EECS 370 - WN21 Final Exam

Question 1: Short Question Multiple Choice [11 points - 1pt per question]

1. Honor Code Signature

2. Having a pipeline with more stages will (always/sometimes/never) decrease the overall execution time.

3. Given the same set of instructions, a pipeline that uses detect and stall will sometimes perform better than a pipeline that uses detect and forward with speculate and squash. Assume both pipelines have the same clock period and are identical in every way except how hazards are handled. (True/False)

4. A pipeline that uses speculate and squash to handle control hazards could have 0 squashes if all branches are predicted correctly. (True/False)

5. Caches are an important part of computers because they generally store more memory relative to the hard disk and have very little latency. (True/False).

6. A fully associative cache will (always/sometimes/never) require more tag bits per entry as the cache size gets larger, given that all cache sizes are powers of 2.

7. A write-through cache system will (always/sometimes/never) write to both cache and memory.

8. A cache with very large block sizes will generally have less compulsory misses than a cache with smaller block sizes. (True/False)

9. SRAM will have higher costs to make but will generally also have higher latency compared to DRAM and the disk. (True/False)

10. The TLB lookup could happen in the pipeline after the virtual address is calculated and before the memory reference is executed. (True/False)

11. The translation look-aside buffer will often have a hit rate that is greater than 90%. (True/False)

Question 2: Detecting Hazards in LC2K Code [12 points]

Suppose the following LC2K assembly code is run through the 5-stage pipeline from lecture until it halts.

Address	Label	Opcode	Arg0	Arg1	Arg2
0		lw	0	1	one
1		add	1	1	3
2		lw	0	2	two
3		add	1	1	4
4		beq	2	3	skip
5		lw	0	4	one
6	skip	add	4	3	4
7		halt			
8	one	.fill	1		
9	two	.fill	2		

1. Select all registers that cause a RAW (Read after write) data hazard in a **detect and stall** pipeline. Assume that data hazards are resolved with **detect and stall**. Assume that control hazards are resolved with **detect and stall**. *Data hazards can only occur from instructions that are executed.* [3 pts]

a. reg0
b. reg1
c. reg2
d. reg3
e. reg4
f. reg5
g. reg6
h. Reg7

2. Using **detect and stall** to resolve data hazards, and **detect and stall** to resolve control hazards, what is the number of stalls that occur only due to data hazards? [3 pts]

3. Multipart Question: Using **detect and forward** to resolve data hazards, and **predict not taken** to resolve control hazards. Answer the statements below. [6 pts]

a. How many cycles are required to empty out the pipeline (at the end of a program) [1.5]

b. How many instructions are run in this program? [1.5]

c. With parts a) and b) in mind, how many cycles does the program take to run? [3]

Question 3: Pipeline Datapath Design
[15 points - Each blank 0.75 point]

In order to speed up pointer dereferences, two instructions are added to LC2K with the following semantics:

Pointer Load Word
plw regA regB offset // regB = memory[memory[regA + offset]]

Pointer Store Word
psw regA regB offset // memory[memory[regA + offset]] = regB

Memory stage of the 5-stage pipeline is also broken up into two stages: MEM1 and MEM2. In MEM1, plw and psw load the pointer from memory. lw and sw access data memory in MEM1. In MEM2, plw loads the data from the pointer address in memory. psw stores the value of regB to the pointer address in memory.

Q3.1) Data memory is provisioned to have single read/write port that allows only one read or write in any cycle. This introduces a **structural hazard:** MEM1 and MEM2 cannot access data memory in the same cycle.

We use detect and stall to address this hazard by stalling the IF and ID stages and inserting Noop(s) into the id_ex register. All the data forwarding is also done to EX stage. We want to simulate our new pipeline, similar to project 3. Below, complete the C expression so that the bool is true if and only if our simulated pipeline should stall for the structural hazard. Use the pipeline registers, constants, variables, and helper functions from the **Simulation Appendix** at the end of the problem.

```
bool stall_for_structural_hazard =
            ( opcode( if_id.instr ) == PLW
              || opcode( __________ ) ==  __________
              || opcode( __________ ) ==  __________
              || opcode( __________ ) ==  __________  )
      &&
            ( opcode( __________ ) ==  __________
              || opcode( __________ ) ==  __________  );
```

Q3.2) Our new instructions also introduce data hazard(s) that require a stall. Below, complete the C expression so that the bool is true if and only if our simulated pipeline should stall for a new data hazard. As before, we stall the IF and ID stages and insert Noop(s) into the id_ex register. This expression should **NOT** account for the pre-existing data hazard stall due to LW.

```
bool stall_for_new_data_hazard =
            ( opcode(id_ex.instr) == PLW
              && regA(if_id.instr) != -1
              && regA(if_id.instr) == regB(id_ex.instr) )
      ||
            ( opcode(id_ex.instr) == PLW
              && regB(if_id.instr) != -1
              && regB(if_id.instr) == regB(id_ex.instr) )
      ||
            ( opcode( __________ ) ==  __________
              &&  ________________  != -1
              &&  ________________  ==  __________  )
      ||
            ( opcode( __________ ) ==  __________
              && ________________  != -1
              && ________________  ==  __________  );
```

Simulation Appendix:

Opcode Constants

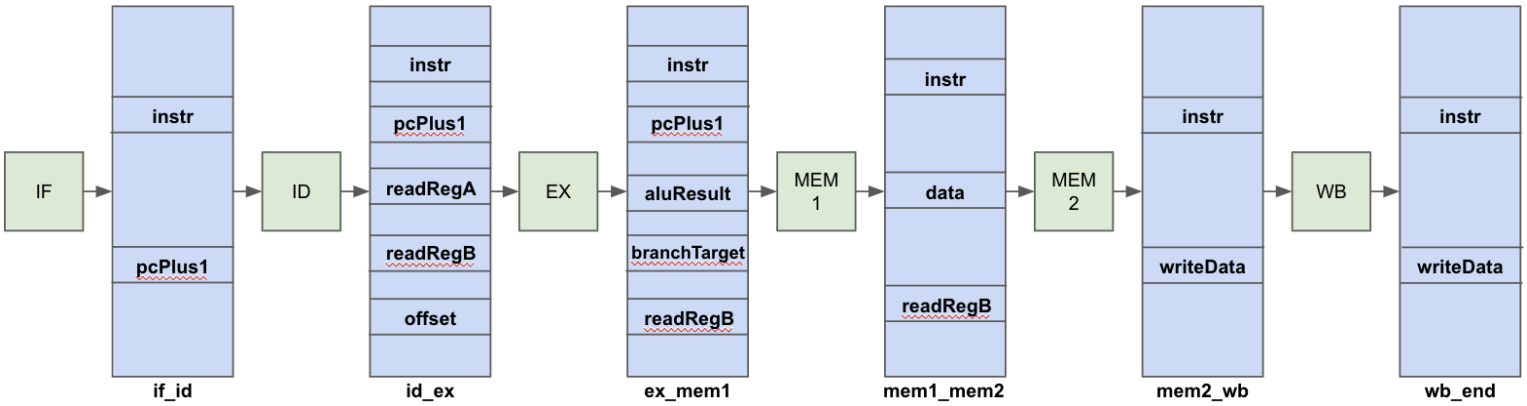
PLW	ADD
PSW	NOR
LW	BEQ
SW	HALT
NOOP	

Helper Functions

```
// Returns the opcode of "instruction"
int opcode(int instruction);

// Returns regA of "instruction" if "instruction" reads regA, otherwise returns -1.
int regA(int instruction);

// Returns regB of "instruction" if instruction reads or writes regB, otherwise returns -1.
int regB(int instruction);
```



- Pipeline registers are implemented as C structs.
- All pipeline variables are integers (int).

Question 4: Pipeline Performance [14 points]

- Consider a 1GHz version of an LC2K pipelined processor (as described in class, **using detect-and-forward as well as predicting not-taken**). You have been looking into designing a new, 1.4GHz version of the processor.
- However, this higher frequency has come at the cost of increasing the number of cycles for all operations in the ALU to complete by one. So, our new pipeline now has 6 stages (IF, ID, EX1, EX2, MEM, and WB).
- The processor is still fully pipelined, and every other stage works exactly the same. Branches are resolved in the MEM stage and data is always forwarded to EX1 stage.
- Assume a benchmark with the following characteristics will be run on the original and new pipeline -
    - 15% instructions are loads
    - 20% instructions are stores
    - 30% instructions are adds
    - 25% instructions are nors
    - 10% instructions are branches
    - 30% of instructions are **dependent** on the instruction immediately in front of them
    - 10% of instructions are **not dependent** on the instruction immediately in front of them but **are dependent** on the instruction after that
    - For the above two lines you may assume that they are true no matter what instructions are involved.
    - The distribution of immediate and non immediate dependencies are exclusive from each other.
    - 30% branches are taken.

  In summary, we have two processors:
    - Original: 1.0 GHz, 5 stages
    - New: 1.4GHz, 6 stages

  Answer the following questions –

1. Assuming no control or data hazards, what is the execution time of the two processors (Assume 100 instructions)? [2]

Processor	**CPI**	**Execution Time**
Original		
New		

2. What is the CPI of the new processor (Taking data and control hazards into account, and now assuming infinite instructions)? [3]

3. Explain the impact of adding the extra stage to the new processor on control hazards and data hazards (Consider the impact of the new cycle on control hazards and data hazards separately) [3]

4. Now consider an LC2K program with the following characteristics –
   - 15% instructions are loads
   - 20% instructions are stores
   - 30% instructions are adds
   - 25% instructions are nors
   - 10% instructions are branches
   - 30% of the instructions are dependent on the instruction immediately in front of them. You may assume this is true no matter what instructions are involved.
   - 30% of the branches are taken
   - (Note that percentage of each instruction and dependencies is the same as the new pipeline)
   - You may assume that infinite instructions are being executed

   i) What would be the expected CPI of your program using detect-and-forward to resolve data dependencies and detect-and-stall for branches? Assume we are using the OLD pipeline (the one described in class). Clearly show your work. [3]

   ii) What would be the expected CPI of your program using detect-and-forward to resolve data dependencies and predict-not-taken for branches? Assume we are using the OLD pipeline (the one described in class). Clearly show your work. [3]

Question 5: Branch Prediction [12 points]

Consider the following C code used for *upsampling* a sequence of input samples:

```
void up_sampler(int *input, int *output){

      int f = 3;                          // upsampling factor
      int count = 0;
      int i;

      for(i = 0; i < 5; ++i){             // B1 LC2K: beq 2 3 endOuterLoop
                                          // NOT taken if i < 5
            int j;

            for(j = 0; j < f; ++j){       // B2 LC2K: beq 2 4 endInnerLoop
                                          // NOT taken if j < f
                  output[count] = input[i];
                  count ++;
            }                             // B3 LC2K: beq 0 0 innerLoop
                                          // Taken if 0 == 0
      }                                   // B4 LC2K: beq 0 0 outerLoop
                                          // Taken if 0 == 0

}
```

In the C code above, some branch instructions in LC2K are provided. Assume each branch is resolved before the next one.

1. How many branches are taken versus not taken? [6 - 1.5 per column]

Branch	B1	B2	B3	B4
# Taken				
# Not taken				

2. How many branches are correctly and incorrectly predicted using local 2-bit saturating counter, initialized to 01 (weakly not taken)? [6 - 1.5 per column]

Branch	B1	B2	B3	B4
# Correctly Predicted				
# Incorrectly Predicted				

Question 6: Reverse Engineering the Cache [10 points]

Due to a power outage at North Campus, Faryab lost the Autograder's Cache configuration. He has trusted you to recover its contents. Before leaving it to you he recalls a few parameters:

- You may assume that the cache is **byte-addressable.**
- You may assume the block size and number of sets are both **powers of two**.
- You may assume that the cache is **empty** before the first access.

In order to gather some data on the lost cache configuration, you have written a program that accesses different memory locations. You use the response time to determine if each access was a hit or a miss. Your results are tabulated below.

