

---

## IC220 SlideSet #2: Instructions (Chapter 2)

Credits: Much course material in some way derived from...  
Official textbook (Copyright Morgan Kaufmann)  
Prof. Margarget McMahon (USNA)

1

### 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 Tivo, Cisco routers, Nintendo 64, Sony PlayStation...

*Design principles: to be found...*

*Design goals:*

3

### 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

2

### MIPS arithmetic

---

- All instructions have 3 operands
- Operand order is fixed

Example:

C code:         $A = B + C$

MIPS code:    `add $s0, $s1, $s2`

4

## MIPS arithmetic

---

- Design Principle #1: simplicity favors regularity. Why?
- Of course this complicates some things...

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

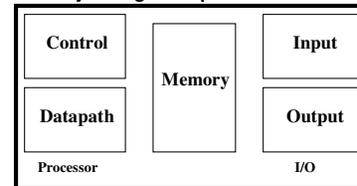
**MIPS code:**    add \$t0, \$s1, \$s2  
                  add \$s0, \$t0, \$s3  
                  sub \$s4, \$s5, \$s0

5

## 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?

6

## 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
...	

7

## Memory Organization

---

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

0	32 bits of data
4	32 bits of data
8	32 bits of data
12	32 bits of data
...	

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?

8

## 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)
```

- For lw/sw, address always = register value + offset
- How about this?
 

```
add $t0, 32($s3), $t0
```

9

## So far we've learned:

---

- MIPS
  - loading words but addressing bytes
  - arithmetic on registers only

<u>Instruction</u>	<u>Meaning</u>
add \$s1, \$s2, \$s3	\$s1 = \$s2 + \$s3
sub \$s1, \$s2, \$s3	\$s1 = \$s2 - \$s3
lw \$s1, 100(\$s2)	\$s1 = Memory[\$s2+100]
sw \$s1, 100(\$s2)	Memory[\$s2+100] = \$s1

10

## 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

11

## 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?

12

## 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

13

## 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 (DECIMAL):

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

14

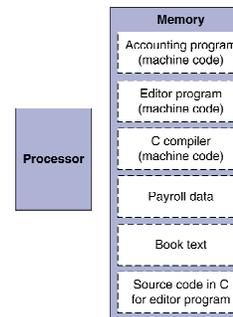
## Example Part 3

- 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 (BINARY):

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		

15

## Stored Program Computers



- Instructions represented in binary, just like data
- Instructions and data stored in memory
- Programs can operate on programs
  - e.g., compilers, linkers, ...
- Binary compatibility allows compiled programs to work on different computers
  - Standardized ISAs
- 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

16