CSC369 - Tutorial 7

No exercise sheet today
Virtual address translation
A3 help
Linear translation

Step by step
Goal: determine the virtual and physical addresses format

Parameters of Paging System

Virtual address space size 32k
Physical memory size 64k
Page size 4k

The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
If the bit is 1, the rest of the entry is thePFN.
If the bit is 0, the page is not valid.

Page Table (from entry 0 down to the max size)

<table>
<thead>
<tr>
<th>VPN</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x8000000a</td>
</tr>
<tr>
<td>1</td>
<td>0x8000000b</td>
</tr>
<tr>
<td>2</td>
<td>0x8000000c</td>
</tr>
<tr>
<td>3</td>
<td>0x80000002</td>
</tr>
<tr>
<td>4</td>
<td>0x80000000</td>
</tr>
<tr>
<td>5</td>
<td>0x80000009</td>
</tr>
<tr>
<td>6</td>
<td>0x00000000</td>
</tr>
<tr>
<td>7</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

Virtual Address Trace

For each virtual address, write down the physical address it translates to OR check the Invalid button.

VADDR 0x508c (decimal: 20520) PADDR 0x........... OR  Invalid
VADDR 0x4cd8 (decimal: 19572) PADDR 0x........... OR Invalid
VADDR 0x40a6 (decimal: 16550) PADDR 0x........... OR Invalid

Submit
How many bits are needed to address an element inside an array of 4K elements?
How many bits are needed to address an element inside an array of 4K elements?

Log base 2 of 4KB! Why?
How many bits are needed to address an element inside an array of 4K elements?

Log base 2 of 4KB! Why?

1 bit lets you index 2 positions: 0 and 1.
How many bits are needed to address an element inside an array of 4K elements?

Log base 2 of 4KB! Why?

1 bit lets you index 2 positions: 0 and 1
2 bits let you index 4 positions: 0, 1, 2, 3
How many bits are needed to address an element inside an array of 4K elements?

Log base 2 of 4KB! Why?

1 bit lets you index 2 positions: 0 and 1
2 bits let you index 4 positions: 0, 1, 2, 3

Each subsequent bit doubles the number of positions you are able to index.

Keep doubling until you reach 4KB…. Log tells you exactly how many times to do that.
How many bits are needed to address an element inside an array of 4K elements?

Log base 2 of 4KB! And how much is that?

**Parameters of Paging System**

- Virtual address space size 32k
- Physical memory size 64k
- Page size 4k

The format of the page table is simple:
- The high-order (left-most) bit is the VALID bit.
- If the bit is 1, the rest of the entry is the PFN.
- If the bit is 0, the page is not valid.

**Page Table (from entry 0 down to the max size)**

<table>
<thead>
<tr>
<th>VPN</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x8000000a</td>
</tr>
<tr>
<td>1</td>
<td>0x8000000b</td>
</tr>
<tr>
<td>2</td>
<td>0x8000000c</td>
</tr>
<tr>
<td>3</td>
<td>0x80000002</td>
</tr>
<tr>
<td>4</td>
<td>0x80000000</td>
</tr>
<tr>
<td>5</td>
<td>0x80000009</td>
</tr>
<tr>
<td>6</td>
<td>0x00000000</td>
</tr>
<tr>
<td>7</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

**Virtual Address Trace**

For each virtual address, write down the physical address it translates to OR check the Invalid button.

VADDR 0x508c (decimal: 20620) PADDR 0x OR 
VADDR 0x4cd8 (decimal: 19672) PADDR 0x OR 
VADDR 0x40a6 (decimal: 16550) PADDR 0x OR 

Submit
How many bits are needed to address an element inside an array of 4K elements?

Log base 2 of 4KB! And how much is that?

4 KB = $2^2 \cdot 2^{10} = 2^{12}$

So 12 bits are needed just to be able to access anything inside a table: those are the "offset" bits.
How many virtual pages do we have?

Virtual address format:

12 bits
offset

Virtual Address Trace

For each virtual address, write down the physical address it translates to OR check the Invalid button.

