Chapter 5 – The Memory System

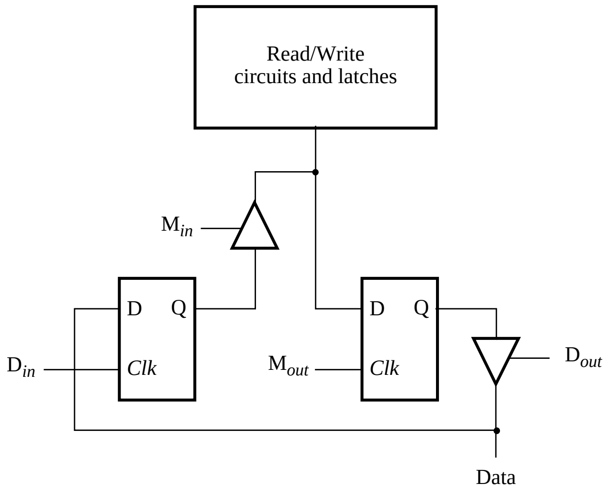
5.1. The block diagram is essentially the same as in Figure 5.10, except that 16 rows (of four 512×8 chips) are needed. Address lines A_{18-0} are connected to all chips. Address lines A_{22-19} are connected to a 4-bit decoder to select one of the 16 rows.

5.2. The minimum refresh rate is given by

$$\frac{50 \times 10^{-15} \times (4.5 - 3)}{9 \times 10^{-12}} = 8.33 \times 10^{-3}\ \mathrm{s}$$

Therefore, each row has to be refreshed every 8 ms.

5.3. Need control signals M_{in} and M_{out} to control storing of data into the memory cells and to gate the data read from the memory onto the bus, respectively. A possible circuit is



5.4. (a) It takes $5 + 8 = 13$ clock cycles.

$$\text{Total time} = \frac{13}{(133 \times 10^6)} = 0.098 \times 10^{-6}\ \mathrm{s} = 98\ \mathrm{ns}$$

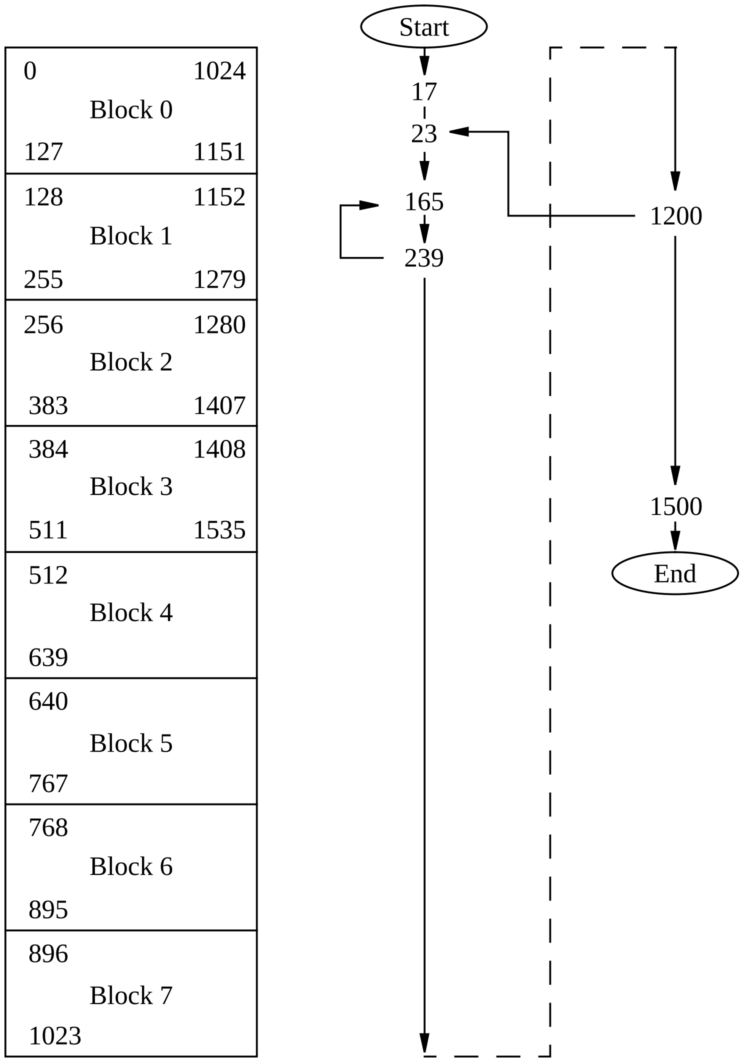
$$\text{Latency} = \frac{5}{(133 \times 10^6)} = 0.038 \times 10^{-6}\ \mathrm{s} = 38\ \mathrm{ns}$$

(b) It takes twice as long to transfer 64 bytes, because two independent 32-byte transfers have to be made. The latency is the same, i.e. 38 ns.

5.5. A faster processor chip will result in increased performance, but the amount of 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.

5.6. (a) Main memory address length is 16 bits. TAG field is 6 bits. BLOCK field is 3 bits (8 blocks). WORD field is 7 bits (128 words per block).

(b) The program words are mapped on the cache blocks as follows:



Hence, the sequence of reads from the main memory blocks into cache blocks is

$$\text{Block}: \underbrace{0,1,2,3,4,5,6,7,0,1,}_{\text{Pass 1}}\ \underbrace{0,1,0,1,}_{\text{Pass 2}}\ 0,1,\ldots,0,1,\ \underbrace{0,1,0,1,}_{\text{Pass 9}}\ \underbrace{0,1,0,1,2,3}_{\text{Pass 10}}$$

As this sequence shows, 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.

The total time for reading the blocks from the main memory into the cache is therefore

$$(10 + 4 \times 9 + 2) \times 128 \times 10\,\tau = 61,440\,\tau$$

Executing the program out of the cache:

$$\begin{aligned}
\text{Outer loop} - \text{inner loop} &= [(1200 - 22) - (239 - 164)]10 \times 1\tau = 11,030\,\tau \\
\text{Inner loop} &= (239 - 164)200 \times 1\,\tau = 15,000\,\tau \\
\text{End section of program} &= 1500 - 1200 = 300 \times 1\,\tau \\
\text{Total execution time} &= 87,770\,\tau
\end{aligned}$$

5.7. 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 BNE is read from the main memory and again stored in location 0. 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 below.

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

5.8. 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.

Instructions and data are best stored in separate caches to avoid the data overwriting instructions, as in Problem 5.7.

5.9. (a) 4096 blocks of 128 words each require $12+7=19$ bits for the main memory address.

(b) TAG field is 8 bits. SET field is 4 bits. WORD field is 7 bits.

5.10. (a) TAG field is 10 bits. SET field is 4 bits. WORD field is 6 bits.

(b) Words 0, 1, 2, $\cdots$, 4351 occupy blocks 0 to 67 in the main memory (MM). After blocks 0, 1, 2, $\cdots$, 63 have been read from MM into the cache on the first pass, the cache is full. Because of the fact that the replacement algorithm is LRU, MM blocks that occupy the first four sets of the 16 cache sets are always overwritten before they can be used on a successive pass. In particular, MM blocks 0, 16, 32, 48, and 64 continually displace each other in competing for the 4 block positions in cache set 0. The same thing occurs in cache set 1 (MM blocks, 1, 17, 33, 49, 65), cache set 2 (MM blocks 2, 18, 34, 50, 66) and cache set 3 (MM blocks 3, 19, 35, 51, 67). MM blocks that occupy the last 12 sets (sets 4 through 15) are fetched once on the first pass and remain in the cache for the next 9 passes. On the first pass, all 68 blocks of the loop must be fetched from the MM. On each of the 9 successive passes, blocks in the last 12 sets of the cache ($4 \times 12 = 48$) are found in the cache, and the remaining 20 ($68 - 48$) blocks must be fetched from the MM.

$$\begin{aligned}\text{Improvement factor} &= \frac{\text{Time without cache}}{\text{Time with cache}}\\ &= \frac{10 \times 68 \times 10\tau}{1 \times 68 \times 11\tau + 9(20 \times 11\tau + 48 \times 1\tau)}\\ &= 2.15\end{aligned}$$

5.11. This replacement algorithm is actually better on this particular "large" loop example. After the cache has been filled by the main memory blocks 0, 1, $\cdots$, 63 on the first pass, block 64 replaces block 48 in set 0. On the second pass, block 48 replaces block 32 in set 0. On the third pass, block 32 replaces block 16, and on the fourth pass, block 16 replaces block 0. On the fourth pass, there are two replacements: 0 kicks out 64, and 64 kicks out 48. On the sixth, seventh, and eighth passes, there is only one replacement in set 0. On the ninth pass there are two replacements in set 0, and on the final pass there is one replacement. The situation is similar in sets 1, 2, and 3. Again, there is no contention in sets 4 through 15. In total, there are 11 replacements in set 0 in passes 2 through 10. The same is true in sets 1, 2, and 3. Therefore, the improvement factor is

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

5.12. For the first loop, the contents of the cache are as indicated in Figures 5.20 through 5.22. 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. The performance can be improved by introducing some randomness in the replacement algorithm.

5.13. 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. 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

5.14. 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

		Contents of data cache after:			
	Block position	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

5.15. The block size (number of words in a block) of the cache should be at least as large as 2^k, in order to take full advantage of the multiple module memory when transferring a block between the cache and the main memory. Power of 2 multiples of 2^k work just as efficiently, and are natural because block size is 2^k for k bits in the ”word” field.

5.16. 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

Smaller size

- more misses

5.17. For 16-word blocks the value of M is $1 + 8 + 3 \times 4 + 4 = 25$ cycles. Then

$$\frac{\text{Time without cache}}{\text{Time with cache}} = 4.04$$

In order to compare the 8-word and 16-word blocks, we can assume that two 8-word blocks must be brought into the cache for each 16-word block. Hence, the effective value of M is $2 \times 17 = 34$. Then

$$\frac{\text{Time without cache}}{\text{Time with cache}} = 3.3$$

Similarly, for 4-word blocks the effective value of M is $4(1+8+4) = 52$ cycles. Then

$$\frac{\text{Time without cache}}{\text{Time with cache}} = 2.42$$

Clearly, interleaving is more effective if larger cache blocks are used.

5.18. The hit rates are

$$\begin{aligned} h_1 = h_2 = h &= 0.95 \text{ for instructions} \\ &= 0.90 \text{ for data} \end{aligned}$$

The average access time is computed as

$$t_{ave} = hC_1 + (1-h)hC_2 + (1-h)^2M$$

(a) With interleaving $M = 17$. Then

$$\begin{aligned} t_{ave} &= 0.95 \times 1 + 0.05 \times 0.95 \times 10 + 0.0025 \times 17 + 0.3(0.9 \times 1 + 0.1 \times 0.9 \times 10 + 0.01 \times 17) \\ &= 2.0585 \text{ cycles} \end{aligned}$$

(b) Without interleaving $M = 38$. Then $t_{ave} = 2.174$ cycles.

(c) Without interleaving the average access takes $2.174/2.0585 = 1.056$ times longer.

5.19. Suppose that it takes one clock cycle to send the address to the L2 cache, one cycle to access each word in the block, and one cycle to transfer a word from the L2 cache to the L1 cache. This leads to $C_2 = 6$ cycles.

(a) With interleaving $M = 1 + 8 + 4 = 13$. Then $t_{ave} = 1.79$ cycles.

(b) Without interleaving $M = 1 + 8 + 3 \times 4 + 1 = 22$. Then $t_{ave} = 1.86$ cycles.

(c) Without interleaving the average access takes $1.86/1.79 = 1.039$ times longer.

5.20. The analogy is good with respect to:

- relative sizes of toolbox, truck and shop versus L1 cache, L2 cache and main memory
- relative access times
- relative frequency of use of tools in the 3 storage places versus the data accesses in caches and the main memory

The analogy fails with respect to the facts that:

- 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, while in the case of a new program that is run on a computer there is no relevant data 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 the subsequent programs
- tools displaced by the need to use other tools are never thrown away, while data in the cache blocks are simply overwritten if the blocks are not flagged as dirty

5.21. Each 32-bit number comprises 4 bytes. Hence, each page holds 1024 numbers. There is space for 256 pages in the 1M-byte portion of the main memory that is allocated for storing data during the computation.

(a) Each column is one page; there will be 1024 page faults.

(b) Processing of entire columns, one at a time, would be very inefficient and slow. However, if only one quarter of each column (for all columns) is processed before the next quarter is brought in from the disk, then each element of the array must be loaded into the memory twice. In this case, the number of page faults would be 2048.

(c) Assuming that the computation time needed to normalize the numbers is negligible compared to the time needed to bring a page from the disk:

Total time for (a) is 1024×40 ms = 41 s

Total time for (b) is 2048×40 ms = 82 s

5.22. The operating system may increase the main memory pages allocated to a program that has a large number of page faults, using space previously allocated to a program with a few page faults.

5.23. 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 the execution has progressed. The alternative of re-executing the instruction from the beginning requires a capability to reverse any changes that may have been caused by the partial execution of the instruction.

5.24. The problem is that a page fault may occur during intermediate steps in the execution of a single instruction. The page containing the referenced location must be transferred from the disk into the main memory before execution can proceed. Since the time needed for the page transfer (a disk operation) is very long, as compared to instruction execution time, a context-switch will usually be made. (A context-switch consists of preserving the state of the currently executing program, and ”switching” the processor to the execution of another program that is resident in the main memory.) The page transfer, via DMA, takes place while this other program executes. When the page transfer is complete, the original program can be resumed.

Therefore, one of two features are needed in a system where the execution of an individual instruction may be suspended by a page fault. The first possibility is to save the state of instruction execution. This involves saving more information (temporary programmer-transparent registers, etc.) than needed when a program is interrupted between instructions. The second possibility is to "unwind" the effects of the portion of the instruction completed when the page fault occurred, and then execute the instruction from the beginning when the program is resumed.

5.25. (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) Need 9 bits to identify a sector, 14 bits for a track, and 5 bits for a surface. Thus, a possible scheme is to use address bits A_{8-0} for sector, A_{22-9} for track, and A_{27-23} for surface identification. Bits A_{31-28} are not used.

5.26. The average seek time and rotational delay are 6 and 3 ms, respectively. The average data transfer rate from a track to the data buffer in the disk controller is 34 Mbytes/s. Hence, it takes 8K/34M = 0.23 ms to transfer a block of data.

(a) The total time needed to access each block is $9 + 0.23 = 9.23$ ms. The portion of time occupied by seek and rotational delay is 9/9.23 = 0.97 = 97%.

(b) Only rotational delays are involved in 90% of the cases. Therefore, the average time to access a block is $0.9 \times 3 + 0.1 \times 9 + 0.23 = 3.89$ ms. The portion of time occupied by seek and rotational delay is 3.6/3.89 = 0.92 = 92%.

5.27. (a) The rate of transfer to or from any one disk is 30 megabytes per second. Maximum memory transfer rate is 4/$(10 \times 10^{-9}) = 400 \times 10^6$ bytes/s, which is 400 megabytes per second. Therefore, 13 disks can be simultaneously flowing data to/from the main memory.

(b) 8K/30M = 0.27 ms is needed to transfer 8K bytes to/from the disk. Seek and rotational delays are 6 ms and 3 ms, respectively. Therefore, 8K/4 = 2K words are transferred in 9.27 ms. But in 9.27 ms there are $(9.27 \times 10^{-3})/(0.01 \times 10^{-6}) = 927 \times 10^3$ memory (word) cycles available. Therefore, over a long period of time, any one disk steals only $(2/927) \times 100 = 0.2$% of available memory cycles.

5.28. 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, pages should be some small integral number of sectors in size.

5.29. The next record, j, to be accessed after a forward read of record i has just been completed might be in the forward direction, with probability 0.5 (4 records distance to the beginning of j), or might be in the backward direction with probability 0.5 (6 records distance to the beginning of j plus 2 direction reversals).

Time to scan over one record and an interrecord gap is

$$\frac{1}{800}\frac{\mathrm{s}}{\mathrm{cm}} \times \frac{1}{2000}\frac{\mathrm{cm}}{\mathrm{bit}} \times 4000\,\mathrm{bits} \times 1000\,\mathrm{ms} + 3 = 2.5 + 3 = 5.5\,\mathrm{ms}$$

Therefore, average access and read time is

$$0.5(4 \times 5.5) + 0.5(6 \times 5.5 + 2 \times 225) + 5.5 = 258\,\mathrm{ms}$$

If records can be read while moving in both directions, average access and read time is

$$0.5(4 \times 5.5) + 0.5(5 \times 5.5 + 225) + 5.5 = 142.75\,\mathrm{ms}$$

Therefore, the average percentage gain is $(258 - 142.75)/258 \times 100 = 44.7\%$ The major gain is because the records being read are relatively close together, and one less direction reversal is needed.