CS61C FINAL EXAM: 8/12/99

N. Isailovic

Last name ________________________ **First name** ________________________

Student ID number ________________ **Login: cs61c-**______

Please circle the last two letters of your login name.

A	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
A	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Discussion section meeting time ___________ TA's name ________________________

You are allowed to use two 8.5" x 11" double-sided handwritten pages of notes. **No calculators!** This booklet contains 13 numbered pages including the cover page plus photocopied pages from COD and the basic datapath design you are to use. Put all answers on these pages, please; don't hand in stray pieces of paper. The exam contains 7 substantive questions, the Signin Question, the statement immediately below (requiring a signature) and one extra credit question. You have three hours, so relax – this exam isn't worth having a heart failure over.
Good luck!

I certify that my answers to this exam are all my own work. If I am taking this exam early, I certify that I shall not discuss the exam questions or answers with anyone until after the scheduled exam time.

Signature ____

Question Name	Time (minutes)	Max Points	Your points
Signin	0	-1 to 0	
Nested Loops	10	5	
Cache	10	5	
Virtual Memory	10	6	
ALU Design	15	10	
Datapath Design	25	20	
Coding	30	20	
Interrupts	30	14	
Extra Credit	0	(1)	
Total	130	80	

Signin Question (-1 point if not followed): *Fill out the front page correctly and write your login at the top of each of the pages.* **Circle your login initials so we can read them.**

Nested Loops Question (5 points)[10 minutes]:

A matrix is a two-dimensional array. That is, a matrix is an array of elements, each of which is an array. Here is a sample matrix:

```
[a    b     c     d]
[e    f     g     h]
[i    j     k     l]
[m    n     o     p]
```

This matrix is a four-element array, the first element being the array `[a b c d]`, the second element being the array `[e f g h]`, the third element being the array `[i j k l]`, and the fourth element being the array `[m n o p]`.

In the C language, a matrix is accessed as a two-dimensional array. For example, suppose the above matrix/array were declared with the name `mat` in some C code. Then the phrase `mat[1][2]` would return element `g`, since this is the element in row 1 and column 2 of the matrix.

You are given the task of writing a short nested loop which initializes each element of a particular matrix named `TestMatrix`. You know of two possible ways to write this loop. (Assume both `i` and `j` are already declared to be integers.)

Option 1:
```
for (i = 0; i < 1000; i++) {
      for (j = 0; j < 1000; j++) {
            TestMatrix[i][j] = i + j + 1;
      }
}
```

Option 2:
```
for (j = 0; j < 1000; j++) {
      for (i = 0; i < 1000; i++) {
            TestMatrix[i][j] = i + j + 1;
      }
}
```

Which of these two loops would result in a more efficient initialization of TestMatrix? In other words, which loop would execute more quickly at run-time? Justify your answer using any knowledge you feel applies to this.

Cache Question (5 points)[10 minutes]:

Imagine that you have a 16 KB direct-mapped cache with 8 word blocks.

a) How many rows does the cache have?

b) Below is a table with increasing array and stride size. The array and stride size correspond to the cache.c program which you used in lab to determine cache dimensions. Instead of indicating access time, fill in approximate hit rate for each entry in the table below. For example, if a certain array and stride size is going to hit every time, then enter 100 for 100%. If instead it would never hit, then put 0 or 0%.

Reminder on how the lab worked: An array of the given size is created and randomly initialized. The code makes an access to the following array elements: 0, Stride, 2xStride, 3xStride, etc. This continues until the end of the array is reached. Then the code makes the same accesses a second time. The hit rates correspond only to this second set of accesses. Ignore the first set.

Array Size (KB)	Stride Size (Bytes)							
	1	2	4	8	16	32	64	128
1	%	%	%	%	%	%	%	%
2	%	%	%	%	%	%	%	%
4	%	%	%	%	%	%	%	%
8	%	%	%	%	%	%	%	%
16	%	%	%	%	%	%	%	%
32	%	%	%	%	%	%	%	%
64	%	%	%	%	%	%	%	%
128	%	%	%	%	%	%	%	%

Virtual Memory Question (6 points)[10 minutes]:

Suppose we are examining a particular computer that uses two-level page tables. The page size is 16 KB. Each page table entry (for both levels) takes up 8 bytes total. The first-level page table takes up four page frames. Each second-level page table takes up one page frame. If you wish, you may leave the answers as powers of 2.

a. How many entries does the first-level page table contain?

b. How many entries does each second-level page table contain?

c. How many bits are in a Virtual Address used by this system?