VADDR 0x508c (decimal: 20620) PADDR 0x________ OR 0 Invalid
VADDR 0x4cd8 (decimal: 19672) PADDR 0x________ OR 0 Invalid
VADDR 0x40a6 (decimal: 16550) PADDR 0x________ OR 0 Invalid

Submit
Parameters of Paging System

Virtual address space size 32k
Physical memory size 64k
Page size 4k

The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
If the bit is 1, the rest of the entry is thePFN.
If the bit is 0, the page is not valid.

Page Table (from entry 0 down to the max size)

<table>
<thead>
<tr>
<th>VPN</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x8000000a</td>
</tr>
<tr>
<td>1</td>
<td>0x8000000b</td>
</tr>
<tr>
<td>2</td>
<td>0x8000000c</td>
</tr>
<tr>
<td>3</td>
<td>0x80000002</td>
</tr>
<tr>
<td>4</td>
<td>0x80000000</td>
</tr>
<tr>
<td>5</td>
<td>0x80000009</td>
</tr>
<tr>
<td>6</td>
<td>0x00000000</td>
</tr>
<tr>
<td>7</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

Virtual Address Trace

For each virtual address, write down the physical address it translates to OR check the Invalid button.

VADDR 0x508c (decimal: 20620) PADDR 0x OR  Invalid
VADDR 0x4cd8 (decimal: 19672) PADDR 0x OR  Invalid
VADDR 0x40a6 (decimal: 16550) PADDR 0x OR  Invalid

Virtual address format: 12 bits offset
How many virtual pages do we have?

8.

How many bits do we need to index an array of 8 elements?

Just like before, log base 2 of 8 = 3.
How many physical pages (page frames) do we have?

16.

How many bits do we need to index an array of 16 elements?

4.
Parameters of Paging System

Virtual address space size 32k
Physical memory size 64k
Page size 4k

0x508C in binary becomes…

0101 0000 1000 1100

The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
If the bit is 1, the rest of the entry is the PFN.
If the bit is 0, the page is not valid.

Page Table (from entry 0 down to the max size)

<table>
<thead>
<tr>
<th>VPN</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x8000000a</td>
</tr>
<tr>
<td>1</td>
<td>0x8000000b</td>
</tr>
<tr>
<td>2</td>
<td>0x8000000c</td>
</tr>
<tr>
<td>3</td>
<td>0x80000002</td>
</tr>
<tr>
<td>4</td>
<td>0x80000000</td>
</tr>
<tr>
<td>5</td>
<td>0x80000009</td>
</tr>
<tr>
<td>6</td>
<td>0x00000000</td>
</tr>
<tr>
<td>7</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

Virtual Address Trace

For each virtual address, write down the physical address it translates to OR check the Invalid button.

Virtual address format: 3 bits virtual page number 12 bits offset
Physical address format: 4 bits frame number 12 bits offset
Parameters of Paging System

Virtual address space size 32k
Physical memory size 64k
Page size 4k

The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
If the bit is 1, the rest of the entry is the PFN.
If the bit is 0, the page is not valid.

Page Table (from entry 0 down to the max size)

<table>
<thead>
<tr>
<th>VPN</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x8000000a</td>
</tr>
<tr>
<td>1</td>
<td>0x8000000b</td>
</tr>
<tr>
<td>2</td>
<td>0x8000000c</td>
</tr>
<tr>
<td>3</td>
<td>0x80000002</td>
</tr>
<tr>
<td>4</td>
<td>0x80000000</td>
</tr>
<tr>
<td>5</td>
<td>0x80000009</td>
</tr>
<tr>
<td>6</td>
<td>0x00000000</td>
</tr>
<tr>
<td>7</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

Virtual Address Trace

For each virtual address, write down the physical address it translates to OR check the Invalid button.

Virtual address format: 3 bits virtual page number 12 bits offset
Physical address format: 4 bits frame number 12 bits offset

0x508C in binary becomes…
0101 0000 1000 1100 ...Is this what we want to work with?
Parameters of Paging System

Virtual address space size 32k
Physical memory size 64k
Page size 4k

The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
If the bit is 1, the rest of the entry is the PFN.
If the bit is 0, the page is not valid.

