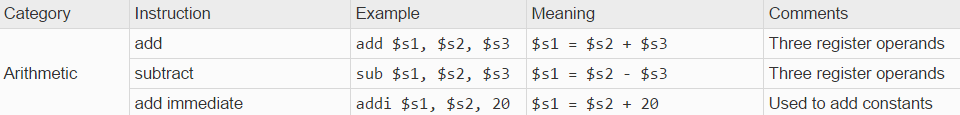
Portfolio 2

MIPS Instructions

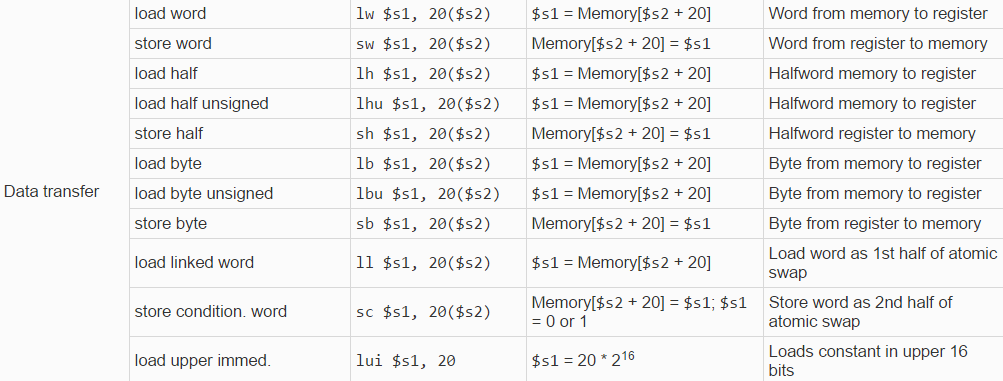


Ex1 x= a+b-c+5 where x =$s0, a=$s1, b=$s2, c=$s3

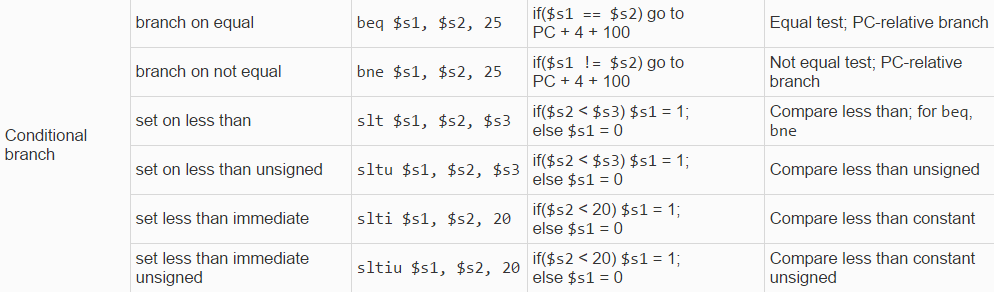
add $t0, $s1, $s2 # t0=a+b

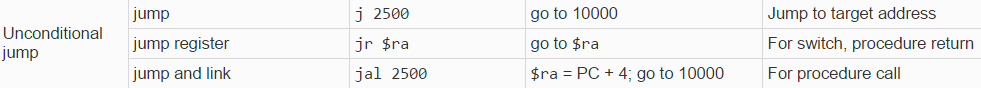
sub $t0, $t0, $s3 # t0=t0-c

addi $t0, $t0, 5 # t0=t0+5









MIPS Registers

* $s0..$s7 Main registers
* $t0..$t9 Temporary register
* $zero The zero register
* $a0..$a3 Registers for arguments
* $v0..$v1 Register for syscalls
* $gp, $fp,
* $sp, Register for top of stack
* $ra Register for return address after subroutines
* $at

