University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Fall 1997 D.A. Patterson

Midterm I - SOLUTIONS
October 8, 1997
CS152 Computer Architecture and Engineering

You are allowed to use a calculator and one 8.5" x 1" double-sided page of notes. You have 3 hours. Good luck!

Your Name:	
SID Number:	
Discussion Section:	

1	/20
2	/10
3	/10
4	/30
Total	**/70**

Question 1

You are running a benchmark on your company's processor, which runs at 200 MHz, and has these characteristics:

Instruction Type	Frequency (%)	Cycles
Arithmetic and Logical	40	1
Load and Store	30	2
Branches	20	3
Floating Point	10	4

Your company is considering offering a cheaper, lower-performance version of the processor. Their plan is to remove some of the floating point hardware to reduce the die size of the chip.

The wafer on which the chip is produced has a diameter of 10cm, a cost of \$1000, and $1/(\mathrm{cm}^2)$ defects. The manufacturing process results in a 90% wafer yield and a value of 2 for α.

The current processor has a die size of 12mm$\times$12mm. After the changes, the die size will be 10mm$\times$10mm, and floating point instructions will take 12 cycles to execute.

Here are some equations you may find useful:

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

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

a) What is the CPI and MIPS rating of the original processor?

$CPI = (1 * .4) + (2 * .3) + (3 * .2) + (4 * .1) = 2$

$MIPS = 200/2 = 100$

b) What is the CPI and MIPS rating of the smaller processor?

$CPI = (1 * .4) + (2 * .3) + (3 * .2) + (12 * .1) = 2.8$

$MIPS = 200/2.4 = 71.4$

Question 1 (cont)

c) What is the old cost per (working) processor?

$$\begin{aligned}\text{dies/wafer} &= \tfrac{\pi*(5)^2}{1.44} \Leftrightarrow \tfrac{\pi*10}{\sqrt{2*1.44}}\\ &= \tfrac{78.54}{1.44} \Leftrightarrow \tfrac{31.42}{1.70}\\ &= 54.54 \Leftrightarrow 18.48\\ &= 36.05 = 36\end{aligned}$$

$$\begin{aligned}\text{die yield} &= .9 * \left(1 + \tfrac{1*1.44}{2}\right)^{-2}\\ &= .9 * (1 + .72)^{-2}\\ &= .9 * .338 = .30\end{aligned}$$

working processors $= .30 * 36 = 10$

die cost $= \$1000/10 = \$100$

d) What is the new cost per (working) processor?

$$\begin{aligned}\text{dies/wafer} &= \tfrac{\pi*(5)^2}{1} \Leftrightarrow \tfrac{\pi*10}{\sqrt{2*1}}\\ &= \tfrac{78.54}{1} \Leftrightarrow \tfrac{31.42}{1.41}\\ &= 78.54 \Leftrightarrow 22.28\\ &= 56.26 = 56\end{aligned}$$

$$\begin{aligned}\text{die yield} &= .9 * \left(1 + \tfrac{1*1}{2}\right)^{-2}\\ &= .9 * (1 + .5)^{-2}\\ &= .9 * .444 = .40\end{aligned}$$

working processors $= .40 * 56 = 22$

die cost $= \$1000/22 = \$45$

e) What is the improvement in price per performance?

$$\frac{\$100/100}{\$45/71.4} = \frac{1}{.63} = 1.59$$

Question 1 (cont)

Your competitors produce a chip that runs at 250 MHz and has the following characteristics for the benchmark:

Instruction Type	Frequency (%)	Cycles
Arithmetic and Logical	40	1
Load and Store	30	3
Branches	20	3
Floating Point	10	5

f) What is the CPI and MIPS rating of your competitor's processor for this benchmark?

CPI = $(.4 * 1) + (.3 * 3) + (.2 * 3) + (.1 * 5) = 2.4$
MIPS = $250/2.4 = 104$

g) Your company's advertising department wants to defend your company's motto ("The Appearance of Excellence") by advertising a higher MIPS rating for your processor than your competitor's processor. They want you to write a benchmark that gives this result. Describe an instruction mix that would accomplish this (give specific percentages of each instruction type).

Your benchmark must have a large amount of load or store instructions, since these instructions execute in less time on your processor than on your competitor's processor. For example, a mix of half arithmetic/logical and half load/store would give your machine:
CPI = $(.5 * 1) + (.5 * 2) = 1.5$; MIPS = $200/1.5 = 167$
and your competitor's machine:
CPI = $(.5 * 1) + (.5 * 3) = 2$; MIPS = $250/2 = 125$

Question 1 (cont)

h) Instead, you decide to improve the compiler that is used to compile this benchmark on your processor. Your compiler reduces the branches by 50%, but it increases the number of arithmetic and logical instructions by 25%; it does not affect the number of other instructions. What is your new CPI and MIPS?

