School of Electrical and Computer Engineering
Georgia Institute of Technology

ECE4100/ECE6100/CS4290/CS6290
Moinuddin K. Qureshi, Instructor

Midterm 1, September 27, 2018

Name : Solutions.

GT Account:____________________________

Problem 1 (30 points): _______

Problem 2 (10 points): _______

Problem 3 (15 points): _______

Problem 4 (15 points): _______

Problem 5 (15 points): _______

Problem 6 (15 points): _______

Total (100 points) : ______

This exam is given under the Georgia Tech Honor Code System. Anyone found to have submitted copied work instead of original work will be dealt with in full accordance with Institute policies.

Georgia Tech Honor Pledge: "I have neither given nor received aid on this exam."

[MUST sign:] __________________________________

Note: Where needed, show all your intermediate results to receive full credit. Do all your work in this examination handout. Use the back of the exam sheets if necessary.

Note: Please be sure your name is recorded on each sheet of the exam.

GOOD LUCK!

Name:____

Problem 1 – "Potpourri" (30 points, there are five parts, each worth 6 points)

(A) We have two machines M1 and M2, both implementing the same ISA. M1 has a CPI of 1 and frequency of 2 GHz. M2 has a CPI of 2 and frequency of 3 GHz.

(1) Which machine would provide higher performance? M1

(2) What is the speedup of the machine in (1) compared to the other machine? 1.33x ($4/3$)

(B) In Lab-2 you implemented a five-stage pipeline (FDEMW) and extended it to a superscalar configuration. If we have a 3-wide superscalar pipeline, how many comparators do you need in the ID stage to perform dependency check? Similar to your lab assignment, you can assume that the register file can b written in the first half of the clock cycle and read in the second half of the clock cycle. For this problem, you can ignore the dependency check for condition codes.

The total number of comparators in the dependency check logic is: 36 + 6/12 = 42 or 48

(C) We want to design a global-history based branch predictor that can track history of 22 bits. One option is to have a Gshare based branch predictor (like the one you implemented in Lab 2). The second option is to use a perceptron predictor where each weight is tracked with 1 byte, and there is a table of 1K perceptrons selected based on instruction address. The total storage overhead for implementing Gshare predictor is 1024 KB and for perceptron is 22 KB.

(D) Register renaming can help with mitigating WAR and WAW dependencies, but not RAW dependency.

To implement register renaming, we need a structure called the *Register Alias Table (RAT)*. We have a machine with 128 registers in the physical register file. The total size of the RAT is 32 bytes. The total number of register in the architectural register file (ARF) for this machine is: 32

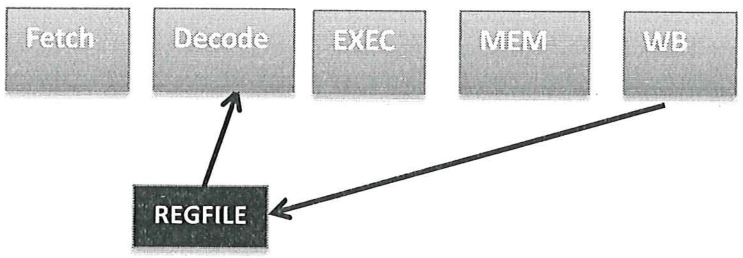
(E) We have a 64KB 2-way cache with 64-byte linesize and LRU replacement. The cache supports write-back and allocate-on-write-miss policy. What would happen to the total size (in terms of number of bits) of the tags-store of the cache, if the following changes were performed (circle one choice):

1. Increasing the associativity from 2-way to 4-way: [✓ Increase]/No-Change/Decrease
2. Making the cache write-through instead of write-back: Increase/No-Change/[✓ Decrease]
3. Make the cache no-allocate-on-write-miss: Increase/[✓ No-Change]/Decrease
4. Use random replacement instead of LRU: Increase/No-Change/[✓ Decrease]

Name:________________________________

Problem 2 “Pipe Check” (10 points)

Consider the 1-wide pipeline shown below, similar to the one you used for Lab 2. **The register file is written in the first half of the clock cycle and can be read in the second half of the clock cycle.** The machine supports full forwarding (from the EXEC and MEM stage to the Decode stage).



