CS61C (Patterson)  Proj3  Fall 00

Administrative details

Turn in this project by 11:59pm, Friday October 6. The project can be done in partnerships of two students (working alone is fine). Partners can be from any lab section. Put your names, login and lab section times in disassemble.c.

Project description

Complete a disassembler for MIPS machine language by filling in code in the framework file disassemble.c. A disassembler reads a file containing binary MIPS instructions and prints a listing of the corresponding MIPS assembly instructions. Just as an assembler translates from assembly language to machine language, your disassembler will translate in the opposite direction. This is rather like what spim gives if you ask it to print the instructions in your program.

Your program should handle all the instructions on the back inside cover of COD except for the instruction with opcode 16 (base 10). This includes a number of instructions that you have not yet used in programs. Don’t be too concerned; all you have to do is print the instruction, not understand how it works.

For an instruction not in the table at the back of COD, you should just print the message “unrecognized instruction”. Because the break instruction has different behavior in spim and xspim, just print “break”. For immediates on logical instructions print them in hex (not zero padded), for other immediates print it in decimal. For memory accesses you should print something like

```
lw $8, 248($29)
```

with a signed offset and a base register as in the instruction. Incorporate the program counter in the address you produce for a branch instruction.

For example, given a binary file corresponding to the program:

```
main:      bltz $4, label1
         add $8, $0, $0
         j label2
label1:   ori $8, 1
label2:   ...
```

Your output would look something like:

```
00400000 04800002  bltz $4, 0, 0040000c
00400004  0004020  add $8, $0, $0
00400008  08100004  j 0x00400010
0040000c  35080001  ori $8, $8, 0$x
00400010 ...
```

(Note: spim loads instructions starting at 0x00400000, not at zero! This matters because of branch offsets.)

From your proj3 directory, copy the files with the command:

```
gmake setup -f ~cs61c/proj3/Makefile
```

Disassemble.c is the only source code file for this project and the only one you should modify. Do not modify the Makefile, or add any other files (i.e. header files, other .c files, etc). Compile the project using:

```
gmake
```

There is a binary test file, and the corresponding sample output. To run with the test file and compare to the sample output, type:

```
gmake test
```

You can also make your own test files, give spim the command dump this will create a file called spim.dmp in your working directory that corresponds to whatever source file has just been loaded. (The dump command is not available on the Mac or PC versions of spim.) With this version of spim you don’t have to worry about the endianness of the machine you use because spim now uses network byte order.

A built-in hook for debugging is available. Run disassembler -d and the variable “debugging” will be set to one; you can then conditionally print debugging information. However, be careful to match the sample output exactly when not using the -d flag because this project will be graded by an automatic script. This project uses the technique of data-directed programming that you learned in CS 61A (see chapter 2 of Abelson and Sussman). In C, this is done with function pointers.

Note: The decode table in figure A. 19 on page A-54 is wrong in the 2nd printing of 2nd edition. The 3rd printing is OK. (If your book says lb is 32 decimal, it is correct!) Jalr on page A-65 is wrong, the correct format is jalr rd,rs,rd.