Page Tables

(c) 2022 Robert Finch

# Page Tables

## Intro

Page tables are part of the memory management system used map virtual addresses to real physical addresses. There are several types of page tables. Hierarchical page tables are probably the most common. Almost all page tables map only the upper bits of a virtual address, called a page. The lower bits of the virtual address are passed through without being altered. The page size often 4kB which means the low order 12-bits of a virtual address will be mapped to the same 12-bits for the physical address.

# Hierarchical Page Tables

Hierarchical page tables organize page tables in a multi-level hierarchy. They are capable of mapping the entire virtual address range. At the topmost level a register points to a page directory, that page directory points to a page directory at a lower level until finally a page directory points to a page containing page table entries. To map an entire 64-bit virtual address range approximately five levels of tables are required.

A picture of multi-level page tables


# Inverted Page Tables

An inverted page table is a table used to store address translations for memory management. The idea behind an inverted page table is that there is a fixed number of pages of memory no matter how it is mapped. It should not be necessary to provide for a map of every possible address, only addresses that correspond to real pages of memory. Each page of memory can be allocated only once. It is either allocated or it is not. Compared to a non-inverted paged memory management system where tables are used to map potentially the entire address space an inverted page table uses less memory. There is typically only a single inverted page table supporting all applications in the system. This is a different approach than a non-inverted page table which may provide separate page tables for each process.

## The Simple Inverted Page Table

The simplest inverted page table contains only a record of the virtual address mapped to the page, and the index into the table is used as the physical page number. There are only as many entries in the inverted page table as there are physical pages of memory. A translation can be made by scanning the table for a matching virtual address, then reading off the value of the table index. The attraction of an inverted page table is its small size compared to the typical hierarchical page table. Unfortunately, the simplest inverted page table is not practical when there are thousands or millions of pages of memory. It simply takes too long to scan the table. The alternative solution to scanning the table is to hash the virtual address to get a table index directly.

Diagram of Inverted Page Table


# Hashed Page Tables

## Hashed Table Access

Hashes are great for providing an index value immediately. The issue with hash functions is that they are just a hash. It is possible that two different virtual address will hash to the same value. What is then needed is a way to deal with these hash collisions. There are a couple of different methods of dealing with collisions. One is to use a chain of links. The chain has each link in the chain pointing the to next page table entry to use in the event of a collision. The hash page table is slightly more complicated then as it needs to store links for hash chains. The second method is to use open addressing. Open addressing calculates the next page table entry to use. The calculation may be linear, quadratic or some other function dreamed up. A linear probe simply chooses the next page table entry in succession from the previous one if no match occurred. Quadratic probing calculates the next page table entry to use based on squaring the count of misses.

# Shared Memory

Another issue to deal with is shared memory. Sometimes applications share memory with other apps for communication purposes, and to conserve memory space where there are common elements. With a hierarchical paged memory management system, it is easy to share memory, just modify the page table entry to point to the same physical memory as is used by another process. With an inverted page table having only a single entry for each physical page is not sufficient to support shared memory. There needs to be multiple page table entries available for some physical pages but not others because multiple virtual addresses might map to the same physical address. One solution would be to have multiple buckets to store virtual addresses in for each physical address. However, this would waste a lot of memory because much of the time only a single mapped address is needed. There must be a better solution. Rather than reading off the table index as the physical page number, the association of the virtual and physical address can be stored. Since we now need to record the physical address multiple times the simple mechanism of using the table index as the physical page number cannot be used. Instead, the physical page number needs to be stored in the table in addition to the virtual page number.

That means a table larger than the minimum is required. A minimally sized table would contain only one entry for each physical page of memory. So, to allow for shared memory the size of the table is doubled. This smells like a system configuration parameter.

# Example

# Thor2022 Page Tables

Thor2022 retains the same page table entry structure for either hashed or hierarchical page tables. Since this is a common element it is described first.

## Page Table Entries - PTE

We have determined that a page table entry needs to store both the physical page number and the virtual page number for the translations. In addition to those two items additional bookkeeping bits are stored. The physical and virtual page numbers are 52 bits each for a total of 104 bits. The other bookkeeping bits require 39 bits of storage, so the total is 141 bits. It is also probably a good idea to leave a little bit of extra space to expand the virtual address for instance. Software often needs to store additional information in the page table entry as well. A size of 160 bits was chosen. This fits six entries into a 1024-bit group for hash tables.

For hierarchical tables to keep things simple the structure is padded out to 256 bits. This allows 128 PTEs to fit into a 4kB page. It may seem wasteful to use a 256-bit structure when only 160 bits are needed, but packing an odd number of entries into a page leads to difficulties breaking up the address into segments for a hierarchical table. A divider would be required instead of a right shift.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Physical Page Number31..0 | | | | | | | | | | | | | | | | | | |
| Av | ~3 | PL8 | Physical Page Number51..32 | | | | | | | | | | | | | | | |
| Virtual Page Number31..0 | | | | | | | | | | | | | | | | | | |
| ~12 | | | Virtual Page Number51..32 | | | | | | | | | | | | | | | |
| ASID12 | | | SC | SR | SW | SX | V | G | ~2 | BC4 | D | U | S | A | C | R | W | X |

Closely related to page table entries are translation look-aside buffer, TLB, entries. TLB entries have more fields to provide access counting and keyed access. The additional field are populated from the access rights table, ART.

## TLB Entry

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 20 | | | | | 19 | | 18 | 17 | 16 | | | 15 | 14 | 13 12 | 11 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Physical Page Number31..0 | | | | | | | | | | | | | | | | | | | | | | | |
| Av | ~3 | PL8 | | Physical Page Number51..32 | | | | | | | | | | | | | | | | | | | |
| Virtual Page Number31..0 | | | | | | | | | | | | | | | | | | | | | | | |
| ~12 | | | | Virtual Page Number51..32 | | | | | | | | | | | | | | | | | | | |
| ASID12 | | | SC | | | SR | | SW | | SX | V | | G | ~2 | BC4 | D | U | S | A | C | R | W | X |
| Key32 | | | | | | | | | | | | | | | | | | | | | | | |
| Access Count32 | | | | | | | | | | | | | | | | | | | | | | | |

Fields Description

|  |  |
| --- | --- |
| ASID | Address Space Identifier |
| V | translation Valid |
| G | Global translation |
| BC | Bounce Count |
| D | dirty |
| U | unspecified use |
| S | Page Size |
| A | Accessed |
| C | Cacheable |
| R | Readable |
| W | Writeable |
| X | Executable |
| SC,SR,SW,SX | system mode C,R,W,X |
| Av | Access rights info valid |
| PL | privilege level |
| Key | protection key |

The page table does not include everything needed to manage pages of memory. There is additional information such as share counts and privilege levels to take care of, but this information is better managed in a separate table.

# Thor2022 Hash Page Table Setup

## Page Table Groups

We want the search for translations to be fast. That means being able to search in parallel. So, PTEs are stored in groups that are searched in parallel for translations. This is sometimes referred to as a clustered table approach. Access to the group should be as fast as possible. There are also hardware limits to how many entries can be searched at once while retaining a high clock rate. So, the convenient size of 1024 bits was chosen as the amount of memory to fetch. This corresponds to two cache lines worth of data, and eight 128-bit burst accesses to the memory controller.

A page table group then contains six page-table entries. All entries in the group are searched in parallel for a match.

|  |  |
| --- | --- |
| 159 0 | |
|  | Group Header64 |
| PTE0 | |
| PTE1 | |
| PTE2 | |
| PTE3 | |
| PTE4 | |
| PTE5 | |

For a system with 512MB of RAM:

There are 131072 4kB pages. 131072 PTEs are required without considering sharing. Since we decided to double the table size for sharing 262144 PTEs need to be accommodated. This works out to 43691 groups of PTEs, or about 5.6 MB. 43691 is an ugly number. Let us round it down to 32768 groups. That gives a large excess of PTEs to allow for shared memory. 32768 PTGs consumes 4MB of RAM. This is about 0.8% of memory. To get to a page table group fast a hash function is needed then that returns a 15-bit number.

## Hash Function

The hash function needs to reduce the size of a virtual address down to a 15-bit number. The asid should be considered part of the virtual address. Including the asid an address is 72 bits. The first thing to do is to throw away the lowest twelve bits as they pass through the MMU unaltered. We now have 60-bits to deal with. We can probably throw away some high order bits too, as a process is not likely to use the full 64-bit address range.

The hash function chosen uses the asid combined with virtual address bits 14 to 28. This should space out the PTEs according to the asid. Address bits 12 and 13 select one of four address ranges. the PTG supports six PTEs. The translations where address bits 12 and 13 are involved are likely consecutive pages that would show up in the same PTG. The hash is the asid shift left three times exclusively or’d with address bis 14 to 28.

## Collision Handling

Quadratic probing of the page table is used when a collision occurs. The next PTG to search is calculated as the hash plus the square of the miss count. On the first miss the PTG at the hash plus one is searched. Next the PTG at the hash plus four is searched. After that the PTG at the hash plus nine is searched, and so on.

## Finding a Match

Once the PTG to be searched is located using the hash function, which PTE to use needs to be sorted out. The match operation must include both the virtual address bits and the asid, address space identifier, as part of the test for a match. It is possible that the same virtual address is used by two or more different address spaces, which is why it needs to be in the match.

## Page Table Group Header

The page table group header contains a link to the next page table group to search in the event of a hash collision, when there is not enough room in the current PTG. This field is not currently used since searches use quadratic probing.

## Locality of Reference

The page table group may be cached in the system read cache for performance. It is likely that the same PTG group will be used multiple times due to the locality of reference exhibited by running software.

## Access Rights

To avoid duplication of data the access rights are stored in another table called the ART for access rights table. The first time a translation is loaded the access rights are looked-up from the ART. A bit is set in the TLB entry indicating that the access rights are valid. On subsequent translations the access rights are not looked up, but instead they are read from values cached in the TLB.

# The TLB – Translation Lookaside Buffer

If every memory access required two or three additional accesses to map the address to a final target access, memory access would be quite slow, slowed down by a factor or two or three, possibly more. To improve performance, the memory mapping translations are stored in another unit call the TLB standing for Translation Lookaside Buffer.

The TLB is a cache specialized for address translations. Thor2022’s TLB is quite large being five way associative with 1024 entries per way. This choice of size was based on the minimum number of block RAMs that could be used to implement the TLB. On a TLB miss the inverted page table is searched for a translation and if found the translation is stored in one of the ways of the TLB. The way selected is determined either randomly or in a least-recently-used fashion as one of the first four ways. The fifth way may not be updated automatically by a page table search, it must be updated by software.

# Access Rights Table

## Overview

After the page number is found the access rights table is referenced to obtain the access rights to the page. The access rights table contains an assortment of information. Pieces of information include the key needed to access the page, the privilege level, and read-write-execute permissions for the page. The table is organized as rows of access rights table entries (ARTEs). There are as many ARTEs as there are pages of memory.

## ARTE Description

Access Rights Table Entry

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 20 | | 19 | | 18 | 17 | 16 | | | | 15 | 14 | 13 12 | 11 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ACL16 | | | | | | | | | Share Count16 | | | | | | | | | | | | |
| Access Count32 | | | | | | | | | | | | | | | | | | | | | |
| Key32 | | | | | | | | | | | | | | | | | | | | | |
| ~4 | PL8 | SC | SR | | SW | | SX | V | | | N | ~2 | ACL19..16 | D | U | ~ | A | C | R | W | X |

### Access Control List

The ACL field is a reference to an associated access control list.

### Access Count

This part uses the term ‘access count’ to refer to the number of times a page is accessed. This is usually called the reference count, but that phrase is confusing because reference counting may also refer to share counts. So, the phrase ‘reference count’ is avoided. Some texts use the term reference count to refer to the share count. Reference counting is used in many places in software and refers to the number of times something is referenced.

Every time the page of memory is accessed, the access count of the page is incremented. Periodically the access count is aged by shifting it to the right one bit.

The access count may be used by software to help manage the presence of pages of memory.

### Key

The access key is a 32-bit value associated with the page and present in the key ring of processes. To obtain access to the page it is necessary for the process to have a matching key OR if the key to match is set to zero in the ARTE then a key is not needed to access the page.

### Privilege Level

The current privilege level is compared with the privilege level of the page, and if access is not appropriate then a privilege violation occurs. For data access, the current privilege level must be at least equal to the privilege level of the page. If the page privilege level is zero anybody can access the page.

### N

indicates a conforming page of executable code. Conforming pages may execute at the current privilege level.

C indicates that the page may be cached by users.

R Indicates the page may be read by users

W Indicates the page may be written by users

X Indicates the page may be executed by user programs.

SC, SR, SW, and SX serve the same purpose as C, R, W, and X except for system software.