Computer Science 252 Quiz #1 **Name:**_____

1. 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 10 cm (≈4 inches) in diameter costs $1000. The manufacturing process creates 4 defects per square centimeter. The microprocessor fabricated in this technology is expect to have a clock cycle rate of 250 MHz, with an average clock cycles per instruction of 1.5 if we assume an infinitely fast memory system. The size of the GaAs microprocessor is 1.0 cm by 1.0 cm. (Assume α is 2.0 for GaAs.)

The second option is a CMOS microprocessor. A 15 cm (≈6 inch) wafer with 2 defects per square centimeter costs $500. The 1.0 cm by 2.0 cm microprocessor 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 50 MHz. (The microprocessor is larger because of on chip caches and executing multiple instructions per clock cycle.)

You found that a memory system can be built for either microprocessor, and that the average extra time for memory accesses per instruction (for instructions and data) is the same for each microprocessor: 10 ns.

a) What is the cost of an untested GaAs die for this microprocessor? (Use information from the book to supplement the information here if anything is missing.) Show your work. **[4 points]**

$$\text{Dies per wafer} = \frac{\pi * (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi * \text{Wafer diameter}}{\sqrt{2 * \text{Die area}}} - \text{Test dies per wafer}$$

$$\text{Dies per wafer} = \frac{\pi * (10/2)^2}{1} - \frac{\pi * 10}{\sqrt{2 * 1}} - 5 = \pi * (25 - \frac{10}{\sqrt{2}}) - 5 = 56 - 5 = 51$$

$$\text{Die yield} = \text{Wafer yield} * \left\{ 1 + \frac{\text{Defects per unit area} * \text{Die area}}{\alpha} \right\}^{-\alpha} = 90\% * \left\{ 1 + \frac{4 * 1}{2} \right\}^{-2} = 10\%$$

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} * \text{Die yield}} = \frac{\$1000}{51 * 10\%} = \$196.08$$

Forget test die -1 (unless said assume no test die)

b) What is the cost of an untested die for the CMOS microprocessor? (Use information from the book to supplement the information here if anything is missing.) Show your work. **[4 points]**

$$\text{Dies per wafer} = \frac{\pi * (15/2)^2}{2} - \frac{\pi * 15}{\sqrt{2 * 2}} - 5 = \pi * (25 - \frac{10}{2}) - 5 = 64 - 5 = 59$$

$$\text{Die yield} = \text{Wafer yield} * \left\{ 1 + \frac{\text{Defects per unit area} * \text{Die area}}{\alpha} \right\}^{-\alpha} = 90\% * \left\{ 1 + \frac{2 * 2}{2} \right\}^{-2} = 10\%$$

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} * \text{Die yield}} = \frac{\$500}{59 * 10\%} = \$84.75$$

c) What is the native MIPS rating of each microprocessor? Which is faster and by what percent? Show your work. **[6 points]**

$$CPI_{GaAs} = 1.5 + 10/4 = 4.0$$

$$MIPS_{GaAs} = \frac{\text{Clock rate}}{CPI * 10^6} = \frac{250}{4} = 62.5$$

$$CPI_{CMOS} = 0.75 + 10/20 = 1.25$$

$$MIPS_{CMOS} = \frac{\text{Clock rate}}{CPI * 10^6} = \frac{50}{1.25} = 40$$

$$\frac{\text{Performance}_{GaAs}}{\text{Performance}_{CMOS}} = 1.5625$$

The GaAs microprocessor is 56% faster than a CMOS microprocessor

Forget to account for memory access time: -3
Forget to say which is faster: -1
Get wrong % for which is faster: -1

d) Based solely on costs of the microprocessors calculated above, what is the cost/performance of each option? What is the ratio of cost/performance of the CMOS to GaAs microprocessors? **[4 points]**

GaAs microprocessor costs per MIPS is $3.08

CMOS microprocessor costs per MIPS is $2.12

The CMOS microprocessor cost/MIPS 0.69 vs GaAs cost per MIPS

(or the CMOS is cost/MIPS is 45% better)

Forget to give ratio (or get it wrong): -1

Name:_____

2. A *predicted-branch* instruction is as a way to improve the performance of delayed branches. The idea is to include another branch instruction that indicates that the instruction in the delay slot should be aborted if the branch is mispredicted. The advantage of predicted branches is that the delay slot can **always** be filled, since the branch can abort the contents of the delay slot if mispredicted. The compiler need not worry about whether the instruction is OK to execute when the branch is mispredicted.

A simple version of predicted branches does not execute the instruction in the delay slot if the branch is not taken; assume this type of predicted branch, which we call *branch likely*, is added to DLX. Assume that 27% of the branch-delay slots are filled with useful instruction using standard delayed branches, and that the rest of the slots are filled using branch likely instructions where the instruction put into the delay slot is from the target of the branch.

a) What is the relative effectiveness of the schemes? First assume the likely branches are taken with the frequency quoted in the book. Next assume that the compilers are changed to try to change the code so that the branch likely instructions are taken 80% of the time. Show the percentage of delayed slots filled and percentage of delayed slots **usefully** filled for GCC. **[8 points]**

% branch taken (3.22 for GCC) 54%

Alternative source: 53% from average of 3.22

Percentage of delayed slots filled for GCC = 100% **[2 points]**

Percentage of delayed slots **usefully** filled for GCC at 54% taken branch =

27% * 100% + (100% -27%) * 54% = 66% **[4 points]**

Percentage of delayed slots **usefully** filled for GCC at 80% taken branch =

27% * 100% + (100% -27%) * 80% = 85% **[2 points more]**

*Use 27% delayed * 58% vs 27% delayed * 100%: -3*
Use 65% taken: -1 (this is of all jumps & branches, not just conditional branches)

b) How much faster would a DLX machine run GCC that added this style of branch, assuming there is no clock-speed penalty compared to a machine with only delayed branches? Show the results for both assumptions on fraction of taken branch likely instructions. **[8 points]**

