ECE 495S/573 Fall 2007, Exam 2
version 1

DO NOT START WORKING ON THIS UNTIL TOLD TO DO SO.

You have until 8:20 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 the 495 test, and 118 points on the 573 test.

Please put the requested information on the test – it is worth 3 points.

Name: key

Student ID: 123-45-678

Class (495 or 573): both updated 12/5 at 9:00 am

Let G_1 be:

$$\begin{array}{rcl} S & ::= & N\ \$ \\ N & ::= & B\ B \\ B & ::= & 0 \\ B & ::= & 1 \end{array}$$

♣a (3 pts): What is First(B)? B B

{0, 1}

♣b (3 pts): What is Follow(B)? {0, 1, $}

♣c (6 pts): Write a recursive descent parser for G_1.

```
S() {
    N();
    match("$");
}
```

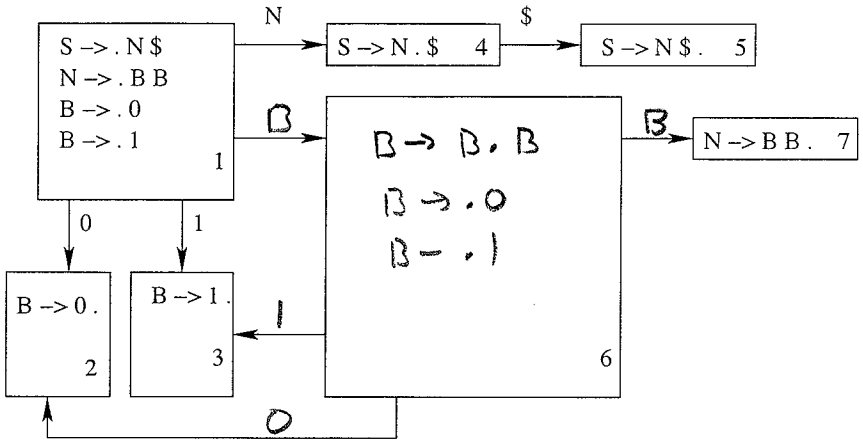
```
N() {
    B();
    B();
}
```

```
B() {
    if nextoken("0") match("0")
    else match("1");
}
```

G_1 (the same grammar as on the previous page) is:

$$\begin{aligned} S &::= N\ \$ \\ N &::= B\ B \\ B &::= 0 \\ B &::= 1 \end{aligned}$$

A partial CFSM for G_1 is shown below:



♠a (5 pts): Fill in state six, and any unlabeled edges.

♠b (2 pts): Based on the CFSM, is the grammar LR(0)?

yes (no shift/reduce conflicts)

♠c (3 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 : *empty*

state stack : 1

next token : 0

- What is the next action? shift
- What is the state of the symbol and state stacks after the action is completed?

symbol: 0 state: 1 2

question ♠d (3 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 : 1

state stack : 1, 3

next token : 0

- What is the next action? Reduce
- What is the state of the symbol and state stacks after the action is completed?

symbol: B state: 1, 6

G_1 (the same grammar as on the previous page) is:

$$\begin{aligned} S &::= N\ \$ \\ N &::= B\ B \\ B &::= 0 \\ B &::= 1 \end{aligned}$$

♠e (3 pts): **(573 only)** Is the language of the grammar finite or infinite? If infinite, how many members (sentences) are in the language?

finite

"in" typo makes this meaningless

♠f (3 pts): **(573 only)** What is the language recognized by this grammar? Is it recognizable by a DFA?

(011)(011)$

Given the 3-address code for a basic block:

1. ld a, T1
2. ld b, T2
3. + T1, T2, T3
4. ld c, T4
5. + T1, T2, T5
6. + T4, T5, T6
7. + T3, T6, T7
8. ST T7, a

♡ (15 pts): Fill in the table below by performing a register allocation using the bottom-up method of Torczon and Cooper. Assume 3 registers are available (r1, r2 and r3). If a register needs to be spilled pick the one whose value is used the furthest away. In case of a tie, spill the lowest numbered register in the tie. When allocating registers, if more than one is free pick the lowest numbered one to allocate. Assume all registers are initially free.

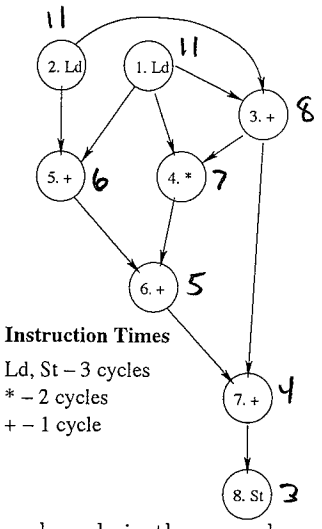
In the table, at row i fill in statement i above in the basic block in the first column. In the column for reach register, show what value is contained in the register *after* the statement has been register allocated. Registers that are free should be left blank. Registers that are marked free during the allocation of the statement should be left blank *even if the register is used in the statement.*

Extra space is provided in the table – show where loads and stores caused by register spills occur.

The first instruction has been allocated as an example.

Num	Instruction	r1	r2	r3
1	ld a, r1	T1		
2	ld b, r2	T1	T2	
3	+ r1, r2, r3	T1	T2	T3
4	st r3, t3 ld c, r3	T1	T2	T4
5	+ r1, r2, r1	T5		T4
6	+ r3, r1, r1 ld t3, r2	T6		
7	+ r2, r1, r1	T7		
8	st r1, a			

Num	Instruction
1	Ld a, T1
2	Ld b, T2
3	+ T1, T2, T3
4	* T1, T3, T4
5	+ T1, T2, T5
6	+ T4, T5, T6
7	+ T3, T6, T7
8	St T7, a



◇a (6 pts): Fill in the cumulative delay next to each node in the precedence graph above.

