Computer Architecture Comprehensive Exam (Answers)

Exam Instructions
Answer each of the questions included in the exam. Write all of your answers directly on the examination paper, including any work that you wish to be considered for partial credit. The examination is open-book, and you may make use of the text, handouts, your own course notes, and a calculator.

On equations: Wherever possible, make sure to include the equation, the equation rewritten with the numerical values, and the final solution. Partial credit will be weighted appropriately for each component of the problem, and providing more information improves the likelihood that partial credit can be awarded.

On writing code: Unless otherwise stated, you are free to use any of the assembly instructions listed in the Appendix at the back of the book, including pseudoinstructions. You do not need to optimize your MIPS code unless specifically instructed to do so.

On time: You will have one hour to complete this exam. Budget your time and try to leave some time at the end to go over your work. The point weightings correspond roughly to the time each problem is expected to take.

THE STANFORD UNIVERSITY HONOR CODE

The Honor Code is an undertaking of the students, individually and collectively:

1. that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
2. that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

I acknowledge and accept the Honor Code.

Magic Number: _____

		Score	*Grader*
1. Short Answer	(15)	_____	_____
2. Pipelining	(15)	_____	_____
3. Memory Heirarchy	(15)	_____	_____
4. Cache Math	(5)	_____	_____
5. MIPS Assembly	(10)	_____	_____
TOTAL	(60)	_____	

Problem 1: Short Answer (15 points)

Please provide short, concise answers.

(1) *(2 points)* Your company, Acme Corp., is deciding between two computer systems to deploy its new killer Road Runner Tracking application. Ben Bitdiddle says that his company's system has the better performance because it has the higher clock speed and the higher IPC. Explain why his logic is flawed.

To evaluate performance, you should use execution time for the applications of interest. CPI, IPC, Ghz, FLOPS, IPS, and benchmarks all can imply that processor A is faster than processor B when processor B actually executes the user's application faster.

(2) *(2 points)* Some RISC architectures require the compiler (or assembly programmer) to guarantee that a register not be accessed for a given number of cycles after it is loaded from memory. Give an advantage and a disadvantage of this design choice.

The advantages stem from a simplification of the pipeline logic, which can improve power consumption, area consumption, and maximum speed. The disadvantages are an increase in compiler complexity and a reduced ability to evolve the processor implementation independently of the instruction set (newer generations of chip might have a different pipeline depth and hence a different optimal number of delay cycles).

(3) *(2 points)* Ben Bitdiddle is writing an optimizing compiler for an architecture that supports virtual memory. He notices that his target processor can execute an unaligned 32 bit load more quickly than an 8 bit load. Is it okay for his compiler to emit 32 bit loads and then ignore the extra 24 bits? Why or why not?

No. If a load occurs near the end of a page the extra three bytes may trigger a page fault.

(4) *(3 points)* Briefly describe the data access pattern of an application for which an LRU (least recently used) cache replacement policy performs worse than a random replacement policy.

One pathological case for LRU would be an application that makes multiple sequential passes through a data set that is slightly too large to fit in the cache. The LRU policy will result in a 0% cache hit rate, while under random replacement the hit rate will be high.

(5) *(3 points)* Briefly describe the data access pattern of an application for which a cache with a random replacement policy performs worse than an LRU replacement policy.

A case where LRU would be expected to outperform random replacement is a randomly-accessed tree structure that is too large to fit in the cache. The LRU policy will do a better job of keeping the root of the tree in the cache and those nodes of the tree are accessed more often.

(6) *(3 points)* Briefly give two ways in which loop unrolling can increase performance and one in which it can decrease performance.

