MIPS Assembly Language Programming

Here is what you will need to understand to write MIPS assembly language programs and test them using the "MARS" simulator.

Register layout:

The MIPS processor we will be using is a 32 bit processor and has 32 general purpose integer registers, each 32 bits long.

In assembly language you can refer to a register using its name or its number. All register references begin with a "$".

Here is a description of the registers:

NAME	NUMBER	PURPOSE
$zero	0	Always equal to zero
$at	1	Reserved for use by assembler
$v0, $v1	2-3	Used for return **v**alues
$a0 .. $a3	4-7	Used to pass arguments
$t0 .. $t9	8-15, 24-25	Temporary registers
$s0 .. $s7	16-23	Saved registers
$k0, $k1	26-27	Reserved for use by kernel
$gp	28	pointer to global data area
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address from a function

Common MIPS Instructions:

(assume rd, rs and rt are general purpose registers)
(imm is an immediate (constant))

Arithmetic/Logic Instructions:

add	rd, rs, rt	rd = rs + rt	
addi	rd, rs, imm	rd = rs + imm	
and	rd, rs, rt	rd = rs & rt	
andi	rd, rs, imm	rd = rs & imm	
div	rd, rs, rt	rd = rs / rt	(rd gets quotient)
mul	rd, rs, rt	rd = rs * rt	
neg	rd, rs	rd = - rs	(two's complement)
not	rd, rs	rd = ! rs	(one's complement)
or	rd, rs, rt	rd = rs | rt	
ori	rd, rs, imm	rd = rs | imm	
rem	rd, rs, rt	rd = rs % rt	
sub	rd, rs, rt	rd = rs - rt	

(NOTE: not, rem are pseudo-instructions)

Constant Manipulation Instructions:

lui	rd, imm	(load upper immediate) load lower 16 bits of imm into upper 16 bits of rd. Note: lower 16 bits of rd set to zero.

li	rd, imm	(load immediate) assign imm to rd. Note: imm can be negative.

(li is a pseudo-instruction)

Branch Instructions:

b	label	unconditional branch - pseudo instruction
beq	rs, rt, label	if(rs == rt) goto label
bne	rs, rt, label	if(rs != rt) goto label
bge	rs, rt, label	>=
bgt	rs, rt, label	>
ble	rs, rt, label	<=
blt	rs, rt, label	<

Jump Instructions:

j	label	unconditional jump
jal	label	jump and link (function call)
jr	rs	jump register (rs has address)

Memory Access Instructions: (load and store only)

la	rd, label	load address of the label into rd
lb	rd, address	load byte from address into rd
lw	rd, address	load word from address into rd
sb	rd, address	store byte rd to address
sw	rd, address	store word rd to address

Data Movement Instructions:

move	rd, rs	move contents of rs into rd (pseudo-instruction)

I-O Services (System calls) NOTE: Service number must be in $v0 register.

SERVICE	NUMBER	ARGS	RESULT
print_int	1	$a0 = int	
print_string	4	$a0 = &string	
read_int	5		int left in $v0
read_string	8	$a0=&buffer, $a1=maxlen	
exit to O.S.	10		
print_char	11	$a0 = char	
read_char	12		char left in $v0

Data Area - Declaring variables

Assembler directives are used to allocate memory for variables

To declare a 32 bit int with the name “num1” with the inital value zero:

```
num1:       .word       0
```

To declare an array of three ints named “nums” with initial values of zero.

```
nums:       .word       0, 0, 0
```

To declare an array of ascii characters name “buffer”.

```
buffer:     .ascii      “                              ”
```

To declare a C-String with a null terminator named “prompt.

```
prompt:     .asciiz     “Enter a value”
```

To declare a group of 40 bytes with no defined purpose.

```
junk:  .space       40
```

Addressing Modes - How to refer to memory locations

Assume that you have a variable named “num1”. To load a copy of num1 into register $t0.

```
lw      $t0, num1
```

Or you could do the same thing like this.

```
la      $t1, num1
lw      $t0, ($t1)
```

Assume that you have an array of characters called “string”. To load the first character from the string into register $s0.

```
li      $t1, 0
lb      $t0, string ($t1)
```

Now, to get the second character from the array called “string”.

```
addi    $t1, 1
lb      $t0, string ($t1)
```

Storing information is simply the reverse of loading. To store one byte from register $t2 to the first byte of an array called “string”.

```
li      $t1, 0
sb      $t0, string ($t1)
```