COLLEGE OF ENGINEERING
COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF MICHIGAN

EECS 370 Final Exam
Fall 2018
SOLUTIONS

Notes:

- Closed book.
- You are allowed one cheat sheet (notes on both sides allowed)
- **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 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: ______________________________

Name: ______________________________

Uniqname: ______________________________

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):

____

Problem 1. True / False Points: ___/13

	True	False
a. A 4-bit 2's complement number can represent values [-8, 8].	○	●
b. Branch prediction with our 5-stage pipeline will always have a lower CPI than detect and stall.	○	●
c. Caller and callee save registers are **not** defined by the hardware architecture.	●	○
d. Sign extension can be used on both loads and stores to preserve the sign (negative or positive) of data.	○	●
e. The number 1.0 * 2^(-127) cannot be represented in 32-bit IEEE-754 floating point format.	●	○
f. When an integer is loaded from memory into a register, endian-ness defines the order bytes are placed in the register.	●	○
g. A control ROM is sequential logic, not combinational.	○	●
h. Function parameters passed in memory are always stored on the stack.	●	○
i. LRU is a concept used to maximize temporal locality within a direct mapped cache.	○	●
j. The relocation table of an object file contains the location of references to symbols in the text section that have to be fixed up during linking.	●	○
k. Write-through, no-allocate caches make better use of temporal locality than write-back, write-allocate caches.	○	●
l. The virtual address space is limited in size by the amount of physical memory (DRAM) on a computer.	○	●
m. Adding an extra stage to a pipelined processor will greatly increase the CPI of a 100,000 instruction test bench. *Assume the clock period, number of hazards, and stalls due to hazards are unchanged.*	○	●

Problem 2: LC2K Assembly **Points: ___/15**

Convert the following C-code to LC2K assembly. For the assembly code, assume all registers are initialized to 0 and the arrays `arr1` and `arr2` are stored somewhere else in memory.

Hints:

- `ind1 → R3, ind2 → R4, num → R5, 2147483647 == 0x7FFFFFFF`
- To do `X < Y` comparison in assembly, compute `-Y`; add `X + (-Y)`; extract the signbit of `X - Y` and compare to `0`.

C-code		LC2K assembly				
`int size = 5;`	0		lw	0	1	one
`int arr1[5] = {1, 3, 5, 7, 9};`	1		lw	0	2	size
`int arr2[5] = {2, 4, 6, 8, 10};`	2	while	add	2	**2**	7
`int dest[10];`	3		beq	5	7	end
`int ind1 = 0, ind2 = 0;`	4	if	beq	4	2	**true**
`int num = 0;`	5		beq	**3**	2	**else**
	6		lw	**3**	6	arr1
`while (num != (size * 2)){`	7		lw	**4**	7	arr2
`  if (ind2 == size ||`	8		nor	7	**7**	7
`     (ind1 != size &&`	9		add	1	7	7
`      arr1[ind1] < arr2[ind2])) {`	10		add	6	**7**	6
`    dest[num] = arr1[ind1];`	11		lw	0	7	mask
`    ind1++;`	12		nor	**6/7**	**7/6**	6
`    num++;`	13		beq	6	0	**true**
`  } else {`	14	else	lw	4	6	arr2
`    dest[num] = arr2[ind2];`	15		sw	**5**	6	dest
`    ind2++;`	16		add	1	4	4
`    num++;`	17		add	1	5	5
`  }`	18		beq	0	0	**while**
`}`	19	true	lw	3	**7**	arr1
	20		sw	5	7	**dest**
	21		add	1	**3**	3
	22		add	1	5	5
	23		beq	0	0	**while**
	24	end	halt			
	25	one	.fill 1			
	26	size	.fill 5			
	27	mask	.fill 2147483647			
	28	dest	.fill 0			
		...				

Problem 3: Repeat Performance Points: ___/10

You are given a benchmark test case with the following makeup:

20% beq	30% lw	10% noop	10% sw	30% add

Assume noop takes 2 cycles in multicycle datapath. In this benchmark, for a pipeline datapath, assume *detect-and-stall* is used for all data hazard resolution and *speculate-and-squash* is used for all control hazard resolution. Also, note that:

- 40% of the beq instructions are incorrectly predicted
- 30% of the lw instructions are immediately followed by a dependent instruction
- 30% of the add instructions are immediately followed by a dependent instruction

Memory Read: 15 ns	Memory Write: 20 ns	Register read/write: 10 ns
ALU: 5 ns	All other operations: 0 ns	

Assume that there is internal register forwarding, the pipeline has five stages, and instructions are read from memory.

a. What is the clock period (in nanoseconds) for the following datapaths?
*Your answers must be numbers, **not** numeric equations.*

- Single Cycle: **55ns (lw: 15 + 10 + 5 + 15 + 10)**
- Multicycle: **20ns**

- Pipelined: **20ns**

b. What is the CPI of the benchmark on the following datapaths?
Your answers may be numeric equations. Simplify them as much as possible.

- Single Cycle: **1 CPI**
- Multicycle: **4.1 CPI = 2(0.10) + 4(0.60) + 5(0.30)**
- Pipelined: **1.6 CPI = 1 + 3(0.20)(0.40) + 2(0.60)(0.30)**

c. A new memory upgrade comes out that reduces the latency of memory reads to 5 ns. All other latencies are unchanged. Now what is the clock period (in nanoseconds) for the following datapaths?
*Your answers must be numbers, **not** numeric equations.*

- Single Cycle: **40ns (sw: 5 + 10 + 5 + 20)**
- Multicycle: **20ns**
- Pipelined: **20ns**

Problem 4. Detecting Hazards in LC2K Points: ___/15

We decide to use the following program as a benchmark for our 5-stage LC2K pipeline. Our pipeline uses detect-and-forward to handle data hazards and speculate and squash (predict not-taken) to handle control hazards.

```
 1          lw     0      1      posOne   # Load +1 into reg 1
 2          lw     0      2      a        # Load a into reg 2
 3          lw     0      3      b        # Load b into reg 3
 4  loop    beq    2      3      end      # If a == b, then branch
 5          nor    2      2      4        # Negate a (i)
 6          add    4      1      4        # Negate a (ii)
 7          add    4      3      5        # Calculate b - a
 8          lw     0      6      mask     # Determine if a > b (i)
 9          nor    5      5      5        # Determine if a > b (ii)
10          nor    6      6      6        # Determine if a > b (iii)
11          nor    5      6      6        # Determine if a > b (iv)
12          beq    0      6      less     # If a <= b, then branch
13          nor    3      3      4        # Negate b (i)
14          add    4      1      4        # Negate b (ii)
15          add    2      4      2        # Set a = a - b
16          beq    0      0      loop     # Branch back to loop
17  less    add    3      4      3        # Set b = b - a
18          beq    0      0      loop     # Branch back to loop
19  end     sw     0      2      result   # Store a into memory
20          halt                          # End program
21  posOne  .fill  1
22  mask    .fill  -32768                 # 0x8000
23  a       .fill  18
24  b       .fill  12
25  result  .fill  1
```

a. How many stall cycles are added due to data hazards throughout the lifetime of the program? **1 cycle**

b. How many stall cycles are added due to control hazards throughout the lifetime of the program? **12 cycles**

c. This benchmark executes 30 *instructions* in total. How many *cycles* does this program take to execute (including stall cycles and loading the pipeline)? **1 + 12 + 30 + 4 = 47 cycles**

d. Given that this benchmark executes 30 instructions, what is the CPI of this benchmark? **47/30 = 1.57 CPI**

e. Write two lines we could swap with each other to reduce the total number of stalls that would occur due to data hazards. **Lines 1 & 3**

Problem 5. Do You C What I C? Points: ___/15

Consider an 8-bit byte-addressable architecture that uses a *32-byte, 2-way set-associative* cache with a block size of *8 bytes*. The cache is initially empty, and uses LRU replacement policy.

