# CMSC430 Spring 2009 Final

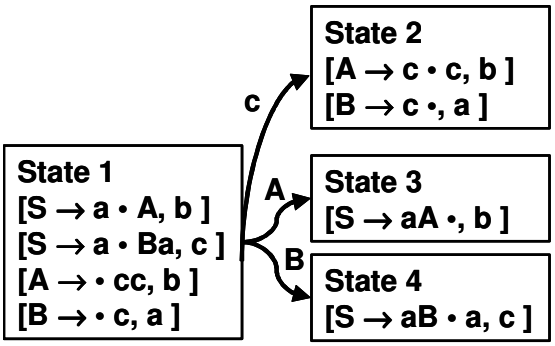
Instructions

- You have until noon to complete the final.
- Feel free to ask questions by raising your hand.
- The exam is out of 160 points, an average of 45 seconds per point.
- 1-2 sentence answers are sufficient for the ``essay'' questions.

1. (9 pts) Compilers
    a. Describe the difference in the how one would implement compiler front ends and back ends when writing a compiler from scratch. Explain the difference
    b. Describe what you think current compilers do well. Explain your answer.
    c. Describe where you think compilers need to improve most, in order to meet future challenges. Explain your answer.

2. (15 pts) LR items
    Consider the following grammar
        S' → S
        S → Sa | b
    a. Construct the canonical set of LR(1) items for the gramar
    b. Is the grammar LR(1)? Explain.
    c. Is the grammar LL(1)? Explain.

3. (14 pts) LR parsing
    a. (10 pts) Construct the ACTION/GOTO table for the following set of LR(1) items




    b. (4 pts) Given the following ACTION/GOTO table, show the parse if the current stack contents are 0 b 4 B 1 (i.e., current state is 1) and the remaining input is "d$".

| State | ACTION | | | GOTO | |
|---|---|---|---|---|---|
| | d | e | $ | S | B |
| 0 | Shift 2 | Reduce S → Bb | Reduce S → bB | 2 | 1 |
| 1 | Shift 3 | Reduce B → d | Accept | 3 | 4 |
| 2 | Shift 1 | Shift 4 | Accept | 2 | 4 |
| 3 | Reduce S → Bd | Reduce B → d | Reduce S → bBd | 1 | 2 |
| 4 | Reduce B → e | Shift 3 | Reduce S → bSe | 4 | 3 |

4. (15 pts) Code generation.

You are generating code for a Java virtual machine which supports the following Java byte codes:

| Java Stack Code | Effect |
|---|---|
| nop | none |
| ldc_int c | push constant *c* onto stack |
| iload index(x) | push local variable X onto stack |
| istore index(x) | pop stack, store in local variable X |
| iadd | pop 2 elems off stack, add, push |
| isub | pop 2 elems, subtract top from 2nd, push |
| imult | pop 2 elems off stack, multiply, push |
| ineg | pop stack, negate, push |
| goto L | jump to handle L |
| ifeq L | pop stack, jump to handle L if zero |
| if_icmpeq L | pop 2 elems, jump to L if equal |
| if_icmpgt L | pop 2 elems, jump to L if 1st greater |
| dup | duplicate top of stack |
| pop | pop top of stack |
| swap | swap top two positions of stack |

You are given the following grammar attributes and helper functions:

| Attribute | Holds |
|---|---|
| AstNode.code | list of instructions |
| **Function** | **Effect** |
| genInst( X ) | create new instruction X<br>returns handle to instruction |
| append( ... ) | concatenates lists of instructions |

A Ruby-style TIMES loop iterates (exp) number of times. For example, *(1+2).times { ... }* would iterate 3 times. Write syntax-directed actions needed to generate code for a TIMES loop for the following grammar production. You may assume that exp evaluates to a non-negative integer value. Note you can only use the Java instructions provided at the beginning of the exam.

*stmt → exp TIMES { stmtList }*

*{: stmt.code = ?? :}*

5. (4 pts) Local information

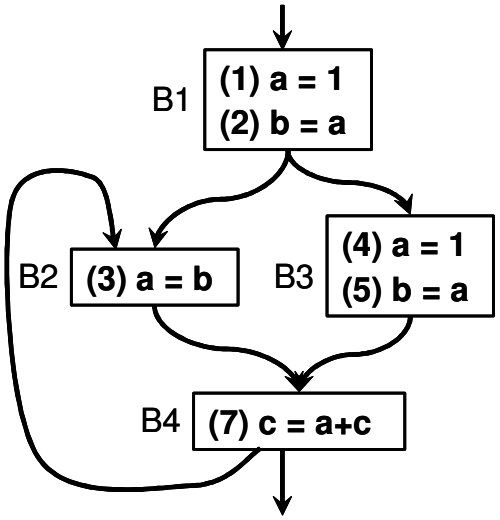
   Consider the following basic block for available expressions:

```
x = a+b
y = b+c
a = c+d
d = a+c
```

   a. What is GEN for the basic block?
   b. What is KILL for the basic block?

6. (22 pts) Live variables

   Consider the following control flow graph for live variables:




   a. (8 pts) Calculate GEN/KILL for each basic block
   b. (14 pts) Initializing IN/OUT to ∅, solve live variables, showing IN/OUT for each pass

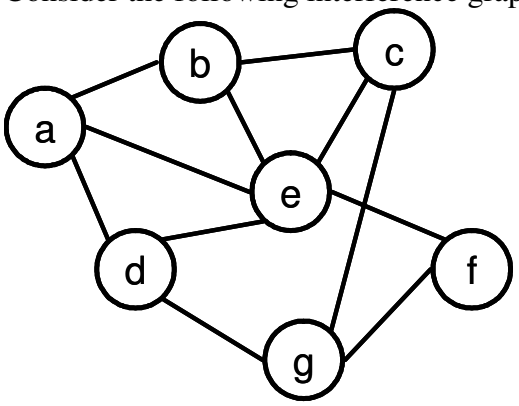
7. (18 pts) Register allocation

   Consider the code in the control flow graph in the previous problem:

   a. (8 pts) What are the live ranges for a global top-down allocator? Provide the variable name and statement numbers for reach live range.
   b. (5 pts) Draw the interference graph for the live range.
   c. (3 pts) If spilling is needed, which live range would be spilled? Explain.
   d. (2 pts) How is code modified when a live range is spilled?

8. (10 pts) Graph simplification

Consider the following interference graph:




a. (6 pts) Apply graph simplification for a machine with 3 registers. Is spilling required? Show your work.

b. (4 pts) Show how to use the results of graph simplification to color the graph with 3 colors 1,2,3.

9. (33 pts) Instruction scheduling

Consider scheduling the code below using list scheduling. All instructions must complete before executing the jmp instruction. Assume the following instruction latencies:

- 2-cycle latency for load
- 1-cycle latency otherwise

| # | op <dest, src1, src2> |
|---|---|
| 1 | load r1, a |
| 2 | add r2, r1, r1 |
| 3 | load r3, b |
| 4 | add r4, r2 r3 |
| 5 | load r1, c |
| 6 | add r3, r1, r1 |
| 7 | add r5, r4, r3 |
| 8 | jmp |

a. (10 pts) Build the precedence graph for the instructions. Mark dependences as flow, anti, or output. You can ignore transitive dependences.

b. (6 pts) Calculate the critical path for the instructions

c. (10 pts) Schedule the instructions for a single-issue processor, using forward list scheduling. Show candidates instructions at each cycle. Prioritize candidates using 1) critical path, 2) latency of instruction, 3) number of children.

d. (7 pts) Schedule the instructions as above, but for a two-issue VLIW processor

10. (20 pts) Dependence analysis

Consider the following loop in Fortran. Recall that Fortran is column-major, so columns are stored contiguously in memory. For both A & B, assume that 4 array elements fit on each cache line.

```
A(N,N), B(N,N)
do i = 1,N-1
  do j = 1,N
    A(i+1, j) = A(i, j)+B(j, i)
  enddo
enddo
```

a. (3 pts) Find any dependences in the loop nest. Give the distance vector for each dependence.
b. (8 pts) What reuse exists in the loop? For each reuse, give type, reference(s), and loop carrying reuse.
c. (5 pts) Which loop permutation is preferred? Calculate by estimating number of cache lines accessed by each loop. Show your work.
d. (2 pts) Is it legal to interchange the i & j loops to make j the outer loop and i the inner loop? Explain.
e. (2 pts) Are either of the loops parallel? Explain.