CS 232 – Computer Architecture I
Rose-Hulman Institute of Technology
Computer Science and Software Engineering Department

Lab 2 — MIPS Byte Addressing and Arrays

1 Objectives

Following completion of this lab you should be able to:

- Write loops in MIPS programs.
- Understand the limitations of `beq/bne`, `j`, and `jr`.
- Properly access memory in MIPS programs.
- Dynamically allocate memory in SPIM.
- Understand some of the issues surrounding register allocation.
- Use the SPIM system call for printing strings.

2 General

This and the remaining labs should be done in pairs – only one partner should submit the files, the other should put a *README* in the appropriate directory.

In evaluating your programs the approximate weightings will be: correctness, 50%; efficiency (minimum instructions and minimum registers), 20%; and, documentation, 30%.

The required output format is mandatory.

In writing these and other assembly language programs:

- Solve the problem before coding the solution. Usually this means writing the code in a high-level language or pseudo-code first, then converting it to assembly language.
- **IMPORTANT:**Write down (perhaps in a comment in your code) the purpose that you have in mind for each register that you use. Here's an example of how you might document register use in your code.

```
# $s0 = i
# $s1 = j
# $s2 = n
# $s3 = key
# $s4 = ptr (address of array)
# $t4 = test
# $t0 = temp
```

3 Find the Minimum

Copy the code in file `main.asm` to `min.asm`. Modify `min.asm` to print out the minimum element of the array V (don't print anything but the minimum value, followed by a newline). Remember to include proper documentation in this and all programs you write in these labs. Try to use only t-registers. You may assume that there are exactly 10 elements in the array. Test `min.asm` by changing the contents of the array V.

In SPIM you can check the contents of the array by clicking *Simulator:Display symbol table*, looking in the "Messages" window for the starting address of V (mine's at `0x10010000`), and then look in the "Data Segment" window at the address of V. Can you identify each element of the array?

4 Rotate Right One

Copy the code in file `min.asm` to `rotate1.asm`. Modify `rotate1.asm` to treat V as a circular array and rotate its contents one position to the right. Rotate the array in place. Specifically, $V(i)$ moves to $V(i + 1 \bmod 10)$. After performing the rotation, your program should print the string "The rotated array:" followed by the elements of the rotated array, each preceded by a single space character, e.g.

```
The rotated array: 18 20 56 -90 37 -2 30 10 -66 -4
```

Here is an example of how you might print out the rotated array. Note: \$t1 contains 40.

```
    .data

message:  .asciiz "The rotated array: "
sep:      .asciiz " "
newline:  .asciiz "\n"

    .text

#-----------------------------------------------------------------------
#       Print the rotated array
#-----------------------------------------------------------------------

        la      $a0, message    # store pointer to message
        li      $v0, 4          # use system call to ...
        syscall                 # ... print message
        li      $t0, 0          # initialize the index
loop2:  lw      $a0, V($t0)     # get next element
        li      $v0, 1          # use system call to ...
        syscall                 # ... print the element
        la      $a0, sep        # store pointer to sep
```

```
li      $v0, 4          # use system call to ...
syscall                 # ... print sep
add     $t0, $t0, 4     # increment the index
bne     $t0, $t1, loop2 # check if we've printed all the elements
la      $a0, newline    # store pointer to a new line
li      $v0, 4          # use system call to ...
syscall                 # ... print new line
```

5 Find the Minimum of a Dynamic Array

Copy the code in file `min.asm` to `dmin.asm`. Modify `dmin.asm` to:

1. Prompt the user for the size k of the input array with the string "How many numbers? ", exactly like that. **Assume** that the user always supplies a positive integer that is not bigger than the system can handle.
2. Dynamically allocate an array of size k in memory using the `SPIM` system call *sbrk*[1]. This system call expects an integer i in register $a0 and returns a pointer to a block of i **bytes** of memory in register $v0.

   Here's an example of how you might dynamically allocate memory.

```
    .data

prompt:   .asciiz "How many numbers? "

    .text

#----------------------------------------------------------------------
#       Read the number of elements
#----------------------------------------------------------------------

        la      $a0, prompt     # load address of prompt
        li      $v0, 4          # use system call to ...
        syscall                 # ... print prompt
        li      $v0, 5          # use system call for reading ...
        syscall                 # ... an integer i
        move    $t1, $v0        # Copy integer i into $t1
        sll     $t1, $t1, 2     # multiply i by 4 to get total number of bytes

#----------------------------------------------------------------------
#       Allocate space for the array
#----------------------------------------------------------------------

        move    $a0, $t1        # store number of bytes needed
        li      $v0, 9          # use system call for ...
```

[1] See page A-49 of your text for more information.

```
syscall                 # ... allocating memory
move    $t4, $v0        # save the base address of array
add     $t1, $t1, $t4   # save the end of array
```

Note: the first element of the array can be loaded using `lw $t0, 0($t4)`, the second can be loaded using `lw $t0, 4($t4)`, etc.

3. Prompt the user for k integers (in exactly the format shown in the example below) and store them in the dynamically allocated array.

4. Print the input array.

5. Find the minimum element and print it (in exactly the format shown in the example below).

Be sure to test `dmin.asm` thoroughly. Executing the program should look like this:

```
How many numbers? 5
Input: 12
Input: 24
Input: 13
Input: 8
Input: 22
The input array: 12 24 13 8 22
Minimum element: 8
```

6 Turning It In

When the lab is complete (not later than 05 December 2001), upload all of your files (`min.asm`, `rotate1.asm`, and `dmin.asm` to:

`/Class/cs/cs232/turnin/`<*section*>`/`<*username*>`/lab2/`.