

I do not like computer jokes ...  
not one bit!

# CMPT 295

Unit - Instruction Set Architecture

Lecture 24 – MIPS

# Last Lecture

- Assembler (part of the compilation process):
  - Transforms assembly code (`movl %edi, %eax`) into machine code (`0xf889 -> 1111100010001001`)
- Instruction Set Architecture (ISA)

A formal specification (or agreement) of ...

  - Registers and memory model, set of instructions (assembly-machine)
  - Conventions, model of computation
  - etc...
- Design principles when creating instruction set (IS)
  1. Each instruction must have an unambiguous encoding
  2. Functionally complete (Turing complete)
  3. Machine instruction format: 1) as few of them as possible 2) of the same length 3) **fields** that have the **same purpose positioned** in the **same location** in the format
- Types of instruction sets: CISC and RISC

# Today's Menu

- Instruction Set Architecture (ISA)
  - Definition of ISA
- Instruction Set design
  - Design guidelines
  - Example of an instruction set: MIPS
  - Create our own instruction sets
  - ISA evaluation
- Implementation of a microprocessor (CPU) based on an ISA
  - Execution of machine instructions (datapath)
  - Intro to logic design + Combinational logic + Sequential logic circuit
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions + Hazards



Example of another ISA: MIPS

# Example of an ISA: MIPS

## Registers and Memory model

➤ # of registers -> 32 registers (each register is 32 bit wide) See Figure on next Slide

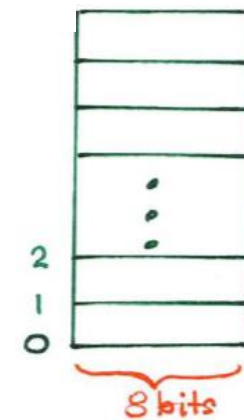
➤ Word size -> 32 bits

➤ Memory size ->  $2^m \times n$

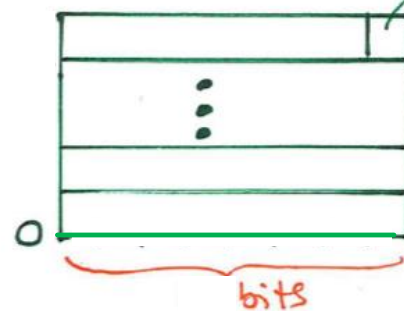
➤ Byte-addressable memory  
so address resolution ->

➤ Size of memory address (# of bits) ->

➤ So, there are \_\_\_\_\_ distinct addressable memory “chunks”  
(or “locations”) and each of these addressable memory  
“chunks” (or “locations”) has \_\_\_\_\_ bits See Figure on next next Slide



\* uncompressed  
view of  
memory.



