San Diego State University
CS 370 Computer Architecture

Homework Assignment 3 [100 points+20 points bonus, Weight 8%]

- You are required to write by hand legibly to solve each problem. When you plot a circuit, please make sure that you use a ruler. You are not allowed to use any software to draw a circuit or answer any question. An answer that is not written by your hand will receive zero credit for the corresponding problem.
- Please write down your full name and SDSU Red ID on the first scratch paper. After you finish writing all your answers on multiple scratch papers (print papers preferred), please take a photo for each of them. Next, please insert these image files into a Word file. Finally, convert the Word file into a single PDF file, and then, submit the PDF file from within Canvas.
- Please note that we only accept your PDF file submissions. A submission in any other format will not be graded and will receive zero credit. For example, submitting multiple image files will receive zero credit.
- Please pay special attention to the due date – a late submission will receive zero.
- For your convenience, the text of each question is appended to this document.
- Please do the following problems from the textbook, and submit solutions.

1. [13 points] Problem 7-3
2. [13 points] Problem 7-9
3. [13 points] Problem 8-5
4. [8 points] Exercise 5.2 (part a) of the reference book (5.2.1 is 1 point; 5.2.2 is 1 point; 5.2.3 is 1 point; 5.2.4 is 2 points; 5.2.5 is 2 point; 5.2.6 is 1 point)
5. [8 points] Exercise 5.4 (part a) of the reference book (5.4.1 is 1 point; 5.4.2 is 1 point; 5.4.3 is 1 point; 5.4.4 is 1 point; 5.4.5 is 2 points; 5.4.6 is 2 points)
6. [8 points] Exercise 5.7 (part a) of the reference book (5.7.1 is 1 point; 5.7.2 is 1 point; 5.7.3 is 1 point; 5.7.4 is 1 point; 5.7.5 is 2 points; 5.7.6 is 2 points)
7. [7 points] Exercise 6.2 (part a) of the reference book (6.2.1 is 1 point; 6.2.2 is 2 points; 6.2.3 is 2 points; 6.2.4 is 2 points)
8. [7 points] Exercise 6.3 (part a) of the reference book (6.3.1 is 2 points; 6.3.2 is 2 points; 6.3.3 is 3 points)
9. [7 points] Exercise 6.6 (part a) of the reference book (6.6.1 is 2 points; 6.6.2 is 2 points; 6.6.3 is 3 points)
10. [8 points] Exercise 4.2 (part a) of the reference book (4.2.1 is 1 point; 4.2.2 is 1 point; 4.2.3 is 1 point; 4.2.4 is 1 point; 4.2.5 is 2 points; 4.2.6 is 2 points)
11. [8 points] Exercise 4.12 (part a) of the reference book (4.12.1 is 1 point; 4.12.2 is 1 point; 4.12.3 is 1 point; 4.12.4 is 1 point; 4.12.5 is 1 point; 4.12.6 is 3 points)
12. [6 points bonus] Exercise 4.13 (part a) of the reference book (4.13.1 is 1 point; 4.13.2 is 1 point; 4.13.3 is 1 point; 4.13.4 is 1 point; 4.13.5 is 1 point; 4.13.6 is 1 point)
13. [6 points bonus] Exercise 4.17 (part a) of the reference book (4.17.1 is 1 point; 4.17.2 is 1 point; 4.17.3 is 1 point; 4.17.4 is 1 point; 4.17.5 is 1 point; 4.17.6 is 1 point)
14. [8 points bonus] Exercise 4.21 (part a) of the reference book (4.21.1 is 1 point; 4.21.2 is 1 point; 4.21.3 is 1 point; 4.21.4 is 2 points; 4.21.5 is 1 point; 4.21.6 is 2 points)

7-3. *A 64K × 16 RAM chip uses coincident decoding by splitting the internal decoder into row select and column select. (a) Assuming that the RAM cell array is square, what is the size of each decoder, and how many AND gates are required for decoding an address? (b) Determine the row and column selection lines that are enabled when the input address is the binary equivalent of $(32000)_{10}$.

7-9. Using the 64K × 8 RAM chip in Figure 7-9 plus a decoder, construct the block diagram for a 1M × 32 RAM.

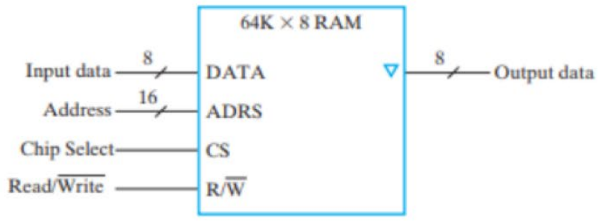

FIGURE 7-9

8-5. Inputs X_i and Y_i of each full adder in an arithmetic circuit have digital logic specified by the Boolean functions

$$X_i = A_i \quad Y_i = \overline{B_i}S + B_i\overline{C_{in}}$$

where S is a selection variable, C_{in} is the input carry, and A_i and B_i are input data for stage *i*.

(a) Draw the logic diagram for the 4-bit circuit, using full adders and multiplexers.

(b) Determine the arithmetic operation performed for each of the four combinations of S and C_{in}: 00, 01, 10, and 11.

Exercise 5.2

In this exercise we look at memory locality properties of matrix computation. The following code is written in C, where elements within the same row are stored contiguously.

a.	`for (I=0; I<8; I++)`
	`  for (J=0; J<8000; J++)`
	`    A[I][J]=B[I][0]+A[J][I];`

5.2.1 [5] <5.1> How many 32-bit integers can be stored in a 16-byte cache line?

5.2.2 [5] <5.1> References to which variables exhibit temporal locality?

5.2.3 [5] <5.1> References to which variables exhibit spatial locality?

Locality is affected by both the reference order and data layout. The same computation can also be written below in Matlab, which differs from C by contiguously storing matrix elements within the same column.

a.

```
for I=1:8
  for J=1:8000
    A(I,J)=B(I,0)+A(J,I);
  end
end
```

5.2.4 [10] <5.1> How many 16-byte cache lines are needed to store all 32-bit matrix elements being referenced?

5.2.5 [5] <5.1> References to which variables exhibit temporal locality?

5.2.6 [5] <5.1> References to which variables exhibit spatial locality?

Exercise 5.4

For a direct-mapped cache design with a 32-bit address, the following bits of the address are used to access the cache.

	Tag	Index	Offset
a.	31–10	9–5	4–0

5.4.1 [5] <5.2> What is the cache line size (in words)?

5.4.2 [5] <5.2> How many entries does the cache have?

5.4.3 [5] <5.2> What is the ratio between total bits required for such a cache implementation over the data storage bits?

Starting from power on, the following byte-addressed cache references are recorded.

Address	0	4	16	132	232	160	1024	30	140	3100	180	2180

5.4.4 [10] <5.2> How many blocks are replaced?

5.4.5 [10] <5.2> What is the hit ratio?

5.4.6 [20] <5.2> List the final state of the cache, with each valid entry represented as a record of <index, tag, data>.

Exercise 5.7

In this exercise, we will look at the different ways capacity affects overall performance. In general, cache access time is proportional to capacity. Assume that main memory accesses take 70 ns and that memory accesses are 36% of all instructions. The following table shows data for L1 caches attached to each of two processors, P1 and P2.

		L1 Size	L1 Miss Rate	L1 Hit Time
a.	P1	2 KB	8.0%	0.66 ns
	P2	4 KB	6.0%	0.90 ns

5.7.1 [5] <5.3> Assuming that the L1 hit time determines the cycle times for P1 and P2, what are their respective clock rates?

5.7.2 [5] <5.3> What is the AMAT for P1 and P2?

5.7.3 [5] <5.3> Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 and P2? Which processor is faster?

For the next three problems, we will consider the addition of an L2 cache to P1 to presumably make up for its limited L1 cache capacity. Use the L1 cache capacities and hit times from the previous table when solving these problems. The L2 miss rate indicated is its local miss rate.

	L2 Size	L2 Miss Rate	L2 Hit Time
a.	1 MB	95%	5.62 ns

5.7.4 [10] <5.3> What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

5.7.5 [5] <5.3> Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 with the addition of an L2 cache?

5.7.6 [10] <5.3> Which processor is faster, now that P1 has an L2 cache? If P1 is faster, what miss rate would P2 need in its L1 cache to match P1's performance? If P2 is faster, what miss rate would P1 need in its L1 cache to match P2's performance?

Exercise 6.2

Mean Time Between Failures (MTBF), Mean Time To Replacement (MTTR), and Mean Time To Failure (MTTF) are useful metrics for evaluating the reliability and availability of a storage resource. Explore these concepts by answering the questions about devices with the following metrics.

	MTTF	MTTR
a.	3 Years	1 Day

6.2.1 [5] <6.1, 6.2> Calculate the MTBF for each of the devices in the table.

6.2.2 [5] <6.1, 6.2> Calculate the availability for each of the devices in the table.

6.2.3 [5] <6.1, 6.2> What happens to availability as the MTTR approaches 0? Is this a realistic situation?

6.2.4 [5] <6.1, 6.2> What happens to availability as the MTTR gets very high, i.e., a device is difficult to repair? Does this imply the device has low availability?

Exercise 6.3

Average and minimum times for reading and writing to storage devices are common measurements used to compare devices. Using techniques from Chapter 6, calculate values related to read and write time for disks with the following characteristics.

	Average Seek Time	RPM	Disk Transfer Rate	Controller Transfer Rate
a.	10 ms	7500	90 MB/s	100 MB/s

6.3.1 [10] <6.2, 6.3> Calculate the average time to read or write a 1024-byte sector for each disk listed in the table.

6.3.2 [10] <6.2, 6.3> Calculate the minimum time to read or write a 2048-byte sector for each disk listed in the table.

6.3.3 [10] <6.2, 6.3> For each disk in the table, determine the dominant factor for performance. Specifically, if you could make an improvement to any aspect of the disk, what would you choose? If there is no dominant factor, explain why.

Exercise 6.6

Explore the nature of FLASH memory by answering the questions related to performance for FLASH memories with the following characteristics.

	Data Transfer Rate	Controller Transfer Rate
a.	120 MB/s	100 MB/s

6.6.1 [10] <6.2, 6.3, 6.4> Calculate the average time to read or write a 1024-byte sector for each FLASH memory listed in the table.

6.6.2 [10] <6.2, 6.3, 6.4> Calculate the minimum time to read or write a 512-byte sector for each FLASH memory listed in the table.

6.6.3 [5] <6.2, 6.3, 6.4> Figure 6.6 shows that FLASH memory read and write access times increase as FLASH memory gets larger. Is this unexpected? What factors cause this?

Exercise 4.2

The basic single-cycle MIPS implementation in Figure 4.2 can only implement some instructions. New instructions can be added to an existing ISA, but the decision whether or not to do that depends, among other things, on the cost and complexity such an addition introduces into the processor datapath and control. The first three problems in this exercise refer to this new instruction:

	Instruction	Interpretation
a.	`SEQ Rd,Rs,Rt`	`Reg[Rd]` = Boolean value (0 or 1) of (`Reg[Rs] == Reg[Rs]`)

4.2.1 [10] <4.1> Which existing blocks (if any) can be used for this instruction?

4.2.2 [10] <4.1> Which new functional blocks (if any) do we need for this instruction?

4.2.3 [10] <4.1> What new signals do we need (if any) from the control unit to support this instruction?

When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance trade-off. In the following three problems, assume that we are starting with a datapath from Figure 4.2, where I-Mem, Add, Mux, ALU, Regs, D-Mem, and Control blocks have latencies of 400ps, 100ps, 30ps, 120ps, 200ps, 350ps, and 100ps, respectively, and costs of 1000, 30, 10, 100, 200, 2000, and 500, respectively. The remaining three problems in this exercise refer to the following processor improvement:

	Improvement	Latency	Cost	Benefit
a.	Add Multiplier to ALU	+300ps for ALU	+600 for ALU	Lets us add MUL instruction. Allows us to execute 5% fewer instructions (MUL no longer emulated).

4.2.4 [10] <4.1> What is the clock cycle time with and without this improvement?

4.2.5 [10] <4.1> What is the speedup achieved by adding this improvement?

4.2.6 [10] <4.1> Compare the cost/performance ratio with and without this improvement.

Exercise 4.12

In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:

	IF	ID	EX	MEM	WB
a.	250ps	350ps	150ps	300ps	200ps

4.12.1 [5] <4.5> What is the clock cycle time in a pipelined and non-pipelined processor?

4.12.2 [10] <4.5> What is the total latency of an `LW` instruction in a pipelined and non-pipelined processor?

4.12.3 [10] <4.5> If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

The remaining problems in this exercise assume that instructions executed by the processor are broken down as follows:

	ALU	BEQ	LW	SW
a.	45%	20%	20%	15%

4.12.4 [10] <4.5> Assuming there are no stalls or hazards, what is the utilization of the data memory?

4.12.5 [10] <4.5> Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit?

4.12.6 [30] <4.5> Instead of a single-cycle organization, we can use a multi-cycle organization where each instruction takes multiple cycles but one instruction finishes before another is fetched. In this organization, an instruction only goes through stages it actually needs (e.g., ST only takes 4 cycles because it does not need the WB stage). Compare clock cycle times and execution times with single-cycle, multi-cycle, and pipelined organization.

Exercise 4.13

In this exercise, we examine how data dependences affect execution in the basic 5-stage pipeline described in Section 4.5. Problems in this exercise refer to the following sequence of instructions:

	Instruction Sequence

a.	`SW R16,-100(R6)`
	`LW R4,8(R16)`
	`ADD R5,R4,R4`

4.13.1 [10] <4.5> Indicate dependences and their type.

4.13.2 [10] <4.5> Assume there is no forwarding in this pipelined processor. Indicate hazards and add `NOP` instructions to eliminate them.

4.13.3 [10] <4.5> Assume there is full forwarding. Indicate hazards and add `NOP` instructions to eliminate them.

	Without Forwarding	With Full Forwarding	With ALU-ALU Forwarding Only
a.	250ps	300ps	290ps

4.13.4 [10] <4.5> What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

4.13.5 [10] <4.5> Add `NOP` instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage).

4.13.6 [10] <4.5> What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?

Exercise 4.17

Problems in this exercise assume that instructions executed by a pipelined processor are broken down as follows:

	ADD	BEQ	LW	SW
a.	40%	30%	25%	5%

4.17.1 [5] <4.6> Assuming there are no stalls and that 60% of all conditional branches are taken, in what percentage of clock cycles does the branch adder in the EX stage generate a value that is actually used?

4.17.2 [5] <4.6> Assuming there are no stalls, how often (percentage of all cycles) do we actually need to use all three register ports (two reads and a write) in the same cycle?

4.17.3 [5] <4.6> Assuming there are no stalls, how often (percentage of all cycles) do we use the data memory?

Each pipeline stage in Figure 4.33 has some latency. Additionally, pipelining introduces registers between stages (Figure 4.35), and each of these adds an additional latency. The remaining problems in this exercise assume the following latencies for logic within each pipeline stage and for each register between two stages:

	IF	ID	EX	MEM	WB	Pipeline Register
a.	200ps	120ps	150ps	190ps	100ps	15ps

4.17.4 [5] <4.6> Assuming there are no stalls, what is the speedup achieved by pipelining a single-cycle datapath?

4.17.5 [10] <4.6> We can convert all load/store instructions into register-based (no offset) and put the memory access in parallel with the ALU. What is the clock cycle time if this is done in the single-cycle and in the pipelined datapath? Assume that the latency of the new EX/MEM stage is equal to the longer of their latencies.

4.17.6 [10] <4.6> The change in 4.17.5 requires many existing `LW/SW` instructions to be converted into two-instruction sequences. If this is needed for 50% of these instructions, what is the overall speedup achieved by changing from the 5-stage pipeline to the 4-stage pipeline where EX and MEM are done in parallel?

Exercise 4.21

This exercise is intended to help you understand the relationship between forwarding, hazard detection, and ISA design. Problems in this exercise refer to the following sequences of instructions, and assume that it is executed on a 5-stage pipelined datapath:

	Instruction sequence

a.	`ADD R5,R2,R1`
	`LW  R3,4(R5)`
	`LW  R2,0(R2)`
	`OR  R3,R5,R3`
	`SW  R3,0(R5)`

4.21.1 [5] <4.7> If there is no forwarding or hazard detection, insert `NOPs` to ensure correct execution.

4.21.2 [10] <4.7> Repeat 4.21.1 but now use `NOPs` only when a hazard cannot be avoided by changing or rearranging these instructions. You can assume register R7 can be used to hold temporary values in your modified code.

4.21.3 [10] <4.7> If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when this code executes?

4.21.4 [20] <4.7> If there is forwarding, for the first five cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in Figure 4.60.

4.21.5 [10] <4.7> If there is no forwarding, what new inputs and output signals do we need for the hazard detection unit in Figure 4.60? Using this instruction sequence as an example, explain why each signal is needed.

4.21.6 [20] <4.7> For the new hazard detection unit from 4.21.5, specify which output signals it asserts in each of the first five cycles during the execution of this code.