Virtual addresses

- Virtual memory addresses (what the process uses)
  - Page number plus byte offset in page
  - Low order n bits are the byte offset
  - Remaining high order bits are the page number

Example: 32 bit virtual address
Page size = $2^{12} = 4\text{KB}$
Address space size = $2^{32}$ bytes = $4\text{GB}$
Physical addresses

- Physical memory addresses (what memory uses)
  - Page “frame” number plus byte offset in page
  - Low order n bits are the byte offset
  - Remaining high order bits are the page frame number

Example: 24 bit physical address
Page frame size = $2^{12} = 4$KB
Max physical memory size = $2^{24}$ bytes = 16MB
Address translation

- Complete set of address mappings for a process are stored in a page table in memory
  - But accessing the table for every address translation is too expensive
  - So hardware support is used to map page numbers to frame numbers at full CPU speed
    - Memory management unit (MMU) has multiple registers for multiple pages and knows how to access page tables
    - Also called a translation look aside buffer (TLB)
    - Essentially a cache of page table entries
The BLITZ architecture

- The page table mapping:
  - Page --> Frame

- Virtual Address (24 bit in Blitz):

- Physical Address (32 bit in Blitz):
The BLITZ page table

- An array of "page table entries"
  - Kept in memory

- $2^{11}$ pages in a virtual address space
  - $\rightarrow$ 2K entries in the table

- Each entry is 4 bytes long
  - 19 bits The Frame Number
  - 1 bit Valid Bit
  - 1 bit Writable Bit
  - 1 bit Dirty Bit
  - 1 bit Referenced Bit
  - 9 bits Unused (and available for OS algorithms)
The BLITZ page table

- Two page table related registers in the CPU
  - Page Table Base Register
  - Page Table Length Register
- These define the page table for the “current” process
  - Must be saved and restored on process context switch
- Bits in the CPU “status register”
  - “System Mode”
  - “Interrupts Enabled”
  - “Paging Enabled”
    - 1 = Perform page table translation for every memory access
    - 0 = Do not do translation
The BLITZ page table

- A page table entry

Diagram showing a page table entry with fields for frame number, unused, dirty bit, referenced bit, writable bit, and valid bit.
The BLITZ page table

- The full page table

<table>
<thead>
<tr>
<th>frame number</th>
<th>unused</th>
<th>D</th>
<th>R</th>
<th>W</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Indexed by the page number
**The BLITZ page table**

<table>
<thead>
<tr>
<th>frame number</th>
<th>unused</th>
<th>D</th>
<th>R</th>
<th>W</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The BLITZ page table

<table>
<thead>
<tr>
<th>page number</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>1312</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>frame number</th>
<th>unused</th>
<th>D</th>
<th>R</th>
<th>W</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page table base register

Virtual address

Physical address
The BLITZ page table

<table>
<thead>
<tr>
<th>page number</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>1312</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>frame number</th>
<th>unused</th>
<th>D</th>
<th>R</th>
<th>W</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>D</td>
<td>R</td>
<td>W</td>
<td>V</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>D</td>
<td>R</td>
<td>W</td>
<td>V</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>D</td>
<td>R</td>
<td>W</td>
<td>V</td>
</tr>
<tr>
<td>2K</td>
<td></td>
<td>D</td>
<td>R</td>
<td>W</td>
<td>V</td>
</tr>
</tbody>
</table>

virtual address

<table>
<thead>
<tr>
<th>virtual address</th>
<th>physical address</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>1312</td>
</tr>
</tbody>
</table>
The BLITZ page table

![Diagram of the BLITZ page table]

- Page table base register
- Virtual address
- Page number
- Offset

<table>
<thead>
<tr>
<th>Frame Number</th>
<th>Unused</th>
<th>D</th>
<th>R</th>
<th>W</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>unused</td>
<td>D</td>
<td>R</td>
<td>W</td>
<td>V</td>
</tr>
<tr>
<td>2</td>
<td>unused</td>
<td>D</td>
<td>R</td>
<td>W</td>
<td>V</td>
</tr>
<tr>
<td>2K</td>
<td>unused</td>
<td>D</td>
<td>R</td>
<td>W</td>
<td>V</td>
</tr>
</tbody>
</table>

Physical address

<table>
<thead>
<tr>
<th>Physical Address</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>1312</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
# The BLITZ page table

The BLITZ page table is a data structure used in computer memory management to map virtual addresses to physical addresses. It consists of two main components: the page table base register and the page table itself.

### Page Table Base Register

The page table base register contains the starting address of the page table. In this example, the page table base register is at virtual address 0x1312.

### Page Table

The page table is a 2K frame (2048 frame numbers) table. Each entry in the page table contains a frame number and flags indicating whether the frame is present (P), readable (R), writable (W), and executable (X). The flags are represented by 'D' for dirty and 'V' for valid.

<table>
<thead>
<tr>
<th>Frame Number</th>
<th>Unused</th>
<th>D</th>
<th>R</th>
<th>W</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2K</td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The page number and offset are combined to form the virtual address, which is then mapped to a physical address using the page table.

### Virtual Address

The virtual address is a 32-bit number that is composed of a page number and an offset. The page number is the highest 12 bits of the virtual address, and the offset is the lowest 20 bits.

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Page Number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1312</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The physical address is calculated by multiplying the page number by the page size (4K) and adding the offset.

<table>
<thead>
<tr>
<th>Physical Address</th>
<th>Frame Number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1280</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Page tables

- When and why do we access a page table?
  - On every instruction to translate virtual to physical addresses?
Page tables

- When and why do we access a page table?
  - On every instruction to translate virtual to physical addresses? NO!
  - In real machines it is only accessed
    - On TLB miss faults to refill the TLB
    - During process creation and destruction
    - When a process allocates or frees memory?
Translation Lookaside Buffer (TLB)

- **Problem:**
  - MMU can’t go to page table on every memory access!
Translation Lookaside Buffer (TLB)

- **Problem:**
  - MMU can’t go to page table on every memory access!

- **Solution:**
  - Cache the page table entries in a hardware cache
  - Small number of entries (e.g., 64)
  - Each entry contains
    - Page number
    - Other stuff from page table entry
  - Associatively indexed on page number (i.e., You can do a lookup in a single cycle)
Translation lookaside buffer
Hardware operation of TLB

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Frame Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>37</td>
<td>unused</td>
</tr>
<tr>
<td>17</td>
<td>50</td>
<td>unused</td>
</tr>
<tr>
<td>92</td>
<td>24</td>
<td>unused</td>
</tr>
<tr>
<td>5</td>
<td>19</td>
<td>unused</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>unused</td>
</tr>
</tbody>
</table>
## Hardware operation of TLB

### Virtual Address

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Frame Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>37</td>
<td>unused</td>
</tr>
<tr>
<td>17</td>
<td>50</td>
<td>unused</td>
</tr>
<tr>
<td>92</td>
<td>24</td>
<td>unused</td>
</tr>
<tr>
<td>5</td>
<td>19</td>
<td>unused</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>unused</td>
</tr>
</tbody>
</table>

### Physical Address

<table>
<thead>
<tr>
<th>Frame Number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1312</td>
<td>0</td>
</tr>
</tbody>
</table>
Hardware operation of TLB

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Frame Number</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>37</td>
<td>unused</td>
</tr>
<tr>
<td>17</td>
<td>50</td>
<td>unused</td>
</tr>
<tr>
<td>92</td>
<td>24</td>
<td>unused</td>
</tr>
<tr>
<td>5</td>
<td>19</td>
<td>unused</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>unused</td>
</tr>
</tbody>
</table>

Key

Physical address

Virtual address

Frame number

Offset

Page number

Offset
### Hardware operation of TLB

#### Virtual Address

- **Page Number**: 23
- **Frame Number**: 37
- **Other**: unused, D, R, W, V

#### Physical Address

- **Frame Number**: 23
- **Offset**: 37

#### Key

- **Virtual Address**: 1312
- **Physical Address**: 31
Hardware operation of TLB

![Diagram of TLB operation]

- **Virtual Address**: 0x1312 0x0
- **Page Number**: 23
- **Frame Number**: 37
- **Other Fields**: Unused, D, R, W, V

- **Physical Address**: 0x31 0x1312 0x0
- **Frame Number**: 37
- **Offset**: 0x0

- **Key**: Virtual Address Map

- **Table**: Page Number, Frame Number, Other
  - 23 | 37 | unused D R W V
  - 17 | 50 | unused D R W V
  - 92 | 24 | unused D R W V
  -  5 | 19 | unused D R W V
  - 12 |  6 | unused D R W V
Hardware operation of TLB

![Diagram showing the relationship between virtual and physical addresses, page numbers, and frame numbers.]

Key:

- **Page Number**
- **Frame Number**
- **Other**
  - Unused
  - D
  - R
  - W
  - V

The diagram illustrates the process of mapping virtual addresses to physical addresses using the TLB. Each entry in the table corresponds to a page in memory, with the page number and frame number indicating where the page resides in physical memory. The other fields (D, R, W, V) represent various permissions and access control settings for the page.
Software operation of TLB

- **What if the entry is not in the TLB?**
  - Go look in the page table in memory
  - Find the right entry
  - Move it into the TLB
  - But which TLB entry should be replaced?
Software operation of TLB

- **Hardware TLB refill**
  - Page tables in specific location and format
  - TLB hardware handles its own misses
  - Replacement policy fixed by hardware

- **Software refill**
  - Hardware generates trap (**TLB miss fault**)
  - Lets the OS deal with the problem
  - Page tables become entirely a OS data structure!
  - Replacement policy managed in software
Software operation of TLB

- What should we do with the TLB on a context switch?
- How can we prevent the next process from using the last process’s address mappings?
  - Option 1: empty the TLB
    - New process will generate faults until its pulls enough of its own entries into the TLB
  - Option 2: just clear the “Valid Bit”
    - New process will generate faults until its pulls enough of its own entries into the TLB
  - Option 3: the hardware maintains a process id tag on each TLB entry
    - Hardware compares this to a process id held in a specific register ... on every translation
Page tables

- Do we access a page table when a process allocates or frees memory?
Page tables

- Do we access a page table when a process allocates or frees memory?
  - Not necessarily
  - Library routines (malloc) can service small requests from a pool of free memory already allocated within a process address space
  - When these routines run out of space a new page must be allocated and its entry inserted into the page table
    - This allocation is requested using a system call
Page table design issues

- **Page table size depends on**
  - Page size
  - Virtual address length

- **Memory used for page tables is overhead!**
  - How can we save space?
  - ... and still find entries quickly?

- **Three options**
  - Single-level page tables
  - Multi-level page tables
  - Inverted page tables
Single-level page tables

A Virtual Address (32 bit):

<table>
<thead>
<tr>
<th>20-bits</th>
<th>12-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>page number</td>
<td>offset</td>
</tr>
</tbody>
</table>

frames in memory

green:
- Single-level page table
- Frames in memory

red:
- Single-level page tables
Single-level page tables

20-bits
page number

12-bits
offset

frames in memory

Single-level page table
Single-level page tables

Problem: requires one page table entry per virtual page!
Single-level page tables

- 20-bits: page number
- 12-bits: offset

Frames in memory

Single-level page table

32 bit addresses and 4KB pages means $2^{20}$ page table entries per process
Multi-level page tables
Multi-level page tables

- A Virtual Address:

  10-bits  10-bits  12-bits
   PT1     PT2    offset

Top-level Page table

2nd-level tables

frames in memory
Multi-level page tables

- A Virtual Address:

<table>
<thead>
<tr>
<th>10-bits</th>
<th>10-bits</th>
<th>12-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT1</td>
<td>PT2</td>
<td>offset</td>
</tr>
</tbody>
</table>

- Frames in memory

- Top-level Page table

- 2nd-level tables
Multi-level page tables

- A Virtual Address:

<table>
<thead>
<tr>
<th>10-bits</th>
<th>10-bits</th>
<th>12-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT1</td>
<td>PT2</td>
<td>offset</td>
</tr>
</tbody>
</table>

frames in memory

Top-level Page table

2nd-level tables
Multi-level page tables

A Virtual Address:

<table>
<thead>
<tr>
<th>10-bits</th>
<th>10-bits</th>
<th>12-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT1</td>
<td>PT2</td>
<td>offset</td>
</tr>
</tbody>
</table>

- Top-level Page table
- 2nd-level tables
- Frames in memory
Multi-level page tables

- A Virtual Address:

```
<table>
<thead>
<tr>
<th>10-bits</th>
<th>10-bits</th>
<th>12-bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT1</td>
<td>PT2</td>
<td>offset</td>
</tr>
</tbody>
</table>
```

- Top-level Page table
- 2nd-level tables
- Frames in memory
Multi-level page tables

- A Virtual Address:
  - 10-bits
  - 10-bits
  - 12-bits

<table>
<thead>
<tr>
<th>PT1</th>
<th>PT2</th>
<th>offset</th>
</tr>
</thead>
</table>

  Top-level Page table

  2nd-level tables

  frames in memory
Multi-level page tables

- Ok, but how exactly does this save space?
Multi-level page tables

- **Ok, but how exactly does this save space?**

- **Not all pages within a virtual address space are allocated**
  - Not only do they not have a page frame, but that range of virtual addresses is not being used
  - So no need to maintain complete information about it
  - Some intermediate page tables are empty and not needed

- **We could also page the page table**
  - This saves space but slows access ... a lot!
VM puzzle