Computer Organization and Architecture

Chapter 8

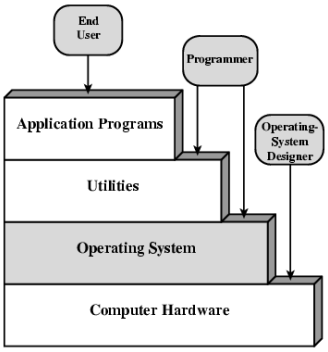
Operating System Support

1. Processes and Scheduling
2. Memory Management

OS Objectives and Functions

- Convenience
  - Making the computer easier to use
- Efficiency
  - Allowing better use of computer resources
- These were two main objectives of an operating system until the 1980's. Now we can also add:
- Security
  - Protecting the computer from malicious programs

Layers and Views of a Computer System



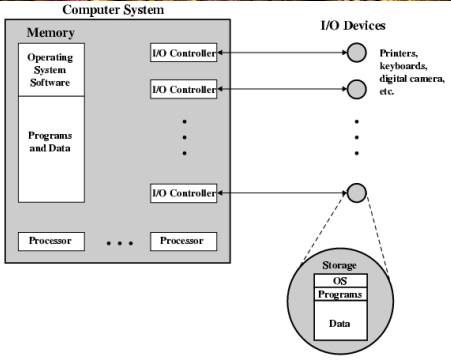
Operating System Services

- Program creation
  - Debuggers depend on very low-level OS access
- Program execution
  - OS loader prepares programs for executions, opens files, allocates computer resources
- Access to I/O devices
  - Programmers think in terms of reads and writes; OS handles the details
- Controlled access to files
  - Files are an abstraction created by the OS, which handles device-specific details
  - OS also provides access control for multi-user access

Operating System Services

- System access
  - OS controls access to system as a whole and to specific system resources
  - A great deal of OS development has taken place with the advent of Internet connectivity
  - Personal computers used to be just that – not any more!
- Error detection and response
  - Internal and external hardware errors, device failure, software errors, access to protected memory, lack of resources
  - The OS should clear/report/repair the error with minimal impact on running processes (and the OS itself)
  - Building a robust OS is not easy
- Accounting
  - Collection of various usage and performance statistics for tuning the system and billing or tracking usage of shared resources

O/S as a Resource Manager



What is an Operating System anyway?

- Just another program, or collection of programs
- O/S has to relinquish control of the processor to allow "useful" programs to operate
- But the O/S cannot take up too much of the processor time, and somehow it must remain in control of the processor even when the processor runs another program

Types of Operating Systems

- Three main types:

1. Interactive
2. Batch
3. Embedded

- Independent of multi-processing type:

1. Single program (Uni-programming)
2. Multi-programming (Multi-tasking)

Early Systems

- Late 1940s to mid 1950s
- No Operating System
  - Programs were run directly from the machine console, with display lights, push buttons, toggle switches, some input device and a printer
- Programs interact directly with hardware
- Two main problems:
  - Scheduling
  - Setup time

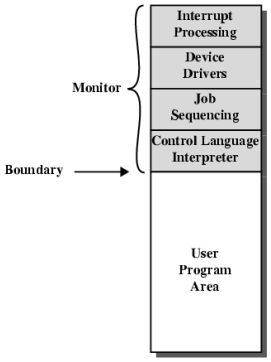
Simple Batch Systems

- Resident Monitor program
  - The first part of what we consider an operating system
  - Tiny machine memories prohibited anything but the simplest OS
  - IBM terminology: "Resident Executive"
- Users submit jobs to operator
- Operator batches jobs
- Resident Monitor controls sequence of events to process batch
- When one job is finished, control returns to Monitor which reads next job
- Monitor handles scheduling

IBM 1130

IBM 1130 Console

Memory Layout for Resident Monitor



Job Control Language

- Instructions to Monitor
- Usually denoted by $
- e.g.

```
$JOB
$FTN
… Some Fortran instructions
$LOAD
$RUN
… Some data
$END
```

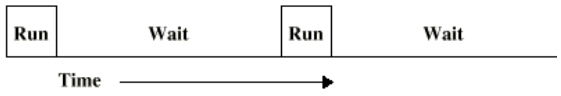
Desirable Hardware Features

- Memory protection
  - To protect the Monitor from erroneous programs
- Timer
  - To prevent a job monopolizing the system
  - Program errors: infinite loops
  - Control returns to monitor with timer interrupt
- Privileged instructions
  - Only executed by Monitor
  - I/O controlled by Monitor; we don't want a user program accidentally ready JCL cards
- Interrupts
  - Allow for relinquishing and regaining control

