The University of Michigan - Department of EECS
EECS 370 – Introduction to Computer Architecture

Final Exam – ANSWER KEY
December 15th, 2010

Name: _____

University of Michigan uniqname: _____
(NOT your student ID number!)

Open book, open notes. No laptops, PDAs, cell phones, etc. (calculators are ok). Questions vary in difficulty; it is strongly recommended that you do not spend too much time on any one question. For questions where a box is provided, please put your final answer in the box.

The rules of the Honor Code of the University of Michigan - College of Engineering apply for this exam. Honor code pledge: "I have neither given nor received aid on this examination, nor have I concealed any violations of the Honor Code".

Signature: _____
(Exams without a signed pledge will not be graded)

Question	Score/Point Value
1. [easy] Memory alignment	/3
2. [easy] Linker/loader	/3
3. [easy] Memory addressing	/6
4. [medium]Caller/callee-saved variables	/6
5. [medium]LC-2K assembly revisited	/10
6. [medium]Virtual memory system design	/12
7. [medium]Floating point	/10
8. [hard] Caches	/14
9. [medium]Multi-cycle datapath	/10
10. [hard] Pipeline timing	/17
11. [medium]Physically addressed caches	/12
12. [medium]Processor performance	/11
13. [medium]Memory system performance	/10
TOTAL	**/124**

1. Memory Alignment [3 points]

How many bytes does the following struct use in memory, assuming that the first available memory location is 0d100 (decimal 100)? Assume we have a 32-bit byte-addressable architecture. **To receive credit for your answer you must also fill in the boxes next to the code** with start and end addresses for each variable.

```
struct a {
     char *b;
     char c[3];
     short d;
};
```

	start		end
char *b;	100	-	103
char c[3];	104	-	106
short d;	108	-	109

The structure requires [12] bytes of memory.

2. Linker/Loader [3 points]

Given the following code, where are the variables located in memory? Circle the appropriate answer.

```
int a[10];
void function(char *c) {
     static int b;
     b = 5;
     double *d = (double *) malloc( b * sizeof(double) );
     c[1] = 'a';
}
```

Variable	Location		
a	Heap	Stack	Static
b	Heap	Stack	Static
c	Heap	Stack	Static
d	Heap	Stack	Static
*d	Heap	Stack	Static

3. Memory Addressing [6 points]

In the MIPS code below, what value is in register $1, after the last instruction is executed? Remember that MIPS is big-endian and byte-addressable. A portion of the initial memory contents is provided. Trace the execution of the code and fill in the table with the new values of any registers changed by each instruction. Then provide your final answer for $1 in the box.

	Register values (in hexadecimal)		
	$1	$2	$3
`lui $1, 0x10`	0x100000		
`lui $2, 0x6`		0x60000	
`and $3, $2, $1`			0x0
`lh $2, 1000($0)`		0xFFFFEF00	
`addi $3, $3, 1`			0x1
`and $1, $2, $3`	0x0		

Initial memory contents	
Address	Content
1000	0xEF
1001	0x00
1002	0x02
1003	0xFF
1004	0xDE

Final value in $1: 0x0

4. Caller-saved and callee-saved variables [6 points]

Below is the "spinning" function (still under development), which we plan to use to determine who, when and where will teach EECS370 next semester. We want to compile it to evaluate the allocation of variables to caller-saved and callee-saved registers. [Note: assume **all** parameters are passed on the stack and then copied to caller-saved or callee-saved registers for computation].

```
char spin (int don, char valeria) {
     int room, time;
     if (don <= 1) return valeria;
     room = 1311; time = 900;
     valeria = spin (don-1, !valeria);
     if (valeria) room += 359;
     valeria = spin (don/2, valeria);
     time += 300;
     if (don % 2) valeria = 0;
     return valeria;
}
```

Your job is to decide how many lw and sw are executed when the spin function is executed **once**. Assume `don>1`. For each variable, analyze both cases: when it is mapped on a callee-saved or a caller-saved register and compile the table below. [Note: a "2" in the table means 1 lw plus 1 sw instruction]. Then, assume we have 2 caller-saved and 2 callee-saved registers and make the best assignment of each variable by circling either option in the last two columns.

