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

CS 152
Fall 1995

D. Patterson & R. Yung

Computer Architecture and Engineering
Midterm I

Your Name:	
SID Number:	
Discussion Section:	

You may bring two pages of notes. You have 180 minutes. Please write your name on this cover sheet and also at the top left of each page. The point value of each question is indicated in brackets after it. Show your work. Write neatly and be well organized.
Good Luck!

Problem	Score
1	
2	
3	
4	
Total	

Question #1: Technology and performance **[20 pts]**

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

The first option is a Gallium Arsenide (GaAs) CPU. A 10 cm (about 4") diameter GaAs wafer costs $2000. The manufacturing process creates 4 defects per square cm. The CPU fabricated in this techonology is expected to have a clock rate of 1000 MHz, with an average clock cycles per instruction of 2.5 if we assume an infinitely fast memory system. The size of the GaAs CPU is 1.0 cm by 1.0 cm.

The second option is a CMOS CPU. A 20 cm (about 8") diameter CMOS wafer costs $1000 and has 1 defect per square cm. The 1.0 cm by 2.0 cm CPU executes multiple instructions per clock cycle to achieve an average clock cycles per instruction of 0.75, assuming an infinitely fast memory, while achieving a clock rate of 200 MHz. (The CPU is larger because it has on-chip caches and executes multiple instructions per clock cycle.)

Assume α is 2.0 for both GaAs and CMOS. Yield for GaAs and CMOS wafers are 0.8 and 0.9 respectively. Most of this information is summarized in the following table:

	Wafer Diam. (cm)	Wafer Yield	Cost ($)	Defects ($1/\text{cm}^2$)	Freq. (MHz)	CPI	Die Area (cm × cm)	Test Dies (per wafer)
GaAs	10	0.80	$2000	4	1000	2.5	1.0×1.0	4
CMOS	20	0.90	$1000	1	200	0.75	1.0×2.0	4

Hint: Here are two equations that may help:

$$\text{dies/wafer} = \frac{\pi \times (\text{wafer diameter}/2)^2}{\text{die area}} - \frac{\pi \times \text{ wafer diameter}}{\sqrt{2 \times \text{ die area}}} - \text{test dies per wafer}$$

$$\text{die yield} = \text{wafer yield } \times \left(1 + \frac{\text{defects per unit area} \times \text{die area}}{\alpha}\right)^{-\alpha}$$

a) Calculate the average execution time for each instruction with an infinitely fast memory. Which is faster and by what factor? Show your work. **[6 pts]**

b) How many seconds will each CPU take to execute a 1 billion instruction program? **[3 pts]**

c) What is the cost of an GaAs die for this CPU? Repeat the calculation for a CMOS die. Show your work. **[7 pts]**

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

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

Question #2: Instruction Set Architecture and Registers Sets **[15 pts]**

Imagine a modified MIPS instruction set with a register file consisting of 64 general-purpose registers rather than the usual 32. Assume that we still want to use a uniform instruction length of four bytes and that the total number of opcodes must remain unchanged. Also assume that you can expand and contract fields in an instruction, but that you cannot omit them.

a) How would the format of R-type (arithmetic and logical) instructions change? Label all the fields with their name and bit length. What is the consequence of this change? **[4 pts]**

b) How does this change the I-type instructions? What is the consequence of this change? **[3 pts]**

c) How does this change the J-type instructions? What is the consequence of this change? **[3 pts]**

d) Imagine we are translating machine code to use the larger register set. Give an example of an instruction that used to fit into the old format, but is impossible to translate directly into a single instruction in the new format. Write a short sequence of instructions that could replace it. **[5 pts]**

Question #3: Left Shift vs. Multiply **[20 pts]**

Design a 16-bit *left* shifter that shifts 0 to 15 bits using only 4:1 multiplexors.

a) How many levels of 4:1 multiplexors are needed? Show your work. **[4 pts]**

b) If the delay per multiplexor is 2ns, what is the speed of this shifter? (Assume zero delay for the wires.) **[3 pts]**

c) Draw the four leftmost bits of the multiplexors with the proper connections. (You might want to practice drawing it on the back of a page, and then transfer the final version here.) **[6 pts]**

d) Multiplies can be accomplished by left shifts if the multiplier is a power of two. Write the MIPS instruction(s) that performs multiply via left shifts for a multiplier that is positive and a power of two. Assume the multiplicand is in register \$4, the multiplier is in register \$5, and that the least significant 32 bits of the product should be left in \$6. You can ignore the potential for overflow in the result register. **[7 pts]**

extra credit) In class, we've seen three versions of the multiply operation (ignoring Booth encoding). Approximately how many clock cycles does a multiply take using the fastest algorithm? Approximately how much faster are multiplies that use the shifting technique (assuming the multiplier is a power of two)? **[+4 pts]**

Question #4: Enhancing the Single Cycle Datapath **[30 pts]**

Recall the 32-bit single-cycle control and datapath from class.

a) A MIPS instruction that would be useful to have for writing loops is Decrement and Branch if Not Zero (DBNZ). For example, the C loop:

```
for (i=n; i!=0; i--) {
   <loop body>
}
```

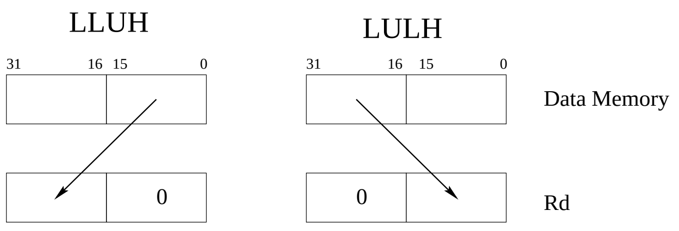
could be translated using DBNZ to:

```
              ld      R2, addrN       # i=n
              beq     R2, skiploop    # skip loop if n=0
     loop:     :
               <loop body>
               :
              dbnz    R2, loop        # if --i !=0 goto loop
 skiploop:     :
               :
```

MIPS currently does not have such an instruction (although other processors, such as the x86, do).

What changes and additions would be needed to the single cycle data path to support DBNZ? Modify the datapath below to show that support. You do not need to write the control signals. **[15 pts]**

b) The basic datapath supports only 32-bit loads. Imagine we wanted to augment the instruction set with new I-type load instructions that do the following:



The Load From Lower To Upper Halfword (LLUH) instruction takes the *least* significant 16 bits from a 32-bit word in the data memory and places them in the *most* significant 16 bits of the indicated result register; the *least* significant bits are all zeroed. The Load From Upper To Lower Halfword (LULH) instruction takes the *most* significant 16 bits from a 32-bit data memory word and places them in the *least* significant 16 bits of the result register; the *most* significant bits are all zeroed. In both cases, the address is generated in the same way as in a normal LW instruction.

Thus,

LLUH: Rd $\leftarrow$ $M[\text{offset} + \text{base}]_{15..0} \mid\mid 0^{16}$

LULH: Rd $\leftarrow$ $0^{16} \mid\mid M[\text{offset} + \text{base}]_{31..16}$

What *datapath* changes must be made to support this style of 16-bit loads? Modify the datapath reproduced below to show that support. **[15 pts]**