Midterm Review Guide

1 Basic MIPS Instructions

Noatation:

is the comment symboal.

Syntax Semantic

#arithmetic instructions

```
add     rd, rs, rt          #Put the sum of registers rs and rt into register rd.
addi    rt, rs, imm         #Put the sum of registers rs and sign-extended immediate imm  into register rt.
and     rd, rs, rt          #Put the logical AND of registers rs and rt into register rd.
andi    rt, rs, imm         #Put the logical AND of register rs and zero-extended imm  into register rt.
sub     rd, rs, rt          #Put the diff erence of registers rs and rt into register rd.
```

#bitwise instructions

```
not     rdest, rsrc         #Put the logical NOT of registers rs and rt into register rd.
or      rd, rs, rt          #Put the logical OR of registers rs and rt into register rd.
ori     rt, rs, imm         #Put the logical OR of register rs and the zero-extended immediate into register rt.

sll rd, rt, shamt       #Shift register rt left  by the distance indicated by immediate shamt or the
                        #register rs and put the result in register rd.

srl rd, rt, shamt       #Shift register rt right by the distance indicated by immediate shamt or the
                        #register rs and put the result in register rd.
```

#Load Instructions

```
lw  rd, offset(rt)      # load a word to register rd from the memory whose address is sum of offset and
                        # register rt.
```

sw rd, offset(rt) # save the register rd to the memory whose address is sum of offset and register rt.

#comparison Instructions

slt rd, rs, rt #Set register rd to 1 if register rs is less than rt, and to 0 otherwise.

slti rt, rs, imm #Set register rt to 1 if register rs is less than imm, and to 0 otherwise.

#Branch Instructions

beq rs, rt, label #Conditionally jump to the instruction with label if register rs equals rt.

bne rs, rt, label #Conditionally jump to the instruction with label if register rs is not equal to rt.

#Jump instructions

j target #Unconditionally jump to the instruction whose lable is target.

jal target #Unconditionally jump to the instruction whose label is target. Save the

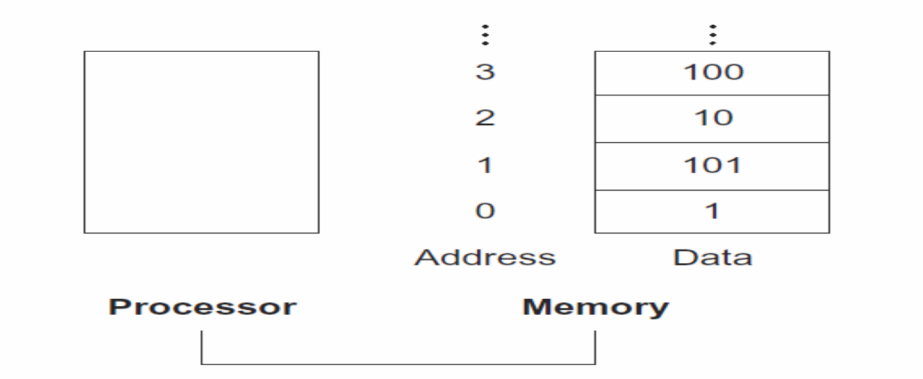
#address of the next instruction in register $ra. It is used to call procedure.

jr rs #Unconditionally jump to the instruction whose address is in register rs.

2 Examples for Instructions

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
Data transfer	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 half	`lh $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	`lhu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	`sh $s1,20($s2)`	Memory[$s2 + 20] = $s1	Halfword 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 linked word	`ll $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	`sc $s1,20($s2)`	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	`lui $s1,20`	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
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
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
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

3 Memory Address



Read page 70 at textbook.

Keep in mind that an address only corresponds a storage unit which is one byte. So the memory is also called byte-addressable.

4 Calculate array element address

Read page 71 at textbook.

An element of array could need more than one byte. For example, one word has 4 bytes. So the index of array is not same as the offset address for the array element.

Address of A[i] = base address of array + i*sizeof(array element)

5 Translate instruction to binary

Read page 80~85 at textbook

6 Translate loop statement to assembly language

Read page 92~93 at textbook

7 Translate function call to assembly language

Read section 2.8 and A.6 at textbook

7.1 Passing parameters with registers

Caller passes the parameters callee and callee returns results with registers.

7.2 Save and restore registers

Since the number of register is very limited. For instance, the MIPS has only 32 registers. In the procedure, it is highly likely to use the registers which hold data for the caller, which means these

registers data loss, causing errors. In avoid this issue, we need save these affected registers before transfer control to procedure and restore them before returning from procedure.

7.3 Where are affected register store?

They are stored in stack.

7.4 Saving data in stack

We save the data in stack. Pushing data to the top of stack needs to change stack pointer(sp), which is always pointing to the top of stack. After pushing, sp is reduced by the size of saved data.

7.5 Restoring data in stack

We load the data from stack. Popping data from the top of stack needs to change stack pointer(sp), which is always pointing to the top of stack. After popping, sp is increased by the size of saved data.