University of New Brunswick

Computer Science

CS3853: Computer Architecture and Organization

Instructor: Joannah Nanjekye, jnanjeky@unb.ca

Due Date: Aug 02, 2024 – 11:59 PM

ASSIGNMENT 3

Submission instructions:

- Submit a pdf file to the Desire2Learn dropbox

Problem 1. Given a memory of 2GB, 8MB cache, blocks size of 64B show the address format for

(a) direct mapping

Solution: 2 Points

tag	line	word
8	17	6

(b) associative mapping

Solution: 2 Points

tag	line	word
25	0	6

(c) 8 way set associative mapping

Solution: 2 Points

tag	line	word
11	14	6

Problem 2. Describe a cache for which this program will not run well (lots of cache misses). Suggest a way to correct the problem — without changing the cache characteristics.

```
int a[8192], b[8192], c[8192];
int i;
for(i = 0; i < 8192; i++)
        c[i] = a[i] + b[i];
```

Assume that each int is 4 bytes. Assume `a`, `b`, and `c` are allocated successively in memory.

The simplest answer is: it a direct mapping with the cache size of 32K. This way a[0], b[0], and c[0] all map to the same line in in the cache. Any smaller power of 2 will have the same effect. Also, a 2-way set associative cache will have the same effect. Accept reasonable answers along these lines. (5 points) Add something like 100 ints to the size of each array. The number should be at least as big as the cache line (which is unknown, so a reasonable assumption is acceptable) (3 points)

Problem 3. When the processor modifies the cache, main memory must reflect this change. Describe two approaches to keep the cache consistent with main memory.

Solution: 6 Points Look for something reasonable along these lines

- Write through
  - Simplest technique
  - All write operations are made to main memory as well as to the cache
  - The main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck
- Write back
  - Minimizes memory writes
  - Updates are made only in the cache
  - Portions of main memory are invalid and hence accesses by I/O modules can be allowed only through the cache
  - This makes for complex circuitry and a potential bottleneck

Problem 4. Consider the following code:

```
int a[100], b[1000], c[1000];
// get some data for a, b, and c
for(i = 0; i < 100; i++)
       for (j = 0; j < 10; j++)
             a[i] += c[j*100 + i] * b[j*100 + i] ;
```

(a) Comment on the spacial locality in the data.
Solution: 2 Points
   i. The data in a is sequentially accessed, therefore it has excellent spacial locality.
   ii. This is not true for b and c

(b) Comment on the temporal locality in the data.
Solution: 2 Points The code is sequential, therefore it has excellent spacial locality

(c) Comment on the spacial locality in the machine code generated by the program.
Solution: 2 Points The code is sequential, therefore it has excellent spacial locality

(d) Comment on the temporal locality in the code.
Solution: 2 Points for loops provide excellent temporal locality

(e) Write the code that does the same but has better cache hits (Assume that the cache can hold at most 500 integer);
Solution: 4 Points

```
for(i = 0; i < 1000; i++)
       a[i % 100] += b[i] * c[i];
```

Problem 5. Can a direct mapped cache sometimes have a higher hit rate than a fully associative cache with an LRU replacement policy (on the same reference pattern and with the same cache size)? If so, give an example. If not explain why not?

Solution: 6 Points

One of the simplest way to show that this is possible is with 4 cache lines. Assume direct mapping is modulo 4.

Then the pattern 0, 1, 2, 3, 1, 5, 0 would have the following hits (shown in red.)

direct: 0, 1, 2, 3, 1, 5, 0 // the 5 replaces the 1

associative 0, 1, 2, 3, 1, 5, 0 // the 5 replaces the 0

Other examples are possible.

Problem 6. Consider the C program below

```
int main(){
    int v[10000], i, max;
    init(v);   // somehow v is initialized
    max = v[0];
    for(i = 1; i < 10000; i++) {
       if ( v[i] > max )
          max = v[i];
    }
}
```

(a) Estimate the cache hit ratio for the code with direct mapping. The code is 256 bytes this includes init() which is adjacent to the other code. The cache 8K bytes, and a cache line has 128 bytes. You may assume, that the cache is for code only. Show your work! Each instruction is 4 bytes long. You don't have to give an exact number. The number of cache misses may be useful in this case.

Solution: 2 Points Close to 100%. There are only two cache misses.

(b) Estimate the cache hit ratio for the data with direct mapping. An int is 4 bytes, the cache has 8K bytes, and a cache line is 128 bytes. You may assume, that the cache is for data only. Show your work!

You may assume that `i` and `max` are stored in a register (no memory access is required for them).

Solution: 2 Points 31/32 Accept reasonable calculations

(c) Calculate the overall hit ratio. Consider that they are both mapped to the same cache.

You may assume that `i` and `max` are stored in a register (no memory access is required for them).

Solution: 4 Points This is more complicated. The data is $40,000/8,192 = 4.88$ larger than the cache. So the code will collide with data about 5 times. How many misses this will create depends on the actual code. Accept reasonable assumptions. For example, each collision may result in 15 to 30 additional cache misses (other assumptions are possible.)

(d) For the above example, describe the impact of going from a direct mapping to a 2-way set associative, and a 4-way set associative mapping.

Solution: 2 Points

- **2-way set associative** Cache misses between code and data are avoided
- **4-way set associative** Cache misses between code and data are avoided

Problem 7. Consider a processor with a PC-relative branch instruction. Memory addresses are given as 32 bits.

(a) Consider branch instruction is a location A00F1AB0 (in hex). The target address branch is A00F3B08. Determine the displacement in the instruction. All instructions are 4 bytes.
Solution:2 points displacement = A00F3B08 - A00F1AB0 + 4 = 205C

(b) Why do we need to know the instruction size?
Solution:1 point The PC is incremented by the size of the instruction—so we need to know how much that is.

(c) The displacement is given as 16 bit signed integer. What is the range of the branch?
Solution:2 points The range of the branch is $[PC - 2^{15}, PC + 2^{15}] = [PC - 32,768, PC + 32,768]$

(d) Do you think the range of relative branches (as specified above) is adequate? Justify your answer.
Solution:2 points Yes, it is adequate. Branches are taken place only with subprograms (or the main program). With current software engineering practices it will be very rare that a program is larger that 32K—you could still jump from the start to the end of the subprogram.

Problem 8. Consider the CPUs with the following instruction set characteristics:

- **CompA:** Instruction ADD, SUB, MUL, DIV have 3 memory addresses, the MOV instruction has 2 (target, source);
- **CompB:** Instruction ADD, SUB, MUL, DIV have 2 memory addresses[1], the MOV instruction has 2 as well (target, source);
- **CompC:** Instruction ADD, SUB, MUL, DIV have 2 registers (R0, R1, … R15) the MOV instruction has one register and one memory address (target, source);

The size of each instruction is determined as follows:

- Opcode – 1 byte
- Memory Address – 4 bytes
- Register - 4 bits

Write assembly code for each of the CPU's given above to make the following calculation:

$$X = \frac{(A - B)(C - D)}{A + B}$$

(a) How much memory does the code for each program take? Show your work.

(b) How many memory accesses are required? Include the loading of the code (assume that each memory access will get 4 bytes.)

Solution: 5 Points — CompA

Instruction		Memory	Mem. Acc.
SUB	T1,A,B	13	3
SUB	T2,C,D	13	3
MUL	T1,T1,T2	13	3
ADD	T2,A,B	13	3
DIV	X,T1,T2	13	3
	Total	65	15

Memory accesses $= 65/4 + 15 = 31$

Solution: 5 Points — CompB

Instruction		Memory	Mem. Acc.
MOV	X,A	9	2
SUB	X,B	9	3
MOV	T1,C	9	2
SUB	T1,D	9	3
MUL	X,T1	9	2
MOV	T1,A	9	2
ADD	T1,B	9	3
DIV	X,T1	9	3
	Total	81	20

Memory accesses $= 81/4 + 20 = 41$

NOTE: the solution does not have to be exactly the same.

[1]ADD M1, M2 means M1 = M1 + M2

Solution: 5 Points — CompB

Instruction	Memory	Mem. Acc.	Comment
MOV R1,A	5.5	1	$R1 = A$
MOV R2,B	5.5	1	$R2 = B$
MOV R3,C	5.5	1	$R3 = C$
MOV R4,D	5.5	1	$R4 = D$
MOV R5,R1	2	0	$R5 = A$
SUB R5,R2	2	0	$R5 = A - B$
SUB R3,R4	2	0	$R3 = C - D$
MUL R5,R2	2	0	$R5 = (A - B)(C - D)$
ADD R1,R2	2	0	$R1 = A + B$
DIV R5,R1	2	0	$R5 = (A - B)(C - D)/(A + B)$
MOV X,R5	5.5	1	$X = (A - B)(C - D)/(A + B)$
Total	40	5	

Memory accesses $= 40/4 + 5 = 15$

Take one mark off if the program has more than 5 memory accesses.

NOTE: 5.5 bytes for an instruction does not make much sense. Students can use 6 instead of 5.5.

Problem 9. Given a RAID 3 (bit-interleaved parity) with k disks, how well will large block transfers work? How well will it handle a high I/O request rate? Compare the performance to a single disk. Give the formula in terms of k. If the read performance is different from write, then provide separate formulas. What is its capacity (compared to a single disk)?

Solution: 3 Points

- Large block read/write $= k - 1$
- High I/O requests read/write $= 1$
- Capacity $= k - 1$

Problem 10. Given a RAID 5 (block-level distributed parity) with k disks, how well will large block transfers work? How well will it handle a high I/O request rate? Compare the performance to a single disk. Give the formula in terms of k. If the read performance is different from write, then provide separate formulas. What is its capacity (compared to a single disk)?

Solution: 5 Points

- Large block read $= k$ Assume the block distribution is slightly different than the one given in the book. Accept $k - 1$ as well.
- Large block read/write $= k - 1$
- High I/O requests read $= k$
- High I/O requests write $= k/2$
- Capacity $= k - 1$