UCT CSC305 2004 :: Compilers :: Exam [25 marks]

Question 1 : Semantic Analysis [5]

1. Displays are an alternative to static chains for non-local name resolution. Explain how displays are modified when scopes are opened and closed. [2]

When a scope is opened the display pointer corresponding to the nesting level of that scope is saved and then updated to point to the new context. [1] When a scope is closed, the display pointer corresponding to the nesting level of that scope is restored to its previously saved value. [1]

2. Briefly discuss 3 cases where main memory (stack frame) is needed during parameter passing, as opposed to using just registers. [3]

- *variables used/passed by reference*
- *nested subprograms*
- *variable is not simple or just too big*
- *arrays*
- *registers are needed for other purposes*
- *too many variables*

Question 2 : Code Generation [10]

1. What is a basic block? How can the selection of traces improve on efficiency of generated code? [3]

A basic block is a linear sequence of code starting with a LABEL and ending with a JUMP or CJUMP. [1]

Selecting a set of traces which maximises the number of JUMPs followed immediately by the LABELs that are the targets of the preceding JUMPs means that those pairs of JUMP/LABEL statements can be eliminated, thereby creating faster code. [2]

2. In the context of instruction selection by tiling, what is the difference between an optimal and optimum algorithm? [2]

Optimum tiling: sum to lowest possible value [1]

Optimal tiling: no two adjacent tiles can be combined to a tile of lower cost [1]

3. Explain how the maximal munch algorithm works. [4]

- *Start at the root.*
- *Find the largest tile that fits. [1]*
- *Cover the root and possibly several other nodes with this tile. [1]*
- *Repeat for each subtree. [1]*
- *Generates instructions in reverse order. [1]*
- *If two tiles of equal size match the current node, choose either.*

4. Is the maximal munch algorithm optimal or optimum? [1]

Optimal [1]

Question 3 : Register Allocation [10]

1. Use the iterative liveness analysis algorithm to calculate the live-in and live-out sets for each of the following statements in a program, with the initial and final live sets indicated - assume live-in (succ (`a=5`)) = {c}.

[live-in: a]

b = 23

c = a + b

b = 12

a = 5

[live-out: c]

Hint: The relevant formula are:

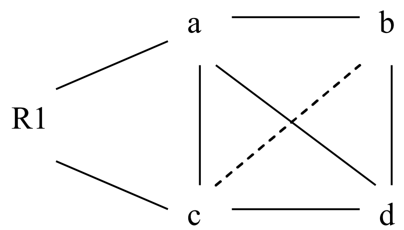
$$out[n] = \bigcup_{s \in succ[n]} in[s]$$

$$in[n] = use[n] \cup (out[n] - def[n])$$

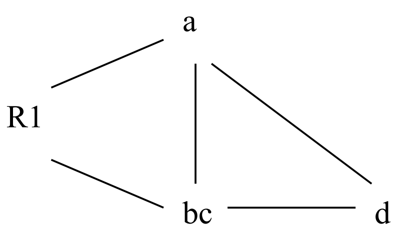
			Iteration 1		*Iteration 2*	
	Use	*Def*	*In*	*Out*	*In*	*Out*
In:A						
B=23		*B*	*A*	*AB*	*A*	*AB*
C=A+B	*AB*	*C*	*AB*	*C*	*AB*	*C*
B=12		*B*	*C*	*C*	*C*	*C*
A=5		*A*	*C*	*C*	*C*	*C*
Out:C						

One mark each for Use set, Def set, In, Out, last two iterations being equal.

2. Consider the following graph with nodes indicating temporaries and arcs indicating interference. Apply a register colouring algorithm to 3-colour the graph. Assume that R1 is a precoloured node and use George's criterion for conservative coalescing. Clearly show all steps in the algorithm and the final register allocation (R1, R2, R3) to temporaries. [5]

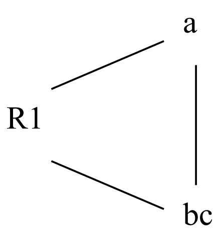


No nodes can be simplified, but b and c can be coalesced since each significant degree neighbour of b interferes with c.



[1]

Then, d can be simplified and pushed onto the stack.



[1]

This makes a and bc of <K degree, so they can be simplified as well.

Popping the nodes off the stack, we can then assign

A: R2[1]

B/C: R3[1]

D: R1[1]