HW-2-MIPS-programs

CS 503 Systems Basics

100 Points

Review Material

Sections 9 and 10 of Appendix A of Patterson and Hennessy. Computer Organization and Design. Specifically:
! System Calls: pages 43-45
! Assembler Syntax: pages 47-49

Things You Need to Know

! Instructions
" add rd, rs, rt; rd = rs + rt
" sub rd, rs, rt; rd = rs - rt
" lw rt, addr(rs); rt = Mem[addr + rs]
" sw rt, addr(rs); Mem[addr + rs] = rt
" jr rt; PC = rt [jump register]
! Assembler syntax: Directives, Pseudo-instructions, labels
" .globl (to make a symbol globally known)
" .text (to indicate that what follows is instructions)
" .data (to indicate that what follows is data)
" .asciiz (to create a null-terminated string)
" li (pseudo-instruction to load a constant)
" la (pseudo-instruction to load an address)
" Defining Strings (with .asciiz). Also be aware of ASCII coding of characters.
! System calls to print and input data
" SPIM uses system calls syscall for input and output. (See page A-43 in COD).
" The register $v0 is used to indicate what the system call should do. Inputs to the system call are passed through the register $a0 and outputs are returned through the register $v0. Note that input is entered and output is printed from/to the console window.
- $v0 = 1 is used to print the integer in $a0.
- $v0 = 4 is used to print the string located at the address in $a0.
- $v0 = 5 is used to read an integer from the console. It is returned in $v0.

! Pseudoinstructions

Pseudoinstruction	Translation

mul	mul $t0, $t1, $t2	$t0 = $t1 X $t2
move $rt, $rs	Copy contents of register s to register t R[t] = R[s]	addi $rt, $rs, $zero
li $rs, small	If immed is 16 bits	addi $rs, $zero, small
li $rs, big	If immed is 32 bits	lui $rs, upper (big) ori $rs, $rs, lower (big)
la $rs, big	Load address into to register s R[s] = addr	lui $rs, upper (big) ori $rs, $rs, lower (big)

small means a quantity that can be represented using 16 bits
big means a 32 bit quantity upper(big) is the upper 16 bits of a 32 bit quantity lower(big) is the lower 16 bits of the 32 bit quantity

Examples

li $8, 0x3BF20	lui $8, 0x0003 ori $8, $8, 0xBF20
la $4, 0x1000056c	lui $4, 0x1000 ori $4, $4, 0x056c

What To Do

Part 1: Concepts (50 points)

This part will be graded based on effort because it is more important to demonstrate intuitive understanding than to provide detailed information to the following questions. You can find almost all answers to this section from the publicly available copy of the appendix on spim from Hennessy and Patterson http://pages.cs.wisc.edu/~larus/HP_AppA.pdf (the file is also available on Blackboard, under Course Information).

Q1: Context

a) What is assembly language?

Assembly language is a symbolic representation of the computer's binary encoding for communication (machine code). It is more readable than machine language because it uses symbols instead of bits.

b) What is MIPS?

MIPS is the name given to processors that consisted of a reduced instruction set architecture. This reduced instruction set consisted of a load / store architecture to access memory.

c) What is spim?

SPIM (MIPS spelled backwards) is a simulator software that runs assembly language programs. It is designed for processors that implement the MIPS32 architecture.

d) We mostly write program in high-level languages (e.g, Python, Java, JavaScript, etc) and not in assembly language. Why is the study of assemblers and assembly language an important part of a Computer Science/Engineering curriculum?

Sometimes, assembly language is still needed in very few cases to write programs where speed or size is critical. Additionally, some programs require an exploit of certain hardware, which may not always have an analogue equivalent in higher-level languages. Additionally, assembly is a low level language, so it served as the predecessor to help create and shape the higher level programming languages that we know today.

Q2: Assembler syntax

The following questions are in reference to the add.asm program (which is available in our class portal in Blackboard), which takes in 2 numbers from the user and prints out the sum of them.

a) In the first part of the MIPS assembly code above, we see what is called assembler directives.

```
   .data
x: .word 0
y: .word 0
z: .word 0
nl: .asciiz "\n"
```

1. What does ".data" indicate?

“.data” is a directive that indicates that the section below it will be used only to declare and initialize data.

2. What does ".word" indicate?

“.word” means that the data type assigned to that variable is an arbitrary size.

3. What does ".asciiz" indicate?

   ".asciiz" means that the data type assigned to that variable is a null-terminated string. Here the new line is assigned to n1.

4. What does ".text" indicate?
   ".text" indicates that the next section will be used to code instructions.

5. What is the meaning of each row?

   Each row is a different variable assignment. The first 3 rows, it stores the value 0 for x, y, and z. The last row stores a null terminated string in memory.

b) In the second part, we see the instructions to read from input x and y, which are integers.

```
main:
   li   $v0, 5     # Read x
   syscall
   la   $t0, x
   sw   $v0, 0($t0)
   li   $v0, 5     # Read y
   syscall
   la   $t0, y
   sw   $v0, 0($t0)
```

1. What are the instructions required to read x from input? Explain what happens in each instruction.

```
   li   $v0, 5     # code 5 (read int) loads immediately
into register $v0

   syscall         # invoke the system call to read int
   from input

   la   $t0, x     # load x's address into temporary
   register $t0

   sw   $v0, 0($t0) # Save the value in $v0 (the int
   read from input) to memory location that register
   $t0 points to (variable x)
```

2. In which register is the value of x stored? 0($t0) is where the x is stored
3. The instructions above read x and y as integers. What change(s) must you make to read x and y as strings.

   You have to change li$v0, 5. to li$v0, 8. This system call code ensures the input will be a string instead of an int.

   Also, you must set the buffer size of the register where the string will be held.

c) The below code contains instructions to print out the value of the integer z.

```
li   $v0, 1    # Print z
lw   $a0, 0($t0)
syscall
li   $v0, 4
la   $a0, nl
syscall

la   $t0, x
lw   $t1, 0($t0)
lw   $t2, 4($t0)
sub $t3, $t1, $t2
sw   $t3, 8($t0)
```

1. Identify the instruction(s) required to print z to output/console.

```
li   $v0, 1    # call code 1 (print int) loads immediately
into register $v0

lw   $a0, 0($t0) # load the register where z is stored
into register $a0

syscall # Make the call to print z
```

2. What changes must we make if we are to print out z, but z is a string instead?

   We would need to change the call code from 1 to 4 (li$v0, 1. to li$v0, 4). This will instead print the string instead of the integer.

Part 3: Trace two programs (20 points)

- Download the MIPS programs add.asm and countdown.asm from our class portal in Blackboard and carefully read through them.
- Trace each program by writing the contents of each relevant register after the execution of each instruction. In the case of a loop, show the contents of each relevant register for the first two iterations and the last one.

  See trace.pdf in the zip folder

Part 4: Develop a program (30 points)

Write a MIPS assembly language program that stores the sum of the integers from 0 to 9. A good starting point would be the C code

```
main(){
int i;
int sum;
sum = 0;
i = 0;
while ( i != 10 ){
    sum = sum + i;
    i = i + 1;
     }
}
```

```
        .data
nl:  .asciiz "\n"

        .text
main:
          addi $s1, $0, 0
          add $s0, $0, $0

          addi $t0, $0, 10
next: beq $s0, $t0, done

          add $s1, $s1, $s0
          addi $s0, $s0, 1
          j next
done:
       li $v0, 1
       move $a0, $s1
       syscall

       li $v0, 4
       la $a0, nl
       syscall
```

```
li $v0,10
syscall
```