EE116C midterm 2 sol

Yuan Liang

Problem 1. (15 points)

Explain terms or answer short problems. For example, Program counter (PC) is the register containing the address of the instruction in the program being executed. (If you do not know how to explain the term precisely, you may use examples to explain).

(1) Explain the concept of speculation in the scene of pipelining, and explain how it works by using ONE example.

Guess instruction/data to be fetched when it is not decided yet.
beq rs, rt, there
add r1, r2, r3
there: add r3, r2, r1
do add r1, r2, r3 when branching result is not settled, and flush the results if wrong.

(2) Explain the concept of loop unrolling and why we perform loop unrolling

Unroll the loop n times and rename the registers to make a big loop. Loop unrolling leads to more instructional level parallelism and therefore improve performance.

(3) Name three techniques (**in hardware**) to resolve data hazard or reduce performance loss of data hazard.

stall
forwarding
register transparent

(4) Name three techniques (**in either software or hardware**) to resolve branch hazard or reduce performance loss of branch hazard.

stall
early detection
speculation

(5) A single-cycle implementation may be divided into five stages for pipelining. Compare the average CPI between single-cycle and ideal pipelining implementations and explain why pipelining may improve performance.

The average CPI of both single-cycle and ideal pipelining implementations is 1. But the critical path length of the single-cycle implementation is much larger (usually N times larger, where N is the number of pipeline stages, N=5 in the problem) than the ideal pipeline. Therefore, pipelining can improve performance.

Problem 2. **(15 points)**
Consider the three properties of execution time. For the following design decisions, indicate how these three properties could impact a MIPS processor running an application A with the following instruction breakdown and latencies:

Instruction	% of Instructions in *A*	Instruction Latency (cycles)
Load	20%	5
Store	10%	4
Simple R-type (i.e. adds, ands, shifts)	45%	4
Multiply	10%	5
BEQ/BNE	10%	3
Jump	5%	3

Note that this architecture is not the single cycle datapath and is not pipelined. Your answer for each should be that it either could increase, could decrease, or will stay the same. You may only circle one answer per property.

a. Design Decision: Increase the size of the immediate field for I-type instructions by reducing the number of registers in the ISA.

- Instruction Count could increase could decrease will stay the same
- Cycle Time could increase could decrease will stay the same
- CPI could increase could decrease will stay the same

Justification for "could increase":
The decrease in the amount of registers could mean more register spilling. This could increase the amount of lw and sw instructions, increasing the overall instruction count.

Justification for "could decrease":
The increase in the size of the immediate field of I-Type instructions means more flexibility in some instructions:
- addi can now add larger immediates
- beq/bne can now branch farther
- lw/sw can now calculate a farther offset.

It is possible that the original code often needed to explicitly create large immediates (case 1):

```
lui $t0, UPPER
ori $t0, $t0, LOWER
add $t1, $t1, $t0
```

...or needed to branch to distant locations using j (case 2):

```
bne $t0, $t1, LABEL
j ADDR
LABEL:
```

...or both (case 3):

```
lui $t0, UPPER
ori $t0, $t0, LOWER
bne $s0, $s1, LABEL
jr $t0
LABEL:
```

...or in a pathological case, used multiple branches to get to a distant destination (case 4):

```
beq $t0, $t1, LABEL1
...
LABEL1:
beq $t0, $t1, LABEL2
```

Case 1 can be reduced to a single addi and the remaining cases could be reduced to a single beq if the new size of the immediate field is large enough to accommodate the desired immediate. As a result, the IC could be reduced. Note, according to the instruction type distribution, there are no I-Type instructions other than lw, sw, and beq. Thus, for this particular set of instructions, this justification is most valid when concerning case 2 (reduce the number of jumps) or case 4 (reduce the number of branches).

Cycle Time: Could decrease

Justification for "could decrease":

Due to the reduced number of registers, the hardware complexity is reduced which may decrease the latency of accessing the register file. If the stage in which register access occurs was previously the stage with the greatest latency, the faster register access could
result in a reduction of cycle time.

a. Design Decision: Increase the size of the immediate field for I-type instructions by reducing the number of registers in the ISA.

- Instruction Count could increase could decrease will stay the same
- Cycle Time could increase could decrease will stay the same
- CPI could increase could decrease will stay the same

CPI: Could increase

Justification for "could increase":
Due to the decrease in registers, there could be more register spilling, resulting in a higher proportion of loads (CPI = 5) and stores (CPI = 4). The current CPI of the instructions is 4.15. If the result of the increased spilling is the addition of new lw/sw pairs, then the CPI will increase (adding a set of instructions that have an average CPI of 4.5).

