Virtual Memory
Virtual Memory

* ISA has large VA space.
  - Allows user to uniquely identify lots
* Physical Memory is smaller
  - Cost issue

* Virtual Memory Management
  - Access Control
  - Translation

* The VAX Model
THREE CONCEPTS

A Process

- The Executable Image (Process Part)
  - Consists of n pages
  - All are on the disk
  - Some (Working Set) are in frames
  - The OS needs to know which pages occupy physical frames.
  - The page table (one per process)
    - Provides that mechanism

The Page Table

- The page table consists of PTEs,
  - One PTE for each page
  - The page tables for all processes are in system virtual space
  - The OS keeps track of the page table via the base address of the page table (i.e., the PBR)

PSR

- Each process has a hardware process control block
  - Intel calls it a Task State Segment.
  - It contains all the state information that the OS needs to control the process
  - I loaded just prior to executing of a process
THE PROCESS EXECUTES LD R1, X

How does the OS figure out where virtual addr X is in physical memory?

1) The OS needs to find the PTE of the page containing X.
   VA of X: 00 Page i Byte on page

2) Knowing the page number (in this case i) allows us to figure out the address in system space of the PTE of page i.

   PBR → PTE of Page 0
       " " 1
       :
      PTE of Page i
       :

   That is, the VA of the PTE for page i is PBR + 4 * i
   BUT THIS ADDRESS IS VIRTUAL

   VA of the PTE of Page i: 10 Page k Byte on page
   (in Page k)

3) Since we know the page number in systems space that contains the PTE of page i, I can find the frame that page occupies by looking at the system page table. The system page table is in physical memory.
That is, the physical addr of the PTE of page k is

\[ \text{SBR} + 4 \times k \]

4. The O/S goes to that addr and gets the PTE of Page k,

\[
\text{PTE of Page } k: \begin{array}{c} 1 \end{array} \begin{array}{c} PFN \end{array} \begin{array}{c} \text{Bytes on page} \end{array} \]

\[ \text{PA of PTE of Page } i \]

5. The O/S goes to that address and gets PTE of Page i,

\[
\text{PTE of Page } i: \begin{array}{c} 1 \end{array} \begin{array}{c} PFN \end{array} \begin{array}{c} \text{Bytes on page} \end{array} \]

\[ \text{PA of } x \]

6. The O/S goes to that address and accesses x.
Page Tables

* One for each region

* For example, the P0 Page Table

<table>
<thead>
<tr>
<th></th>
<th>PTE for Page 0</th>
<th>P0 Base Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PTE for Page 1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PTE for Page 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>P0 Length Register</td>
</tr>
<tr>
<td>L-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Sequentially stored in System Virtual Space

* P0LR used for ACV checks

* PTE used for ACV, TNV checks
The Abstraction

Page No.
Which Page Table
Example: 00 P0PT

1. Is Page No. < P0LR?
   NO: ACV Fault!

2. Get Correct PTE
   Index by Page No.
   PTE for Page 1

3. The PTE
   Check Protection
   NO: ACV Fault!

4. Is Page Resident (i.e. V = 1)
   NO: TNV Fault!

5. All Cool: PA
The PTE

31  V  Prot  M  PFN

Can I Believe The PFN

Has the Page Been Written

In which frame is the Page Stored

Do I have the Right to do this access

\{NO,R,R/W\} \{K,E,S,U\}

81 Possibilities in 4 bits

Note:

No Ref. Bit!
VA of x

PXBR

SBR

PTE of Page Containing X

PTE of Page of System Space

PTE of Page Containing X

PA of x

And now we can access x
Access Control and Translate

VA | PAGE NO | BYTE ON PAGE

TRANSLATE

USE PTE TO TRANSLATE

PA | FRAME NO

ACCESS CONTROL

VA | PAGE NUMBER

PTE

V | PROT | PFN

4

(KESU) PRIVILEGE

(R/W, R, N) ACCESS

COMB LOGIC

YES/NO
ONE FINAL EXAMPLE

Let's modify the VAX ISA to make it easier to see what is going on. We will retain the essential elements, but we will reduce all the numbers.

