

Name (please print): \_\_\_\_\_ Total points: \_\_\_/120

**Instructions**

This is a **CLOSED BOOK** and **CLOSED NOTES** exam. However, you may use calculators, scratch paper, and the green MIPS reference card from your textbook. Ask the instructor if you have any questions. Good luck, and have a good summer!

- (10 points) Intel is considering two different enhancements to their newest i8 line of processors. They only have enough time to implement one of these enhancements before the scheduled release date of the processor.

The instruction set of their original processor can be divided into three different types of instructions having the following CPIs:

Instruction type	CPI
A	1.4
B	2.4
C	2

The benchmarks that are used to evaluate processor performance consist of the following instruction mixture:

Instruction type	Percentage
A	30%
B	60%
C	10%

Intel must choose one of the following options:

- decrease the CPI of instruction type A by a factor of 15 (divide by 15), **or**
- decrease the CPI of instruction type B by a factor of 2 (divide by 2).

These enhancements will not affect the number of instructions or the clock rate. Which enhancement is preferable? You must justify your answer numerically (with calculations).

$$\text{option 1 cpi} = 1.4/15 * .30 + 2.4 * .60 + 2 * .10 = 1.668$$

$$\text{option 2 cpi} = 1.4 * .30 + 2.4/2 * .60 + 2 * .10 = 1.340 \text{ (better)}$$

2. Consider the following program, which converts a ASCII-encoded string representation of a decimal number into binary.

```

.data
str: .asciiz "891"
.text
main: li $t0,0           # initialize index register to 0
      li $s0,0           # initialize running sum to 0
loop: lb $s1, str($t0)   # load a character from the string
      beqz $s1, exit     # if we read the null character, then see ya!

      sll $t2, $s0, 3    # this instruction, along with the next two
      sll $s0, $s0, 1    # instructions are used for:
      add $s0, $s0, $t2  # multiplying accumulator ($s0) by ten

      addi $s1, $s1, -48 # convert from ASCII to binary
      add $s0, $s0, $s1  # accumulate!
      addi $t0, $t0, 1   # increment index
      j loop            # here we go again!
exit: jr $31           # end proggie

```

- a. (10 points) In the program, fill in the blank comment line to describe what computation the program is performing through the use of the corresponding instruction sequence (two **sll** instructions followed by an **add** instruction).
- b. (10 points) Suppose we need to run this program on an architecture that has a branch delay slot. Would we need to make any changes to the program to ensure that it behaves properly? Why or why not? If so, what change(s) must be made?
- no – the shift after the branch won't hurt anything if the branch is taken
- c. (10 points) In the program code, show where all the data dependences exist, and indicate if any pipeline stalls would need to be inserted to deal with any of these dependencies (assuming the 5-stage MIPS pipeline).
- d. (10 points) Write a short sequence of instructions to be inserted before the first instruction (**li \$t0,0**), as well as a short sequence of instructions that could be inserted immediately before the return instruction (**jr \$31**) which would save and later restore the caller's **\$s** registers using the program's stack. You only need to save the **\$s** registers that are changed by the program.

“Header” code for saving the caller’s registers to the stack:	“Footer” code to restoring the caller’s registers from the stack:
<pre> addi \$sp,\$sp,-8 sw \$s0,0(\$sp) sw \$s1,4(\$sp) </pre>	<pre> lw \$s0,0(\$sp) lw \$s1,4(\$sp) addi \$sp,\$sp,8 </pre>

3. This problem compares the performance of the **single-cycle** MIPS architecture, the **multi-cycle** MIPS architecture, and the **pipelined** MIPS architecture.

Assume it requires **10 ns** to perform any of the following operations: main memory access, register file access, and arithmetic operation. Assume the delay for multiplexors, registers (i.e. setup/hold time), and lookup tables is negligible.

- a. (10 points) What is the minimum **clock period** for each of the three implementations?

single=>50 ns, multi=>10 ns, pipelined=>10 ns

- b. (10 points) What is the CPI for each of the implementations, assuming a program execution with the following instruction mix:

Instruction	Execution Frequency
r-type arithmetic, logical, comparison	60%
branch	15%
load	15%
store	10%

Assume the pipelined CPU performs forwarding, that the compiler schedules the code to avoid load hazards, branches have a fixed 3-cycle latency (i.e. requires 2 trailing no-ops), and you may disregard the time required to fill the pipeline.

single=>1,  
 multi=.6\*4+.15\*3+.15\*5+.10\*4=4,  
 pipelined=.85\*1+.15\*3=1.3

- c. (10 points) What is the speedup of the pipelined processor relative to the single-cycle processor ? What is the speedup of the pipelined processor relative to the multi-cycle processor?

pipe vs. single =  $(50 * 1) / (10 * 1.3) = 3.85$   
 pipe vs. multi =  $(10 * 4) / (10 * 1.3) = 3.08$

4. (10 points) Suppose we want to improve the **MISS RATE** of a cache. List three different design enhancements that we could *reasonably* consider to accomplish this. Also, describe any possible side-effects, in terms of possible reductions in other aspects of cache performance, that could result from implementing each of these enhancements (and why).

more associativity => higher hit time (comparing more tags), higher miss penalty (replacement decisions)

more lines => higher hit time (bigger mux)

wider lines (bigger blocks) => higher miss penalty (more data to bring in)

5. (10 points) What is the minimum number of bits that are required to represent the decimal value 11.0625 in binary **without introducing any rounding error**? Provide the minimum number of bits only, you do not need to consider any particular representation format (i.e. you do not need to use the IEEE floating-point format, just the raw value in base-2).

.0625 => .1250 => .25 => .5 => 1 → .0001 in base 2

11 → 1011 in base base

11.0625 → 1011.0001 so 8 bits!

6. (10 points) Assume there's a 2-way set associative cache with LRU replacement where memory addresses are interpreted in the following way:

tag	index	word offset	byte offset
	2 bits	2 bits	2 bits

Fill in the following table to simulate the behavior of this cache, based on the sequence of memory references shown in column 1 of the table.

Byte address	Index	Tag	Hit or miss?
541	1	8	miss line 1, got 8
662	1	10	miss line 1, got 8, 10
533	1	8	hit
607	1	9	miss line 1, got 9,10
542	1	8	miss line 1, got 9, 8
656	1	10	miss line 1, got 10, 8

7. (10 points) It's 1997 and you're a graduate student at Stanford named Larry Page. You're trying to build a new Internet search engine and your strategy is to optimize its performance by ensuring that during a search, neither the CPU nor its disk array are idle during a search.

The search database is logically divided into 100 MB contiguous (sequential) blocks. After the first block is read, the engine reads subsequent blocks while using the CPU to search the previously read block. It takes 100 ms for the CPU to search each block.

You decide to use disks that rotate at 170 revolutions/sec (about 10,000 RPM), have an average seek time of 8 ms, have a transfer rate of 50 MB/sec, and have a controller overhead of 2 ms. How many disks do you need in your disk array? You do not need to include check (redundant) disks.

$$100 \text{ ms} = ((1/170 \text{ rev/s}) * (1/2) \text{ rev}) + 8 \text{ ms} + ((100 \text{ MB}) / (50 \text{ MB/s})) * n$$

$$100 \text{ ms} = 3 \text{ ms} + 8 \text{ ms} + 2 * n$$

$$89 \text{ ms} = 2 * n$$

$$45 = n$$