ALU Design Question (10 points)[15 minutes]:

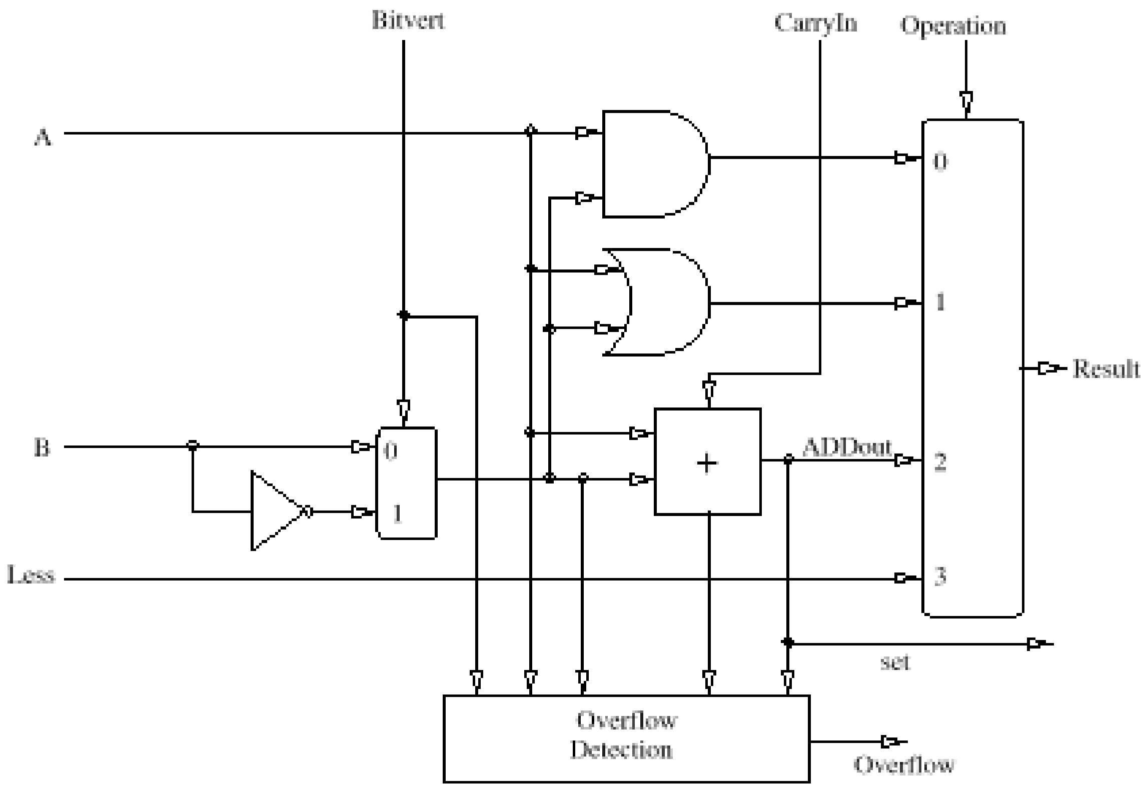
The ALU presented in class supports the set on less than instruction (`slt`) by subtracting one of the two values to be compared from the other and then looking at the sign of the result. Let's try the set on less than operation using the values -7_{ten} and 6_{ten}. To make it simpler to follow the example, let's limit the binary representation to 4 bits: 1001_{two} and 0110_{two}.

$$1001_{two} - 0110_{two} = 1001_{two} + 1010_{two} = 0011_{two}$$

The result suggests that -7 > 6, which is clearly wrong. Hence we must factor in overflow in the decision. Below is a diagram of a portion of the ALU corresponding to the highest order bit of the output. (Note: Not all necessary functions are illustrated in this schematic. Only a few basic ones are included.) There is also a hardware mechanism (don't worry about its implementation) which detects overflow. Show what hardware modifications would be necessary to fully implement `slt` correctly. Your job is basically to fix the `set` output, which in this schematic is the same as the ADD_{out} signal. Explain your modifications thoroughly. Assume that the overflow signal is correct and can be used. Also assume that you have at your disposal any number of AND gates, OR gates and inverters, plus any length of wire.

Notes about the schematic: `Bitvert`, `CarryIn` and `Operation` are control signals. Technically, `Operation` represents two bits, not just one. The two vertical rectangular blocks (pointed to by `Bitvert` and `Operation`) are multiplexors. Remember that your job is the modify the `set` output so that it is 1 if the result should be set or 0 if the result should not be set. At the moment, this works fine if there is no overflow, but fails if there is overflow.

Write out your answer on the next page.



[ALU Design Question continued]
Write out your answer on this page.

Datapath Design Question (20 points)[25 minutes]:

Attached at the end of this exam is a drawing of the datapath designed in class. This datapath does not handle either branch or jump instructions in any way. Given what you know about datapath design, optimizing the pipeline and handling hazards, describe the modifications to the datapath that would need to be made in order to implement the `jal` instruction. You do not need to re-draw the entire datapath. Just be sure to describe in great detail the necessary modifications and make sure to draw the hardware you would use to make these. Please use illustrations to aid in your explanation.

You are allowed to use any number of AND gates (any number of inputs), OR gates (any number of inputs), inverters and muxes (any number of data and control lines). For the muxes, be very clear about what the Control will need to do in order to make your design run.

You **do not** need to design anything that handles hazards caused by this new instruction. However, after completing your design, briefly discuss what hazards do arise as a result of this modification. Also mention what would need to be done in order to solve these hazards (no gates for this part: just a high-level discussion).

Write out your answer on this page and the next one. Just be sure to be clear. If it's undecipherable, it won't be graded.

[Datapath Design Question continued]
Write out the rest of your answer on this page.

Coding Question (20 points)[30 minutes]:

A list is a series of elements that are sequentially connected to each other. A list node is a struct which represents a single element of a list. For the purposes of this problem, a list node will be defined as follows:

```
struct ListNode {
      int         data;  // this is the data contained in this element
      ListNode*   next;  // this is a pointer to the next element in the list
      ListNode*   prev;  // this is a pointer to the previous element in the list
};
```

A list is simply a series of these nodes connected using pointers.

Write a **recursive** MIPS procedure that completely reverses the ordering of the elements in a list.

Your procedure should be named "`Reverse`" and should take one argument: a **pointer** to a `ListNode` object. It should return a pointer to the new first element of the list (which was previously the last element). **You must follow register conventions as well as standard procedure calling**

conventions for full credit on this question (in other words, make no assumptions about the calling procedure).

You may assume that a NULL pointer corresponds to a value of 0. You may also assume that a `ListNode` element is stored in memory exactly as it appears above (that is, as three consecutive words).

Here is a C function which accomplishes the desired task.

```
ListNode* Reverse (ListNode* aNode) {
      ListNode* tempNode;

      if (aNode == NULL) {return;}       // check for incorrect input

      tempNode = aNode->next;            // save the next pointer
      aNode->next = aNode->prev;         // switch the prev and
      aNode->prev = tempNode;            //   next pointers

      if (aNode->prev == NULL) {         // stop if end of list has
            return aNode;                 //  been reached
      } else {
            return Reverse (aNode->prev);  // otherwise continue recursively
      }
}
```

Important:

1. Solutions that are not recursive will not get **any** credit.
2. You must stick to all register conventions and procedure calling conventions.
3. **No pseudoinstructions.** TAL only. This means use only the instructions from the back of the textbook.
4. You must write comments. Code that is not adequately commented will be penalized.

Write your procedure on the next page.

[Coding Question continued]

(Fill in the code for `Reverse` here.)

```
Reverse:
```

Interrupts Question (14 points)[30 minutes]:

The PDS 1000_2 (PDS stands for Poorly Designed System) is a multiprocessor machine (that is, it contains multiple CPUs) which uses the exact same MIPS instruction set that we've been using. Each processor has its own TLB and caches. Each processor is running a different program. The only shared state between the machines is the physical memory. A PDS 1000_2 has 8 processors in it. The page size is 16 KB and a physical address contains 32 bits.

In the normal case, programs are happily running on each processor, loading the pages that they want into memory. What happens, though, when the memory fills up? A program running on one processor, when it needs to bring another page in, must ask for a page to be taken out of memory. The only problem, though, is that the page that was paged out of memory might have been referenced in another processor's TLB. Since the valid entries in each TLB must refer to pages that are in physical memory, the TLB that has a reference to the page that was paged out, must mark that entry as invalid. Usually, this is done in hardware, but on the PDS 1000_2, they didn't have enough time to add that hardware in. They thought of a possible solution: whenever a page is replaced (kicked out of memory) by one processor, an interrupt is thrown to all the other processors. This forces the processors to switch to the interrupt handler. The designers of the PDS 1000_2 set up a data register, and 8 control registers (one for each processor). The data register holds the Physical Address of the page that was kicked out. The control registers are set to 1 initially, and when all of the registers are set to 0, the processor that removed the page can start up again.

You are in charge of writing the interrupt handler. The designers of the PDS 1000_2 give you three functions to help you out.

- FindTLBIndex: This takes a Physical Page Number in $a0 and returns in $v0 the index of the TLB entry that references that PPN or -1 if the PPN was not in the TLB.
- InvalidateTLBEntry: This takes an index into the TLB in $a0 and invalidates that entry.
- TurnOffMyControlRegister: This turns off the control register corresponding to the processor running the interrupt handler.

The address of the data register is 0xffff0050.

Thankfully, you have a copy of the interrupt handler for the PDS 111_2 (a single processor machine). Consequently, you just need to add the code to the handler to deal with the new type of interrupt. You have already added the code that checks the control register and ascertains what type of instruction it is, so you just need to code the necessary commands in the case that it is an interrupt of the new type.

Write the function `TLBCheck` in TAL to accomplish the handling of this new type of interrupt. Remember that you are not writing the main interrupt handler, so you don't have to deal with the instruction `rfe`. You are writing a function to be used by the interrupt handler.

Important:

1. You must stick to all register conventions and procedure calling conventions. (**Note:** Remember that you are writing part of the interrupt handler, so you need to observe the additional restrictions that are imposed on this type of code.)
2. **No pseudoinstructions.** TAL only. This means use only the instructions from the back of the textbook.
3. You must write comments. Code that is not adequately commented will be penalized.

Write your procedure on the next page.

[Interrupts Question continued]

(Fill in the code for `TLBCheck` here.)

```
TLBCheck:
```

Extra Credit Question (1 point)[0 minutes]:

What was Orson Welles' final movie?