Release Date: 04/20/2020 Due Date: 05/01/2020 – 6:30pm

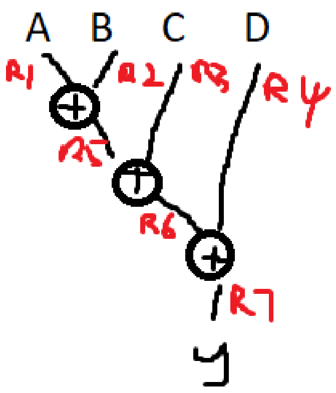
CSE 141 – Homework 2 (11 pts): Dataflow Graphs and Computing Stack Essentials

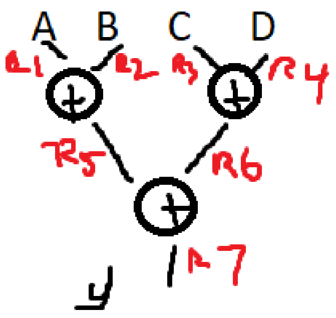
1. **(6 pts)** Consider the following expression:

$$y = A + B + C + D$$

We will do a series of exercise problems to understand the steps involved in compiling this instruction to an important class of intermediate representation for the compiler: the dataflow graph (DFG). DFGs effectively capture the operations of the computation and the data dependencies among the computations. Modern day compilers such as LLVM converts source program to "virtual instruction set architecture (V-ISA)" instructions in static single assignment (SSA) form, which is effectively captured by the DFG, before generating the target machine instructions.

a) **(1 pt)** There are two different DFGs that can be drawn for this expression. Draw the two DFGs and make sure to write the operations on the nodes and registers (i.e. namespace) on the edges for each DFG.





b) **(1 pt)** Recall that DFGs capture the notion of virtual dataflow machines. These machines do not physically exist, but provide a virtual instruction set architecture, and the idea is that we can "execute" instructions on this virtual machine (Note that Java Virtual Machine, for example, is another type of virtual machine that is stack-based). Suppose we want to execute the two DFGs you generated in the previous questions on this virtual dataflow machine. How many cycles does it take to execute each DFG? Note that each addition takes one cycle each.

3 for the one on the left, 2 for the one on the right

c) **(1 pt)** For each DFG, write an instruction sequence in terms of the registers and operations you used in the previous question.

LHS	RHS
LD R1 0(A)	LD R1 0(A)
LD R2 0(B)	LD R2 0(B)
LD R3 0(C)	LD R3 0(C)
LD R4 0(D)	LD R4 0(D)
ADD R5 <- R1, R2	ADD R5 <- R1, R2
ADD R6 <- R5, R3	ADD R6 <- R3, R4
ADD R7 <- R6, R4	ADD R7 <- R5, R6
STR (Y) <- R7	STR (Y) <- R7

d) **(1 pt)** Convert each instruction sequence to static single assignment (SSA) form. Let us use the alpha notation (α) to name the registers, as we did in class. Assume you have infinite number of registers.

LHS	RHS
LD α1 0(A)	LD α1 0(A)
LD α2 0(B)	LD α2 0(B)
LD α3 0(C)	LD α3 0(C)
LD α4 0(D)	LD α4 0(D)
ADD α5 <- α1, α2	ADD α5 <- α1, α2
ADD α6 <- α5, α3	ADD α6 <- α3, α4
ADD α7 <- α6, α4	ADD α7 <- α5, α6
STR (Y) <- α7	STR (Y) <- α7

e) **(1 pt)** Suppose we are generating instructions for a target architecture. Let us assume our target architecture has only one unit that can perform arithmetic operations. In this case, which one of the two DFGs you have previously generated should we consider? Explain why this particular DFG is suits our needs and why the other one does not.

The one on the right, as there are fewer cycles and will be more efficient per cycle.

f) **(1 pt)** Now suppose we want to generate instructions for stack machine instead of dataflow machine. Write the instruction sequence for a stack machine such as Java Virtual Machine that represents the expression given in the problem.

