

University of Michigan

EECS 370
Winter 2002

Final Exam

Instructions

1. This is an open books/notes exam. No computers may be used.
2. It comprises 20 multiple-choice questions, each worth 5 points.
3. Each question has only one correct answer.
4. For each question, indicate your answer on the answer sheet provided.
5. You have 2 hours to complete the exam.
6. Please make sure you enter your name in the space below AND on the answer sheet.
7. By signing below, you certify that your conduct throughout the exam has been in accordance with the College of Engineering Honor Code.
8. At the end of the exam, please turn in this exam booklet AND your answer sheet.

Name **Pat Answers**

Signature **Pat Answers**

Overview and Assembly Programming

1. A(n) _____ is used to position code and data from distinct program phases in a computer's memory for systems without VM.
 - a. Assembler
 - b. Overlay**
 - c. Linker
 - d. Program Heap
 - e. Compiler

2. If a register is callee-saved, which of the following is true:
 - a. When a called function returns to its caller, the value of the register must be the same as when it was called.**
 - b. All functions must save the register on the stack at the beginning of the function and load it back at the end of the function.
 - c. a and b
 - d. b, but only for non-leaf functions
 - e. None of the above

Note: Answer "b" seems correct, but it is only required when the callee uses the given register. If the callee does not use the register, but calls a function that does, it is only the responsibility of that 2nd called function to save the register.

3. The exponent of the number 4.125 when represented in IEEE 32-bit floating point format is:
 - a. 00000000
 - b. 00000001
 - c. 10000000
 - d. 10000001**
 - e. 10000010

4. Consider the following sequence of LC2k2 instructions. The *final values* of registers r4, r5 are 55 and -1, respectively.

```
      add      1 1 2
      nand     2 2 2
      add      1 2 3
      add      1 2 4
loop   add      5 1 5
      beq      5 1 done
      add      3 2 3
      add      4 3 4
      beq      0 0 loop
done   halt
```

What was the initial value of register r5?

- a. 8
- b. 9
- c. 10**
- d. 11
- e. None of the above

Note: The key observation is to see that the value of register 5 at the end of the loop is equal to the initial value of register 1 (since 1 never changes in the loop).

$$-1 + -1 = -2$$

$$\text{nand}(-2, -2) = 1 \quad \text{-- not sure? build a truth table}$$

$$-1 + 1 = 0 \quad \text{-- twice}$$

The body of the loop sums the first n integers. When $n = 10$, the sum = 55.

Processor Core

5. Assume the following CPIs for a machine and the observed frequencies of instructions for two benchmarks.

Instruction Class	Machine CPI	Benchmark A	Benchmark B
Arithmetic	1.0	44%	50%
Data Transfer	1.4	35%	25%
Branch	1.7	19%	21%
Jump and Link	1.2	2%	4%

These two programs have the same number of instructions, and execute with the same frequency. What is the combined CPI for these benchmarks on this machine? Round to get the nearest number.

- a. 1.1
- b. 1.2
- c. 1.3**
- d. 1.5
- e. 1.6

Note: This is just a straightforward CPI calculation. You need to compute the average of the instruction usage (for arithmetic, it's 47%), but then it's just:

$$\begin{aligned} \text{CPI} &= \text{arith} * 1.0 + \text{data} * 1.4 + \text{branch} * 1.7 + \text{jump} * 1.2 \\ &= 1.3 \end{aligned}$$

6. Given Project 3 assumptions but with no forwarding, what is the total number of the following code sequence:

```
nand 0 0 1
lw    0 2 5
lw    0 3 6
add   1 1 4
add   1 3 5
add   4 4 6
```

