CS3350B
Computer Architecture
Winter 2015

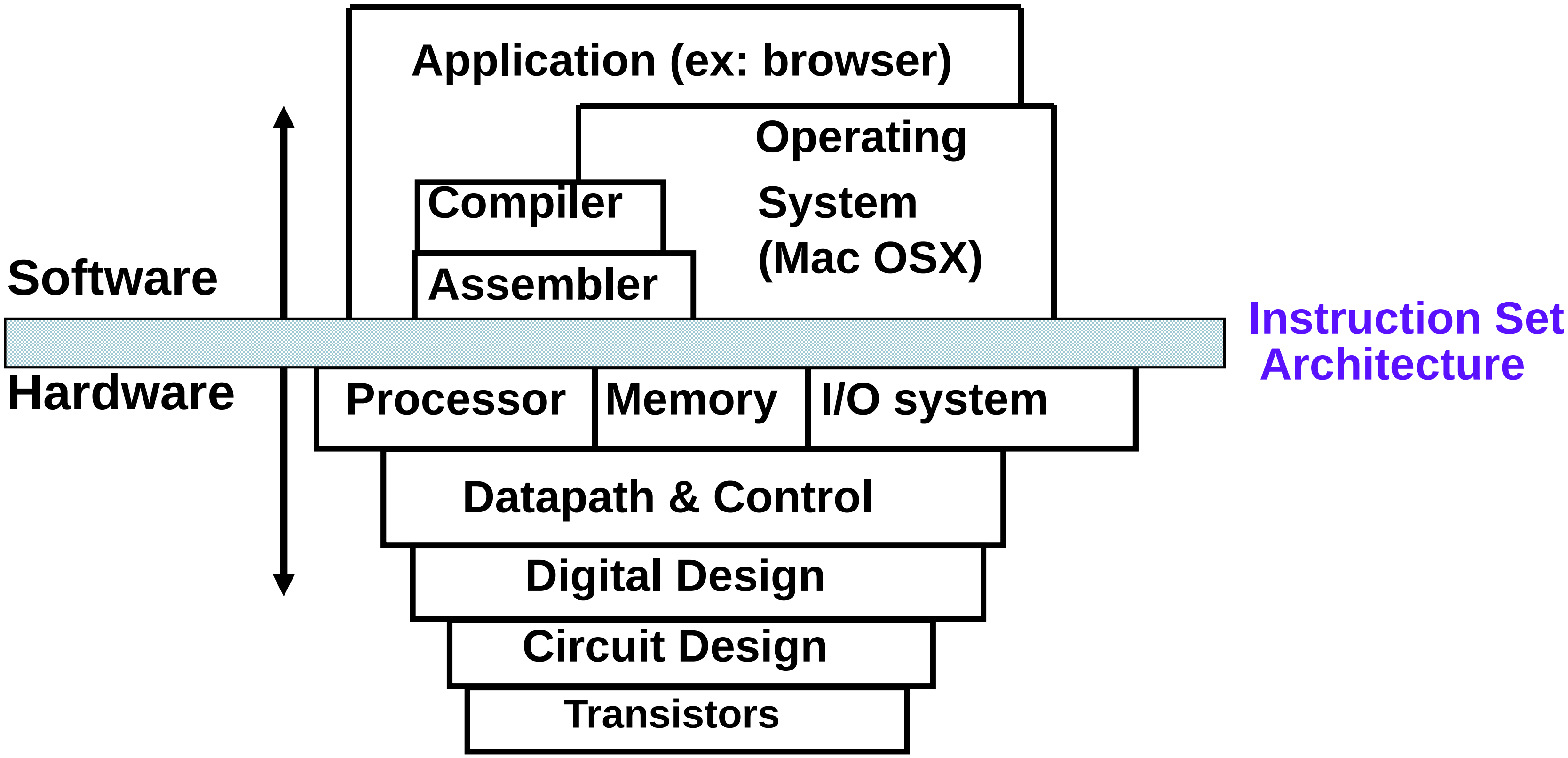
Lecture 4.1: MIPS ISA: Introduction

Marc Moreno Maza
www.csd.uwo.ca/Courses/CS3350b

[Adapted from lectures on
Computer Organization and Design,
Patterson & Hennessy, 5th edition, 2013]

Abstraction of Machine Structures

- Levels of representation



Instructions:

