Final examination: ECE467
Exam Duration (for writing answers): 150 minutes
Extra time provided (for scanning and uploading sheets): 30 minutes
Maximum marks: 64 points
Note: *You may submit either handwritten notes/ typed notes/notes written on tablet and then export it to PDF*

If you think something is ambiguous, write down your assumptions clearly and proceed to solve the question based on your assumptions.

Run-time Environments:

The frame pointer/control link in an activation record by itself not sufficient to build a backtrace. What else is needed and what is it used for? (2 points)

Parsers:

For each gramma below, show if it's LL(1) and if it's SLR(1). If it's not, explain why (e.g. there is a conflict) (8 points)

a. S->AaAb|BbBa
A->ε
B->ε

b. S->SA|A
A->a

Recursive Descent Top-Down Parsers:

Construct recursive descent parsers starting with the following grammar:

S → SS + | SS * | a | ε

Add necessary features to your parser so that the parser should be able to backtrack.
(5+2 points)

Semantic Analyzer:

Consider the grammar: (6 points)
S → (L) | a and L → L , S | S with **string** ((a,a), a, (a)).
Design an L-attributed SDD for the grammar so that it finally prints the parsed input string as S.*string*, where *string* is a synthesized attribute of S
Also provide an annotated parse tree for the given **string** ((a,a), a, (a)).

Intermediate Code Generation:

Que1: Using the translation of figure 6.43 (page 411 in book), translate the following expression. Show the true and false lists for each sub-expression and give the annotated parse tree as well. You may assume the address of the first instruction generated is 150.
(6 points)

(x==y || p==q) && !(e==f)

Que2: Use the translation of figure 6.22 (page 383 in book) to translate the following assignment:
X = a[b[i][j]] [c[k]] (6 points)

Also provide an annotated parse tree for the above statement.

Machine Code Optimization:

Draw the control flow graph of the following code, and apply the variable liveness analysis. For liveness analysis, you only need to show the first two iterations of the loop. You would need to provide the intermediate code representation of the source code within the basic blocks.

NOTE: A correction to the earlier version of ece467_data_flow_notes.pdf under chapter 9 (the file on the course webpage has been updated). For live variable analysis:

Kill(s): variable x is defined (assigned to) in s and is not ~~subsequently~~ previously used in s (equivalently, x is defined in s prior to any use of x in s)

(8 points)

```
int i, j, k, size = 50;
int dest[100];
for (i = 0; i < size * 2; i++){
    j = i + 2;
    k = i * 2
    if(i % 2 == 0){
        dest[i] = k;
    }
    else{
        dest[i] = j;
    }
}
```

Code Generation:

Use the output of liveness analysis that you have obtained from the previous question (for the first two iterations of the loop) here. Use this information of live variables existing simultaneously at various program points to construct a register-interference graph. What is the minimum number of registers to be allocated for execution? Use the graph coloring algorithm. (5 points)

Instruction Scheduling:

Consider the assembly instructions given below.

(6 points)

```
1       LD R1, a
2       ADD R3, R1, #1
3       LD R2, b
4       SUB  R4, R1, R2
5       ST b, R4
6       ST a, R3
7       LD R5, c
8       ADD R6, R5, R4
9       ST c, R6
```

(i) Identify all the true dependencies
(ii) Perform instruction scheduling (refer lecture slides on the topic). First construct a scheduling directed acyclic graph and using the static scheduling heuristics, perform instruction scheduling. Remember to give candidate list for all iterations. Identify an optimal schedule.
(iii) Assuming that it is a five-stage pipeline and no forwarding is allowed, determine how many pipeline stalls were avoided by your scheduled code as compared to the original code.

Labs:

Que 1: Dynamically allocated arrays can semantically be implemented in our labs using external functions in runtime.cpp (i.e. not the array syntax, but we can emulate the functionality). What external functions do we have to add to support integer arrays only? Write the signature for each function as well as a short description of what it does. You may assume pointers are 32 bits. (Hint: how do you dynamically allocate arrays in C?) (4 points)

Que 2: Consider the following C code:

```
int a = input();
int b = input();
print(a + b);
while (cond1()) {
   if (cond2()) {
      a = input();
      print(a + b);
   }
}
print(a + b);
```

Assuming we had a special phi function in C (ignore scoping rules), the pseudocode in SSA might look something like this (the basic blocks and predecessors have not been indicated, but you should be able to deduce what they would be):

```
int const a0 = input();
int const b0 = input();
int const t0 = a0 + b0;
print(t0);
while (cond1()) { int const a1 = phi(a0, a3); // this happens in the header basic block before
evaluating the expression and branching
   if (cond2()) {
      int const a2 = input();
      int const t2 = a2 + b0;
      print(t2);
   }
   int const a3 = phi(a1, a2);
}
int const t4 = a1 + b0;
print(t4);
```

Write the corresponding C code in SSA form, as if it was using MemorySSA as used in your LLVM lab. The code should work as if all variables are declared to be constant (if you are familiar with the C syntax for const, you may declare your variables as const. If you are not familiar with the syntax, you may declare your variables as regular, mutable variables). (Hint: no phi functions are necessary) (Hint: think pointers) (6 points)