15-745: Optimizing Compilers

Midterm

Thursday, April 10, 2008

Name: _____

Email: _____

Instructions:

- This is a 1-page-of-notes test.
- Make sure that your exam is not missing any sheets, then write your full name and email address on this sheet.
- Write your answers in the space provided beside or below the problem. If you make a mess, make sure that your final answer is neatly drawn, and then circle it.
- The exam is of 80 points, i.e., about 1 point per minute.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.

1	2	3	4	Total
(20pt)	(20pt)	(20pt)	(20pt)	(80pt)

Problem 1: Scalar Replacement (20 pts)

Scalar replacement is a transformation which can eliminate memory references from within a loop. The idea is to replace subscripted array references by a scalar inside an inner loop. For example, in matrix multiply it significantly reduces memory references by replacing the reads and writes to `c[i][j]` with the temporary scalar `sum`.
(In all the examples and questions assume that `N` is **guaranteed** to be more than 10.)

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++) {

    for (k=0; k<N; k++)
      C[i][j] = C[i][j] + A[i][k]*B[k][j];

  }
```

(a)

becomes

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++) {
    sum = c[i][j];
    for (k=0; k<N; k++)
      sum = sum + A[i][k]*B[k][j];
    c[i][j] = sum;
  }
```

(b)

A) How many memory references in terms of `N` are made by the code on the left (Fig. a)? _____

B) How many are made for the code on the right (Fig. b)? _____

As you can see from the below example, scalar replacement is sometimes referred to as register pipelining.

```
for (i=2; i<N; i++)
  A[i] = A[i-1]+A[i-2];
```

(c)

becomes

```
t0 = A[0];
t1 = A[1];
for (i=2; i<N; i++) {
  t2 = t0 + t1;
  t0 = t1;
  t1 = t2;
  A[i] = t2;
}
```

(d)

Notice how we have to add extra temporary scalars to hold values as they "age." Using these two examples as a guideline, perform scalar replacement on the following two examples.

C) Write your code to the right of the original

```
for (i=0; i<N; i++) {
   for (j=0; j<N; i++) {
      A[i] = A[i] + B[j]*B[j-1];
   }
}
```

D) Write your code to the right of the original

```
for (i=0; i<N; i++) {
        b[i+1] = b[i] + f
        a[i] = 2 * b[i] + c[i]
}
```


Continued . . .

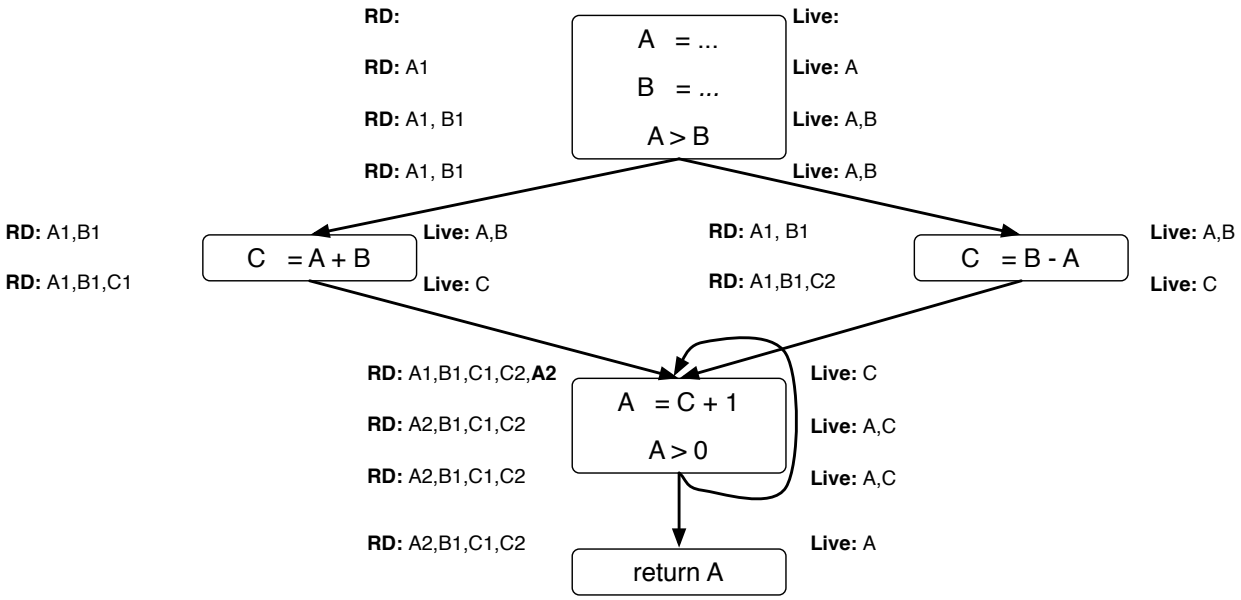
(Question 1 cont'd)