Fill out the following table for the given sequence of memory accesses. If the access is a hit, fill in the "Hit" bubble. If the access is a miss, indicate the type of miss (Compulsory / Capacity / Conflict) by filling in the bubble in the correct column.

		Misses		
Address	**Hit**	**Compulsory**	**Capacity**	**Conflict**
0x3E	○	●	○	○
0xAE	○	●	○	○
0x3C	●	○	○	○
0x35	○	●	○	○
0xB3	○	●	○	○
0x77	○	●	○	○
0x30	○	○	○	●
0x7F	○	●	○	○
0xAF	○	○	●	○
0x70	●	○	○	○
0xB4	○	○	●	○

Problem 6. Level Three Points: ___/15

Consider a 64-bit byte-addressable system that supports virtual memory with the following specifications:

- A page is 2 KB.
- The page table is hierarchical with 3 levels.
- The first-level occupies 4 pages of memory.
- Each second-level occupies 8 pages of memory.
- A page table entry is 8 B and is the same for all levels.
- 32 GB of physical memory is installed.
- Virtual addresses are 64 bits

1. How many bits are used for the page offset?

   Ans: **11 bits**

2. How many virtual pages exist in this system? How many physical pages exist?

   # Virtual Pages: **2^{53} pages**

   # Physical Pages: **2^{24} pages**

3. How many bits in a virtual address are used to index the first-level page table?

   Ans: **10 bits**

4. How many bits in a virtual address are used to index a second-level page table?

   Ans: **11 bits**

5. How many bits in a virtual address are used to index a third-level page table?

   Ans: **32 bits**

6. How much memory does this page table system occupy in the worst case? *Express your final answer* ***in bytes*** *and in powers of 2. Do not simplify (e.g. if your answer is $2^x + 2^y$, leave it at that).*

   Ans: **$2^{13} + 2^{24} + 2^{56}$ B**

7. If this system used a single-level page table instead of a three-level page table, how much memory would it occupy? Express your final answer in bytes and in powers of 2. Do not simplify (e.g. if your answer is $2^x + 2^y$, leave it at that).

   Ans: **2^{56} B**

Problem 7. Historical Eviction Points: ___/20

It's 1987 and Apple has just released the Macintosh II featuring a Motorola 68030 processor. The 68030 contains a state of the art *64 byte data-cache* and uses *byte-addressable memory*. Having taken 370, you know there's more to cache design than just size, so you write a C program to help you determine the block size, number of blocks per set, and number of sets in the cache. *Assume all variables other than the data[ ] array are stored in registers and that the cache has been cleared before each call to miss_rate(...).*

```
#define CACHE_SIZE XX
double miss_rate(int stride) {
  char data[CACHE_SIZE * 2]; /* note array is 2x cache size */
  int misses = 0;

  for (int i = 0; i < stride; i++)
    for (int j = i; j < (CACHE_SIZE * 2); j += stride)
      if (ACCESS(data[j]) == CACHE_MISS))
        misses++;

  return (misses / (CACHE_SIZE * 2));
}
```

a. In order to test your code, you first run it on a *32 B fully-associative* cache with *8 B blocks*. You set the `CACHE_SIZE` macro to `32`. Fill in the miss rate for each stride in the following table. *Only answers in the table will be graded. Fractions are OK.*

