
SI 232
SlideSet #2: Instructions
(Chapter 2)

1

Slide Credits

- Much material in some way derived from...
 - Official textbook (Copyright Morgan Kaufmann)
 - Prof. Margarget McMahon (USNA)
 - Major W. Clay James (USNA)

2

Chapter Goals

- **Teach a subset of MIPS assembly language**
- **Introduce the stored program concept**
- **Explain how MIPS instructions are represented in machine language**
- **Illustrate basic instruction set design principles**

3

Instructions:

- Language of the Machine
- More primitive than higher level languages
- Very restrictive
 - e.g., MIPS Arithmetic Instructions
- We'll be working with the MIPS instruction set architecture
 - similar to other architectures developed since the 1980's
 - used by NEC, Nintendo, Silicon Graphics, Sony

Design principles: to be found...
Design goals:

4

MIPS arithmetic

- All instructions have 3 operands
- Operand order is fixed

Example:

C code: $A = B + C$

MIPS code: `add $s0, $s1, $s2`

5

MIPS arithmetic

- Design Principle #1: simplicity favors regularity. Why?

- Of course this complicates some things...

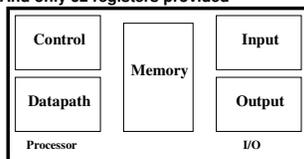
C code: $A = B + C + D;$
 $E = F - A;$

MIPS code: `add $t0, $s1, $s2`
 `add $s0, $t0, $s3`
 `sub $s4, $s5, $s0`

6

Registers vs. Memory

- Design Principle #2: smaller is faster. Why?
- Therefore, arithmetic instruction operands must be “registers”
 - And only 32 registers provided



- Compiler associates variables with registers
- What about programs with lots of variables?

7

Memory Organization

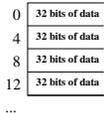
- Viewed as a large, single-dimension array, with an address.
- A memory address is an index into the array
- "Byte addressing" means that the index points to a byte of memory.

0	8 bits of data
1	8 bits of data
2	8 bits of data
3	8 bits of data
4	8 bits of data
5	8 bits of data
6	8 bits of data
...	

8

Memory Organization

- Bytes are nice, but most data items use larger "words"
- For MIPS, a word is 32 bits or 4 bytes.



Registers hold 32 bits of data

- 2^{32} bytes with byte addresses from 0 to $2^{32}-1$
- 2^{30} words with byte addresses 0, 4, 8, ... $2^{32}-4$
- Words are aligned
i.e., what are the least 2 significant bits of a word address?

9

Memory Instructions

- Load and store instructions
- Example:

C code: `A[8] = h + A[8];`

MIPS code: `lw $t0, 32($s3)`
`add $t0, $s2, $t0`
`sw $t0, 32($s3)`

- How about this?
`add $t0, 32($s3), $t0`

10

So far we've learned:

- MIPS
 - loading words but addressing bytes
 - arithmetic on registers only
- | <u>Instruction</u> | <u>Meaning</u> |
|-----------------------------------|--------------------------------------|
| <code>add \$s1, \$s2, \$s3</code> | <code>\$s1 = \$s2 + \$s3</code> |
| <code>sub \$s1, \$s2, \$s3</code> | <code>\$s1 = \$s2 - \$s3</code> |
| <code>lw \$s1, 100(\$s2)</code> | <code>\$s1 = Memory[\$s2+100]</code> |
| <code>sw \$s1, 100(\$s2)</code> | <code>Memory[\$s2+100] = \$s1</code> |

11

Exercise #1

- What is the MIPS assembly code for the following:
`g = g + h - i;`
Variables g, h, & i are assigned registers \$s1, \$s2, and \$s4

12

Exercise #2

- What is the MIPS assembly code for the following:
 `g = h + A[3];`
Variables `g`, `h`, & `i` are assigned registers `$s1`, `$s2`, and `$s4`
Array `A` base address is assigned register `$s3`

13

Exercise #3

- What is the MIPS assembly code for the following:
 `g = h + A[i];`
Variables `g`, `h`, & `i` are assigned registers `$s1`, `$s2`, and `$s4`
Array `A` base address is assigned register `$s3`

14

Extra space

Machine Language

- Instructions, like registers and words of data, are also 32 bits long
 - Example: `add $t0, $s1, $s2`
 - registers have numbers, `$t0=8`, `$s1=17`, `$s2=18`
- Instruction Format (r-type):

000000	10001	10010	01000	00000	100000
--------	-------	-------	-------	-------	--------

op	rs	rt	rd	shamt	funct
----	----	----	----	-------	-------

15

16

Machine Language

- Consider the load-word and store-word instructions,
 - What would the regularity principle have us do?
 - Principle #3: Make the common case fast
 - Principle #4: Good design demands a compromise
- Introduce a new type of instruction format
 - I-type for data transfer instructions
- Example: `lw $t0, 44($s2)`

35	18	8	44
op	rs	rt	16 bit number

- Where's the compromise?

17

Example Part 1

- What is the machine code for the following:

$$A[300] = h + A[300];$$

Variable h is assigned register \$s2
 Array A base address is assigned register \$t1

- Do the assembly code first, then machine language instructions, and then machine code

18

Example Part 2

- What is the machine code for the following: $A[300] = h + A[300];$
 - Variable h is assigned register \$s2 & Array A base address is assigned register \$t1
- First part of answer:


```
lw $t0, 1200($t1) # Temporary reg $t0 gets A[300]
add $t0, $s2, $t0 # Temporary reg $t0 gets h + A[300]
sw $t0, 1200($t1) # Stores h + A[300] back into A[300]
```
- Second part of answer:

op	rs	rt	rd	shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

19

Example Part 3

- What is the machine code for the following: $A[300] = h + A[300];$
 - Variable h is assigned register \$s2 & Array A base address is assigned register \$t1
- First part of answer:

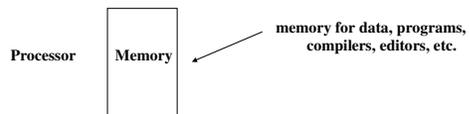

```
lw $t0, 1200($t1) # Temporary reg $t0 gets A[300]
add $t0, $s2, $t0 # Temporary reg $t0 gets h + A[300]
sw $t0, 1200($t1) # Stores h + A[300] back into A[300]
```
- Second part of answer:

op	rs	rt	rd	shamt	funct
100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	010000
101011	01001	01000	0000 0100 1011 0000		

20

Stored Program Concept

- Instructions are composed of bits / bytes / words
- Programs are stored in memory
 - to be read or written just like data



- Fetch & Execute Cycle
 - Instructions are fetched and put into a special register
 - Bits in the register "control" the subsequent actions
 - Fetch the "next" instruction and continue