Instruct entation

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`)
  - Shift right arithmetic (   ) divides by
    close but strange rounding for negative
    (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, d
  - Do signed/unsigned compare (slt,slti/sltu,sltiu)

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-comple

ntation 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 form
- MIPS instruction form transfer instructions

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro

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 writte
     Assignment Project Exam Help
     .

- Simplifies SW/
  https://eduassistpro.github.io/
  ns:

  - Memory technology for data
    Add WeChat edu_assist_pro
    r 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 jump   esses
- C pointers are just mem   t to anything in memory
  - Unconstrained use of addresses can lead t
  - 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

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro

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 t https://eduassistpro.github.io/ as well as programs compiled to new instructi

- Leads to instruction set evolving ov

- 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

Add WeChat edu_assist_pro

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro

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 represe https://eduassistpro.github.io/
  - Remember: Computer only understan ,so
  - "`add $t0,$0,$0`" is meaningless
  - MIPS wants simplicity:
    - Since data is in words, let the instructions be words too

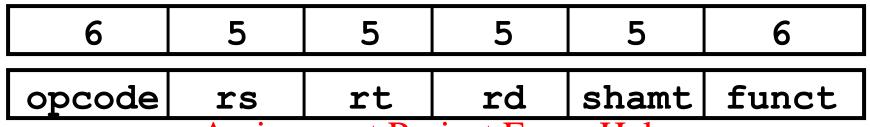# Instructions as Numbers (2/2)

- Divide the 32-bit instruction word into "fields"

- Each field tells something about the instruction

- We could define differ   ruction, but MIPS is based on simplicity, so   l instruction formats:
  - R-format
  - I-format
  - J-format (next lecture)

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 instru ~~ ~~)
- J-format: *jump* format used for `j` an
- R-format: used for all other instructi
  - R stands for *register* format
- It will soon become clear why the instructions have been partitioned in this way!

Assignment Project Exam Help
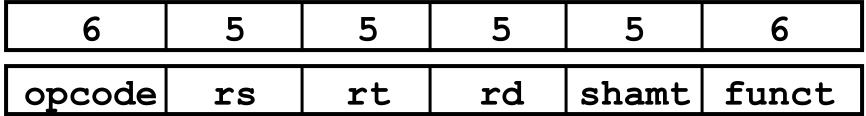
https://eduassistpro.github.io/

Add WeChat edu_assist_pro

R-Format Instructions (1/5)

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

Assignment Project Exam Help

- Break 32 bit "   fields
  - For simplicit https://eduassistpro.github.io/

- **Important**: On these slides   k, each field is
  viewed as a 5 or 6 bit unsig   Add WeChat edu_assist_pro   r, 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
opcode	**rs**	**rt**	**rd**	**shamt**	**funct**

Assignment Project Exam Help

- What do thes   ell us?

  opcode parti https://eduassistpro.github.io/ t io

  Note: This number is equal to 0   at instructions.
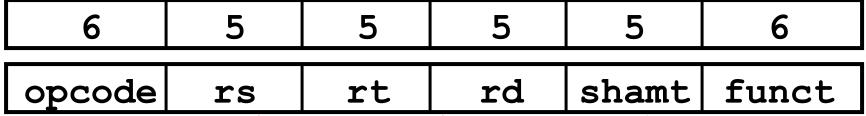
  funct combined with **opco** Add WeChat edu_assist_pro ber exactly specifies the instruction

- 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)

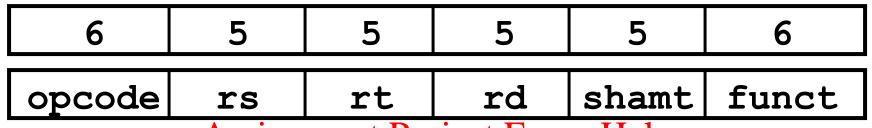| 6	5	5	5	5	6
opcode	**rs**	**rt**	**rd**	**shamt**	**funct**

- More fields

rs (Source R ster
containing **first operand**

rt (Target Register): *generall* 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)