CPI $= (.5 * 1) + (.3 * 2) + (.1 * 3) + (.1 * 4) = 1.8$
MIPS $= 200/1.8 = 111$

i) Using the original instruction mix, which machine is faster? Why?

$$Speedup = \frac{ExecutionTime_{excellence}}{ExecutionTime_{competitor}} = \frac{CPI_{excellence}TCLK_{excellence}}{CPI_{competitor}TCLK_{competitor}} = \frac{2*5}{2.4*4} = 1.0416$$

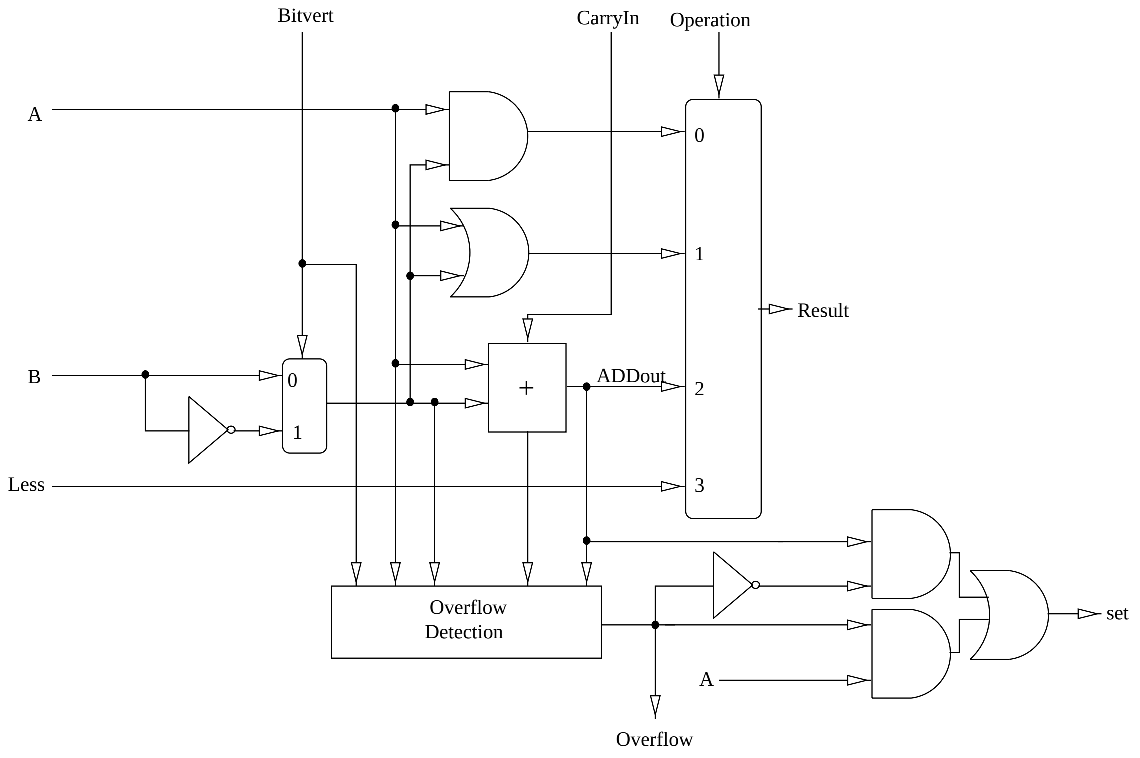
Thus, the competitor's processor is 1.0416 times faster or 4.16% faster.

Question 2

The ALU presented in the book supported set less than (*slt*) using the sign bit of the adder ($a < b \Leftrightarrow a \Leftrightarrow b < 0$). Let's try the set less than operation using the values $\Leftrightarrow 7_{ten}$ and 6_{ten}. To make it simpler to follow the example, let's limit the binary representation to 4 bits: 1001_{two} and 0110_{two}.

$1001_{two} \Leftrightarrow 0110_{two} = 1001_{two} + 1010_{two} = 0011_{two}$

The result suggests that $\Leftrightarrow 7 > 6$, which is clearly wrong. Hence we must factor in overflow in the decision. Modify the given schematic below of the 1-bit ALU for the most significant bit to handle *slt* correctly. You have to fix the *set* output, which is same as ADD_{out} in this schematic. Explain your modifications and give the new function for the *set* output as well. Assume that the *overflow* signal is correct and can be used and that the gates available to you are: inverters, AND and OR.



