CS 61C (Clancy)
April 19, 2002

Exam 3

Read and fill in this page now.
Do NOT turn the page until you are told to do so.

Your name: ______

Your login name: ______

Your lab section day and time: ______

Your lab t.a.: ______

Name of the person sitting to your left: ______

Name of the person sitting to your right: ______

Problem 0 ____

Problem 1 ____

Problem 2 ____ ____

Problem 3 ____

Total: ____ /20

Problem 4 ____ ____ ____

This is an open-book test. You have approximately fifty minutes to complete it. You may consult any books, notes, or other paper-based inanimate objects available to you. Use of calculators is not allowed.

To avoid confusion, read the problems carefully. If you find it hard to understand a problem, ask us to explain it. If you have a question during the test, please come to the front or the side of the room to ask it.

This exam comprises 10% of the points on which your final grade will be based. Partial credit may be given for wrong answers. Your exam should contain five problems (numbered 0 through 4) on eight pages. Please write your answers in the spaces provided in the test; in particular, we will not grade anything on the back of an exam page unless we are clearly told on the front of the page to look there.

A few students are taking the exam Monday morning. Please don't post any newsgroup items about the exam until then.

Relax—this exam is not worth having heart failure about.

C

Problem 0 (1 point, 1 minute)

Put your login name on each page. Also make sure you have provided the information requested on the first page.

Problem 1 (2 points, 5 minutes)

Consider the Boolean function represented by the following truth table.

input A	*input* B	*output* Q
0	0	1
0	1	0
1	0	1
1	1	1

Provide a Boolean expression that relates the output Q to the input values A and B. Your expression should be immediately translatable to a circuit with at most three gates. The gates of the circuit should include only inverters and two-input AND and OR gates.

Problem 2 (6 points, 12 minutes)

At the end of this exam is the code for handling output (excerpted from our solution to project 3).

Part a

Suppose the project solution program is modified to print the string "Hello\n" prior to accepting any input, and then is run on the nova computer. This results in print being called with a six-character string as argument. What are the contents of nextIn and nextOut when print returns (i.e. at the label alldone in the code)? Briefly explain your answer.

nextIn's value =

nextOut's value =

Part b

Now consider the unmodified project solution program, again run on nova. Its first call to print uses the argument

```
"(Input'x') Running Total =>  0x00000000 \X0D\n"
```

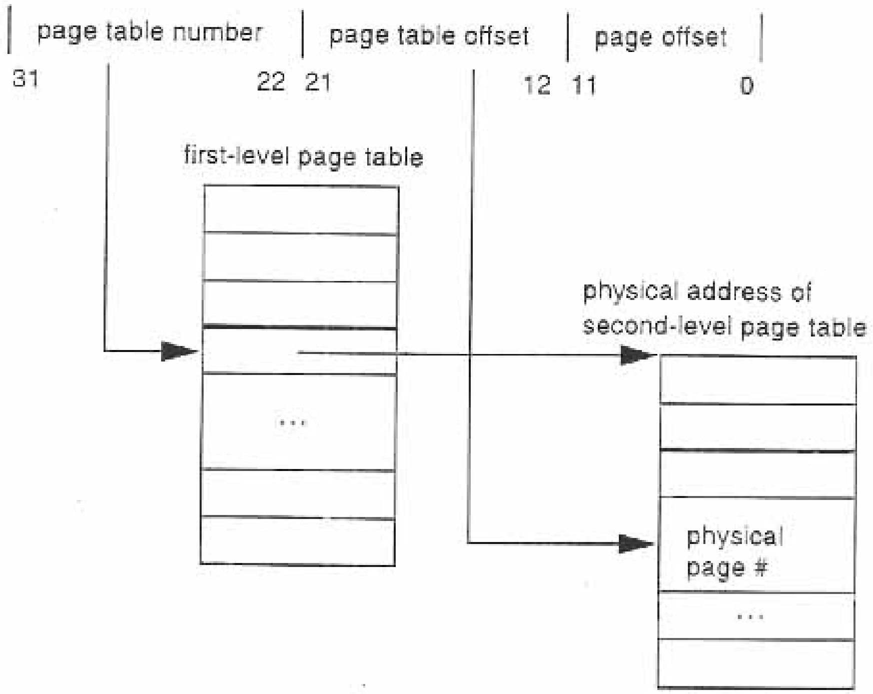
i.e. a 42-character string. What are the contents of nextIn and nextOut when print returns from this call? Briefly explain your answer.

nextIn's value =

nextOut's value =

Problem 3 (3 points, 8 minutes)

One of the peer instruction questions involved a two-level page table, represented in the diagram below.



For the peer instruction question, we assumed that the length of a virtual address was 32 bits. List three different ways of accommodating a *33-bit* virtual address in this address translation system while retaining the two-level table structure and 32-bit physical address length. For each method you list, describe its specific effect on page sizes and the page tables.

Problem 4 (8 points, 24 minutes)

Consider a 16-word (not counting tags) 2-way associative cache with a block size of 4 words using LRU replacement.

Part a

4 words = 16 bytes.
= 2^4 bytes

Indicate in the figure below which bits of a 32-bit address form the tag, the cache index, and the byte offset.

bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
										tag						

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Part b

Suppose that the contents of memory between byte addresses 52 and 83 are as shown below.

byte address	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67
contents	8				6				3				1			

byte address	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
contents	4				5				9				2			

In the diagram below, fill in the result of loading the word at address 68, using a cache that's initially empty.

set #	*data (contents)*			
0	1	4	5	9
~~1~~				

Part c

The cache.c program from lab assignment 10 (listed at the end of this exam), run on a computer with the cache just described (and no secondary cache), produces a "read+write" time of 200ns in the situation where the number of cache hits is maximized and a time of 800ns when the number of cache hits is minimized.

