CS 250 Practice Final Exam Solutions

Name:

There are 14 questions, with the point values as shown below. You have 180 minutes with a total of 150 points. Pace yourself accordingly.

I recommend that you work this like a real exam under timed conditions and with only your page of notes. Of course, since this is not graded, you may do anything you want with it. On the real final, you will not be allowed an external resources other than 1 page of notes.

#	Question	Points Earned	Points Possible
1	Vocabulary		10
2	C Programming I		8
3	C Programming II		8
4	MIPS Assembly		12
5	Digital Logic		12
6	Finite State Machines		12
7	Datapaths		15
8	Caches I		10
9	Caches II		10
10	Virtual Memory I		11
11	Virtual Memory II		10
12	IO etc.		8
13	Pipelining		8
14	Short Answer		16
	Total		150
	Percent		100

Question 1: Vocabulary [10 pts]

Match each of the following definitions with the appropriate vocab word:

1. Multiple hard disks combined for performance and/or reliability. M. RAID
2. A memory technology which maintains its state as long as the power is on, but loses its contents when power is turned off. R. SRAM
3. An asynchronous notification of an external event, requiring the attention of the OS. H. Interrupt
4. The structure which holds all of the translations from virtual addresses to physical addresses K. Page Table
5. Discarding incorrect instructions from a pipeline F. Flush
6. A piece of logic which selects between two inputs, based on the value of a third input J. Mux
7. The idea that most of a program's data accesses are likely to be contained within a small range of nearby addresses. Q. Spatial Locality
8. A class of ISAs characterized by simple instructions which are easily implemented in high performance hardware. O. RISC
9. A type of datapath in which the CPI is always 1.0 (by definition). P. Single-cycle
10. The part of the branch predictor responsible for prediciting the taken target of branches (except for returns). B. BTB

A	ALU
B	Branch Target Buffer
C	CISC
D	DRAM
E	Exception
F	Flush
G	Hard Disk
H	Interrupt
I	Multi-cycle
J	Mux
K	Page Table
L	Pipeline
M	RAID
N	Return Address Stack
O	RISC
P	Single-cycle
Q	Spatial Locality
R	SRAM
S	Stall
T	Temporal Locality
U	XOR-gate

Question 2: C Programming I [8 pts]

Given the following binary tree node definition:

```
struct _bst_node {
  int key;
  struct _bst_node * left;
  struct _bst_node * right;
};
typedef struct _bst_node bst_node;
```

Write the function `duplicate` which makes a deep copy of the tree and returns it:

```
bst_node * duplicate(bst_node * root) {



 if (root == NULL) {
  return NULL;
 }
 bst_node * ans = malloc(sizeof(*ans));
 ans->key = root->key;
 ans->left = duplicate(root->left);
 ans->right = duplicate(root->right);
 return ans;

}
```

Question 3: C Programming II [8 pts]

A mis-guided C programmer wrote the following code to manipulate a data structure. Unfortunately, this code leaks memory. Correct the code so that it does not leak memory (without changing its functionality):

```
ll_node * update(ll_node * x) {

  ll_node * temp; // <- Changed

  ll_node ** p = &x;

  while (*p != NULL) {

     temp = *p;

     if ((temp->data & 0x1F) == 0) {

         *p = (*p)->next;
         free(temp);   //Added
     }
     else {

          p = &((*p)->next);

     }

  }

  return x;

}
```

Question 4: MIPS Assembly [12 pts]

Translate the `arrayManip` function (written in C below) to MIPS assembly:

