Computer Architecture ELE 475
Problem Set #3

This problem set is **ungraded and not collected**. We will release solutions two weeks after the problem set is released. Please stop by office hours if you have questions.

Problem #1 (20 Points): For this problem, assume a VLIW processor with three integer units (X, Y, Z), one multiply unit (M), and two load-store units (LS0, LS1). ALU instructions have a latency of 1, multiply instructions have a latency of 5, and loads have a latency of 3. One branch can execute per cycle and executes in the Z pipeline. For the following code, fully unroll and software pipeline the code to improve performance. Show the prolog and epilog code. Feel free to use more registers, reorder code, and rename registers for performance. What high order functionality does the function implement? What is the average rate of multiplies per cycle in the middle of the loop?

```
//R0 is hardwired to zero
//R31 is the link register
//R1 contains pointer to input array (elements are word sized)
//R2 contains pointer to output array (elements are word sized)
//    (Output array is larger than input array)
//R3 contains the size of the input array in words
function:
ADDI R16, R0, 0x456
ADDI R17, R0, 0x789
ADDI R18, R0, 0x901
ADD R4, R0, R0
ADD R5, R0, R0
loop:
LW R6, 0(R1)
MUL R8, R4, R16
MUL R9, R5, R17
MUL R10, R6, R18
ADDI R1, R1, 4
SUBI R3, R3, 1
ADD R8, R8, R9
ADD R8, R8, R10
ADD R4, R5, 0
ADD R5, R6, 0
SW R8, 0(R2)
ADDI R2, R2, 4
BNEZ R3, loop
MUL R8, R4, R16
MUL R9, R5, R17
ADD R8, R8, R9
SW R8, 0(R2)
ADDI R2, R2, 4
ADD R4, R5, 0
MUL R8, R4, R16
SW R8, 0(R2)
JR R31
```

Problem #2 (10 Points): For this problem, assume a VLIW processor with three integer units (X, Y, Z), one multiply unit (M), and two load-store units (LS0, LS1). ALU instructions have a latency of 1, multiply instructions have a latency of 5, and loads have a latency of 2. One branch can execute per cycle and executes in the Z pipeline. The following code has been bundled assuming the EQ scheduling model. What is the value of R12, R13, and R14 after this code executes? Not changing register names, reschedule this code assuming the LEQ model. Why is the LEQ model more flexible?

```
{ADDI R9, R0, 9; ADDI R10, R0, 10;}
{ADDI R6, R0, 6; ADDI R8, R0, 8; ADDI R5, R0, 5;}
{LW R6, 0(R7); LW R8, 4(R7);}
{ADDI R12, R6, 1; ADDI R13, R8, 2;}
{MUL R7, R6, R9;}
{MUL R5, R8, R10;}
{LW R14, 8(R7);}  // Assume that 8(R7) contains the value 0x1
{ADD R15, R16, R17;}
{ADD R14, R14, R5;}
{SUB R19, R18, R22;}
{ADD R5, R7, R5;}
```

Problem # 3 (10 Points): Rewrite the following code assuming the instruction set has been augmented with conditional move instructions, movz and movn. Assume that the branch misprediction penalty is 10 cycles and that the branch to “forward” is random and data-dependent. Does it make sense to predicate? Movz and movn have the following semantics:

```
movz rd, rs, rt         if ( R[rt] == 0 ) then R[rd] <- R[rs]
movn rd, rs, rt         if ( R[rt] != 0 ) then R[rd] <- R[rs]
```

Code Sequence for problem 3:

```
ADDI R6, R0, 1
ADDI R3, R0, 50
loop:
LW R8, 0(R9)
BEQZ R8, forward
ADD R12, R15, R8
SUB R24, R24, R12
J done
forward:
ADDI R24, R24, 10
done:
SUBI R3, R3, 1
BNEZ R3, loop
```

Problem #4 (20 Points): Assume a computer architecture with a 1024-entry Branch History Table (BHT). Each BHT entry is a two bit predictor that implements a two-bit saturating counter that starts in the strong not-taken state. After the following code execution, what is the state of the BHT? What is the prediction accuracy of each branch?

```
p_4:
// Assume that at address 0x1000, there is an array of word sized integers
// with the following data [0x0, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1]
ADDI R7, R0, 7
ADDI R11, 0x1000
loop:
SUBI R7, R7, 1
ANDI R8, R7, 1
b_0:
BEQZ R8, l_1
LW R12, 0(R11)
b_1:
BEQZ R12, l_2
l_1:
ADD R9, R9, R16
l_2:
ADDI R11, R11, 4
b_3:
BNEZ R7, loop
```

Problem #5 (20 Points): Assume a computer architecture has a 11-bit global Branch History Register (BHR) that is used to index into a 2048-entry Pattern History Table (PHT), where each PHT entry is a single bit predictor. Assume that function p_4 in the code for problem 4 (above) is executed 50 times. On the 51st execution, what is the branch prediction accuracy for the branch b_1? Draw the final state of the global BHR.

Problem #6 (20 Points): Branch Target Buffers (BTBs) are used to determine the address of the target of branch or jump. At aggressive clock frequencies why are BTBs used? When is the destination of a branch typically know in a 5 stage pipeline with a dedicated branch address adder? Does a branch address adder help for JALR? Does a BTB help with determining the target of a JALR?