

School of Engineering and Computer Science

## NWEN 242 Computer Organization

### Project 1

The objective of this lab is for you to become familiar with assembly programming. You will be involved in writing several assembly programs. In the meantime, to help you understand the compiler concept, you will also be required to convert two c programs to assembly code. It is worth **10%** of your final grade. The project will be marked out of **100**.

#### Contents

1. Introduction
2. How to prepare for the lab
3. What you have to do during the lab session
  - STEP 1: Get a copy of the sample program file
  - STEP 2: Load the sample program into MARS
  - STEP 3: Understand the MARS display
  - STEP 4: Run the program
  - STEP 5: Modify the program
  - STEP 6: Basic arithmetic
  - STEP 7: Debug a program
  - STEP 8: Recursion (manually convert a c program into assembly)
  - STEP 9: Write a MIPS program from scratch
4. Submission Method
5. Remarks on marking

#### 1. Introduction

This lab project will use the MARS simulator, a Java application that emulates the MIPS RISC. The purpose of this lab is to introduce you to the simulator and to have you practice editing, running and debugging MIPS programs. You can download the MARS assembler from the course Web site.

You should complete this lab during week 2 to week 5 and submit your lab project electronically through the course web site.

#### 2. How to prepare for the lab

Before you attend your lab session you must ensure that you:

- Have an account on the Unix machines. If you do not currently have an account, you should get one online. If there is any problem, you need to contact the School Office ([CO358](#)).
- Know what *assembly language* is and how it's different from high-level programming languages such as C, Java, or Visual Basic.
- Understand the role of a compiler for high-level programming languages.
- Know what *CPU registers* are and know which registers are available on the MIPS CPU.
- Know what it means for assembly language instructions to have different *addressing modes*.

If you need to revise any of these concepts then look at your lecture notes and/or read the corresponding chapters of the course textbook (Chapter 2).

- For detailed information about using the simulator program, read the paper that describes the simulator (available for download from the course Web site).
- For a list of assembly instructions, their syntax and semantics refer to the help section in the simulator.

### 3. What you have to do during the lab session

#### STEP 1: Get a copy of the sample program

Start up the simulator, create a new file and paste in the following program. Save the file, on your account somewhere, as **printstring.asm**.

You start the simulator in your computer with the command "**java -jar Mars.jar**". Please make sure that you have download the simulator first from our course Web site.

```
# NWEN 242 Lab 1
# Program: printstring
# Author:

.data
    str:    .asciiz "Mr Anderson, 19800801"    #replace with your name
and birthday
.text

        la      $a0, str        # load the address of the string
        addi    $t0, $a0, 0     # starting address of array
        #TODO: find length of string, it's zero terminated, so loop until a
zero
        addi    $t1, $a0, 21    # initialize loop counter to array end position
(replace with your str length)
loop:   lb      $a0, 0($t0)     # load a single character
        li      $v0, 11        # specify print character service
        syscall                               # print
        addi    $t0, $t0, 1     # add 1 to the loop index
        blt     $t0, $t1, loop  # continue if not at string length

        li      $v0, 10        # system call for exit
```

```
syscall                                # we are out of here.
```

## STEP 2: Load the sample program into MARS

Once you've saved the file you are ready to run it by using the MARS simulator.

## STEP 3: Understand the MARS display

Once the program is loaded, MARS displays the state of all the registers, the assembly language program or 'text segment', the memory or 'data segment' and a messages window. The image below shows a typical MARS display. If your display looks significantly different then it's possible that there is an error in your program: look in the Message Pane at the bottom of the MARS window for an error message.

A. The **Registers Pane** shows the 32 general-purpose registers 'R0' to 'R31' (which you'll be using most often) and below that the double and single-floating point registers (which you will not use in this lab project).