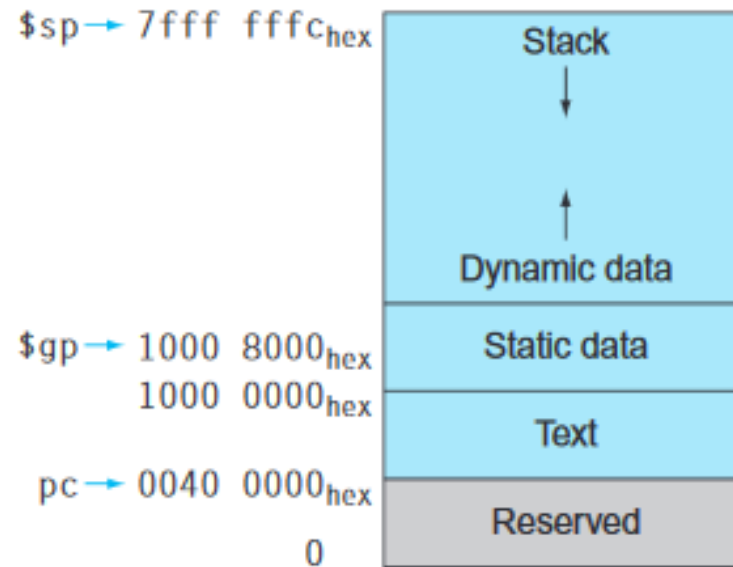
\* compressed  
view of  
memory

Register name	Number	Usage
\$zero	0	constant 0
\$at	1	reserved for assembler
\$v0	2	expression evaluation and results of a function
\$v1	3	expression evaluation and results of a function
\$a0	4	argument 1
\$a1	5	argument 2
\$a2	6	argument 3
\$a3	7	argument 4
\$t0	8	temporary (not preserved across call)
\$t1	9	temporary (not preserved across call)
\$t2	10	temporary (not preserved across call)
\$t3	11	temporary (not preserved across call)
\$t4	12	temporary (not preserved across call)
\$t5	13	temporary (not preserved across call)
\$t6	14	temporary (not preserved across call)
\$t7	15	temporary (not preserved across call)
\$s0	16	saved temporary (preserved across call)
\$s1	17	saved temporary (preserved across call)
\$s2	18	saved temporary (preserved across call)
\$s3	19	saved temporary (preserved across call)
\$s4	20	saved temporary (preserved across call)
\$s5	21	saved temporary (preserved across call)
\$s6	22	saved temporary (preserved across call)
\$s7	23	saved temporary (preserved across call)
\$t8	24	temporary (not preserved across call)
\$t9	25	temporary (not preserved across call)
\$k0	26	reserved for OS kernel
\$k1	27	reserved for OS kernel
\$gp	28	pointer to global area
\$sp	29	stack pointer
\$fp	30	frame pointer
\$ra	31	return address (used by function call)

**FIGURE A.6.1 MIPS registers and usage convention.**

Source: Page A-24 in Patterson and Hennessy

# MIPS Memory Model



**FIGURE 2.13 The MIPS memory allocation for program and data.** These addresses are only a software convention, and not part of the MIPS architecture. The stack pointer is initialized to  $7fff\ fffc_{hex}$  and grows down toward the data segment. At the other end, the program code (“text”) starts at  $0040\ 0000_{hex}$ . The static data starts at  $1000\ 0000_{hex}$ . Dynamic data, allocated by `malloc` in C and by `new` in Java, is next. It grows up toward the stack in an area called the heap. The global pointer,  $\$gp$ , is set to an address to make it easy to access data. It is initialized to  $1000\ 8000_{hex}$  so that it can access from  $1000\ 0000_{hex}$  to  $1000\ ffff_{hex}$  using the positive and negative 16-bit offsets from  $\$gp$ . This information is also found in Column 4 of the MIPS Reference Data Card at the front of this book.

# Example of an ISA: MIPS

- ▶ Instruction set

- ▶ MIPS assembly language notation

- `add a, b, c`    **Meaning:  $a = b + c$**

- ▶ 3 operand assembly language

- ▶ Requiring all instructions to have 3 operands would keep the design of the microprocessor hardware simple

- ▶ Hardware for a variable number of operands is more complicated



# Activity

- ▶ If we want to write an assembly program that sums four variables **b**, **c**, **d** and **e**, how many MIPS **add** instructions would we need?
- ▶ **Solution:**

# Example of an ISA: MIPS

➤ (Sub)set of instructions

MIPS assembly language instructions	Semantic (i.e., Meaning)	Corresponding MIPS machine instructions
lw \$s1, 20(\$s2)	$\$s1 = M[\$s2 + 20]$	?
sw \$s1, 20(\$s2)	$M[\$s2 + 20] = \$s1$	?
add \$s1, \$s2, \$s3	$\$s1 = \$s2 + \$s3$	?
sub \$s1, \$s2, \$s3	$\$s1 = \$s2 - \$s3$	?
beq \$s1, \$s2, 25	if ( $\$s1 == \$s2$ ) go to $PC + 4 + 100$	?
j 2500	go to 10000 (2500 * 4 bytes)	?
jal 2500	$\$ra = PC + 4$ ; go to 10000	?

- Memory addressing modes -> **Direct (absolute)**, **base + displacement** and **Indirect**
- Operand model ->
- Format/syntax of corresponding MIPS machine instructions?
  - Length of machine instruction -> 32 bits (**4 bytes**)
  - Size of opcode? Size of other fields? Order of operands?

Format/syntax of these bit patterns?

# A closer look at MIPS' add instruction

- MIPS assembly language instruction:

```
add $s0, $a1, $t7
```

- Corresponding MIPS machine instruction:

```
0x00af8020
```

```
-> 0000 0000 1010 1111 1000 0000 0010 0000
```

```
-> 000000 00101 01111 10000 00000 100000
```

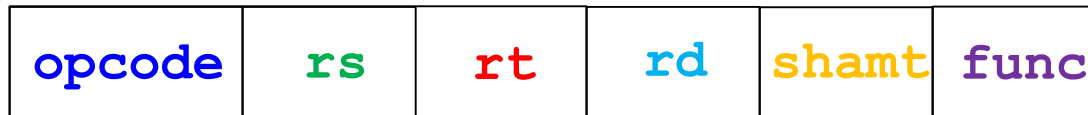
```
-> opcode rs rt rd shamt func
```

Corresponding  
machine  
instruction

Format of  
corresponding  
machine  
instruction

# MIPS machine instruction - fields

## Format of R-type MIPS instructions



- **opcode**: operation of the instruction
- **rs**: first register source operand
- **rt**: second register source operand
- **rd**: register destination operand (contains results of operation)
- **shamt**: shift amount
- **func**: function – often called *function code*, which indicates the specific variant of the operation in the **opcode** field