Justification for "could increase":
Due to the increased immediate field, there may be a reduction in the number of the following instructions:
- Simple R-Type, CPI = 4 (case 1)
- Non-branch/lw/sw I-Type (addi, ori, lui)
- beq/bne, CPI = 3 (case 4)
- j, CPI = 3 (case 2)

Because the current CPI is 4.15, reducing the number of any one of these would increase the CPI.

Justification for "could decrease":
Consider a case where the original datapath had multiple instruction decode stages. If the reduced number of registers means that multiple decode stages could be reasonably coalesced into one stage due to the decreased complexity, the CPI could decrease due to a reduction of stages in the datapath.

b. Design Decision: A new physical design technology is used which can reduce wire latency.

- Instruction Count could increase could decrease will stay the same

- Cycle Time could increase could decrease will stay the same
- CPI could increase could decrease will stay the same

Instruction Count: Will stay the same
Justification for "will stay the same":
This is purely a change to the hardware. Since the instruction count exists in the software domain, a change to the hardware cannot affect the instruction count unless the instructions/software explicitly specify that particular hardware component (ex. instructions specify registers).

Cycle Time: Could decrease
Justification for "could decrease":
Because the wire latency has decreased, the latency through each stage of the data path would likely decrease. As a result, the cycle time could decrease to account for the reduced latency of each stage.

CPI: Could decrease
Justification for "could decrease":
If the reduced wire latency means that multiple stages in the datapath could be coalesced into a single stage as part of the hardware modification, the CPI could be reduced.

c. Design Decision: A new compiler is used to compile application A. The compiler uses shifts in place of multiply instruction – it takes multiple shift instructions to replace a single multiply.

- Instruction Count could increase could decrease will stay the same
- Cycle Time could increase could decrease will stay the same
- CPI could increase could decrease will stay the same

Instruction Count: Could increase
Justification for "could increase":
Since a single multiply is replaced by several shifts, the number of instructions will increase if this multiply replacement is done.

Cycle Time: Could decrease, must stay the same
Justification for "could decrease":
If we make the assumption that not only are we using a different compiler that eliminates multiply instructions, but we are also using different hardware that doesn't include the multiply hardware, the complexity of the hardware is reduced. If the multiply hardware was part of the stage with the maximum latency, then eliminating the multiply hardware could result in a reduced cycle time.

Justification for "must stay the same":
If we make the assumption that ONLY the compiler has changed, then the hardware is unchanged and therefore the cycle time is also unchanged.

CPI: Could decrease
Justification for "could decrease":
Replacing each multiply with multiple shifts has the following effects:
- Reduces the number of multiply instructions which have a CPI of 5
- Increases the number of shift instructions which have a CPI of 4

Since the current CPI is 4.15, both of these changes will bring the overall CPI closer to 4, which will reduce the CPI. Justification for "could decrease": As with the possible justification for why CT may decrease, the reduced complexity of not requiring multiply hardware may mean if the EX stages can be reduced or coalesced, the CPI decreases.

Problem 3 (15 points):

