University of California at Berkeley
College of Engineering
Computer Science Division - EECS

CS 152 D. Patterson S. Kong
Spring 1995

Computer Architecture and Engineering
Midterm I

Your Name: _____
SID Number: _____
Discussion Section: _____

You may bring two pages of notes. You have 180 minutes. Each question carries 20 points.
Show your work. Write neatly and be well organized.
Good Luck!

Problem	Score
1	
2	
3	
4	
5	
Total	

1. You work at Hariprasad Industries. You are designing a computer and testing it on your favorite program: a video game called Mr. Oza's Nut Factory.

The clock rate for your machine is 100MHz. Your machine has a floating-point unit. The video game has 500 million total instructions.

Here are the measurements for your video game:

Instruction	CPI	Frequency
A	2	35%
B	5	30%
C	4	20%
D	4	15%

a) What is the average CPI for your machine when running this program? **[4 points]**

b) A friend of yours gives you a new compiler to try out. The instruction count improvements resulting from this compiler are as follows. **[4 points]**

Instruction Class	Percentage of Instructions Executed vs. Original Machine
A	80%
B	100%
C	95%
D	80%

What is the average CPI for your machine when running your program as compiled with this new compiler?

Extra Credit: What is your opinion of this friend?

c) What is the execution time of the program without the new compiler? **[4 points]**

d) What is the execution time of the program with the new compiler? **[4 points]**

Extra Credit: What is your opinion of this friend now?

The compiler was recalled because the compiler writer used useless toy benchmarks to exaggerate the improvement resulting from his compiler. Therefore, use the measurements WITHOUT the new compiler from now on.

e) Your boss gets mad at you because you used a metric other than execution time to measure performance (How shameful!). Your boss takes your floating-point unit away from you. As a result, floating-point instructions take four times longer now. Assume that the class B instructions are the floating-point instructions. How much faster is the original machine (with the floating point unit and uses the old compiler) than this crippled machine (without the floating point unit and uses the old compiler)? **[4 points]**

2. The cost/performance of two microprocessors is to be examined, each running the same instruction set.

The first option is a Galium Arsenide (GaAs) microprocessor. A GaAs wafer that is 10cm (≈ 4 inches) in diameter costs $2000. The manufacturing process creates 4 defects per square centimeter. The micorprocessor fabricated in this technology is expected to have a clock cycle rate of 1000MHz, with an average clock per instruction of 2.0 if we assume an infinitely fast memory system. The size of the GaAs microprocessor is 1.0cm by 1.0cm. (Assume α is the same for GaAs as CMOS = 2.0.)

The second option is a CMOS microprocessor. A 15cm (≈ 6 inch) wafer with 1 defect per square centimeter costs $1000. The 1.0cm by 2.0cm microprocessor executes multiple instructions per clock cycle to achieve an average clock cycles per instruction of 1.0, assuming an infinitely fast memory while achieving a clock rate of 200MHz. (The microprocessor is larger because of on chip caches and executing multiple instructions per clock cycle.)

Neither wafer has test dies on them.

Here are two equations you may find useful:

$$Dies\ Per\ Wafer = \frac{\pi \times (\frac{Wafer\ Diameter}{2})^2}{Die\ Area} - \frac{\pi \times Wafer\ Diameter}{\sqrt{2 \times Die\ Area}} - Test\ Dies\ Per\ Wafer$$

$$DieYield = (1 + \frac{DefectsPerUnitArea \times DieArea}{\alpha})^{-\alpha}$$

a) What is the cost of an untested GaAs die for this microprocessor? Show your work. **[4 points]**

b) What is the cost of an untested die for the CMOS microprocessor? Show your work. **[4 points]**

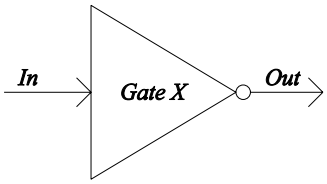
c) What is the ratio of cost of the GaAs die to the CMOS die? **[1 points]**

d) Calculate the average time for each instruction with an infinitely fast memory. Which is faster and by what factor? Show your work. **[3 points]**

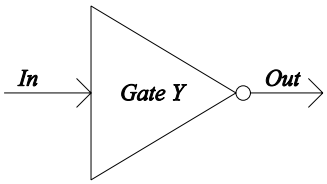
e) Based on costs and performance ratios of the microprocessors calculated above, what is the ratio of cost/performance of the CMOS to GaAs microprocessors? **[3 points]**

f) What happens to these ratios when you attach a memory system to these microprocessors? The memory system is the same for both. Assume the average memory access time per instruction is 3 ns and the cost is $200. Explain the changes to the ratios. **[5 points]**