By filling in the bottom row of the table below, indicate what times this run of cache.c might produce for a 32-word array with strides ranging from 1 to 16 words. Each value will be one of the following: 200ns, 350ns, 500ns, 800ns.

	stride in words	1	2	4	8	16
	stride in bytes	4	8	16	32	64
size in words	bytes					
8	32	200	200	200		
16	64	200	200	200	200	
32	128					

Code from the project 3 solution

```
indone:                          # Next attend to output.
        lw $t0,nextOut           # Is buffer empty? I.e. is nextOut equal to nextIn?
        lw $t1,nextIn
        bne $t0,$t1,notempty
        lui $t3,0xffff           # If so, disable interrupts in the transmitter.
        sw $0,8($t3)
        j intDone
notempty:
        lui $t1,0xffff           # Get base address of device registers.
        lw $t2,8($t1)            # Get status word for output.
        andi $t2,$t2,0x1         # Mask out all but ready bit.
        beq $t2,$0,intDone       # Return if not ready.
        la $t2,buffer            # Get buffer base address.
        addu $t3,$t2,$t0         # Add offset.
        lb $t2,0($t3)            # Get the character from the buffer.
        sb $t2,12($t1)           # Output character to terminal.
        addiu $t1,$t0,-1         # Decrement index.
        andi $t1,$t1,31          # Wrap around from -1 back to 31.
        sw $t1,nextOut
intDone:
        ...

buffer:   .space 32              # Characters waiting to be output.
nextIn:   .word 0                # Index of next place to insert character into buffer.
nextOut:  .word 0                # Index of next character to remove from buffer

print:
        lb $t3,0($a0)            # Fetch next character to store in buffer.
        addiu $a0,$a0,1          # Check for end of string.
        beq $t3,$0,alldone
        lw $t0,nextIn            # Fetch current input index.
        addiu $t1,$t0,-1         # Compute new input index after we store the next character.
        andi $t1,$t1,31          # Wrap around from -1 back to 31.
chkfull:
        lw $t2,nextOut           # Is buffer full? I.e. is nextIn just before nextOut?
        beq $t2,$t1,chkfull      # If so, just keep checking until things get better.
        la $t4,buffer
        add $t4,$t4,$t0
        sb $t3,0($t4)            # There is space in the buffer; store character.
        sw $t1,nextIn            # Update "nextIn" index.
        lui $t3,0xffff           # Make sure interrupts are enabled in the transmitter.
        addiu $t2,$0,2
        sw $t2,8($t3)
        j print                  # Go back for more characters.
alldone:
        jr $31                   # Return to caller.
```

cache.c

```
#define CACHE_MIN (1024)                              /* smallest cache (in words) */
#define CACHE_MAX (256*1024)                                     /* largest cache */
#define STRIDE_MIN 1                                 /* smallest stride (in words) */
#define STRIDE_MAX 128                                          /* largest stride */
#define SAMPLE 10                                   /* to get a larger time sample */
#define CLK_TCK 60                                 /* number clock ticks per second */
int x[CACHE_MAX];                                  /* array going to stride through */

double get_seconds () {                                     /* routine to read time */
   struct tms rusage;
   times (&rusage);                           /* UNIX utility: time in clock ticks */
   return (double) (rusage.tms_utime) / CLK_TCK;
}

int main () {
   int register i, index, stride, limit, temp;
   int steps, tsteps, csize;
   double sec0, sec;                                          /* timing variables */

   for (csize = CACHE_MIN; csize <= CACHE_MAX; csize = csize * 2)
      for (stride = STRIDE_MIN; stride <= STRIDE_MAX; stride = stride * 2) {
         sec = 0;                                               /* initialize timer */
         limit = csize - stride + 1;                          /* cache size this loop */

         steps = 0;
         do {                                          /* repeat until collect 1 second */
            sec0 = get_seconds ();                                     /* start timer */
            for (i = SAMPLE * stride; i != 0; i = i - 1)            /* larger sample */
               for (index = 0; index < limit; index = index + stride)
                  x[index] = x[index] + 1;                            /* cache access */
            steps = steps + 1;                           /* count while loop iterations */
            sec = sec + (get_seconds () - sec0);                          /* end timer */
         }
         while (sec < 1.0);                                   /* until collect 1 second */

         /* Repeat empty loop to loop subtract overhead */
         tsteps = 0;                          /* used to match number of while iterations */
         do {                          /* repeat until same number of iterations as above */
            sec0 = get_seconds ();                                        /* start timer */
            for (i = SAMPLE * stride; i != 0; i = i - 1)               /* larger sample */
               for (index = 0; index < limit; index = index + stride)
                  temp = temp + index;                                     /* dummy code */
            tsteps = tsteps + 1;                                /* count while iterations */
            sec = sec - (get_seconds () - sec0);                             /* –overhead */
         }
         while (tsteps < steps);                /* until equal to number of iterations */

         if( stride==STRIDE_MIN ) printf("\n");/* extra line to separate array sizes */
         printf("Size(bytes): %7d Stride(bytes): %4d read+write: %4.0f ns\n",
             csize * sizeof (int), stride * sizeof (int),
             (double) sec*1e9 / (steps*SAMPLE*stride*((limit-1)/stride + 1)));
      };                                                 /* end of both outer for loops */
}
```