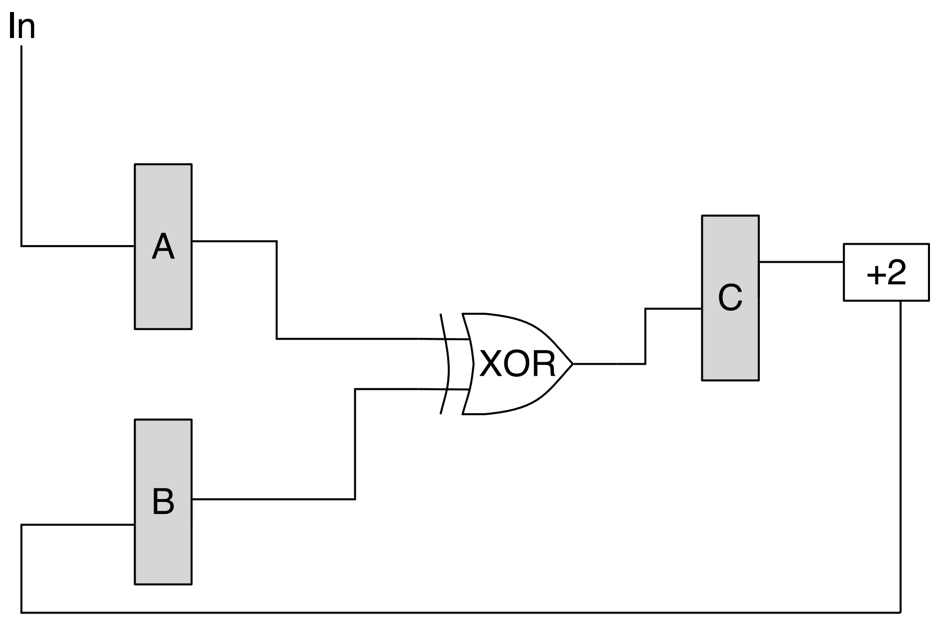
```
int arrayManip(int * p, int n) {
  int sum = 0;
  int * p2 = p;
  int * pend = &p[n];
  while (p2 != pend) {
      int x = *p2;
      int a = f(x);
      if (a > 4) {
        *p2 = a;
      }
      else {
         sum =  x + sum;
      }
      p2++;
  }
  return sum;
}
```

The C code is repeated on the next two page line-by-line. Please do any scratch work you need on this page and answer on the next pages, with your translation of each line of C under that line.

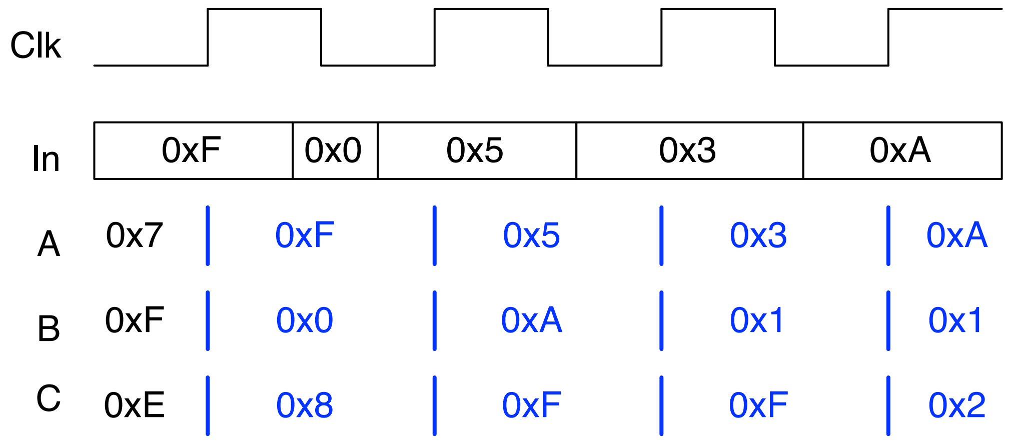
```
# int arrayManip(int * p, int n) {

        addiu $sp, $sp, -32
        sw $fp, 0($sp)
        sw $ra, 4($sp)
        sw $s0, 8($sp)
        sw $s1, 12($sp)
        sw $s2, 16($sp)
        sw $s3, 20($sp)


#  int sum = 0;

        li $s0, 0

#  int * p2 = p;

        move $s1, $a0


#  int * pend = &pn;

        sll $t0, $a1, 2
        add $s2, $a0, $t0


#  while (p2 != pend) {

.L_while:
       beq $s1, $s2, .L_while_end


#      int x = *p2;

        lw $s3, 0($s1)


#      int a = f(x);

        move $a0, $s3
        jal  f
#       if (a > 4) {

         li $t0, 4
         ble $v0, $t0, .L_else
#         *p2 = a;

         sw $v0, 0($s1)

#       }
         b .L_endif

#       else {

.L_else:

#          sum =  x + sum;

         add $s0, $s3, $s0

#       }

.L_endif:

#       p2++;

         addiu $s1, $s1, 4

#   }

         b .L_while
.L_endwhile:
#   return sum;
#}
         move $v0, $s3
         lw $fp, 0($sp)
         lw $ra, 4($sp)
         lw $s0, 8($sp)
         lw $s1, 12($sp)
         lw $s2, 16($sp)
         lw $s3, 20($sp)
         addiu $sp, $sp, 32
         jr $ra
```

