# CMPT 295

Name: Junchen Li

Student ID: 301385486

Date: 2020/2/9

# Assignment 4

Objectives:

- Range of C data types
- Hand tracing assembly code
- Translating assembly code into C code
- Writing assembly code

---

Submission: Assignment 4 is a little unusual

- Do Assigment 4 (all 4 questions) as part of your study for Midterm 1.
- You do not have to submit your solutions to Question 1 and to Question 2.
    - Solutions to Question 1 and Question 2 are posted.
- You do have to submit your solutions to Question 3 and Question 4.
    - Solutions to Question 3 and Question 4 will be posted after their due date.
- Submit your document called **Assignment_4.pdf**, which must include your answers to Question 3 and Question 4.
    - Add your full name and student number at the top of the first page of your document **Assignment_4.pdf**.
- When creating your assignment, first include the question itself and its number then include your answer, keeping the questions in its original numerical order.
- **If you hand-write your answers (as opposed to using a computer application to write them):** When putting your assignment together, do not take photos (no .jpg) of your assignment sheets! Scan them instead! Better quality -> easier to read -> easier to mark!
- Submit your assignment electronically on CourSys.

---

Due:

- Submit your document called **Assignment_4.pdf**, which must include your answers to Question 3 and Question 4 on Thursday, Feb. 13 at 3pm
- Late assignments will receive a grade of 0, but they will be marked in order to provide the student with feedback.

---

Marking scheme:

This assignment will be marked as follows:

- Questions 1 and 2 are not marked (we do not need to submit them). They are to bo done only as part of your study for Midterm 1.
- Questions 3 and 4 will be marked for correctness (we need to submit them). They are also to bo done as part of your study for Midterm 1.

The amount of marks for each question is indicated as part of the question.

A solution to Question 3 and Question 4 will be posted after the due date.

**[10 marks] Also for study purposes - Translating assembly code into C code**

**Read the entire question before answering it!**

**Consider the following assembly code:**

```
# long func(long x, int n)
# x in %rdi, n in %esi, result in %rax
func:
     movl  %esi, %ecx
     movl  $1,   %edx
     movl  $0,   %eax
     jmp   cond
loop:
     movq  %rdi, %r8
     andq  %rdx, %r8
     orq   %r8,  %rax
     salq  %cl,  %rdx  # shift left rdx by content of cl
cond:
     testq %rdx, %rdx  # rdx <- rdx & rdx
     jne   loop        # jump if not zero (when rdx & rdx != 0)
                       # fall thru to ret (when rdx & rdx == 0
     ret
```

**The preceding code was generated by compiling C code that had the following overall form:**

```
long func(long x, int n) {
     long result = ____1_____;
     long mask;
```

```
    for (mask = _mask&mask_ ;mask _!=0___ ;mask =_mask<<c1_ )
        result |= _______ mask&x _____________
    return result;
}
```

**Your task is to fill in the missing parts of the C function above to get a program equivalent (note: it may not be exactly the same) to the generated assembly code displayed above it. You will find it helpful to examine the assembly code before, during, and after the loop to form a consistent mapping between the registers and the C function variables.**

**You may also find the following questions helpful in figure out the assembly code (you do not have to provide an answer for them as their answer will be reflected in the C function you are asked to complete):**

a. **Which registers hold program values `x`, `n`, `result`, and `mask`?**
b. **What is the initial value of `result` and of `mask`?**
c. **What is the test condition for `mask`?**
d. **How is `mask` updated?**
e. **How is `result` updated?**

1. **[10 marks] Also for study purposes - Writing x86-64 assembly code**

**Download `Assn4_Q4_Files.zip`, in which you will find a `makefile`, `main.c` and an incomplete `calculator.s`. The latter contains four functions implementing arithmetic and logical operations in assembly code.**

**Your task is to complete the implementation of the three incomplete functions, namely, `plus`, `minus` and `mul`. In doing so, you must satisfy the requirements found in each of the functions of `calculator.s`. You must also satisfy the requirements below.**

**You will also need to figure out what the function `lt` does and add its description in the indicated place in `calculator.s`.**

**<u>Requirements:</u>**

- **Your assembly program must follow the following standard:**
  - **Your code must be commented such that others (i.e., TA's) can read your code and understand what each instruction does. Also, describe the algorithm you used to perform the multiplication in a comment at the top of `mul` function.**
  - **Your code must compile (using gcc) and execute on the computers in CSIL (using the Linux platform).**

  - **Your program file must contain a header comment block including the filename, the purpose/description of your program, your name and the date.**

- **For all of the four functions, the register %edi will contain the argument x and the register %esi will contain the argument y. The register %eax will carry the return value.**
- **You may use registers %rax, %rcx, %rdx, %rsi, %rdi, %r8, %r9, %r10 and %r11 as temporary registers.**
- **You must not modify the values of registers %rbx, %rbp, %rsp, %r12, %r13, %r14 and %r15. We shall soon see why.**
- **You must not modify the values of any memory locations.**
- **You cannot modify the given makefile.**

**Hint for the implementation of the `mul` function:**

**Long ago, computers were restricted in their arithmetic prowess and were only able to perform additions and subtractions. Multiplications and divisions needed to be implemented by the programmer using the arithmetic operations available.**

```
#name: calculator.s
#Junchen Li
#2020.2.13
# add a header comment block
    .globl  lt
    .globl  plus
    .globl  minus
    .globl  mul
# x in edi, y in esi  return value in eax

lt: # add a description to this function
    xorl    %eax, %eax
    cmpl    %esi, %edi
    setl    %al
    ret

plus:  # performs integer addition
# Requirement:
# - you cannot use add* instruction
  andl   %edi, %esi     //add then together
  movl   %esi, %eax     //move the result to the return value
```

```
   ret

minus: # performs integer subtraction

# Requirement:
# - you cannot use sub* instruction
   negl    %esi, %edi      // change to negative value
   andl    %edi, %esi      // and add them together as same as the minus
   movl    %esi, %eax
   ret

mul: # performs integer multiplication - when both operands are non-negative!
# You can assume that both operands are non-negative.
# Requirements:
# - you cannot use imul* instruction
#   (or any kind of instruction that multiplies such as mul)
# - you must use a loop

# int mul (int x, int y)
# x in edi, y in esi  return value in eax
#we write a loop that let condition part less that y and there are y of x so add all x together,
we can do the same as mul x*y  4*3=4+4+4
func :
   movl $0, %eax   // assignment the value zero to the return value
   xorq %rax, %rax   // assignment the value zero to the resiger x^x=0
loop:
   cmpq %rax, %esi    //compare the count value (eg. i) and y if it is less or equal
   jg  endloop       // if it is bigger than y then endloop and return!
   addl %edi, %eax     // add the value of x to the return value
   incq %rax       // and the count value i should be increase by one
   jmp loop       // when condition is correct then jump the loop and do it again
endloop:
   ret

# algorithm:
#return^return
#compare between the x and y, y-x=0
#set low byte of %esi to one or zero
# and return the value
```