CS/ECE 6810: Example Midterm Exam

Notes: The actual midterm exam will be an open notes and open book exam. You will be allowed to use calculators if necessary. Laptops will be allowed only for browsing through class material (textbook CD, notes, or slides), but you will **NOT** be allowed to access other websites.

1. **(Pipelining)** Consider an unpipelined processor where it takes 20ns to go through the circuits and 0.2ns for the latch overhead. Assume that the Point of Production and Point of Consumption in the unpipelined processor are separated by 10ns. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction. What is the throughput of the processor (in BIPS) for (i) an unpipelined processor, and (ii) a 10-stage pipeline?

2. **(Power and Energy)** Consider a system where the processor is running at its peak frequency of 3 GHz and executing a CPU-bound program. At this peak frequency, the system dissipates 100 W of power, of which 40 W is dissipated as dynamic power in the processor, 30 W is dissipated as leakage in the processor, and 30 W is dissipated in other system components (monitor, keyboard, power supply, etc.). The processor power can be lowered with DVFS, with voltage and frequency being scaled proportionally, i.e., if voltage is lowered by 10%, frequency is also lowered by 10%. Assume that voltage and frequency can be made arbitrarily low. Processor DVFS only impacts dynamic and leakage power in the processor, and does not impact power in other system components. At what frequency does the system consume the least energy to execute the program?

3. **(Forwarding/Bypassing)** Consider the following in-order pipeline:

```
BP :IC :DEC:DEC:RR :IntAdd :WB
                   :Effadd :DC :DC :WB
                   :FPA1 :FPA2 :FPA3 :FPA4 :WB
```

The initial part of the pipeline involves branch prediction (BP), instruction cache fetch (IC), two decode stages (DEC and DEC), and register read (RR). An instruction is held back in the second decode stage if its input operands are not available. After the RR stage, Integer-adds go through “IntAdd” and “WB” (register writeback). Loads and stores go through “Effadd” (where the load/store address is calculated), then two data-cache stages, and finally the “WB” stage. Floating-point adds go through four “FPA” stages and then the “WB” stage. What are the stall cycles introduced between the following pairs of successive instructions with and without full bypassing? Assume that a register read and a register write take up an entire cycle each. (Show at least a couple of pipeline diagrams to convince me that you have understood the concepts; don’t just provide the final numbers.)

(a) FP-add, providing data for a store, e.g.,

```
ADD.D R1 <-- R2+R3 SD R1, 8(R7)
```

(b) Load, providing address for a store, e.g.,

```
LD R1, 8(R2) SD R3, 8(R1)
```

(c) Load, providing data for an FP-add, e.g.,

```
LD R1, 8(R2) ADD.D R4 <-- R1+R3
```

4. **(Branch Predictors)** Consider the following tournament branch predictor that employs a selector with 32K entries (2-bit saturating counters). The selector picks a prediction out of either a global predictor (15-bit global history is XOR-ed with 15 bits of branch PC to index into 3-bit saturating counters) or a local predictor (11 bits of branch PC index into level-1, 12 bits of local history are concatenated with 2 bits of branch PC to generate the index into level-2 that has 2-bit saturating counters). What is the total capacity of the entire branch prediction system?

5. **(Load and Stores)** The table below lists a sequence of loads and stores in the LSQ, when their one/two input operands are made available, and their computed effective addresses. Estimate when the address calculation happens for each ld/st and when each ld/st accesses the data memory. The processor assumes that loads do not depend on prior stores and issues loads speculatively. If the speculation is incorrect, the load must be re-issued. Assume that there are enough memory ports that you don’t have to worry about structural hazards.

LD/ST	The register for the address calculation is made available	The register that must be stored into memory is made available	The calculated effective address	Address calculation happens	Data memory accessed
LD	5	-	0xabcd		
ST	7	4	0xabbb		
LD	1	-	0xabbb		
LD	3	-	0xabcd		
ST	12	17	0xabbb		
LD	2	-	0xabbb		

6. **(OoO Execution)** Consider an out-of-order processor that has 32 architected (logical) registers and 36 physical registers. On power up, assume that logical register LR1 is mapped to physical register PR1, LR2 is mapped to PR2, and so on. On power up, the following program starts executing:

```
ADD LR1, LR1, LR2
ADD LR3, LR1, LR1
LD LR2, 8(LR3)
SD LR3, 16(LR4)
ADD LR3, LR3, LR1
ADD LR7, LR2, LR3
SD LR7, 32(LR4)
```

Show the renamed version of this code.

7. **(Loop Scheduling)** Consider a basic in-order pipeline with bypassing (one instruction in each pipeline stage in any cycle). The pipeline has been extended to handle FP add and FP mult. Assume the following delays between dependent instructions:
(a) Load feeding any instruction: 1 stall cycle
(b) FP MUL feeding store: 4 stall cycles
(c) Int add feeding a branch: 1 stall cycle
(d) Int add feeding any other instruction: 0 stall cycles
(e) A conditional branch has 1 delay slot (an instruction is fetched in the cycle after the branch without knowing the outcome of the branch and is executed to completion)
Below is the source code and default assembly code for a loop.

Source code

```
for (i=1000;i>0;i--) {
     x[i] = x[i] * y[i];
}
```

Assembly code

```
Loop: L.D     F1, 0(R1)
      L.D     F2, 0(R2)
      MUL     F1, F1,F2
      S.D     F1, 0(R1)
      DADDUI  R1, R1, #-8
      DADDUI  R2, R2, #-8
      BNE     R1, R3, Loop
      NOP
```

(a) Show an optimized schedule for this loop without resorting to unrolling or software pipelining.

(b) Show a correct software pipelined version of this loop (including registers, offsets, etc.).