Question 5: Digital Logic [12 pts]

The following figure shows a digital logic circuit with 3 4-bit registers (A, B, and C). These registers are comprised of DFFs which are triggered by the rising edge of the clock. The circuit has an external input (In) which is also 4 bits.



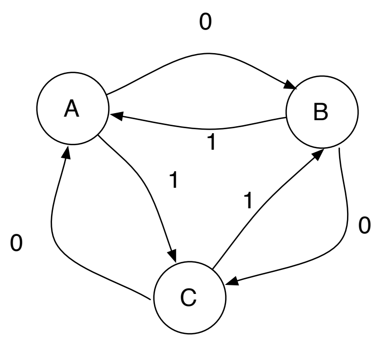
Show the output values of each register over time. Clearly deliinate where the output value changes for each register by drawing a vertical line at the point the value changes.



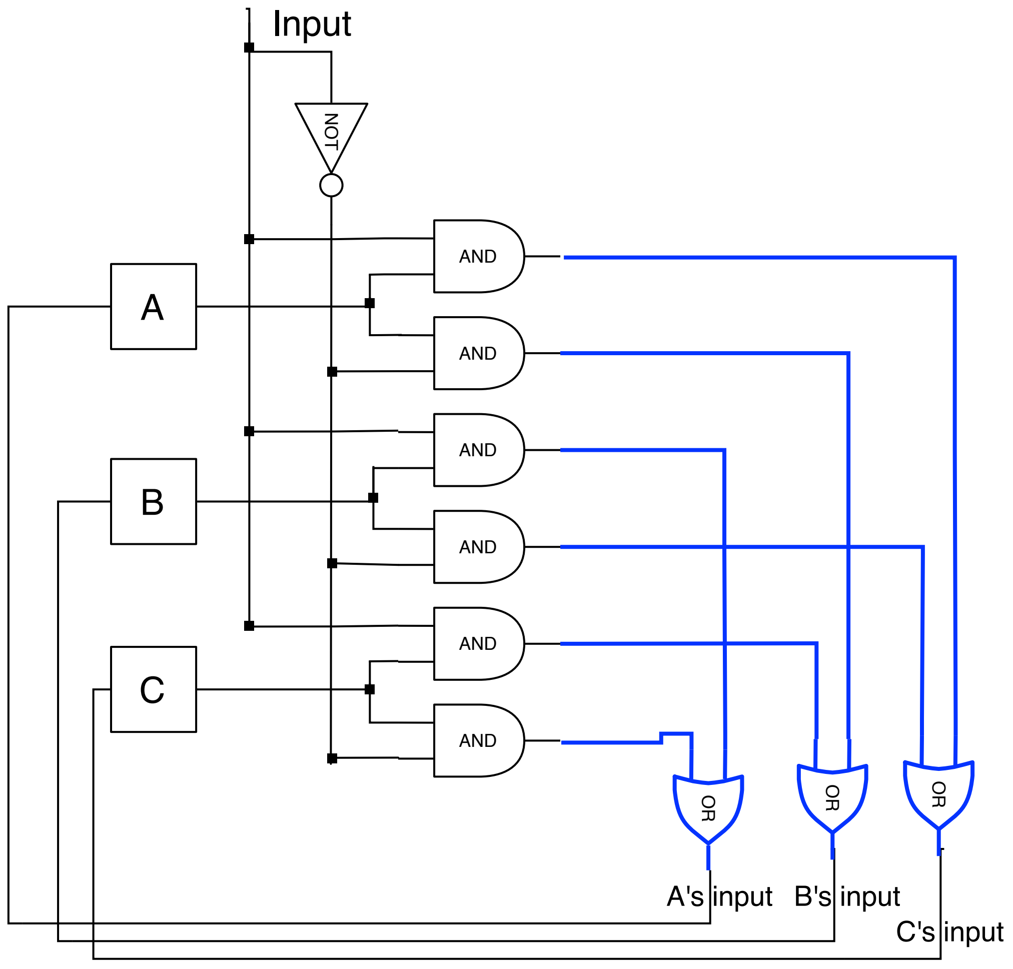
In	0xF	0x0	0x5	0x3	0xA
A	0x7	0xF	0x5	0x3	0xA
B	0xF	0x0	0xA	0x1	0x1
C	0xE	0x8	0xF	0xF	0x2

