EECS 370 Final Exam

Winter 2020

Notes:

- Closed book. Closed notes.
- **10 problems on 16 pages**. Count them to be sure you have them all.
- Calculators without wireless are allowed, but no PDAs, Portables, Cell phones, etc.
- This exam is fairly long: *don't spend too much time on any one problem*.
- You have **120 minutes** for the exam.
- Some questions may be more difficult than others. You may want to skip around.
- **Partial credit cannot be given if work is not shown.**

Write your uniqname on the line provided at the top of each page.

You are to abide by the University of Michigan/Engineering honor code. Please sign below to signify that you have kept the honor code pledge:

I have neither given nor received aid on this exam, nor have I concealed any violations of the Honor Code.

Signature: _____

Name: _____

Uniqname: _____

Problem 1: Short Questions **Points: ___/ 5**

A) We have two caches with the same block size, number of sets and associativity. When comparing the one with a LRU policy versus the one with a MRU policy (the most recently inserted block is the first to be evicted), the LRU cache will have better spatial locality

Circle one: **True** or **False**

B) Multi-level page tables will use less memory than a single level page table when the process accesses the whole virtual memory address space.

Circle one: **True** or **False**

C) A 64 bit system has more virtual addresses than a 32 bit system, so the page size must be larger for the 64 bit system.

Circle one: **True** or **False**

D) What combination of data hazard and control hazard resolution will always result in the lowest CPI on the pipeline discussed in class. Circle all the right choices:

a. Detect and forward/ detect and stall
b. Detect and stall/ speculate and squash
c. Detect and stall/ detect and stall
d. Detect and forward/ speculate and squash

E) Assuming the cache is physically addressed what combination of events is not likely to occur, Circle all the possible choices:

a. Cache: Hit, TLB: Hit, Page Fault: No.
b. Cache: Hit, TLB: Miss, Page Fault: Yes.
c. Cache: Miss, TLB: Miss, Page Fault: No.
d. Cache: Miss, TLB: Hit, Page Fault: Yes.

Problem 2: Noop! Any Closer and It's a Hazard **Points: ___/ 10**

```
1           lw          0           2           neg1
2           lw          0           1           social
3           lw          0           3           dist
4  loop     add         3           3           5
5           add         0           0           4
6           nor         5           5           6
7           add         1           2           1
8           beq         0           1           end
9           beq         0           0           loop
10 end      halt
11 neg1     .fill       -1
12 social   .fill       5
13 dist     .fill       3
```


Suppose you ran the above LC2K assembly code through the 5-stage pipeline from lecture until it halts.

A. Select all registers that cause a RAW data hazard in a detect and stall pipeline? **[4 pts]**

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

Answer:___b,d,f_____

B. How many noops would result from using **detect and stall** to resolve data hazards? **[3 pts]**

(3, 4) → 2 noop
loop runs **5** times
(4, 6) → 1 noop * **5** (7, 8) → 2 noop * **5**
2 + (1 * 5) + (2 * 5) = 17 noops

Answer:___**17**_____

C. How many noops would result from using **data forwarding** to resolve data hazards? **[3 pts]**

(3, 4) → 1 noop

Answer:____**1**_____

Problem 3: Trees and Branches Points: ___/ 15

Consider the following assembly program which counts the number of odd numbers. Assume all registers are initialized to zero.

```
        lw 0 1 one      //r1 = 0x1
        lw 0 4 four     //r4 = 4
loop    beq 3 4 end     //beqA
        nor 3 3 5
        nor 1 1 2
        nor 2 5 5       //r5 = r3 & r1
        beq 0 5 even    //beqB
        add 6 1 6       //r6 counts the odd numbers
even    add 3 1 3
        beq 0 0 loop    //beqC
end     halt
one     .fill 1
four    .fill 4
```


A) Write the sequence of branch decisions for each beq instruction. Let "taken" be denoted as "T" and "not taken" as "N" **[4 pts]**

a) beqA:____NNNNT_____
b) beqB:____TNTN_____
c) beqC: _____TTTT_____
d) Combined sequence of branches seen by a global branch predictor:_____NTT NNT NTT NNT T_____

B) What percentage of total branches do we predict correctly for each of the following prediction schemes? (express answers as fractions)

a) Predict always not taken **[1 pts]**
_____6/13_____
b) Predict backwards taken, forwards not taken **[3 pts]**
_____10/13_____
c) Global 1-bit Branch Predictor: Initialized to not taken **[3 pts]**
_____6/13_____
d) Local 2-bit Branch Predictor: Initialized to weakly taken **[4 pts]**
_____9/13_____

Problem 4: Tracing the Pipe Points: ___/ 15

Consider the LC2K program below. Assume the same pipeline as presented in the lecture. Recall that we have IF/ID, ID/EX, EX/MEM and MEM/WB intermediate pipeline registers to pass data between stages for each clock cycle.

```
      lw          0     1     covid       // lw1
      lw          0     2     var2        // lw2
loop  add         1     2     3           // add1
      nor         3     3     3           // nor1
      beq         4     1     1           // beq1
      nor         4     4     2           // nor2
      beq         2     3     loop        // beq2
      halt
covid .fill       19
var2  .fill       999
```


PART 1) Assume that the pipeline resolves data hazards using detect and forward, and control hazards using speculate and squash *(predicting not taken).*

a) What pipeline register(s) will supply the operand valA (**regA** value) to ALU in the execution stage for instruction **lw1**? **[2 pts]**

IDEX, (There is no dependency for the first instruction)

b) What pipeline register(s) will supply the operands to ALU inputs in the execution stage for instruction **add1**? **[2 pts]**

MEMWB for reg2 (Forward from lw1 in its WB stage),
IDEX for reg1 (lw1 finishes execution so the registers are updated)

c) What pipeline register(s) will supply the operands to ALU inputs in the execution stage for instruction **nor1**? **[2 pts]**

EXMEM for both, regA and regB are the same,(Forward from add1 in it's MEM stage)

d) What pipeline register(s) will supply the operands to ALU inputs in the execution stage for instruction **beq2**? **[3 pts]**

IDEX for reg2 and reg3

e) You observe weird behavior with this program; **nor2** seems to never have been executed. Will it ever be fetched into the pipeline? Why/Why not? **[2 pts]**

.**Yes.** We are predicting not taken so our pipeline will fetch the next 3 instructions after **beq1** until it realizes that we are branching. Then it will set those to noop and actually fetch the proper instruction

PART 2) You want to check if anything changes if you reconfigure your pipeline. Now you use <u>detect and stall</u> to resolve both data and control hazards.

f) What pipeline register(s) will supply the operands to ALU inputs in the execution stage for instruction `add1`? **[2 pts]**

`IDEX, (We always stall until the hazard is resolved)`

g) Will `nor2` ever be fetched into the pipeline? Why/Why not? **[2 pts]**

No. We stall until the BEQ instruction finishes execution. Then after 3 cycles we learn that the branch is taken, because of this we branch to `beq2` after instead of fetching `nor2`.

Problem 5: The C Musketeers **Points: ___/ 10**

Consider a cache with the following specifications:

- Byte-addressable Memory
- 2-Way Set Associativity
- Block Size: 16 bytes
- Cache Size: 64 bytes
- Memory Size: 64 KB (64 * 1024 bytes)

The following addresses are referenced. Assume cache is empty initially. Determine whether each reference is a hit or a miss and, given a miss, categorize the type of miss as compulsory, capacity, or conflict.

Reference	Hit / Miss	Type (if miss)	
0x0342	Miss	Compulsory	**[1 pts]**
0x054A	Miss	Compulsory	**[1.5 pts]**
0x034B	Hit		**[1.5 pts]**
0x020F	Miss	Compulsory	**[1.5 pts]**
0x0543	Miss	Conflict	**[1.5 pts]**
0x083C	Miss	Compulsory	**[1.5 pts]**
0x017D	Miss	Compulsory	**[1.5 pts]**

Problem 6: Caches to caches, disk to disk Points: ___/ 20

Consider a virtual memory system with a **single-level page-table, fully-associative TLB, physically addressed data cache**, and the following specs:

- 1 MB physical memory, 4 GB virtual memory (32-bit virtual address),
- Data Cache, fully associative, 16 byte block size

Note that upon retrieval, data is sent to CPU instantly with updates occurring in negligible time

Given are the following latency and following states of hardware components for an arbitrary process PID 11 (note that valid bits, LRU bits, etc. were removed for the sake of brevity).

Hardware Component	TLB	Data Cache	Main Memory	Disk
Access Time	1 cycle	1 cycle	30 cycles	100,000 cycles

TLB		PID 11 Page Table		Data Cache Tag (Physically Addressed)
Virtual Page #	Physical Page #	Virtual Page #	Physical Page #	0xFBC5
0xFE	0xFBC	0xA7	0x000	0x4210
0x00	0xAE4	0x01	0x87C	0x87E2
0x4C	0x421	0xFE	0xFBC	0xAE4A
0x77	0xABC	0x99	0x5A1	0x54FE

A) Determine the **page size** based on given parameters and contents of TLB, page table, data cache tags. **[10 pts]**

Answer: ____256_____ bytes

B) Please give the latency of following **virtual** address accesses. Assume (1) Components remain unmodified for each access **(i.e. each accesses should be simulated independently).** (2) If the PTE is not present in the page table, the data is not present in cache or memory and it takes one disk access to fetch both data and PTE. (3) Page Table is never cached. To get partial credit show the breakdown of cycles in your answer.

1. 0x01E2 Answer:1 + 30 + 1 + 30 = 62 cycles **[2.5 pts]**
2. 0xE9A5 Answer:1 + 30 + 100,000 = 100,031 cycles **[2.5 pts]**
3. 0x00B0 Answer:1 + 1 + 30 = 32 cycles **[2.5 pts]**
4. 0xFE59 Answer:1 + 1 = 2 cycles **[2.5 pts]**

Problem 7: Benchmarking Again! Points: ___/20

We have a pipeline processor with the following component latencies:
Memory Access: 60ns; Register Read: 5ns, Register Write: 10ns; ALU: 20ns; Other: 0ns

We run a program with 1000 instructions:

- 40% add/nor
  - 20% immediately followed by a dependency
  - 30% followed by a dependency 1 instruction away
  - 10% followed by a dependency 2 instructions away
- 30% lw
  - 10% immediately followed by a dependency
  - 30% followed by a dependency 1 instruction away
  - 5% followed by a dependency 2 instructions away
- 15% sw
- 10% beq
  - 40% taken
- 5% nop/halt

A) On the normal 5-stage pipeline discussed in lecture (using **detect-and-forward** for data hazards, **internal forwarding**, and **speculate-and-squash-predict-not-taken** control hazards) what is the Execution Time? (show your work to get partial credit) **[2 pts, 6 pts]**

Clock Period: 60ns Cycles: 1154

#Cycles (Base) = 1000+4 = 1004
#Cycles (Data Hazards) = 1000(0.3*0.1*1) = 30
#Cycles (Control Hazards) = 1000(0.1*0.4*3) = 120
#Cycles (Total) = 1004+30+120 = 1154 Cycles
Clock Period = 60ns
Execution Time = 1154*60 = 69240ns

B) Letao comes in and thinks he might be able to do better. He makes the following changes.

- He creates his Magic Box v2.0, capable of handling all *beq* logic (equality check and PC setting) in a single stage of execution. The component itself runs in **25ns**. He places MBv2.0 in the **Execute** stage of the Pipeline where it runs in parallel with the normal ALU unit.

- On top of this, Letao also decides that the Memory Access time is too long, so he splits the **Memory** stage into 3 different cycles: Mem1, Mem2, Mem3; each stage taking **20ns**.

The *sw* instruction is capable of finishing its write to memory by Mem2, but *lw* doesn't finish reading from Memory until Mem3.

- The **Fetch** stage uses a new memory which takes 20ns.

- All data still gets forwarded to the **Execute** stage.
- Letao's pipeline still uses **detect-and-forward** for data hazards, internal forwarding, and **speculate-and-squash predict not-taken** for control hazards. Show your work to get partial credit.

With his updated design, how long will Letao's Pipeline take to run the above program?
[2 pts, 10 pts]

Clock Period: 25ns Cycles: 1371

Number of extra cycles for incorrect predictions: 2
Now need to add 3 stalls for lw followed 1-away by dependency
Now need to add 2 stalls for lw followed 1-away by dependency
Now need to add 1 stall for lw followed 2-away by dependency
Clock Period = 25ns

#Cycles (Base) = 1000+6 = 1006
#Cycles (Data Hazards) = 1000(0.3*0.1*3 + 0.3*0.3*2 + 0.3*0.05*1) = 285
#Cycles (Control Hazards) = 1000(0.1*0.4*2) = 80
#Cycles (Total) = 1006+285+80 = 1371
Execution Time = 1371*25 = 34275ns

Problem 8: Cubical Hierarchy Points: ___/10

You are given a byte-addressable processor which uses 42-bit virtual addressing. The system has 4 GB of RAM installed. You've been asked to design a virtual memory system with three-level page tables for this processor given the following parameters:

- Page size: 4 KB
- All page table entries are 4 bytes.
- Second and third level page tables occupy one page.
- First (super) and second level page table entries store the physical page number of the next level page table.
- Third level page tables entries store physical page numbers.

The breakdown of bits of virtual address and physical address is shown below.

Virtual Address: 42 bits

Super page table	Second level	Third level	Page offset
10 bits	10 bits	10 bits	12 bits

Physical Address: 32 bits

Physical page number	Page offset
20 bits	12 bits

How many memory pages in total needed to store **all the hierarchical page tables** for accessing **every** element of an array: **[10 pts]**

int my_big_array[4*1024][1024*1024]?

Answer: ___1 (super) + 4 (2nd) + 4096 (3rd) = 4101____ pages

Array size = 16 GB, every element is accessed so the whole of 16 GB needs to be addressed.

Each 3rd level page-table covers 1024 (#PTE) * (physical page size) 4 K = 4 MB of virtual memory space, need 16 GB/ 4MB = 4K 3rd level page tables.

Each 2nd level page-table covers 1024 (#PTE) * 4 MB (mem. addressed by each 3rd level table) = 4 GB of virtual memory space, thus need 16 GB/ 4GB = 4 2nd level page tables.

Problem 9: Virtualizing the Virtual Points: ___/20

Consider the following system:

- Virtual address space size : 4KB
- Page size : 256 bytes
- Physical memory size : 8 pages
- Page replacement policy : LRU
- Page table entry size : 16 bytes
- Byte addressable architecture
- Single-level page table
  - Page table size : 256 bytes

Notes:
On a page fault, the page table is updated before allocating a physical page.
Physical page #0 (reserved for OS) and pages allocated to page tables cannot be replaced.
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.

Assume that two processes with Process ID 270 and 370 respectively, have been running on this new architecture.

The initial state of physical memory is shown below:

Physical Page # (PPN)	Memory Contents
0x0	Reserved for OS
0x1	Page Table of PID 270
0x2	Page Table of PID 370
0x3	
0x4	
0x5	PID 270: VPN 0
0x6	PID 370: VPN 3
0x7	

Fill in the blank space(s) for each row below.

Part	Time	Process ID	Virtual Address (VA)	Virtual Page # (VPN)	Physical Page # (PPN)	Page Fault? (Y/N)	Physical Address (PA)
A.	0	270	0x027	0x0	0x5	N	0x527
B.	1	270	0xF8A	0xF	0x3	Y	0x38A
C.	2	370	0x901	0x9	0x4	Y	0x401
D.	3	370	0x027	0x0	0x7	Y	0x727
E.	4	270	0xFCA	0xF	0x3	N	0x3CA
F.	5	370	0xA00	0xA	0x6 (evicted)	Y	0x600
G.	6	370	0x92D	0x9	0x4	N	0x42D

A **[2 pts]**
B **[2 pts]**
C **[2 pts]**
D **[4 pts]**
E **[3 pts]**
F **[4 pts]**
G **[3 pts]**

Problem 10: Let's Use That Cache Nicely **Points: ___/25**

Consider the pseudo code below for an Algorithm A:

```
float A[4];
float B[16]; //float size is 4 bytes
....
 for(int i=0; i < 4; i++)
   for(int j=0; j < 16; j++)
      result += 0.561 * A [i] + 0.99 * B[j] + 3.142;
```


The computer used to run this algorithm has a byte addressable main memory of size 64KB, and 32 byte fully associative cache with LRU replacement policy. The block size of cache is 4 bytes. Assume array A starts at address 0x0000, and array B starts at address of 0x0100. Assume only arrays A and B are accessed from cache and memory, all other variables are mapped to registers. Note, cache is empty when the program starts executing the loops.

A) How many data cache hits and cache misses are there for Algorithm A? **[6 pts]**

Answer ___60_____ hits and _____68_____ misses

A new programmer proposes a new Algorithm B for the math, but he took EECS 370 (Aha!).


```
#define TILE 4
float A[4];
float B[16]; //float size is 4 bytes
....
 for(int t=0; t < 16; t+=TILE)
   for(int i=0; i < 4; i++)
     for(int j=t; j < t + TILE && j < 16; j++)
         result += 0.561 * A [i] + 0.99 * B[j] + 3.142;
```

B) How many data cache hits and cache misses are there for the new Algorithm B? **[13 pts]**

Answer ___99_____ hits and _____29_____ misses

C) Will a larger or smaller value of TILE result in the highest number of hits? Which array's (array A or array B) locality is improved with your choice of answer? **[6 pts]**

Answer: ____smaller ___A's locality___

```
Algorithm A: Total accesses = 4 * 16 * 2 = 128
 (1 M + 15 H (Array A, A is not replaced because of LRU)
+ 16 M (Array B)) * 4
60 H, 68 M

Algorithm B: Total accesses = 4 * 16 * 2 = 128
99 H, 29 M


Access pattern for Algorithm B:
0, 100, 101, 102, 103,
1, 100, 101, 102, 103,
2, 100, 101, 102, 103,
3, 100, 101, 102, 103,
0, 104, 105, 106, 107,
1, 104, 105, 106, 107,
2, 104, 105, 106, 107,
3, 104, 105, 106, 107,
…

t=0, i=0 to 4, j=0 to 3
24 H, 8M
1 M + 3 H (Array A) 4 M (Array B)
1 M + 3 H (Array A) + 4 H (Array B)
1 M + 3 H (Array A) + 4 H (Array B)
1 M + 3 H (Array A) + 4 H (Array B)

t=4, i=0 to 4, j=4 to 7
LRU based pattern. 25 H, 7M
4 H (Array A) + 4 M (Array B)
1 M + 3 H (Array A) + 4 H (Array B)
1 M + 3 H (Array A) + 4 H (Array B)
1 M + 3 H (Array A) + 4 H (Array B)

t=8, i=0 to 4, j=8 to 11
LRU based pattern. 25 H, 7M
4 H (Array A) + 4 M (Array B)
1 M + 3 H (Array A) + 4 H (Array B)
```

1 M + 3 H (Array A) + 4 H (Array B)
1 M + 3 H (Array A) + 4 H (Array B)

t=12, i=0 to 4, j=12 to 15
LRU based pattern. 25 H, 7M

4 H (Array A) + 4 M (Array B)
1 M + 3 H (Array A) + 4 H (Array B)
1 M + 3 H (Array A) + 4 H (Array B)
1 M + 3 H (Array A) + 4 H (Array B)

TILE	Cache Hits	Cache Misses		
1	108	20		Tile size = 1 keeps A local in cache
2	108	20		
3	100	28		
4	99	29		
5	97	31		
6	100	28		Tile size =6, keeps B local, but still misses because can't keep all of it due its larger size
7	65	63		

Byte address Algorithm A: ((1M + 3H) + 15*4H)*4 for Array A +
(1 M + 3H)*16 * 4
= 68 M + 444 H

Byte address Algorithm B:
29 M, 483 H