Language of the Computer

Instruction Set

- The repertoire of instructions of a computer
- Different computers have different instruction sets
  - But with many aspects in common
- Early computers had very simple instruction sets
  - Simplified implementation
- Many modern computers also have simple instruction sets

The MIPS Instruction Set

- Used as the example throughout the book
- Stanford MIPS commercialized by MIPS Technologies (www.mips.com)
- Large share of embedded core market
  - Applications in consumer electronics, network/storage equipment, cameras, printers, …
- Typical of many modern ISAs
  - See MIPS Reference Data tear-out card, and Appendixes B and E

spim Assembler and Simulator

- ***spim*** is a simulator that runs MIPS32 assembly language programs
  - It provides a simple assembler, debugger and a simple set of operating system services
  - Interfaces: *Spim, XSpim, PCSpim,* ***QtSpim*** *(new UI, cross-platform)*
- See installation and user guide at
  - http://pages.cs.wisc.edu/~larus/spim.html

Arithmetic Operations

- Add and subtract, three operands
  - Two sources and one destination

```
add a, b, c  # a gets b + c
```

- All arithmetic operations have this form
- ***Design Principle 1:*** Simplicity favors regularity
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at lower cost

Arithmetic Example

- C code:

```
f = (g + h) - (i + j);
```

- Compiled MIPS code:

```
add t0, g, h    # temp t0 = g + h
add t1, i, j    # temp t1 = i + j
sub f, t0, t1   # f = t0 - t1
```

Register Operands

- Arithmetic instructions use **register** operands
- MIPS has a **32 × 32-bit register file**
  - Use for frequently accessed data
  - Numbered 0 to 31
  - 32-bit data called a "word"
- Assembler names
  - $t0, $t1, …, $t9 for temporary values
  - $s0, $s1, …, $s7 for saved variables
- ***Design Principle 2:*** Smaller is faster
  - c.f. main memory: millions of locations

Register Operand Example

- C code:

```
f = (g + h) - (i + j);
```

  - f, …, j in $s0, …, $s4
- Compiled MIPS code:

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

Memory Operands

- Main memory used for composite data
  - Arrays, structures, dynamic data
- To apply arithmetic operations
  - **Load** values from memory into registers
  - **Store** result from register to memory
- Memory is **byte addressed**
  - Each address identifies an 8-bit byte
- Words are **aligned** in memory
  - Address must be a multiple of **4**
- MIPS is Big Endian
  - Most-significant byte at least address of a word
  - *c.f.* Little Endian: least-significant byte at least address

Memory Operand Example 1

- C code:

```
g = h + A[8];
```

  - g in $s1, h in $s2, base address of A in $s3
- Compiled MIPS code:
  - Index 8 requires offset of 32
    - 4 bytes per word

```
lw  $t0, 32($s3)    # load word
add $s1, $s2, $t0
```

offset → 32; base register → $s3

Memory Operand Example 2

- C code:

```
A[12] = h + A[8];
```

  - h in $s2, base address of A in $s3
- Compiled MIPS code:
  - Index 8 requires offset of 32

```
lw  $t0, 32($s3)     # load word
add $t0, $s2, $t0
sw  $t0, 48($s3)     # store word
```

Registers vs. Memory

- Registers are faster to access than memory
- Operating on memory data requires **loads** and **stores**
  - More instructions to be executed
- Compiler must **use registers for variables** as much as possible
  - Only spill to memory for less frequently used variables
  - Register optimization is important!

Immediate Operands

- **Constant data** specified in an instruction

  ```
  addi $s3, $s3, 4
  ```

- No subtract immediate instruction
  - Just use a **negative constant**

    ```
    addi $s2, $s1, -1
    ```

- ***Design Principle 3:*** Make the common case fast
  - Small constants are common
  - Immediate operand avoids a load instruction

The Constant Zero

- MIPS register 0 (**$zero**) is the constant 0
  - Cannot be overwritten
- Useful for common operations
  - E.g., move between registers

```
add $t2, $s1, $zero
```

Overview: MIPS R3000 ISA

- Instruction Categories
  - **Computational**
  - **Load/Store**
  - **Jump and Branch**
  - Floating Point
    - coprocessor
  - Memory Management
  - Special

Registers

Registers
R0 - R31
PC
HI
LO

3 Basic Instruction Formats: all **32** bits wide

OP	**rs**	**rt**	**rd**	**sha**	**funct**	R-format
OP	**rs**	**rt**	**immediate**			I-format
OP	**jump target**					J-format