Page Table (from entry 0 down to the max size)

<table>
<thead>
<tr>
<th>VPN</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x8000000a</td>
</tr>
<tr>
<td>1</td>
<td>0x8000000b</td>
</tr>
<tr>
<td>2</td>
<td>0x8000000c</td>
</tr>
<tr>
<td>3</td>
<td>0x80000002</td>
</tr>
<tr>
<td>4</td>
<td>0x80000000</td>
</tr>
<tr>
<td>5</td>
<td>0x80000009</td>
</tr>
<tr>
<td>6</td>
<td>0x00000000</td>
</tr>
<tr>
<td>7</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

Virtual Address Trace

For each virtual address, write down the physical address it translates to OR check the Invalid button.

Virtual address format:

<table>
<thead>
<tr>
<th>Virtual address format:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 bits virtual page number</td>
</tr>
<tr>
<td>12 bits offset</td>
</tr>
</tbody>
</table>

Physical address format:

<table>
<thead>
<tr>
<th>Physical address format:</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 bits frame number</td>
</tr>
<tr>
<td>12 bits offset</td>
</tr>
</tbody>
</table>
Parameters of Paging System

Virtual address space size 32k
Physical memory size 64k
Page size 4k

0x508C in binary becomes…

101 0000 1000 1100

The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
If the bit is 1, the rest of the entry is the PFN.
If the bit is 0, the page is not valid.

Page Table (from entry 0 down to the max size)

<table>
<thead>
<tr>
<th>VPN</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x800000a</td>
</tr>
<tr>
<td>1</td>
<td>0x800000b</td>
</tr>
<tr>
<td>2</td>
<td>0x800000c</td>
</tr>
<tr>
<td>3</td>
<td>0x8000002</td>
</tr>
<tr>
<td>4</td>
<td>0x8000000</td>
</tr>
<tr>
<td>5</td>
<td>0x8000009</td>
</tr>
<tr>
<td>6</td>
<td>0x0000000</td>
</tr>
<tr>
<td>7</td>
<td>0x0000000</td>
</tr>
</tbody>
</table>

First bit = 1 (0x8 = 1000) => page is valid

Virtual Address Trace

For each virtual address, write down the physical address it translates to OR check the Invalid button.

Virtual address format:

3 bits virtual page number
12 bits offset

Physical address format:

4 bits frame number
12 bits offset
0x508C in binary becomes…

101 0000 1000 1100

Page Table (from entry 0 down to the max size)

<table>
<thead>
<tr>
<th>VPN</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x8000000a</td>
</tr>
<tr>
<td>1</td>
<td>0x8000000b</td>
</tr>
<tr>
<td>2</td>
<td>0x8000000c</td>
</tr>
<tr>
<td>3</td>
<td>0x80000002</td>
</tr>
<tr>
<td>4</td>
<td>0x80000000</td>
</tr>
</tbody>
</table>
| 5   | 0x80000009| First bit = 1 ( 0x8 = 1000) => page is valid
| 6   | 0x00000000|
| 7   | 0x00000000|

Virtual Address Trace

For each virtual address, write down the physical address it translates to OR check the Invalid button.

Virtual address format:

<table>
<thead>
<tr>
<th></th>
<th>3 bits</th>
<th>12 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual page number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical address format:

<table>
<thead>
<tr>
<th></th>
<th>4 bits</th>
<th>12 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parameters of Paging System

Virtual address space size 32k
Physical memory size 64k
Page size 4k

The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
If the bit is 1, the rest of the entry is the PFN.
If the bit is 0, the page is not valid.

Page Table (from entry 0 down to the max size)

<table>
<thead>
<tr>
<th>VPN</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x8000000a</td>
</tr>
<tr>
<td>1</td>
<td>0x8000000b</td>
</tr>
<tr>
<td>2</td>
<td>0x8000000c</td>
</tr>
<tr>
<td>3</td>
<td>0x80000002</td>
</tr>
<tr>
<td>4</td>
<td>0x80000000</td>
</tr>
<tr>
<td>5</td>
<td>0x80000009</td>
</tr>
<tr>
<td>6</td>
<td>0x00000000</td>
</tr>
<tr>
<td>7</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

First bit = 1 (0x8 = 1000) => page is valid
Last four bits = 1001 (0x9) => frame number

0x508C in binary becomes…

101 0000 1000 1100

Physical address is:

1001 0000 1000 1100

0x908C is the answer

Virtual Address Trace

For each virtual address, write down the physical address it translates to OR check the Invalid button.

Virtual address format: 3 bits virtual page number 12 bits offset

Physical address format: 4 bits frame number 12 bits offset
Multi-level translation
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

| VALID | PFN6 ... PFN0 |

Format of PDE:

| VALID | PT6 ... PT0 |

PDBR: 100 (decimal) [This means the page directory is held in this page frame]
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

| VALID | PFN6 ... | PFN0 |

Format of PDE:

| VALID | PT6 ... | PT0 |

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

How many bits for the page offset?

Virtual address format:

Physical address format:
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

| VALID | PFN6 ... PFN0 |

Format of PDE:

| VALID | PT6 ... PT0 |

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

How many bits for the page offset?
5.

Virtual address format:

Physical address format:
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

| VALID | PFN6 ... PFN0 |

Format of PDE:

| VALID | PT6 ... PT0 |

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

We have 1024 virtual pages. How many bits do we need to index those?

Virtual address format: 5 bits
 offset

Physical address format: 5 bits
 offset
We have 1024 virtual pages. How many bits do we need to index those?

Virtual address format:

| VALID | PFN6 ... PFN0 |

Physical address format:

| VALID | PT6 ... PT0 |

PDBR: 100 (decimal) [This means the page directory is held in this page frame]
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

|VALID|PFN6 . . . PFN0|

Format of PDE:

|VALID|PT6 . . . PT0|

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

We have 1024 virtual pages.
How many bits do we need to index those?

10.

Virtual address format:

<table>
<thead>
<tr>
<th>10 bits virtual page number</th>
</tr>
</thead>
</table>

Physical address format:

<table>
<thead>
<tr>
<th>5 bits offset</th>
</tr>
</thead>
</table>
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)

Format of PTE:

| VALID | PFN6 ... PFN0 |

Format of PDE:

| VALID | PT6 ... PT0 |

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Virtual address format: 10 bits virtual page number 5 bits offset

Physical address format: 7 bits frame number 5 bits offset

We have 128 page frames. How many bits do we need to index those?

7.
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

|V|PT6|PT5|PT4|PT3|PT2|PT1|PT0|

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

The system uses a two-level page table. Thus, the upper five bits of a virtual address are used to index into a page directory.

Virtual address format:

5 bits page directory

Physical address format:

7 bits frame number

5 bits offset
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

|VALID|PFN6 ... PFN0|

Format of PDE:

|VALID|PT6 ... PT0|

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Translate virtual address 0x543f:

Virtual address format: 5 bits page directory 5 bits page table 5 bits offset

Physical address format: 7 bits frame number 5 bits offset
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

|VALID|PFN6 ... PFN0|

Format of PDE:

|VALID|PT6 ... PT0|

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Translate virtual address 0x543f:

In binary: 0101 0100 0011 1111

Virtual address format: 5 bits 5 bits 5 bits
page directory page table offset

Physical address format: 7 bits 5 bits
frame offset number
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

| VALID | PFN6 ... | PFN0 |

Format of PDE:

| VALID | PT6 ... | PT0 |

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Translate virtual address 0x543f:

In binary: 0101 0100 0011 1111
Our addresses are 15 bits, so get rid of the leading zero.