Order of operation for arithmetic

To encode an && nest two if statements and if false jump to end to short circuit

Example i<j && j<k where i=, $s0, j=$s1, and k=$s2

slt $t0, $s0, $s1 # i<j

beq $t0, $zero, label to else/end

slt $t0, $s0, $s2 #i<k

beq $t0, $zero, label to else/end

if body

To encode a || nest two if statements and either true jump to if body

Example i<j || j<k where i=, $s0, j=$s1, and k=$s2

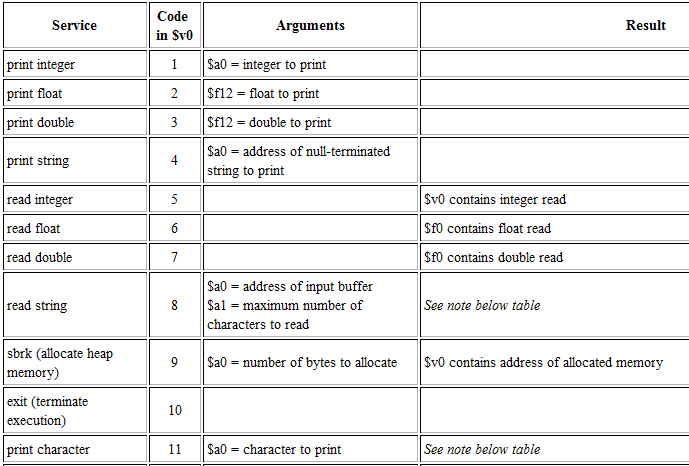
slt $t0, $s0, $s1 # i<j

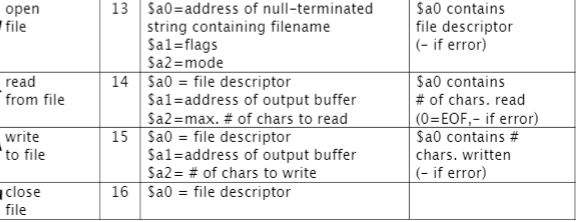
bne $t0, $zero, label to if body

slt $t0, $s0, $s2 #i<k

beq $t0, $zero, label to else/end if body

Syscall list





How to Syscall

Step 1. Load the service number in register $v0.  
Step 2. Load argument values, if any, in $a0, $a1, $a2, or $f12 as specified.  
Step 3. Issue the SYSCALL instruction.  
Step 4. Retrieve return values, if any, from result registers as specified.

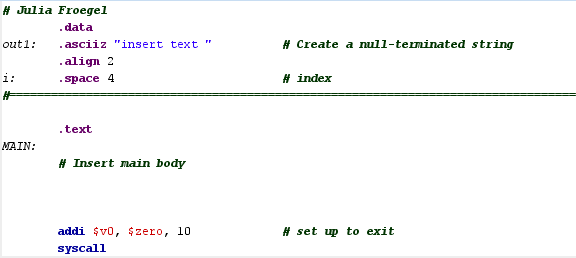
Example

add $a0, $zero, $s0 # put address of hi in $a0 to print

addi $v0, $zero,4 # put 4 in $v0 for printing a string

syscall

Mips program outline



Mips Style Guide

* All named variables must be stored in $s registers. All temporary or intermediate results must be stored in $t registers.
* All lines of code must be commented with a description of what that line is doing. Be sure to reference the names of the variables from the C code.
* Evaluation of complex statements must be done in accordance with the order of operations and then left-to-right within a particular level. For example:
  + C code :

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

* + Corresponding MIPS code:

add $t0, $s1, $s2 # $t0 = g + h

add $t1, $s3, $s4 # $t1 =i+ j

sub $s0, $t0, $t1 # f =(g + h)-(i+ j)

* Proper indentation such that all labels start in the leftmost column, and all statements are aligned at least one column to the right of the colon (:) for the longest label.
* Only use instructions we have talked about in class and use them in the ways we have talked about. Do not use pseudoinstructions.
* Must use syscall to exit program.
* Any variable declared in the .data segment shall be loaded into an $s register for use.
* The base pointer of an array shall not be changed during the execution of the program.
* The first instruction in a method manipulates the stack pointer ($sp).

MET: addi $sp, $sp,-16

* All local variables shall be represented by $s registers and must be saved on the stack upon entry into the method.
* For non-leaf subroutines, the return address and all arguments must be saved on the stack upon entry into the method.
* All arguments will be passed into a method via the $a registers.
* Only one value may be returned and that can only be done via the $v0 register.
* All values saved to the stack shall be restored from the stack before exiting, unless they are not changed or have been restored elsewhere.
* The stack pointer ($sp) shall be restored to its original position before exiting.
* There shall be exactly one exit point for each method (jr $ra).

How to call a function

* To call a function first move all the arguments into the $a registers. There are 4 $a registers.
* Then use the jump and link command with the name of the function

Example:

add $a0, $s0,$zero #sends $s0 to argument

jal FUNCTION #jump to function

How to write a function

* First allocate space on the stack for s registers to be stored by subtracted 4 time the number of s registers that your function will need to use
* Store the s contents of the s registers on the stack so that the s registers can be freely used in your program without losing data.
* Do whatever you want the function to do and store any value that you want to return in the $v0 register.
* Restore the old s registers from the stack and add what you originally subtracted for the stack pointer back to the stack pointer.
* Use the jr $ra to return to where the function was called from

Example:

addi $sp, $sp, -4 # allocate space on the stack to store 1 s variable on the sack

sw $s0, 0($sp) # store what is in $s0

# do method stuff and put return stuff in $v0 register

lw $s0, 0($sp) # load $s0 register again

addi $sp, $sp, 4 #restore stack pointer

jr $ra # return to where method was called

How to call a non-leaf function

* Just like calline
* Move all the arguments into the $a registers. There are 4 $a registers.
* Then use the jump and link command with the name of the function

Example:

add $a0, $s0,$zero #sends $s0 to argument

jal FUNCTION #jump to function

How to write a non-leaf function

* Store the return address of the function on the stack and any variable that will be needed after the inner function is called
* Call inner function
* Restore stack after call

Example:

addi $sp, $sp, -4 # allocate space on the stack to store return address on the sack

sw $ra, 0($sp) # store what is in $sa

jal INNNERFUNCTION

lw $ra, 0($sp) # load $s0 register again

addi $sp, $sp, 4 #restore stack pointer

Exceptions and Interrupts

* The difference between an exception and an interrupt is that that an exception is a type of interrupt that occurs when there is an error either compiling or running the program. Not all interrupts are exceptions though. Interrupts can also occur when the hardware when it gets information for the program or when software wants to ask the OS for something.

Pros and Cons to iterative vs recursive

* A recursive algorithm will be quicker than an iterative algorithm when processing a large amount of data but a recursive algorithm takes up a lot more space since it has to move data to and from the stack to store it which makes it take up a lot more space than iterative and for smaller data samples it can make it have more operations than iterative