

Name:

**CSc 656 Midterm
3/28/2018**

Problem 1:

Show the timing of this instruction sequence through the MIPS pipeline:

```
sub    $10, $10, $14
lw     $26, 0($10)
or     $25, $26, $16
add    $3, $13, $25
```

You can use either notation for showing the timing of instructions through the pipeline. Assume that all possible data forwarding is available. For every instance/cycle where the pipeline has to forward data or stall, show how ForwardA or ForwardB are set. (You don't have to show the equations.) (25 points)

```
ForwardA = 01    forward from MEM/WB to upper input of ALU
ForwardA = 10    forward from EX/MEM to upper input of ALU
ForwardB = 01    forward from MEM/WB to lower input of ALU
ForwardB = 10    forward from EX/MEM to lower input of ALU
```

ANS:

	1	2	3	4	5	6	7	8	9
sub \$10, \$10, \$14	IF	ID	EX	MEM	WB				
lw \$26, 0(\$10)		IF	ID	EX	MEM	WB			
or \$25, \$26, \$16			IF	ID	st	EX	MEM	WB	
add \$3, \$13, \$25				IF	st	ID	EX	MEM	WB

Cycle 4: ForwardA = 10

Cycle 6: ForwardA = 01

Cycle 7: ForwardB = 10

[this page left intentionally blank for work]

Problem 2:

Consider the following loop:

```
loop:    lw      $10, 0($18)
         add     $12, $12, $10
         sw      $12, 0($22)
         addi    $18, $18, -4
         addi    $22, $22, -4
         bne     $18, $13, loop
```

This loop reads an array of integers, sums them, and stores the partial sums in another array. The overall sum is stored in \$12. (\$12 is initialized outside the loop; you don't have to worry about the initialization code.)

Unroll the loop 3 times (i.e., 4 old iterations become one new iteration). Assume that the number of iterations is divisible by 4. Optimize the code; reduce unnecessary arithmetic operations, and reschedule the instructions to minimize stalls in a 5-stage MIPS pipeline. You don't have to show the timing.

Make sure that the arrays and \$12 contain the correct results when the loop exits.

(30 points)

ANS:

```
loop:  lw      $10, 0($18)
        lw      $1, -4($18)
        lw      $2, -8($18)
        lw      $3, -12($18)

        add     $4, $12, $10
        add     $5, $4, $1
        add     $6, $5, $2
        add     $12, $6, $3

        sw      $4, 0($22)
        sw      $5, -4($22)
        sw      $6, -8($22)
        sw      $12, -12($22)

        addi    $18, $18, -16
        addi    $22, $22, -16
        bne     $18, $13, loop
```

Problem 3:

Assume a branch predictor, with an array of 16 2-bit predictors with the state diagram given in the provided figure (this is the same state diagram from your textbook). As discussed in lecture, the low bits of the PC of a branch (after removing the least significant 2 zeros) are used to index into the array of 2-bit predictors. ***Before the trace begins, all 16 2-bit predictors are in state 10.***

You are given the following trace with several conditional branches. The branch behavior is already given. Fill out the table. Show work clearly to indicate how you computed each entry; points will be deducted for lucky guesses. (30 points)

Address of branch	Current behavior	Predictor Index	Current State	Next State	Current prediction	Prediction Correct?
0x8314	NT	0101	10	01	T	N
0x89d4	NT	0101	01	00	NT	Y
0x8e24	T	1001	10	11	T	Y
0x8314	T	0101	00	01	NT	N
0x82a4	NT	1001	11	10	T	N

Note:

Current behavior: whether current branch is taken/not taken

Predictor index: index of current branch predictor accessed by branch

Current state: state of branch predictor accessed (00, 01, 10, or 11)

