**Worksheet 7**

**Q1. Branch Prediction**

Consider the following code, a loop which alternately takes one of two paths each iteration.

while(1){

if((i & 1) == 0) {

// path A

} else {

// path B

}

i++;

}

The assembly code is as follows. The variable “i” is kept in register x1.

loop: andi x2, x1, 1

addi x1, x1, 1

bnez x2, B

// path A

j loop

B: // path B

j loop

1. What is the misprediction rate on the branch at steady state if we predict the branch is always not taken?
2. What is the misprediction rate if we use a BHT indexed by PC with one-bit counters?
3. What is the misprediction rate if we use a BHT indexed by PC with two-bit counters?
4. What is the misprediction rate if we use a BHT indexed by PC and one bit of global history with two-bit counters?

**Q2. VLIW and software pipelining**

We want to execute the following code on a VLIW processor.

for (i = 0; i < N; i++)

A[i] = B[i] \* C;

Where A, B, and C are arrays of double-precision floating point numbers.

The compiled assembly is shown below. The registers x1 and x2 are initialized with the pointers to A and B and the register x3 is initialized with the value N. The register f0 holds the value C.

loop:

fld f1, 0(x1)

fmul f2, f1, f0

fsd f2, 0(x2)

addi x1, x1, 8

addi x2, x2, 8

addi x3, x3, -1

bnez x3, loop

Assuming our VLIW processor has two integer pipelines with 1 cycle latency, two memory pipelines with 2 cycle latency, and a floating point multiply pipeline with three cycles of latency.

1. To fully utilize the machine, how many loop iterations can you unroll? Write the assembly for the unrolled loop. Pay attention to pointer increment and load/store immediate value.
2. Fill the cycle diagram below. Make sure to show the preamble and epilogue. Assume you have an unlimited number of floating-point registers and that N is evenly divisible by whatever number you unroll the loop by.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Label | I0 | I1 | M0 | M1 | F\* |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |