COLLEGE OF
INNOVATION & TECHNOLOGY

Project #2 - Implementing the YMC instruction set

Your task

You are given the requirements for the YMC instruction set and a high-level programing language HLC (high-level code) below.
Your task is
1. To design an encoding system for the YMC instruction set to convert to machine code
2. Design a compiler that converts the HLC code to YMC assembly code and the corresponding machine code
3. Demonstrate how a processor running YMC changes its state as it executes the machine code. You will demonstrate this by outputting the step-by-step change in the processor's register states as each YMC instruction executes.

HLC description

Here's a sample program in HLC. Its features are **bolded** in the table below:

```
1  unsigned a b c
2  signed x y z
3
4  a = 3
5  b = 15 + a
6  c = b * a / 10
7
8  x = -5
9  y = 13
10
11 if c <= 10
12     x = y + 10
13 else
14     x = y - 20
15
16 while y > 0
17     print y
18     print \n
19     print x
20     print \n
21     y = y - 1
22
```

Line #	Description
1-2	**Variables** are declare at the top of the program. First **unsigned**, then **signed**.
1-2	**All variables are implicitly of type 8-bit integer**. No need to declare that. **Only 3 unsigned** variables are allowed: **a, b, c.** **Only 3 signed** variables are allowed: **x, y, z**.
4-9	One assignment is performed per line. Don't write *b = 0, x = 4* etc
5, 6, 12, 14	The **four arithmetic operations** allowed are + - * /
6	A single line can have max of two arithmetic operations. Operators are evaluated left to right. For example, Line 6 means *c = (b * a) / 10*
6	**Only the integer part of an operation is preserved**. So the output of Line 6 right-side is (18*3)/10 = 5
11, 16	Two types of conditional structures are allowed: **if-else** and **while loop.** No nested structures.
11-21	**Indentation** indicates whether a line is inside an if-else or a while loop
11, 16	**Only a single relational operator** is allowed in *if* and *while* statements. So *if x > 10 && y < 5* is not allowed.
17-21	The print command is the only mechanism to output a value. There is no user input command, i.e. a *cin* equivalent

YMC assembly - Description

The YMC instruction set is a hybrid of the LMC and Y86 instruction sets. Its features are:

1. It supports 8-bit signed and unsigned integer operations.
2. It has 4 general purpose registers `eax`, `ebx`, `ecx`, `edx`. These are 8-bit wide.
3. It has four flags: `CF`, `OF`, `SF`, `ZF`.
   a. These are computed when an arithmetic or compare operation is performed.
4. The operations it must support are:
   a. Two mov operations:
      i. `mov ecx, 3` // ecx = 3
      ii. `mov ecx, eax` // ecx = eax
   b. Two-operand arithmetic operations: `add, sub, mult, div`
      i. `add ecx, eax, ebx` //ecx = eax + ebx
   c. Three-operand arithmetic operations
      i. These are pairs of arithmetic operations using the above two-operand ops. For example,
         `addsub eax, eax, ebx, ecx` //eax = eax + ebx - ecx
         `subadd edx, eax, ebx, ecx` //edx = eax - ebx + ecx
         `multdiv eax, eax, ebx, ecx` //eax = (eax * ebx) / ecx
         …
         All arithmetic operations use the first mentioned register as the destination, and the rest as operands.
   d. Relational operations `>, >=, <, <=, !=, ==`
   e. Compare operation
      i. `cmp eax, ecx`
   f. Jump operations
      i. Conditional: `jg, jge, jl, jle, jne, je`
      ii. Unconditional: `jmp`
   g. loop

Assume a 1kB byte-addressable memory for storing instructions and data

Your implementation's output

Your final software will take a HLC program as input, decompose it into YMC assembly and machine code, and output a CSV file. This file will show the processor's state after executing each line of the YMC. It is formatted as:

HLC instruction	YMC Address	YMC assembly	YMC encoding	Modified registers (if any, after execution)	Modified flags (if any, after execution)
while y > 0	XXXX	cmp eax, 0 Jg xxx	AB CD 20 XXXX	----	SF = , OF, ZF, CF

There are multiple stages that you have to execute successfully to get the complete version. Your project's success will be based on how well you executed each project stage, as well as whether you have the complete software.

Your deliverables

1. Implementation code - hosted on GitHub

2. A report containing:
    a. The link to your Github page
    b. Your encoding system for the YMC assembly
    c. Flowchart showing the logic to convert a HLC while loop to YMC assembly
    d. Flowchart showing the logic to convert other HLC instructions to YMC assembly
    e. A summary of
        i. What you implemented and what you didn't
        ii. Where the potential problems are.
3. Your demo in class of your implementation - what works and what doesn't work. Details mentioned in a section below called 'Your demo'.
4. Poster that you will present on CIT Project day

Deadlines

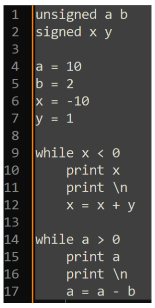
Deadline	Column1	Task	Notes
Every Tue		Rotating manager	The team manager will change every phase. Use the link to the left to learn what your responsibilities as a manager are.
Every Thur		Asset-based assignments	Submit on Canvas. Each week's manager will evaluate the currently assigned tasks, the tasks for the upcoming deadline, and assign them to the team members based on their strengths and areas of improvement.
		Asset maps	In class. Bring your old asset maps to class, or click link to the left. You will modify/expand your map for this project. Some skills to think about: *Version control, software project experience, your programming experience, people skills, managing and assigning tasks, which languages you are comfortable with, ability to read a textbook and explain to others, comfort with bits and bytes and encoding instructions*
		Team contract	Submit on Canvas. Partially in class on Oct 5. The rest will require you to read a paper, meet with your team and finish it.
	Draft	YMC assembly encoding	Submit on Canvas. Initial draft of your team's encoding. A table showing your team's mapping from assembly to machine encoding.
	Final	YMC assembly encoding	Submit on Canvas. Final version that you will use.
	Draft	Flowchart, HLC to YMC assembly	Submit on Canvas. Program logic to convert HLC to YMC. ~50% complete.
	Final	Flowchart, HLC to YMC assembly	Submit on Canvas.
	Test inputs & expected outputs	HLC simple instructions to YMC assembly	
	Flowchart	HLC simple instructions to YMC assembly	
		Draft - Poster	Submit on Canvas. Click here for template. Keep the top and bottom ribbon unchanged. For the text in the middle, you need not follow the suggested format. In fact, this is where your creativity and ability to convey your project clearly comes to the fore.
		Team processing	Submit revised contract on Canvas. Click the link to the left. In the class, your team will discuss current team strengths (& issues) and develop a plan to strengthen (& rectify) them.
		Final - Poster	Submit on Canvas. All will be sent to the CIT community via the CIT newsletter. The best ones will feature on the CIT posterboards and TVs.
	Final	1. Recording of software demo 2. Written report	Submission links on Canvas. Demo details are provided below. Report contents are provided in the section 'Your deliverables'
		Individual reflections	Submit on Canvas. Each of you will individually complete this assignment.

Breakdown of each team's tasks (suggested)

1. Create a git project shared between your team members. As you go about the below tasks, create sub-folders that store different aspects of the design and documents.
2. Using the information in the section *YMC assembly - Description*, create all possible YMC assembly instructions that the architecture will have.
3. Group them according to their type. Create a table mapping a YMC assembly instruction to its encoding in hex format.
    a. E.g. how will you encode `b = a + 15` versus `b = a + c`, or `x = 3` versus `x = y` ?
    b. Refer to Chapter 4.1 in your textbook on how to encode instructions with an immediate value. Figure 4.2 and the related text is a good starting point.
4. Implement program logic to
    a. Translate a given YMC assembly instruction to YMC machine code
    b. Translate a HLC line (that is not a loop) to its YMC assembly instruction while keeping a variable's signed and unsigned status in mind.
    c. Translate a loop control structure to YMC assembly. Things to think about:
        i. In a loop, you will encounter conditional and unconditional jumps. How will you encode the jump address when you havent yet loaded the program into the memory and don't know that instruction's address?
        ii. Maybe the function that 'loads' the program into the memory 'adjusts' the code before loading it? Maybe some other way?
    d. Manage register usage. Notice that your program can have up to 3 signed and 3 unsigned variables. But you only have 4 registers! How will you handle this resource limitation?
    e. Calculate flags when an arithmetic, relational or compare operation is executed.
    f. Calculate the registers that change when an operation is performed, given the initial state of the registers.
    g. Generate the memory address of each YMC instruction when it is loaded into the memory.
    h. Generate the CSV file that for an HLC program to
5. Test mechanisms to verify your code at various stages
    a. For each program logic mentioned above, you should conduct unit tests to determine if its working correctly. Some example tests are:
    b. Given YMC assembly instruction, your code translates to the right YMC encoding
    c. Given YMC assembly instruction and an initial state of flags and registers, it correctly calculates the next state of the flags and registers.
    d. Given a YMC assembly program, it loads the program correctly into the memory while accounting for jump addresses.
    e. Given a YMC assembly program and an initial state of flags and registers, it correctly calculates the state of the flags and registers after every instruction.
    f. Given a single HLC instruction, your code translates to the correct YMC assembly instructions. Check this for various HLC instructions.
    g. Given a HLC program, your code translates to a set of YMC assembly instructions - for single line instructions, then if-else, then for while loop.
    h. Given a HLC program, your code generates the output in the form shown in the *Implementation output* section above.

Test code to use for your final testing

Use the following test HLC code to showcase that your solution can generate the right assembly code. The test code covers the absolute basic features required of your solution. Your solution should generate the CSV file mentioned in the 'Implementation Output' above.



```
unsigned a b
signed x y

a = 10
b = 2
x = -10
y = 1

while x < 0
    print x
    print \n
    x = x + y

while a > 0
    print a
    print \n
    a = a - b
```

Bonus points if you can showcase these other features using other test HLC code:

1. The test code you wrote to verify your software behavior
2. Your code handles overflow and wrap-around for unsigned and signed variables. (e.g. unsigned will change from 255 to 0 if it's increased by 1, and it will change from 0 to 255 if it's decreased by 1)
3. Your code can handle the usage of more than 4 variables in the HLC program - that is, it can handle swapping values in and out of the registers.

Your demo

Please record a demo of your solution. It should cover the following topics and should be no more than 15 mins long. The submission deadline is in the Deadlines table above:

1. Your encoding system
2. List of features that you got to work
3. List of features that don't work
4. A live demo using the above test HLC code (or using a simpler HLC code if you havent got all features running)