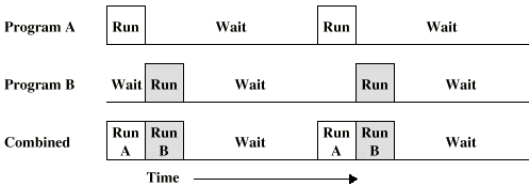
Multi-programmed Batch Systems

- I/O devices are very slow compared to CPU
- Loading a program with its compiler from cards is a long slow process
- When one program is waiting for I/O, another can use the CPU

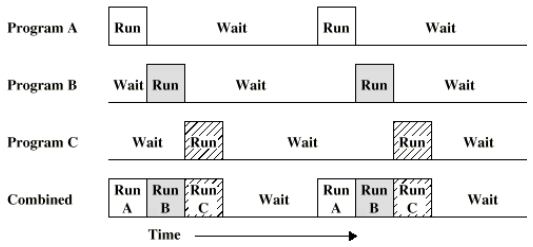
Single Program



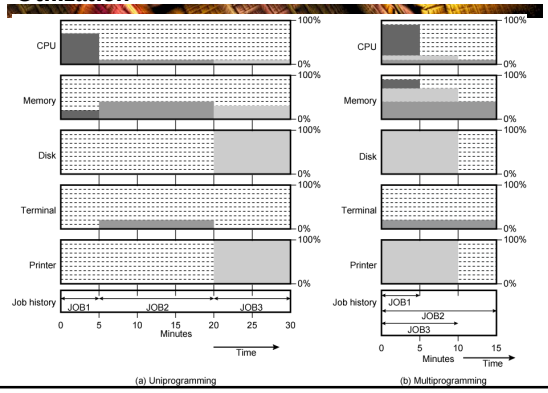
Multi-Programming with Two Programs



Multi-Programming with Three Programs



Utilization



Microsoft MS-DOS

- MS-DOS (Microsoft Disk Operating System) launched Microsoft in the early 1980's
- Microsoft did not invent MS-DOS
  - They purchased source code for 86-DOS (aka QDOS - Quick and Dirty Operating System), derived from CP/M (the most popular 8 bit OS at the time) and modified it
- MS-DOS is a uniprogramming OS
- Multitasking is very difficult due to design of O/S
  - Extensive usage of static global variables
  - Software interrupt interface

Time Sharing Systems

- Allow users to interact directly with the computer
  - i.e. Interactive
- Multi-programming allows a number of users to interact with the computer

Scheduling

- Key to multi-programming
  - Concept of program as a process started with Multics in the 1960's
- Four types of scheduling

1. Long term
   - Adding a new process to execution pool
2. Medium term
   - Bringing a process into main memory
3. Short term
   - Allocating processor time to a process
4. I/O
   - Allocating an I/O device to a pending I/O request

Long Term Scheduling

- Determines which programs are submitted for processing
- i.e. controls the degree of multi-programming
  - Once submitted, a job becomes a process for the short term scheduler
  - (or it becomes a swapped out job for the medium term scheduler)
- Batch systems queue jobs to disk
- Interactive system admit user processes immediately until system reaches maximum load

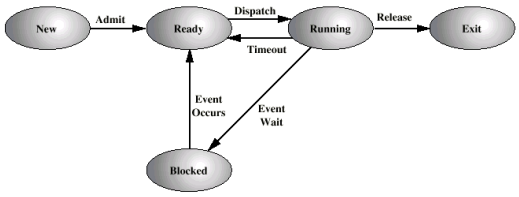
Medium Term Scheduling

- Part of the swapping function (later…)
- Usually based on the need to manage multi-programming
- If no virtual memory, memory management is also an issue

Short Term Scheduler

- a.k.a Dispatcher
- Fine grained decisions of which job to execute next
  - i.e. which job actually gets to use the processor in the next time slot
- Long term scheduler executes infrequently compared to short-term scheduler
- Time slices may be allocated in milliseconds

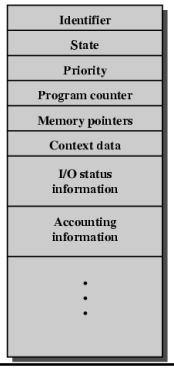
Five State Process Model



Process Control Block

- Data maintained by OS encapsulating process state
  - Identifier
  - State (new, ready, blocked, etc)
  - Priority
  - Program counter
  - Memory pointers (location of process in mem)
  - Context data (processor registers and other data)
  - I/O status (I/O requests, devices assigned, open files, etc)
  - Accounting information
- Created when new process is added by long-term scheduler

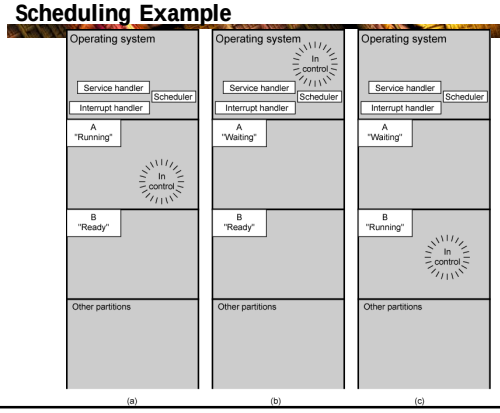
PCB Diagram



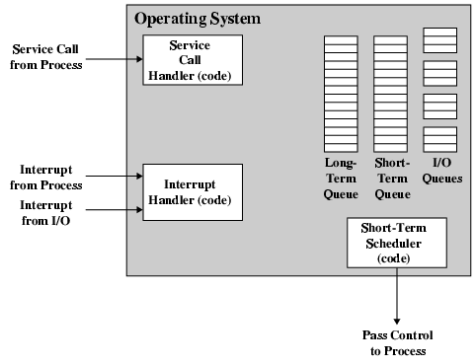
Context-switch triggers

- A running process has control of CPU
- Control returns to OS when:
  - I/O request is made
  - Interrupts occur
    - Process causes error (e.g., divide by 0)
    - Process attempts illegal operation (privileged instruction)
    - Timeout interrupt
  - I/O device interrupts process

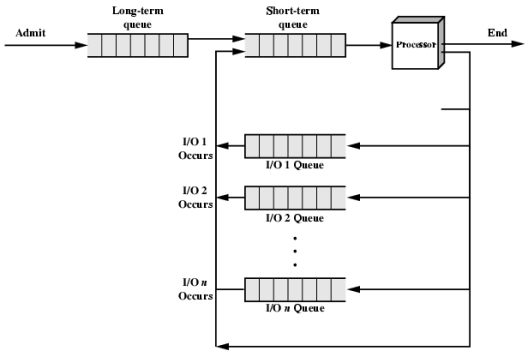
Scheduling Example



Key Elements of O/S



Process Scheduling



Memory Management

- Uni-program computer
  - Memory split into two
  - One for Operating System (monitor)
  - One for currently executing program
- Multi-program computer
  - "User" part is sub-divided and shared among active processes
- Current PC operating systems
  - Program has a large virtual address space
  - Part of this space is shared with OS and other programs (shared libraries and OS code)

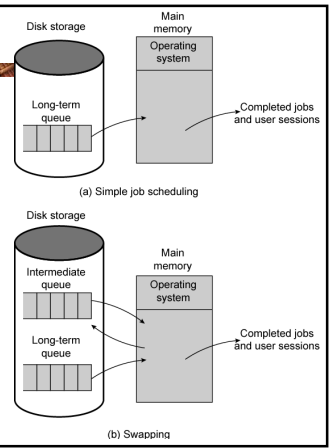
Swapping

- Problem: I/O is so slow compared with CPU that even in multi-programming system, CPU can be idle most of the time
- Solutions:
  - Increase main memory
    - An expensive solution
    - Inevitably leads to larger programs
  - Swapping memory out to disk

What is Swapping?

- Long term queue of processes stored on disk
- Processes "swapped" in as space becomes available
- As a process completes it is moved out of main memory
- If none of the processes in memory are ready (i.e. all I/O blocked)
  - Swap out a blocked process to intermediate queue
  - Swap in a ready process or a new process
  - But swapping is an I/O process...
    - Use fastest device available (disk)
    - Use virtual memory (discussed later)

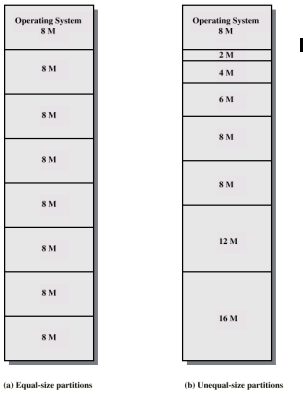
Use of Swapping



Partitioning

- Splitting memory into sections to allocate to processes (including Operating System)
- Fixed-sized partitions
  - May not be equal size
  - Process is fitted into smallest hole that will take it (best fit)
  - Always wastes some memory