	LW+SW if caller-saved	LW+SW if callee-saved
`don`	**4**	**2**
`valeria`	**0**	**2**
`time`	**4**	**2**
`room`	**2**	**2**

caller	(callee)
(caller)	callee
caller	(callee)
(caller)	callee

5. LC-2K Assembly Revisited [10 points]

After graduation you join ShoeLace Inc. Your boss shows you the LC-2K code below. The code has almost no comments and was developed by an employee who had recently disappeared. Your task is to determine what the code computes. Assume that all registers are initially zero.

```
0:         lw    0    7    x
1:         lw    0    1    arg1
2:         add   0    1    2
3:         lw    0    3    arg2
4:  loop   nand  2    7    5
5:         nand  5    5    5
6:         beq   0    5    skip
7:         add   1    6    6
8:  skip   beq   2    3    done
9:         add   2    2    2
10:        beq   0    0    loop
11: done   halt
12: arg1   .fill 1
13: arg2   .fill -2147483648 //0x80000000
14: x      .fill 16910859    //0x01020a0b
```

Before going any further, you decide to explore a few questions.

a) What do instructions 4 and 5 compute? Circle the best option among the choices below.

A. reg5 = reg2 OR reg7

C. reg5 = reg2 XOR reg7

B. reg5 = reg2 AND reg7

D. reg5 = reg2 NOR reg7

b) Fill in the blanks in the sentence below:

"Instruction 9 is equivalent to performing a logical left [left/right] shift operation on the value in register 2 by 1 [a value] bit(s)".

c) How many times is instruction 4 executed?

32 times

d) What is the value contained in register 6 when this program concludes? [In order to receive credit, you must provide comments next to each code line to explain how you computed your answer]

reg6 = **7**

e) Describe what this code computes. Use only one sentence.

It computes the number of set bits (ones) for a given value x

6. Virtual Memory System Design [12 points]

You need to design a 2-level hierarchical virtual memory system. Your system is byte-addressable, it has 2MB of physical memory (and cannot be expanded to support more); memory addresses are 29-bits long. Each page is 1KB in size and each page table entry is 4 bytes. Your design has the following constraints: the superpage table must require precisely one page of storage, while each second-level page table must occupy precisely an integer number of pages. Answer the following questions:

a) How many address bit are dedicated to page offset?

Answer: **10** bits

b) How many bits of the physical address are for physical page number?

Answer: **11** bits

c) How many bits of the virtual address are used to index the super page table? [Show your work to receive credit]

1KB/4B = 256 bytes

Answer: **8** bits

d) How many bits of the virtual address are used to index a second-level page table?

Answer: **11** bits

e) How many pages does each second-level page table occupy? (answer in pages, not KB) [Show your work to receive credit]

2^11*4 = 2^13 (size of a second-level PT)
2^13/1KB = 2^3 =8

Answer: **8** pages

f) In the worst case scenario, how many pages does this page table system occupy? [Show your work to receive credit]

1+8*256 = 2^11 + 1 = 2049

Answer: **2,049** pages

g) In a typical case scenario, where only 8 second-level page tables are needed, how many pages does this page table system occupy? [Show your work to receive credit]

1+8*8 = 65

Answer: **65** pages

7. Floating Point [10 points]

IEEE half precision floating point format is generally similar to the IEEE single precision floating point format taught in class, except for the following:

- 10 bits are stored in the mantissa field
- 5 bits are stored in the exponent field
- the exponent bias is 15

Your job is to compute what decimal value 0xAAAA represents in this format. To receive credit, you must compute your result in three steps as follows:

a) (mantissa) Convert the mantissa to decimal. You can use any of the methods studied in class and provide your answer as a decimal value or a fraction. [Show your work to receive credit]