% branch taken (3.22 for GCC) 54%

Alternative source: 53% from average of 3.22

% usefully filled of delayed branch for GCC 48% (Fig 6.22)

Assume non-branch instruction CPI = 1

CPI_{base} = 22% * (1 + (100% - 48%)*1) + (100% - 22%) * 1 = 1.11

$CPI_{54\% \text{ taken}}$ = 22% * (1 + (100% - 27%)*(100% - 54%)*1) + (100% - 22%) * 1 = 1.07

$CPI_{80\% \text{ taken}}$ = 22% * (1 + (100% - 27%)*(100% - 80%)*1) + (100% - 22%) * 1 = 1.03

54% taken: DLX with branch likely is 1.11/1.07 = 1.04 or 4% faster than delayed branch only

80% taken: DLX with branch likely is 1.11/1.03 = 1.08 or 8% faster than delayed branch only

Forget to multiply by frequency of instruction when calculating speed: -3
(e.g., speedup of branch instructions vs. overall speedup)
Assuming performance of delayed branch only case is that (100%-27%) of brnaches have a full stall: -2
Comparing 54% to 80% vs. 54% to delayed and 80% to delayed: -1

Don't assume branch delay is one branch cycle: -1
No source for branch frequency: -1
No source mentioned for % branch taken: -1

Name:_____

3. You are charged with investigating a new addressing mode that allows one source operand to be in memory for ALU instructions on DLX. To offset this increase in complexity you restrict all memory addressing to be register indirect only (e.g., no displacement addressing.)

a) Propose a change to the DLX pipeline in Chapter 6 to include this addressing mode. Draw the new pipeline and explain the changes. **[8 points]**

Stage	ALU instruction	Load or store instruction	Branch instruction
IF	IMAR←PC; IR←Mem[IMAR]; PC←PC+4;	IMAR←PC; IR←Mem[IMAR]; PC←PC+4;	IMAR←PC; IR←Mem[IMAR]; PC←PC+4;
ID	A ← DMAR ←Rs1; B ← MDR ←Rs2;	A ← DMAR ←Rs1; B ← MDR ←Rs2;	A ← DMAR ←Rs1; B ← MDR ←Rs2;
MEM	MDR2 ← Mem[DMAR];	MDR2← Mem[DMAR]; or Mem[DMAR]←MDR;	if (cond) PC←ALUoutput;
EX	ALUoutput ← A *op* B; or ALUoutput←A *op* $((IR_{16})^{16}\#\#IR_{16..31})$; or ALUoutput ← MDR2 *op* B; or ALUoutput ← MDR2 *op* $((IR_{16})^{16}\#\#IR_{16..31})$;	MDR3 ← MDR2	ALUoutput←PC + $((IR_{16})^{16}\#\#IR_{16..31})$; cond←(Rs1 *op* 0);
WB	Rd ← ALUoutput;	Rd ← MDR3;	

Best alternative is basically switch 2 phases and still take 5 phases; **[8 points]**
*next best is realizing that need only one memory access phase, but have 6 phases;***[6 points]**
3rd best is more than 5 phases with 2 memory access phases **[4 points]**
Combine operations that are 2 clock periods in DLX into one clock period in revised DLX **[1 points]**

b) What new data hazards, if any, are created by this addressing mode? Give an example of each, classify the type of data hazard, and show the conflict using the data above.

Best alternative (switch): **[4 points]**
RAW hazards

```
ADD R1,R2,R3
NOP
ADD R4,(R1),R5
```

Since need to have register value by end of ID, must forward the result of EX phase 2 instructions prior

```
ADD R1,R2,R3
ADD R4,(R1),R5
```

Since need to have register value by end of ID, and the prior instruction does not calculate the result until the next clock phase, there is no alternative but to stall (Analogous to the delayed load case.)

WAW and WAR:
registers: none since register writes always in last phase and register reads always in 2nd phase.
memory: none since memory reads or writes are always in the 3rd phase

Second alternative (one memory): **[4 points]**
Same as above
Also more forwarding hardware/hazard detection since the time between readings operands (ID) and changing them (second EX phase) is longer. Will need to stall change followed by use if change occurs in second EX step.

Third alternative (two memory phases): **[4 points]**
memory RAW: may be reading what prior instruction is writing
plus hazards from above

c) List all changes or additions must be made to the hardware to accommodate this mode. **[4 points]**
Best alternative (switch): **[4 points]**
New MDRs; new paths register file to DMAR and DMDR; mux on DMAR to handle forwarding; hardware to stall on RAW hazard;

Second alternative (one memory): **[4 points]**
As above plus hazard detection due to longer gap between read and calculate new values

Third alternative (two memory): **[4 points]**
Another memory port, hazard detection on memory addresses which may lead to stall, plus items mentioned above

d) List all questions that must be answered to estimate the impact of this change on performance. **[4 points]**

Impact on clock cycle time?

% loads and stores use displacement addressing? (suggests number extra instructions)

% loads put a value in a register only used once by an ALU operation? (suggests savings of loads due to new addressing option)

After compilers changed so that displacement addressing must be calculated in a register, what % of instructions that calculate an address that are immediately prior to the instruction that uses it? (suggests the number of stalls that must be added)

Extra credit for no points: if you have the time, try to see if you can find evidence from the statistics on DLX or other architectures to answer the questions on the impact on performance.