CS 352 – Prof. Mark **Assignment #2** **Due Feb 15, 2007**

13 problems total

Problems #1-8: (150 points total)

P&H 2.6
P&H 2.30
P&H 2.31
P&H 2.32
P&H 2.37
P&H 2.38
P&H 2.49
P&H 2.51

Problem #9: Variable length instructions (30 points)

Read about the x86 instruction set in section 2.16 of the textbook and answer the following questions:

a) Based on the 6 opcode formats (Fig. 2.45) shown in the chapter, if we wanted to fetch four instructions per cycle, what is the minimum number of bytes that might need to be fetched (you may assume a certain combination of instructions), and what is the maximum?
b) How many different possible combinations of instruction layouts (where the opcodes might start) are there? Note that we do not care about the actual opcodes, just the locations at which they start.

Ungraded: Please think about, and write short paragraphs discussing the following two questions. I'll cover these in class after people have considered them.

a. What are the challenges for quick decoding of any four of these instructions in a given cycle?
b. Suggest how to build the fastest possible decoder for four of these instructions.

Problem #10: Designing instruction formats for a simple ISA (30 points)

Consider the following extremely simple (and nearly useless) instruction set architecture:

```
// Add Register 1 to Register 2 and put the results in Register 3
ADD $R1, $R2, $R3

// Load into Register 1 the contents of memory located at the address
// in (Register 2 * 256) + Register3
LOAD $R1, $R2, $R3

// Store the contents of register 1 into the memory located at the
// address in (Register 2 * 256) + Register3
STORE $R1, $R2, $R3
```

Assume that the registers are 8 bits wide and there are 16 general purpose registers. Memory addresses are 16 bits (which is why we need 2 registers to address memory).

a) If this is all of the instructions in our ISA, what is the minimum number of bits that we can use to encode all of these instructions? Don't forget, each instruction needs an opcode, and to specify all of its arguments

b) Consider adding a new instruction, ADDI, which has format as follows:

```
// Add the constant value specified in the instruction to the contents
// of register 1, and store the results in register 3.
ADDI $R1, immediate, $R3
```

If we keep the number of bits the same as before, then what is th largest immediate that we can express? If we want to be able to express as large an immediate as can be stored in a register, then how many bits wide do the instructions need to be?

c) Now, if we add 5 more instructions to our set, giving us a total of 9 instructions, how many bits wide does the opcode field need to be?

d) In general, it is nice to have the instruction length be roughly byte-aligned. That is, some multiple of 8 bits wide. If we round our instruction length (for all 9 instructions, assuming that none is longer that ADDI) up to the next 8 bits, how wide will our instructions be?

e) Could any of these instructions be made shorter, and still be byte aligned, if we don't require the instructions to all be the same length? Which instructions? What are the tradeoffs involved with choosing variable or fixed length instructions?

Use Pair Programming for Problems #11-13 Below

For the following programming problems, you will complete the problems using the "pair programming" approach. With this approach, you work on the assignment side-by-side with your partner. You may choose your partners, but stick with one partner for all three problems below. One person is the "driver" (i.e. he/she types) while the other person watches and thinks. Switch roles from time to time during the assignment.

The names of both partners must be in a comment at the start of the program. Only one partner should run the "turnin" program for the assignment.

All of your code turned in should be be adequately commented, and should obey the guidelines set forth on the course *homework* page. Points will be deducted if the submissions do not follow the guidelines, so read them carefully. Also, please note that we will not grade illegible assignments.

Problem #11: Assembly language programming #1 (50 points)

The following two programs will require you to use the `XSPIM` simulator to write, debug, test, and run MIPS assembly programs. The XSPIM simulator runs on the departmental Linux machines. Professor Keckler has written an excellent tutorial on XSPIM which we have made available on our class's web page (under homework #2).

Write a MIPS assembly program that contains the string "`A longhorn says Go Texas!`" in simulated memory, and prints it out to the screen in reverse. You may assume that the previous location in memory (before the start of the string) contains a zero. Turn in the file `moo.asm`:

```
csh> turnin --submit impjdi hw2 moo.asm
```

Hints:

1) You will need to use the XSPIM "syscall" directive to ask the operating system to do the printing
2) The "la" pseudo-instruction may be useful.
3) The "lb" instruction may be useful.

Problem #12: Assembly language programming #2 (60 points)

Write a function `choose(x,y)` where the function you implement is:

```
                       x!
choose(x,y) =      ----------
                   y!(x-y)!
```

Remember that ! is the factorial operator, so 4! = 4 x 3 x 2 x 1 = 24. Also remember that 0! = 1. The program should have `x` and `y` be entered by the user as integers, and then output the result.
Please write the output for the following quantities. If anomalies occur, explain why you get the answers you do for each:

```
a) choose(10,2)
b) choose(100,50)
c) choose(1,0)
```

Turn in the file `factorial.asm`:

```
csh> turnin --submit impjdi hw2 factorial.asm
```

Problem #13: MIPS Opcode decoding (120 points)

Write a program in Java to open a file of binary numbers, read the 32-bit numbers one-by-one, and print out the MIPS opcode that each 32-bit number represents. You are responsible for all real MIPS instructions, but not pseudo instructions as defined in Appendix A. For example, an instruction with the number 0x00430820 would correspond to the "add" instruction on the back inside cover of the textbook, and should be printed:

```
ADD
```

Each output result should be capitalized, no space/tab before and after the instruction name, and separated by a new-line. Turn in a file mips_decode.java that can be run by typing:

```
csh> javac mips_decode.java
csh> java mips_decode <input file>
```

Turn in your program using the following command.:

```
csh> turnin --submit impjdi hw2 mips_decode.java
```

Hints:

1) The class web page (under assignments) contains a simple input file and output file. When you download "insts.in", you should get a 12 byte file. If the length is different, you've done something wrong with the download.
2) On a linux machine you can run "od –txC insts.in" to see the contents of the file displayed in hexadecimal. If you have correctly downloaded the file, you should see:

   ```
   % od -txC insts.in
   0000000 64 00 62 28 19 00 09 01 20 00 68 8e
   0000014
   %
   ```

   The file contains instructions for a **little-endian** machine

   Thus, if the first byte in the file is byte #0:
     * The most significant byte (bits 25-32) of the first word is byte 3.
     * The least significant byte (bits 0-7) of the first word is byte 0.
     * The most significant byte (bits 25-32) of the second word is byte 7.
     * The least significant byte (bits 0-7) of the second word is byte 4.
     * etc.

3) The following routines are useful for reading data from a binary file in Java:

   ```
   File f = new File("myfilename.bin");
   FileInputStream in = new FileInputStream(f);
   ... in.read() ...
   ```