Fixed Partitioning



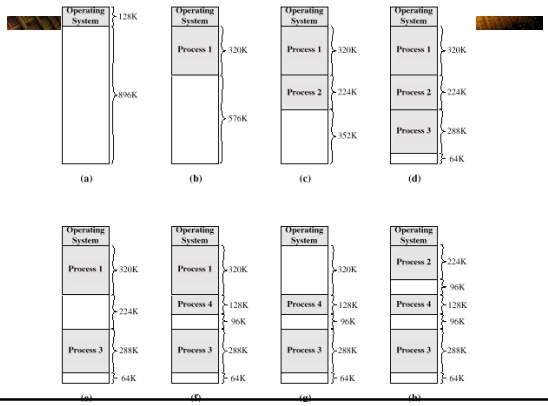
Variable Sized Partitions (1)

- Allocate exactly the required memory to a process
- This leads to a hole at the end of memory, too small to use
  - Only one small hole - less waste
- When all processes are blocked, swap out a process and bring in another
- New process may be smaller than swapped out process
- Another hole

Variable Sized Partitions (2)

- Eventually have lots of holes (fragmentation)
- Solutions:
  - Coalesce - Join adjacent holes into one large hole
  - Compaction - From time to time go through memory and move all hole into one free block (c.f. disk de-fragmentation)

Effect of Dynamic Partitioning



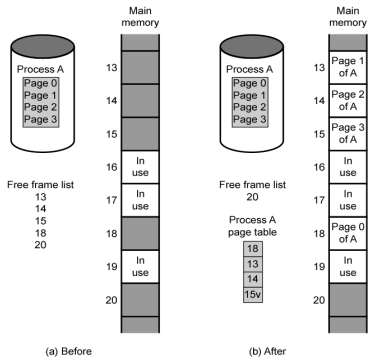
Relocation

- No guarantee that process will load into the same place in memory
- Instructions contain addresses
  - Locations of data
  - Addresses for instructions (branching)
- To get relocatable programs use some form of logical address - relative to the beginning of program
- The physical address (actual location in memory) is computed on the fly by adding logical address to a base address
- A processor feature designed to meet an OS requirement
  - Note that Intel x86 processors provide four or six base address registers (segment registers) – an unnecessary complication

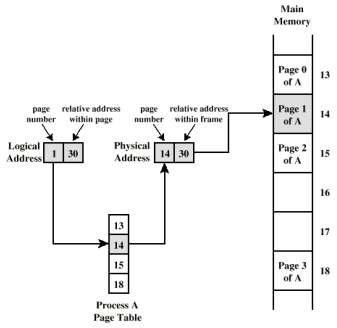
Paging

- Partitioned memory, whether fixed or variable, is inefficient due to fragmentation
- Paging is a solution that:
  - Splits memory into equal sized, small chunks -page frames
  - Splits programs (processes) into equal sized small chunks - pages
  - Allocates the required number page frames to a process
- Operating System maintains list of free frames
- A process does not require contiguous page frames
  - A process "believes" that is has a flat, linear address space
  - OS uses a page table to map linear addresses to page frames

Allocation of Free Frames



Logical and Physical Addresses - Paging



Demand Paging

- Demand paging: a refinement of simple paging
  - Do not require all pages of a process in memory
  - Bring in pages as required
- Page fault
  - Required page is not in memory
  - Operating System must swap in required page
  - May need to swap out a page to make space
  - Select page to throw out based on recent history

Virtual Memory

- With demand paging we have "virtual memory"
- Some consequences:
  - More effective multiprogramming because more physical memory is available
  - Process can have a linear address space that is much larger than physical memory
  - Memory can be shared between processes (and the operating system) because page table entries for different processes can point to the same page frame

Thrashing

- Occurs with too many processes in too little memory
- Operating System spends all its time swapping
- Little or no real work is done; disk light is on all the time; disk sounds like a coffee grinder
- Solutions
  - Good page replacement algorithms
  - Reduce number of processes running
  - Add more memory!