Stride	Miss Rate
1	0.125
2	0.25
4	0.5
8	1
16	0.125
32	0.125

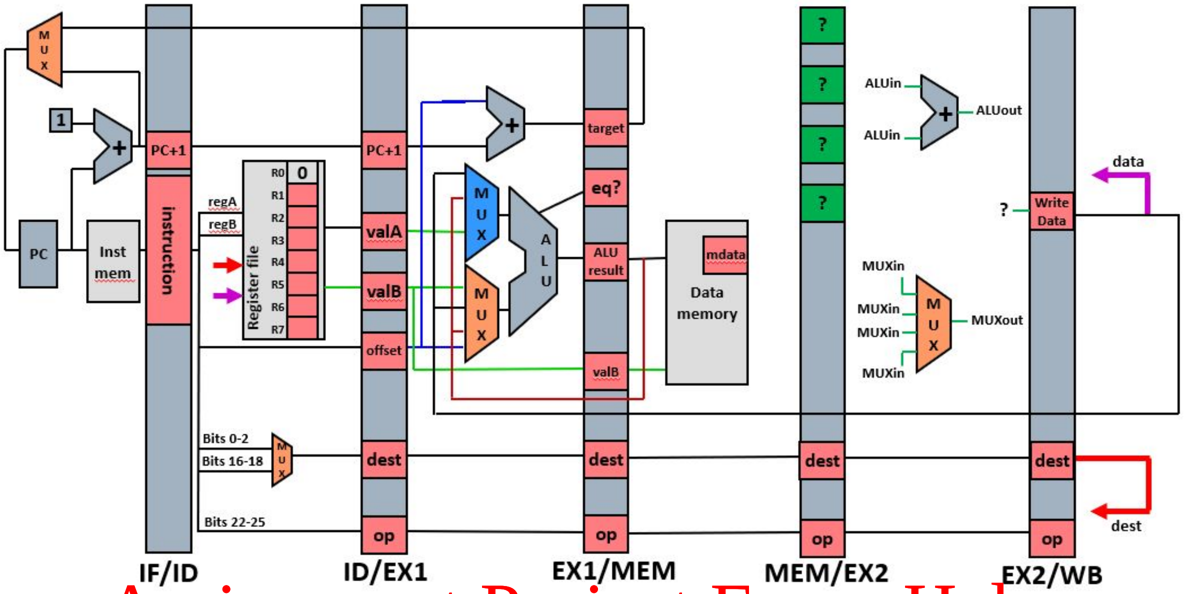
b. You now run your program on the Motorola 68030 processor with a *64 byte data-cache*. You set the `CACHE_SIZE` macro to `64` and collect the following data:

Stride	Miss Rate
1	0.25
2	0.50
4	1.00
8	1.00
16	1.00
32	1.00
64	0.25

Fill in one bubble completely for each question.

	64	32	16	8	4	2	1
What is the block size in bytes?	○	○	○	○	●	○	○
What is the number of blocks per set?	○	○	○	○	○	●	○
What is the number of sets?	○	○	○	●	○	○	○

Problem 8. Modified Pipeline Datapath Points: ___/20



We once again want to upgrade our datapath to run the new LC2K instruction `madd`.

The instruction has the following functionality:

`madd regA regB offset` **`# regB = regB + Mem[regA + offset]`**

To accommodate this new instruction, the pipeline has been expanded to 6 stages:
`Fetch, Decode, Execute1, Memory, Execute2, Writeback.`

a. What values from the `EX1/MEM` pipeline registers and `Data memory` need to be sent to the `MEM/EX2` pipeline register in order to allow for correct execution?

- Select up to four connections to be made to the `MEM/EX2` stage
- Fill in the bubble next to any values you want to send to `MEM/EX2`

ALUresult	●
valB	●
mdata	●
eq?	○
target	○

b. Using the values you assigned in part a, make connections to the components in the `Execute2` stage and the `WriteData` block in the `EX2/WB` pipeline register.

- Up to 4 connections can be made to `MUX` (may not need all)
- Up to 2 connections can be made to `ALU` (may not need all)
- Possible connections are `ALUout`, `MUXout`, and any connections you made in part a.

