Department of Electrical and Computer Engineering
University of Wisconsin – Madison

ECE 552 Introduction to Computer Architecture

Homework 5 (Due 10/13)

For the question below (verilog based), paste your entire verilog code in a single text/doc file and submit the text/doc file on Canvas. Note that these verilog questions need to follow the verilog rules specified in the 552 project/verilog rules document that can be found on Canvas.

1. **Program Counter Control**

   In this homework, you will focus on the Branch (**B**) instruction in the WISC ISA (refer to project description) to modify the 16-bit PC. You will need only combinational logic in this module.

   Branch instructions conditionally jump to the address obtained by adding the 9-bit immediate (signed) offset to PC+2.

   You are provided with 3 inputs: a 3-bit condition (C), a 9-bit offset (I) and a 3-bit flag (F).

   'C' specifies the condition as in Table 1 and 'I' represents the 9-bit signed offset in two's-complement representation, but right-shifted by one. The target is computed as: target = PC + 2 + ('I' << 1).

   Note: More details on why you do PC+2 and right-shift 'I' by 1 are provided in the project description.

   Table 1: Encoding for Branch conditions

ccc	***Condition***
000	Not Equal (Z = 0)
001	Equal (Z = 1)
010	Greater Than (Z = N = 0)
011	Less Than (N = 1)
100	Greater Than or Equal (Z = 1 or Z = N = 0)
101	Less Than or Equal (N = 1 or Z = 1)
110	Overflow (V = 1)
111	Unconditional

   The eight possible conditions are Equal (EQ), Not Equal (NEQ), Greater Than (GT), Less Than (LT), Greater Than or Equal (GTE), Less Than or Equal (LTE), Overflow (OVFL) and Unconditional (UNCOND).

   As you noticed above, many of these conditions are determined based on the flags N, V and Z. Flag bits are stored in a 3-bit flag (F) register. There are three bits in 'F': Zero (Z), Overflow (V) and Sign bit (N). In your project, these flags will be set by arithmetic instructions. But for this homework, you assume that the 'F' is an input to your design. The True condition corresponds to the branch being taken. The False condition results in normal PC increment i.e., PC + 2.

   The required module is listed below:

   module **PC_control**(input C[3], input I[9], input F[3], input PC_in[16], output PC_out[16]);

The following questions are non-verilog. Submit them as a text doc or a snapshot.

2. **Branch Prediction**

Consider the following MIPS code with three branches at addresses 0x104, 0x10c and 0x130:

```
0x100       LOOP:       lw    $s1, 0($s2)
0x104                   beq   $s1, $s0, DECR
0x108                   addi  $s1, $s1, 1
0x10c                   beq   $s0, $s0, STORE
            DECR:
0x120                   addi  $s1, $s1, -1
            STORE:
0x124                   sw    $s1, 0($s2)
0x128                   addi  $s2, $s2, 4
0x12c                   sub   $s4, $s3, $s2
0x130                   bne   $s4, $s0, LOOP
```

Assume the program executes and produces the sequence of branches in the table below. The processor's branch predictor employs a branch history table (BHT) with 8 entries, each containing a 1-bit prediction (1=taken). BHT entries are indexed using bits 4 to 2 of the PC of the branch instructions. Assume that all BHT entries are initially set to 0=not taken.

(a) Fill in the BHT entries and predictions below.

Instruction PC	Taken (T) or not (N)?	BHT index (binary)	BHT entry value before the prediction	Prediction	BHT entry value after the branch executes
0x104	T	001	0	N	1
0x130	T				
0x104	N				
0x10c	T				
0x130	T				
0x104	T				
0x130	T				
0x104	N				
0x10c	T				
0x130	T				

(b) What is the branch misprediction rate?

3. **Control Hazards**

Assume the five-stage pipeline with full forwarding and bypassing. Assume that branch decisions are made at the ID stage of the pipeline and that **data forwarding** from the EX stage is available to the branch decision circuit at the ID stage. Assume no branch delay slot. Use a predict-not-taken policy: until the branch decision is resolved, instructions on the branch-not-taken path continue to be fetched and are then **flushed** if the branch is found to be taken. Consider the following two code segments:

Code Segment A	Code Segment B

`L1: lw  $t1, 0($t2)`
`L2: lw  $t3, 0($s2)`
`L3: beq $t1, $t3, L1`
`L4: sub $t4, $t2, $s1`
	`L1: add $t1, $s0, $s1`
	`L2: add $t1, $s0, $t1`
	`L3: beq $t1, $0, L1`

(a) For code segment A, complete the pipeline timing diagram until L4 is executed. Use "*" to indicate an instruction stalling (e.g., "D*" implies stalling in decode) and "=" to indicate an instruction getting flushed. Assume the branch is taken the first time and not taken the next time.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
L1	F	D	X	M	W												

(b) For code segment B, identify all the data dependences. For each one, specify if it leads to a data hazard, if forwarding is needed and if any stall cycles are needed.