**MIPS ASSEMBLY LANGUAGE BASICS**

**Registers**

* 32 general-purpose registers
* register preceded by $ in assembly language instruction  
  two formats for addressing:
  + using register number e.g. $0 through $31
  + using equivalent names e.g. $t1, $sp
* special registers Lo and Hi used to store result of multiplication and division
  + not directly addressable; contents accessed with special instruction mfhi ("move from Hi") and mflo ("move from Lo")
* stack grows from high memory to low memory

|  |  |  |
| --- | --- | --- |
| **Register Number** | **Alternative  Name** | **Description** |
| 0 | zero | the value 0 |
| 1 | $at | (**a**ssembler **t**emporary) reserved by the assembler |
| 2-3 | $v0 - $v1 | (**v**alues) from expression evaluation and function results |
| 4-7 | $a0 - $a3 | (**a**rguments) First four parameters for subroutine. Not preserved across procedure calls |
| 8-15 | $t0 - $t7 | (**t**emporaries) Caller saved if needed. Subroutines can use w/out saving. Not preserved across procedure calls |
| 16-23 | $s0 - $s7 | (**s**aved values) - Callee saved.  A subroutine using one of these must save original and restore it before exiting. Preserved across procedure calls |
| 24-25 | $t8 - $t9 | (**t**emporaries) Caller saved if needed. Subroutines can use w/out saving. These are in addition to $t0 - $t7 above. Not preserved across procedure calls. |
| 26-27 | $k0 - $k1 | reserved for use by the interrupt/trap handler |
| 28 | $gp | **g**lobal **p**ointer.  Points to the middle of the 64K block of memory in the static data segment. |
| 29 | $sp | **s**tack **p**ointer  Points to last location on the stack. |
| 30 | $s8/$fp | **s**aved value / **f**rame **p**ointer Preserved across procedure calls |
| 31 | $ra | **r**eturn **a**ddress |

## **Program Structure**

* just plain text file with data declarations, program code (name of file should end in suffix .s to be used with SPIM simulator)
* data declaration section followed by program code section

### Data Declarations

* placed in section of program identified with assembler directive **.data**
* declares variable names used in program; storage allocated in main memory (RAM)

### Code

* placed in section of text identified with assembler directive **.text**
* contains program code (instructions)
* starting point for code execution given label **main:**
* ending point of main code should use exit system call (see below under System Calls)

### Comments

* anything following # on a line   
  # This stuff would be considered a comment
* Template for a MIPS assembly language program:
* # Comment giving name of program and description of function
* # Template.s
* # Bare-bones outline of MIPS assembly language program
* .data # variable declarations follow this line
* # ...
* .text # instructions follow this line
* main: # indicates start of code (first instruction to execute)
* # ...
* # End of program, leave a blank line afterwards to make SPIM happy

## **Data Declarations**

format for declarations:

name: storage\_type value(s)

* + create storage for variable of specified type with given name and specified value
  + value(s) usually gives initial value(s); for storage type .space, gives number of spaces to be allocated

Note: labels always followed by colon ( : )

example

var1: .word 3 # create a single integer variable with initial value 3

array1: .byte 'a','b' # create a 2-element character array with elements initialized

# to a and b

array2: .space 40 # allocate 40 consecutive bytes, with storage uninitialized

# could be used as a 40-element character array, or a

# 10-element integer array; a comment should indicate which!

**Load / Store Instructions**

* RAM access only allowed with load and store instructions
* all other instructions use register operands

load:

lw register\_destination, RAM\_source

#copy word (4 bytes) at source RAM location to destination register.

lb register\_destination, RAM\_source

#copy byte at source RAM location to low-order byte of destination register,  
# and sign-extend to higher-order bytes

store word:

sw register\_source, RAM\_destination

#store word in source register into RAM destination

sb register\_source, RAM\_destination

#store byte (low-order) in source register into RAM destination

load immediate:

li register\_destination, value

#load immediate value into destination register

example:

.data

var1: .word 23 # declare storage for var1; initial value is 23

.text

\_start:

lw $t0, var1 # load contents of RAM location into register

$t0: $t0 = var1

li $t1, 5 # $t1 = 5 ("load immediate")

sw $t1, var1 # store contents of register $t1 into RAM:

var1 = $t1

done

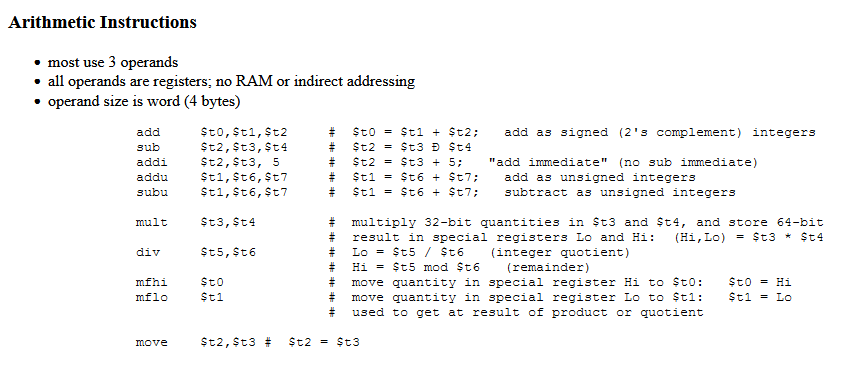
**Indirect Addressing**

* Used only with load and store instructions

load address:

la $t0, var1

* copy RAM address of var1 (presumably a label defined in the program) into register $t0

NOn

Note

The mul and div can take 3 operand instead of two e.g

mul $t0, $t1,$t2 # This multiplies the content of register $t1 and $t2 and store the result in $t0. This is more preferred.

The same applies to div

div $t0, $t1,$t2 # This divides the content of register $t1 by $t2 and store the result in $t0. This is more preferred.