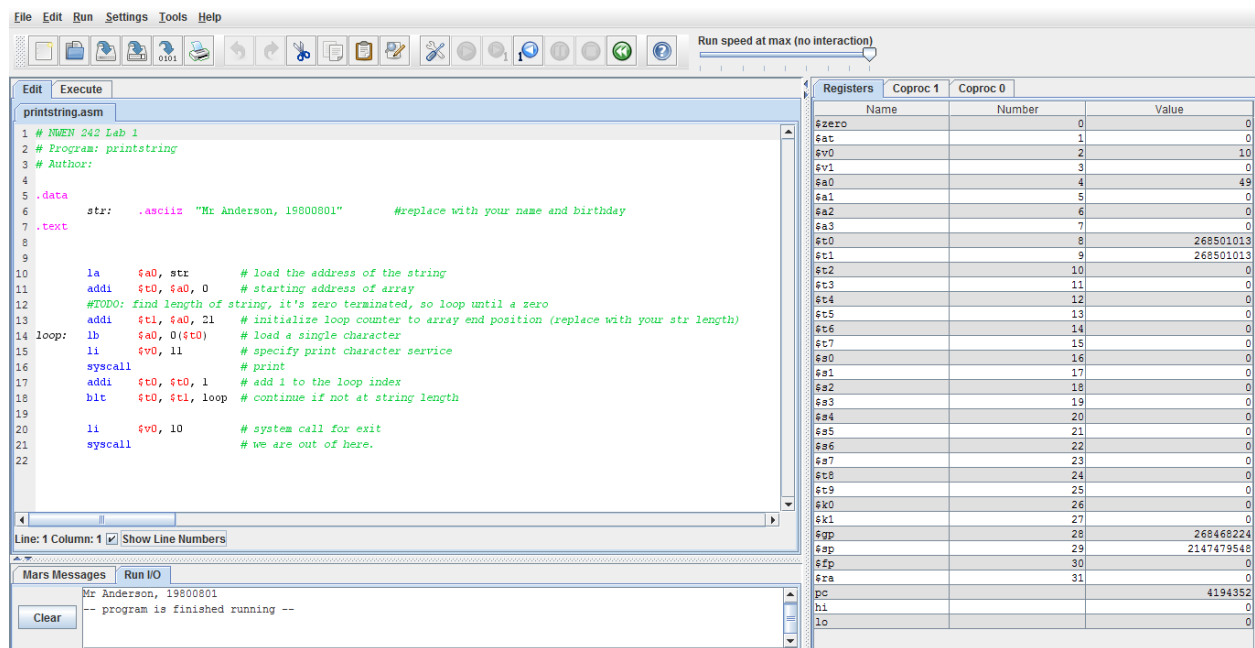



Figure 1: A typical MARS display after loading printstring.asm.

**Note:** Each of the 32 general purpose registers has a **number** and a **name**. When you're writing programs to run on MARS you can use either. The register names are intended to serve as mnemonics. For example, register number 2 is named 'v0' because it is often used to return **values** from functions; register number 4 is named 'a0' because it is often used to pass **arguments** to functions. Table 1 below gives more information on register names.

Name	Register Number	Usage	Preserved on call
\$zero	0	the constant value 0	n.a.
\$at	1	reserved for the assembler	n.a.
\$v0-\$v1	2-3	value for results and expressions	no
\$a0-\$a3	4-7	arguments (procedures/functions)	yes
\$t0-\$t7	8-15	temporaries	no
\$s0-\$s7	16-23	saved	yes
\$t8-\$t9	24-25	more temporaries	no
\$k0-\$k1	26-27	reserved for the operating system	n.a.
\$gp	28	global pointer	yes
\$sp	29	stack pointer	yes
\$fp	30	frame pointer	yes
\$ra	31	return address	yes

Table 1: MIPS Register Conventions.

**B. The Text Segment and Data Segment Panes**, as shown in Figure 2, are available under the Execute tab. They show your program code and program data respectively. *They will both be empty until you click the Assemble button* (i.e. ). The Text Pane has 5 columns showing (from left to right) breakpoint, memory addresses, program instructions in hexadecimal, and two versions of the program instructions in assembly language. The Data Pane shows the data being used by your program, and via the drop down menu can also show the stack and other parts of the memory.

#### **Brief summary of the text segment pane:**

*First column:* by clicking the tick box in the first column, you can set a breakpoint where you want your program to stop and wait for you to debug it.

*Second column:* the second column shows the memory address at which each machine instruction has been loaded. Notice that the addresses increase in steps of 4 because each machine instruction requires 32 bits of storage space.

*Third column:* the third column shows the hexadecimal representation of the machine instructions.

*Fourth column:* the fourth (rightmost) column shows the program instructions pretty much as you typed them into `printstring.asm`, but without the comments. Now, here's the tricky part. When you write programs for MARS you are allowed to use **pseudo-instructions**. Pseudo-instructions are instructions that programmers find useful but which the MIPS CPU cannot execute directly. Pseudo-instruction must be translated (by the assembler part of MARS) into **machine instructions** before they can be executed by the MIPS CPU. Thus, column 3 is the same as column 4 but with the pseudo-instructions converted into machine instructions.

Pseudo-instructions simplify the programmers' job by allowing them to use a richer set of instructions than is actually implemented in hardware. For more information, look up "pseudoinstructions" in the index of the course textbook.

*Note: when you write MIPS code that includes numbers, remember that numbers starting with "0x" (the digit zero followed by a lowercase letter x) are hexadecimal (base 16) numbers. Numbers starting with "\$" (dollar sign) refer to registers, e.g. \$5 means register number R5.*

C. The **Message Pane** at the bottom of the window reports status, prints output and prompts, and also shows error messages to you. If you see any error messages now you may have to go back to STEP 1 and make sure you entered the program as given.