➤ 6 bits  
➤ 5 bits  
➤ 5 bits  
➤ 5 bits  
➤ 5 bits  
➤ 6 bits  
Total: 32 bits

# Let's examine an ISA: MIPS (3 of 3)

## ➤ Function call conventions

➤ **caller** saved registers

➤ **callee** saved registers

Register name	Number	Usage
\$zero	0	constant 0
\$at	1	reserved for assembler
\$v0	2	expression evaluation and results of a function
\$v1	3	expression evaluation and results of a function
\$a0	4	argument 1
\$a1	5	argument 2
\$a2	6	argument 3
\$a3	7	argument 4
\$t0	8	temporary (not preserved across call)
\$t1	9	temporary (not preserved across call)
\$t2	10	temporary (not preserved across call)
\$t3	11	temporary (not preserved across call)
\$t4	12	temporary (not preserved across call)
\$t5	13	temporary (not preserved across call)
\$t6	14	temporary (not preserved across call)
\$t7	15	temporary (not preserved across call)
\$s0	16	saved temporary (preserved across call)
\$s1	17	saved temporary (preserved across call)
\$s2	18	saved temporary (preserved across call)
\$s3	19	saved temporary (preserved across call)
\$s4	20	saved temporary (preserved across call)
\$s5	21	saved temporary (preserved across call)
\$s6	22	saved temporary (preserved across call)
\$s7	23	saved temporary (preserved across call)
\$t8	24	temporary (not preserved across call)
\$t9	25	temporary (not preserved across call)
\$k0	26	reserved for OS kernel
\$k1	27	reserved for OS kernel
\$gp	28	pointer to global area
\$sp	29	stack pointer
\$fp	30	frame pointer
\$ra	31	return address (used by function call)

# MIPS - Design guidelines

3. In terms of machine instruction format:

- a. Create as few of them as possible
- b. Have them all of the same length and same format!
- c. If we have in different machine instruction formats, then position the **fields** that have the **same purpose** in the **same location** in the format



► **Can all MIPS machine instructions have the same length and same format?**

► For example: `lw $s1, 20($s2)`

opcode	rs	rt	rd	shamt	func
--------	----	----	----	-------	------

► When designing its corresponding machine instruction ...

► Must specify source register using 5 bits -> OK!

► Must specify destination register using 5 bits -> OK!

► Must specify a constant using 5 bits -> Hum...

► Value of constant limited to  $[0..2^5-1]$

► Often use to access array elements so needs to be  $> 2^5 = 32$



# MIPS ISA designers compromise

- Keep all machine instructions format the same length
- Consequence -> different formats for different kinds of MIPS instructions

- *R-format* for register
- *I-format* for immediate
- *J-format* for jump

opcode 6 bits	rs 5 bits	rt 5 bits	rd 5 bits	shamt 5 bits	func 6 bits
opcode 6 bits	rs 5 bits	rt 5 bits	Address/immediate 16 bits		
opcode 6 bits	Target address 26 bits				

- **opcode** indicates the format of the instruction
  - This way, the hardware knows whether to treat the last half of the instruction as 3 fields (*R-format*) or as 1 field (*I-format*)
  - Also, position of **fields** with **same purpose** are in the **same location** in the 3 formats 😊

# Summary

- ▶ Types of instruction sets: CISC and RISC
- ▶ Looked at an example of a RISC instruction set: MIPS
  - ▶ Registers and Memory model
  - ▶ (Sub)set of instructions (assembly + machine instructions)
  - ▶ Function call conventions
  - ▶ Model of computation
- ▶ MIPS design principles
  1. Unambiguous binary encoding of instruction
  2. **IS** functionally complete ("Turing complete")
  3. Machine instruction format -> only 3 of same length **but of different format!**
    - ▶ *R-format* for register
    - ▶ *I-format* for immediate
    - ▶ *J-format* for jump
    - ▶ Also, position of **fields** with **same purpose** are in the **same location** in the 3 formats 😊



# Next lecture

- Instruction Set Architecture (ISA)
  - Definition of ISA
- Instruction Set design
  - Design guidelines
  - Example of an instruction set: MIPS
  - Create our own instruction sets
  - ISA evaluation
- Implementation of a microprocessor (CPU) based on an ISA
  - Execution of machine instructions (datapath)
  - Intro to logic design + Combinational logic + Sequential logic circuit
  - Sequential execution of machine instructions
  - Pipelined execution of machine instructions + Hazards