CS/CoE 447 Computer Organization and Assembly Programming
Fall 2017

Programming Project 1
Assigned: October 12. Due: November 12 by 11:59 PM.

1 Description

In this programming project, you will implement a computer game called "Auto Coder." The objective of the game is to enhance the understanding of instruction encoding by filling in the missing fields of MIPS instructions.

Your program will take the input of the opcodes of five (5) instructions, fill in the rest of fields of the instructions, and output the completed instructions.

2 Project Requirements

The project has several requirements, which must be followed. The requirements are divided into 1) Getting the input of five opcodes, 2) filling the missing instruction fields, 3) displaying the completed assembly instructions. The requirements are described below.

The project output must be the same as shown in Section 4; Use the same prompts. This will simplify grading.

The game must run with the MARS simulator and be implemented as a MIPS assembly language program. The programming project is an individual effort. You may ask other students how they are approaching the problem, but you must *not* work together on the coding.

2.1 Game Input

- The program prompts to get five opcodes from the user.
- The allowed opcodes are these: `add`, `addi`, `or`, `ori`, `lw`, `sw`, `j`, `beq`, `bne`.

2.2 Completing Assembly Instructions

- Based on the opcode and its instruction type (R-format, I-format, and J-format), you need to fill in missing registers, and/or immediate values.
- The allowed registers for `rs`, `rt`, and/or `rd` are from $t0 to $t9 (with register index ranging from R#8 to R#15, and from R#24 to R#25), a subset of 32 general purpose registers (GPRs).
  The registers need to be chosen in sequential order. For example, if the first opcode is `add`, we choose two registers $t0 and $t1 as source registers, and $t2 as destination/target register. The completed instruction would be `add $t2,$t0,$t1`.
  To build the connections between consecutive instructions, the target register of a preceding instruction (if any) must be one of the source registers (if any). For example, if the first and the second opcodes are all `add`, the competed instruction would be `add $t2,$t0,$t1` and `add $t4,$t2,$t3`.
  If we need more registers after using $t9, we reuse $t0 to $t9 sequentially.
- Most instructions (i.e., `addi`, `ori`, `lw`, `sw`, `j`, `beq`, `bne`) require immediate values.
  The allowed constant immediates are 10X, where X is initially zero and increments after each use. Following the above example, if the third opcode is `j`, the completed instruction would be `j L100`.

The allowed labels are L10X, where X is initially zero and increments after each use. The constant and label share the same X.

- If an instruction has no destination/target register, its next instruction chooses next GPR registers as source registers. We never have two `j` opcode next to each other.
- Unused fields are set to all 0s. The opcode and funct field values are available from the book's green card.

2.3 Displaying the Completed Assembly Instructions

- The completed instructions must be displayed as correctly formatted assembly language instructions that can be assembled by MARS.
- In the displayed assembly instruction: register operands (i.e., `rs`, `rt`, and/or `rd`) should be shown as symbolic names (e.g., `$t0`), branch and jump immediate operands should be displayed using "L" with decimal values. and load/store immediates should be displayed as positive decimal (base 10) values.
- You may print all label L100 to L104. You will gain extra 10% credits if you only print used labels.

As an example, suppose a branch equal instruction with register operands `$t2` and `$t3` and an offset (immediate) of `0x100` are randomly created. This instruction would be shown in assembly as `beq $t3,$t2,0x100`. The machine code is shown in the next section.

2.4 Displaying the Machine Instruction

- You also need to display the completed instruction in hexidecimal values.
- You can use syscall 34 to print the machine code. Printing without using syscall 34 will gain extra credits.

3 Turning in the Project

3.1 What to Submit

You must submit a compressed file (`.zip`) containing these files:

- `autoCoder.asm` (the Auto Coder game)
- `README.txt` (a help file - see next paragraph)

Put your name and e-mail address in both files at the top. Use `README.txt` to explain the algorithm you implemented for your programming assignment. Your explanation should be detailed enough that the teaching assistant can understand your approach without reading your source code.

If you have known problems/issues (bugs!) with your code (e.g., doesn't select some opcodes, has odd display behavior, etc.), then you should clearly specify the problems in this file.

The explanation and bug list are critical to grading. So, if you're uncertain whether to include something, then err on the side of writing too much and just include it.

Your assembly language program must be properly documented and formatted. Use enough comments to explain your algorithm, implementation decisions, and anything else necessary to make your code easily understandable.

The filename of your submission (the compressed file) should have the format:

`<your username>-pa01.zip`
Here's an example: `adam-pa01.zip`

Files submitted after November 12, 11:59 PM will not be graded. **It is strongly suggested that you submit your file well before the deadline.** That is, if you have a problem during submission at the last minute, it is very unlikely that anyone will respond to e-mail and help with the submission before the deadline.

3.2 Where to Submit

Follow the directions on the courseweb.

4 Example Dialog between Auto Coder Program and the User

```
Welcome to Auto Coder!
The opcode (1-9 :  1=add, 2=addi, 3=or, 4=ori, 5=lw, 6=sw, 7=j, 8=beq, 9=bne)
please enter the 1st opcode:  2
please enter the 2nd opcode:  1
please enter the 3rd opcode:  5
please enter the 4th opcode:  7
please enter the 5th opcode:  9


The completed code is
L100:  addi $t1, $t0, 100
L101:  add $t3, $t1, $t2
L102:  lw $t4, 101($t3)
L103:  j L102
L104:  bne $t6, $t5, L103


The machine code is
   0x21090064
   0x012a5820
   0x8d6c0065
   0x08100002
   0x15cdfffe
```

5 Programming Notes

Think Before Coding: Write pseudo-code for the game prior to thinking about the assembly language. Use procedures to simplify the program.

Displaying Assembly Language: This is probably the hardest part. Here are two reasonable ways to do this; or better yet, come up with your own approach! One possibility is the following. Notice that some instructions can be displayed as assembly language in similar ways; e.g., `add` and `or` are essentially the same, except their `name` is different. Instructions like `beq` and `bne` are similar. Thus, group instructions by the way to print them in assembly language. For each group, write a procedure that prints out the assembly for the group. For example, you might have the procedure `print_add_or(instruction, name)`. This procedure prints an instruction passed in argument `instruction` using the name in the argument `name`.

A more sophisticated approach might add a "format string" to each entry in `optable`. The format string says how to print the assembly instruction. For instance, "`o d,s,t`" might be the format string for `add`. This format string says to print the opcode name (`o`), the destination register (`d`), the source operand (`s`), and the second source operand (`t`). With the format string, you can

write a *single* procedure that takes the opcode table entry for the instruction and the instruction to print as inputs. The procedure uses the format string to print the fields for the instruction. I used this approach in my prototype project; it worked well and helped to keep my code short.

Hint #1: Be careful when printing `lw` and `sw`. The requirements say you must print the offsets as signed decimal values. However, the immediate for the branches and jump must be printed in hexadecimal. You'll need to write a procedure to print a hexadecimal number.

Hint #2: When you print register numbers, the requirements say that you must print the symbolic name. I suggest that you use a table to do this. Each entry in the table corresponds to the symbolic name; e.g., entry 0 is "$zero". With the table, you can "look up" the name for a register number and print the name.