Chapter 8
The Memory System

8.1. The time it takes for the cell voltage to drop from 1.5 V to 0.9 V is

$$\frac{30 \times 10^{-15} \times (1.5 - 0.9)}{0.25 \times 10^{-12}} = 0.072s$$

Therefore, each row has to be refreshed every 72 ms or less.

8.2. Each column address strobe causes $8 \times 4 = 32$ bytes o be transferred.

(*a*) Latency = 5 clock cycles or 12.5 ns
Total time = $5 + 8 = 13$ clock cycles, or 32.5 ns.

(*b*) A second column strobe is needed to transfer the second burst of 32 bytes. Therefore:

Latency = 5 clock cycles or 12.5 ns
Total time = $5 + 8 + 2 + 8 = 23$ clock cycles, or 57.5 ns.

8.3. A 16M module can be structured as 16 rows, each containing eight 1M $\times$ 4 chips. A 24-bit address is required. Address lines A_{19-0} should be connected to all chips. Address lines A_{23-20} should be connected to a 4-bit decoder to select one of the 16 rows.

8.4. A faster processor chip will result in increased performance. But, the amount of the increase will not be directly proportional to the increase in processor speed, because the cache miss penalty will remain the same if the main memory speed is not improved.

8.5. (a) Main memory address length is 32 bits. The BLOCK field is 3 bits (8 blocks), and the WORD field is 7 bits (128 bytes per block), leaving 22 bits for the TAG field.

(b) Program sections are mapped onto cache blocks as follows:

Program segment	Cache blocks
8 - 52	0
56 - 136	0, 1
140 - 240	1
244 - 1200	1 - 7, 0, 1
1204 - 1504	1, 2, 3

Note that the inner loop (140-240) is entirely within cache block 1. The last segment of the outer loop will fill blocks 2 to 7 and overwrite blocks 0 and 1. Hence, the sequence of reads from the main memory blocks into cache blocks will be

$$\text{Block}: 0,\ \underbrace{1,2,3,4,5,6,7,0,1,}_{\text{Pass 1}}\ \underbrace{0,1,0,1,}_{\text{Pass 2}}\ 0,1,\ \ldots,\ 0,1\ \underbrace{0,1,0,1,}_{\text{Pass 10}}\ 2,3$$

as illustrated in Figure PS8.5. Since both the beginning and the end of the outer loop use blocks 0 and 1 in the cache, they overwrite each other on each pass through the loop. Blocks 2 to 7 remain resident in the cache until the outer loop is completed.

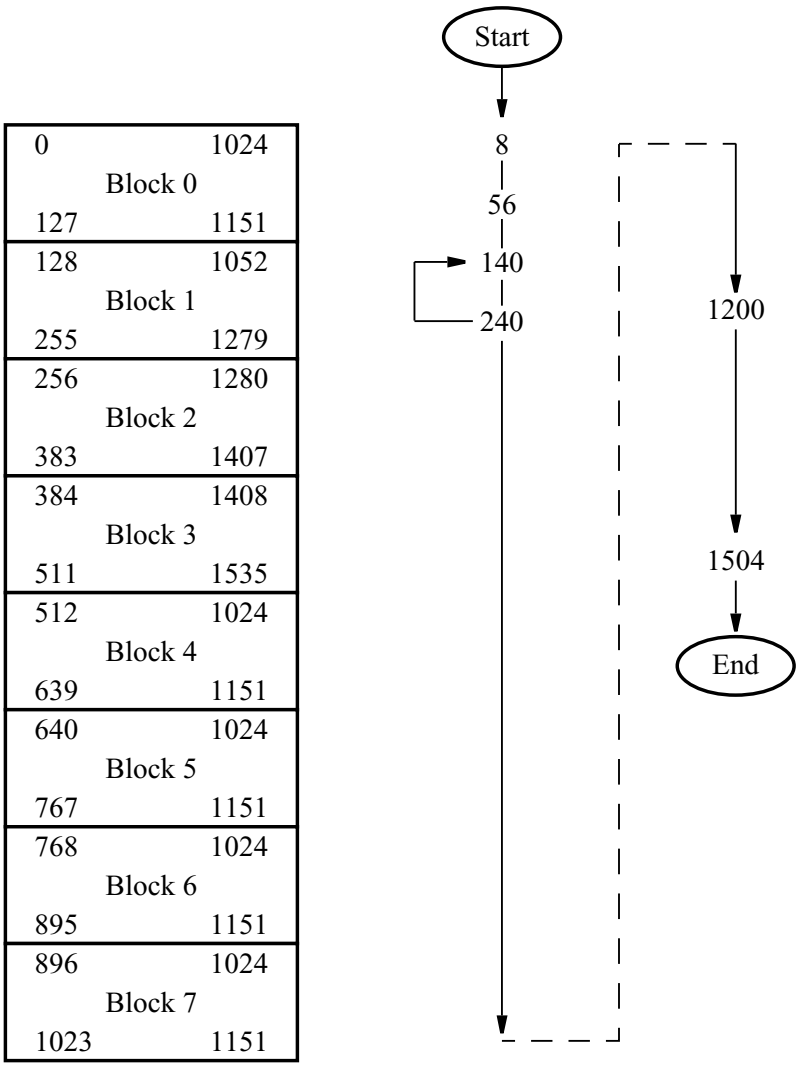

Figure PS8.2. Memory blocks for Problem 8.5.

Miss penalties	$= (1 + 9 + 9 \times 4 + 2) \times 80\,\tau$	$= 3,840\,\tau$
Execution time out of the cache:		
Start section of program	$= (56 - 8)\,\tau$	$= 48\,\tau$
Inner loop	$= 200\,(244 - 140)\,\tau$	$= 20,800\,\tau$
Outer loop − inner loop	$= 10\,((1204 - 56) - (244 - 140))\,\tau$	$= 10,440\,\tau$
End section of program	$= 1508 - 1204$	$= 304\,\tau$
Total execution time		$= 35,432\,\tau$

8.6. In the first pass through the loop, the Add instruction is stored at address 4 in the cache, and its operand (A03C) at address 6. Then the operand is overwritten by the Decrement instruction. The BNE instruction is stored at address 0. In the second pass, the value 05D9 overwrites the BNE instruction, then that instruction is read from the main memory and again stored in location 0. In the third pass, the data operand is stored at address 2, and all three instructions are read from the cache. The contents of the cache, the number of words read from the main memory and from the cache, and the execution time for each pass are as shown in Figure PS8.2.

After pass No.	Cache contents	MM accesses	Cache accesses	Time
1	005E BNE [ ] 005D Add 005D Dec	4	0	40 τ
2	005E BNE [ ] 005D Add 005D Dec	2	2	22 τ
3	005E BNE 00AA 10D7 005D Add 005D Dec	1	3	13 τ
Total		7	5	75 τ

Figure PS8.2. Cach contents for Problem 8.6.

8.7. All three instructions are stored in the cache after the first pass, and they remain in place during subsequent passes. In this case, there is a total of 6 read operations from the main memory and 6 from the cache. Execution time is 66 τ.

This example shows that instructions and data are best stored in separate caches to avoid the data overwriting instructions.

8.8. Each block contains 128 bytes, thus requiring a 7-bit Word field. There are 16 sets, requiring a 4-bit Set field. The remaining 21 bits of the address constitute the tag field.

8.9. After the cache has been filled by main memory blocks 0 to 63 during the first pass, the most-recently-used (MRU) block in each set is that in block location 3. Hence, when blocks 64 to 67 are read they replace blocks 48 to 51 in sets 0, 1, 2 and 3, respectively. Consider first the activity in set 0, which contains blocks 0, 16, 32 and 48 at the end of the first pass. During the second pass, blocks 0, 16 and 32 are found in the cache. Block 48 has to be read from the memory, and it replaces block 32, which is the MRU block in set 0 at that time. Block 64 is still in the cache and does not have to be reloaded. During the next two passes, block 32 replaces block 16, then block 16 replaces block 0. Thus, only one block replacement takes place in this set in each of the second, third and fourth pass.

There are two replacements during the fifth pass: block 0 replaces 64, then 64 replaces 48. Then the pattern repeats, with one block replacement in each of passes 6, 7 and 8, two replacements in pass 9 and one in pass 10. Similar activity takes place in sets 1, 2, and 3. Throughout, there is no contention, hence no replacements, in sets 4 through 15. In total, there are 11 replacements in each of sets 0 to 3. Therefore, the improvement factor is

$$\frac{10 \times 68 \times 10\tau}{1 \times 68 \times 11\tau + 4 \times 11 \times 11\tau + (9 \times 68 - 44) \times 1\tau} = 3.8$$

The MRU replacement algorithm is better than LRU for this case, in which the size of the loop is larger than that of the cache.