We are interested in knowing the timeline of execution for the code snippet with four instructions.

A. ADD R1, R0, R0
B. LOAD R2, R1, R0
C. LOAD R3, R2, R0
D. ADD R4, R3, R0

CYCLE	FETCH	DECODE	EXECUTE	MEMORY	WRITEBACK
1	A				
2	B	A			
3	C	B	A		
4	D	C	B	A	
5		•	•	B	A
6		D	C	•	B
7			•	C	•
8			D	•	C
9				D	•
10					D
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

Name:________________________________

Problem 3 "A Simple Insight" (15 points)

In class, we develop a simple superscalar model to understand some of the basic performance trade-offs. Shown below are the expected IPC for a 2-wide and 4-wide in-order machine, when the machine is assumed to have perfect branch prediction and perfect caches.

	In-order	Out-of-order
IPC-Perf	**1.6**	1.9

Lets assume that the baseline machine is two-wide in-order with 20-stage pipeline. We are concerned with a workload that has conditional branch every 5 instructions, and a load every 4 instructions.

1. For the baseline, if we have a branch predictor with accuracy 90% (and still have perfect caches), what is the expected IPC? 0.975

For 100 inst, $62.5 + 2 * 20 = 102.5$ cycles

2. What happens if we now add realistic data cache to this model. The cache hit-rate is 80% and cache miss takes 10 cycles to service. What is the new IPC? 0.6557

$102.5 + (5 * 10) = 152.5$ cycles

3. What if we now add out-of-order execution to this machine. This enhancement would increase the perfect IPC from 1.6 to 1.9. However, it also increases the pipeline stages from 20 to 25. So what would be the IPC with out-of-order execution with the branch predictor and data caches as described above? 0.655.

$52.6 + (2 \cdot 25) + (5 \cdot 10) = 152.6$ cycles.

4. We found that out-of-order execution increased the number of parallel misses in the data-cache from 1 to 2 (recall the concept of Memory Level Parallelism, or MLP). Taking MLP into account, the revised IPC is 0.793.

$52.6 + (2 \cdot 25) + \left(\frac{5 \times 10}{2}\right) = 127.6$

5. Start with the baseline 2-wide in-order machine that has a branch predictor of (1) and data cache of (2). You have a choice of any two of the three enhancements below:
   a. A branch predictor with 98% accuracy
   b. A data cache with 96% hit rate (miss latency remains the same)
   c. Out-of-order execution with MLP of 2

   You would recommend choosing enhancements A and B because IPC = 1.23

   (Hint: There are three combinations, ab, ac, and bc. Which combination has highest IPC?)

AB

$62.5 + (0.4 * 20) + 10$

$\boxed{81.5}$

BC

$52.63 + \left(\frac{10}{2}\right) + 50$

$\boxed{108}$

AC

$52.63 + (0.4 * 25) + \left(\frac{5 \cdot 10}{2}\right)$

$\boxed{87.5}$

Name:____

Problem 4 "Caching Insights" (15 points)

Vector dot product (vdotp) kernel used in many scientific applications. Shown below is a typical implementation:

```
double vdotp (double *a, double *b, int size){
       double sum=0;
       for(i=0; i<size; i++)
         sum += a[i]*b[i];
       return sum
}
```

The size of each double variable is 8 bytes. We are interested in computing the vdotp for vectors having 1K elements. The variables for ***sum*** and ***i*** are kept in a register, they do not incur any cache/memory access for operating on them.

We have a system with two levels of caching (L1 and L2). L1 cache is 2-way, 32KB, and uses 64B linesize. L2 cache is 256KB 8-way and uses a linesize of 256 bytes.