3. Assume we have two magic inverters X and Y:

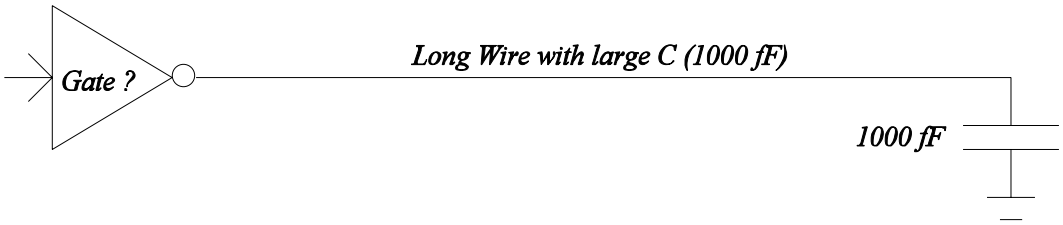


Input Load = 50 fF
Internal Delay (Output going from Low to High) = TPlh = 0.3ns
Internal Delay (Output going from High to Low) = TPhl = 0.3ns
Load Dependent Delay (Output going from L to H) = TPlhf = 0.010ns/fF
Load Dependent Delay (Output going from H to L) = TPhlf = 0.002ns/fF

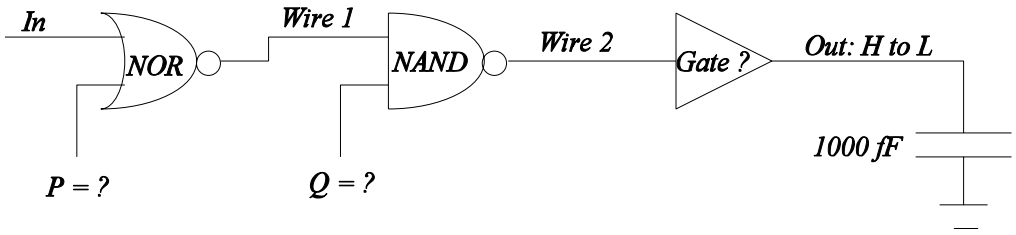


Input Load = 50 fF
Internal Delay (Output going from Low to High) = TPlh = 0.3ns
Internal Delay (Output going from High to Low) = TPhl = 0.3ns
Load Dependent Delay (Output going from L to H) = TPlhf = 0.002ns/fF
Load Dependent Delay (Output going from H to L) = TPhlf = 0.010ns/fF

a) Assume we need to use one to these drivers to drive a long wire with high capacitive load (see diagram below) and our goal is to have a VERY FAST High to Low transition at the output while the Low and High transigion can be slow. Which gate should we use and why? **[3 points]**



b) Now assume the gate you chose from above is driven by the following logic:



i. In order to cause Out to make a High to Low transition, what kind of transition does Wire 2 have to make (H to L or L to H)? **[2 points]**

ii. In order to propagate a signal from Wire 1 to Wire 2, what value do we need to have at Q (0 or 1)? **[2 points]**

iii. In order to cause Out to make a High to Low transition, what kind of transition does Wire 1 have to make (H to L or L to H)? **[2 points]**

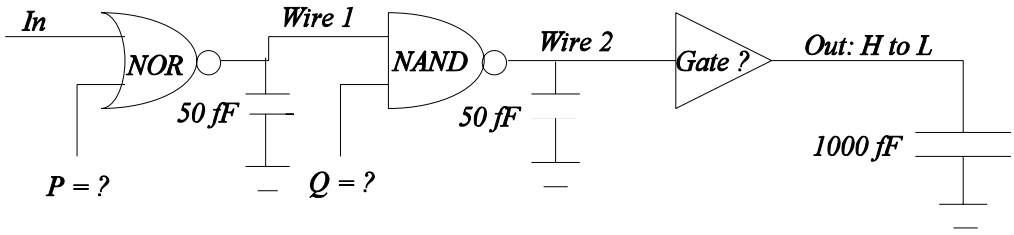
iv. In order to propagate a signal from In to Wire 1, what value do we need to have at P (0 or 1)? **[2 points]**

v. In order to cause Out to make a High to Low transition, what kind of transition do we have to apply at In (H to L or L to H)? **[2 points]**

c) After filling out the answers from Part a &b into the diagram below, calculate the delay from In to Out assuming:

We want Out to make a High to Low transition.
Wire 1 and Wire 2 each has 50 fF capacitance.

