148

CSL 862 - Advanced Compiler Construction
Jan-May 2007

V. Krishna Nandivada

May 9, 2007

There are total three questions. Each question carrying 30 marks each. In each question there are three parts. Part (a) very easy [5 marks], part (b) needs a bit of thinking [15 marks] aud (c) requires considerahle amount of thinking [10 marks]. Q2 has a bonns question (d) for 10 marks. Utilize your time judiciously.

Q1. SSA

(a) Define SSA intermediate form.

(b) List at least three disadvantages of SSA intermediate form compared to the non-SSA form intermediate form. (Clue - Think from the perspective of compilation, and code transformation.)

(c) *SSA and flow sensitivity* Flow (in)sensitive analysis is the analysis that is (in)dependent of the control-flow of the program. So if we have a program

L1: c = 1
L2: a = c + b
L3: c = d
L4: a = c + d

then a flow insensitive def-nse algorithm would decide that both the uses of "c" have two definitions each (from L1, and L3). A flow sensitive def-use algorithm would however give the answer to be L1 for the use of variable "c" at L2 and L3 for the one at L4. Flow sensitive analysis are relatively more time consuming.

Does SSA form obliterate the need for flow sensitive analysis? Qualify your answer in a couple of sentences. You can take cxample analyses as case studies.

Q2. Register Allocation

(a) Define Register Allocation.

(b) Compare and contrast graph based register allocation algorithms with linear scan register allocator.

(c) *Bitwidth Sensitive Analysis*: Assuming that we have only integer variahles in our program, and each register is a 32 hit register. In general, it might not be judicious to allocatc one full register to a variable. Say, wc have two statements:

c = 2
d = 4

we could allocate both "c" and "d" in the same register; "c" taking first two bits and "d" taking next three bits, and still have another 27 bits free!

For eaeh variable in the bitwidth-sensitive-analysis outputs the max-width of the variable in bits. To compute the bitwidth information list the instructions that are of interest (i.e. based on which instruction can you determine the bitwidth information of different variables) and how you will propagate the bitwidth information at join/merge nodes. (Note - Full algorithm is not required to be written).

(d) (*Bonus*) : Devise a scheme to apply bitwidth-information to register allocation. i.e. write a bitwidth-aware-register-allocator. A rough sketch would do. (Clue - Recollect the passes in a graph based register allocation or linear scan register allocation). Side clue - Bitwidth aware register allocation is a NP-complete problem. Hence a heuristic based solution is good enough.

Q3. Control-flow

(a) Define a basic block.

(b) Any data flow analysis (constant propagation, liveness analysis etc) can be done at a basic block level or at each instruction level (each instruction being treated as a "node"). What are the advantages (if any) of treating a basic block as a node? What are the disadvantages (if any) of treating a basic block as a node?

(c) *Extended basic blocks*: An extended basic block is a maximal sequence of instructions beginning with a leader and the list contains no join nodes other than its first node. Note that the leader need not be a join node, it can be the "entry" node as well. So extended basic blocks have single entry and multiple exits. Some local optimizations are more effective when done on extended basic blocks. Such optimizations tend to treat the paths through an EBB as if they were in a single block. Fig. 1 shows a set of basic blocks and extended basic blocks. Write an algorithm to compute extended basic blocks.

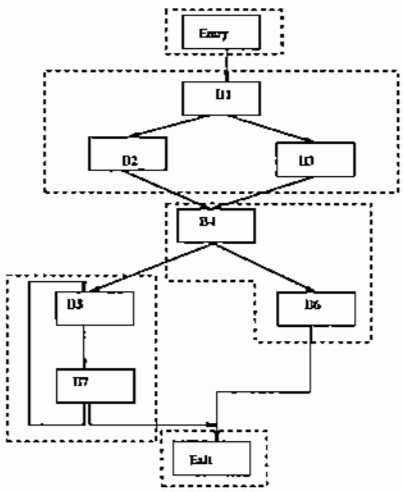

Figure 1: Solid lines indicate basic blocks and dashed lines show the extended basic blocks.

149

CSL 101 - Introduction to Computers and Programming
03 May 2007, 8:00 AM to 10:00 AM
Major Exam
Maximum Marks: 120

1. [5+5+5+5+5+5=30 Marks] Consider an extension of the binary search algorithm, where we divide the array A in which we are searching into 3 parts at each step, instead of 2 parts. The recursive search procedure continues in one of the selected ranges.

(a) Indicate the recurrence relation representing the computational complexity of this algorithm, if the array size is n.
(b) Solve the recurrence. What is the computational complexity of this algorithm?
(c) Is the complexity different from that of binary search? Explain clearly.
(d) What does the recurrence relation become if we divide the array into k parts in each step?
(e) Solve for the recurrence in step (d) above. What is the computational complexity of the algorithm in (d)?
(f) Verify your solution in (e) above by substituting $k = n$ and give an argument that the solution is reasonable.

2. [7.5+7.5+7.5+7.5=30 Marks] We are required to maintain a LINKED LIST L using one or more STACK data structures. That is, we are given the STACK operations: INIT, PUSH, POP, and EMPTY. Using only these, we should be able to perform the following operations on L:
(a) INIT (x): create a new empty LIST
(b) INSERT (x): insert a new element x into the LIST
(c) DELETE (x): delete the node containing x from the LIST
(d) SEARCH (x): search for element x; if found return TRUE, else return FALSE

Give algorithms for each of the above operations. Remember you are not allowed to create a LIST. You are only allowed to create a STACK, and realise the list operations using an equivalent STACK operation (or sequence of operations). Remember that the POP operation REMOVES the element from the top of the stack.

3. [5+5+5+5=20 Marks] What does the following program print? Explain why.

```
#include   <stdio.h>
int  a,d;
void  f  ()  {
  a++;  d++;
}
void  p  (int  r,  int  s)  {
  s = r;
  switch   (r)  {
    case  0:  f  ();
    case  5:  f  ();
    case  10:  f  ();
    default:    break;
  }
}
main  ()  {
  int  a,b,c;
  a = 0;  b = 0;  c = 5;  d = 6;
  p  (c,  b);
  printf   ("a  =  %d,  b  =  %d,  c  =  %d,  d  =  %d\n",   a,  b,  c,  d);
}
```

4. [20 Marks] Develop an algorithm REVERSE for determining the decimal digit reversal of a given integer. It takes as input a positive integer parameter x, and returns the integer that results when the decimal digits of x are reversed. For example:

REVERSE(12355) should return 55321
REVERSE(9) should return 9
REVERSE(1221) should return 1221
REVERSE of a negative number is not defined.

5. [10+10=20 Marks] Consider the assembly-language simulator you developed in Assignments 5,6,7. Instructions in a program are first loaded into memory, and the execution of each instruction involves:

- first accessing memory to feteh the instruction
- possibly reading some operands from memory, if required by the instruction
- possibly writing the results to memory, if required by the instruetion

(a) Suppose a program exeeutes m ADD instructions, n SUB instructions, p BRAG instructions, q STORE instructions, and one final HALT instruction. Compute the total number of memory accesses for the program. (Reading an operand from memory counts as one aeeess. Writing the result to memory counts as one aecess).

(b) Suppose a total of k instructions have been executed in a program. What is the minimum number of memory aecesses? What is the maximum number of memory accesses? Explain both answers.