1. **Short definitions**
   a. Translation lookaside buffer
   b. Virtual address
   c. Control-flow graph
   d. Redundant expression
   e. Set associative cache

2. **In Local Value Numbering**, the compiler can identify algebraic identities and use them to simplify the IR form of the code.
   a. List four algebraic identities that LVN might use
   b. For each identity, explain how LVN might simplify the code

3. **Time**
   In the lectures and in the reading, we discussed many events. Some of those occur at compile-time (while the compiler is running) and some that occur at run-time (while the compiled code is running).
   For each of the following events, does it occur at compile time or at runtime?
   a) Spill code is created
   b) Access link is traversed
   c) Activation record is instantiated
   d) Display is referenced
   e) Static coordinate is created
   f) Dependence graph is created
   g) False zero is computed ($A_0$ is the “false zero” of A)
   h) Dope vector is used
   i) New lexical level is added to the symbol table
   j) Tree pattern is matched
4. **Code Shape for Arrays** (30 pts)

Consider an integer array $A$ declared with bounds $A[1:3,1:4]$. (That is, the array has four rows of three elements each.)

\[
\begin{array}{cccc}
A: & 1,1 & 1,2 & 1,3 & 1,4 \\
 & 2,1 & 2,2 & 2,3 & 2,4 \\
 & 3,1 & 3,2 & 3,3 & 3,4 \\
\end{array}
\]

a. Show how the elements of $A$ appear in memory if $A$ is laid out in row-major order and as a set of indirection vectors.

b. Write down low-level pseudo-code to load the array element $A[i,j]$ for each of those storage layouts, assuming that each element of $A$ occupies a 4-byte word.

c. How does the cost of the array access increase with the size of the array and with the number of dimensions, for each of those storage layouts?

5. **LR(1) Parsing**

The LR(1) table construction algorithm starts by building the Canonical Collection of Sets of LR(1) Items. An LR(1) item is a pair $[\alpha, \beta]$ where $\alpha$ is a production in the grammar with a placeholder, $\bullet$, somewhere in it, and $\beta$ is the “lookahead” symbol, a single word (or terminal symbol) from the grammar.

1. If the grammar contains the production $A \rightarrow \beta \delta \gamma$, what LR(1) items can that production generate. Assume a lookahead symbol of “w”.
2. Define the term “handle” in an LR(1) parser?
3. Which of the LR(1) items, if any, in your answer to part (a) represents a handle?

6. **The Procedure Abstraction**

Supporting separate compilation requires that the compiler adopt a set of conventions to govern the externally visible behavior of individual procedures. This convention is often termed a calling sequence. We refer to the calling procedure as the caller and the procedure being called as the callee.

a) Outline the actions that the caller must take at a call and on the subsequent return from the call. Outline the actions that the callee must take on entry and exit. Assume that the language uses call-by-reference parameter passing and does not support lexically nested scopes.

b) To support lexically nested scopes, the calling sequence must be extended to ensure that all variables that are lexically visible in the callee are accessible at run-time. Describe
an efficient mechanism for ensuring such an access. Explain how the call sequence must be modified to accommodate it.

c) Given the mechanism you described in (b), explain how the callee would locate (1) a variable declared locally in the callee, (2) a variable declared in some lexically outer block, and (3) a global variable – that is, one visible throughout the entire program.

7. Symbol Tables
You are writing a compiler for a simple lexically-scoped language. Describe the data structures that you would build to resolve textual names to specific declarations. Describe how the compiler maintains those data structures, with particular mention of the actions taken on entry to and exit from a scope.

Consider the following example program:

```plaintext
procedure main {
    integer a, b, c;
    procedure f1( integer w, integer x ) {
        integer a, y, z;
        call f2(w,x);
    }

    procedure f2( integer y, integer z ) {
        double precision a, y, z;
        procedure f3( double precision m, integer n ) {
            c = m * n * z;
        } call f3( 5.32, z );
    } call f1( a, b );
}
```

Sketch the symbol tables and their contents at the point labelled here. State your assumptions, as necessary, to explain the structures.