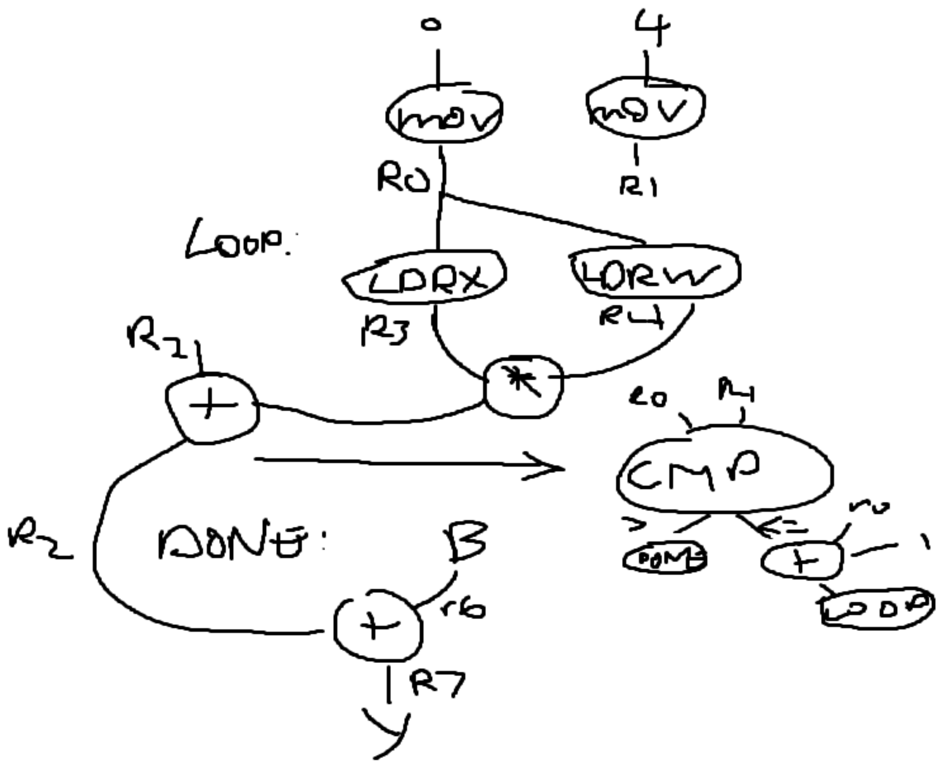
PUSH R1 0(A)
PUSH R2 0(B)
PUSH R3 0(C)
PUSH R4 0(D)
+ R1 R2 -> R5
+ R3 R4 -> R6
+ R5 R6 -> R7
! R7 -> Y // Store

2. **(3 pts)** Consider the following expression:

$$\sum_{i=0}^{4}(x_i * \ w_i) + b$$

(Note: this computation is basic operation in artificial neural networks.)

a) **(1 pt)** Draw a DFG for this expression. Write the operations on the nodes and registers (i.e. namespace) on the edges for each DFG.



b) **(1 pt)** For the DFG you have drawn, write an instruction sequence in terms of the registers and operations you used in the previous question.

MOV r0, 0(0)
MOV r1, 0(4)
LOOP:
LDRX r3, r0 // Load X_i
LDRW r4, r0 // LOAD W_i
MUL r5 <- r3, r4
ADD r2 <- r2, r5
CMP r0, r1
BGE DONE
ADD r0 <- r0, 1
B LOOP
DONE:
LD r6, 0(b)
ADD <- r7, r2, r6
STR (Y) <- r7

c) **(1 pt)** Convert each instruction sequence to SSA form. Use the alpha notation (α) to name the registers. Assume you have infinite number of registers.

