The University of Alabama in Huntsville
ECE Department
Homework Assignments
CPE 431/531 01/91/92
Fall 2015
Homework #5 Solution

Use the code below for 4.18, 4.18.2(10), 4.18.3(10), 4.18.4(10), 4.18.5(10), 4.19 (given below) (30), 5.1.1(5), 5.1.2(5), 5.1.3(5), 5.1.4(10), 5.1.6(5)

```
        add   R5, $zero, $zero
Again:  beq   R5, R6, End
        sll   $t0, R5, 2
        add   R10, R1, $t0
        lw    R11, 0(R10)
        lw    R10, 4(R10)
        sub   R10, R11, R10
        add   R11, R2, $t0
        sw    R10, 0(R11)
        addi  R5, R5, 2
        beq   R0, R0, Again
End:
```

4.18 In this exercise, we compare the performance of 1-issue and 2-issue processors, taking into account program transformations that can be made to optimize for 2-issue execution. Problems in this exercise refer to the following loop (written in C):

```
for (i = 0; i != j; i += 2)
    b[i] = a[i] - a[i+1]
```

For the MIPS code, assume that variables are kept in registers as follows, and that all registers except those indicated as Free are used to keep various variables, so they cannot be used for anything else.

i	j	a	b	c	Free
R5	R6	R1	R2	R3	R10, R11, R12

```
        add   R5, $zero, $zero
Again:  beq   R5, R6, End
        sll   $t0, R5, 2
        add   R10, R1, $t0
        lw    R11, 0(R10)
        lw    R10, 4(R10)
        sub   R10, R11, R10
        add   R11, R2, $t0
        sw    R10, 0(R11)
        addi  R5, R5, 2
        beq   R0, R0, Again
End:
```

4.18.2 If the loop exits after executing only two iterations, draw a pipeline diagram for your MIPS code executed on a 2-issue processor show in Figure 4.69. Assume the processor has perfect branch prediction and can fetch any two instructions (not just consecutive instructions) in the same cycle.

Issue Slot	Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	add R5	IF	ID	EX	ME	WB																		
	beq R5	IF	ID	ID	EX	ME	WB																	
1	sll $t0		IF	IF	ID	EX	ME	WB																
	add R10		IF	IF	ID	ID	EX	ME	WB															
2	lw R11					IF	ID	EX	ME	WB														
	lw R10					IF	ID	EX	ME	WB														
3	sub R10						IF	ID	ID	EX	ME	WB												
	add R11						IF	ID	ID	EX	ME	WB												
4	sw R10							IF	IF	ID	EX	ME	WB											
	addi R5							IF	IF	ID	EX	ME	WB											
5	beq R0									IF	ID	EX	ME	WB										
	nop									IF	ID	EX	ME	WB										
6	beq R5										IF	ID	EX	ME	WB									
	sll $t0										IF	ID	EX	ME	WB									
7	add R10											IF	ID	EX	ME	WB								
	lw R11											IF	ID	ID	EX	ME	WB							
8	lw R10												IF	IF	ID	EX	ME	WB						
	sub R10												IF	IF	ID	ID	ID	EX	ME	WB				
9	add R11														IF	IF	IF	ID	EX	ME	WB			
	sw R10														IF	IF	IF	ID	ID	EX	ME	WB		
10	addi R5																	IF	IF	ID	EX	ME	WB	
	beq R0																	IF	IF	ID	EX	ME	WB	
11	beq R5																			IF	ID	EX	ME	WB
	nop																			IF	ID	EX	ME	WB