ALUout______ to `MUXin` **mdata**_______ to `ALUin`

ALUresult__ to `MUXin` **valB**________ to `ALUin`

mdata_______ to `MUXin`

____________ to `MUXin` **MUXout**______ to `WriteData`

c. Assuming all connections are correctly done (independent of part a/b), fill out the timing diagram for the following program. Assume data hazards are handled using detect-and-forward, with stalls are inserted when necessary. Use the symbol * for stalls.

```
        lw      0    1    five     # lw1
        lw      0    2    six      # lw2
        add     1    2    3        # add1
        madd    0    3    ten      # madd
        add     3    3    4        # add2
        halt                       # halt
five    .fill   5
six     .fill   6
seven   .fill   7
eight   .fill   8
nine    .fill   9
ten     .fill   10
```

Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw1	IF	ID	EX1	MEM	EX2	WB									
lw2		**IF**	**ID**	**EX1**	**MEM**	**EX2**	**WB**								
add1			**IF**	*****	**ID**	**EX1**	**MEM**	**EX2**	**WB**						
madd					**IF**	**ID**	**EX1**	**MEM**	**EX2**	**WB**					
add2						**IF**	*****	*****	**ID**	**EX1**	**MEM**	**EX2**	**WB**		
halt									**IF**	**ID**	**EX1**	**MEM**	**EX2**	**WB**	

Problem 9. Virtual Memory + Data Cache Points: ___/15

A. Consider a system with the following specs:
   - 1 MB physical memory
   - 32-bit virtual address
   - Single-level Page Table
     - 64 KB page size
   - 64 KB Data-cache
     - Direct Mapped
     - 4 KB block size

Determine the cache tag, set index, and block offset for virtual address `0x0EEC5370` if the system uses a virtually addressed data-cache. Then, do the same for a physically addressed data-cache. *Answers must be in hexadecimal. Assume the virtual page for this address maps to physical page `0x7`.*

a. Virtually addressed data-cache

   - Tag: 0x**0EEC**

   - Set Index: 0x**5**

   - Block Offset: 0x**370**

b. Physically addressed data-cache

   - Tag: 0x**7**
   - Set Index: 0x**5**
   - Block Offset: 0x**370**

B. Consider a virtual memory system with **two-level hierarchical page-table**, TLB, *virtually addressed* data-cache, and the following specs:

- ○ TLB
  - Access time: 1 cycle
- ○ Data-Cache
  - Access time: 1 cycle
- ○ Memory
  - Access time: 30 cycles
- ○ Disk
  - Access time: 100,000 cycles
- ○ *NOTES*
  - The TLB and Data-cache are **accessed in parallel**
  - On a 1st level page fault, the creation of a 2nd level page table occurs in parallel with accessing the disk
  - The page table cannot be cached and is available in main memory
  - Pages that aren't in memory are on disk
  - Upon retrieval from cache, main memory or disk, the data is sent immediately to the CPU, while other updates occur in parallel

- What are the possible memory access latencies, in cycles, for the following situations? *For the following three questions, there may be one or more answers. Include all possible answers. Your answer(s) must be a number, not a numeric equation.*

a. When data is in the data-cache? Ans: **1 cycle**

b. When data must be fetched from memory? Ans: **31 and 91 cycles**

Hit TLB: 1 (dcache/tlb) + 30 (memory)

Miss TLB, hit page table: 1 (dcache/tlb) + 30 (1st level pt) + 30 (2nd level pt) + 30 (memory)

c. When data must be fetched from disk? Ans: **100,031/100,061 cycles**

miss in 1st level page table: 1 (dcache/tlb) + 30 (1st level pt) + 100,000 (disk)

miss in 2nd level page table: 1 (dcache/tlb) + 30 (1st level pt) + 30 (2nd level pt) + 100,000 (disk)

Problem 10. The Lost Page Table Points: ___/20

A process, PID=739, just crashed, and we would like to recover some of its data. In order to recover its memory, we need *you* to find the value of the **page table base register**.