Question 6: Finite State Machines [12 pts]

Consider the following finite state machine diagram (with three states and one input):



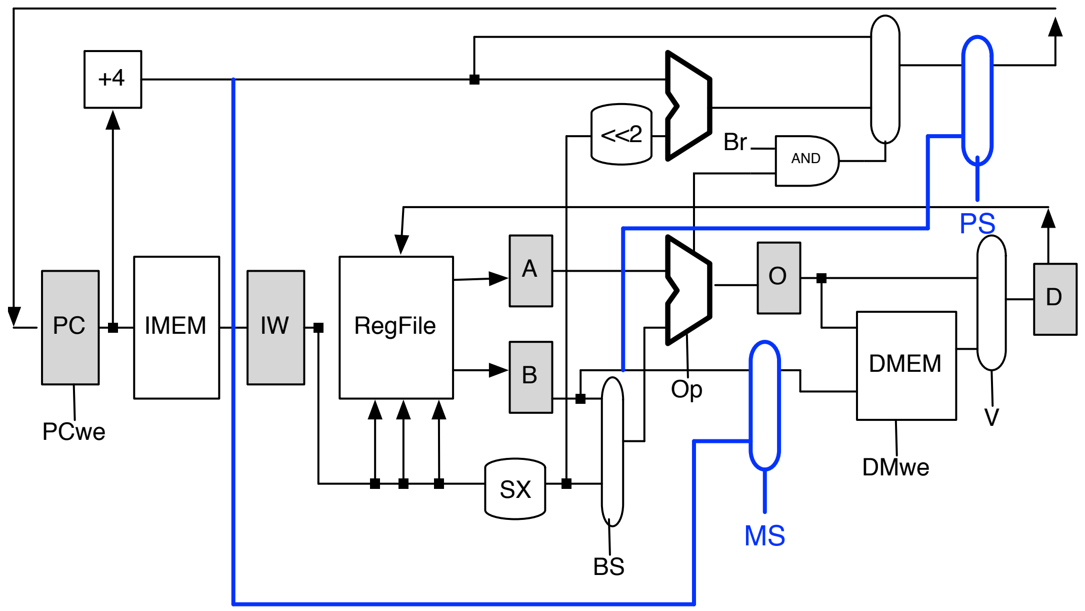
Complete the circuit diagram below which implements the above FSM using a one hot representation and three DFFs (one for each state, labelled accordingly).



Question 7: Datapaths [15 pts]

Modify the basic multi-cycle data-path (shown below) to implement a new instruction, `jrlm $rt, offset($rs)`. This instruction (jump register and link into memory) has the following semantics:

```
Mem[ $rs + offset ]  = PC + 4
PC = $rt
```



After drawing your modifications above, (briefly) explain what happens in the data path on each cycle of the execution of this new instruction (you may not need all the cycles which space has been provided for). The first two are done for you.

1. Instruction is fetched from instruction memory
2. The instruction is decoded and the register file is read.
3. The ALU is used to generate the address (rs + offset)
4. The new MS control signal is set to 1 to pick PC+4 as the data input for data memory. Data memory is write-enabled to perform the store operation. Also, PS is set to 1 to select rt as the next PC, and the PC is write enabled to perform the jump.
5.
6.