0xAAAA = 1010101010101010 binary
Fields: Sign=1 Exponent=01010 Mantissa=1010101010
Putting in the leading 1 gives: 1.1010101010
Can shift the binary point 9 places to the right: $1101010101 * 2^{-9}$
Converting to decimal gives $512+256+64+16+4+1=853*2^{-9}$
which is 853/512

(1.666 is good enough for full credit)

Answer: 1.666015625 (or 853/512)

b) (exponent) What power of two must your mantissa value be multiplied by to determine the actual value?

Binary exponent 01010 is ten (10) decimal, subtract bias 15 so 10 – 15 = -5

(accept 1/32 or -5 for full credit also)

Answer: 2^{-5}

c) (final answer) What decimal value does the floating point value 0xAAAA represent? Please provide your answer as a decimal (base 10) number (not a fraction). Provide at least 4 significant digits in your answer.

-(853/512) * (1/32) = -853 / 16384

(-0.05206 is good enough for full credit)

Answer: -0.05206298828125

8. Caches [14 points]

You are given a byte-addressable cache with the following characteristics:

- 8 bit addresses
- Cache size: 64 bytes
- Replacement policy: LRU
- Write policy: write back

A program runs and it generates the following memory accesses. For each access we indicate if it is a hit or miss in the cache. Assume the cache is initially empty.

Instruction	Memory address	Hit/Miss	Write back? (you need to fill this column when answering part c) of the problem)
lw	0xBB	Miss	**N**
sw	0xF7	Miss	**N**
sw	0xF9	Miss	**N**
lw	0xBE	Hit	**N**
sw	0xD5	Miss	**N**
sw	0x37	Miss	**Y**
lw	0xF1	Miss	**Y**

a) What is the block size and associativity of this cache? Provide your answer below and justify it by filling in the blanks in the sentences provided.

Block Size: **8**

- It can't be less than this because of the hit/miss for address(es): BB and BE
- It can't be more than this because of the hit/miss for address(es): F7 and F9

Associativity: **2**

- It can't be less than this because of the hit/miss for address(es): BB,F9, BE
- It can't be more than this because of the hit/miss for address(es): D5,37,F1

b) Based on your answers above, how many bits are used for the tag, set index, and block offset of this cache?

3	**2**	**3**
tag	set index	block offset

c) In the table at the top of the page, mark with an X in the "Write back?" column each access that causes a block to be written back to memory.

9. Multicycle Datapath [10 points]

You have been asked to extend the multicycle LC-2K Datapath as described in class to include a new instruction called `swapped_sw_inc`. The instruction performs a memory-direct store-word operation where regB contains the address to access. The value stored is obtained by adding the content of regA and the offset. **After** the addition completes, regA is also incremented.