Performance can be increased by:

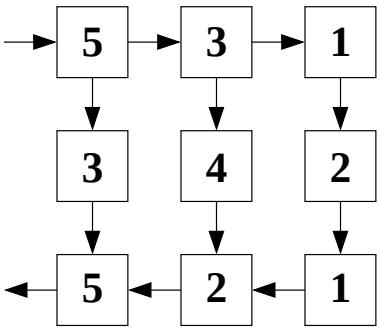
- *Fewer loop conditional evaluations*
- *Fewer branches/jumps*
- *Opportunities to reorder instructions across iterations*
- *Opportunities to merge loads and stores across iterations*

Performance can be decreased by:

- *Increased pressure on the I-cache*
- *Large branch/jump distances may require slower instructions*

Problem 2: Pipelining (15 points)

The National Security Agency has designed a new encryption device called the Conundrum, composed of nine combinational modules connected as shown in the diagram below:

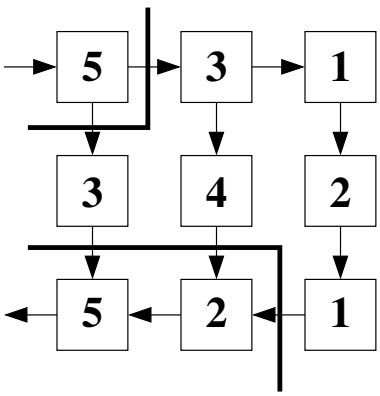


The device takes an integer value X and computes an encrypted version C(X). In the diagram above each combinational component is marked with its propagation delay in microseconds; contamination delays are zero for each component. Assume an ideal (zero-delay) register on both the input and the output of the device.

(1) *(4 points)* What is the latency and throughput of the Conundrum device?

latency (μs) __19__ throughput ($1/\mu s$) __1/19__

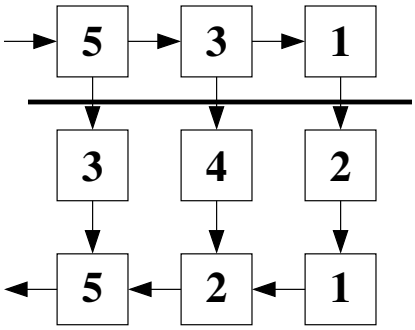
(2) *(9 points)* The NSA needs to produce a version of the Conundrum device that has a throughput larger than 1/15 but wants the implementation with smallest latency (cycle time * pipeline depth) that meets the throughput constraint. Using the diagram below indicate the locations of ideal (zero-delay) registers to create a pipelined implementation that meets these goals. You may use as many registers as necessary. We'll give partial credit for designs with throughput larger than 1/15; full credit for achieving the smallest possible latency. (Extra copies of this diagram can be found on the following page, but only solutions written on the diagram below will be graded).



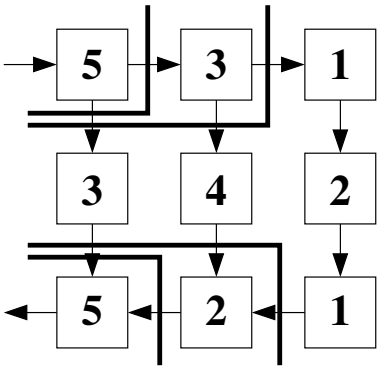
(3) *(2 points)* What is the latency of your pipelined implementation?

latency (μs) __21__ throughput ($1/\mu s$) __1/7__

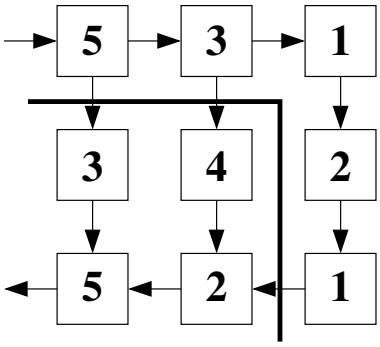
Some valid pipelines with throughput larger than 1/15, but with sub-optimal latency, are given here.



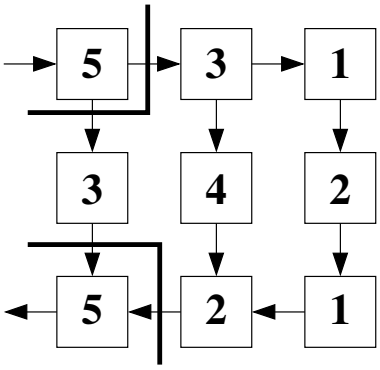
latency: 22
throughput: 1/11



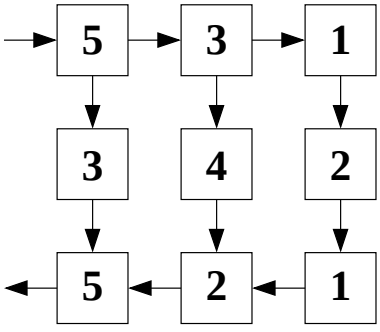
latency: 25
throughput: 1/5
(best possible throughput)

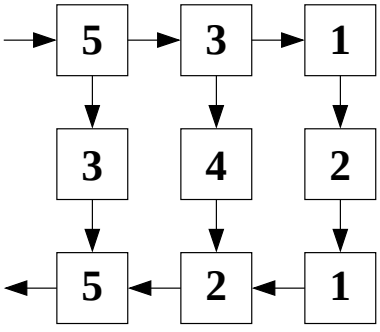


latency: 24
throughput: 1/12



latency: 27
throughput: 1/9





(These diagrams are for scratch work; no solution written here will be graded. Record your solution on the previous page.)

Problem 3: Memory Heirarchy (15 points)

Assume you have a 1 GHz processor with 2-levels of cache and DRAM main memory. The first level cache is split for instructions and data. The system does not use early restart or critical word first, i.e. data blocks must be completely transfered before their results are available. The memory system has the following parameters (Note that here 1KB = 1024 bytes):

	Hit Time	Miss Rate	Block Size
Level-1 cache	1 cycle	6% for data 2% for instruction	32 bytes
Level-2 cache	12 cycles + (1 cycle per 64 bits)	2%	256 bytes
DRAM	70ns + (10ns per 8 bytes)	–	–

The system includes a TLB with a miss rate of 0.5% for data and never incurs a TLB miss for instructions. The TLB miss penalty is 300 cycles and TLB hits take place in parallel with level-1 cache access. All caches in the system are virtually indexed and physically tagged. Assume that the system never swaps memory out to disk.

(1) *(5 points)* What is the average memory access time (AMAT) in clock cycles for instructions?

First you need to calculate the AMAT at every level, taking into account the block size at the highest level and the miss rate and miss penalty at the lower level. Note that at 1 GHZ, 1ns = 1 cycle.

$$
\begin{aligned}
AMAT_{DRAM} &= 70ns + 10ns * (256/8) = 390ns = 390 \; cycles \\
AMAT_{L2} &= hit\ time + miss\ rate * miss\ penalty = (12 + (32/8)) + 0.02 * 390 = 23.8 \; cycles \\
AMAT_{L1} &= hit\ time + miss\ rate * miss\ penalty = (1 + 0.02 * 23.8) = 1.476 \; cycles
\end{aligned}
$$

Thus the overall AMAT is ***1.476 cycles****.*

(2) *(5 points)* What is the AMAT in clock cycles for <u>data</u>? Assume all data accesses are loads.

Only two things change when considering data accesses: the different L1 miss rate and the TLB misses. Note that since the TLB is accessed in parallel with the L1, its hit time is 0 cycles.

$$\begin{aligned} AMAT_{L1} &= hit\ time + miss\ rate * miss\ penalty = (1 + 0.06 * 23.8) = 2.428\ cycles \\ AMAT_{TLB} &= hit\ time + miss\ rate * miss\ penalty = (0 + 0.005 * 300) = 1.5\ cycles \end{aligned}$$

Thus the overall AMAT is $AMAT_{L1} + AMAT_{TLB} =$ ***3.928 cycles***.

(3) *(5 points)* Suppose that we measure the following instruction mix for a program:

Loads: 25%, Stores: 15%, Integer: 30%, Floating-Point: 20%, Branches: 10%

Assume that the processor is using the 5-stage pipeline (base CPI of 1.0). Data hazards cause an average penalty of 0.9 cycles for floating point operations. Integer operations run at maximum throughput. The processor uses the predict-branch-not-taken technique, which turns out to be correct for 80% of the branches. For the remaining branches, there is a 1 cycle stall. What is the average CPI of this program including memory misses (from questions a and b)? Assume that stores have the same AMAT as loads.