(Hint: calcuate the delay from In to Wire 1, Wire 1 to Wire 2, and from Wire 2 to Out, and then add them together) **[7 points]**



NOR gate characteristics:

Input Load = 50 fF
Internal Delay (Output going from L to H) = TPlh = 0.2ns
Internal Delay (Output going from H to L) = TPhl = 0.2ns
Load Dependent Delay (Output going from L to H) = TPlhf = 0.004ns/fF
Load Dependent Delay (Output going from H to L) = TPhlf = 0.002ns/fF

NAND gate characteristics:

Input Load = 50 fF
Internal Delay (Output going from L to H) = TPlh = 0.2ns
Internal Delay (Output going from H to L) = TPhl = 0.2ns
Load Dependent Delay (Output going from L to H) = TPlhf = 0.003ns/fF
Load Dependent Delay (Output going from H to L) = TPhlf = 0.006ns/fF

4) You are given a MIPS assembly program that implements Bubble Sort. However, some lines are missing. Your job is to fill in the blanks to make the program works. Besides filling in the missing instructions, you should also fill in the corresponding comments. To remind you of how Bubble Sort works, you are also given a Bubble Sort program written in C. **[20 points]**

C Program

```
#include <stdio.h>
#include <stdlib.h>

extern void BubbleSort(int*, int);
extern void PrintResults(int*, int);

void
main()
{
  int test1[10] = {5 ,4, 3, 2, 1, 6};
  int test2[10] = {1, 3, 5, 3, 6, 5};
  int test3[10] = {5, 4, -4, 13, -1, 3, 2, 1, 6};
  int test4[10] = {5};
  int test5[10];

  BubbleSort(test1, 6);
  PrintResults(test1, 6);

  BubbleSort(test2, 6);
  PrintResults(test2, 6);

  BubbleSort(test3, 9);
  PrintResults(test3, 9);

  BubbleSort(test4, 1);
  PrintResults(test4, 1);

  BubbleSort(test5, 0);
  PrintResults(test5, 0);
}
```

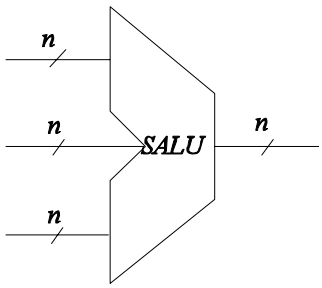
```
void
PrintResults(int *values, int numEntries)
{
  int i;
  if (numEntries == 0)
    {
     printf("empty\n");
     return;
    }

  for (i = 0; i < numEntries; i++)
    printf("%d ", values[i]);
  printf("\n");
}

/* sorts normal size integers only */
void
BubbleSort(int *input, int numEntries)
{
  int temp, store;
  int i, j;

  if (numEntries == 0)
    return;

  for (i = 0; i < numEntries; i++)
    {
     temp = i;
     for (j = i; j < numEntries; j++)
       {
        if (input[j] < input[temp])
          {
           temp = j;
          }
       }
     store = input[i];
     input[i] = input[temp];
     input[temp] = store;
    }

  return;
}
```

Assembly Program

```
#Known : start of input array is at address 1500
#            the value of numEntries is in r4

# Don't worry about negative numbers (although it should
# work), and don't worry about overflow conditions.

#Hint:  You don't need to use coprocessor or floating
#         point instructions
#
#Assume that numEntries is non-negative
#NOPs count as instructions.  (assume in bare mode)

         .data 1500
input:   .word 5, 4, 3, 2, 1

         .text
         .globl main

main :   beq r4, r0, exit        #if (NumEntries == 0)
         add r8, r0, r4          #r8 <- NumEntries
         ___________             #NE * 4: bytes -> words
         add r9, r0, r0          #i = 0

loop2:   add r10, r9, r0         #j = i
         add r11, r9, r0         #temp = i

loop:    _______________         #load input[j]
         _______________
         nop
         sub  r15, r13, r12
         bgtz r15, swap
         nop

cont:    addiu r10, r10, 4
         sub   r15, r8, r10      #NE4 - j
         bgtz  r15, loop         #if (j < NE4)
         lw    r14, 1500(r11)    #r14 <- input[temp]
         lw    r15, 1500(r9)     #r15 <- input[i]
         sw    r14, 1500(r9)     #input[i] <- r14
         sw    r15, 1500(r11)    #input[temp] <- r15
         addi  r9, r9, 4         #i++
```

```
                ____________
                ____________
                ____________
exit:   j r31               #return
        nop

swap:   add r11, r10, r0    #temp <- j
        j cont
        nop
```

5. You are a new Cal grad entrusted with the design of a new MIPS machine. The newest feature of the implementation is the Super ALU (SALU) shown below. It can take up to three n-bit inputs and output the n-bit result. The processor still has 32 registers.



One example of a new MIPS instruction that uses the SALU would be

```
add3    rd, rs, rt, ra          ; Reg[rd] = Reg[rs] + Reg[rt] + Reg[ra]
```

What other changes would you make to the instruction set to take advantage of the SALU?

a) What other instructions would you add? Use the notation above, and explain why they might be useful. **[4 points]**

b) What new data addressing mode(s) would you add? Why would they be useful? **[3 points]**

c) Let's assume that the word size of the instruction must be kept at 32-bits. Show the format of the new instructions add addressing modes. **[5 points]**

d) Sketch in the connections for the subsection of the datapath below. MUXes may be added when necessary. Be sure to label all the ports. **[8 points]**