<u>MIPS registers</u>

MIPS has 32 registers r0-r31. The conventional use of these registers is as follows:

register	assembly name	Comment
r0	$zero	Always 0
r1	$at	Reserved for assembler
r2-r3	$v0-$v1	Stores results
r4-r7	$a0-$a3	Stores arguments
r8-r15	$t0-$t7	Temporaries, not saved
r16-r23	$s0-$s7	Contents saved for use later
r24-r25	$t8-$t9	More temporaries, not saved
r26-r27	$k0-$k1	Reserved by operating system
r28	$gp	Global pointer
r29	$sp	Stack pointer
r30	$fp	Frame pointer
r31	$ra	Return address

Loading a 32-bit constant into a register

lui $s0, 42 # load upper-half immediate

ori $s0, $s0, 18 # (one can also use andi)

What is the end result?

Review the logical operations

Shift left logical sll

Shift right logical srl

Bit-by-bit AND and, andi (and immediate)

sll $t2, $s0, 4 # register $t2 := register $s0 << 4

s0 = 0000 0000 0000 0000 0000 0000 0000 1001

t2 = 0000 0000 0000 0000 0000 0000 1001 0000

Op = 0	rs = 0	rt = 16	rd = 10	shamt = 4	function = 0

(s0 = r16, t2 = r10)

What are the uses of shift instructions?

Multiply or divide by some power of 2.

Implement general multiplication using addition and shift

Making decisions

if (i == j) f = g + h; else f = g - h

Use **bne** = branch-nor-equal, **beq** = branch-equal, and **j** = jump

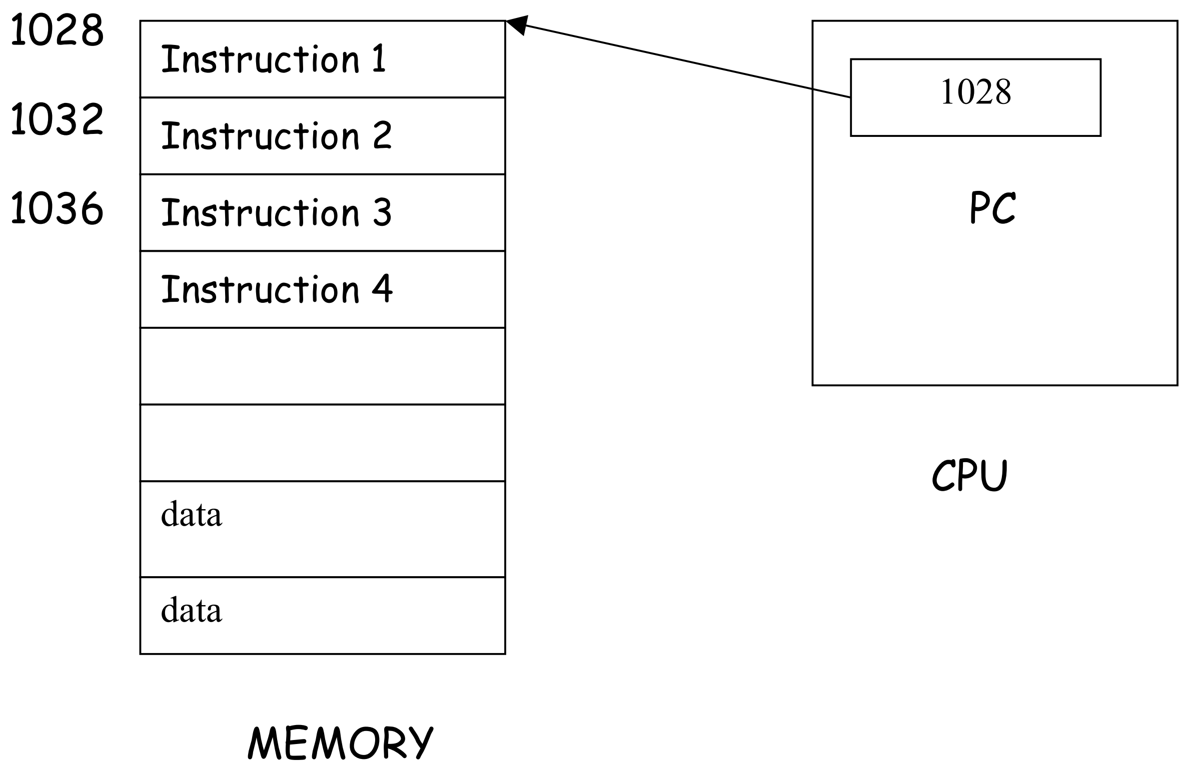
Assume that f, g, h, are mapped into $s0, $s1, $s2

i, j are mapped into $s3, $s4

```
        bne $s3, $s4, Else      # goto Else when i=j
        add $s0, $s1, $s2       # f = g + h
        j     Exit              # goto Exit
Else:   sub $s0, $s1, $s2       # f = g – h
Exit:
```

The program counter

Every machine has a **program counter** (called PC) that points to the next instruction to be executed.



Ordinarily, PC is incremented by 4 after each instruction is executed. A branch instruction alters the flow of control by modifying the PC.

Compiling a while loop

while (A[i] == k)   i = i + j;

Initially $s3, $s4, $s5 contains i, j, k respectively.

Let $s6 store the base of the array A. Each element of A is a 32-bit word.

```
Loop:   add $t1, $s3, $s3      # $t1 = 2*i
        add $t1, $t1, $t1      # $t1 = 4*i
        add $t1, $t1, $s6      # $t1 contains address of A[i]
        lw $t0, 0($t1)         # $t0 contains $A[i]
        add $s3, $s3, $s4      # i = i + j
        bne $t0, $s5, Exit     # goto Exit if A[i] ≠ k
        j   Loop               # goto Loop
Exit:   <next instruction>
```

Note the use of pointers.

Exercise

Add the elements of an array A[0..63]. Assume that the first element of the array is stored from address 200. Store the sum in address 800.

System Call

The program takes the help of the operating system to do some input or output operation. Example

```
li    $v0, 5          # System call code for Read Integer
syscall               # Read N into $v0
```

Compiling a switch statement

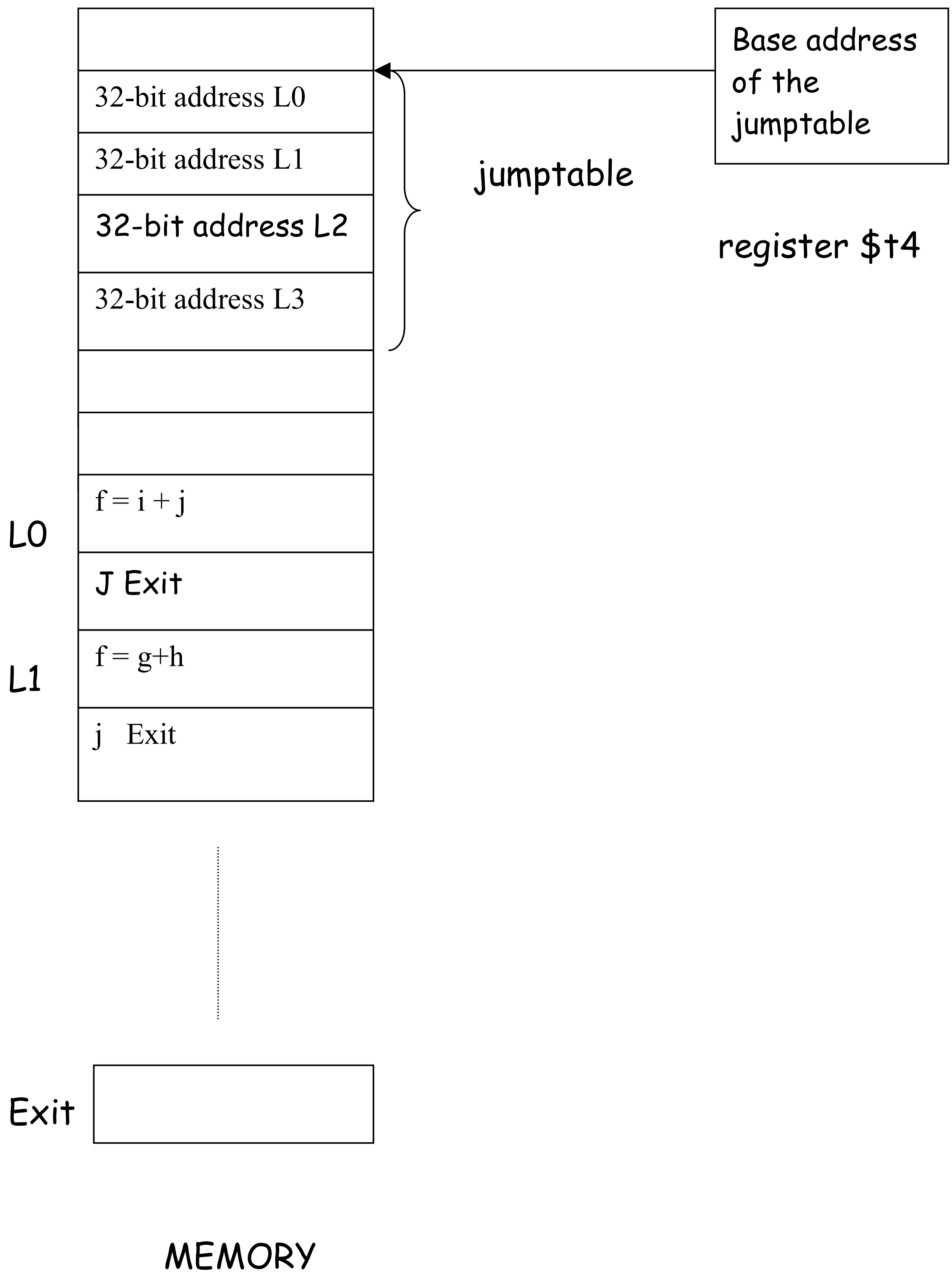
```
switch (k) {
    case 0:    f = i + j; break;
    case 1:    f = g + h; break;
    case 2:    f = g – h; break;
    case 3:    f = i – j; break;
}
```

Assume, \$s0-\$s5 contain f, g, h, i, j, k.

Assume \$t2 contains 4.

```
slt $t3, $s5, $zero       # if k<0 then $t3 = 1 else $t3=0
bne $t3, $zero, Exit      # if k<0 then Exit
slt $t3, $s5, $t2         # if k<4 then $t3 = 1 else $t3=0
beq $t3, $zero, Exit      # if k≥ 4 the Exit
```

What next? Jump to the right case!


Base address of the jumptable
register $t4
jumptable
32-bit address L0
32-bit address L1
32-bit address L2
32-bit address L3
L0
f = i + j
J Exit
L1
f = g+h
j Exit
Exit
MEMORY

Here is the remainder of the program;

```
        add $t1, $s5, $s5      # t1 = 2*k
        add $t1, $t1, $t1      # t1 = 4*k
        add $t1, $t1, $t4      # t1 = base address + 4*k
        lw $t0, 0($t1)         # load the address pointed
                               # by t1 into register t0
        jr $t0                 # jump to addr pointed by t0
L0:     add $s0, $s3, $s4      # f = i + j
        J Exit
L1:     add $s0, $s1, $s2      # f = g+h
        J Exit
L2:     sub $s0, $s1, $s2      # f = g-h
        J Exit
L3:     sub $s0, $s3, $s4      # f = i - j
Exit: <next instruction>
```