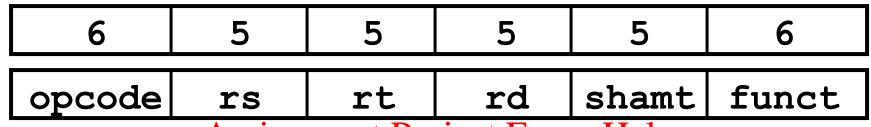| 6	5	5	5	5	6
opcode	**rs**	**rt**	**rd**	**shamt**	**funct**

Assignment Project Exam Help

- Notes about r

  - Each register https://eduassistpro.github.io/

  - It can specify any unsigned in   range 0-31

  - It specifies one of the 32 regi Add WeChat edu_assist_pro ber

- "**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

mfhi and mflo have nothing important in the rs and rt fields since the source is determined by the instruction
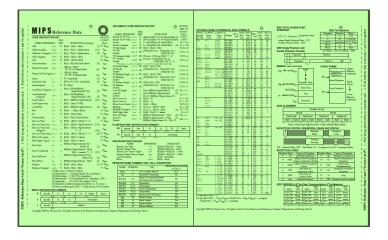
R-Format Instructions (5/5)

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

- One more fie   cussed

  shamt conta https://eduassistpro.github.io/ rction will shift

  Shifting a 32-bit word by more   less, so this field is
  only 5 bits (se Add WeChat edu_assist_pro

- 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

R-Format Example (1/2)

- MIPS Instruction:

    add $8 $9 $10

opcode = 0 (look up in tabl

funct = 32 (look up in table)

rs = 9 (first *operand*)

rt = 10 (second *operand*)

rd = 8 (destination)

shamt = 0 (not a shift)

R-Format Example (2/2)

- MIPS Instruction:

`add    $8 $9 $10`

Decimal/field repres

0	9			0	32

Binary/field represen

000000	01001	01010	01000	00000	100000

hex

hex representation: $012A4020_{hex}$

decimal representation: 19546144_{ten}

- Called a <u>Machine Language Instruction</u>

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 ha   rmat (for simplicity)
  - Unfortunately, we need to compromis

- Define new instruction format parti   ent with R-format
  - Note that if the instruction has an immediate, then it uses at most 2 registers

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro

I-Format Instructions (2/5)

6	5	5	16
opcode	**rs**	**rt**	**immediate**

- Define "fields   bits each
6 + 5 + 5 + 16

- Again, each field

- **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**

- What do thes

  - **opcode**: sa**func**t field, **opcode** uniquely specifies   n in I-format

- ***This finally answers the qu***   hy 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

I-Format Instructions (4/5)

6	5	5	16
opcode	**rs**	**rt**	**immediate**

Assignment Project Exam Help

- More fields:

  rs specifies t https://eduassistpro.github.io/ if there is one)

  rt specifies the register whic   e result of the
  computation (this is why it's   Add WeChat edu_assist_pro   *get* register "rt")

I-Format Instructions (5/5)

6	5	5	16
opcode	**rs**	**rt**	**immediate**

- The Immedia
  - **addi**, **slt** is sign-extended to 32 bits. Thus, it's treate   integer.
  - 16 bits ➔ can be used to re ediate 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.

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)

I-Format Example (2/2)

- MIPS Instruction:

addi $21 $22 -50

Decimal/fie

8	2		-50

Binary/field representati

001000	10110	10101	1111111111001110

hexadecimal representation: $22D5FFCE_{hex}$

decimal representation: 584449998_{ten}

I-Format Problem (1/3)

Problem:

- Chances are that `addi lw sw` and `slti` will often use immediates small enough to fit in t

- But what if the value i

  - We need a way to deal with that immediate in a*ny* 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 ins

- 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 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
    ori     $at $at 0xCDC
    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 Ins   enting a single
instruction   https://eduassistpro.github.io/

Add WeChat edu_assist_pro

R	opcode	rs	rt		hamt	funct
I	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 S

    - Just the section on the **Assembler** with re   oinstructions (pg 124, 125, 5th edition)

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro