8.1

a) Following are the steps to go from virtual address to physical address:

* Divide the virtual address into 2 fields: the page number and the offset address.
* The page number is then used to refer to the page table to get the corresponding frame number.
* Once the page number is found in the page table, the corresponding frame number is found.
* Then, the frame number and the offset are concatenated to give the physical address.

b) Page number = Virtual Address / Page size

1. Page # = 1052 / 1024 = 1.027

We can discard the fraction from the decimal. So, the page number is 1. From the given page table, the frame number is 7.

1052 is 10000011100. We know the page number is 1 (leftmost bit), so the offset is 0000011100 which is 28 in decimal form. So offset is 28.

Physical Address = (frame number \* page size) + offset

= (7 \* 1024) + 28

= 7196

1. Page # = 2221 / 1024 = 2.168

We can discard the fraction from the decimal. So, the page number is 2. From the given page table, there is a page fault at page number 2.

2221 is 100010101101. We know the page number is 2 (2 leftmost bits), so the offset is 0010101101 which is 173 in decimal form. So offset is 173.

Physical Address = (frame number \* page size) + offset

= (? \* 1024) + 173

= page fault

|  |  |
| --- | --- |
| - | 221 |

If we aren’t handling page faults, it’s

1. Page # = 5499 / 1024 = 5.37

We can discard the fraction from the decimal. So, the page number is 5. From the given page table, the frame number is 0.

5499 is 1010101111011. We know the page number is 5 (3 leftmost bits), so the offset is 0101111011 which is 379 in decimal form. So offset is 379.

Physical Address = (frame number \* page size) + offset

= (0 \* 1024) + 379

= 379

8.6 a)

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

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

F F F F F F F F F F F F F

Initially, 3 frames are filled with 1 0 2. Then since 2 is already in memory, we don’t need to do anything. Then 7 is placed in 4th frame.

Page fault at reference 6 is because we need 6 but it’s not in memory. So, we need to replace 0 (least recently used) with 6.

Page fault at the reference 0 is because we need 0 but it’s not in memory. So, we need to replace 2 (least recently used) with 0.

Page fault at the reference 2 is because we need 2 but it’s not in memory. So, we need to replace 6 (least recently used) with 2.

Page fault at the reference 3 is because we need 3 but it’s not in memory. So, we need to replace 7 (least recently used) with 3.

Page fault at the reference 4 is because we need 4 but it’s not in memory. So, we need to replace 1 (least recently used) with 4.

Page fault at the reference 5 is because we need 5 but it’s not in memory. So, we need to replace 2 (least recently used) with 5.

Page fault at the reference 1 is because we need 1 but it’s not in memory. So, we need to replace 3 (least recently used) with 1.

Page fault at the reference 2 is because we need 2 but it’s not in memory. So, we need to replace 0 (least recently used) with 2.

Page fault at the reference 6 is because we need 6 but it’s not in memory. So, we need to replace 1 (least recently used) with 6.

Page fault at the reference 7 is because we need 7 but it’s not in memory. So, we need to replace 2 (least recently used) with 7.

Page fault at the reference 2 is because we need 2 but it’s not in memory. So, we need to replace 4 (least recently used) with 2.

Page fault at the reference 4 is because we need 4 but it’s not in memory. So, we need to replace 5 (least recently used) with 4.

Page fault at the reference 3 is because we need 3 but it’s not in memory. So, we need to replace 6 (least recently used) with 3. Rest of the references are in memory, so no page faults.

Number of page faults: 13

Total Page references: 33

Hit ratio = # Page faults / # Page references

= 13 / 33 = 0.39

So hit ratio using LRU replacement policy is 0.39 (39%).

b)

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

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

F F F F F F F F F F F F F

When we need to replace, we replace the page which is brought into the memory first.

Number of page faults: 13

Total Page references: 33

Hit ratio = # Page faults / # Page references

= 13 / 33 = 0.39

So hit ratio using FIFO replacement policy is 0.39 (39%).

c) Both replacement policies have a hit ratio of 0.39, so both are equally effective when it comes to performance.

8.10

Page table entry size = 4 bytes Page size: 4 Kbytes # of pages = 4 \* 1024 = 4096 = 212

Total address = 1024 \* 4096 = 222

The given address space is 264 bytes. So, top page table would be 210 page tables, so 232 bytes. Repeating this we get,

Level 1 = 222 bytes

Level 2 = 232 bytes

Level 3 = 242 bytes

Level 4 = 252 bytes

Level 5 = 262 bytes

Level 6 = 272 bytes

We need 6 levels of page tables to map a 64-bit address space.