A. How many L1 accesses does this kernel generate? 2K

B. What is the hit rate of the L1 cache? 7/8

C. What is the hit rate of the L2 cache? 3/4

D. How many accesses go to memory? 64.

E. Assume L1 access time is 1 cycle, L2 access time is 10 cycles, and memory access time is 100 cycles.

What is the Average Memory Access Time (AMAT) for our system? __________
(Hint: Use answers A, B, C, and D to calculate this)

$$1 + \frac{1}{8}\left(10 + \frac{1}{4}\cdot 100\right) \simeq 5.4$$

Name:____________________________________

Problem 5 "Loop Unrolling and Branch Prediction" (15 points)

This problem analyzes the effects of loop unrolling. We are interested in computing the sum of all integers in a cache line of 64 bytes. Each integer element is 4B. The baseline code is shown on the left. Each iteration of the loop contains five instructions: a Load, an ADD, an array index increment, a counter decrement, and a conditional branch. To optimize this loop we perform loop unrolling four times, as shown on the right.

```
sum=0;
for(i=0; i<16; i++)
   sum += array[i];
```

```
sum=0;
for(i=0; i<16; i+=4){
   sum += array[i];
   sum += array[i+1];
   sum += array[i+2];
   sum += array[i+3];
}
```

A. Estimate the number of instructions executed in the baseline code: $(16*5)+1+1 \simeq 80$

B. Estimate the number of instructions executed in the unrolled code, assuming that the ISA supports a base + offset addressing mode: __________ $(11*4)+1+1 \simeq 45$.

C. If the code gets executed on a machine with last time predictor (single bit history), then we would expect the baseline code to have 2 mispredictions and the unrolled code to have 2 mispredictions (for the first time the branch is encountered assume that the branch is predicted not taken).

D. Now assume that the last time predictor is implemented with a two-bit counter, which is initialized to a weakly taken state. We would expect the baseline code to have 1 mispredictions and the unrolled code to have 1 mispredictions.

E. For this kernel, we want to rewrite the code so that we avoid branch mispredictions completely. Can you write the revised code in the box?

What is the disadvantage of this code (in five words or less):

Code Bloat

Name:____

Problem 6 "Out of order timeline" (15 points)

We are interested in comparing the performance of an in-order machine with an out-of-order machine (with and without in order retirement). We will consider the following sequence of instructions (format OPCODE RD, RS1, RS2):

Inst1: FDIV **R2**, R1, R0
Inst2: FMUL R4, R3, **R2** **(Dependent on R2)**
Inst3: FDIV R7, R6, R0

The latency to perform the execution stage of FMUL is 10 cycles and the latency for FDIV is 40 cycles **Assume that the ALU is fully pipelined** (so, can start the execution of a new instruction every cycle, if required).

A. Consider a four-stage (IF, ID, EX, WB) 1-wide in-order pipeline executing the above snippet of three instructions. Initially the pipeline is empty. We are interested in knowing the cycle count at which the instruction starts execution, finishes execution, and performs writeback.

Instruction	Starts EX	Finishes EX	Writeback
Inst1	3	42	43
Inst2	43	52	53
Inst3	44	83	84

(OK if Fwd assumed)

B. Now consider an out-of-order 1-wide pipeline that performs renaming but there is no in-order retirement. So, as soon as the instruction finishes execution, the results are updated to the ARF. The stages in this machine are: IF, ID, RN (and into Reservation Station), Scheduling, Execute, and Broadcast/WB. Initially the pipeline is empty. We are interested in knowing the cycle count at which the instruction starts execution, finishes execution, and performs writeback

Instruction	Starts EX	Finishes EX	Writeback
Inst1	5	44	45.
Inst2	45	54	55.
Inst3	7	46.	47.

C. Now consider that we added a ROB to the above out-of-order pipeline to support in order retirement, similar to what you had for Lab3. The ROB can commit at most one instruction in a given cycle. The stages in this machine are: IF, ID, RN (and into Reservation Station), Scheduling, Execute, Broadcast and into ROB, and Commit. Initially the pipeline is empty. We are interested in knowing when the instruction starts execution, finishes execution, and commits.

Instruction	Starts EX	Finishes EX	Commit
Inst1	5	44	46.
Inst2	45	54	56.
Inst3	7	46	57

D. Modern processors ensure in-order retirement for out-of-order machines to provide (five words or less)

Precise Exception.