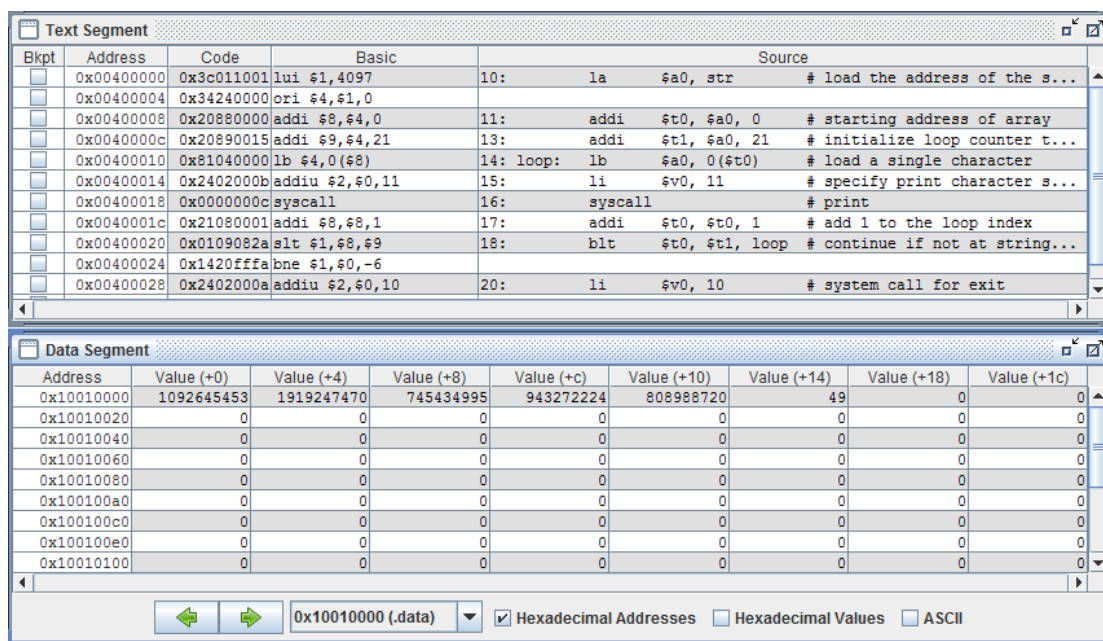




Figure 2: The text and data segment pane.

#### STEP 4: Run the program


Once you have got MARS to accept your program without reporting any errors, run the program with the **run** button . By default, MARS will start running your program from address **0x00400000** (which is what you want) - the initial value in the PC register. The default argument passed to MARS is the name of your program - you do not need to change this.

If all goes well, your name and student ID number will be printed out in the bottom window of the execute pane.

Now run your program again, but this time step through your program one instruction at a time by using the **step** button .

While you step through the program, have you noticed the slider at the top of the MARS GUI? You can adjust it to slow down your program so that you can debug it while you run it. This is very useful if you're not sure where to place a breakpoint.

To familiarize yourself with the MARS simulator, please carry out the following procedures:

1. Step several instructions into your program and then reset the program back to the beginning of the program by using the **Reset** button . Now step your program again to check that the program counter value has changed as expected.
2. Set a breakpoint at the first syscall instruction and then run your program. What is the value of register 4? If you click the 'Run' button again, you will continue to the end of the program.
3. Modify the string 'str' by changing the contents of the data segment where 'str' is stored. Once you've changed some of the contents of 'str', run your program and check that what is printed out reflects the changes you made.

***Note:** all steps before this note will not be marked. All steps in the following, however, will be marked by our tutors. The total marks for each step is indicated at the beginning of these steps. Our tutor will not mark your program during any help desk sessions. Marking will be conducted only after you have submitted all your program code online.*

### **STEP 5: Modify the program [20 marks]**

Make a new copy of printstring.asm, and save it as **printpal.asm**. Modify printpal.asm so that it prints out a palindrome version of your string, i.e. it prints out a string and then prints out the string in reverse. For example:

```
Hello, Mr AndersonnosrednA rM ,olleH
```

When it is the time for our tutor to mark your project submission, he/she may test your program by changing 'str' to some value of his/her choosing and seeing what it prints out.

### **STEP 6: Basic arithmetic [20 marks]**

Copy the following program from the project description, and save it as **power.asm**.

```
#Power

.data

prompt: .asciiz "Enter the number: "
mesg:   .asciiz "When two is raised to that power, the answer is: "
endl:   .asciiz "\n"

.text

main:   la $a0, prompt          # print the prompt
        li $2, 4
```

```

        syscall

        li $2, 5                # read an integer from the user
        syscall                  # into $v0

power:   # your code to compute the power goes here
        # the number is in $v0, you should put the result into $t1

        # t1 = 1
        # t2 = v0
        # while (t2 != 0) {
        #     t2 = t2-1
        #     t1 = t1*2
        # }

output:  la $a0, mesg            # print some text
        li $2, 4
        syscall

        li $2, 1                # print the answer
        move $a0 $t1
        syscall

        la $a0, endl            # print return
        li $2, 4
        syscall

        li $2, 10               # let's get out of here.
        syscall

```

The program is intended to read a number, compute 2 to the power of that number, and print the result. Study the whole program, and try it with MARS: you will see that the program is incomplete, and the code to calculate the power is missing. You need to complete that part of the program. To help you, the program includes a comment giving a C fragment that makes the required computation.