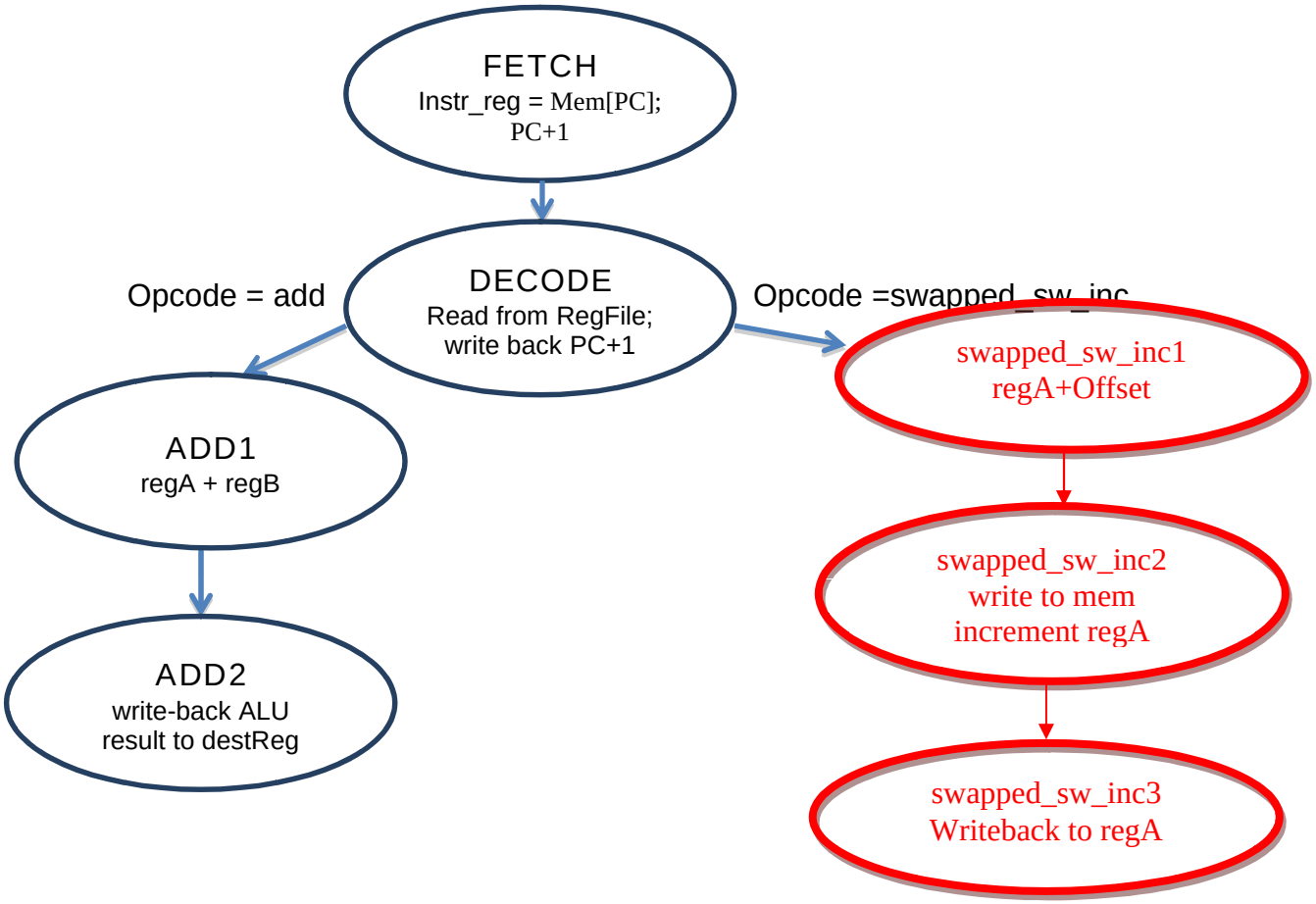
```
swapped_sw_inc regA regB offset  //Mem[regB] = regA++ + offset;
```

The instruction uses the same format as LC-2K lw and sw (shown below):

