In the name of Allah

Computer Architecture Midterm Overview

Zohre Soorani
Mahdi Haghverdi
Hussein Husseini
Hosna Rajaei

Isfahan University

November 14, 2023

Content

Computer Abstraction and Technology

Operations of the Computer Hardware

Operands of the Computer Hardware

- Memory Operands
- Constant or Immedidate Operands

Representing Instructions

Logical Operations

Instuctions for Making Decisions

Suporting Precedures in Computer Hardware

MIPS Addressing for 32-Bit Immediates and Addresses

A C Sort Example to Put It All Together

Computer Abstraction and Technology

Computer Abstraction and Technology

Operations of the Computer Hardware

Operations of the Computer Hardware

Figure: Arithmatic Instructions in MIPS

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3	Three register operands
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3	Three register operands
	add immediate	`addi $s1,$s2,20`	$s1 = $s2 + 20	Used to add constants

Operations of the Computer Hardware (Cont'd)

load word	`lw $s1, 20($s2)`	`$s1 = Memory[$s2 + 20]`	Word from memory to register
store word	`sw $s1, 20($s2)`	`Memory[$s2 + 20] = $s1`	Word from register to memory
load byte	`lb $s1, 20($s2)`	`$s1 = Memory[$s2 + 20]`	Byte from memory to register
load byte unsigned	`lbu $s1, 20($s2)`	`$s1 = Memory[$s2 + 20]`	Byte from memory to register
store byte	`sb $s1, 20($s2)`	`Memory[$s2 + 20] = $s1`	Byte from register to memory
load upper immed	`lui $s1, 20`	`$s1 = 20 *` 2^{16}	Loads constant in upper 16 bits

Table: Data Transfer Instructions in MIPS

Operations of the Computer Hardware (Cont'd)

Figure: Logical Instructions in MIPS

Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant

Operations of the Computer Hardware (Cont'd)

Figure: Conditional Branch Instructions in MIPS

Conditional branch	branch on equal	`beq $s1,$s2,25`	if (`$s1` == `$s2`) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1,$s2,25`	if (`$s1`!= `$s2`) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1,$s2,$s3`	if (`$s2` < `$s3`) `$s1` = 1; else `$s1` = 0	Compare less than; for `beq`, `bne`
	set on less than unsigned	`sltu $s1,$s2,$s3`	if (`$s2` < `$s3`) `$s1` = 1; else `$s1` = 0	Compare less than unsigned
	set less than immediate	`slti $s1,$s2,20`	if (`$s2` < 20) `$s1` = 1; else `$s1` = 0	Compare less than constant
	set less than immediate unsigned	`sltiu $s1,$s2,20`	if (`$s2` < 20) `$s1` = 1; else `$s1` = 0	Compare less than constant unsigned

Operations of the Computer Hardware (Cont'd)

Figure: Unconditional Jump Instructions in MIPS

Unconditional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to `$ra`	For switch, procedure return
	jump and link	`jal 2500`	`$ra` = PC + 4; go to 10000	For procedure call

Example - Compiling a Complex C Assignment into MIPS

A somewhat complex statement contains the five variables `f`, `g`, `h`, `i`, and `j`:

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

What might a C compiler produce?

Answer

- `add t0, g, h # temporary variable t0 contains g + h`
- `add t1, i, j # temporary variable t1 contains i + j`
- `sub f, t0, t1 # f gets t0 - t1, which is (g + h) - (i + j)`

Operands of the Computer Hardware

Example - Compiling a C Assignment Using Registers

It is the compiler's job to associate program variables with registers.
Take, for instance, the assignment statement from our earlier example:

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

The variables `f`, `g`, `h`, `i`, and `j` are assigned to the registers `$s0`, `$s1`, `$s2`, `$s3`, and `$s4`, respectively.
What is the compiled MIPS code?

Answer

- `add $t0, $s1, $s2 # register $t0 contains g + h`
- `add $t1, $s3, $s4 # register $t1 contains i + j`
- `sub $s0, $t0, $t1 # f gets $t0 - $t1, which is (g + h) - (i + j)`

Example - Compiling Using Load and Store

Assume variable `h` is associated with register `$s2` and the base address of the array `A` is in `$s3`.
What is the MIPS assembly code for the C assignment state ment below?

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

Answer

- `lw $t0, 32($s3) # Temporary reg $t0 gets A[8]`
- `add $t0, $s2, $t0 # Temporary reg $t0 gets h + A[8]`
- `sw $t0, 48($s3) # Stores h + A[8] back into A[12]`

Constant or Immediate Operands

- `addi $s3, $s3, 4    # $s3 = $s3 + 4`

Representing Instructions

Instructions Big Picture

Name	Format	Example						Comments
Field Size		6 bit	5 bit	5 bit	5 bit	5 bit	6 bit	All MIPS instructions are 32 bits long
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
`add`	R	0	18	19	17	0	32	add `$s1,$s2,$s3`
I-format	I	op	rs	rt	address			Data transfer format
`lw`	I	35	18	17	100			lw `$s1,100($s2)`
J-format	J	op	address					Unconditional Branch
`j`	J	8	300					jump to address

Logical Operations

Logical Operations Big Picture

Logical operations	C operators	Java operators	MIPS instructions
Shift left	<<	<<	`sll`
Shift right	>>	>>>	`srl`
Bit-by-bit AND	&	&	`and`, `andi`
Bit-by-bit OR	|	|	`or`, `ori`
Bit-by-bit NOT	~	~	`nor`

Instuctions for Making Decisions

Instructions for Making Decisions

- `beq register1, register2, L1`
  This instruction means go to the statement labeled `L1` if the value in `register1` equals the value in `register2`.
  The mnemonic beq stands for *branch if equal*.

- `bne register1, register2, L1`
  It means go to the statement labeled `L1` if the value in `register1` does not equal the value in `register2`.
  The mnemonic `bne` stands for *branch if not equal*.

Example - Compiling *if-then-else* into Conditional Branches

In the following code segment, `f`, `g`, `h`, `i`, and `j` are variables.
If the five variables, `f` through `j`, correspond to the five registers `$s0` through `$s4`, what is the compiled MIPS code for this C if statement?

```
if i == j:
    f = g + h
else:
    f = g - h
```

Answer

```
bne $s3, $s4, Else          # go to Else if i != j
add $s0, $s1, $s2           # f = g + h (skipped if i != j)
j   Exit                    # go to Exit
Else:
    sub $s0, $s1, $s2       # f = g - h (skipped if i = j)
Exit:
```

Example - Compiling a *while* Loop in C

Here is a traditional loop in C:

```c
while (save[i] == k){
    i += 1;
}
```

Assume that `i` and `k` correspond to registers `$s3` and `$s5` and the base of the array `save` is in `$s6`.

What is the MIPS assembly code corresponding to this C segment?

Answer

```
Loop:   sll    $t1,$s3,2        # Temp reg $t1 = i * 4
        add    $t1,$t1,$s6      # $t1 = address of save[i]
        lw     $t0,0($t1)       # Temp reg $t0 = save[i]
        bne    $t0,$s5, Exit    # go to Exit if save[i] != k
        addi   $s3,$s3,1        # i = i + 1
        j      Loop             # go to Loop
Exit:
```

Suporting Precedures in Computer Hardware

MIPS Addressing for 32-Bit Immediates and Addresses

A C Sort Example to Put It All Together