CS F342

Computer Architecture

Semester 1 – 2020-2021

Lab Sheet 5

Goals for the Lab: We build up on prior labs and explore i) load/store instructions ii) loops, arrays and string manipulations.

Data Declaration (Recap):

Format for variable name (label) declarations in .data segment:

name-or-label: 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 a number of bytes to be allocated.
- Note: Name (or labels) always followed by colon (:).
- Some Examples

var1: .word 3 *#create a single integer variable with initial value 3*
list: .word 17 5 92 87 41 30 23 55 -72 36 *#an array of 10 integers*
array1: .byte 'a','b' *# 2- element char array- values of a and b (decimal ascii 97, 98)* Array2: .space 40 *# 40 consecutive bytes,* ***not initialized****; could be used as a 40 element char array, or a 10- element integer array; comment should be used to clarify*

Load / Store Instructions

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

LOAD_EXAMPLES:

format:

`lw register_dest, RAM_source` *#copy word (4 bytes) at source RAM location to destination register.*
`lb register_dest, RAM_source` *#copy byte at source RAM location to low order byte of destination register*

STORE_EXAMPLES:

format:

`sw register_source, RAM_destination` *#store word in src register into RAM dest.* `sb register_source, RAM_destination` *#store byte (low order)*in src reg into RAM dest.

sw $t2, ($t0) *#store word in register $t2 into RAM at address contained in $t0*
sw $t2, 12($t0) *#store word in register $t2 into RAM at address ($t0 12)*

swc1 $f0, 4($t4) *# Mem[$t4 + 4] = $f0; Store word(into RAM) from coprocessor 1.* sdc1 $f0, 0($t4) *# Mem[$t4 + 0] = $f0; Mem[$t4 + 4] = $f1 ; Store double(into RAM) from CP 1.*

Exercise 1: A program to take a string from a user and check whether it is a palindrome or not.

```
.data
theStr : .space 6 #declare a space of 6 bytes
isPal : .asciiz "Its is a Palindrome" notPal : .asciiz "Not a Palindrome" newLine : .asciiz "\n"
.text main :
lb $t4, newLine
li $v0, 8 #8=> read string; $a0 is buffer; $a1 is length
la $a0, theStr #load the base address of theStr
li $a1, 6 #load the length of string(max length of string+1 for '\0' )
syscall
add $t2,$a0,$zero #load base address in $t2; find input string length

slen_0 : # loop label to find the last char
lb $t3, ($t2) # load current byte
addi $t2, $t2, 1 # increment for next iteration
beq $t3, $t4, next #if current byte is '\n'
bne $t3,$zero, slen_0 # if current byte isn't '\0', repeat

next : # label to exit the above loop
add $t1,$a0,$zero #load base address
addi $t2,$t2,-2 # -2 because moved beyond '\0' or '\n'; need char before
test_loop :
bge $t1, $t2, is_palin # if lower pointer >= upper pointer, yes
lb $t3, 0($t1) # grab the char at lower ptr lb $t4, 0($t2) # grab the char at upper ptr bne $t3,
$t4, not_palin # if different, it's not addi $t1, $t1, 1 # advance lower ptr
addi $t2, $t2, -1 # advance upper ptr
j test_loop # repeat the loop
is_palin :
li $v0, 4
la $a0, isPal syscall
j exit not_palin :
li $v0, 4
la $a0, notPal syscall
exit :
li $v0,10 syscall
```

Arrays :

Since we have only a small number of registers, it is infeasible to use the registers for long term storage of the array data. Hence, arrays are stored in the Data Segment of a MIPS program. Fundamentally, there are three operations which one can perform on an array:

- Getting the data from an array cell, e.g, x = list[i];
- Storing data into an array cell, e.g. list[i] = x;

• Determining the length of an array, i.e. list.length.

To access the data in the array requires that we know the address of the data and then use the **load word (lw)** or **store word (sw) instructions**. Words (which is how integers are stored) in MIPS take up 32 bits or 4 bytes. Therefore, if we have a declaration such as:

```
list: .word 3, 0, 1, 2, 6, -2, 4, 7, 3, 7
```

the address that is loaded by the instruction la $t3, list is the address of the first '3' in the list. The address of the '0' is 4 greater than that number, and the address of the '6' is 16 greater than that number.

The following snippet of code will place the value of list[6] into the $t4:

```
la $t3, list # put address of list into $t3
li $t2, 6 # put the index into $t2
add $t2, $t2, $t2 # double the index
add $t2, $t2, $t2 # double the index again (now 4x)
add $t1, $t2, $t3 # combine the two components of the address
lw $t4, 0($t1) # get the value from the array cell
```

If we wish to assign to the contents of $t4 to list[6] instead, the last line would simply be:

```
sw $t4, 0($t1) # store the value into the array cell
```

Exercise 2: Write a program to search for a character in a given character array.

```
.data
char: .byte 'u'
vowels: .asciiz "aeiou"
.text
main:
lb $t0, char # load character to look for
li $t1, 0 # it's not found yet
la $s0, vowels # set pointer to vowels[0]
lb $s1, ($s0) # get vowels[0]
srchlp:
beq $s1, $zero, srchdn # check for terminator
seq $t1, $s1, $t0 # compare characters
bgt $t1, $zero, srchdn # check if found
addi $s0, $s0, 1 # no, step to next vowel
lb $s1, ($s0) # load next vowel
b srchlp
srchdn: li $v0, 10 syscall
```

Exercise 3: Write a program to take string of length 5 as input from user and store its reverse string in different array and then print both the strings. Observe the values in

data segment by stepping through the code. Do we need to worry about '\0' termination? Why / why not?

```
la $a0, my_arr2
sb $t2,4($a0) #store byte at $t2 into array cell my_arr2 [4] 5th value
```

Exercise 4 : Write a program to find the maximum and minimum element in an array.

Exercise 5 : Explore disassembly for the new instructions

1. 814c0000
2. c08a0000
3. a08a0000
4. e08a0000
5. e48a0000
6. f48a0000
7. 4604103e

References:

[1] Green Sheet and text book appendix.
[2] http://tfinley.net/notes/cps104/mips.html
[3] https://www.doc.ic.ac.uk/lab/secondyear/spim/node20.html
[4] https://people.cs.pitt.edu/~childers/CS0447/lectures/SlidesLab92Up.pdf