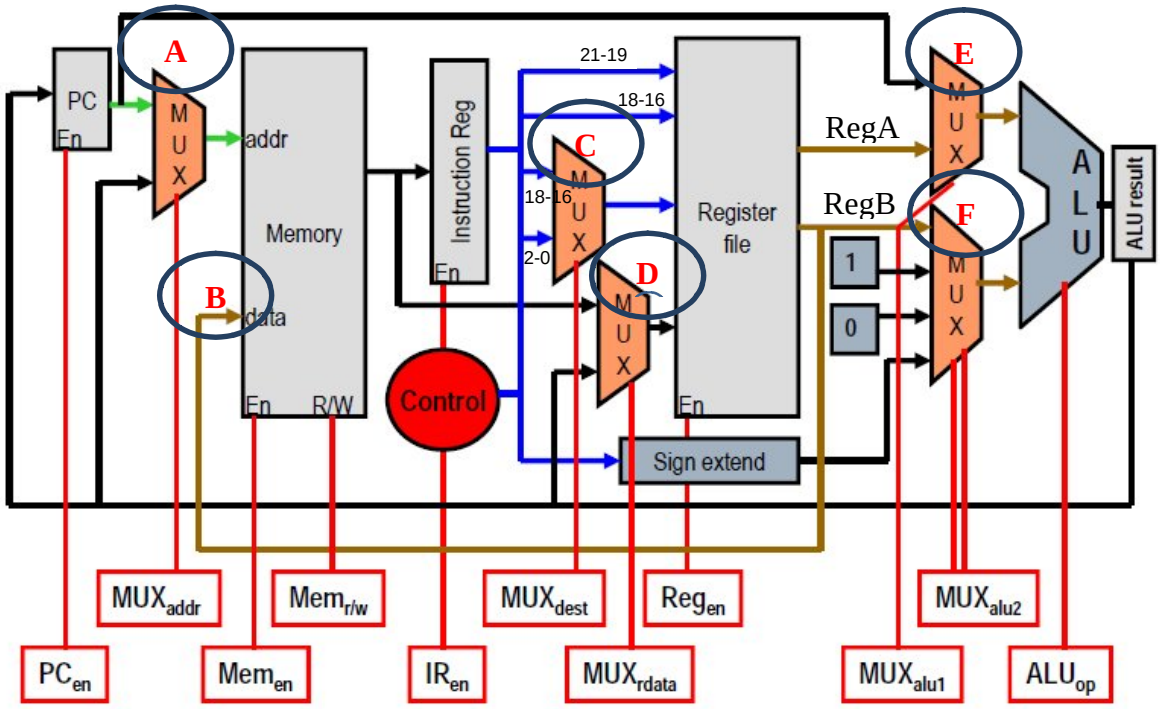
unused	Opcode	regA	regB	offset
31-25	24-22	21-19	18-16	15-0

Note: the LC-2K sw instruction uses regA as base register, and regB for the value to be stored. Thus, our `swapped_sw_inc` in a sense swaps the use of these registers.

a) Consider the FSM implementing the control of our LC-2K multi-cycle datapath and add the **minimum** number of states required to implement `swapped_sw_inc`. We are showing below a portion of the FSM including the states to implement the `add` instruction. You only need to draw the additional states to implement `swapped_sw_inc`, do not be concerned with the other LC-2K instructions. For each state, provide a few words describing what is computed in each of the states that you include (we provide examples of what to write in the states for `add`).



b) What changes do we need to make to the multicycle datapath from class to implement `swapped_sw_inc` ? The datapath is reported below. For each of the circled areas A-F, indicate what changes should be made by circling the appropriate answer below.



Mux A a) Extend mux to take input "RegA" b) Extend mux to take input "RegB"
c) Extend mux to take input "RegA++" d) No changes

Data to Mem. (B) a) Add mux taking input "RegA" b) Add mux taking input "RegB"
c) Add mux taking input "ALU result" d) No Changes

Mux C a) Extend mux to take input "bits 24-22" b) Extend mux to take input "bits 15-0"
c) Extend mux to take input "bits 21-19" d) No changes