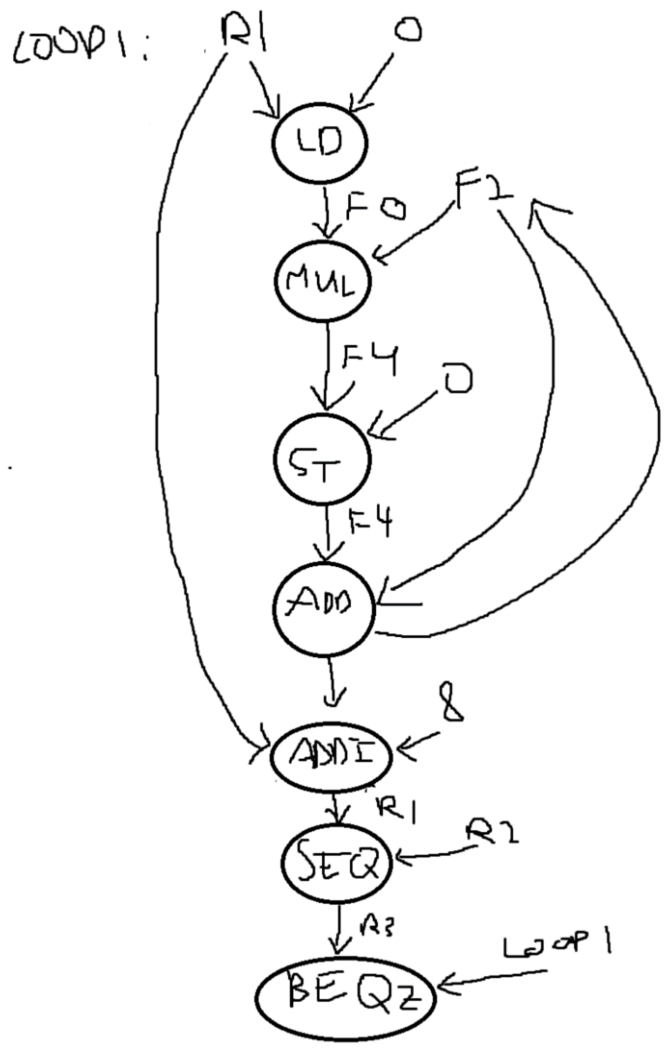
MOV α0, 0(1)
MOV α1, 0(4)
LOOP:
LDRX α3, α0 // Load X_i
LDRW α4, α0 // LOAD W_i
MUL α5 <- α3, α4
ADD α2 <- α2, α5
CMP α0, α1
BGE DONE
ADD α0 <- α0, 1
B LOOP
DONE:
LD α6, 0(b)
ADD <- α7, α2, α6
STR (Y) <- α7

3. **(2 pts)** Consider the following C code and corresponding assembly program below.

```
// C code
int s = 0;
for (int i = 0; i < 1024; i++) {
        x[i] = s * x[i];
        s += x[i];
}

;; assembly
LOOP1:       LD   F0,  0(R1)
             MUL  F4, F0, F2
             ST   F4, 0(R1)
             ADD  F2, F2, F4
             ADDI R1, R1, 8
             SEQ  R3, R1, R2
             BEQZ R3, LOOP1
             NOP
```

a) **(1 pt)** Convert the assembly code above to a DFG where the nodes of the graph represent the instructions.



b) **(1 pt)** Using the DFG above, convert the assembly program above to SSA form, meaning each register only gets a single assignment and each register uses the alpha notation as we did in class.

```
LD   α0,   0(α1)
MUL  α4, α0, α5
ST   α6, 0(α1)
ADD  α5, α5, α6
ADDI α1, α1, 8
SEQ  α3, α1, α2
BEQZ α3, LOOP1
NOP
```

4. **(Bonus 1 pt).** Explain one advantage of compiling source program to virtual instruction sets before generating target instructions.

   It can achieve flexibility when running on different platforms.

5. **(Bonus 1 pt).** Explain how Java Virtual Machine achieves its platform-independence.
   The source code or text is converted into bytecode by the java compiler. This code can be run anywhere from the Java Virtual Machine, which is an abstract computer that then translates that bytecode into machine code.

6. **(Bonus 1 pt).** Explain what Just-In Time (JIT) compilation is, and give an example of where this is used.

   It is used in the Java Runtime Environment. A JIT compiles bytecode right after execution of a program into the native machine code of a platform.

7. **(Bonus 1 pt).** Give an example of programming languages that are 1) compiled, 2) interpreted, 3) both compiled and interpreted.

   Compiled: C, C++

   Interpreted: JavaScript, Python

   Both: Java, compiled into bytecode, which is then interpreted via the JVM

SUBMISSION INSTRUCTIONS (READ CAREFULLY):

1. Please type your responses in a separate document and submit the PDF file to gradescope. ONLY typed PDF documents will be accepted as a valid submission. However, if there are any questions that ask you to draw a diagram, those can be hand-drawn, but must be attached to the final PDF file. Please do not submit a separate file for hand-drawn diagrams. Only one, final PDF file will be accepted.

2. If you have any questions about the homework problems, please post your question on piazza.

3. No late submissions allowed!