8.10. For the first loop, the contents of the cache are as indicated in Figures 8.21 through 8.23. For the second loop, they are as follows.

(a) Direct-mapped cache

Block position	Contents of data cache after pass:					
	$j = 9$	$i = 1$	$i = 3$	$i = 5$	$i = 7$	$i = 9$
0	A(0,8)	A(0,0)	A(0,2)	A(0,4)	A(0,6)	A(0,8)
1						
2						
3						
4	A(0,9)	A(0,1)	A(0,3)	A(0,5)	A(0,7)	A(0,9)
5						
6						
7						

(b) Associative-mapped cache

Block position	Contents of data cache after pass:			
	$j = 9$	$i = 0$	$i = 5$	$i = 9$
0	A(0,8)	A(0,8)	A(0,8)	A(0,6)
1	A(0,9)	A(0,9)	A(0,9)	A(0,7)
2	A(0,2)	A(0,0)	A(0,0)	A(0,8)
3	A(0,3)	A(0,3)	A(0,1)	A(0,9)
4	A(0,4)	A(0,4)	A(0,2)	A(0,2)
5	A(0,5)	A(0,5)	A(0,3)	A(0,3)
6	A(0,6)	A(0,6)	A(0,4)	A(0,4)
7	A(0,7)	A(0,7)	A(0,5)	A(0,5)

(c) Set-associative-mapped cache

	Block position	Contents of data cache after pass:			
		j = 9	*i* = 3	*i* = 7	*i* = 9
Set 0	0	A(0,8)	A(0,2)	A(0,6)	A(0,6)
	1	A(0,9)	A(0,3)	A(0,7)	A(0,7)
	2	A(0,6)	A(0,0)	A(0,4)	A(0,8)
	3	A(0,7)	A(0,1)	A(0,5)	A(0,9)
Set 1	0				
	1				
	2				
	3				

In all 3 cases, all elements are overwritten before they are used in the second loop. This suggests that the LRU algorithm may not lead to good performance if used with arrays that do not fit into the cache. Performance can be improved by introducing some randomness in the replacement algorithm.

8.11. The two least-significant bits of an address, A_{1-0}, specify a byte within a 32-bit word. For a direct-mapped cache, bits A_{4-2} specify the block position in the cache. For a set-associative-mapped cache, bit A_2 specifies the set.

(a) Direct-mapped cache

Block position	Contents of data cache after:			
	Pass 1	Pass 2	Pass 3	Pass 4
0	[200]	[200]	[200]	[200]
1	[204]	[204]	[204]	[204]
2	[208]	[208]	[208]	[208]
3	[24C]	[24C]	[24C]	[24C]
4	[2F0]	[2F0]	[2F0]	[2F0]
5	[2F4]	[2F4]	[2F4]	[2F4]
6	[218]	[218]	[218]	[218]
7	[21C]	[21C]	[21C]	[21C]

Hit rate = 33/48 = 0.69

(*b*) Associative-mapped cache

Block position	Contents of data cache after:			
	Pass 1	Pass 2	Pass 3	Pass 4
0	[200]	[200]	[200]	[200]
1	[204]	[204]	[204]	[204]
2	[24C]	[21C]	[218]	[2F0]
3	[20C]	[24C]	[21C]	[218]
4	[2F4]	[2F4]	[2F4]	[2F4]
5	[2F0]	[20C]	[24C]	[21C]
6	[218]	[2F0]	[20C]	[24C]
7	[21C]	[218]	[2F0]	[20C]

Hit rate = 21/48 = 0.44

(*c*) Set-associative-mapped cache

	Block position	Contents of data cache after:			
		Pass 1	Pass 2	Pass 3	Pass 4
Set 0	0	[200]	[200]	[200]	[200]
	1	[208]	[208]	[208]	[208]
	2	[2F0]	[2F0]	[2F0]	[2F0]
	3	[218]	[218]	[218]	[218]
Set 1	0	[204]	[204]	[204]	[204]
	1	[24C]	[21C]	[24C]	[21C]
	2	[2F4]	[2F4]	[2F4]	[2F4]
	3	[21C]	[24C]	[21C]	[24C]

Hit rate = 30/48 = 0.63

8.12. The two least-significant bits of an address, A_{1-0}, specify a byte within a 32-bit word. For a direct-mapped cache, bits A_{4-3} specify the block position. For a set-associative-mapped cache, bit A_3 specifies the set.

(a) Direct-mapped cache

Block position	Contents of data cache after:			
	Pass 1	Pass 2	Pass 3	Pass 4
0	[200]	[200]	[200]	[200]
	[204]	[204]	[204]	[204]
1	[248]	[248]	[248]	[248]
	[24C]	[24C]	[24C]	[24C]
2	[2F0]	[2F0]	[2F0]	[2F0]
	[2F4]	[2F4]	[2F4]	[2F4]
3	[218]	[218]	[218]	[218]
	[21C]	[21C]	[21C]	[21C]

Hit rate = 37/48 = 0.77

(b) Associative-mapped cache

Block position	Contents of data cache after:			
	Pass 1	Pass 2	Pass 3	Pass 4
0	[200]	[200]	[200]	[200]
	[204]	[204]	[204]	[204]
1	[248]	[218]	[248]	[218]
	[24C]	[21C]	[24C]	[21C]
2	[2F0]	[2F0]	[2F0]	[2F0]
	[2F4]	[2F4]	[2F4]	[2F4]
3	[218]	[248]	[218]	[248]
	[21C]	[24C]	[21C]	[24C]

Hit rate = 34/48 = 0.71

(c) Set-associative-mapped cache

	Block position	Contents of data cache after:			
		Pass 1	Pass 2	Pass 3	Pass 4
Set 0	0	[200]	[200]	[200]	[200]
		[204]	[204]	[204]	[204]
	1	[2F0]	[2F0]	[2F0]	[2F0]
		[2F4]	[2F4]	[2F4]	[2F4]
Set 1	0	[248]	[218]	[248]	[218]
		[24C]	[21C]	[24C]	[21C]
	1	[218]	[248]	[218]	[248]
		[21C]	[24C]	[21C]	[24C]

Hit rate = 34/48 = 0.71

8.13. Larger size

- fewer misses if most of the data in the block are actually used
- wasteful if much of the data are not used before the cache block is ejected from the cache
- Larger miss penalty

Smaller size

- more misses
- smaller miss penalty

8.14 The average access time for a two-level cache is given by:

$$t_{avg} = h_1 C_1 + (1 - h_1)(h_2 C_2 + (1 - h_2)M)$$

For $C_1 = \tau$, $C_2 = 15\tau$, and $M = 100\tau$. The average access times are given in the following table:

h_1	0.90	0.92	0.94	0.96
$h_2 = 0.75$	4.53τ	3.82τ	3.12τ	2.41τ
$h_2 = 0.85$	3.68τ	3.14τ	2.61τ	2.07τ

8.15 For an average access time of 1.5 τ:

Average access time = $(h_1 + (1 - h_1)(0.8 \times 15 + 0.2 \times 100)\tau = 1.5\tau$
Therefore: $h_1 = 0.984$

The value of h_2 needed to achieve the same result can be derived from the equation

Average access time = ($0.96\tau + 0.04(15h_2 + 100(1 - h_2))\tau = 1.56\tau$

which yields $h_2 = 1.018$. This is impossible, because hit rate cannot be greater that 1.

8.16. The analogy is good with respect to:

- relative sizes of the toolbox, truck and shop versus the L1 cache, L2 cache and main memory
- relative access times

- relative frequency of use of tools in the 3 places versus the rate of data accesses in the caches and the main memory

The analogy fails in the following respects:

- At the start of a working day the tools placed into the truck and the toolbox are preselected based on the experience gained on previous jobs. However, when the operating system selects a new program for execution, no relevant data are loaded into the caches before execution begins.
- Most of the tools in the toolbox and the truck are useful in successive jobs, while the data left in a cache by one program are not useful for subsequent programs.
- Tools displaced by the need to use other tools are never thrown away. In the case of a cache, a new block simply overwrites an existing block. The contents of a block being replaced are written to the memory before the block is replaced in the cache only if the block is flagged as dirty.

8.17 As the size of a cache increases, two things happen. The hit rate increases, which means that fewer memory accesses will suffer the miss penalty. At the same time, a larger memory has a longer access time. This means that all accesses to the cache will take longer to service. Thus, as the cache size is increased, the effective memory access time as seen by the processor may initially improve. However, at some point the second factor becomes dominant, and further increases in the size of the cache become detrimental to performance.

8.18 Write-back: The penalty is the same for both read and write misses. A new block is brought into the cache and a word of data is either read from it or written into it.

Write-through: For a write operation, whether or not a miss occurs, the new information is written directly into the main memory and no further action is taken. The penalty incurred is the time to write one word into the main memory. This penalty is reduced if a write buffer is used. A read miss, on the other hand, causes the addressed block to be loaded into the cache. Thus, the penalty for a read miss is higher than that for a write miss.

Although the write-through protocol has a lower penalty for write misses than the write-back protocol, this does not necessarily mean that it will always perform better. Since the addressed block is not loaded into the cache, all subsequent Write operations to the same block, if any, will incur a penalty. Thus, the relative performance of the two algorithms depends on the frequency of the write operations and the extent to which they exhibit locality of reference.

8.19. Periodically reallocate some of the memory pages used by the program with the lowest page transfer activity to the program with the highest activity. This should not be done too frequently. Otherwise, the time cost in doing the reallocation would outweigh its potential performance benefit. A reasonable strategy is to reallocate pages only when the difference in the page transfer rate exceeds some threshold.

8.20. Continuing the execution of an instruction interrupted by a page fault requires saving the entire state of the processor, which includes saving all registers that may have been affected by the instruction as well as the control information that indicates how far its execution has progressed. The alternative of re-executing the instruction from the beginning is simpler. It requires that partial results, if any, generated during instruction execution be stored in temporary locations, so that they can be safely discarded if the execution of that instruction is interrupted. This is particularly easy to do in a RISC-style processor, as the results of the instruction are recorded only in the very last execution step, as described in Chapter 5, and there are no side effects.

8.21. A page fault can occur part-way during the execution of an instruction that has an operand on a different page. When this happens, the operating system has to suspend execution of the program containing this instruction and start a DMA operation to transfer the required page. (It may also perform a context switch — see Chapter 4.) It resumes execution of the suspended program after the required page has been read from the disk. The difficulty that arises in this case is that the instruction that caused the page fault has been partially executed.

There are two ways to proceed. When execution of the program is suspended, the processor may discard the instruction that caused the page fault together with any partial results that may have been produced. It

adjusts the program counter, so that the discarded instruction is executed again when the suspended program resumes execution. Alternatively, the full state of the processor may be saved at the time the page fault occurs, including how many execution steps have been completed for the instruction that caused the page fault and the contents of any temporary registers it is using. With this information, the processor can later resume execution of the instruction at the point of suspension.

8.22. (a) The maximum number of bytes that can be stored on this disk is $24 \times 14000 \times 400 \times 512 = 68.8 \times 10^9$ bytes.

(b) The data transfer rate is $(400 \times 512 \times 7200)/60 = 24.58 \times 10^6$ bytes/s.

(c) We need 9 bits to identify a sector, 14 bits for a track, and 5 bits for a surface. Thus, using a 32 bit word $\mathrm{b}_{31}...\mathrm{b}_0$, a possible scheme is to use bits b_{8-0} for sector, b_{22-9} for track, and b_{27-23} for surface identification. Bits b_{31-28} would not be used.

8.23. (*a*) The time needed to access and transfer each block is $8 + 3 + (64/60)/10^3 = 12.07$ ms. The portion of time occupied by seek and rotational delay is 11/12.07 = 91%.

(*b*) It takes 11 ms to reach the first block and 3 ms to reach every subsequent block. Therefore, the total time needed to transfer 20 blocks is $11 + 19 \times 3 + 20 \times 1.07 = 89.4$ ms.

8.24. (*a*) Data are transferred from the disk data buffer to the memory at the maximum data rate of the bus. Hence, only one disk can be transferring data at any given instant. After transferring the contents of the buffer, there is a break in data transfer while the next block is read from the disk and stored in the buffer. The maximum sustained data rate occurs when blocks are stored in contiguous sectors on the same track. In this case, the data rate from any one disk is 30 Mbytes per second. This assumes that the hardware is organized such that it can start reading a block from the disk while the previous block is being sent to the memory. The maximum memory transfer rate is $4/(10 \times 10^{-9}) = 400$ Mbytes per second. Therefore, up to 13 disks can be simultaneously transferring data to/from the main memory.

(*b*) The average time it takes to transfer one block is $6 + 3 + 8/30 = 9.27$ ms, during which 8K bytes are transferred. This corresponds to a data rate of 863K bytes/s. Therefore, over a long period of time, any one disk uses $863 \times 10^3/400 \times 10^6 = 0.22\%$ of the available memory cycles.

8.25. The sector size should influence the choice of page size, because the sector is the smallest directly addressable block of data on the disk that is read or written as a unit. Therefore, page size should be some small integral number of sectors.