1 Paging

1.1 Address translation: implementation

A logical (physical) address is **relative** to the start of the program (memory) and consists of two parts:

- The **left** most n bits that represent the page (frame) **number** e.g. 4 bits for the page number allowing 16 (24) pages (frames)
- The **right** most m bits that represent the **offset** within the page (frame) e.g. 12 bits for the offset, allowing up to 4096 (212) bytes per page (frame)

The offset within the page and frame remains the same (**they are the same size**). The page number to frame number **mapping** is held in the *page table*.

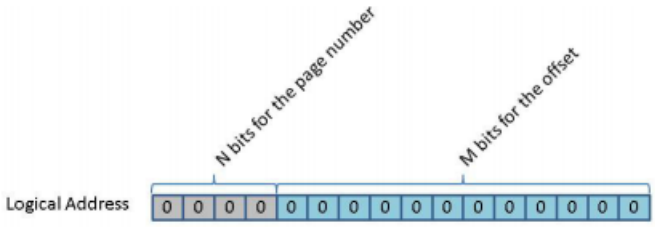

Figure: Logical Address

Steps in address translation:

- Extract the page number from logical address
- Use page number as an index to retrieve the frame number in the page table
- Add the logical offset within the page to the start of the physical frame

Hardware implementation of address translation

- The CPUs memory management unit (MMU) intercepts logical addresses
- MMU uses a page table as above
- The resulting physical address is put on the memory bus

1.2 Entry contents

- A **present/absent bit** that is set if the page/frame is in memory
- A **modified bit** that is set if the page/frame has been modified (**only modified pages** have to be written back to disk when evicted)
- A **referenced bit** that is set if the page is in use
- **Protection and sharing bits**: read, write, execute or combinations thereof

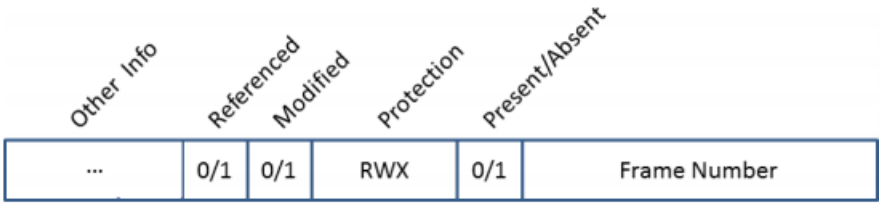

Figure: Page Table Entry

1.3 Dealing with large page tables

How do we deal with the increasing size of page tables, i.e., where do we store them?

- Their size **prevents** them from being stored in registers.
- They have to be stored in (**virtual**) main memory:
  - **Multi-level** page tables.
  - **Inverted** page tables (for **large** virtual address spaces)
- How can we maintain acceptable speeds?: address translation happens at every memory reference, it has to be fast!
- Accessing main memory results in memory stalls

Solution: **Page** the *page table*!

- We keep tree-like structures to hold page tables
- Divide the page number into
  - An index to a page table of second level
  - A page within a second level page table
- No need to keep all page tables in memory all time

Memory organisation of multi-level page tables:

- The root page table is **always** maintained in **memory**
- Page tables themselves are maintained in *virtual memory* due to their **size**

2 Virtual memory

2.1 Locality

Are there any other benefits of *paging*?

- Code execution and data manipulation are usually restricted to a **small subset** (i.e. limited number of pages) at any point in time I.e. code and data references within a process are usually **clustered**. This is called the *principle of locality*.
- Not all pages have to be loaded in memory at the same time virtual memory

Loading an entire set of pages for an entire program/data set into memory is **wasteful**. Desired blocks could be loaded on demand

2.2 Page faults

The *resident set* refers to the pages that are loaded in main memory.A *page fault* is generated if the processor accesses a page that is not in memory.

- A page fault results in an **interrupt** (process enters blocked state).
- An I/O operation is started to **bring the missing page into** main memory.
- A *context switch* (may) take place
- **An interrupt** signals that the I/O operation is complete (process enters *ready state*)

2.2.1 Processing page faults

- Trap operating system
  - Save registers/process state
  - Analyse interrupt (i.e., identify page fault)
  - Validate page reference, determine page location
  - Issue disk I/O: queueing, seek, latency, transfer
- Context switch (optional)
- Interrupt for I/O completion
  - Store process state/registers
  - Analyse interrupt from disk
  - Update page table (page in memory)
  - Wait for original process to be scheduled
- Context switch to original process

2.3 Benefits

Being able to maintain more processes in main memory through the use of virtual memory **improves** CPU utilisation.

- Individual processes take up **less** memory since they are only **partially loaded**
- Virtual memory allows the *logical address space* (i.e processes) to be **larger** than *physical address space* (i.e. main memory) 64 bit machine 2^{64} logical addresses (theoretically)

Reference section

virtual memory
: a storage area that holds the files on your hard drive for retrieval when a computer runs out of RAM.