◇b (9 pts): Using the delays in the precedence graph above, schedule the code using the table below. In row i of the table below, write the number of the instruction that is scheduled to begin in cycle i. If no instruction can be scheduled at cycle i, leave that row of the table blank. Only one instruction is scheduled each cycle. If two instructions are ready to be scheduled, and both have the same delay, schedule the small numbered instruction first. More rows are provided in the table than you need.

Cycle	Instruction Scheduled
1	1
2	2
3	
4	
5	3
6	4
7	5
8	6

Cycle	Instruction Scheduled
9	7
10	8
11	
12	
13	
14	
15	
16	

Given the loop nest:

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

△a (5 pts): What transformation other than parallelizing the loop could help the performance of the loop?

tiling

△b (5 pts): How does it help the performance of the loop nest? (One or two sentences is enough to answer the question.)

improves spatial locality.
reducing cache misses

Loop nest 1		Loop nest 2		Loop nest 3

for $(i = 0; i < n; i++)$ {
a[i] = a[i-1] + b[i]
}
		for $(i = 0; i < n; i++)$ {
		a[i] = a[i] + b[i]
		}
				for $(i = 0; i < n; i++)$ {
				a[4*i] = a[2*i-1]
				}

∇a (5 pts) **(495S only)**: Which of the loop nests 1 and 2 above can be parallelized?

2
(1 has loop carried flow dependence from a[i] to a[i-1]

∇b (5 pts) **(573 only)**: Which of the loop nests 1, 2 and 3 above can be parallelized?

2, 3

Given the loop nest below:

```
for (i = 0; i < n; i++) {
    x[i] = x[i-1] + x[i+1]
}
```

♯ (10 pts): Fill in the table below:

Pair of references	Dependent? (Yes or No)	Kind of dependence	Distance of the dependence	Direction of the dependence	Loop Carried
x[i], x[i-1]	Y	flow or true	1	<	Y
x[i], x[i+1]	Y	anti	1	<	Y

∞ (7 pts) (573 only): Consider the loop nest:

```
for (i = 0; i < n; i++) {
    x[i] = x[i-1] + x[i+1]
}
```

The machine we are compiling for has just enough registers to allocate the loop without spilling. Moreover, the machine has a slow cache – loading a value into the register from the L1 cache is at least 10 times slower than simply accessing the value from the register.

Given the list of transformations below, say which are *more* profitable (useful), *less* profitable (useful), or the *same* usefulness given the slow L1 cache. Check the appropriate column for each transformation listed below.

Assume for all of your answers that creating extra loops does not use any additional registers – only the body of the loop counts.

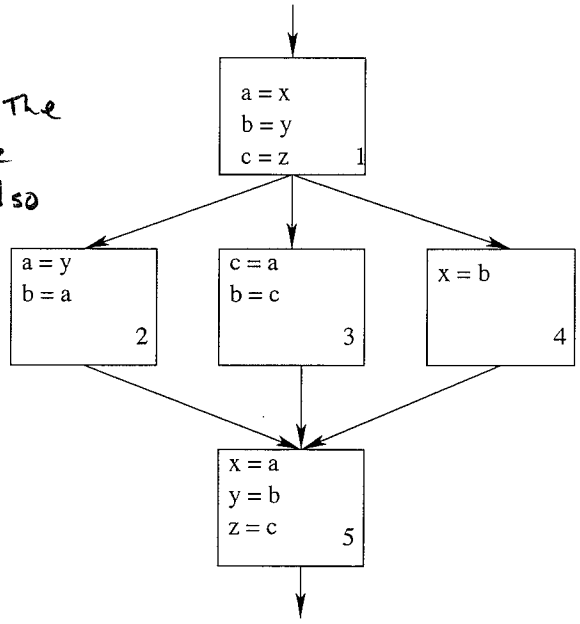
Transformation	More Likely	Less Likely	The Same
Loop fusion (with another loop)		✓	
Loop Blocking (Stripmining)			✓
Tiling	✓*		✓
Loop unrolling		✓	
Software Pipelining		✓	

* if register blocking. Either "the same" or "more likely" got credit

⋆a (11 pts): This question is concerned with the *live value* dataflow analysis algorithm. A value is alive if it is used after the basic block it is defined in. For each block in the control flow graph below, say what the contents of the *KILL, GEN, IN and OUT* sets are for the block.

1 poor wording of the underline phrase allows "a" to also be correct

2 as 1, except "c" allowed



Basic Block	*GEN*	*KILL*	*IN*	*OUT*
1	x, y, z	a, b, c	x, y, z	a, b, c, y
2	y see 1	a, b	c, y	a, b, c
3	a see 2	b, c	a	a, b, c
4	b	x	a, b, c	a, b, c
5	a, b, c	x, y, z	a, b, c	∅

⋆b (6 pts): Assume a, b and c are in registers at the end of basic block 1. Which of them must be saved at the end of the basic block?

a, b, c because they are alive, as seen by their presence in OUT (1)

⋆c (5 pts) (573 only): How many iterations will need to be made over the graph for the algorithm to converge?

1 since acyclic