CS 308 Final Examination, Dec. 17 & 18, 1998

Name:..

Section (noon or 2pm): ....

Solution Guide (in bold italics)

Pledge:

I have not given or received aid during this examination from anyone, and I will not discuss this examination before 5:00 pm on Friday, Dec. 18, 1998.

Signed:

This is an "open book" exam. You may make use of any written material you bring with you. Your pledge is that you have not copied anyone else's work, and that you will not talk to anyone about the exam after you leave (until after Friday at 5:00pm).

Work out the problems on scrap paper first, and then rewrite your solution **legibly** in this answer book. When the question asks you to justify the result, include that with your solution. Illegible answers count as wrong answers!

1. (6 points). — a warm-up question!
Aliens have landed; besides being green, they have five fingers and a thumb on each hand. So, naturally, they use a base twelve (12_{10}) number system. Their "numerals" are the letters 'A' through 'L' for the digits zero thru' eleven.

(a) Give the earth-equivalent values (base ten) for the following Alien values:

(i)	CAKE	***3580***
(ii)	HEAD	***12675***
(iii)	GEEK	***11002***

(b) Write down the Alien equivalent of the following Erth (base ten) values:

(i)	534	***DIG***
(ii)	1864	***BALE***
(iii)	1539	***KID***

2. (10 points)

A disk drive has its outside track at a radius of 6 cm. This outer track is for calibration, and cannot be used for data. All tracks on the disk contain the same number of sectors (i.e. it is a single-zone disk). The maximum bit-packing density along a track is δ bits/cm., and the distance between tracks is λ cm.

(a) If the innermost track is at radius ρ cm., and the disk has ξ recording surfaces, write down an expression for the capacity of the disk in bits.

Bits/track = $2 \pi \rho \delta$ bits

Tracks/surface = $(6 - \rho) / \lambda$

Total capacity = bits/track * tracks/surface * surfaces/disk

$= 2 \pi \delta \rho (6 - \rho) \xi / \lambda$ (4)

(b) For maximum storage capacity, what will be the radius of the innermost track? Show your reasoning.

differentiate the above wrt ρ, set this to zero:

$dC/d\rho = (2 \pi \delta \xi / \lambda)*(6 - 2\rho) = 0$

so, have an extremum when ρ equals 3 cm. (5)
Differentiating again, we see that $d^2C/d\rho^2$ is negative, hence a maximum in capacity. (1)

3. (10 points)

Write below, **one sentence** for each, what is meant by (a) temporal locality and (b) spatial locality.

(a) ***Items referenced in the recent past tend to be referenced in the near future. (1)***

(b)***Items with addresses close to those recently referenced tend to be referenced in the near future***. (1)

(c) As an illustration of the importance of a program's locality characteristics to virtual memory system performance, we consider the performance of a program which exhibits **no locality whatsoever** (i.e. each memory reference is to a randomly distributed address).

The program, which has a linear virtual address space of 64 MB, is executing on a system with no cache and no TLB. The page size of the sytem is 4 KB. Memory access time is 100 ns., and disk access time is 10 ms.
Suppose that the operating system allocates 1024 pages of main memory for use by this program, plus what is needed for its page tables. Remember that the program generates addresses which are randomly distributed.

What is the average memory reference time when this program executes? Show your work.

Total virtual address space = 64 MB =2^{26} bytes

Page size = 4 KB = 2^{12} bytes

Thus there are 2^{14} pages in lvas.

Thus the fraction of pages which can be in main memory is $2^{10}/2^{14}$ or 1/16 (4)

So, average access time is $^{1}/_{16}$ * 100 ns + $^{15}/_{16}$ * 10 ms

which is 0.0625 * 0.0001 + 0.9375 * 10 millisec

or 9.3750625 ms. (i.e. almost disk speed!) (4)

4. (6 points).

The “SEAS 16-bit floating point standard” is just like the “IEEE floating point standard” except that it uses 1 bit for the sign, 5 bits for the exponent, and 10 bits for the fraction (or mantissa). The exponent is in excess 15 representation.

a) What is the decimal representation of the following SEAS floating point number?

0 01001 0101000000

(a) sign is positive..... (b) exponent, e, is given by 15 + e = 9, or e = -6

so, the value is $1.0101_2 * 2^{-6}$***, or*** $(1 + {}^1/_4 + {}^1/_{16}) / 64 = 2.05078 * 10^{-2}$ ***(3)***

(b) Now go the other way. What is the “SEAS standard 16-bit” representation of the following decimal number? (show your answer in binary, all 16 bits of it)

- 50.75

Well, 50_{10} ***is*** 110010_2 ***. So, in binary, the value is minus*** 110010.11_2

this is $-1.1001011 * 2^{5}$.
we leave off the leading 1, and the biassed exponent value will be 15 + 5 or 21, or 10100

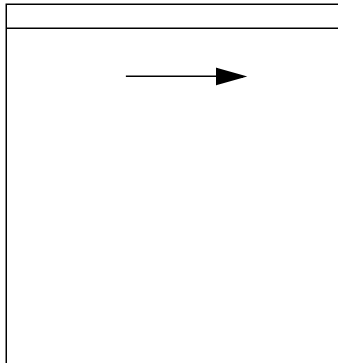
So, the SEAS representation will be: 1 10100 1001011000 (3)

5. (10 points).
Assume that the following section of C++ code is executes on a system with a 1 KB, fully associative, data cache. The cache uses LRU replacement. The compiler stores `float` variables in 4 bytes (1 word). Further, assume that local variables, `i`, `j`, and `sum`, are assigned to registers.

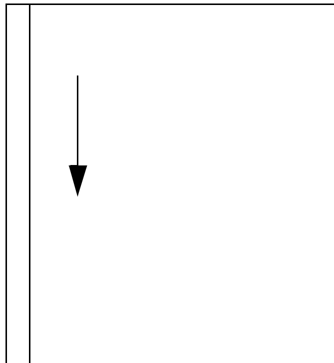
```
float a[1024, 1024], b[1024, 1024];
// assume these arrays are given values somewhere.

void f()
{
     int i, j;
     float sum = 0.0;
     for (i = 0; i < 1024; i++)
          for (j = 0; j < 1024; j++)
               sum = sum + a[i, j] * b[ j,i];
}
```

Remember that, in C++, arrays are stored by row, i.e. storage for `a[0,1]` is in the word next to that holding `a[0,0]`.

What is the cache hit ratio if the cache line (block) size is 16 words (64 bytes)?
Show your reasoning.



array a



array b

line size = 64 bytes or 16 words, 16 lines in the cache. So, each row or column will take up 64 (1024/16) cache lines. (2)

So, as we traverse a given row of array a, we will have 64 cache misses and 1024 - 64 hits. (2)

Every reference to an element of array b will cause a cache miss, since b is stored by rows. (2)

Thus, for one pass thru the outer loop, we will have

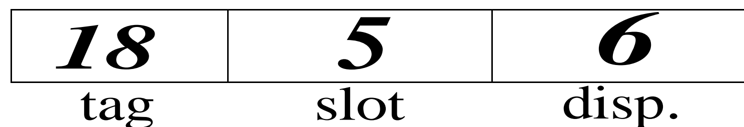
1024 +64 misses and 1024 - 64 hits, giving a hit ratio

of 960/2048 or 0.46875 (4)

6. (8 points).

Consider a machine with a 4-way associative cache with a capacity of 8 KB. Suppose that main memory can have a maximum of 512 MB.

If the cache line (block) size is 64 bytes, indicate how many bits of the address are used for the tag, the cache slot number, and the displacement within the cache line.



Show your reasoning.

Memory address is total of 29 bits. (2)
If line size is 64 bytes, the displacement field takes 6 bits. (2)

Cache capacity is 8 KB, and number of lines will be 8 * 2^{10} / 2^6 or 2^7. But since it is 4-way associative, there are only $^1/_4$ of this number of slots, or 2^5 slots. (3)

Thus slot field takes 5 bits, and the tag is 29 - 5 - 6 or 18 bits. (1)

7. (12 points).

Joe User's program is running on the Simple RISC Computer (SRC) when an interrupt occurs.

(a) What actions automatically take place in the SRC? Explain both in words and in abstract RTN.

IPC <-- PC;   II<15..0> <-- Isrc_info <15..0> ;   iack <-- 1;   IE <-- 0;
PC <-- int.vector<31..0>   (2)

(also need to say it in words)

(b) Following the above actions, the "interrrupt handler" code is executed. Here is an outline of the interrrupt handler code:

```
// begin handler
........
........
rfi
```

Describe in words the actions invoked by the return-from-interrupt (**`rfi`**) instruction.

The program counter, PC, is given the value in the IPC.
The interrupt enable flag (IE) is set to 1.   (2)

(c) One might want this handler to acknowledge and process additional interrupts which have occurred since the original interrupt to Joe User's program. Here is example code:

```
// begin handler
// save registers I'll use here
........
........
een
........
........
restore registers I've used here
rfi
```

This code will not work properly. Explain why.

Because if another interrupt is fielded after the een instruction above opens the gate, the current program counter will be put into the IPC, so that when the handler above finally gets to the rfi instruction it cannot get back to Joe's program, since his PC value has been lost. (4)

(d) Write a version of the handler above which will work properly.

Before the een, have the command:

svi(RX, RY) which saves the IPC and II into registers X and Y

then , after the een, (i.e. after the new interrrupt handler returns to here) do:

edi (disable interrupts)
ri(RX, RY) to restore IPC and II to their old value (4)

8. (20 points).

Your company is designing the new MIPS (MIdget Processor with Stack) machine. The MIPS is an 8-bit zero-address machine, and has been formally specified by the abstract RTN below. Unlike the SRC machine, which contains all operands in a single instruction word, the MIPS stores operands adjacent to their instructions in memory. For all instructions that require an operand (i.e. push, pop, and branches), the operand is located in memory at the next byte after the instruction.

```
PC<7..0> : IR<7..0> : SP<7..0> : Z :
M[0..2^8..-1]<7..0> :

(PC ← 0 ; start)
start := fetch :
fetch := (IR ← M[PC] : PC ← PC + 1 ; execute)
execute := (instruction ; fetch)
instruction := (
        push    (:= IR=0) → (SP ← SP + 1 ; M[SP] ← M[M[PC]] : PC ← PC + 1) :
        pushi   (:= IR=1) → (SP ← SP + 1 ; M[SP] ← M[PC] : PC ← PC + 1) :
        pop     (:= IR=2) → (M[M[PC]] ← M[SP] : SP ← SP - 1 : PC ← PC + 1) :
        add     (:= IR=3) → (M[SP-1] ← M[SP-1] + M[SP] ; SP ← SP - 1 ; setcond) :
        sub     (:= IR=4) → (M[SP-1] ← M[SP-1] - M[SP] ; SP ← SP - 1 ; setcond) :
        br      (:= IR=5) → (PC ← M[PC]) :
        brz     (:= IR=6) → ( (Z=1) → PC ← M[PC] : (Z=0) → PC ← PC + 1) :
        brnz    (:= IR=7) → ( (Z=0) → PC ← M[PC] : (Z=1) → PC ← PC + 1)
)
setcond := ( (M[SP] = 0) → Z ← 1 : (M[SP] ≠ 0) → Z ← 0 )
```

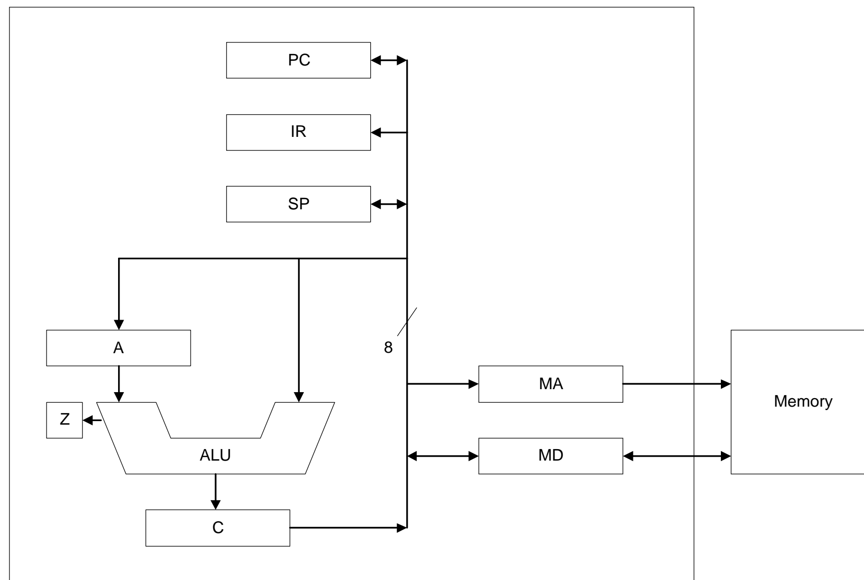
(a) Assume that the variable X is stored at memory location 200. Translate the following MIPS program fragment into machine code - i.e. show the sequence of MIPS machine code bytes, giving byte values in decimal. Note, the machine code should include operand bytes when appropriate. Thus, for example, the instruction “branch to location 44” would bewritten as “ `br, 44` ”.

```
push X
pushi 100
add
pop X
pushi 32
push X
sub
brz 128
```

This was a dumb question, which I copied from Grimshaw’s exam.

push, 200, pushi, 100, add, pop, 200, etc., etc. ***(3)***

(b) A possible one-bus data path design for the MIPS is depicted below:



The control signals for this data path are:

PC_{out} - gate PC onto the bus
PC_{in} - read into PC from bus
IR_{in} - read into IR from bus
SP_{out} - gate SP onto the bus
SP_{in} - read into SP from bus
A_{in} - read into A from bus
C_{out} - gate C onto the bus
C_{in} - read into C from bus
MA_{in} - read into MA from bus
Read - read from memory

MD_{out} - gate MD onto the bus
MD_{in} - read into MD from bus
MD_{wr} - send MD to memory
MD_{rd} - read memory value into MD
INC - ALU increments bus input
DEC - ALU decrements bus input
ADD - ALU adds inputs, sets Z
SUB - ALU subtracts A from bus input, sets Z
C=B - ALU passes through bus input unchanged
Write - write to memory

Using the above data path design and the abstract RTN specification of the MIPS ISA given in part (a), write the concrete RTN and control sequences for the following (note, in each case you may not need all of the rows in the table):

(b-1) The fetch cycle for the MIPS machine:

Time Step	Concrete RTN	Control sequence
T0	$MA \leftarrow PC : C \leftarrow PC + 1$	$MA_{in}, PC_{out}, INC, C_{in}$
T1	$PC \leftarrow C : MD \leftarrow M[MA]$	$PC_{in}, C_{out}, MD_{rd}, Read$
T2	$IR \leftarrow MD$	IR_{in}, MD_{out}

(3)

(b-2) The pop instruction:

Time Step	Concrete RTN	Control sequence
T3	***MA :=SP***	MA_{in}, SP_{out}
T4	***C := MD: READ***	***Cin, MDout, READ***
T5	***MA := PC***	MA_{in}, PC_{out}
T6	***MD := C***	***Cout, MDin, write***
T7	***C := SP - 1***	***SPout, DEC, Cin***
T8	***SP := C***	***Cout, SPin***
T9	***C := PC + 1***	***PCout, INC, Cin***
T10	***PC := C***	C_{out}, PC_{in} ***END***

(5)

(b-3) The add instruction:

Time Step	Concrete RTN	Control sequence
T3	$MA \leftarrow SP : C \leftarrow SP - 1$	$MA_{in}, SP_{out}, C_{in}, DEC$
T4	$MD \leftarrow M[MA] : SP \leftarrow C$	$MD_{rd}, Read, SP_{in}, C_{out}$
T5	$A \leftarrow MD$	A_{in}, MD_{out}
T6	$MA \leftarrow SP$	MA_{in}, SP_{out}
T7	$MD \leftarrow M[MA]$	$MD_{rd}, Read$
T8	$C \leftarrow A + MD$	C_{in}, MD_{out}, ADD
T9	$MD \leftarrow C$	MD_{in}, C_{out}
T10	$M[MA] \leftarrow MD$	$MD_{wr}, Write, End$

(5)

(b-4) The brz instruction:

Time Step	Concrete RTN	Control sequence
T3	$MA \leftarrow PC : C \leftarrow PC + 1$	$MA_{in}, C_{in}, INC, PC_{out}$
T4	$MD \leftarrow M[MA] : PC \leftarrow C$	$MD_{rd}, Read, PC_{in}, C_{out}$
T5	$(Z=1) \rightarrow PC \leftarrow MD$	$Z=1 \rightarrow (PC_{in}, MD_{out}), End$

(4)

9. (10 points).

You have been hired as the memory systems architect for Mega Computer Corporation, which is designing a new computer system which has a virtual address space of 2^{64} bytes. The maximum amount of actual memory for this machine is to be 256 MB. The page size is 4 KB.

(a) If the virtual-to-real mapping of addresses is to be implemented using the conventional page table method, how much memory would the page table take? A page table entry is 4 bytes.

64 bits, of which 12 bits are for displacement within page, i.e. 52 bits for vpn.

but each page table entry takes 4 bytes, so page table is 2^{54} bytes

(b) Describe a more practical way of implementing the mapping between virtual and real addresses for this machine. Discuss the more important factors in getting acceptable performance from your implementation.

Inverted page table; only has one entry per physical frame of main memory.

Entry shows vpn, and process id, of the page actually resident in that frame.

Take incoming vpn, hash function it, which takes you to the beginning of a chain in the virtual page table - see if you find the vpn there. If not, not in memory, generate fault, go conventional, page table all on disk.

Reduce hash collisions by hashing into a "hash anchor table" with bigger range than no. of entriesin the virtual page table.

BUT, important to have a TLB, which will hold most-recently referenced vpn's together with theiractual frame location in MM. Note don't need separate table per process (tho' still need a regularpage table per process)

10. (10 points).

This is a variation on the SRC computer question on the mid-term exam.
(i) the first 128 bytes of memory (`M[0]` through `M[127]`) contain the 32-bit values 0, 1, 2,.... 31 (decimal). i.e. a 32-bit load from location 124_{10} gives you the value $0000001F_{16}$.

(ii) register 10_{10} contains the value 8_{10} and register 20_{10} contains the value 12_{10} .

(iii) the program counter, PC, is set to the value 256_{10}.

__In turn__, the following instructions are each loaded into `M[256`$_{10}$`..259`$_{10}$`]` and executed. For each,

(a) write down the assembly language version of the instruction
(b) write down what is changed (e.g. register 4), and
(c) its new value as an **__8-digit hexadecimal value__.**

Do your calculations on scrap paper - **__write only your answers__** here.

Example: `1D000008` (this is `st 20, 8`) changes the value of `M[8..11]` to `0000000C`$_{16}$

(i) `0B940008` ***(3)***

(a) ***ld r14, 8(10)***

(b) ***register 14 and PC***

(c) ***r14 = 00000004, PC = 00000104*** ***(deduct total of 1 if forget (PC on (i) and (ii))***

(ii) `71954000` ***(3)***

(a) ***sub r6, r10, r20***

(b) ***register 6 and PC***

(c) ***r6 = FFFFFFFC, PC = 00000104***

(iii) `4FE8A003` ***(4)***

(a) ***brlnz r31, r20, r10***

(b) ***register 31 and PC***

(c) ***r31 = 00000104, PC = 0000000C***

11. (10 points).

A Motorola 68000 computer's memory is loaded with the following bytes, starting at location 1000_{16}.

```
DA
87
51
CB
FF
FC
XX
```

Execution begins with the following initial conditions:

$PC = 1000_{16}$
D3 = 3
D5 = 2
D7 = 1

The processor will eventually attempt to execute the instruction beginning "XX".
At this time, what will be the values of:

PC:	***1006_{16}***	***00001006*** ***(2)***	***(1008 if interpret "execute" literally - OK)***
D3:	***-1***	***FFFFFFFF*** ***(1)***	
D5:	***6***	***00000006*** ***(2)***	
D7:	***1***	***00000001*** ***(1)***	

Notes:

(i) you will get partial credit by writing down the assembly language version of the code above. Write it here:

```
LOOP    ADD.L    D7, D5
        DBF      D3, LOOP      (4)
```

(ii) there is an error in the pseudocode for DBcc in table 3.6 on page 114 of the textbook. The notes at the bottom of that page, and the text in the book, correctly describe the action of this instruction.