ECE 573 Fall 2008, Exam 2
version 2

Leave this on your desk until asked to start

You have until 10:15 to take this exam.

Your exam should have 11 pages total (including this cover sheet). *Please contact Prof. Midkiff immediately if it does not.*

This exam is open book and open notes. If you have a question, please ask Prof. Midkiff. If the answer is still not clear, state whatever assumptions you need to make to answer the question, and answer it under those assumptions. Check the front board occasionally for corrections.

There are 100 points on this test.

Please put your name on the test – it is worth 1 point.

Name:

In the following questions, you will be given a loop nest and asked which transformations could help the performance of the loop nest, and why. For best cache locality, in an array reference `a[i,j]`, the `j` subscript should be changing the fastest. Assume any code indicated by "..." has no effect on the effectiveness of the transformations. The possible transformations you should consider are: *interchange, fusion, fission, blocking, tiling, software pipelining.*

Question △a (3 pts.): Given the loop nest: & unrolling

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        a[i, j] = b[j, i]
    }
}
```

Give the transformation above would that would most help the performance, and briefly say why.

tiling - will improve cache locality

Question △b (3 pts.): Given the loop nest:

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        a[j, i] = ...
    }
}
```

Give the transformation above would that would most help the performance, and briefly say why.

interchange - will improve cache locality

Question △c (4 pts.): In the loop nest:

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        a[i, j] = b[i, j] + c[i, j]
    }
}
```

cache locality is good, but functional units within the processor are not being fully utilized. Give two transformations that could help the performance of the loop, and briefly say why they would help.

loop unrolling
software pipelining

will increase use of functional units and give better instruction level parallelism

Given the 3-address code for a basic block:

1. li A, 57 /* A = 57 */
2. li C, 0
3. li E, 0
4. ld (A), B /* B = *A */
5. + C, B, C
6. + C, A, D /* D = C + A */
7. + E, D, F

∇b (12 pts): Fill in the table below by performing a register allocation using the bottom-up down method of Torczon and Cooper. Assume 3 registers are available to be allocated (r1, r2 and r3). Ig there is a tie between registers to be spilled or loaded, pick the lowest numbered register.

If you need to spill the register `rx` containing the value for some variable `A`, insert the instruction `st rx, A` into the table in the proper place. There is no need to spill registers at the end of the basic block. Space has been provided (columns labeled `r1`, `r2`, `r3`) to keep track of what variable is what register. I may use these to figure out what you have done in order to assign partial credit, but they will not be directly graded.

Extra space is provided in the table to peform loads and stores necessary to bring values into registers and to spill them. Some rows may be empty after you have finished.

The three instruction have been allocated as an example.

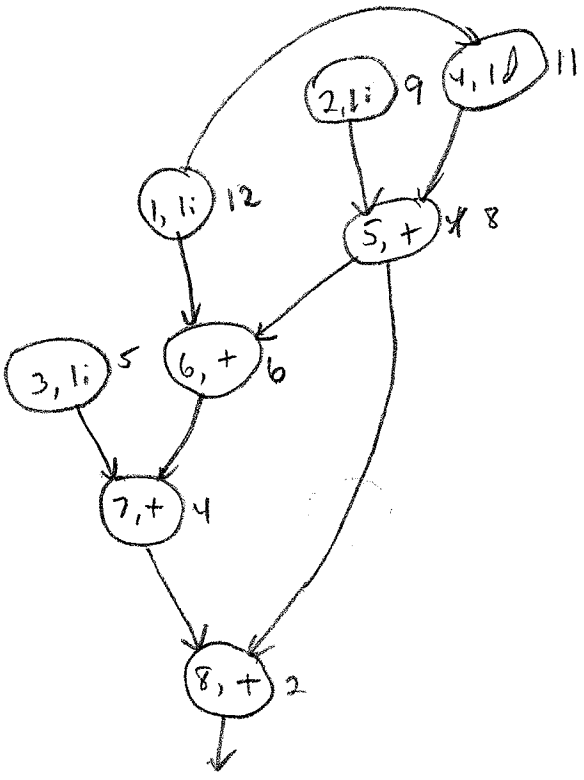
Num	Instruction	r1	r2	r3
1	`li r1, 57`	A		
2	`li r2, 0`		~~B~~C	
3	`li E, 0`			~~C~~ E
4	st r3, E			free
5	ld (r1), r3			B
6	+ r2, r3, r2			
7	st r2, C		free	
8	st r1, A	free		
9	+ r2, r1, r1	D		
10	ld r2, E		E	
11	st r1, D	free		
12	+ r2, r1, r1	F		
13				
14				
15				

Consider the 3-address code for a basic block:

1. li A, 57
2. li C, 0
3. li E, 0
4. ld (A), B
5. + C, B, C
6. + C, A, D
7. + E, D, F
8. + C, F, G

♯a (12 pts): Schedule the instructions above using the linear instruction scheduling algorithm described in the class notes. Assume 1 cycle for a load immediate (`li`) instruction, 2 cycles for an add (+) instruction, and 3 cycles for a load from memory (`ld`) instruction.

Draw the precedence graph. Each node in the precedence graph should be labeled with the instruction number and operation (e.g. `li`). Then show when each instruction is scheduled in the table below. If an instruction is scheduled in cycle 5, it should be put in line 5 of the table. Some lines may be empty after the instructions are scheduled.



Cycle	Instruction
1	li A, 57 (1)
2	ld (A), B (4)
3	li C, 0 (2)
4	li E, 0 (3)
5	+ C, B, C (5)
6	
7	+ C, A, D (6)
8	
9	+ E, D, F (7)
10	
11	+ C, F, G (8)
12	
13	

Consider the code:

```
void main( ) {
   int a = 4;
   int b;
   int c = 12;
   switch(a,a,c); }

void switch(int u, int v, int w) {
   S1: v = u;        u=4
   S2: u = w;        v=4
   S3: w = v;        w=12
}
```

Question ⋆a (3 pts): Assume that all parameters are passed by reference.

- What are the values of u, v and w after $S1$ executes?

  u=4, v=4, w=12

- What are the values of u, v and w after $S2$ executes?

  u=12, v=12, w=12

- What are the values of u, v and w after $S3$ executes?

  u=12, v=12, w=12

Question ⋆ (3 pts)b: Assume that all parameters are passed by value.

- What are the values of u, v and w after $S1$ executes?

  u=4, v=4, w=4

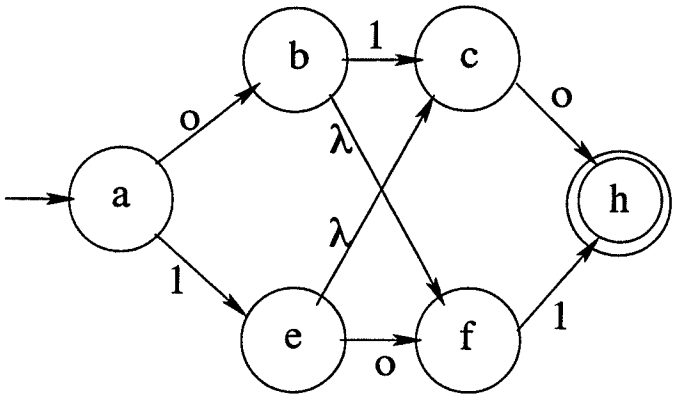
- What are the values of u, v and w after $S2$ executes?

  u=12, v=4, w=12

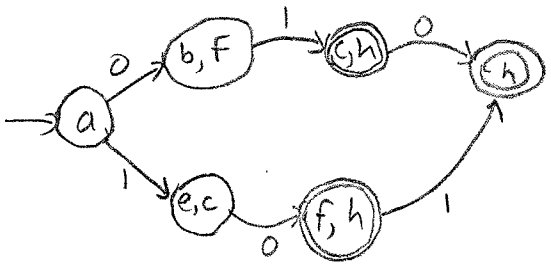
- What are the values of u, v and w after $S3$ executes?

  u=12, v=4, w=4

You will use the NDFA below in this question.



Question ♣a (4 pts): Given the NDFA above, show the DFA.



a = p
b,f = q
e,c = r
c,h = s
f,h = t
h = u

Question ♣b (4 pts): Given your DFA, give the minimal DFA. Hint: transitions to the error state need to be considered.

{p, q, r} {s, t, u}

{p} {q, r} {s, t, u} {q, r} go to {s, t, u} and {p} does not

{p} {q} {r} {s, t, u} q → error on 0, r goes to {s, t, u}
r → error on 1, q " " {s, t, u}

{p} {q} {r} {s} {t} {u} u → error on 0, 1 {s, t,} go to {s, t, u} on one of these

s → error on 1, goes to {s, t, u} on 0
t → " " 0, " " " " 1

An LR(0) grammar G_1 and its associated LR(0) action and goto tables are shown below.

1. $S \longrightarrow A\ B\ \$$
2. $A \longrightarrow a\ B$
3. $B \longrightarrow b\ A$
4. $B \longrightarrow d$

Goto table						
	A	B	$	a	b	d
1	2			5		
2		3			8	7
3			4			
4						
5		6			8	7
6						
7						
8	9			5		
9						

Action table								
1	2	3	4	5	6	7	8	9
S	S	S	A	S	R2	R4	S	R3

Question ♠a (4 pts): Given the current configuration of an LR(0) parser (where the rightmost item on a stack is the one pushed on last):

symbol stack : a b

state stack : 1 5 8

next token : a

- What is the next action? shift
- What is the state of the symbol and state stacks after the action is completed? a b a
