ELEC3725 Computer Architecture

Assignment #4 – Building an Assembler

**Instructions:** Submit your source code, the executable (if Blackboard lets you upload it), the output text files, and a PDF file containing the names and Sections of the students in the group, and contents of the output text files pasted in – do not zip the files together.

**You must work in a group** for Assignment 4. You can work in a group of 2 or 3 students – you may work with students in the other Section if you choose to do so. Refer to the Syllabus for Late Submission Policy.

**[40pts]** Create an assembler that converts LEGv8 into decimal (Chapter 2). This is the first part of a larger assignment where your group will create a simulator for the ARM processor (Chapter 4). Your group can choose your language – C, C++, Java, Python, Verilog, etc. The assembler will take a file input containing LEGv8 code and create a file containing the code assembled in decimal – you may choose as a group to leave the opcode as a string, but the remainder of the text file must be in decimal. Refer to the examples on the following pages.

You must implement the following: `ADD, SUB, AND, ORR, EOR, LSL, LSR, ADDI, SUBI, ANDI, ORRI, EORI, SUBS, SUBIS, LDUR, STUR, CBZ, CBNZ, B,` and `B.Cond (EQ, NE, GT, GTE, LT, LTE)`. You do not have to implement anything for functions.

In code with loops and branches, labels must be replaced with a numerical value – a positive value for a forward jump and a negative value for a backward jump – which correctly indicates the numbers of instructions you need to move. Hint: use a two-pass assembler – in the first pass convert to decimal and look at the labels counting the number of instruction that you need to jump, in the second pass replace the label with the numerical value – refer to the examples on the following pages.

**Note:** Your code must for the input files provided, as well as, for <u>any</u> input file containing LEGv8 code. I will test your code on additional testcases that have not been provided.

**Bonus (optional) [10pts]:** output a file with the code assembled in binary. The bit length of the binary must be correct i.e. 5 bits for register values, correct number of bits for opcodes, 32-bit instruction, etc. Refer to the examples on the following pages.

For example, if the input is `code1.txt`

```
ADD X9, X21, X22      //X9 = X21 + X22
SUBI X23, X9, #7      //X23 = X9 – 7 = (X21 + X22) – 7
```

Option 1: `code1_dec.txt` (all decimal)

```
1112 22 0 21 9
836 7 9 23
```

Option 2: `code1_dec.txt` (opcode as a string with all other parts of the instruction in decimal)

```
ADD 22 0 21 9
SUBI 7 9 23
```

For groups attempting the bonus: `code1_bin.txt` (all binary)

```
10001011000 10110 000000 10101 01001
1101000100 000000000111 01001 10111
```

For example, if the input is `code2.txt`

```
Loop: SUBS XZR, X21, X22
      B.GE Exit
      LDUR X9, [X23, #16]
      ADD X10, X9, X11
      ADDI X21, X21, #1
      B Loop
Exit: SUB X10, X9, X11
```

Option 1: `code2_dec.txt` (all decimal)

```
1880 22 0 21 31
84 5 10
1986 16 0 23 9
1112 11 0 9 10
580 1 21 21
5 -5
1624 11 0 9 10
```

Option 2: `code2_dec.txt` (opcode as a string with all other parts of the instruction in decimal)

```
SUBS 22 0 21 31
B.GE 5 10
LDUR 16 0 23 9
ADD 11 0 9 10
ADDI 1 21 21
B -5
SUB 11 0 9 10
```

For groups attempting the bonus: `code2_bin.txt` (all binary)

```
11101011000 10110 000000 10101 11111
01010100 0000000000000000101 01010
11111000010 000010000 00 10111 01001
10001011000 01011 000000 01001 01010
1001000100 000000000001 10101 10101
000101 11111111111111111111111011
11001011000 01011 000000 01001 01010
```

**Two-pass Assembler**

| **LEGv8 Assembly** | **After first pass:**<br>Leave in the labels but keep counters to keep track of # instructions.<br>Label #instructions to jump<br>Exit 5<br>Loop -5 | **After second pass:**<br>Replace labels with the numerical values to get the final code. |
|---|---|---|
| Loop: SUBS XZR, X21, X22<br>B.GE Exit<br>LDUR X9, [X23, #16]<br>ADD X10, X9, X11<br>ADDI X21, X21, #1<br>B Loop<br>Exit: SUB X10, X9, X11 | Loop: SUBS 22 0 21 31<br>B.GE 10 Exit<br>LDUR 16 0 23 9<br>ADD 11 0 9 10<br>ADDI 1 21 21<br>B Loop<br>Exit: SUB 11 0 9 10 | SUBS 22 0 21 31<br>B.GE 5 10<br>LDUR 16 0 23 9<br>ADD 11 0 9 10<br>ADDI 1 21 21<br>B -5<br>SUB 11 0 9 10 |