**9.3** Consider the page table shown in Figure 9.30 for a system with 12-bit virtual and physical addresses and with 256-byte pages. The list of free page frames is $D$, $E$, $F$ (that is, $D$ is at the head of the list, $E$ is second, and $F$ is last).

Convert the following virtual addresses to their equivalent physical addresses in hexadecimal. All numbers are given in hexadecimal. (A dash for a page frame indicates that the page is not in memory.)

- 9EF
- 111
- 700
- 0FF

0FF
211
D00
EFF

| Page | Page Frame |
|---|---|
| 0 | – |
| 1 | 2 |
| 2 | C |
| 3 | A |
| 4 | – |
| 5 | 4 |
| 6 | 3 |
| 7 | – |
| 8 | B |
| 9 | 0 |

**Figure 9.30** Page table for Exercise 9.3.

**9.6** An operating system supports a paged virtual memory. The central processor has a cycle time of 1 microsecond. It costs an additional 1 microsecond to access a page other than the current one. Pages have 1,000 words, and the paging device is a drum that rotates at 3,000 revolutions per minute and transfers 1 million words per second. The following statistical measurements were obtained from the system:

- One percent of all instructions executed accessed a page other than the current page.
- Of the instructions that accessed another page, 80 percent accessed a page already in memory.
- When a new page was required, the replaced page was modified 50 percent of the time.

Calculate the effective instruction time on this system, assuming that the system is running one process only and that the processor is idle during drum transfers.

