

Computer Architecture

Instructions: Language of Machine

Chapter 3

saeed_zafar@comsats.edu.pk
Comsats Institute of Information Technology

1

Introduction

- Words of a machine language are called *Instructions*.
- Vocabulary of instructions is called as *Instruction set*.
- MIPS instruction set is used by *Nintendo, Silicon graphics and Sony*

2

Operations of a computer HW

- MIPS Assembly Language notation,
- Arithmetic operations
 - add a, b, c # *The sum of b and c is placed in a.*
 - add a, a, d # *The sum of b, c and d is placed in a.*
 - add a, a, e # *The sum of b, c, d and e is now in a.*

Example:

a = b + c;

d = a - e;

MIPS assembly language?

3

Compiling C into MIPS

- C language statement
 - $f = (g+h) - (i+j);$
 - What would a C compiler produce?

4

Operands of the Computer HW

- Operands of arithmetic instructions cannot be any variables.
- Must be some special locations called *Registers*.
- Register size in MIPS architecture is *32 bits*.
- MIPS has a 32 registers.

5

MIPS Instructions

- In MIPS transfer of data b/w memory and registers takes place by the data transfer instructions.
 - Instruction must supply memory address.
- Instruction which moves data from memory to register is called Load.
 - Memory address is formed by sum of constant portion of instruction and contents of second register.
 - lw stands for load instruction.
 - sw stands for store instruction.
- Example: $g = h + A[8];$

6

Compiling using Load and store

- $A[12] = h + A[8]$

Solve it?

7

MIPS assembly language

- MIPS
 - loading words but addressing bytes, i.e sequential words differ by 4.
 - 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 = \text{Memory}[\\$s2+100]$
sw \$s1, 100(\$s2)	$\text{Memory}[\\$s2+100] = \\$s1$

8

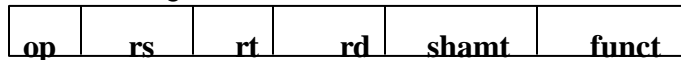
Representing Instructions in the Computer

- Human thinks in base 10 called decimal numbers.
- Computer uses base 2 numbers called binary numbers
- E.g : 123 base 10 = 1111011 base 2.
- In MIPS total 32 registers
 - \$s0 to \$7 map to 16 to 23.
 - \$t0 to \$t7 map to 8 to 15.

9

MIPS Fields

- MIPS fields are given



op: basic operation of instruction, called the opcode.

rs: The first register source operand.

rt: The second Register source operand.

rd: The register destination operand, it gets the result of the operation.

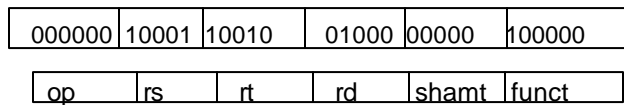
shamt: Shift amount

funct: This field selects the specific variant of the operation in the field, & called as function code.

10

R-type Format

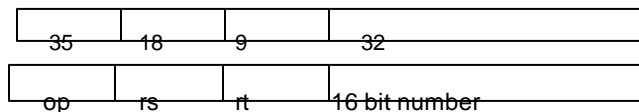
- Instructions, like registers and words of data, are also 32 bits long.
 - **Example:** add \$t0, \$s1, \$s2
 - registers have numbers, \$t0=8, \$s1=17, \$s2=18
- **R-type Format:**



11

I-type

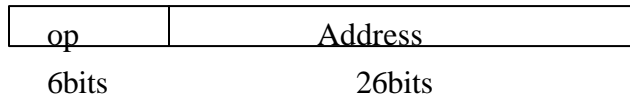
- Consider the load-word and store-word instructions,
- Introduce a new type of instruction format
 - **I-type** for data transfer instructions
 - other format was R-type for register
- **Example: lw \$t0, 32(\$s2)#** Temporary reg \$t0 gets A[8]



12

J-Type

- Jump type format



- J 10000 # go to location 10000

13

MIPS Assembly into Machine Language

- Example:

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

- Assume that \$s1 has the base of array A and \$s2 corresponds to h.
- Find MIPS assembly language code, MIPS machine language for above instructions.

14

MIPS Assembly code

- MIPS Assembly language code

```
lw $t0,1200($t1) # temp reg $t0 gets A[300]
add $t0,$s2,$t0 #Temp reg $t0 gets h+A[300]
sw $t0, 1200($t1) # stores h+A[300] back into
A[300]
```

15

MIPS Machine language code

opcode	rs	rt	rd	Address/shamt	funct
35	9	8		1200	
0	18	8	8	0	32
43	9	8		1200	

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

16

Supporting procedures in Computer HW

In the execution of the procedure, the program must follow these six steps:

1. Place parameters in a place where the procedure can access them.
2. Transfer control to the procedure.
3. Acquire the storage resources needed for the procedure.
4. Perform the desired task.
5. Place the result value in place where the calling program can access it.
6. Return control to the point of origin.

MIPS software allocates following registers for procedure calling:

- *\$a0-\$a3*: four argument registers in which to pass parameters
- *\$v0-\$v1*: two value registers in which to return values
- *\$ra*: one return register to return to the point of origin

Instruction for procedures (*jump-and-link*): *jal ProcedureAddress*

***jal* saves next instruction address (PC+ 4) to *\$ra*.**

Instruction *jr \$ra* does the return jump.

17

Stack Usage

- The Process of less commonly used variables into the memory is called ***spilling registers***.
- **The ideal data structure for spilling the registers is called *stack- Last in first out*.**
- **It needs a pointer (*stack pointer, \$sp*) to the most recently allocated address in the stack.**
- **Placing the data on the stack is called a *PUSH* & removing a data from the stack is called a *POP*.**
- **Stack grows from higher address to the lower address.**

18

MIPS Assembly

Category	Instruction	Example	meaning	Comments
Arithmetic	add	add \$s1,\$s2,\$s3	$\$s1 = \$s2 + \$s3$	3 operands; data in registers
	subtract	sub \$s1,\$s2,\$s3	$\$s1 = \$s2 - \$s3$	3 operands, data in registers
Data transfer	load word	lw \$s1,100(\$s2)	$\$s1 = M[\$s2 + 100]$	data from memory to register
	store word	sw \$s1,100(\$s2)	$M[\$s2 + 100] = \$s1$	data from register to memory
Conditional branch	on equal	beq \$s1,\$s2,L	if($\$s1 = \$s2$) goto L	Equal test and branch
	on not equal	bne \$s1,\$s2,L	if($\$s1 \neq \$s2$) goto L	Not equal test and branch
	set on less	slt \$s1,\$s2,\$s3	if($\$s2 < \$s3$) $\$s1 = 1$; else $\$s1 = 0$	Compare less than; used with <i>beq</i> , <i>bne</i>
Unconditional jump	jump	j 2500	goto 10000	Jump to target address
	jump register	jr \$t1	goto \$t1	For <i>switch</i> statement

19

Addressing in MIPS

- **Register addressing**, where the operand is a register.
- **Base or displacement addressing**, where the operand is at the memory location whose address is the sum of a register and a constant in the instruction.
- **Immediate addressing**, where the operand is a constant within the instruction itself.
- **PC-relative addressing**, where the address is the sum of the PC and a constant in the instruction.
- **Pseudodirect addressing**, where the jump address is the 26-bits of the instruction concatenated with the upper bits of the PC.

20