E) For the final part of this question you should describe the analysis neccesary to determine when a memory references in a loop can be replaced by a scalar and the algorithm to perform scalar replacement. Things to keep in mind and/or simplify the problem:

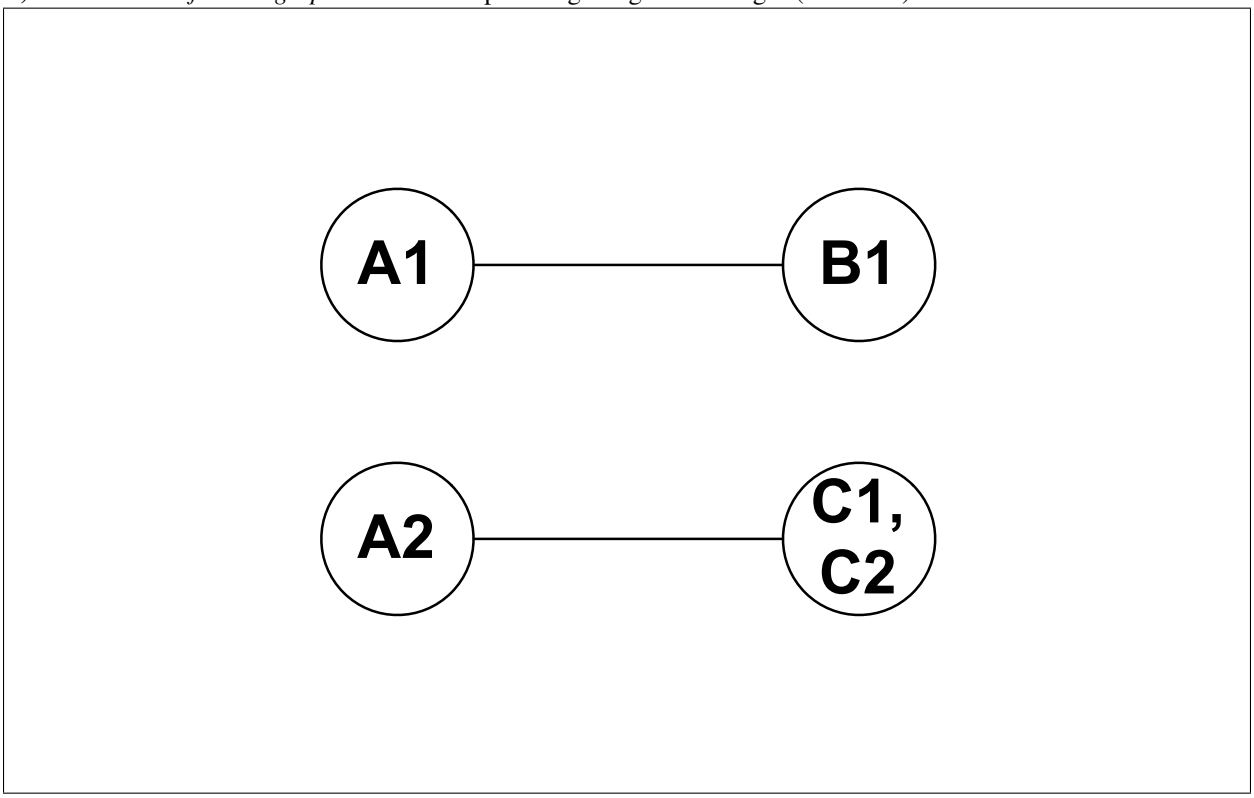
- Scalar replacement will only be performed in loop nests without conditionals.
- Asssume that the trip count of the loop is larger than the number of temporaries you will insert into the loop.
- Only perform scalar replacement on array references where the index expression consists of an induction variable possibly offset by a constant.

Problem 2: Register Allocation (20 pts)

A) Construct the liveness sets and the reaching definitions sets for each program point in the example below.



B) Build the *interference graph* for this example using merged live ranges (aka webs) as the nodes.

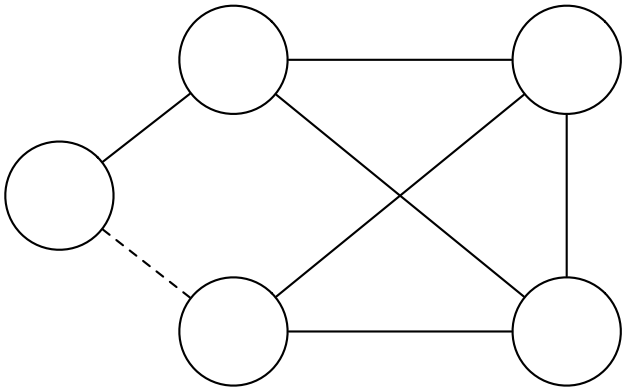


C) What is the minimum number of registers required to avoid spilling in this example? **2**

Continued . . .

(Question 2 cont'd)

D) Draw an interference graph which contains a move edge (represented by a dashed line) for which coalescing the nodes connected by the move edge is harmful if there are only three general-purpose registers available.



