[rt])	0 / 27 <sub>hex</sub>		
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>		
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>		

# R-Format Example (2/2)

- MIPS Instruction:

**add     \$8 \$9 \$10**

Assignment Project Exam Help

Decimal/field representation:

0	9	10	8	0	32
---	---	----	---	---	----

<https://powcoder.com>

Binary/field representation:

000000	01001	01010	01000	00000	100000
--------	-------	-------	-------	-------	--------

Add WeChat powcoder

hex

hex representation:            012A4020<sub>hex</sub>

decimal representation:        19546144<sub>ten</sub>

- Called a Machine Language Instruction

# I-Format Instructions (1/5)

- What about instructions with immediates?
  - 5-bit field only represents numbers up to the value 31
  - Immediates may be much larger than this
  - Ideally, MIPS would have only one instruction format (for simplicity)
  - Unfortunately, we need to compromise
- Define new instruction format partially consistent with R-format
  - Note that if the instruction has an immediate, then it uses at most 2 registers

# I-Format Instructions (2/5)

6	5	5	16
opcode	rs	rt	immediate

Assignment Project Exam Help

<https://powcoder.com>

- Define “fields” of a fixed number of bits each  
 $6 + 5 + 5 + 16 = 32$  bits
- Again, each field has a name
- **Key Concept:** Only one field is inconsistent with R-format. Most importantly, **opcode** is still in same location.

# I-Format Instructions (3/5)

6	5	5	16
opcode	rs	rt	immediate

Assignment Project Exam Help

- What do these fields mean?
  - **opcode**: same as before, but with no **funct** field, **opcode** uniquely specifies an instruction in I-format
- ***This finally answers the 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

<https://powcoder.com>

Add WeChat powcoder

# I-Format Instructions (4/5)

6	5	5	16
opcode	rs	rt	immediate

Assignment Project Exam Help

- More fields:

rs specifies the only register operand (if there is one)

rt specifies the register which will receive result of the computation (this is why it's called the *target* register "rt")

<https://powcoder.com>  
Add WeChat powcoder

# I-Format Instructions (5/5)

6	5	5	16
opcode	rs	rt	immediate

Assignment Project Exam Help

- The Immediate Field:
  - **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  $2^{16}$  different values
  - This is large enough to handle the offset in a typical **lw** or **sw**, plus a vast majority of values that will be used in the **slti** instruction.

<https://powcoder.com>

Add WeChat powcoder



# I-Format Example (1/2)

- MIPS Instruction:

**addi      \$21 \$22 -50**

**opcode = 8**  
(look up in table)

**rs = 22**  
(register containing operand)


**rt = 21**  
(target register)

**immediate = -50**  
(by default, specified in decimal)

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

MIPS Reference Data				①	
NAME, MNEMONIC	FOR- MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)		
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>		
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>		
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>		
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>		
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>		
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>		
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>		
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>		
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>		
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>		
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>		
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2) 24 <sub>hex</sub>		
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2) 25 <sub>hex</sub>		
Load Linked	ll	I R[rt] = M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>		
Load Upper Imm.	lui	I R[rt] = {imm, 16'b0}	f <sub>hex</sub>		
Load Word	lw	I R[rt] = M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>		
Nor	nor	R R[rd] = ~(R[rs]   R[rt])	0 / 27 <sub>hex</sub>		
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>		
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>		

# 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: 22D5FFCE<sub>hex</sub>

decimal representation: 584449998<sub>ten</sub>

# I-Format Problem (1/3)

Problem:

- Chances are that `addi`, `lw`, `sw` and `slti` will often use immediates small enough to fit in the immediate field
- But what if the value is too big?
  - We need a way to deal with a 32-bit immediate in *any* I-format instruction!

# I-Format Problems (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

Assignment Project Exam Help

<https://powcoder.com>

- New instruction:

**lui      register immediate**

Add WeChat powcoder

- Stands for Load Upper Immediate
- Takes 16-bit immediate and puts these bits in the upper half (high order half) of the specified register
- Sets lower half word to zero

# I-Format Problems (3/3)

- Solution to Problem (continued):

- Example of how lui helps:

`addi $t0 $t0 0xABABCDCD`

becomes:

`lui $at 0xABAB`

`ori $at $at 0xCDCD`

`add $t0 $t0 $at`

- Now I-format instructions have only 16 bit immediates
  - Assembler can do this for us automatically!  
(more on pseudoinstructions next lecture)

# 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 **lw** and **sw**). **Assignment Project Exam Help**
- Machine Language Instruction: 32 bits representing a single instruction <https://powcoder.com>

**Add WeChat powcoder**

<b>R</b>	opcode	rs	rt	rd	shamt	funct
<b>I</b>	opcode	rs	rt	immediate		

- Remember: The computer actually stores programs as a series of these 32-bit numbers.

# Review and More Information

- TextBook

- 2.5 Representing Instructions in the computer
- 2.10 Addressing for 32-bit immediates
- 2.12 Translating and Starting a Program
  - Just the section on the **Assembler** with respect to pseudoinstructions (pg 124, 125, 5<sup>th</sup> edition)

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder