uniqname: _____

# EECS 370 Final Exam

## Winter 2018 **answers**

**Notes:**

- Closed book. 1 8.5x11 inch sheet of paper as notes.
- **9 problems on 12 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 legibly and dark enough for the scanners to read your work.**

**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: _____Key_____

Name: _____Key_____



Uniqname: _____Key_____

First and last name of person sitting to your **right** (write the symbol ⊥ if at the end of a row):

_____

First and last name of person sitting to your **left** (write the symbol ⊥ if at the end of a row):

_____

uniqname: _____

## Problem 1: Short Answer — Points: 16

For questions **a-h**, determine whether the given statements are true or false. Circle true if the statement is true, or false if **any** part of the statement is false. Each question is worth one point.

| | |
|---|---|
| a) A C struct containing two `char` variables and one `short` variable isn't always larger than a pointer on a 32-bit architecture, but it can be. | True False |
| b) In an IEEE floating point number, exponent bits of 0b10000001 denotes a 2 for the exponent. | True False |
| c) A clock with a 100ns period has a frequency of 10MHz. | True False |
| d) In our five stage pipeline, not having internal forwarding in the register file (as we did in project 3) means we need to squash 3 instructions on a mispredicted branch as opposed to 2 in lecture. | True False |
| e) An XOR gate can be created using NOR gates. | True False |
| f) A cache with high associativity will generally have a greater hit rate than one with low associativity, assuming all other parameters are constant. | True False |
| g) In a hypothetical machine with infinite physical memory, virtual memory is still useful. | True False |
| h) An SR latch's output can only change on the edge of its "clock" input. | True False |



### Fill in the blank

Answer the following questions by filling in the blank. **Provide all answers as a decimal integer.** Each question is worth 2 points.

i) In the C programming language if X=5 and Y=9 then X | Y is ___13___

j) In the C programming language if X=5 and Y=9 then X || Y is ___1___

k) In the C programming language if X=3 and Y=2 then X << Y is ___12___

l) If you have 32-bit addresses with byte addressable memory, then a two-way associative 64KB cache with 32-byte cache blocks will need ___10___ bits for the set index.

uniqname: _____

## Problem 2: The Offspring of Assignments *Can* Repeat. Points: 14

a) Consider a 64-bit, byte-addressable system with 2GB of DRAM installed. Each page size is 4KB, and the system uses a two-level page table. For each address, the 12 bits most-significant are used to index into the first-level page table, while the rest (other than the page offset) index into the second-level page tables. All page table entries occupy 4 bytes.

- How many entries are in the first-level page table? **[2]**

  _____4096_____ entries

- How many bits are used to index into the second-level page table? **[2]**

  ____40_____ bits

- How many physical pages does it take to store the first-level page table? **[2]**

  _____4_____ pages



b) A new company has proposed a number of different cache layouts for their system and you've been asked to come in and calculate the overhead for each of the different caches. Their system uses a cache with 2KB of data storage capable of addressing 2GB of byte-addressable memory. Stores will be handled by write-back and allocate-on-write policies. *You **must** show the work for your calculations to receive credit.*



- The first design is a fully-associative cache with a block size of 16 bytes, how many *bits* of overhead would the cache keep in total (including any necessary **tag bits**, **valid bits, dirty bits**, or **LRU bits**)? **[4]**



  2KB/16 Bytes=128 entries. Each entry has a tag of 31-4=27 bits. 1 valid bit, 1 dirty bit, 7 LRU bits. 27+9=36 bits. 36*128

  ____4608_____ bits

- Their next design utilizes a direct-mapped cache with 64 different cache blocks. How many *bits* of overhead would the cache keep in total (including any necessary **tag bits**, **valid bits, dirty bits**, or **LRU bits**)? **[4]**

  Block size=2048bytes/64=32 bytes. Offset =5 bits, index=6 bits. Tag =31-11=20 bit. 1 bit for valid, 1 bit for dirty, so 22 bits per block. 22*64=1408

  _____1408_____ bits

uniqname: _____

## Problem 3: Reverse Engineering the Cache Points: 20

Your company has just purchased a new processor and you have been asked to determine the structure of the cache. In order to gather some data on the cache, you have written a program that accesses different memory locations in the byte-addressable data cache. You use the response time to determine if each access was a hit or a miss. Below are the results from your testing of the cache. Assume the block size and number of sets are both powers of two. Partial credit will be rare, and perhaps non-existent, on this problem.

| Access number | Address | **Hit/Miss** |
|---|---|---|
| 1 | 0x01 | **M** |
| 2 | 0x00 | **H** |
| 3 | 0x0C | **M** |
| 4 | 0x09 | **M** |
| 5 | 0x0B | **H** |
| 6 | 0x10 | **M** |
| 7 | 0x39 | **M** |
| 8 | 0x0E | **M** |
| 9 | 0x3B | **H** |
| 10 | 0x00 | **H** |

**Part A)**
*Looking only at the first **three** accesses*, what is the range of possible block sizes (in bytes)? Fill in the two blanks. **[4]**

__2__ ≤ Block Size ≤ __8__

**Part B)**
*Looking only at the first **four** accesses*, what is the range of possible block sizes (in bytes)? Fill in the two blanks. **[4]**

__2__ ≤ Block Size ≤ __4__



**Part C)**
After access number 10, we have enough data to determine the structure of the cache.

- What is the block size? **[4]**

_____4_____ Bytes

- How associative is the cache? Circle the correct answer. **[4]**

***Direct mapped / 2-way associative / fully-associative***

- How big is the cache? **[4]**

_____32_____ Bytes

uniqname: _____

## Problem 4: I'm Willing to Wait for It, Wait for It Points: 18

a) Draw arrows indicating any possible data hazards that arise in this segment of an LC2K program (Like the example on the right), running on the five-stage pipeline from lecture. Assume there is internal forwarding in the register file. **[4]**

```
1.          lw    2     5     14
2.          add   5     1     6
3.          sw    1     6     7
4.          nor   1     7     3
5.          sw    0     1     19
6.          add   3     7     5
7.          beq   3     4     end
8.          lw    5     0     6
9.          nor   5     6     6
10.   end   sw    0     5     15
```

**EXAMPLE:**

```
nor   1     1     2
add   2     2     3
```



For parts b-d assume the branch at instruction 7 is *not taken.* Assume branches are predicted to be not-taken.

b) How many noops would get inserted into this code snippet if we were using ***detect and stall*** to handle data hazards? **[4]**

Ans: 7

c) How many noops would get inserted into this code snippet if we were using ***detect and forward*** to handle data hazards and between which instructions would noops appear? (circle all that apply) **[4]**

Instructions: 1,2 2,3 8,9 9,10 # of noops: 1

Which two <u>adjacent</u> instructions could be swapped to reduce the number of noops inserted using ***detect and stall*** to resolve data hazards **while not** changing the functionality of the snippet. How many stalls would this save? **[6]**

Instructions to swap: 3, 4 Stalls saved: 2

uniqname: _____

## Problem 5: Making Cache Hurt: My Empire of Dirt Points: 23

Consider a 1024-byte physically-addressed cache with 16-byte blocks. Memory is byte-addressable. Assume all entries in the cache start as "invalid" and addresses are 32-bits. All loads and stores are to 4 byte values.

a) Identify as short a memory access pattern as possible (specific addresses) for the cache described above where a two-way associative cache will get a better hit-rate than a direct-mapped cache. Make it so that the sum of the addresses is as small as possible. Your answer must be written in hex and placed in the box below. **[5]**

0x000, 0x400, 0x000

b) Identify as short a memory access pattern as possible (specific addresses) for the cache described above where a direct-mapped cache will get a better hit-rate than a two-way associative cache. Make it so that the sum of the addresses is as small as possible. Your answer must be written in hex and placed in the box below. **[6]**



0x000, 0x200, 0x600, 0x000 (middle two can be in any order)

0x200, 0x000, 0x400, 0x200 also works. again middle two can swap



c) Say the processor is accessing the cache as shown below. Indicate the total number of bytes read and written to and from *memory* in each of the following cases. **[12]**

Load 0x1000; Load 0x1008; Store 0x1000; Store 0x8008; Store 0x8010; Store 0x1000

1) A direct-mapped write-back, write-allocate cache.

   Bytes Read from Memory: 64 Bytes Written to Memory: 32

2) A direct-mapped write-through, no write-allocate cache.

   Bytes Read from Memory: 16 Bytes Written to Memory: 16

3) A two-way associative write-back, write-allocate cache

   Bytes Read from Memory: 48 Bytes Written to Memory: 0

uniqname: _____

## Problem 6: LC2K or Not — Points: 15

A now ex-coworker of yours wrote a function Find() and left a few comments, but very limited documentation. You know that the function takes two arguments. The first is a pointer to an array (passed in r1) and the second is the number of elements in the array (passed in r2). The return value is found in r5.

```
        lw      0       1       ArrayS
        lw      0       2       Num
        lw      0       3       Fcall
        jalr    3       7
        halt
Find    lw      0       6       NegOne  // r6=-1
        add     0       2       5       // r5=Num
Top     add     2       6       2       // Decrement Num
        add     1       2       4       // 4 is address
        lw      4       3       0       // ld array element
        beq     3       0       skip    // is  element 0?
        add     5       6       5       // if not sub 1
skip    beq     0       2       Done    // if Num=0 we are done
        beq     0       0       Top     // next iteration
Done    jalr    7       3               // return
Zero    .fill   0
NegOne  .fill   -1
Fcall   .fill   Find
ArrayS  .fill   Array
Num     .fill   8
Array   .fill   0
        .fill   1
        .fill   7
        .fill   2222
        .fill   0
        .fill   777
        .fill   0
        .fill   1
        .fill   -5
```



a) How many times will the line with the label “Find” get executed during this program? **[2]** ____1_____

b) What is the value of r7 when the program halts? **[3]** ____4_____

c) What is the value in r5 when the program halts? **[5]** _____3_____

d) *In 10 words or less,* describe what the return value of the function is in terms of the array. Your answer should be something like “returns the maximum value in the array”. **[5]**

Returns the number of zeros in the array

uniqname: _____

## Problem 7: Hunt You Down: $Cache and Page Table. Points: 20

Consider a byte-addressable architecture with 12-bit virtual addresses. The system uses 1KB of physical memory with 64B page sizes. The system has a 16 byte 2-way set associative cache with a 2-byte block size and a fully-associative TLB with 8 entries. The TLB, cache contents, and a snippet of the single level page table are provided below:

**Page Table**

| VPN | PPN | Valid |
|---|---|---|
| 0x10 | 0x3 | 1 |
| 0x11 | 0xF | 1 |
| 0x12 | 0x3 | 0 |
| 0x13 | 0xF | 0 |
| 0x14 | 0x2 | 1 |
| 0x15 | 0x1 | 1 |
| 0x16 | 0xE | 0 |
| 0x17 | 0x2 | 0 |

| VPN | PPN | Valid |
|---|---|---|
| 0x18 | 0x4 | 0 |
| 0x19 | 0x8 | 1 |
| 0x1A | 0x7 | 1 |
| 0x1B | 0xA | 1 |
| 0x1C | 0x5 | 1 |
| 0x1D | 0x4 | 0 |
| 0x1E | 0xD | 0 |
| 0x1F | 0x1 | 1 |

**TLB**

| Tag | PPN | Valid |
|---|---|---|
| 0x05 | 0xB | 1 |
| 0x1F | 0x1 | 1 |
| 0x00 | 0xB | 0 |
| 0x17 | 0xA | 0 |
| 0x11 | 0xF | 1 |
| 0x0E | 0xA | 0 |
| 0x0D | 0x0 | 1 |
| 0x02 | 0x3 | 0 |

**Cache**

| Set Index | Tag | Valid | Byte 0 | Byte 1 |
|---|---|---|---|---|
| 0 | 0x37 | 1 | 0xDE | 0xAD |
| | 0x1A | 0 | 0x12 | 0xB0 |
| 1 | 0x26 | 0 | 0x0A | 0xC1 |
| | 0x33 | 1 | 0x99 | 0x1F |
| 2 | 0x1C | 1 | 0x84 | 0x92 |
| | 0x00 | 1 | 0xBE | 0xCF |
| 3 | 0x7B | 1 | 0xCC | 0xA0 |
| | 0x0A | 1 | 0x45 | 0x67 |

a) Say that the processor reads one byte from virtual address **0x71A** and that the cache is ***physically indexed***. Answer the following questions. Provide all numeric answers in hex. If the answer is unknown, write "unknown". Note that early errors on this problem could cause later answers to also be wrong.

i) What is the virtual page number associated with that address? **[2]**
0x1C

ii) Is this a TLB hit? **[2]**

No

iii) What is the physical page number associated with this access? **[2]**

5

iv) What is the set number where the data could be found? **[2]**

1

v) What is the value of that byte of memory being accessed? **[2]**

Tag=0x2B, **unknown**



b) Now say that the processor reads one byte from virtual address **0x45F** and that the cache is ***virtually indexed***. Answer the following questions. Provide all numeric answers in hex. If the answer is unknown, write "unknown". Note that early errors on this problem could cause later answers to also be wrong.

i) What is the virtual page number associated with that address? **[2]**
0100 0101 1111, 010001 for the VPN, **0x11**

ii) Is this a TLB hit? **[2]**
Yes

iii) What is the physical page number associated with this access? **[2]**

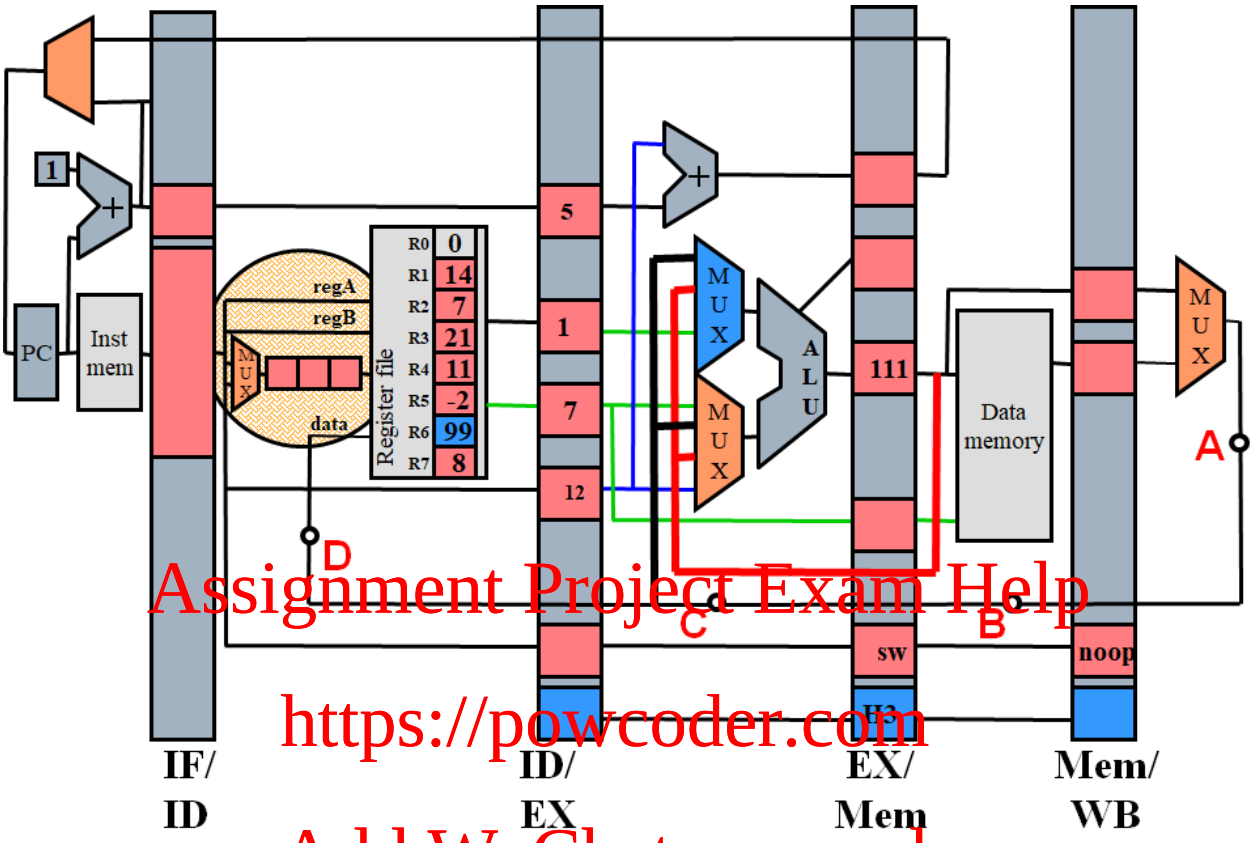
0xF

iv) What is the set number where the data could be found? **[2]**
0x3

v) What is the value of that byte of memory being accessed? **[2]**
Tag=010001011 =0x8B, **unknown**

uniqname: _____

## Problem 8: Pipes — Points: 18






Pipeline Datapath for Reference

Consider the pipeline from lecture (above) with times below, using **detect and forward** for data hazards and **speculate not taken and squash** for control hazards. The current cycle time is 30ns.

- Data Memory Read/Write: 30 ns
- Instruction Memory Read: 8 ns
- ALU: 20 ns
- Register file read/write: 5 ns
- Backwards wire delay: 2 ns per stage *
- All other wires: 0 ns
- All other delays: 0 ns

* Wires that go backwards across pipeline stages (eg. Mem/WB to the Register File) have a delay of **2 ns per stage**. Thus the delay from point **A** (the black circle labeled A in WB) to **B** (in MEM) is 2 ns, the delay from **B** to **C** is another 2 ns and the delay from **A** to **D** 6 ns.

The current pipeline is: IF → ID → EX → MEM → WB

Suppose we have the option to add new stages to the pipeline and split memory transactions and ALU operations into ***up to* 4 stages each**. (e.g. we could split the ALU into two stages to create a new pipeline)

IF → ID → EX1 → EX2 → MEM1 → WB

In this new pipeline, the 20 ns ALU operation is divided into two 10 ns operations, one in EX1 and one in EX2. However, the delay from point **A** (in WB1) to **D** (in ID1) is now 8 ns. (Branches always resolve in MEM1, data can only be forwarded from MEM1 and WB1, and data is always forwarded to EX1).

a) Design a pipeline that will have the lowest clock period. **[7]**
**Pipeline:** Example: IF → ID → EX1 → EX2 → MEM1 → MEM2 → WB

__IF_____ →__ID___ →__EX1__ →___EX2__ →_EX3_ →_MEM1_ →_MEM2_ →
__WB___ →_____ →_____ →_____



Critical path is from WB to ID to register file read complete (12ns+5ns)



b) What is the clock period of your design for part a? **[3]** ____17ns_____

c) Given your design for part a, *what would be your CPI on the following program?* **[8]**



The program has 1,000 instructions and the following characteristics:

- 50% add/nor; 20% are followed by an immediately dependent instruction
- 20% lw; 20% followed by an immediately dependent instruction
- 20% beq; 40% taken
- 10% sw

There are no data hazards other than those mentioned above. **You must *clearly* show your work to receive credit.**

100 add/nor with dep. Instruction (stall for add/nor to finish EX3—2 cycles)
40 lw with dependent instruction (stall for lw to finish MEM2 – 4 cycles)
80 mispredicted branches. (squash IF, ID, EX1, EX2, EX3 so 5 instructions)

Cycles = 1000 (instructions) +7 (pipeline startup) + 100*2 (nor/nand) +
40*4 (lw) + 80*5 = 1767 cycles.

**CPI=**__1.767_____

uniqname: _____

## Problem 9: No School Like Old School — Points: 16

Your EECS 280 program is generating accesses on a processor with a cache. The cache has a block size of 4 bytes. The only data structure in your program is an array with 4096 elements. Each array element is 1 byte. There is one input parameter (**stride**) to your program. Your program reads the array according to the parameter stride in the following pattern (yes, this is an infinite loop).

```
byteAddress = 0;
while (1) {
      read array[byteAddress % 4096];
      byteAddress += stride;
}
```

For example, with stride = 4, the access pattern would be: byte 0, 4, 8, 12, 16, ... , 4088, 4092, 0, 4, 8... Ignore the accesses during the first time through the array (i.e. give the miss rate after the program has been running for a very long time and has gone through the entire array many times).

a) Assume the CPU cache is **direct-mapped**. Fill in each entry of the following table with the CPU cache's **miss rate** for a given combination of stride and cache size in bytes. **[8 points, -1 per wrong/blank answer, minimum 0]**



| Stride | Cache Size in Bytes | | | |
|---|---|---|---|---|
| | 512 | 1024 | 2048 | 4096 |
| 1 | .25 | .25 | .25 | 0 |
| 4 | 1 | 1 | 1 | 0 |
| 16 | 1 | 1 | 1 | 0 |

b) Now assume the CPU cache is **fully associative and LRU**. Fill in each entry of the following table with the **miss rate** for a given combination of stride and cache size in bytes. **[8 points, -1 per wrong/blank answer, minimum 0]**

| Stride | Cache Size in Bytes | | | |
|---|---|---|---|---|
| | 512 | 1024 | 2048 | 4096 |
| 1 | .25 | .25 | .25 | 0 |
| 4 | 1 | 1 | 1 | 0 |
| 16 | 1 | 0 | 0 | 0 |