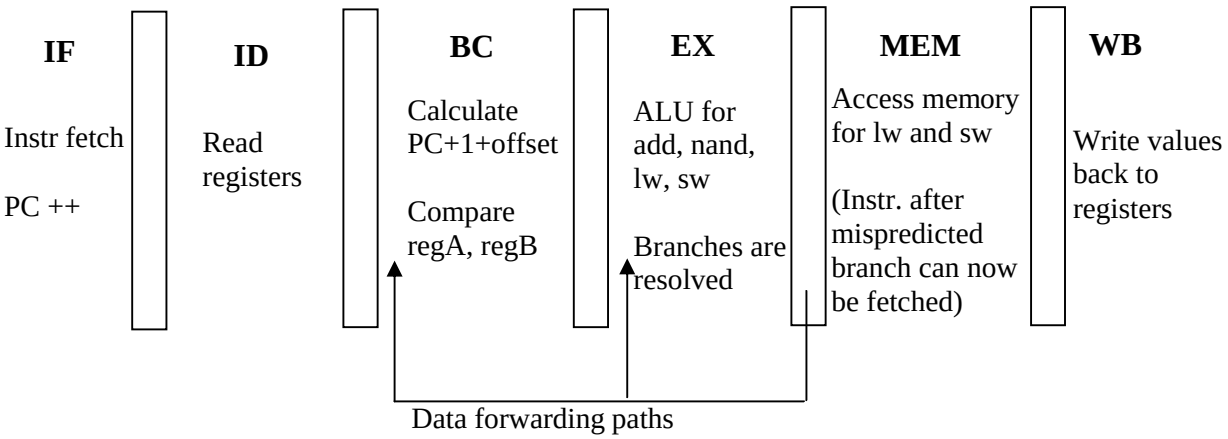
Mux D a) Extend mux to take input "RegA" b) Extend mux to take input "RegB"
c) Extend mux to take input "PC" d) No changes

Mux E a) Extend mux to take input "SignExtend" b) Extend mux to take input "RegB"
c) Extend mux to take input "ALU result" d) No changes

Mux F a) Extend mux to take input "ALU result" b) Extend mux to take input "RegA"
c) Extend mux to take input "PC" d) No changes

10. Pipeline Timing [17 points]

You are working on a LC-2K pipeline that contains 6 stages. There is a new BC stage between the ID and EX that evaluates equality for the branches and calculates the branch offset so that they can be resolved in the EX stage. The only forwarding paths available are EX-to-EX and EX-to-BC. The diagram below provides an overview of this pipeline.



Branches are predicted not taken and mis-predicted branches will be squashed. Any data hazard that cannot be resolved by forwarding will be stalled. All stalls occur in the ID stage.
Your job is to compute how many cycles each of the following programs takes. (Don't forget to include cycles to fill the pipeline). For each program, show your work by filling in the timing diagram -- only up to cycle 12 -- for the instructions provided and determining the number of stalls needed for each dependency.

a) A program of 100 instructions in the form "`add, nand, add, nand`…" (50 add and 50 nand), where each `add` is dependent only on the preceding `nand` and each `nand` is dependent only on the preceding `add`.

	1	2	3	4	5	6	7	8	9	10	11	12
add	IF	ID	BC	EX	MEM	WB						
nand		IF	ID	BC	EX	MEM	WB					
add			IF	ID	BC	EX	MEM	WB				
nand				IF	ID	BC	EX	MEM	WB			

Cycles from the completion of an `add` to the completion of the following `nand`: **1**

Cycles from the completion of a `nand` to the completion of the following `add`: **1**

Total cycles to complete the program [Show your work for credit]: **6+99=105**

b) A program of 100 `lw` instructions, each `lw` depending only on the instruction immediately preceding it.

	1	2	3	4	5	6	7	8	9	10	11	12
`lw`	IF	ID	BC	EX	MEM	WB						
`lw`		IF	ID	ID	ID	ID	BC	EX	MEM	WB		
`lw`			IF	IF	IF	IF	ID	ID	ID	ID	BC	EX

Cycles from the completion of an `lw` to the completion of the following `lw`: **4**

Total cycles to complete the program [Show your work for credit]: **6+4*99=402**

c) A program of 100 instructions in the form "`beq, add, beq, add`…" (50 beq and 50 add), where each `beq` is dependent only on the preceding `add` and all `beq`s are taken. No other dependencies exist.

	1	2	3	4	5	6	7	8	9	10	11	12
`beq`	IF	ID	BC	EX	MEM	WB						
`add`					IF	ID	BC	EX	MEM	WB		
`beq`						IF	ID	ID	BC	EX	MEM	WB
`add`											IF	ID

Cycles from the completion of a `beq` to the completion of the following `add`: **4**

Cycles from the completion of a `add` to the completion of the following `beq`: **2**

Total cycles to complete the program [Show your work for credit]: **10+6*49=304**

11. Physically Addressed Caches [12 points]

The TLB, cache contents, and a portion of the single-level page table are shown below for a byte-addressable system with a 64 byte page size. The system has a 16 byte 2-way set associative cache with a 2 byte block size and a fully associative, 8 entries TLB. Virtual addresses are 12 bits and physical addresses are 10 bits. The cache is **physically indexed.**

Page Table

VPN	PPN	Valid
0x010	A	1
0x011	2	1
0x012	7	0
0x013	C	0
0x014	9	1
0x015	1	0
0x016	8	1
0x017	4	0

VPN	PPN	Valid
0x018	F	0
0x019	D	1
0x01A	3	1
0x01B	0	0
0x01C	E	1
0x01D	5	1
0x01E	6	1
0x01F	B	0

TLB

Tag	PPN	Valid
0x05	E	1
0x2F	B	0
0x00	1	1
0x17	4	1
0x11	2	1
0x1F	B	1
0x0D	0	1
0x02	C	0

2-way cache

Index	Tag	Valid	Byte0	Byte1
0	0x77	1	0xDE	0xAD
	0x1A	0	0x12	0xB0
1	0x26	0	0x0A	0xC1
	0x73	1	0x99	0x1F
2	0x0C	1	0x84	0x92
	0x00	1	0xBE	0xEF
3	0x11	1	0xCC	0xA0
	0x2A	1	0x45	0x67

If the cache returns to the processor data byte 0x67, what was the virtual address that the processor used for the access? To compute your answer, indicate what happens during the address translation process and the cache access by answering the questions below. All numeric answers should be given in **hexadecimal**. [Show your work to receive credit for your answers]

Cache block offset	0x**1**
Cache set index	0x**3**
Cache tag	0x**2A**
Physical address	0x**157**
Page offset	0x**17**
TLB hit (yes/no)	**No**
TLB tag (if yes above)	0x
Virtual page number	0x**1D**
Virtual address	0x**757**

12. Processor Performance [11 points]

Consider an LC-2K benchmark with the following distribution of instructions:

add: 10% nand: 10% lw: 40% sw: 10% beq: 25% noop: 5%

- Branches are always predicted not taken, and 60% of the branches are taken.
- 75% of the lw instructions are followed by an immediate dependency.

Now, consider the following three different implementations of the LC-2K benchmark:

Single-cycle Datapath	**Multi-cycle Datapath** • Takes 5 cycles on loads and 4 cycles on all other instructions	**5-stage Pipelined Datapath** • Register file supports internal forwarding • Data hazards are resolved via forwarding • Branches are resolved in some stage (*not necessarily* MEM stage) • Everything else is as discussed in class

The pipelined implementation operates at 500 MHz and it runs the benchmark above with a CPI of 1.6. You may ignore the impact of memory latencies, i.e. assume all memory references are resolved in one cycle.

a) Assuming a **speculate-and-squash** approach, how many instructions are squashed when a branch is mispredicted in the pipelined implementation? [Show your work to receive credit]

Let number of stages squashed be *x*
Pipeline CPI = 1 + 0.4*0.75*1 + 0.25*0.6*x = 1.6 => x = 2

2 stages

[Fill in the blank in this sentence] "Thus, branches are resolved in the **EX** stage".

b) What is the average execution time per instruction for the pipelined implementation? [Show your work to receive credit]