  1 5 8 5

Question ♠b (4 pts): Given the current configuration of an LR(0) parser (where the rightmost item on a stack is the one pushed on last):

symbol stack : a b a d

state stack : 1 5 8 5 7

next token : d

- What is the next action? Reduce 4 (R4)
- What is the state of the symbol and state stacks after the action is completed? a b a B
  1 5 8 5 6

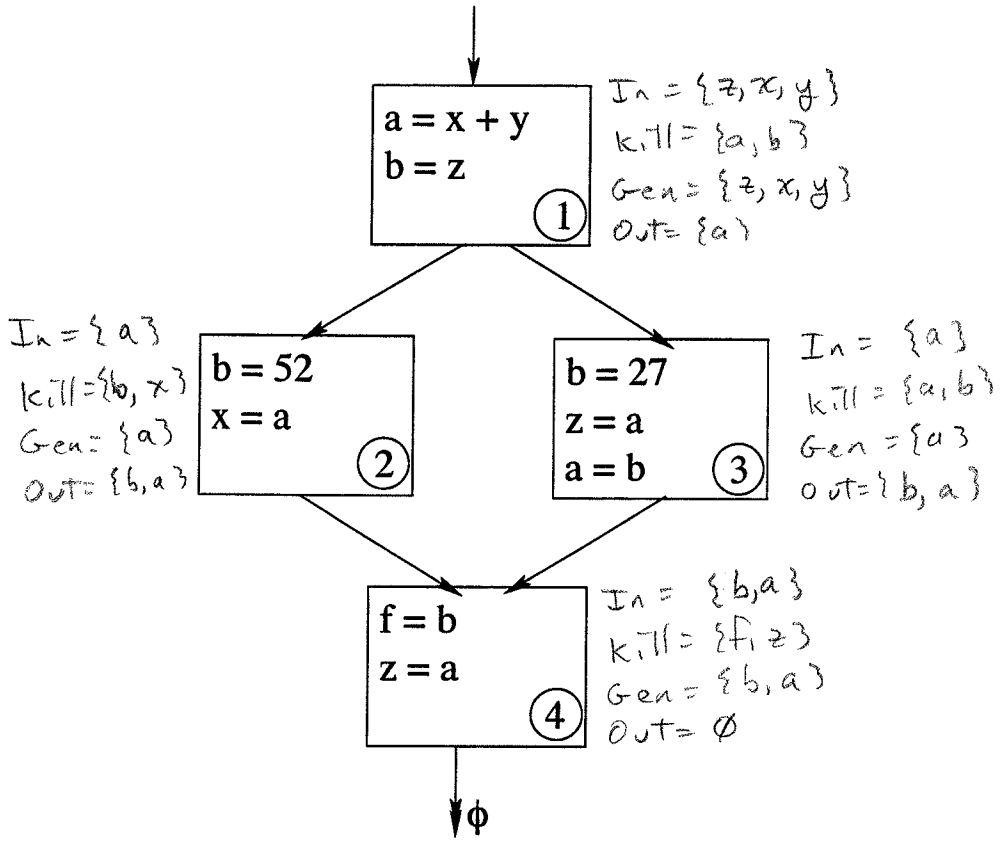
Question question ♠c (4 pts): Given the current configuration of an LR(0) parser (where the rightmost item on a stack is the one pushed on last):

symbol stack : (A) b A ← should be "a" but does not affect the answer

state stack : 1 5 8 9

next token : $

- What is the next action? Reduce 3 (R3)
- What is the state of the symbol and state stacks after the action is completed? 1 5 6
  A B



Live variable analysis determines what variables, at the end of a basic block, are used later in the execution of a program before the variables is assigned. Answer the following:

- The set *GEN(B)* contains variables that are used before being defined in the basic block B.
- The set *KILL(B)* contains variables that are defined in the basic block B.
- The dataflow equations for the analysis are

$$\begin{aligned} IN(B) &= (OUT(B) - KILL(B)) \bigcup GEN(B) \\ OUT(B) &= \bigcup_{S \in succ(B)} IN(S) \end{aligned}$$

Question ♡a (10 pts): Beside each basic block above, give the contents of the IN, OUT, GEN and KILL set for the block.

Question ♡a (8 pts): Dead variable analysis determines what variables in a basic block B are *not* used later in the program before, or are not used before being redefined. Answer the following:

- describe informally what is in the the GEN set? variables written in the block before being used
- describe informally what is in the the KILL set? variables read (used) in the block
- What is the dataflow equation for the IN set? $In = (Out - kill) \cup Gen$
- What is the dataflow equation of the OUT set? $Out^{(B)} = \bigcap_{s \in succ(B)} In(B)$
- Is this a forwards or backwards analysis? backwards
- Is this an any or all paths algorithm? all paths

Given the loop below:

```
for (i = 0; i < n; i++) {
    a[i + 1] = b[2 * i] + c[i + 3]
    b[2 * i - 1] = a[i]
    c[i] = 52
}
```

Question ◇a (9 pts): Fill in the table below with the requested dependence information. Note that the references are simply listed in the order they appear in the program – the dependence is not necessarily from the first reference listed to the last one.

reference pairs	dependence type (or none)	distance	direction
a[i + 1], a[i] pairs	flow	1	<
b[2 * i], b[2 * i - 1] pairs	none		
c[i + 3], c[i]	anti	3	<

Answer the question below using this loop:

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        a[i, j] = ...
        ... = a[i + 1, j]
    }
}
```

Question ◇b (6 pts): In the following questions, assume no other transformations other than parallelization, are performed.

- What kind of dependence, if any, exists between the two references?

  anti

- If a dependence exists, what is its distance vector?

  1

- If a dependence exists, what is its direction vector?

  <

- Can the i and j loop both be parallelized?

  no - loop carried dep. on i

- Can the i loop alone be parallelized? Why or why not?

  no - see above

- Can the j loop alone be parallelized? Why or why not? yes - no loop carried dependence on j

Question ◇c (6 pts):

Given the loop nest at the top of the page, Is the transformation shown below legal? Why or why not?

no - writes to a[i,j] occur before read, anti-dependence is violated

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        a[i, j] = ...
    }
}
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        ... = a[i + 1, j]
    }
}
```

Given the loop nest at the top of the page, Is the transformation shown below legal? Why or why not?

yes - anti-dependence honored (use-def chain replaces anti)

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        ... = a[i + 1, j]
    }
}
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        a[i, j] = ...
    }
}
```