Virtual address format: 5 bits page directory 5 bits page table 5 bits offset

Physical address format: 7 bits frame number 5 bits offset
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

|VALID|PFN6 ... PFN0|

Format of PDE:

|VALID|PT6 ... PT0|

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Translate virtual address 0x543f:

In binary: 101 0100 0011 1111
Our addresses are 15 bits, so get rid of the leading zero.
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

| VALID | PFN6 ... | PFN0 |

Format of PDE:

| VALID | PT6 ... | PT0 |

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Virtual address format:

<table>
<thead>
<tr>
<th>page directory</th>
<th>page table</th>
<th>offset</th>
</tr>
</thead>
</table>

Physical address format:

<table>
<thead>
<tr>
<th>frame number</th>
<th>offset</th>
</tr>
</thead>
</table>

Translate virtual address 0x543f:

In binary: 101 0100 0011 1111
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:
|VALID|PFN6 ... PFN0|

Format of PDE:
|VALID|PT6 ... PT0|

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Virtual address format: 5 bits page directory 5 bits page table 5 bits offset
Physical address format: 7 bits frame number 5 bits offset

Translate virtual address 0x543f:
In binary: 101 0100 0011 1111
We need to look at directory entry 101 01 = 21
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:
|VALID|PFN6 ... PFN0|

Format of PDE:
|VALID|PT6 ... PT0|

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Translate virtual address 0x543f:

In binary: 101 0100 0011 1111
We need to look at directory entry 101 01 = 21

Where is the directory located? In which frame?
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:
|VALID|PFN6 ... PFN0|

Format of PDE:
|VALID|PT6 ... PT0|

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Virtual address format:  
5 bits page directory  
5 bits page table  
5 bits offset

Physical address format:  
7 bits frame number  
5 bits offset

Translate virtual address 0x543f:

In binary: 101 0100 0011 1111
We need to look at directory entry 101 01 = 21

Where is the directory located? In which frame?
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

| VALID | PF6 \ldots | PF0 |

frame 100: 7f \ldots 7f \ldots f1 7f 7f 7f cd 7f cd e6 d9 7f 7f f7 f7 fa a2 f0 7f 7f ae db 7f 7f 92 ee 7f 7f 93 ed 7f 7f

Format of PDE:

| VALID | PT6 \ldots | PT0 |

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Translate virtual address 0x543f:

In binary: 101 0100 0011 1111

We need to look at directory entry 101 01 = 21
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

|VALID|PFN6 ... PFN0|

Format of PDE:

|VALID|PT6 ... PT0|

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Translate virtual address 0x543f:

In binary: \( 101 \ 0100 \ 0011 \ 1111 \)
We need to look at directory entry \( 101 \ 01 = 21 \)
We found \( db = 1101 \ 1011 \)
So this is a valid frame (first bit)
And the frame containing the page table we're looking for is: \( 101 \ 1011 = 91 \)
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

|VALID|PFN6 ... PFN0|

Virtual address format: 5 bits page directory 5 bits page table 5 bits offset
Physical address format: 7 bits frame number 5 bits offset

Translate virtual address 0x543f:

In binary: 101 0100 0011 1111
We need to look at directory entry 101 01 = 21
Frame containing the page table we’re looking for is: 101 1011 = 91

PDBR: 100 (decimal) [This means the page directory is held in this page frame]
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

|VALID|PFN6 ... PFN0|

Translate virtual address 0x543f:
In binary: 101 0100 0011 1111
We need to look at directory entry 101 01 = 21
Frame containing the page table we’re looking for is: 101 1011 = 91

Format of PTE:

|VALID|PT6 ... PT0|

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Virtual address format:

5 bits page directory
5 bits page table
5 bits offset

Physical address format:

7 bits frame number
5 bits offset
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

| VALID | PFN6 ... PFO |

Format of PDE:

| VALID | PT6 ... PT0 |

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Virtual address format: 5 bits page directory 5 bits page table 5 bits offset

Physical address format: 7 bits frame number 5 bits offset

Translate virtual address 0x543f:

In binary: 101 0100 0011 1111
We need to look at directory entry 101 01 = 21
Frame containing the page table we’re looking for is: 101 1011 = 91

We found 0XFC = 1111 1100.
First bit says it’s valid. Other seven bits tell us....
Translate virtual address 0x543f:

In binary: 101 0100 0011 1111
We need to look at directory entry 101 01 = 21
Frame containing the page table we’re looking for is: 101 1011 = 91

We found 0XFC = 1111 1100.
First bit says it’s valid. Other seven bits tell us….
Frame number = 111 1100.

Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

|VALID|PFN6  ...  PFN0|

Format of PDE:

|VALID|PT6  ...  PT0|

PDBR: 100 (decimal) [This means the page directory is held in this page frame]
Parameters of the Paging System

Virtual address space size 32768
Number of virtual pages 1024
Physical memory size 4096
Page size 32
Number of physical page frames 128
Size of PTE 1 byte(s)
Format of PTE:

| VALID | PFN6 ... PFN0 |

Format of PDE:

| VALID | PT6 ... PT0 |

PDBR: 100 (decimal) [This means the page directory is held in this page frame]

Translate virtual address 0x543f:

In binary: 101 0100 0011 1111
Frame number = 111 1100 (124)
Physical address in binary: 111 1100 1 1111 = 0xF9F

Virtual address format: 5 bits page directory 5 bits page table 5 bits offset
Physical address format: 7 bits frame number 5 bits offset
A3
Basics

- You are given a sequence a **trace:**
  - A sequence of virtual addresses
- **Goal: simulate a page replacement algorithm**
  - Given a vaddr: return the position in memory where the respective page is.
  - If “physical memory” is full: choose a page to swap out.

```
vaddr1 vaddr2 vaddr3 ...
  \downarrow           \downarrow           \downarrow
find_pyhspage(vaddr1) find_pyhspage(vaddr3)
                          \downarrow
find_pyhspage(vaddr2)
```
Data structures

- Coremap: an array of frames (pagetable.h)
  - Represents everything that you can store in actual memory.
  - Each frame is either in use or not by some page.
  - If in use, which page is using it?

```c
struct frame {
    char in_use;  // True if frame is allocated, False if frame is free
    pgtbl_entry_t *pte; // Pointer back to pagetable entry (pte) for page
        // stored in this frame
};

/* The coremap holds information about physical memory.
 * The index into coremap is the physical page frame number stored
 * in the page table entry (pgtbl_entry_t).
 */
extern struct frame *coremap;
```
Data structures

- The page itself: (pagetable.h)
  - If valid: then where is it in physical memory? The integer will tell you how to find it.
  - Might be in swap.

```c
struct frame {
    char in_use; // True if frame is allocated, False if frame is free
    pgtbl_entry_t *pte; // Pointer back to pagetable entry (pte) for page
    // stored in this frame
};

typedef struct {
    unsigned int frame; // if valid bit == 1, physical frame holding vpage
    off_t swap_off; // offset in swap file of vpage, if any
} pgtbl_entry_t;
```
Data structures

● The page itself: (pagetable.h)
  ○ If valid: then where is it in physical memory? The integer will tell you how to find it.
  ○ Might be in swap.

```c
typedef struct {
    unsigned int frame;
    off_t swap_off;
} pgtbl_entry_t;
```

This is calculating an “offset” from the beginning of physical memory:

1. p->frame >> PAGE_SHIFT is removing some of the bits containing extra information.
2. The multiplication is “jumping over” previous pages.
```c
char *find_physpage(addr_t vaddr, char type) {
    pgtbl_entry_t *p=NULL; // pointer to the full page table entry for vaddr
    unsigned idx = PGDIR_INDEX(vaddr); // get index into page directory

    // IMPLEMENTATION NEEDED
    // Use top-level page directory to get pointer to 2nd-level page table
    (void)idx; // To keep compiler happy - remove when you have a real use.

    // Use vaddr to get index into 2nd-level page table and initialize 'p'

    // Check if p is valid or not, on swap or not, and handle appropriately

    // Make sure that p is marked valid and referenced. Also mark it
    // dirty if the access type indicates that the page will be written to.

    // Call replacement algorithm's ref_fcn for this page
    ref_fcn(p);

    // Return pointer into (simulated) physical memory at start of frame
    return &physmem[(p->frame >> PAGE_SHIFT)*SIMPAGESIZE];
}
```
Data structures