3000 rev/min

$\frac{3000}{60} = 50$ rev/sec

0.02 sec/rev

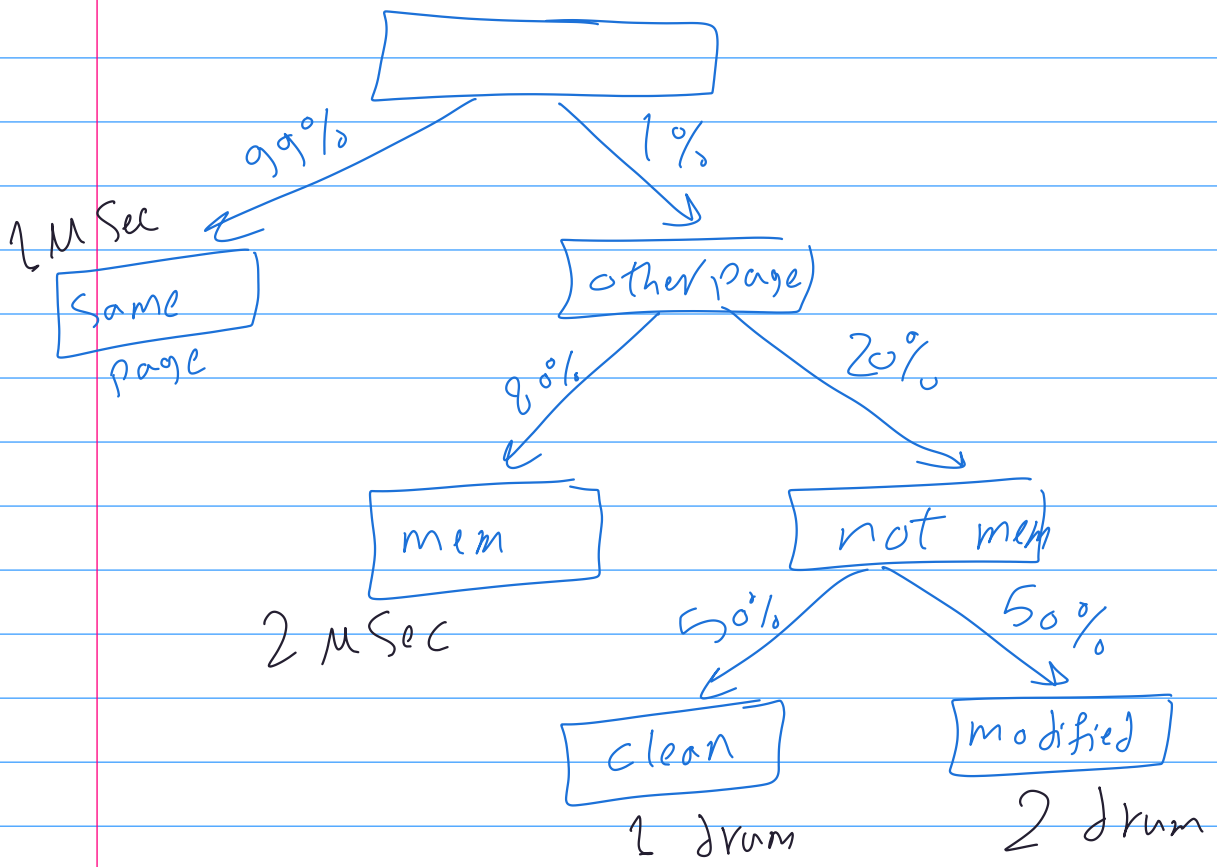
page 1000 word

1 000 000 word/sec

1000 page/sec

1 mSec/page

drum = 0.02 + 1m sec



0.02 + 1m

$$0.99\,(1\,\mu) + 0.01\,(0.8\,(2M) + 0.2\,(0.5\,(1D) + 0.5\,(2D)))$$

$$= 3.4006 \times 10^{-5}$$

$$= 34.006\ \mu Sec$$

**9.7** Consider the two-dimensional array `A`:

```
int A[][] = new int[100][100];
```

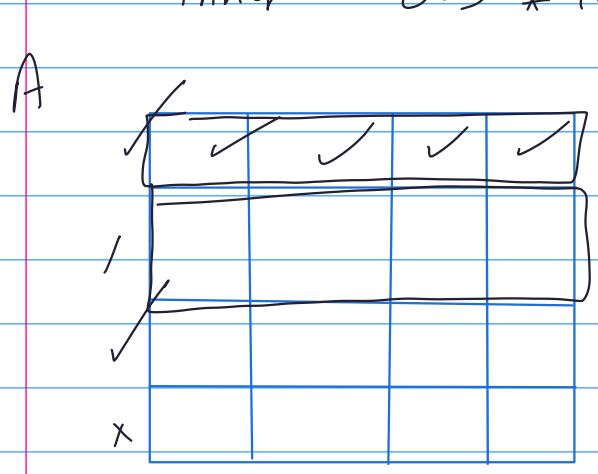
where `A[0][0]` is at location 200 in a paged memory system with pages of size 200. A small process that manipulates the matrix resides in page 0 (locations 0 to 199). Thus, every instruction fetch will be from page 0.

For three page frames, how many page faults are generated by the following array-initialization loops? Use LRU replacement, and assume that page frame 1 contains the process and the other two are initially empty.

a.
```
for (int j = 0; j < 100; j++)
    for (int i = 0; i < 100; i++)
        A[i][j] = 0;
```
5000 page fault

b.
```
for (int i = 0; i < 100; i++)
    for (int j = 0; j < 100; j++)
        A[i][j] = 0;
```
50 page fault

inner 0.5 x 100 = 50



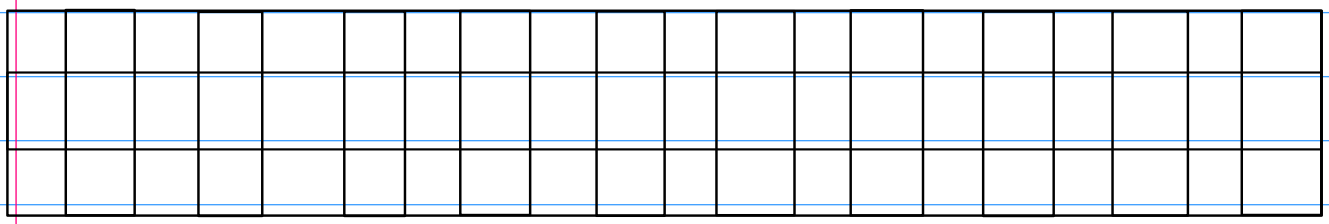

**9.8** Consider the following page reference string:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, and seven frames? Remember that all frames are initially empty, so your first unique pages will cost one fault each.