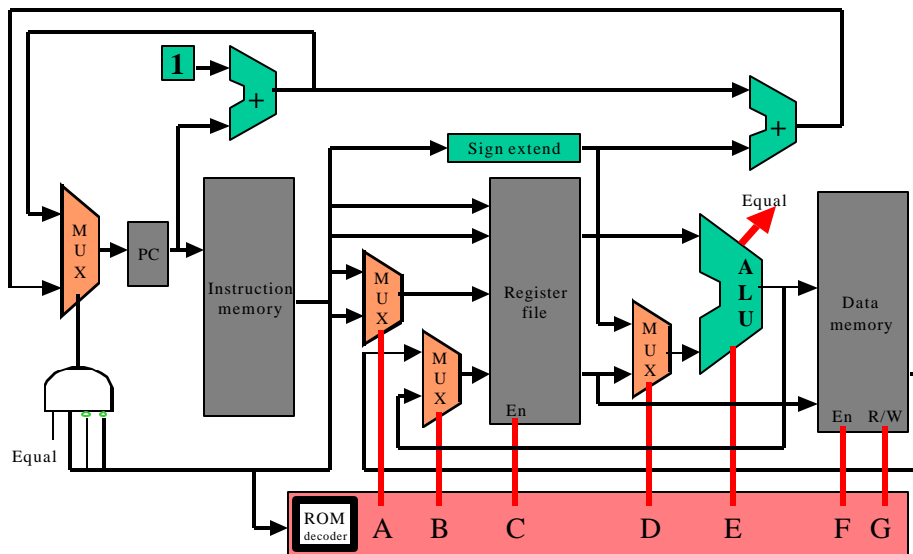
- a. 6
- b. 10
- c. 12

d. 13

e. 15

Note: The assumption implies that you need to stall the processor when there are data dependency hazards in the code. There are 2 data dependencies requiring 3 stall cycles. The code length is 6, stalls are 3 and the pipeline drain is 4.

7. Consider the following single-cycle datapath



What is its cycle time, given the following propagation delays?

- memory access = 8 ns
- register read/write = 1ns
- ALU/Adder operation = 4 ns

a. 8 ns

b. 13 ns

c. 21 ns

d. 22 ns

e. none of the above

Caches and Virtual Memory

8. Let x be a single dimension integer array with 50 four (4) byte elements. Consider the following set of references:

X[17]
X[9]
X[16]
X[14]
X[6]
X[21]
X[3]
X[7]
X[21]
X[30]
X[9]
X[6]
X[28]
X[17]

Assume the processor has split data and instruction caches, and that the data cache is an 8-set direct mapped cache with a block size of 4 bytes (1 word). Define **H** as a *hit*, **M** as a *miss*, and **E** as an *eviction* (a miss forcing a block replacement). Which sequence characterizes the references above?

- a. MEMEMMHMHEHEME
 - b. MEMEMMMHEHEME**
 - c. MMMEMMMHEHHME
 - d. MEMEMMHHEMHEME
 - e. None of the above
9. Use the same sequence as defined in the question above, but now allow for a 2-way set associative cache with 64 bytes of data, a block size of 8 bytes, and an LRU replacement policy.
- a. MMMHEMMEEHHEME
 - b. MMMEMEHEHHME
 - c. MMMMEMMMHEHHME
 - d. MEMEMMMHEHEME
 - e. None of the above**

Note: The access to X[16] is a hit due to the 8 byte block.

10. Given the following cache description

- Cache Data Size = 64 bytes
- Associativity = 2 way set
- Block size = 16 bytes
- Write policy = write back
- 16 bit addresses, byte addressable
- Random replacement policy

What is the smallest storage requirement in *bits* to implement this cache?

- a. 548
- b. 556
- c. 560
- d. 564**
- e. 580

11. You have been asked to build a byte addressable, direct-mapped instruction cache with eight bytes per set for a 32-bit virtually addressed RISC processor. You are given eighteen (18) 32K x 8-bit SRAMs (each part has 32,768 independently addressable bytes). What is the *set count* for the largest I-cache you can build? You need to incorporate all required data and control information into your design. You may not need all the parts.

- a. 8K
- b. 16K
- c. 32K**
- d. 64K
- e. 128K

Note: The block size for each set of the cache requires 8 SRAMs (32bits = 4 bytes, 8 bytes per block). The question is: is the set count 32K or 64K (since the latter would require 16 SRAMs for data). You need to establish the tag size and validity requirements. The *Set index* and *block offset* requires [15,3] or [16,3] bits (depending upon 32K or 64K cache). This leaves 14 or 13 bits for the tag and 1 for the validity bit, both of which comfortably fit into 16 bits. So, 10 parts are required for each 32K sets.

However, you only have 18 parts, so you can only implement a 32K cache. You'll have 8 parts left over.

12. You have inherited the role of architect for a machine with a CPI of 4.0 and a miss penalty of 100 cycles. Your company's primary customer has a code base that exhibits 30% memory accesses. Your current system exhibits a 4% instruction cache miss rate and an 8% data cache miss rate over your customer's code.

You have been instructed to improve the caches' performance. You have a choice of either doubling your level 1 data cache, or implementing a second level unified cache. Although doubling your data cache cuts the data miss rate in half, but it increases the cycle time by 5%. The second level cache has a hit cost of 25 cycles, but 90% of all 1st level misses will hit in the 2nd level cache.

Which choice do you recommend? (*Hint: establish a new system miss penalty based upon the performance of the L2 cache.*)

- a. Doubling D-Cache is better by at least 20%
- b. Doubling D-Cache is better by no more than 10%
- c. Second level Unified Cache is better by at least 20%**
- d. Second level Unified Cache is better by no more than 10%
- e. Neither significantly improves current configuration

Note: The AMAT (average memory access time) is:

$$\text{hitCost} + \text{missRate} * \text{missPenalty}$$

so the AMAT for the base system is:

$$\text{baseAMAT} = 4 + (.04 + .08 * .3) * 100 = 10.4$$

Doubling your D-cache gives you

$$\begin{aligned} \text{DbIAMAT} &= (1.05 * 4) + (.04 + (.08 * .5 * .3)) * 100 \\ &= 9.66 \end{aligned}$$

With the L2 cache, the L1 miss penalty is:

$$\text{L1_Miss} = 25 + .1 * 100 = 35$$

So the overall Level 2 system AMAT is:

$$\text{L2AMAT} = 4 + (.04 * .08 * .3) * 35 = 6.24$$

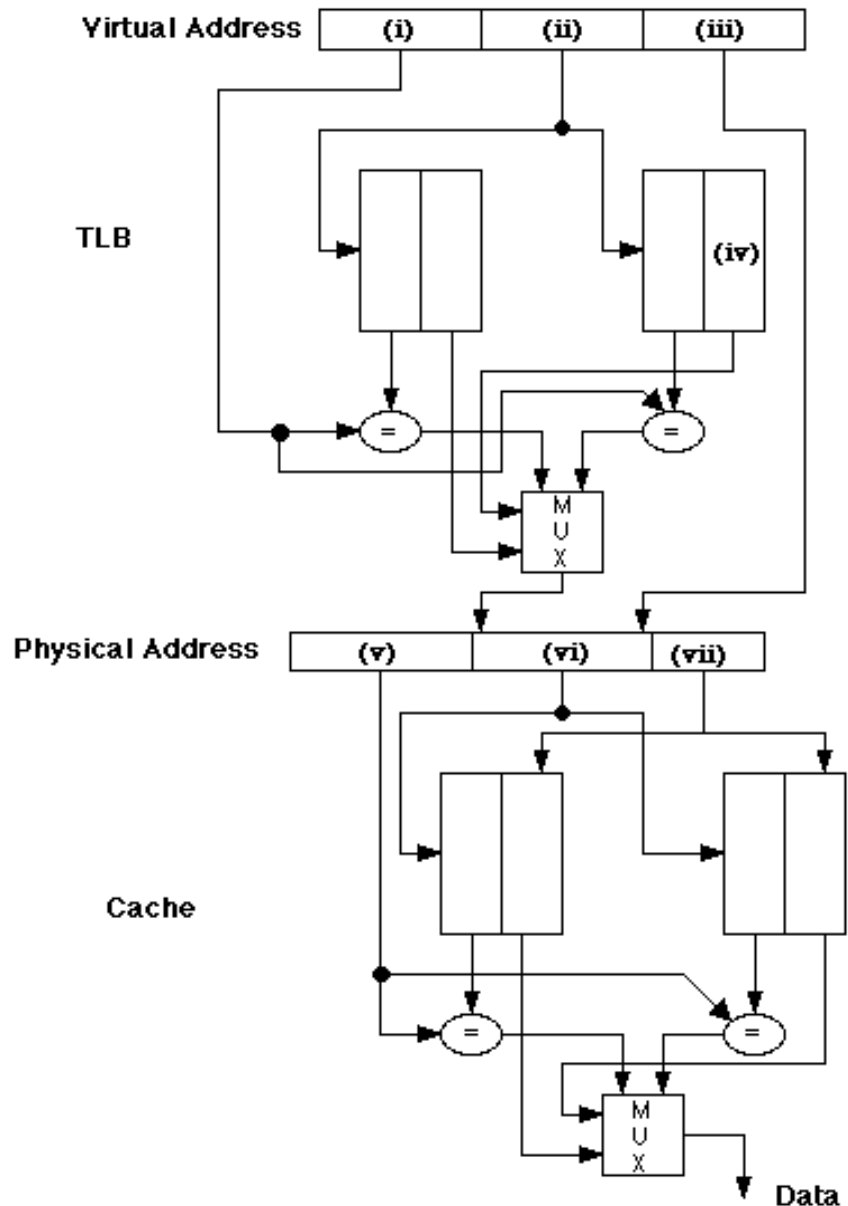
The speedup is:

$$\text{DbIAMAT/L2AMAT} = 9.66/6.24 = 1.548$$

Which is a 55% improvement

For questions 13, 14 and 15, consider byte-addressable memory system with the characteristics outlined in this table.

Virtual address	32 bits
TLB	2-way set associative
Total entries in TLB	512
Page size	2KB
Physical address	36 bits
Total cache (unified) data size	256KB
Block (Line) size	32 bytes
Cache	2-way set associative



13. What are the bit field widths for fields [i, ii, iii] respectively?

- a. 14, 7, 11
- b. 13, 8, 11**
- c. 12, 9, 11
- d. 13, 7, 12
- e. None of the above

Note: The system page size is $2K=2^{11}$.
There are 256 sets = 2^8 in the TLB.
The TLB Tag size must be $32 - (11 + 8) = 13$ bits.

14. What is the bit field width for field “iv”?

- a. 16
- b. 21
- c. 23
- d. 26
- e. None of the above**

Note: The *Physical* address space is 36 bits.
Since the page size is $2K = 2^{11}$,
the data value retrieved from the TLB must be $36 - 11 = 25$ bits.

15. What are the bit field widths for fields [v, vi, vii] respectively?

- a. 15, 12, 5
- b. 19, 11, 6
- c. 19, 12, 5**
- d. 20, 11, 5
- e. None of the above

Note: The cache block size is $32 = 2^5$ bytes.
The data size per set is $128K = 2^{17}$, so the number of sets is $2^{(17-5=12)}$.
The tag size must be $36 - 17 = 19$.

16. Assume you have a machine with a direct mapped data cache with 1K sets and four (4) words per block. Why would a compiler (or programmer) perform the following program transformation?

```
int i, j, x[100][20];

/* ----- Before */
for (j=0 ; j < 20 ; j++)
    for (i = 0 ; i < 100 ; i++)
        x[i][j] = x[i][j] + 1;

/* ----- After */
for (i = 0 ; i < 100 ; i++)
    for (j=0 ; j < 20 ; j++)
        x[i][j] = x[i][j] + 1;
```

- a. It improves performance by enhancing spatial locality
- b. It improves performance by enhancing temporal locality
- c. It reduces compulsory misses
- d. It improves performance by enhancing both spatial and temporal locality
- e. There is no benefit in performing this transformation**

Note: The data size of the cache is $1K * 16\text{bytes} = 16K$. The total storage used by the array “x” is 8K. Therefore, after compulsory misses, the entire array fits into the data cache, and all conflict misses are avoided.

17. Which one of the following statements about virtual memory is true?
- a. The OS updates an LRU bit upon every valid page reference.
 - b. A change bit helps support *write through* for VM page references.
 - c. A physically addressed cache can hit even if the page is invalid in the page table.
 - d. The TLB is only updated upon a page fault.
 - e. A paging operation is triggered by a TLB miss.**

Note: The OS cannot be activated for every valid page reference. VM is never supported with *Write Through*. If the page is invalid, neither cache nor TLB can hit. The TLB is updated for every miss to a valid page.

18. Keeping the address size, block size and associativity fixed, what would be the effect upon the TAG of decreasing the number of sets in the cache?

- a. TAG size increases
- b. TAG size decreases
- c. TAG is shifted left
- d. TAG is shifted right
- e. TAG is unaffected

19. Given this LC2K2 code:

```
        lw  0 2 n10
loop1   lw  0 3 p14
loop2   lw  3 4 p14
        add 3 2 3
        beq 0 3 loop1
        beq 0 0 loop2
n10     .fill -1024
p14     .fill 4096
```

Which of the following data caches will miss on more than 75% of the loads? Assuming all caches use an LRU replacement policy. (Note that the program is an infinite loop, so the effect of compulsory misses is eliminated.)

- a. 2-way 4KB cache
- b. 4-way 4KB cache
- c. 8-way 4KB cache
- d. (a) and (b)
- e. (a) (b) and (c)

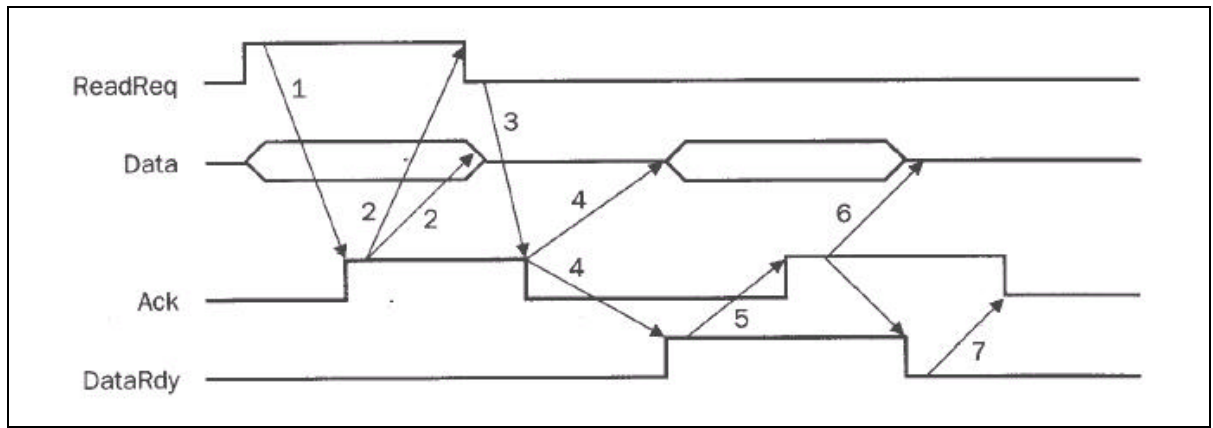
Note: In the steady state, the program generates the reference address stream:

“c”, 4K+c, 3K+c, 2K+c, 1K+c

where “c” is the address of “p14”. These will all generate references to the same set in any of these caches. Only “c” has the capacity to hold all 5 references without evictions.

Bus Activity

20. Consider the following timing diagram for a bus protocol between a data requester and a data provider. (This is figure 8.10 from C.O.D. – the course text.) The figure shows the timing of the signals synchronizing the exchange of addresses and data between the requestor and provider. In this model, *addresses* and *data* are multiplexed on the same data line. Assume that the provider is a memory system, and the requestor is a device wishing to access that memory.



Assume that all signals require 40ns for propagation between sender and receiver (this includes the signal setup time), and that the memory system has a 200ns access time. Memory accesses can be concurrent with bus activities and can be initiated immediately upon receipt of the address. Both sender and receiver have any necessary registers for latching values to and from the bus.

We want to compare an 8-word “burst” mode transfer against a 1-word transfer. In “burst” mode, the memory system posts the new datum from sequential addresses as quickly as it is available for the duration of the burst; in all other ways the protocol operates as above. If we are transferring 256 byte blocks, what is the speedup for burst mode relative to the 1-word mode? Select the value closest to your result.

- a. 8x speedup
- b. 4x speedup
- c. 2x speedup
- d. 1.5x speedup**
- e. 1.15x speedup

Note: We have stated that the memory system can operate in parallel with the bus protocol.

The protocol timing is:

1. Requester sends address & raises *ReadRequest* (40ns)
2. Memory reads address and initiates memory transaction (200ns)
3. Memory sends data and raises *DataReady* (40ns)
4. Requestors raises *ACK* (40ns)

The entire transaction is $(3 \cdot 40 + 200) = 320$ ns. If we do this 8 times, we get

$$8 * 320 = 2560\text{ns.}$$

For the “burst” mode, steps 3&4 can be overlapped with subsequent memory transactions. Therefore, the overhead for these data transmission is only incurred for the final datum in the burst. The cost of this operation is therefore:

$$(3 * 40) + 8 * 200 = 1720 \text{ ns.}$$

The speedup is $2560/1720 = 1.49$

The fact that we are transferring 256 byte blocks is irrelevant.