Computer Architecture - CS2323. Autumn 2024

Lab-3 (RISC-V Assembler)

As we just finished discussing instruction encoding and decoding in the current lectures, this assignment will help you assimilate those concepts through practical implementation. You need to write C/C++ code to convert a given RISC-V (assume RV64I variant) assembly code to its equivalent machine code. The input assembly code will be provided as a separate file named **input.s**

Example: If the input file is as follows :
add x3, x4, x7
beq x4, x7, L1
jal x0, L1
sd x5, 12(x6)
lui x9, 0x10000
L1: addi x9, x10, 12

Output should be (**output.hex** file) (consider each term below is a hex-digit). The file should be a human readable text file:
007201b3
00720863
00c0006f
00533623
100004b7
00c50493

We provide staged problem statements and you may complete all or only a few parts (based on your available time and interest).

Part-1 (50% weightage): Support only R-format instructions to their equivalent machine code. The following instructions must be supported:
add, sub, and, or, xor, sll, srl, sra

Example Input (input.s):

sra x9, x11, x18
sub x4, x23, x19
or zero, s9, x18
sll x14, x31, x17
and t2, t5, t1
srl s3, a2, a5

Corresponding Output (output.hex):
4125d4b3
413b8233
012ce033
011f9733
006f73b3

00f659b3

Part-2 (20% weightage): Part-1 + I-format + S-format instructions to be supported. The following instructions must be supported:
add, sub, and, or, xor, sll, srl, sra, addi, andi, ori, xori, slli, srli, srai, ld, lw, lh, lb, lwu, lhu, lbu, sd, sw, sh, sb, jalr

Example Input (input.s):
addi t1, x8, 100
xor t3, t5, t4
ld x8, 8(x9)
sw x3, 16(x5)
srai x29, x3, 4

Corresponding Output (output.hex):
06440313
01df4e33
0084b403
0032a823
4041de93

Part-3 (30% weightage): Part-2 + B-format + J-format + U-format: The following instructions must be supported:
add, sub, and, or, xor, sll, srl, sra, addi, andi, ori, xori, slli, srli, srai, ld, lw, lh, lb, lwu, lhu, lbu, sd, sw, sh, sb, beq, bne, blt, bge, bltu, bgeu, jal, lui

Example Input (input.s):
beq x1, x2, L1
jal x0, L2
L1: xor a5, a3, a7
lui x9, 0x10000
L2: add t1, x8, s10

Corresponding Output (output.hex):
00208463
00c0006f
0116c7b3
100004b7
01a40333

More details:

1. You may assume that the input test program to be used during evaluation would have a maximum of 50 lines within it.
2. The register names in the input program can be specified as x0, x1, … x31 or using their calling convention based aliases like a0, t0, s0, … or a mix of these. e.g., add t2, x3, s1
3. If there is a syntax error in any line, or a missing label referred during branch/jump instructions, your code should gracefully exit and report that error along with line number.
4. If it helps, you may assume that there is only 1 space in between the instruction and the first operand. Also, only 1 space between "," and second operand and so on. Similarly, one colon after the label and then one space.
5. The program starts on the first character in each line
6. There is only one instruction in each line
7. There are no blank lines in the input file
8. The input file will contain only the real RISC-V instructions, not the pseudo instructions.
9. You should verify your code's correctness using various example hex files generated from RIPES or any other RISC-V simulator.
10. Immediate/offset values provided with instructions can be assumed to be in decimal only (with a - sign for negative values).
11. Optional: A line starting with a semicolon (;) can be treated as a comment and omitted. Similarly, anything after a semicolon (;) within a line can be omitted.
12. Optional: You may support translation of specific pseudo instructions

Submission instructions:

The assignment is to be done in groups of up to two. Submit your code and a Makefile which would compile your code and generate an executable named riscv_asm. Prepare a short report on your coding approach and what all you did for testing your code to be correct. Submit a zip file (Lab3_CSYYBTECHZZZZZ_CSYYBTECHXXXXX.zip) containing the following:

1. Source files of your code
2. Makefile - A makefile that would compile your code and generate an executable named riscv_asm
3. README - Containing information about various files in the directory, usage instructions, etc.
4. Test Cases, if you tried out specific input files
5. report.pdf - a short report on your implementation/coding approach and what all you did for testing your code to be correct.

If working in a team, one submission by any of the team members is sufficient.

Demo: We may later conduct a demo where you will show the functioning of your code to the TAs and accordingly be assigned marks.

Note: Submissions are subject to similarity check and hence submit only your own work. Any similarity will be strictly dealt with.

Your submissions will be evaluated on Ubuntu 20.04 machines and hence you must check them to be working on such a setup. If you do not have those machines, please visit the B-524 lab.