- LRU replacement
- FIFO replacement
- Optimal replacement

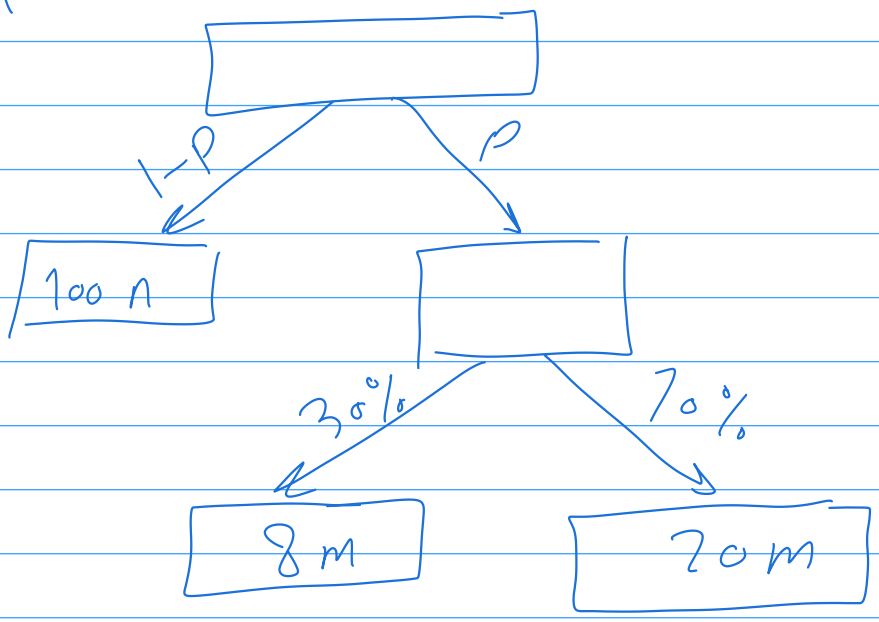
1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.



**9.19** Assume that we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty frame is available or if the replaced page is not modified and 20 milliseconds if the replaced page is modified. Memory-access time is 100 nanoseconds.

Assume that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?

p = page fault rate




$$200n = (1-p)(100n) + p(0.3(8m) + 0.7(20m))$$

$$200n = 100n - p100n + p\ 16.4m$$

$$p = \frac{100n}{16.4m - 100n} = 6.0976 \times 10^{-6}$$

**9.21** Consider the following page reference string:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1.

Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?

- LRU replacement
- FIFO replacement
- Optimal replacement

LRU

19

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1.

| 7 | 7 | 7 | 1 |  | 1 | 3 | 3 | 3 | 7 |  | 7 | 7 | 5 | 5 | 5 | 2 | 2 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | 2 | 2 | 2 |  | 2 | 2 | 4 | 4 | 4 |  | 1 | 1 | 1 | 4 | 4 | 4 | 3 | 3 | 3 |
|  |  | 3 | 3 |  | 5 | 5 | 5 | 6 | 6 |  | 6 | 0 | 0 | 0 | 6 | 6 | 6 | 0 | 0 |

→ ~~7, 2, 3, 1, 5, 4, 6, 7, 1, 0, 5, 4, 6, 2~~ 3 , 0

FIFO

17

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1.

| 7 | 7 | 7 | 1 |  | 1 |  | 1 | 6 | 6 |  | 6 | 0 | 0 | 0 | 6 | 6 | 6 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | 2 | 2 | 2 |  | 5 |  | 5 | 5 | 7 |  | 7 | 7 | 5 | 5 | 5 | 2 | 2 | 2 | 1 |
|  |  | 3 | 3 |  | 3 |  | 4 | 4 | 4 |  | 1 | 1 | 1 | 4 | 4 | 4 | 3 | 3 | 3 |

optimal

13

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1.

| 7 | 7 | 7 | 1 |  | 1 |  | 1 | 1 | 1 |  |  | 1 |  | 1 | 1 | 1 | 1 |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | 2 | 2 | 2 |  | 5 |  | 5 | 5 | 5 |  |  | 5 |  | 4 | 6 | 2 | 3 |  |  |
|  |  | 3 | 3 |  | 3 |  | 4 | 6 | 7 |  |  | 0 |  | 0 | 0 | 0 | 0 |  |  |

**9.22** The page table shown in Figure 9.32 is for a system with 16-bit virtual and physical addresses and with 4,096-byte pages. The reference bit is set to 1 when the page has been referenced. Periodically, a thread zeroes out all values of the reference bit. A dash for a page frame indicates the page is not in memory. The page-replacement algorithm is localized LRU, and all numbers are provided in decimal.

a. Convert the following virtual addresses (in hexadecimal) to the equivalent physical addresses. You may provide answers in either hexadecimal or decimal. Also set the reference bit for the appropriate entry in the page table.

- 0xE12C
- 0x3A9D
- 0xA9D9
- 0x7001
- 0xACA1

b. Using the above addresses as a guide, provide an example of a logical address (in hexadecimal) that results in a page fault.

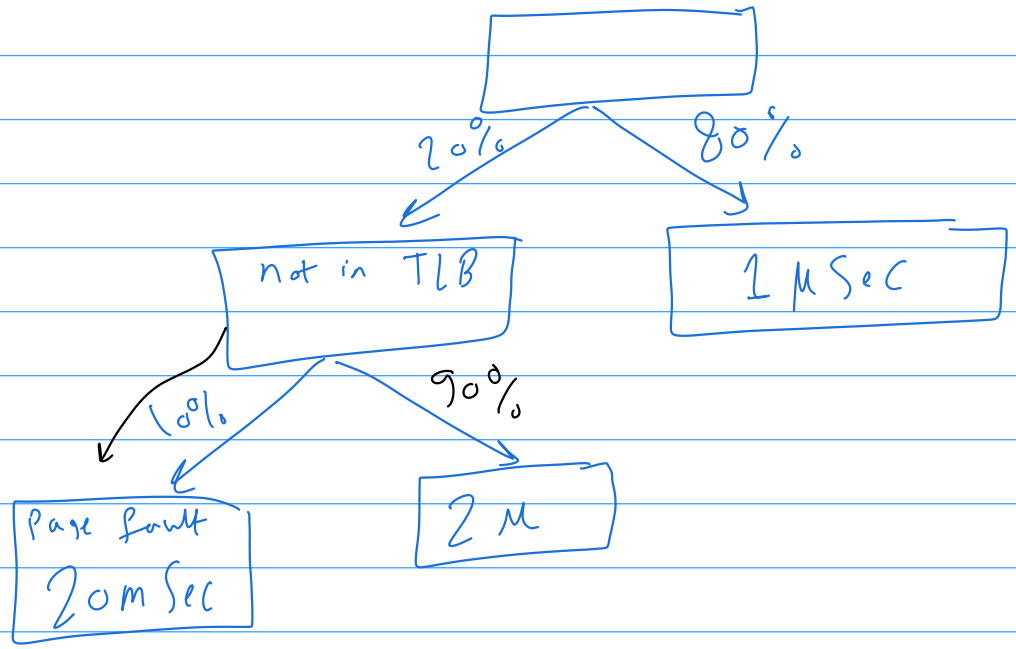
c. From what set of page frames will the LRU page-replacement algorithm choose in resolving a page fault?

| Page | Page Frame | Reference Bit |
|---|---|---|
| 0 | 9 | 0 |
| 1 | 1 | 0 |
| 2 | 14 | 0 |
| 3 | 10 | 0 |
| 4 | – | 0 |
| 5 | 13 | 0 |
| 6 | 8 | 0 |
| 7 | 15 | 0 |
| 8 | – | 0 |
| 9 | 0 | 0 |
| 10 | 5 | 0 |
| 11 | 4 | 0 |
| 12 | – | 0 |
| 13 | – | 0 |
| 14 | 3 | 0 |
| 15 | 2 | 0 |

**Figure 9.32** Page table for Exercise 9.22.

**9.31** Consider a demand-paging system with a paging disk that has an average access and transfer time of 20 milliseconds. Addresses are translated through a page table in main memory, with an access time of 1 microsecond per memory access. Thus, each memory reference through the page table takes two accesses. To improve this time, we have added an associative memory that reduces access time to one memory reference if the page-table entry is in the associative memory.

Assume that 80 percent of the accesses are in the associative memory and that, of those remaining, 10 percent (or 2 percent of the total) cause page faults. What is the effective memory access time?




$20m + 2\mu$

$$0.8(1\mu) + 0.2(0.1(20m) + 0.9(2\mu))$$

$+2\mu$

$$= 400 \ \mu Sec$$

$$\approx 400\mu$$