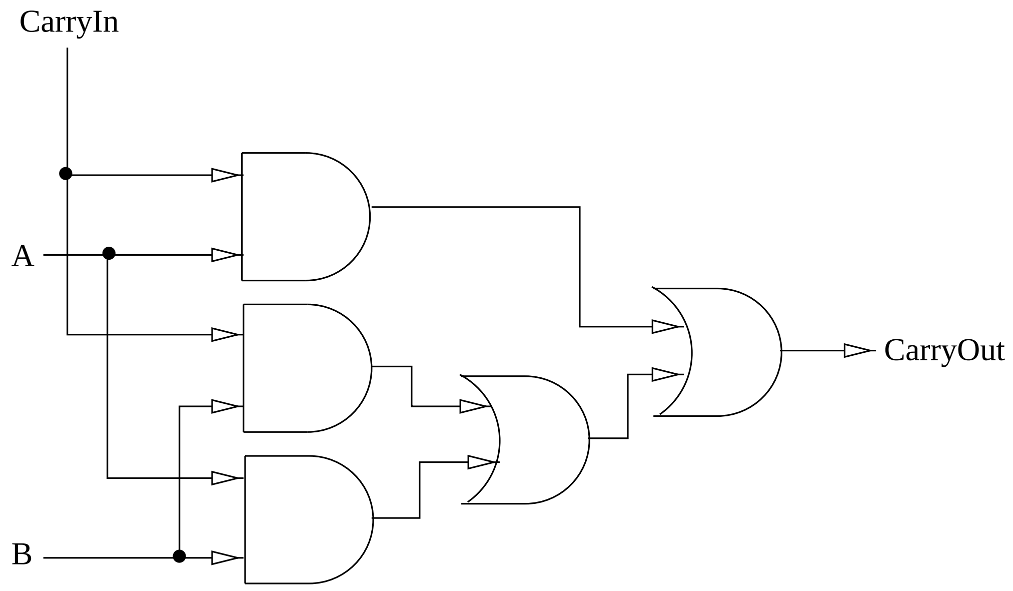
When a and b have the same sign there can be no overflow, so *set* is the output of the adder (just like before). An overflow can happen only when $a > 0$ and $b < 0$, or $a < 0$ and $b > 0$. In the first case *set* should be 0 ($a > b$), while in the second one it should be 1 ($a < b$). So the function for *set* is:

$set = \overline{overflow} \cdot adder\_output + overflow \cdot a$
where a is the MS bit of the first operand, in other words its sign.
This can be implemented with 1 inverter, 2 AND and 1 OR gates as shown in the schematic above.

Another valid solution is to XOR the *overflow* and *ADDout.* signals. In the case of non-overflow, *set* is the same with *ADDout.* When overflow occurs, *set* is the inverse of *ADDout.*

Question 3

The figure picture presents the portion of the schematic of a full adder that calculates the carry out signal.



Assume the following characteristics for the gates:

AND2: Input load=150fF, propagation delay low-to-high TPlh=0.2ns, propagation delay high-to-low TPhl=0.5ns, load dependent delay TPlhf=TPhlf=0.002ns/fF .

OR2: Input load=100fF, propagation delay low-to-high TPlh=0.5ns, propagation delay high-to-low TPhl=0.1ns, load dependent delay TPlhf=TPhlf=0.002ns/fF .

Identify the critical path in this schematic and fully characterize its delay using the linear delay model. Assume that the last OR2 gate drives a capacitance of 300fF.

The critical path for CarryOut consists of 3 gates, one AND and two OR in the row. There is no need to calculate the delay for the path with the 2 gates (one AND and one OR) since, excluding the additional OR gate, the two paths have the same gates with the same loads.

$TP_{total} = TP_{AND} + TP_{OR_1} + TP_{OR_2}$
For each gate: $TP = TP_{inherent} + TP_{loaddepedent} \cdot Cap_{load}$

For low-to-high:
$TP_{LH} = (0.2 + 0.002 \cdot 100) + (0.5 + 0.002 \cdot 100) + (0.5 + 0.002 \cdot 300) = 2.2ns$

For high-to-low:
$TP_{HL} = (0.5 + 0.002 \cdot 100) + (0.1 + 0.002 \cdot 100) + (0.1 + 0.002 \cdot 300) = 1.7ns$

Since the cell delay is the worst case one, the delay of the CarryOut calculation is 2.2ns.

