Purpose: Introduction to assembly language.

In class last Thursday we saw a brief demonstration of the MARS simulation environment for MIPS. We’ll learn a bit more about MIPS in class Tuesday, but we can start creating programs using just a few basic instructions.

See the attached sheet for a quick overview of the MIPS instructions we will be using today.

Details: All programs submitted for this lab must be fully commented; comments should include your name, the lab number, the problem number, the lab date, the Honor Code pledge, and a description of what the program does.

1. [Simple calculation.] Write a MIPS program (modeled after example1.asm) that carries out the computation:

   $$x = a - (2b + c - d);$$

   where a, b, . . ., x are labeled locations in the .data section of your program and the initial values of a, b, c, and d are 10, 20, -30, and -40, respectively. Use as few registers as possible, but don’t declare any additional memory locations and don’t overwrite the values of a, . . ., d. You may not use any MIPS instructions other than the ones given on the last page (so, no multiplication operator, no shift operator). At the end of your computation all memory locations except x should be unchanged; x will contain the result.

2. [More calculation.] Write a MIPS program (modeled after example2.asm) that carries out the computation:

   $$x = (a-b)*3 + (a+b)*6;$$

   Assume a is 10, b is -1. The same guidelines apply as in the previous problem (use only the given instructions, minimum number of registers, don’t overwrite a, b). In addition, try to use as few MIPS instructions as possible.

3. [An introduction to syscalls] You may have noticed that when your program (successfully) completes, you get a message saying “(dropped off bottom)”. We need to tell MIPS to end the program. This is done using the following two commands:

   li $v0,10
   syscall
Write a MIPS program that computes $x = 7*y$ for $y = 13$ using the same guidelines as in the previous problem. In addition, make sure your program terminates correctly.

4. [Optimize.] Look at example1.asm. Can we achieve the same result using one less MIPS instruction? If so, show me the program.

5. [Characters.] Enter the following code in a new MIPS program:

```asm
.data
.align 2
letter: .ascii "A"
.text
lw $t0, letter
li $v0, 10
syscall
```

When you run it, note what value goes into register $t0$.

Now write a MIPS program that has an uppercase ASCII letter saved in a variable named $c$. The program should convert the letter stored in $c$ into an integer between 0 and 25, representing the position of the letter in the alphabet. All other requirements from previous problems still apply. (In particular, you must have comments.)

6. Push your programs, fully commented and including your name and the Honor Code Pledge in the header comments, into the repository you shared with me.

Questions about the lab? Bring them to class on Tuesday morning!
MIPS Basics

Sections of a MIPS Program

The MIPS programs we will write usually have two sections: the .data section (for storing variables and constants) and the .text section (for the actual instructions). Within the .text section we can reserve (uninitialized) memory, measured in bytes, and assign a name to this space. We can also place data in memory and assign it a name. Example (fully-commented version example1.asm in repository):

```
# Example 1: creating space in memory and computing:
#   int a = 42;
#   int b = a + 17;
#
.data
a: .word 42  # 4 bytes of memory containing 42
b: .space 4  # 4 bytes of memory

.text
lw  $t0,a  # load contents of address a into $t0
li  $t1,17 # $t1 = 17
add $t0,$t0,$t1 # $t0 = $t0 + $t1
sw  $t0,b  # store result in address b
```

The spacing into columns is not required by the assembler, but makes reading the program easier:

```
label: instruction operand(s)  # comments
```

The “label:” is optional.

Assembling and Running a MIPS Program

To run this in MIPS, type the command mars example1.asm & in the terminal window. Adjust the sizes of the window components so you have a reasonably-sized edit area. (If you have no file to begin with, just type mars & and select “File/New”.)

No program can be assembled or run until you have saved it. Once that is done, selecting “Run/Assemble” will assemble the program. If there are no errors, “Run/Go” will execute it; “Run/Step” will step through it one instruction at a time. The contents of the registers are shown in the right column; the contents of the “.data” section are shown below the code. See Figure

Registers and Instructions

MIPS uses 32 registers. For today’s lab, we will use only the registers $t0 through $t7 and $v0. Register $v0 will be used (today) solely for making system calls.
Here are quick descriptions of a few of the MIPS instructions (but see note below!):

lw: “Load word.” Form: “lw register, address”. Transfer from memory to register.


sw: “Store word.” Form: “sw register, address”. Transfer from register to memory.

add: “Add.” Form: “add destination, op1, op2”, where destination, op1, and op2 are registers.

addi: “Add immediate.” Form: “addi destination, op1, constant”, where destination and op1 are registers.

sub: “Subtract.” Form: “sub destination, op1, op2”, where destination, op1, and op2 are registers.

subi: “Subtract immediate.” Form: “subi destination, op1, constant”, where destination and op1 are registers.

Several more instructions are covered in the lab 3 exercises.

Note: commands such as “.word”, “.data”, etc. are not machine instructions, but so-called “assembler directives” that do things like set up memory, etc. Furthermore, some of the “instructions” above are actually “pseudoinstructions,” designed for convenience of the programmer, but actually represented by other (real) MIPS instructions. For instance, li is actually a pseudoinstruction that translates into an “addu” (“add upper immediate”) instruction.

What about multiplication and division? They are not simple one-line instructions. We will cover these later, when we discuss arithmetic in MIPS.

See file example2.asm for an example of a more complicated formula.