Question 8: Caches I [10 pts]

You are desigining the memory hierarchy for a new processor. The access latency of main memory is 200 cycles. You have the following choices for the L2 design:

- 2MB, 16-way set-associative. Thit = 30 cycles. %miss=1%.
- 1MB, 8-way set-associative. Thit = 20 cycles. %miss=5%.
- 512KB, 4-way set-associative. Thit = 15 cycles. %miss=10%.

and the following choices for L1 designs:

- 64KB, 8-way set-associative. Thit = 2 cycles. %miss=4%.
- 32KB, 4-way set-associative. Thit = 1 cycle. %miss=5%.

Which cache designs would you choose and why?

Answer:
The L2 design is independent of the L1 design, so we can/should select it first. For the L2, we can compute Tavg for each design:
2MB: 30 + 0.01 * 200 = 32
1MB: 20 + 0.05 * 200 = 30
512K: 15 + 0.10 * 200 = 35
This means the 1MB (Tavg=30) is the best L2 design. Now we can pick the L1:
64KB: 2 + 0.04 * 30 = 2 + 1.2 = 3.2
32KB: 1 + 0.05 * 30 = 1 + 1.5 = 2.5
This means 32KB (Tavg = 2.5) is the best L1 design.

Question 9: Caches II [10 pts]

A 16-byte cache has 8-byte blocks, has 2 sets, and is 2-way set-associative. The cache initially is empty (all valid bits are off: indicated by a blank box in the table below). The cache receives requests in the sequence listed below. For each address in the sequence (a) split it into the tag, index, and offset; (b) categorize the access as a *hit*, a *compulsory miss*, a *conflict miss*, or a *capacity miss* (You can abbreviate hit=H, Compulsory=O, Conflict=F, Capacity=P); (c) show the new contents of the cache after the access—write the tags for each way, and note which way is LRU. The first one is done for you:

Address	Split Address				Set 0			Set 1		
	Tag	Index	Offset	Result	Way 0	Way 1	LRU Way	Way 0	Way 1	LRU Way
FF	F	1	7	O			0	F		1
F7	F	0	7	O	F		1	F		1
0B	0	1	3	O	F		1	F	0	0
24	2	0	4	O	F	2	0	F	0	0
FB	F	1	3	H	F	2	0	F	0	1
CA	C	1	2	O	F	2	0	F	C	0
08	0	1	0	F	F	2	0	0	C	1
F8	F	1	0	F	F	2	0	0	F	0
35	3	0	5	0	3	2	1	0	F	0
F7	F	0	7	P	3	F	0	0	F	0
32	3	0	2	H	3	F	1	0	F	0

Question 10: Virtual Memory I [11 pts]

Suppose that a system has a 32-bit (4GB) virtual address space. It has 1GB of physical memory, and uses 1MB pages.

1. How many bits are there in the page offset? 20

2. How many virtual pages are there in the address space? 4096

3. How many physical pages are there in the address space? 1024

4. How many bits are there in the virtual page number? 12

5. How many bits are there in the physical page number? 10

6. Some entries of the page table are shown to the right (all values are in hex, and all entries shown are valid). Translate virtual address 0x410423 to a physical address, using the translations in this page table. 0xDD10423

Entry Number	Value
0	1F
1	3C
2	55
3	9C
4	DD
5	EE
6	99
...	...
20	2F
21	4C
22	65
23	AC
24	ED
25	FE
26	100
...	...
40	11F
41	13C
42	155
43	19C
44	1DD
45	1EE
46	199
...	...

Question 11: Virtual Memory II [10 pts]

Suppose that virtual page numbers are 36 bits, and page table entries, as well as pointers to other page tables are 4 bytes each. The program we are interested in has memory allocated on virtual page numbers (not addresses) 0x1000–0x10000, 0x7FFFFF000–0x800001000.

1. How much memory would a single level page table require?

   Answer:
   $2^{36} * 4 = 2^{38}$ bytes, which is 256 GB

   .