CPI = (base CPI) + (CPI due to branch hazards) + (CPI due to FP hazards) + (CPI due to instruction accesses) + (CPI due to data accesses)
We look at each of these in turn:

component	*explanation*	*value*
base	*Given in problem*	*1*
branch	*branch % * misprediction % * misprediction penalty*	$0.1 * 0.2 * 1 = .02$
FP	*FP % * avg penalty*	$0.2 * 0.9 = .18$
instructions	*AMAT for instructions minus 1 to take out the 1 cycle of L1 cache hit that is included in the base CPI*	$1.476 - 1 = 0.476$
data	*(load % + store %) * AMAT for data adjusted as above*	$(0.25 + 0.15) * (3.928 - 1) = 1.171$

This gives us a total CPI of ***2.847***

Problem 4: Cache Math (5 points)

Answers to this question may be in the form of a bare number (decimal or hex), 2^n, or nk (= $n*2^{10} = n*1024$), for example, 65536, 0x10000, 2^{16}, or 64k are all equivalent and acceptable answers.

Consider a 256kb 4-way set associative cache with 256 byte cache lines for a processor that uses 64-bit data words and 48-bit byte addresses. Assume the variable x, of type uint64_t, is stored in memory at location 0x4A85_B413_A518.

(1) *(2 points)* Fill in the bit ranges in the following diagram. Bit ranges should be inclusive. For example, if a field uses bits 0, 1, 2, and 3, label it 3:0.

64-bit word / 8 bits per byte = 8 bytes per word ⇒ 3 bits to select byte in word
256 byte line / 8 bytes per word = 32 words per line ⇒ 5 bits to select word in line
256k cache / 256 bytes per line / 4 ways per set = 256 sets ⇒ 8 bits to select set
The rest of the bits are the tag ⇒ 32 bit tag.

47 : 16	*15 : 8*	*7 : 3*	*2 : 0*
tag	set	word in line	byte in word

(2) *(3 points)* Assume that x is present in the cache, and char* ptr = 0x4A85_B400_0000. Determine if the following accesses will cause a cache miss. For each access circle MISS if it must cause a miss, HIT if it will never cause a miss, or NOT ENOUGH INFO.

*(ptr + 0x11_A538)	MISS	HIT	**(NOT ENOUGH INFO)**
*(ptr + 0x13_A588)	MISS	**(HIT)**	NOT ENOUGH INFO
*(ptr + 0x13_0218)	MISS	HIT	**(NOT ENOUGH INFO)**

**(ptr + 0x11_A538) is in the same set but has a different tag, it may or may not be in another way in that set. *(ptr + 0x13_A588) is in same the line as x (it is in the same set and has the same tag), and since x in present in the cache, it must be also. *(ptr + 0x13_0218) has the same tag, but is in a different set, so it may or may not be in the cache.*

Problem 5: MIPS Assembly (10 points)

Here is mips assembly for a function with the signature (in C):

```
int f(char* a, char* b);
```

Remember, in a MIPS function call, ra contains the return address, a0 and a1 contain the first and second arguments, respectively, and upon return v0 contains the return value. The s# registers must be preserved across procedure calls. Also remember that standard MIPS has a branch delay slot.

```
00000000 <f>:
   0:   lb      v1,0(a0)
   4:   lb      v0,0(a1)
   8:   addiu   a0,a0,1
   c:   subu    v0,v1,v0
  10:   bnez    v0,20 <f+0x20>
  14:   addiu   a1,a1,1
  18:   bnez    v1,0 <f>
  1c:   nop
  20:   jr      ra
  24:   nop
```

(1) *(7 points)* Explain briefly, in english, what this function does.

The function compares the two strings a and b. It returns an integer less than, equal to, or greater than zero if a is found, respectively, to be less than, to match, or be greater than b. In other words, f() is strcmp(). Note that the comparison of characters is done using 8-bit signed arithmetic.

(2) *(3 points)* Give one optimization that can be performed on the assembly code.

The instruction at location 0x8 (addiu a0,a0,1) can be moved into the branch delay slot at location 0x1c. Note that the instruction at 0x14 is already in a branch delay slot, so moving it to the other branch delay slot does not improve the code.