4.18.3 Rearrange the code to achieve better performance on a 2-issue statically scheduled processor from Figure 4.69.

```
        add   R5, $zero, $zero
Again:  beq   R5, R6, End
        sll   $t0, R5, 2
        add   R10, R1, $t0
        lw    R11, 0(R10)
        lw    R10, 4(R10)
        sub   R10, R11, R10
        add   R11, R2, $t0
        sw    R10, 0(R11)
        addi  R5, R5, 2
        beq   R0, R0, Again
End:
```

```
        add   R5, $zero, $zero
Again:  sll   $t0, R5, 2
        beq   R5, R6, End
        add   R10, R1, $t0
        add   R12, R2, $t0
        lw    R11, 0(R10)
        lw    R10, 4(R10)
        sub   R10, R11, R10
        addi  R5, R5, 2
        sw    R10, 0(R11)
        beq   R0, R0, Again
End:
```

4.18.4 Repeat 4.18.2, but this time use your MIPS code from 4.18.13

Issue Slot	Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0	add R5	IF	ID	EX	ME	WB																	
	nop	IF	ID	EX	ME	WB																	
1	sll $t0		IF	ID	EX	ME	WB																
	beq R5		IF	ID	EX	ME	WB																
2	add R10			IF	ID	EX	ME	WB															
	add R12			IF	ID	Ex	ME	WB															
3	lw R10				IF	ID	EX	ME	WB														
	lw R11				IF	ID	EX	ME	WB														
4	sub R10					IF	ID	ID	EX	ME	WB												
	addi R5					IF	ID	ID	EX	ME	WB												
5	sw R10						IF	IF	ID	EX	ME	WB											
	beq R0						IF	IF	ID	EX	ME	WB											
6	sll $t0								IF	ID	EX	ME	WB										
	beq R5								IF	ID	EX	ME	WB										
7	add R10									IF	ID	EX	ME	WB									
	add R12									IF	ID	EX	ME	WB									
8	lw R10										IF	ID	EX	ME	WB								
	lw R11										IF	ID	EX	ME	WB								
9	sub R10											IF	ID	ID	EX	ME	WB						
	addi R5											IF	ID	ID	EX	ME	WB						
10	sw R10												IF	IF	ID	EX	ME	WB					
	beq R0												IF	IF	ID	EX	ME	WB					
11	sll $t0														IF	ID	EX	ME	WB				
	beq R5														IF	ID	EX	ME	WB				

4.18.5 What is the speedup of going from a 1-issue processor to a 2-issue processor from Figure 4.69? Use your original code for both 1-issue and 2-issue, and assume that 1,000,000 iterations of the loop are executed. As in 4.18.2, assume that the processor has perfect branch predictions, and that a 2-issue processor can fetch any two instructions in the same cycle.

```
        add   R5, $zero, $zero
Again:  beq   R5, R6, End
        sll   $t0, R5, 2
        add   R10, R1, $t0
        lw    R11, 0(R10)
        lw    R10, 4(R10)
        sub   R10, R11, R10
        add   R11, R2, $t0
        sw    R10, 0(R11)
        addi  R5, R5, 2
        beq   R0, R0, Again
End:
```

For 2-issue, each iteration after the first takes 10 cycles to execute (add R10 – lw R11 +1, LW R10 – sub R10 +2, add R11 – sw R10 +1). Pipeline fill is negligible because the loop is repeated 1,000,000 times. For 1-issue, the loop will take 11 cycles to execute (10 instructions + 1 cycle stall due to load followed by a use).

Speedup = ET1-issue/ET2-issue = (1,000,000 * 11)/(1,000,000 * 10) = 1.1

4.19 In this exercise, we consider the execution of a loop in a statically scheduled superscalar processor. To simplify the exercise, assume that any combination of instruction types can execute in the same cycle, e.g., in a 3-issue superscalar, the three instructions can be three ALU operations, three branches, three load/store instruction, or any combination of these instructions. Note that this only removes a resource constraint, but data and control dependences must be still be handled correctly. Problems in this exercise refer to the following loop:

```
Loop:  lw    $1, 40($6)
       add   $5, $5, $1
       sw    $1, 20($5)
       addi  $6, $6, 4
       addi  $5, $5, -4
       beq   $5, $0, Loop
```

(a) Unroll this loop so that four iterations of it are done at once and schedule it for a 2-issue static superscalar processor. Assume that the loop always executes a number of iterations that is a multiple of 4. You can use registers **$10** through **$20** when changing the code to eliminate dependences.