2. How much memory would a two level page table require (Assume the bits are divided evenly between the two levels)?

   Answer:
   Each table is $2^{18} * 4$ bytes = 1MB. We need 1 first level plus 3 second level tables, so 4 MB.

3. How much memory would a three level page table require (Assume the bits are divided evenly between the two levels)?

   Answer:
   Each table is $2^{12} * 4$ bytes = 16KB. We need 1 first level plus 3 second level, plus 19 $(16 + 1 + 2)$ third level tables, so 368KB.

   .

4. What performance problem would naive use of a three level page table pose?

   Answer:
   Every load or store would require 4 memory accesses (3 to translate, plus its own).

5. How do we fix that performance problem in real processors?

   Answer:
   Cache the translations.

Question 12: IO etc [8 pts]

For each question, circle the best answer. If none of the selections are appropriate, then choose "e. None of the above"

1. Which of the following is an advantage of interrupts over polling? b

   a. Interrupts are a simpler mechanism for both the hardware and software.

   b. Interrupts allow for more efficient CPU utilization.

   c. Interrupts are compatible with virtual memory, while polling is not.

   d. Polling causes bad hit rates in the L1 instruction cache.

   e. None of the above

2. What three operations (in the correct order) are required to read data from the hard disk? d

   a. jal, sw, jr

   b. Wait for rotation, seek, read data as it spins under

   c. Interrupt, seek, read data as it spins under

   d. Seek, wait for rotation, read data as it spins under

   e. None of the above

3. In order to acheive high performance from a hard disk, software must c

   a. Track the rotational position of the hard disk to estimate rotational delay.

   b. Use virtual memory.

   c. Request large sequential reads/writes.

   d. Directly access the disk without the overhead of the OS.

   e. None of the above

4. Which of the following would **not** cause an exception? b

   a. Division by zero.

   b. A branch is mis-predicted.

   c. Attempting to execute a priveledged instruction in a normal application.

   d. A load or store does not have a valid virtual to physical translation.

   e. None of the above (all cause exceptions).

Question 13: Pipelining [8 pts]

Fill in the blanks below to correctly complete the explaination of some of the difficulties and solutions of pipelining.

One complexity that can arrise with pipeling is _data_ hazzards—in which the producer has not yet written its result to register file before the consumer reads it. The original MIPS ISA fixed this with software interlocks. However, most ISAs avoid these for both _compatability_ and performance reasons. Hardware interlocks fix the first problem, but still suffer poor performance. A better solution is _bypassing_ , which is implemented by adding muxes to the X and M stages to choose between values coming from older in-flight instruction or the register file. Even when all combinations of (word in previous blank) are implemented, there is one situation where the pipeline must stall: a _load_ instruction followed immediately by a dependent use.

Another type of hazzard is _control_ hazzards, which arise from branches. These hazzards are best addressed through speculation—guessing the correct outcome of the branch before it executes. In doing so, the pipeline may guess incorrectly, and need to _flush_ incorrect instructions.

Question 14: Short Answer [10 pts]

1. Explain what a segmentation fault is. In your answer, be sure to address what type of programming errors/program actions cause it, as well as how the hardware detects the situation.

   Answer:
   A segmentation fault occurs when a program attempts to access invalid memory. The canonical example is an attempt to dereference a NULL pointer. The hardware detects this situation by finding no valid translation for the requested address, causing a page fault. The OS then determines that the requested address lies outside of the program's address space and terminates it with a segmentation fault.

2. Compare and contrast caches and virtual memory. Give at least one similarity and one difference between the two. For the difference, explain *why* this difference exists.

   Answer:
   (Many possible)
   Similarity: both split memory into fixed sized chunks (blocks/pages), and manipulate memory at this granularity.
   Difference: In virtual memory, software (the OS) makes a replacement decision, while in caches, the hardware makes the replacement decision. This difference exists because Tmiss is so much larger for memory (which misses to disk) than any other level of the memory hierarchy—this justifies the extra time for software transfer control into a software routine to make a complex decision, in the hopes of reducing %miss.