Problem 3: List Scheduling (20 pts)

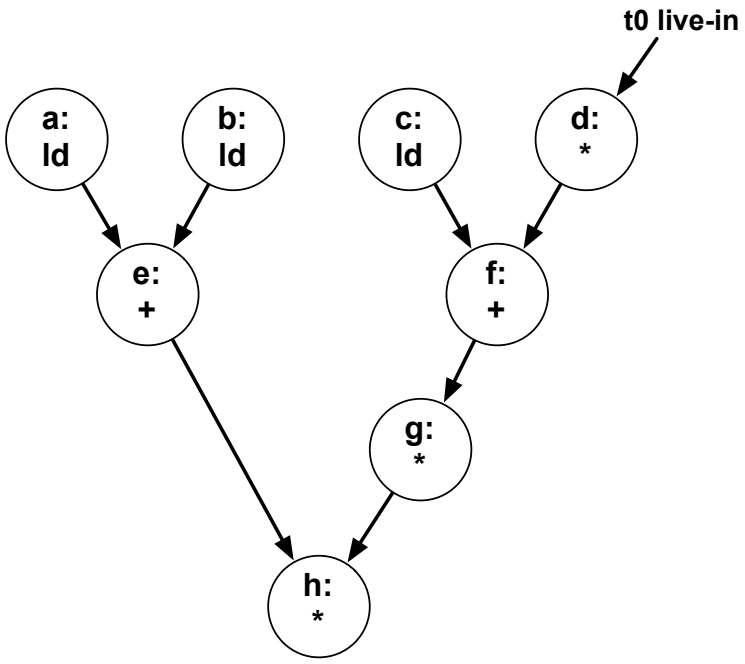
Assume we are scheduling for a processor with the following characteristics:

- There are **2** identical fully-pipelined functional units that can execute all instruction types.
- Multiplications take **2** cycles, loads take **3** cycles, and all other instructions take 1 cycle.
- There are only **3** general purpose registers.
- If two instructions are scheduled at the same cycle and one instruction writes the same register that the other reads the result is undefined. That is, this schedule:

`r0 ← r1*33`	`r1 ← r2+1`

  is illegal (the add should be scheduled in the next cycle).

Consider the following data dependence graph:



In this example the source of operation **d** is the only value live into the code.
Your task is to devise a selection heuristic for use within the list scheduling algorithm that attempts to maximize the parallelism of the schedule (i.e., minimize the schedule length) but under the constraint that it should not introduce unnecessary spill code. For example, it is possible to schedule the above code so that there are never more than 3 simultaneously live values (no need to spill) and the schedule length is ≤ 11.

A) Ignoring register constraints, what is the minimum schedule length for this code? Recall that the schedule length refers to the cycle when the result of the last operation to complete is ready (not when the last operation is issued).

Continued . . .

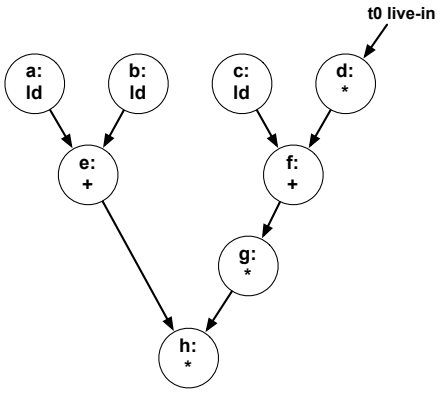
(Question 3 cont'd)

B) Provide pseudo-code for your selection heuristic. Your code should take as input the list of available nodes as per the standard list scheduling. You may assume any pertinent properties of the data dependence graph are already computed, but be sure to describe exactly what properties you are using.

C) Use your heuristic and the list scheduling algorithm to fill out the following table for the provided code.

FU #1	FU #2	Cycle
		0
		1
		2
		3
		4
		5
		6
		7
		8
		9
		10
		11
		12
		13
		14
		15

The data dependence graph is repeated here for your convenience.



Problem 4: Dataflow Analysis for Bitwidth (20 pts)

For this problem, you will define a dataflow analysis to conservatively reduce the number of bits required to hold the result of each operation. The only data type is unsigned 32-bit integer. The only operation types besides constants, comparisons, and branches are addition, subtraction, multiplication, and bitwise-AND.
The result of this analsis for the program fragment:

```
a = 3;
b = 1;
c = a + b;
d = c & 0x1;
e = d + c;
```

would indicate that `a` needs to be only 2-bits wide, `b` 1-bit, `c` 3-bits, `d` 1-bit, and `e` 4-bits wide.

A) Is this a forward or backward problem?

B) What is the lattice of values on each output wire for this problem?

C) Describe the transfer function for addition, subtraction, multiplication, and bitwise-AND, in terms of `in1`, `in2`, and `out`.

D) What is the meet function?

Continued . . .

(Question 4 cont'd)

E) What are initial values for constants? For additions?

F) Is iterative dataflow analysis for this problem guaranteed to terminate?