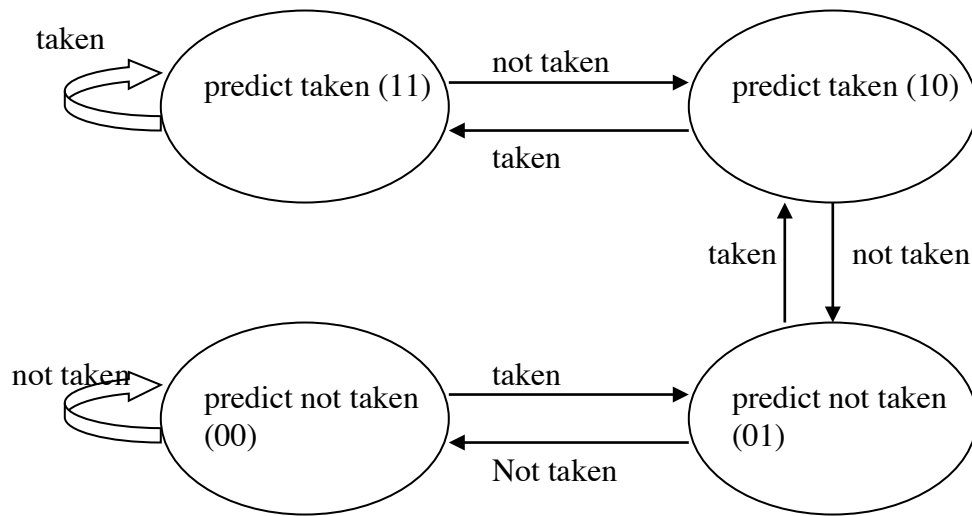
Current prediction: taken or not taken (this follows directly from current state)

Predictor index: 2 ea.

Current state: 1 ea.

Next state: 1 ea.

Prediction: 1 ea.



2-bit Branch predictor state diagram

Short Problem 2 (8 points): In 30 or fewer words, define what is a *load/store architecture*. (Good interview question.)

ANS:

See Chapter 2 Slide 3, 256 slides, etc

Short Problem 1 (7 points): For the MIPS 5-stage pipeline, show an instruction sequence and timing diagram in which 1) an earlier instruction causes a bad data address exception, and a later instruction has an invalid opcode exception, and 2) the invalid opcode occurs in an earlier cycle than the bad data address exception. (That is, the two exceptions occur out of order.) Identify the relevant instructions clearly, show where each exception occurs and with which instruction, and show timing clearly.

ANS:

See Chapter 4 Slide 88

[rs, rt, rd are any registers. I is a 16-bit constant.]

add rd, rs, rt	rd = rs + rt
sub rd, rs, rt	rd = rs - rt
addu rd, rs, rt	rd = rs + rt (overflow ignored)
subu rd, rs, rt	rd = rs - rt (overflow ignored)
addiu rt, rs, I	rt = rs + I (sign-extended)
mult rs, rt	Hi Lo = rs * rt
div rs, rt	Hi = rs % rt
	Lo = rs / rt
mfhi rd	rd = Hi
mflo rd	rd = Lo
mthi rs	Hi = rs
mtlo rt	Lo = rs
and rd, rs, rt	rd = rs & rt
or rd, rs, rt	rd = rs rt
andi rt, rs, I	rt = rs & I (zero-extended)
ori rt, rs, I	rt = rs I (zero-extended)
sll rd, rt, I	rd = rt << I
srl rd, rt, I	rd = rt >> I
lw rt, I(rs)	rt = Mem[rs + I] (load word)
sw rt, I(rs)	Mem[rs + I] = rt (store word)
lbu rt, I(rs)	rt = mem[rs + I] (load byte zero-extended)
lb rt, I(rs)	rt = mem[rs + I] (load byte sign-extended)
sb rt, I(rs)	Mem[rs + I] = rt (store byte)
beq rs, rt, label	if (rs == rt) goto label
bne rs, rt, label	if (rs != rt) goto label
slt rd, rs, rt	if (rs < rt) rd = 1; else rd = 0
slti rt, rs, I	if (rs < I) rd = 1; else rd = 0
j label	goto label
jr rs	PC = rs
jal label	\$31 = return address; goto label