Question 4

In October of 1996, Silicon Graphics introduced a new set of instructions known as MIPS Digital Media Extensions (MDMX). Similar to the Intel MMX, the MDMX specification uses a single instruction multiple data (SIMD) data path to perform parallel narrow data operations on bytes and halfwords within a single instruction.

The MDMX has yet to be implemented on a commercially available microprocessor. We will explore some MDMX ideas by extending the single cycle datapath discussed in class. Where the real MDMX uses 64-bit floating point registers, we will use the 32-bit integer registers to perform parallel operations on two half words ("Dual Halfs" instructions).

Consider two pseudo-MDMX instructions (based on the real MDMX!), ADD.DH and MAX.DH. ADD.DH adds two 16-bit signed integers in parallel. MAX.DH is more unusual. It performs two simultaneous comparisons and stores the larger results in a third register. The register transfer operations are given below. R[x][0] refers to the half word in bits 15:0 of register x, and R[x][1] refers to bits 31:16.

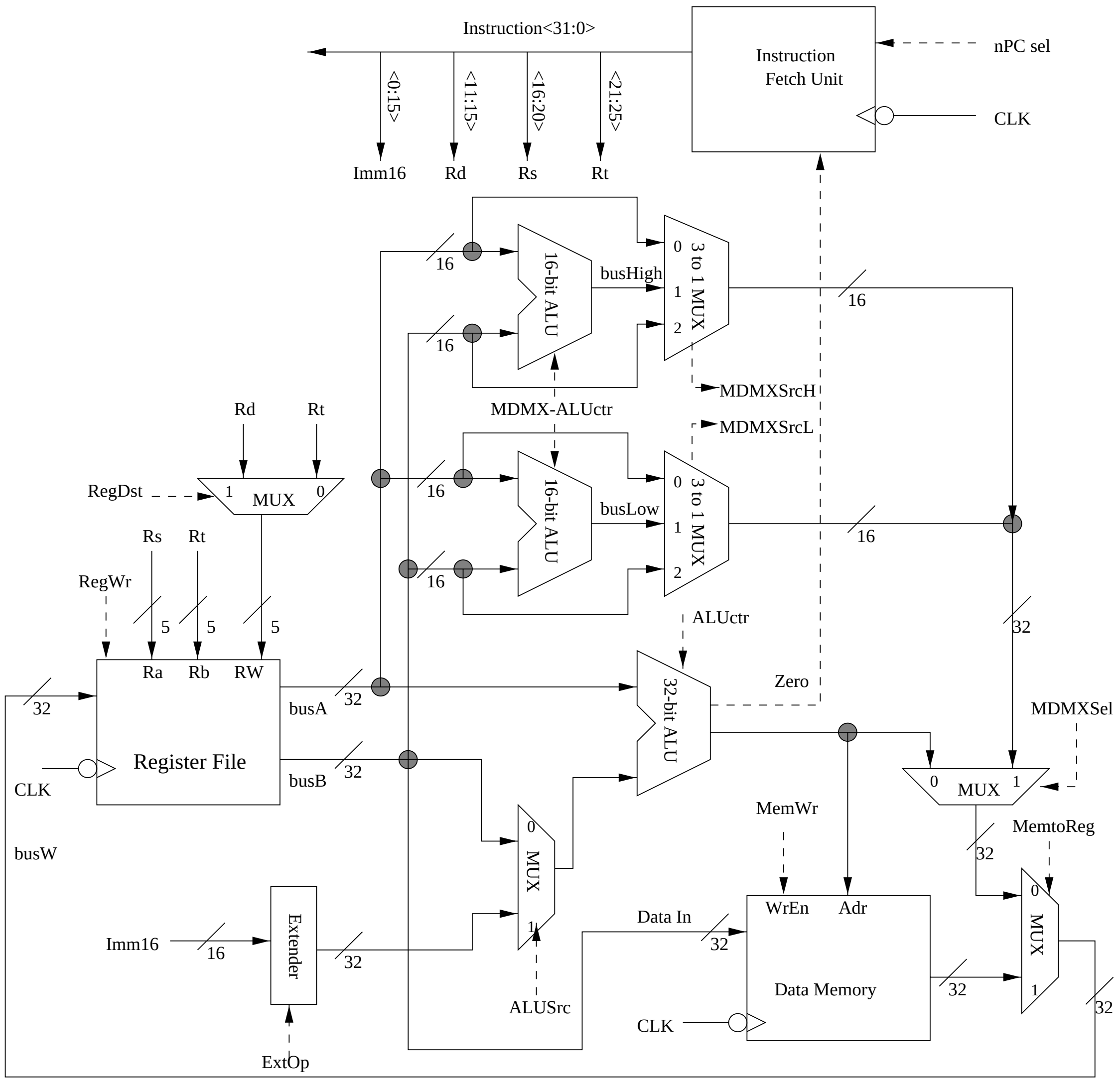
```
INSTRUCTION rd, rs, rt
ADD.DH $r1, $r2, $r3 R[rd][0]⇐R[rs][0]+R[rt][0]
                     R[rd][1]⇐R[rs][1]+R[rt][1]
                     PC⇐PC+4

MAX.DH $r1, $r2, $r3   for i=0,1 begin
                           if (R[rs][i] < R[rt][i]) then
                               R[rd][i]⇐R[rt][i]
                           else
                               R[rd][i]⇐R[rs][i]
                       end
                       PC⇐PC+4
```