Average execution time per instruction = 1.6*2 = 3.2 ns

Average execution time per instruction: **3.2** ns

c) Hence, at what clock frequency (in Mhz) must the single-cycle implementation run in order to execute the benchmark in the **same time** as the pipelined implementation? (*Hint*: What is the CPI for the single-cycle implementation?) [Show your work to receive credit]

Single Cycle CPI = 1
Hence, Single Cycle Clock Frequency = 1/3.2 = 0.3125 GHz = 312.5 MHz

Single cycle clock frequency **312.5** MHz

d) Finally, at what clock frequency (in Mhz) must the multi-cycle implementation run in order to execute the benchmark in the **same time** as the pipelined implementation? (*Hint*: What is the CPI for the multi-cycle implementation?) [Show your work to receive credit]

Multi Cycle CPI = (0.1 + 0.1 + 0.1 + 0.25 + 0.05)*4 + 0.4*5 = 4.4
Hence, Multi Cycle Clock Frequency = 4.4/3.2 = 1,375 MHz

Multi-cycle clock frequency **1,375** MHz

13. Memory System Performance [10 points]

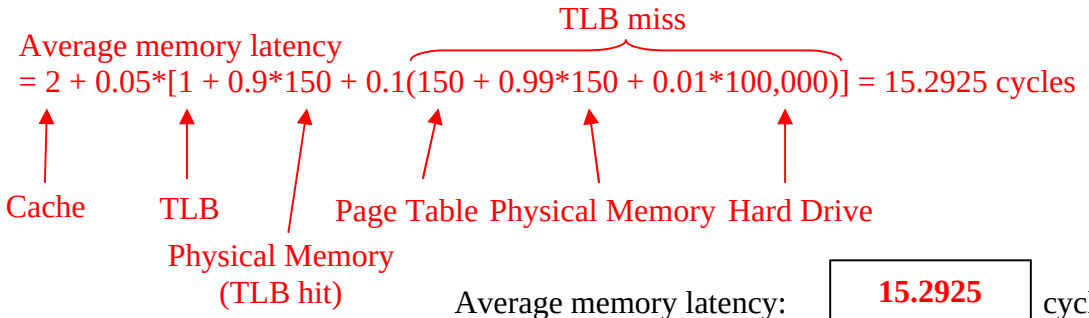
You are given a byte-addressable system with virtual memory, unified data and instruction **virtually addressed** cache and the following characteristics:

- The page table is single-level. TLB and cache accesses are sequential.
- For data swapped out to hard drive, the TLB always misses. Also, TLB hits will **not** cause a page fault. The page table is never swapped out to hard drive.
- Upon retrieval from cache, physical memory or hard drive, the data is sent immediately to the CPU, while the various data structures are updated in parallel.

In addition, you are provided with the following additional information:

Value used			Num.cycles
X	Processor clock frequency	500 MHz	
	TLB size	16 bytes	
X	TLB hit rate	90%	
X	TLB access time	2 ns	1
	TLB associativity	Fully associative	
	TLB replacement policy	Least Recently Used (LRU)	
	Cache size	64 KB	
	Cache block size	8 bytes	
X	Cache hit rate	95%	
X	Cache access time	4 ns	2
	Cache associativity	2 way set associative	
	Cache replacement policy	First In First Out (FIFO)	
X	"Page not loaded in memory" rate	1%	
	Page size	4 KB	
X	Physical memory access time	300 ns	150
	Physical memory size	6 GB	
X	Hard drive access time	200,000 ns	100,000

Compute the average memory latency for this system, specifying your answer in cycles. Write your answer first as a formula with values from the table above and indicate which part relates to (i) TLB; (ii) Page Table; (iii) Physical Memory; (iv) Cache; (v) Hard Disk. **All time values in your formula should be in cycles, not seconds (or nanoseconds).** For any information you use from the table above, check the corresponding first column and, if appropriate, provide the corresponding "Num.cycles" value. Finally, write your final answer in the box below. [No credit will be given for providing only the final answer].



Average memory latency: **15.2925** cycles