## STEP 7: Debug a program [20 marks]

Copy the following program from the project description, and save it as **integercopy.asm**.

```

.data
source:   .word   3, 1, 4, 1, 5, 9, 0
dest:     .word   0, 0, 0, 0, 0, 0, 0
countmsg: .asciiz " values copied. "

.text
main: add  $s0, $0, $ra          # Save our return address
      la   $a0, source
      la   $a1, dest
loop: lw   $v1, 0($a0)           # read next word from source
      addi $v0, $v0, 1           # increment count words copied
      sw   $v1, 0($a1)           # write to destination
      addi $a0, $a0, 1           # advance pointer to next source
      addi $a1, $a1, 1           # advance pointer to next dest

```

```

        bne    $v1, $zero, loop    # loop if word copied not zero
loopend:
        move   $a0, $v0           # We want to print the count
        li     $v0, 1
        syscall                    # Print it
        la     $a0, countmsg      # We want to print the count message
        li     $v0, 4
        syscall                    # Print it
        li     $a0, 0x0A          # We want to print '\n'
        li     $v0, 11
        syscall                    # Print it
        jr     $s0                # Return from main. We stored $ra in $s0

```

The program is intended to copy integers from memory address \$a0 to memory address \$a1, until it reads a zero value. The zero value should not be copied. The number of integers copied (up to, but not including the zero value), should be stored in \$v0. Debug this program so that it works as intended.

## STEP 8: Recursion [20 marks]

Convert the following C-program into assembler.

```

#include < stdio.h >

void rec(int n) {
    if (n < 1)
        return;
    rec(n/2);
    printf("%d\n", n);
}

int main() {
    rec(64);
    return 0;
}

```

Use the assembler program below as the base. You're not allowed to change the operands of the instructions given here, add labels, or add code after the last line.

```

.data
    endl:    .asciiz "\n"

.text
main:    addi $a0, $zero, 64
        #TODO: jump to F1, making sure we return to the right address
        li   $2, 10          # exit
        syscall

# F1 takes one parameter in $a0
F1:
        #TODO: handle the base case of the recursion
        #TODO: set the right parameters for the recursion
        #TODO: store relevant registers before jump

```



```

    #TODO: perform the recursive call
    #TODO: reset data after return from recursion
    li    $2, 1                # print parameter, it's already in $a0
    syscall
    move  $t0, $a0             # copy parameter
    la    $a0, endl            # print end of line
    li    $2, 4
    syscall
    move  $a0, $t0             # copy parameter back
BaseCase:
    jr    $ra                 # return to the address given in register $ra

```

Where your code goes is indicated by #TODO. You need to be familiar with stack structure in order to finish this step successfully.

## STEP 9: Write a MIPS program from scratch [20 marks]

Write a complete MIPS program that performs the following:

- Ask the user to enter a number by printing the message “Enter a decimal number.”
- Keep the entered number in register \$s0
- Print the message “The number in hex representation is:”
- Check each hexadecimal digit of the number in \$s0 one by one, starting from the most significant digit.
- Convert each hexadecimal digit to the corresponding ASCII character.
- Print each ASCII character.

As you can see from the above, you are asked to build a MIPS program that converts a decimal number entered by the user to its corresponding hexadecimal representation.

## 4. Submission Method

Your project needs to be submitted electronically through the course web site by **Midnight Sunday 16 August**. Please refer to course outline for policies regarding late submissions.

You are only requested to submit your MIPS source code. It is your responsibility to make sure that your code can run nicely on the MARS simulator. You can either package all your source code into a single zip file or submit each program separately.

## 5. Remarks on Marking

Each step, from step 5 to step 9, is fulfilled through a MIPS program. Each MIPS program deserves 20 marks. Below is the spreading of marks for each program.

- **10 Marks:** your MIPS program can be compiled successfully on the MARS simulator and it produces correct output.
- **5 Marks:** your MIPS program is robust. This means that, after changing the initial data for your program to start with, it can still produce desirable output. For example, by changing 'str' to a different value in step 5, your program should still print out a palindrome version of the changed str.
- **5 Marks:** your program shows a good effort and does not violate any restrictions. For example, in step 8, you cannot add any labels to the existing code or add any code after the last line.