Question 4 (cont)

a) The single cycle processor developed in class is shown below. Make the necessary datapath modifications for the two MDMX instructions. You may use a 16-bit version of the 32-bit ALU. If you define your own component, be sure to specify its behavior. Label your control signals with descriptive names. To maximize your chances for partial credit, write down anything else that will help us evaluate your work (you do not need to specify the control functions until part b). You will be graded for correctness more than efficiency. An additional datapath (in case you needed it) is provided in the following page.

The meaning of some control signals are given below.
$nPCSel : 0 \Rightarrow PC \leftarrow PC + 4; 1 \Rightarrow PC \leftarrow PC + 4 + SignExt(Imm16) \mid\mid 00$
$ALUCtr$: "add", "sub", "and", "or", "slt"(signed comparison)
$ExtOp$: "$zero$", "$sign$"



b) For ADD.DH and MAX.DH, give the values of all control signals, including those you added in part (a). The control signals can be functions of other control signals, values labeled on the datapath, or don't cares. Use *b[i]* to denote bit *i* on bus *b*. You may use high level specifications such as *if-then-else*. The number of grids below is not an indication of the number of control signals you will need.

Control Line	ADD.DH	MAX.DH
nPCSel	0	0
RegDst	1	1
RegWr	1	1
ExtOp	X	X
ALUSrc	X	X
ALUCtr	X	X
MemWr	0	0
MemtoReg	0	0
MDMX-ALUctr	add	slt
MDMXSrcH	1	if(busHigh[0]==1) 0 else 2
MDMXSrcL	1	if(busLow[0]==1) 0 else 2
MDMXSel	1	1

Extra Credit. 16 16-bit signed integers are stored in memory as follows:

0x00000000 (half-word 0)
0x00000002 (half-word 1)
0x00000004 (half-word 2)
.
.
0x0000001E (half-word 15)

The following MIPS code (assuming no delay slots) finds the largest integer and stores the result in lower 16 bits of \$v0. Since we are dealing with half words, the final value of the upper 16 bits of \$v0 is irrelevant.

```
   lh   $v0, 0x001E($zero)        #assume the last number is the largest
   addi $t0, $zero, 0x001C        #initialize the half-word pointer
LARGEST:
   lh   $s0, 0($t0)               #load next half word
   slt  $t1, $v0, $s0             #update $v0 if $s0 is larger
   beq  $t1, $zero, NEXT
   addi $v0, $zero, $s0
NEXT:
   addi $t0, $t0, -2              #continue the search until pointer becomes negative
   slt  $t1, $t0, $zero
   beq  $t1, $zero, LARGEST
END:
```

Take advantage of the parallelism in MAX.DH to write a faster version of this code. Exactly how many instructions are executed in your code? What are the minimum and maximum number of instructions executed in the non-MDMX code given above?

Answer:

```
   lw      $v0, 0x001C($zero)
   addi    $t0, $zero, 0x0018
LARGEST:
   lw      $v1, 0($t0)
   max.dh  $v0, $v0, $v1
   addi    $t0, $t0, -4
   slt     $t1, $t0, $zero
   beq     $t1, $zero, LARGEST

   srl     $v1, $v0, 16  #put the larger half-word in $v0 into the lower
   max.dh  $v0, $v0, $v1 #16-bits of $v0
END:
```

Instruction counts:
MDMX $\Rightarrow 2 + 7*5 + 2 = 39$
non-MDMX min $\Rightarrow 2 + 15*6 = 92$
non-MDMX max $\Rightarrow 2 + 15*7 = 107$