Right before the crash, a portion of physical memory (bytes given **in hexadecimal)** was saved:

Address	Data from <Address + 0x0> to <Address + 0xF>															
	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0x30	89	DF	EE	83	2F	AA	B2	C7	EF	A9	2A	33	85	01	A0	29
0x40	22	46	BE	4D	32	2F	B9	CC	AB	13	90	60	24	A0	D8	34
0x50	5A	B7	D4	50	E9	A8	24	38	A7	E9	13	8B	6A	3B	22	41
0x60	46	D9	B7	C5	88	91	25	67	44	22	EE	FF	BB	A2	1F	7D
0x70	16	95	2F	B0	E7	E9	B2	10	19	14	BE	C0	FF	EE	25	66
0x80	78	55	20	10	14	D0	3A	3F	20	6D	3B	C5	D4	A9	FC	10
0x90	00	07	0F	A2	0A	0C	6F	88	16	14	13	08	00	C3	21	6B
0xA0	31	26	2B	45	4F	35	8A	09	CC	4D	44	0B	4A	C9	8A	89
0xB0	A0	BE	3B	29	85	86	75	30	16	BA	EE	C5	0A	24	08	09

Figure 1: Physical Memory before crash (bytes given in hexadecimal)

Process 739's memory accesses (from process start to crash) are reproduced below.
You may **assume that:**

- no other processes are running in the system.
- process 739 had no other memory accesses besides those in Figure 2

Hint: When the process started, its page table was empty.

Time (ms) before crash	Virtual Address	Page Fault	Data Retrieved
13	0x8A	Yes	0xEE
8	0x92	Yes	0xBE
3	0x83	No	0xC5
1	0xAC	Yes	0x85

Figure 2: Virtual Memory Address accesses from process 739

You are given the following clues:

- Virtual Addresses are 10 bits (10-bit byte addressable system)
- 256 bytes of physical memory
- Single-level page table
- All sizes are powers of 2
- Page table entries are 8 bits (1 byte) and of the form:

← 8 bits →		
<unused>	<valid bit>	<PPN>

$\# \, unused \; bits \; + \; \# \, PPN \; bits \; = \; 7$

1. What is the page size? Ans: **16 B**

2. How many page table entries are in the page table? Ans: **2^6 = 64 entries**

3. How many bits is a physical page number (PPN)? Ans: **4 bits**

4. How are Virtual Addresses and Physical Addresses composed? Recall that these can be broken into [VPN || Page Offset] and [PPN || Page Offset], respectively. Identify the number of bits for each below:

6 bits	**4 bits**
Virtual Page Number (VPN)	Page Offset

4 bits	**4 bits**
Physical Page Number (PPN)	Page Offset

5. For each of the 4 virtual address accesses, write down the Physical Page Number (PPN)

Time (ms)	Virtual Address	Page Fault	Data Retrieved	PPN
13	0x8A	Yes	0xEE	0x**6**
8	0x92	Yes	0xBE	0x**4**
3	0x83	No	0xC5	0x**6**
1	0xAC	Yes	0x85	0x**3**

6. What is the value of the Page Table Base Register (PTBR), and thus the address of the start of the page table?

Starting Address of Page Table: **0x90**

Problem 10 Explanation:

1. **Know page size is >= 16 b/c 0x83 does not cause page fault. Know page size < 32 since 0x92 does result in a page fault**
2. **Page size = 16b -> Page offset = 4 bits. # PTE entries = VPN bits = VA - page offset = 2^6 entries**
3. **P.A. = log(256) = 8 bits. P.A. - page offset = 8 - 4 = 4 bits.**
4. **(follows from 1-3)**
5. **Asdf**
    a. **0x8A -> look for "A" column where data is 0xEE -> row 0x60 or 0xB0**
    b. **0x92 -> look for "2" column where data is 0xBE -> row 0x40**
    c. **0x83 -> look for "3" column where data is 0xC5 -> row 0x60 (so we know 0x8A is in row 0x60 too, since a,c are on the same page)**
    d. **0xAC -> look for "C" column where data is 0x85 -> row 0x30**
6. **Look for row where offset 8,9,A are 6, 4, 3, respectively. This happens in row 0x90.**