**MIPS registers**

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

<table>
<thead>
<tr>
<th>register</th>
<th>assembly name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>$zero</td>
<td>Always 0</td>
</tr>
<tr>
<td>r1</td>
<td>$at</td>
<td>Reserved for assembler</td>
</tr>
<tr>
<td>r2-r3</td>
<td>$v0-$v1</td>
<td>Stores results</td>
</tr>
<tr>
<td>r4-r7</td>
<td>$a0-$a3</td>
<td>Stores arguments</td>
</tr>
<tr>
<td>r8-r15</td>
<td>$t0-$t7</td>
<td>Temporaries, not saved</td>
</tr>
<tr>
<td>r16-r23</td>
<td>$s0-$s7</td>
<td>Contents saved for use later</td>
</tr>
<tr>
<td>r24-r25</td>
<td>$t8-$t9</td>
<td>More temporaries, not saved</td>
</tr>
<tr>
<td>r26-r27</td>
<td>$k0-$k1</td>
<td>Reserved by operating system</td>
</tr>
<tr>
<td>r28</td>
<td>$gp</td>
<td>Global pointer</td>
</tr>
<tr>
<td>r29</td>
<td>$sp</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>r30</td>
<td>$fp</td>
<td>Frame pointer</td>
</tr>
<tr>
<td>r31</td>
<td>$ra</td>
<td>Return address</td>
</tr>
</tbody>
</table>
**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 0000 1001 0000

<table>
<thead>
<tr>
<th>Op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>16</td>
<td>10</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

(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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bne $s3, $s4, Else</code></td>
<td># goto Else when i=j</td>
</tr>
<tr>
<td><code>add $s0, $s1, $s2</code></td>
<td># f = g + h</td>
</tr>
<tr>
<td><code>j Exit</code></td>
<td># goto Exit</td>
</tr>
<tr>
<td>Else: <code>sub $s0, $s1, $s2</code></td>
<td># f = g - h</td>
</tr>
<tr>
<td>Exit:</td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>add $t1, $s3, $s3</td>
<td># $t1 = 2*i</td>
</tr>
<tr>
<td></td>
<td>add $t1, $t1, $t1</td>
<td># $t1 = 4*i</td>
</tr>
<tr>
<td></td>
<td>add $t1, $t1, $s6</td>
<td># $t1 contains address of A[i]</td>
</tr>
<tr>
<td></td>
<td>lw $t0, 0($t1)</td>
<td># $t0 contains $A[i]</td>
</tr>
<tr>
<td></td>
<td>add $s3, $s3, $s4</td>
<td># i = i + j</td>
</tr>
<tr>
<td></td>
<td>bne $t0, $s5, Exit</td>
<td># goto Exit if A[i] ≠ k</td>
</tr>
<tr>
<td>j Loop</td>
<td></td>
<td># goto Loop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exit:</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;next instruction&gt;</td>
</tr>
</tbody>
</table>

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!
jumptable

Base address of the jumptable

register $t4

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
      J Exit
Exit: <next instruction>
```