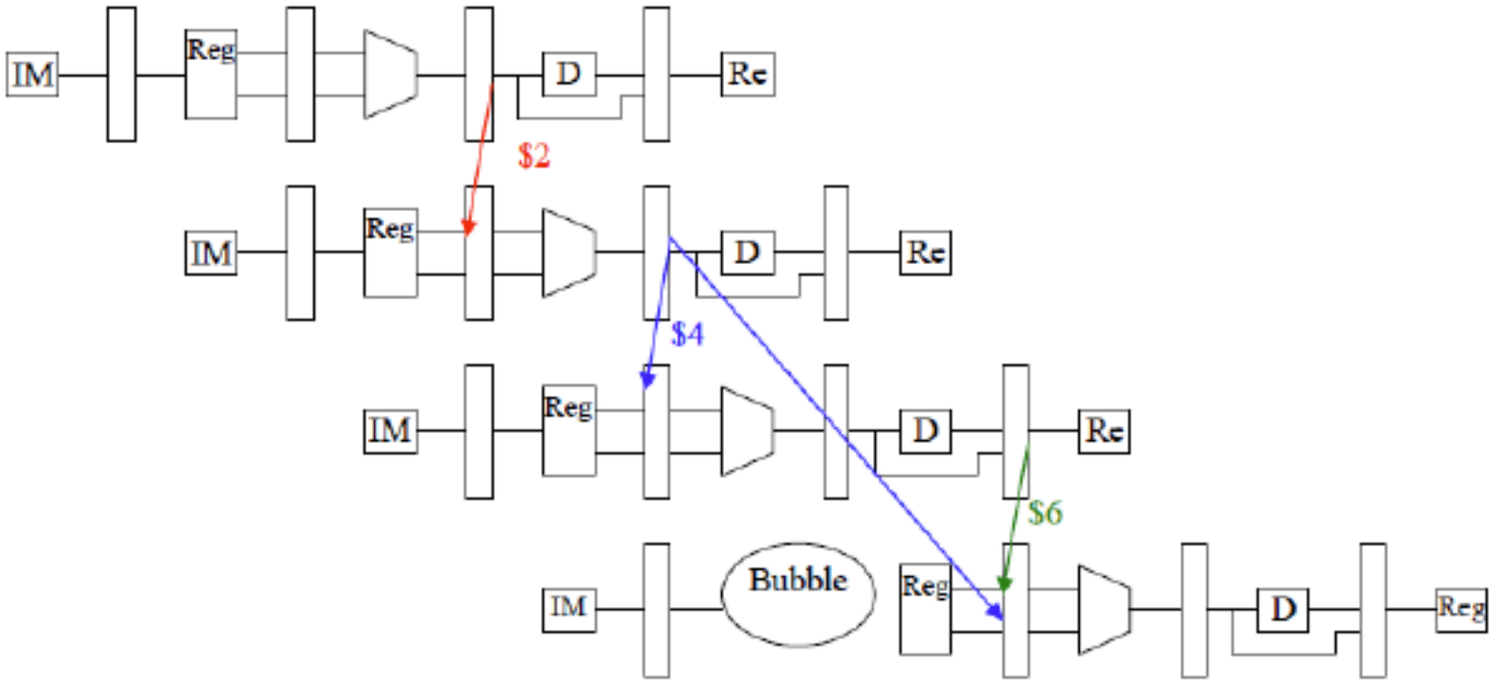
Assume that we have a five-stage machine same as the one in textbook. For the following code:

(a) sub $2, $5, $4
(b) add $4, $2, $5
(c) lw $2, 100($4)
(d) add $5, $2, $4

(1) Name all data dependencies (3 points)

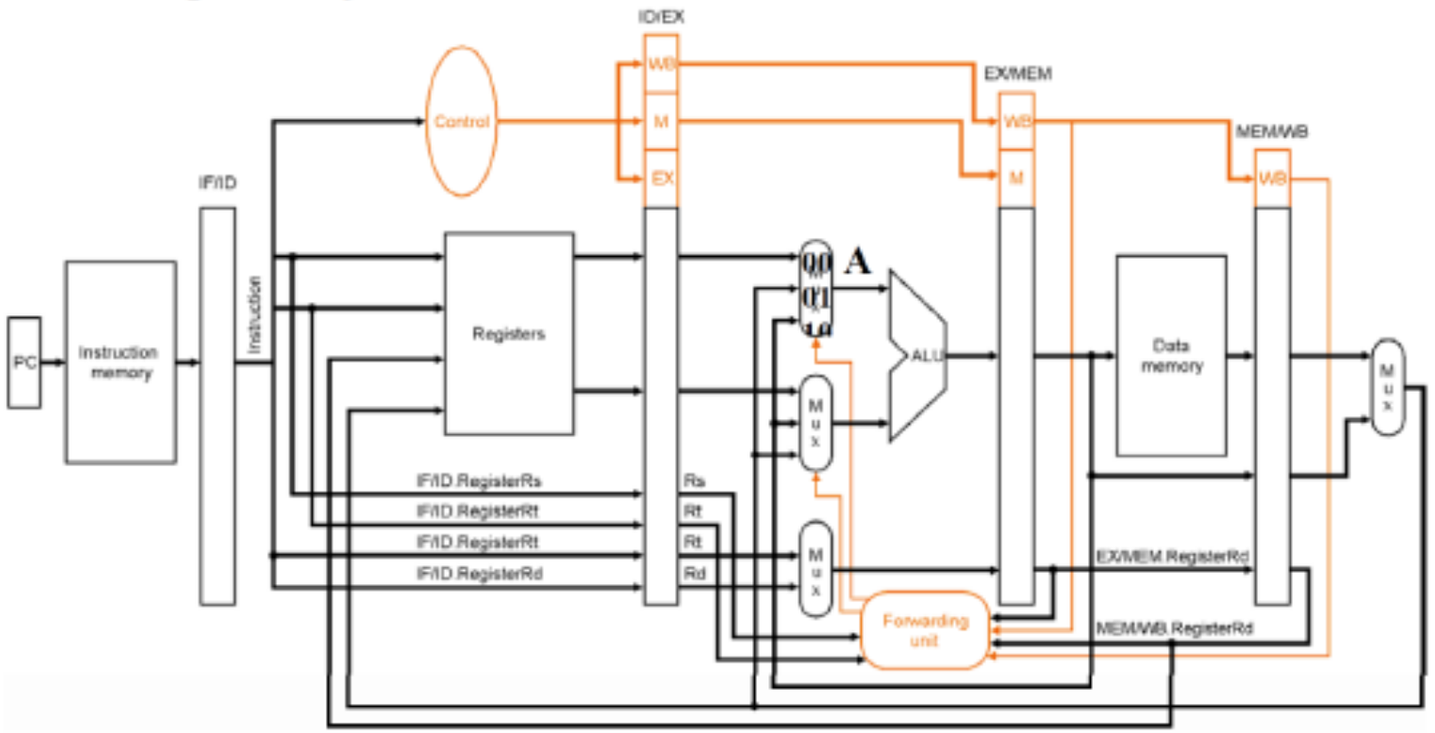
(a) sub $2, $5, $4
(b) add $4, $2, $5 **$2 depends on (a)**
(c) lw $2, 100($4) **$4 depends on (b)**
(d) add $5, $2, $4 **$2 depends on (c), $4 depends on (b)**

(2) After renaming, which data hazard can be resolved via forwarding? Illustrate all the forwarding using 5-stage by using pipeline diagram similar to those in the lectures. (7 points)



Problem 4 (15 points)

Considering data forwarding for the pipeline below, state how to generate control signals for MUX A. I.e., use plain English AND logic function such as EX MEM.RegisterRd != 0 to explain when the control signal for MUX A should be 00, 01 and 10 respectively.



I. control signal = 00

No data forwarding.
Neither condition below holds.

II. control signal = 01

Forward result from MEM/WB register.
If ((MEM/WB.RegWrite) &&
(IF/ID.RegisterRs != EX/MEM.RegisterRd) &&
(EX/MEM.RegWrite) // this condition is not in the book
(IF/ID.RegisterRs == MEM/WB.RegisterRd) &&
(IF/ID.RegisterRs != 0))

III. control signal = 10

Forward result from EX/MEM register.
If ((EX/MEM.RegWrite) &&
(IF/ID.RegisterRs == EX/MEM.RegisterRd) &&
(IF/ID.RegisterRs != 0))

Problem 5 (20 points):
Consider the 2-way superscalar processor we covered in class – a five stage pipeline where we can issue one ALU or branch instruction along with one load or store instruction every cycle. Suppose that the branch delay penalty is two cycles and that we handle control hazards with branch delay slots (since the penalty is two cycles, and this is a 2-way superscalar processor, that would be four instructions that we need to place in delay slots).
This processor has full forwarding hardware. This processor is a VLIW machine. How long would the following code take to execute on this processor assuming the loop is executed 200 times? Assume the pipeline is initially empty and give the time taken up until the completed execution of the instruction sequence shown here.

```
Loop:   lw $t0, 0 ($s0)
        lw $t1, 0 ($t0)
        add $t1, $s1, $t1
        sw $t1, 0 ($t0)     # you may assume that this store never goes to the same address as the first load
        addi $s0, $s0, 4
        bne $s0, $s2, Loop
```

1. **First you will need to schedule (i.e. reorder) the code (use the table below) to reduce the total number of cycles required (but don't unroll it…yet).**

Total # of cycles for 200 iterations: _____1204_____
(Hint – schedule the code first for one iteration, then figure out how long it will take the processor to run 200 iterations of this scheduled code)

Cycle	1st Issue Slot (ALU or Branch)	2nd Issue Slot (LW or SW)
1	addi $s0, $s0, 4	lw $t0, 0($s0)
2		
3		lw $t1, 0($t0)
4	bne $s0, $s2, loop	
5	add $t1, $s1, $t1	
6		sw $t1, 0($t0)
7		
8		
9		
10		
11		
12		
13		

2. Now unroll the loop once to make two copies of the loop body. Schedule it again and record the total # of cycles for 200 iterations:

_____804_____

Cycle	1st Issue Slot (ALU or Branch)	2nd Issue Slot (LW or SW)
1	addi $s0, $s0, 8	lw $t0, 0($s0)
2		lw $t2, -4($s0)
3		lw $t1, 0($t0)
4		lw $t3, 4($t2)
5	add $t1, $s1, $t1	
6	bne $s0, $s2, loop	sw $t1, 0($t0)
7	add $t3, $s1, $t3	
8		sw $t3, 0($t2)
9		
10		
11		
12		
13		