MIPS Register Convention

Name	Register Number	Usage	Preserve on call?
$zero	0	constant 0 (**hardware**)	n.a.
$at	1	reserved for assembler	n.a.
$v0 - $v1	2-3	returned values	no
$a0 - $a3	4-7	arguments	**yes**
$t0 - $t7	8-15	temporaries	no
$s0 - $s7	16-23	saved values	**yes**
$t8 - $t9	24-25	temporaries	no
$k	26-27	Interrupt/trap handler	**yes**
$gp	28	global pointer	**yes**
$sp	29	stack pointer	**yes**
$fp	30	frame pointer	**yes**
$ra	31	return addr (**hardware**)	**yes**

MIPS ISA Selected Instruction Set

Category	Instr		OP/funct	Example	Meaning
Arithmetic	add	R	0/32	add $s1, $s2, $s3	$s1 = $s2 + $s3
	subtract	R	0/34	sub $s1, $s2, $s3	$s1 = $s2 - $s3
	add immediate	I	8	addi $s1, $s2, 6	$s1 = $s2 + 6
	or immediate	I	13	ori $s1, $s2, 6	$s1 = $s2 v 6
Data Transfer	load word	I	35	lw $s1, 24($s2)	$s1 = Memory($s2+24)
	store word	I	43	sw $s1, 24($s2)	Memory($s2+24) = $s1
	load byte	I	32	lb $s1, 25($s2)	$s1 = Memory($s2+25)
	store byte	I	40	sb $s1, 25($s2)	Memory($s2+25) = $s1
	load upper imm	I	15	lui $s1, 6	$s1 = 6 * 2^{16}
Cond. Branch	br on equal	I	4	beq $s1, $s2, L	if ($s1==$s2) go to L
	br on not equal	I	5	bne $s1, $s2, L	if ($s1 != $s2) go to L
	set on less than	R	0/42	slt $s1, $s2, $s3	if ($s2<$s3) $s1=1 else $s1=0
	set on less than immediate	I	10	slti $s1, $s2, 6	if ($s2<6) $s1=1 else $s1=0
Uncond. Jump	jump	J	2	j 250	go to 1000
	jump register	R	0/8	jr $t1	go to $t1
	jump and link	J	3	jal 250	go to 1000; $ra=PC+4

Unsigned Binary Integers

- Given an n-bit number

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Range: 0 to $+2^n - 1$
- Example
  - $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2$
    $= 0 + \ldots + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
    $= 0 + \ldots + 8 + 0 + 2 + 1 = 11_{10}$
- Using 32 bits
  - 0 to +4,294,967,295

2s-Complement Signed Integers

- Given an n-bit number

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Range: -2^{n-1} to $+2^{n-1} - 1$
- Example
  - $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2$
    $= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^2 + 0\times2^1 + 0\times2^0$
    $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = -4_{10}$
- Using 32 bits
  - –2,147,483,648 to +2,147,483,647

2s-Complement Signed Integers

- Bit 31 is sign bit
  - 1 for negative numbers
  - 0 for non-negative numbers
- $-(-2^{n-1})$ can't be represented
- Non-negative numbers have the same unsigned and 2s-complement representation
- Some specific numbers
  - 0: 0000 0000 … 0000
  - –1: 1111 1111 … 1111
  - Most-negative: 1000 0000 … 0000
  - Most-positive: 0111 1111 … 1111

Signed Negation

- **Complement** and **add 1**
  - Complement means 1 → 0, 0 → 1

$$x + \overline{x} = 1111...111_2 = -1$$

$$\overline{x} + 1 = -x$$

- Example: negate +2
  - $+2 = 0000\ 0000 \ldots 0010_2$
  - $-2 = 1111\ 1111 \ldots 1101_2 + 1$
    $= 1111\ 1111 \ldots 1110_2$

Sign Extension

- Representing a number using more bits
  - Preserve the numeric value
- In MIPS instruction set
  - `addi`: extend immediate value
  - `lb`, `lh`: extend loaded byte/halfword
  - `beq`, `bne`: extend the displacement
- Replicate the sign bit to the left
  - c.f. unsigned values: extend with 0s
- Examples: 8-bit to 16-bit
  - +2: 0000 0010 => 0000 0000 0000 0010
  - –2: 1111 1110 => 1111 1111 1111 1110

Next Lecture:

- MIPS Instruction Representation