contact: sujinohkor@gmail.com / Operating System Laboratory (A1409)

MIPS Register Set

Name		Number	Use
$0	R0	0	the constant value 0
$at	$1	1	assembler temporary
$v0-$v1	$2-$3	2-3	function return value
$a0-$a3	$4-$7	4-7	function arguments
$t0-$t7	$8-$15	8-15	temporary variables
$s0-$s7	$16-$23	16-23	saved variables
$t8-$t9	$24-$25	24-25	temporary variables
$k0-$k1	$26-$27	26-27	operating system (OS) temporaries
$gp	$28	28	global pointer
$sp	$29	29	stack pointer
$fp	$30	30	frame pointer
$ra	$31	31	function return address

Preserved and Non-preserved Registers

Preserved		Non-preserved	
Saved registers	$s0-$s7	Temporary registers	$t0-$t9
Return address	$ra	Argument registers	$a0-$a3
Stack pointer	$sp	Return value registers	$v0-$v1
Stack above the stack pointer		Stack below the stack pointer	

MIPS System Calls

System Call Code	Service	Arguments	Result
1	Print integer	$a0 : integer value	none
2	Print float	$f12 : float value	none
3	Print double	$f12 : double value	none
4	Print string	$a0 : pointer to string	none
5	Read integer	none	integer returned in $v0
6	Read float	none	float returned in $f0
7	Read double	none	double returned in $f0
8	Read string	$a0 : address where string to be stored $a1 : length of string buffer	none
9	Memory allocation	$a0 : amount	$v0 : address of block
10	Exit program	none	none
11	Print character	$a0 : integer	none
12	Read character	none	a character returned in $v0

Instruction Formats

R-Type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

J-Type

op	addr
6 bits	26 bits

MIPS Assembly Instructions

<table>
<tr><th>R-Type</th><th>I-Type</th><th>J-Type</th></tr>
<tr><td>add rd, rs, rt
e.g. add $s0, $s1, $s2
$s0 <- $s1+$s2</td><td>addi rt, rs, imm
e.g. addi $s1, $s0, 5
$s1 <- $s0+5</td><td>j / jal
e.g. j target
jr is R-Type instruction.
e.g. jr $s0
$s0's value will be the address</td></tr>
<tr><td>sub rd, rs, rt
e.g. sub $t0, $s1, $s2
$t0 <- $s1-$s2</td><td>lw rt, imm(rs)
e.g. lw $s1, 8($0)
read data word 2 (start from $0) into $s1</td><td rowspan="2">Branch Instructions (I-Type)
beq / bne
e.g. beq $s0, $s1, target
if $s0 == $s1, then branch
e.g. bne $s0, $s1, target
if $s0 != $s1, then branch</td></tr>
<tr><td>sll / srl / sra
e.g. sll $t0, $s1, 4
Left/Right Logical/Arithmetic</td><td>sw rt, imm(rs)
e.g. sw $s3, 4($0)
write $s3 to data word 1</td></tr>
<tr><td>and / or / xor / nor
e.g. xor $s5, $s1, $s2
xor: both are different, then 1
nor: both are 0, then 1</td><td>andi / ori / xori
e.g. xori $s4, $s1, 0x34
nori is not provided</td><td>Please read our textbook!!! :D
Digital Design & Com Architecture</td></tr>
</table>

Conditional Statements; if and if-else

If

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

f = f – i;
```

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j
bne     $s3, $s4, L1      # if i != j, then skip if block
add     $s0, $s1, $s2     # if block: f = g + h

L1:
sub     $s0, $s0, $s3     # f = f - i
```

If Else

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

else
        f == f – i;
```

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j
bne     $s3, $s4, else    # if i != j, then branch
add     $s0, $s1, $s2     # if block: f = g + h
j       L2                # skip else block

else:
sub     $s0, $s0, $s3     # else block: f = f - i

L2:
```

Conditional Statements; switch-case

Switch Case

```
switch(amount) {
case 20:
        fee = 2;
        break;

case 50:
        fee = 5;
        break;

default:
        fee = 0;

}
```

```
# $s0 = amount, $s1 = fee
case20:
addi    $t0, $0, 20         # $t0 = 20
bne     $s0, $t0, case50    # if i != 20, then branch
addi    $s1, $0, 2          # if i == 20, then fee = 2
j       done                # break out of case

case50:
addi    $t0, $0, 50         # $t0 = 50
bne     $s0, $t0, default   # if i != 50, then branch
addi    $s1, $0, 5          # if i == 50, then fee = 5
j       done                # break out of case

default:
add     $s1, $0, $0         # charge = 0

done:
```

Loop Statements

While

```
int pow = 1;
int x = 0;

while(pow != 128) {
        pow = pow * 2;
        x = x + 1;

}
```

```
# $s0 = pow, $s1 = x
addi    $s0, $0, 1          # pow = 1
addi    $s1, $0, 0          # x = 0

addi    $t0, $0, 128        # $t0 = 128 for comparison
while:
beq     $s0, $t0, done      # if pow == 128, exit while
sll     $s0, $s0, 1         # pow = pow * 2
addi    $s1, $s1, 1         # x = x + 1
j       while

done:
```

For

```
int sum = 0;

for(i = 0; i != 10; i = i + 1) {
        sum = sum + i;

}
```

```
# $s0 = i, $s1 = sum
add     $s1, $0, $0         # sum = 0
addi    $s0, $0, 0          # i = 0
addi    $t0, $0, 10         # $t0 = 10

for:
beq     $s0, $t0, done      # if i == 10, then branch
add     $s1, $s1, $s0       # sum = sum + i
addi    $s0, $s0, 1         # increment i
j       for

done:
```