Index in the top level table:

```c
char *find_physpage(addr_t vaddr, char type) {
pgtbl_entry_t *p=NULL; // pointer to the full page table entry for vaddr
unsigned idx = PGDIR_INDEX(vaddr); // get index into page directory

// IMPLEMENTATION NEEDED
// Use top-level page directory to get pointer to 2nd-level page table
(void)idx; // To keep compiler happy - remove when you have a real use.

// Use vaddr to get index into 2nd-level page table and initialize 'p'

// Check if p is valid or not, on swap or not, and handle appropriately

// Make sure that p is marked valid and referenced. Also mark it
// dirty if the access type indicates that the page will be written to.

// Call replacement algorithm's ref_fcn for this page
ref_fcn(p);

// Return pointer into (simulated) physical memory at start of frame
return &physmem[(p->frame >> PAGE_SHIFT)*SIMPAGESIZE];
}
```

It is essentially an array of pointers to the second level table: some bits might have information extra info (check handout!)
Data structures

It is essentially an array of pointers to the second level table: some bits might have information extra info (check handout!)

The first thing you’ll do is to access this table, so that you can then access the second level table. Careful: might need to initialize it!
Data structures

```c
char *find_physpage(addr_t vaddr, char type) {
    pgtbl_entry_t *p=NULL; // pointer to the full page table entry for vaddr
    unsigned idx = PGDIR_INDEX(vaddr); // get index into page directory

    // IMPLEMENTATION NEEDED
    // Use top-level page directory to get pointer to 2nd-level page table
    (void)idx; // To keep compiler happy - remove when you have a real use.

    // Use vaddr to get index into 2nd-level page table and initialize 'p'

    // Check if p is valid or not, on swap or not, and handle appropriately

    // Make sure that p is marked valid and referenced. Also mark it
    // dirty if the access type indicates that the page will be written to.

    // Call replacement algorithm's ref_fcn for this page
    ref_fcn(p);

    // Return pointer into (simulated) physical memory at start of frame
    return &physmem[(p->frame >> PAGE_SHIFT)*SIMPAGESIZE];
}
```

Is the page in the core map? If not and if the core map is full, call eviction algorithm to make space for it!
Every replacement policy has a function that gets called on every access: the “on reference” function.
Replacement algorithms

extern void rand_init();
extern void lru_init();
extern void clock_init();
extern void fifo_init();
extern void opt_init();

// These may not need to do anything for some algorithms
extern void rand_ref(pgtbl_entry_t *);
extern void lru_ref(pgtbl_entry_t *);
extern void clock_ref(pgtbl_entry_t *);
extern void fifo_ref(pgtbl_entry_t *);
extern void opt_ref(pgtbl_entry_t *);

extern int rand_evict();
extern int lru_evict();
extern int clock_evict();
extern int fifo_evict();
extern int opt_evict();
Replacement algorithms

Initialization functions: maybe set clock hand to its initial position, maybe do nothing at all, or maybe do some magic for opt (wink wink)...

```c
extern void rand_init();
extern void lru_init();
extern void clock_init();
extern void fifo_init();
extern void opt_init();

// These may not need to do anything for some algorithms
extern void rand_ref(pgtbl_entry_t *);
extern void lru_ref(pgtbl_entry_t *);
extern void clock_ref(pgtbl_entry_t *);
extern void fifo_ref(pgtbl_entry_t *);
extern void opt_ref(pgtbl_entry_t *);

extern int rand_evict();
extern int lru_evict();
extern int clock_evict();
extern int fifo_evict();
extern int opt_evict();
```
Replacement algorithms

Initialization functions: maybe set clock hand to its initial position, maybe do nothing at all, or maybe do some magic for opt (wink wink)...

As we saw: called on every reference to a page that is in the core map, i.e., in physical memory. For instance: LRU might want to update “the time” of the last access to the page being referenced.
Replacement algorithms

Initialization functions: maybe set clock hand to its initial position, maybe do nothing at all, or maybe do some magic for opt (wink wink)...

If the core map is full, these will make space for a new page: returns the index of the frame to be evicted.

As we saw: called on every reference to a page that is in the core map, i.e., in physical memory.

For instance: LRU might want to update “the time” of the last access to the page being referenced.

If the core map is full, these will make space for a new page: returns the index of the frame to be evicted.

For instance: OPT will look into the future and see which of the pages in the coremap will be referenced latest.