For example: VA will go from 32 bits to 9 bits.
Page size will go from 512 B to 16 B.
PA will be 7 bits.
PTE will still be 4 bytes.

Page 2 shows a snapshot of virtual / physical memory. Several things are worth noting:

1. Virtual memory = 512 bytes. :: 32 pages possible.
P0 has a max of 8. In our example: 6 pages were needed.
P1 " " " " " " In our example: 2 pages " "
P5 " " " " " " In our example: 5 pages " "

(P0,BR)

2. P0 Page Table starts at VA = 120 (Note: 2, 0 are hex digits). Since there must be 6 entries, the P0 page table consumes all of page 2, half of page 3 of 5.

SBR

3. System Page Table starts at PA = 50 (Note: 5 is octal, 0 is hex). Since there must 5 entries, system page table consumes all of frame 5 and 1/4 of frame 6.

4. Note that sys page table indicates 3 pages resident (pages 0, 1).
Note that part of P0 page table indicates page 5 in, page 4 out.
**Finally, a page table computation:**

& We wish to execute: `LD R1, X`

Where `X` has VA: `001011000`

1. VA of `X`: `001011000`  
   - Byte 8 on Page 5
   - Because PTE contains 4 bytes
   - `101000`  
   - `PBR + 4` to index into `PBR` page table
   - `1001000000`

2. VA of PTE of Page 5, PBR space (which is on Page 3 of virtual system space)
   - `1001000010 1B00`
   - SS  
   - `Page 3`  
   - `X4`  
   - `1001000010 1B00`  
   - `SBR + 4`

3. VA of PTE of Page 3, system space  
   - `1010001100`
   - `PA of PTE of Page 3, system space`
   - `101000 1100`
   - `PA of PTE of Page 3, system space`
   - PTE of Page 3 of system space: it occupies PFN 1

   - VA of PTE of Page 5 of PBR space `101011000` is mapped to PA `001101000`
So, if we look at PFN 1

```
0
1
2
```

We can read the PTE of page 5, PF space. We see page 5, PF space occupies PFN 2

The VA of X \[001011000\] therefore maps to

PA \[01010001\]

We look in that location and find 17, which we load into R1.
**TLB Structure**

1. **Page No.** 10 bits
   - (Hashing Function)
2. **TLB**
   - Page No.
   - PTE

**Examples:**

- 16 Entry TLB
- Page No. 10 Bits
- 4 Bits Used To Hash Into TLB.
- TLB is a CAM
IA-32

Actually, Intel calls it Linear Address
Flat Model

Read Address

Segmented Model
Segment Registers

16 bits

TI (which tells)

GDT

GDTR

LDT

LDTR

$2^{13} \times 8$ bytes each

Segment Descriptors

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE [31:24]</td>
<td>8 bits</td>
</tr>
<tr>
<td>BASE [15:0]</td>
<td>16 bits</td>
</tr>
<tr>
<td>LIMIT [15:0]</td>
<td>16 bits</td>
</tr>
<tr>
<td>P</td>
<td>Segment Present</td>
</tr>
<tr>
<td>PL</td>
<td>Privilege Level</td>
</tr>
<tr>
<td>Type</td>
<td>Segment type</td>
</tr>
</tbody>
</table>

G: Granularity 1 byte/4k bytes
P: Segment Present
PL: Privilege Level
Type: Segment type
Segmentation AND Paging

A is Logical Address

Virtual translation enabled

PTE

Priv: S U R W W

Linear Address

Frame of Physical Mem.

Page Table

Page Directory
Task State Segment (TSS)

<table>
<thead>
<tr>
<th>I/O Map</th>
<th>0</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>LDT</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDI</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EAX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ESP, EBP, ESI, EDI</td>
<td></td>
</tr>
<tr>
<td>CR3</td>
<td>SS 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ESP, SS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ESP, SS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ESP, SS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ESP, SS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CR3 Back Link</td>
<td></td>
</tr>
</tbody>
</table>

Segment Registers

CS
X86 Extension to 3 Page Sizes