Unrolled

```
Loop:  lw    $1, 40($6)
       add   $5, $5, $1
       sw    $1, 20($5)
       lw    $1, 44($6)
       add   $5, $5, $1
       sw    $1, 16($5)
       lw    $1, 48($6)
       add   $5, $5, $1
       sw    $1, 12($5)
       lw    $1, 52($6)
       add   $5, $5, $1
       sw    $1, 8($5)
       addi  $6, $6, 16
       addi  $5, $5, -16
       beq   $5, $0, Loop
```

Cycle		
1	lw $1, 40($6)	lw $10, 44($6)
2	lw $11, 48($6)	addi $6, $6, 16
3	lw $12, 36($6)	add $5, $5, $1
4	sw $1, 20($5)	add $5, $5, $10
5	sw $10, 16($5)	add $5, $5, $11
6	sw $11, 12($5)	add $5, $5, $12
7	sw $12, 8($5)	addi $5, $5, -16
8	beq $5, $0, Loop	

(b) Reschedule it for a 2-issue processor like the one in the book which can issue one memory and one non-memory instruction in each cycle.

Cycle		
1	lw $1, 40($6)	addi $5, $5, -16
2	lw $10, 44($6)	addi $6, $6, 16
3	lw $11, 32($6)	
4	lw $12 , 36($6)	add $5, $5, $1
5	sw $1, 36($5)	add $5, $5, $10
6	sw $10, 32($5)	add $5, $5, $11

7	`sw $11, 28($5)`	`add $5. $5, $12`
8	`sw $12, 24($5)`	`beq $5, $0, Loop`

5.1 In this exercise we look at memory locality properties of matrix computation. The following code is written in C, where elements within the same row are stored contiguously. Assume each word is a 32-bit integer.

```
for (I = 0; I < 8; I++)
  for (J = 0; J < 8000; J++)
    A[I][J] = B[I][0] + A[J][I];
```

5.1.1 How many 32-bit integers can be stored in a 16-byte cache block?
16 bytes x 1 integer/4 bytes = 4 integers

5.1.2 References to which variables exhibit temporal locality?
I and J are used many times successively for address calculation, they exhibit temporal locality. B[I][0] is used 8000 times in a row, it also exhibits temporal locality. A[I][J] and A[J][I] access different locations each time, no temporal locality.

5.1.3 References to which variables exhibit spatial locality?

The variable A[J][I] does not exhibit spatial locality. The accesses occur A[0][0], A[1][0], A[2][0], … and do not appear sequentially in memory. The variable A[I][J] does exhibit spatial locality. The accesses occur A[0][0], A[0][1], A[0][2], … and do appear sequentially in memory. The variables I and J have no relationship to each other so no spatial locality. The variable B[I][0] does not have spatial locality as the accesses occur B[0][0], B[1][0] and do not appear sequentially in memory.

Locality is affected by both the reference order and data layout. The same computation can also be written below in Matlab, which differs from C by storing matrix elements within the same column contiguously in memory.

```
for I = 1:8
  for J = 1:8000
    A(I,J) = B(I,1) + A(J,I)
  end
end
```

5.1.4 How many 16-byte cache blocks are needed to store all 32-bit matrix elements being referenced?
A: 8000 x 8000 x 4/16 = 16,000,000, B: 8 x 1 x 4/16 = 2, Total 16,000,002

5.1.6 References to which variables exhibit spatial locality?

The variable A(J,I) does exhibit spatial locality. The accesses occur A(1,1), A(2,1), A(3,1), … and appear sequentially in memory. The variable A(I,J) does not exhibit spatial locality. The accesses occur A(1,1), A(1,2), A(1,3), … and do appear sequentially in memory. The variables I and J have no relationship to each other so no spatial locality. The variable B(I,1) does have spatial locality as the accesses occur B(1,1), B(2,1), … and do appear sequentially in memory.