Note: Partial credit will be rare, and perhaps non-existent, on this problem.

Access	Address (Hex)	Binary	Hit/Miss
1	0xC0	0b 1100 0000	M
2	0xC1	0b 1100 0001	H
3	0xDE	0b 1101 1110	M
4	0xC3	0b 1100 0011	H
5	0xDF	0b 1101 1111	H
6	0xC7	0b 1100 0111	M
7	0x2E	0b 0010 1110	M
8	0xDE	0b 1101 1110	M
9	0xC1	0b 1100 0001	H
10	0xC7	0b 1100 0111	H

1. Looking only at the **first three accesses,** what is the *smallest possible* range of block sizes (in bytes)? [4]

Give your answer in the form _____ **≤ Block Size ≤** _____

explanation:

2. Looking only at the **first six accesses**, you are able to determine the blocksize. What is it? [2]

After the ten accesses you have all the information needed to determine the configuration of the cache.

3. How associative is the cache? Pick one answer. [2]

Direct Mapped **2-Way Associative** **Fully Associative**

4. What is the size of the cache? (In Bytes) Pick one answer. [2]

8 Bytes **16 Bytes**

Question 7: Hierarchical Page Tables [12 points]

You are provided with a new 64-bit processor that uses a three-level page table. Page size is 8KB, and memory is byte-addressable.

1. If each level's page tables hold the same number of page table entries, how many page table entries does a page table contain? [2]

   2^_____ entries

2. If there are 19 bits necessary to represent a physical page number, how much physical memory is there (in bytes)? [2]

   _____

3. Now consider the following changes to the system: page size is changed to 2KB, while physical and virtual memory address size and the number of entries in the top-level page table are held constant. Any changes to second- or third-level page tables should happen to both.

   Answer the following questions.

   a) The number of entries in each second- and third-level page table is now 2^_____ entries [2]

   b) The number of bits necessary to represent a physical page number is now _____ bits [2]

   c) **In the OLD system,** accessing every element of a 1GB array would require a minimum of _____ third-level page tables. **In the NEW system**, it would require a minimum of _____ third-level page tables. [4]

Question 8: Virtual Memory [15 points]

Consider a 16-bit system with following specifications:

Cache:

- Cache Size is 16 Bytes
- Cache is two way set associative
- Cache block size is 4 Bytes

Memory:

- Byte-addressable
- Physical memory is 16KB
- Page Size 4KB
- Physical page 0 is used by OS to store the page table
- Single level page table.

TLB:

- TLB has 2 entries
- TLB is fully associative
- TLB and cache access are sequential

Latency:

- Cache access time is 2ns
- TLB access time is 1ns
- Memory access time is 50ns
- Disk access time is 1000ns

Notes:

- All updates are in parallel. Upon retrieval from cache, main memory or disk, the data is sent immediately to the CPU, while other updates occur in parallel
- If more than one free page is available, the smallest physical page number is chosen.
- If there are no free pages, LRU policy is used to select the page to be replaced.

1. The table below shows the access latency of a sequence of requests (top to bottom) if the cache is virtually addressed.

Access ID	Process ID	Virtual Address	Latency
1	1	0x1234	1053
2	1	0x023A	1053
3	2	0x1237	1053
4	1	0x1236	103

If the cache were instead to use physical addresses, which access/accesses will have a different latency, and what will it be? [4]

2. Assume following accesses (top to bottom) are given and the cache is virtually addressed.

Process ID	Virtual Address
1	0x1234
1	0x023A
2	0x1237
1	0x1236
2	0x02AB
2	0x0123
1	0x1123

What is the final value in the first two entries of the processes' page tables, as well as the TLB? Assume page tables for both of the processes would be allocated on physical page 0.

Process 1 Page Table [4]

VPN	On disk or memory	PPN(if in the memory)
0		
1		

Process 2 Page Table [4]

VPN	On disk or memory	PPN(if in the memory)
0		
1		

Continued in next page.

TLB (Order doesn’t matter) [3]

Entry	VPN	PPN
0		
1		