Quiz 4

EECS665 - Compiler Construction
2019, Fall

Name: ____________________ Student ID: ______________

Do Not Open Until Instructed!

Before the Quiz starts:

- Read all of the instructions on this page
- Write your name and student ID on this page
- Retrieve your page of notes and writing materials
- Put all other materials away and silence your devices

After the Quiz starts:

- Write your student ID (**not** your name) on all subsequent pages
- If you feel a question is wrong or impossible, notify course staff.
- Announcements / corrections will appear on the projector
- Turn in all your related paper when finished, including:
  - your notes page
  - the provided quiz itself
  - provided reference pages
  - provided scratch paper
- You may leave when done (no new material will be presented).
- Work quickly, move on if you are stuck.

Feel free to draw **Drew**
in the box below to pass the time

Total Questions: 5
Time Limit: 35 minutes
Total Pages:

- 6 pages total

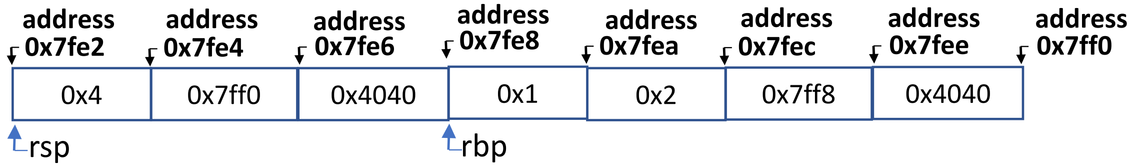
Score: ______ / 50 pts

Question 1 (10 Points)

Imagine a compiler for our language where:

- All arguments are passed on the stack
- The target is a 16-bit architecture (the only significance being that addresses and ints are 2-bytes - you may use the X64 instructions and registers as if they worked on a 2-byte architecture).

Consider this (partial) memory snapshot of the stack for a running program whose executable had been compiled by this compiler



Write a snippet of source code that could generate a program inducing this snapshot. You may include code that is not part of the snapshot, but be sure to indicate what part of your program corresponds to the memory snapshot here.

Question 2 (10 Points)

Student ID: ________________

Part I (5 points)

Apply a peephole optimization to the following code. Assume that the below snippet occurs within a single basic block:

```
I1: subq $8, %rsp
I2: movq $8, (%rsp)
I3: movq (%rsp), %rax
I4: addq $8, %rsp
```

Part II (5 points)

The below code has a data hazard. Optimize the code such that the hazard is lessened / avoided (Assume that the below snippet occurcs within a single basic block):

```
addq %rax, %rax
subq $12, %rax
addq %rbx, %rbx
subq $12, %rbx
```

Question 3 (5 Points)

Student ID: ____________________

The compiler toolchain that we discussed in class corresponds to 4 components. Explain what each of these components do:

- The compiler:

- The assembler:

- The linker:

- The loader:

Question 4 (10 Points)

```
 1. int haw(int a){
 2.    int local;
 3.    local = 2;
 4.    a = 3;
 5. }
 6. int hem(){
 7.    int a;
 8.    int b;
 9.    int c;
10.    b = 1;
11.    haw(b);
12. }
```

Assume pass-by-reference for parameter passing in the above code. Write out the X64 code corresponding to the call to `haw` at line 11. Assume that all parameters are pushed onto the stack such that the final argument is pushed first. Also, make sure you include code to deallocated the pushed arguments from the stack after the call instruction completes (assume no optimization).

QUESTION 5 (10 POINTS)

Student ID: ____________________

PART I

Write out a snippet of X64 code for the *function prologue* for the function `hem`.

PART II

Write out a snippet of X64 code for the *function epilogue* for the function `hem`. Your function epilogue should correspond to the prologue you wrote in Part I.

X64 Reference

- movq <opd$_1$> <opd$_2$>

  Copy the value of opd$_1$ into opd$_2$

- addq <opd$_1$> <opd$_2$>

  Put the result of $\text{opd}_2 + \text{opd}_1$ into opd$_2$

- subq <opd$_1$> <opd$_2$>

  Put the result of $\text{opd}_2 - \text{opd}_1$ into opd$_2$

- callq <lbl>

  Push the address of the next instruction onto the stack and move %rip (the instruction pointer) to the address <lbl>

- retq <lbl>

  Pop the stack and put the result into %rip

- cmpq <opd$_1$> <opd$_2$>

  Set rflags according to <opd$_2$> - <opd$_1$>

- je <lbl>

  jump to <lbl> if rflags indicates a $=$ relation on prior operands

- jne <lbl>

  jump to <lbl> if rflags indicates a $\neq$ relation on prior operands

- jge <lbl>

  jump to <lbl> if rflags indicates a $\geq$ relation on prior operands

- jl <lbl>

  jump to <lbl> if rflags indicates a $<$ relation on prior operands

- jle <lbl>

  jump to <lbl> if rflags indicates a $\leq$ relation on prior operands