translate virtual address to physical address

Asked 5 years, 7 months ago Modified 2 years, 10 months ago Viewed 105k times

The following page table 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.All numbers are provided in decimal.

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

I want to convert the following virtual addresses (in hexadecimal) to the equivalent physical addresses. Also I want to set the reference bit for the appropriate entry in the page table.

- `0xE12C`
- `0x3A9D`
- `0xA9D9`

I found and tried This but it did not help me much.

memory | memory-management | operating-system | virtual

Share Follow

edited Apr 14, 2017 at 13:55

joke
101 ● 1 ● 6

asked Oct 27, 2016 at 19:31

Ruan
397 ● 1 ● 5 ● 16

3 Answers

Sorted by: Highest score (default)

26

It is given that virtual address is 16 bit long.
Hence, there are 2^16 addresses in the virtual address space.
Page Size is given to be 4 KB (there are 4K (4 * (2 ^ 10))addresses in a page), so the number of pages will be (2^16) / (2 ^ 12) = 2 ^ 4.
To address each page 4 bits are required.
The most significant 4 bits in the virtual address will denote the page number being referred and the remaining 12 bits will be the page offset.

One thing to remember is page size (in the virtual address space) is always same as the frame size in the main memory. Hence the last 12 bits will remain same in the physical address as that of the virtual address.

To get the frame address in the main memory just use the first 4 bits.
Example: Consider the virtual address 0xACA1
Here A in **A**CA1 denotes the page number (10) and corresponding frame no is 5 (0101) hence the resulting physical address will be → 0x5CA1.

Share Follow

answered Oct 28, 2016 at 7:58

Akash Mahapatra
2,688 ● 1 ● 12 ● 28

17

To translate a **virtual** address to a **physical** address (applies **ONLY** to this homework question), we need to know **2** things:

- Page Size
- Number of bits for virtual address

In this example: **16-bit system**, **4KB page size** and **physical memory size** is **64KB**.

First of all we need to determine the number of needed bits to act as offset inside page.

`log2(Page-Size)` = `log2(4096)` = **12** bits for **offset**

Out of the 16 bits for virtual address, 12 are for offset, that means each process has `2^4 = 16` virtual pages. Each entry in page table stores the corresponding frame accommodating the page. For example:

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

Now lets translate!

First of all for ease of work lets convert `0xE12C` to `binary`.

```
0xE12C = (1110 0001 0010 1100) in base 2
1110 = 14 in decimal
Entry 14 in P.T => Page frame 3.
```

Lets concatenate it to the 12 offset bits

```
Answer: (0011 0001 0010 1100) = 0x312C
```

Another example: `0x3A9D`

```
0x3A9D = 0011 1010 1001 1101
0011 = 3
PageTable[3] = 10
10 in decimal = 1010 in binary
1010 1010 1001 1101 in binary = 0xAA9D
```

Share Follow

edited Jul 9, 2019 at 11:07

answered Oct 27, 2016 at 19:46

Tony Tannous

13.2k ● 8 ● 44 ● 80

6

To help you solve this question, we need to get our details right:

1. 16 bit of virtual address space = 2^16 = 65,536 address space
2. 16 bit of physical address space = 2^16 = 65,536 address space
3. 4096 Byte page size determines the offset, which is Log(4096) / Log (2) = 12 bit. This means, 2^12 for Page size
4. As per @Akash Mahapatra, the offset from virtual address is directly mapped to the offset onto physical address

As such, we now have:

- 2^16 (16bit) for virtual address - 2^12 (12bit) for offset = 4-bit for pages, or rather total number of pages available.
- I won't repeat the calculation for physical since it's the same numbers.
- 2^4 (4bit) for pages = 16, which correlates to the number of table entries above!

We're getting there... be patient! :)

Memory Address 0xE12C in hex notation is also known to be holding 16-bit of address. (Since it's stated in the question.)

Let's butcher the address now...

We first remove '0x' from the info.

We can convert E12C to binary notation like @Tony Tannous, but I am going to apply a little short-cut.

```
E             |  1   2   C
^             |  ^   ^   ^
Address       |  Offset
first 4-bit   |  the next 12-bit
```

I simply use a ratio. Well, the address is notated in 4 characters above, and since 16/4 = 4, I can define the first letter as virtual address, while the other 3 are offset address.

With the information, 'E' in hexadecimal format, I need to convert to Decimal = 14. Then I look at your table provided, and I found page frame '3'. Page frame 3 is noted in decimal format, which then need to be converted back to Hexadecimal format... Duh!... which is 3!

So, the Physical address mapping of the virtual memory location of 0xE12C can be found at 0x312C on the physical memory.

You will then go back to the table, and refer to the reference bit column and put a '1' to the row 14.

Apply the same concept for these -

```
0x3A9D → 0xAA9D
0xA9D9 → 0x59D9
0x7001 → 0xF001
0xACA1 → 0x5CA1
```

If you notice, the last 3 digits are the same (which determines the offset). And the 1st of the 4-digits are mapped according to the table:

```
table entry 3 -> page frame 10 -> hex notation A
table entry A (10) -> page frame 5 -> hex notation 5
table entry 7 -> page frame 15 -> hex notation F
table entry A (10) -> page frame 5 -> hex notation 5
```

Hope this explanation helps you and others like me! :)

Share Follow

edited Jul 5, 2018 at 7:34

answered Apr 14, 2017 at 7:33

joke

101 ● 1 ● 6