One small problem…

- Most systems allocate 1 page table per process
- Page tables can get large
  - Vax 2GB mem; 512 byte pages; process can have 2^{22} (4,194,304) pages
  - Pentium 4GB mem, 4096 byte pages, process can have 2^{20} (1,048,576) pages
- Page tables can be swapped out to disk
- Some systems use 2-level page tables (x86, ARM)
- Other systems use inverted page tables

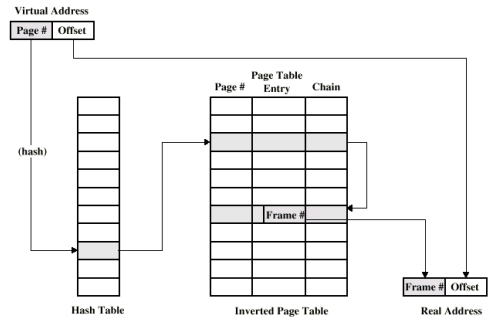
Hash Functions

- A hash function maps values in the range 0 to M to a smaller range 0 to N, where M>N
  - Simple example: Modulus function
- Potential for collision: 2 values X,Y map to same hash value H
- A variety of algorithms are used to handle collisions
  - Chaining hash buckets
  - Compute new hash function

Inverted Page Tables

- In an inverted page table there is one entry per physical page of memory, rather than one entry per page of virtual memory
- Fixed proportion of memory is used for page tables
- Structure is called inverted because entries are indexed by frame number rather than virtual page number
  - Term derives from indexing techniques used in non-relational databases

Inverted Page Table Structure



Another small problem

- Use of page tables implies that each memory access requires two (or more) memory accesses
  - First fetch page table entry
  - Then fetch data
- This would double memory access time without even considering swapped out memory or swapped out page tables!

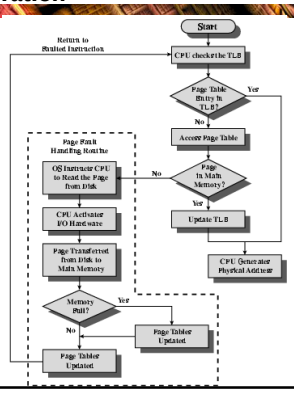
Translation Lookaside Buffer

- Solution: cache most recently used page table entries in a special cache
- Translation Lookaside Buffer (TLB)
- The TLB is a cached of recently used page table entries
  - has a fixed number of slots
  - Uses associative mapping
- TLB can sit between processor and cache (virtual addressing) or between cache and memory (physical addressing)

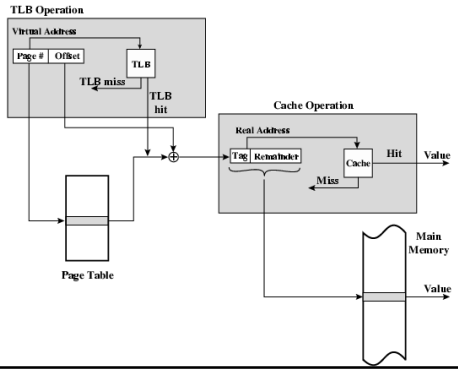
So for one small memory access...

- Translate virtual address to physical address, which generates:
- Reference to page table, which may be in TLB, main memory or disk
- Referenced word may be in cache, main memory, or disk
- If disk: load page, load block into cache, update page tables

TLB Operation



TLB and Cache Operation



Segmentation

- Paging is transparent to the programmer
- Segmentation is another way to divide memory
- Segmentation is program-visible
- Usually different segments allocated to program and data
- May be a number of program and data segments

Advantages of Segmentation

- Simplifies handling of growing data structures
- Allows programs to be altered and recompiled independently, without re-linking and re-loading
- Lends itself to sharing among processes
- Lends itself to protection
- Some systems combine segmentation with paging

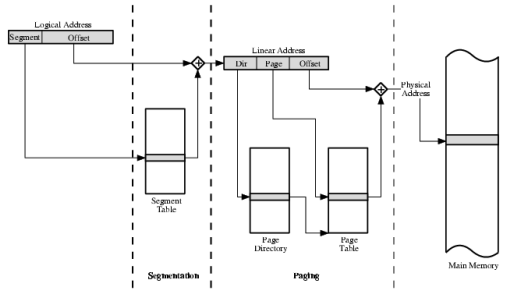
Pentium

- Has hardware for both segmentation and paging
- Unsegmented unpaged
  - virtual address = physical address
  - Used Low complexity, high performance controller applications
- Unsegmented paged
  - Memory viewed as paged linear address space
  - Protection and management via paging
  - Berkeley UNIX, Linux
- Segmented unpaged
  - Collection of local address spaces
  - Protection to single byte level
  - Translation table needed is on chip when segment is in memory; result is predictable access times
- Segmented paged
  - Segmentation used to define logical memory partitions subject to access control
  - Paging manages allocation of memory within partitions
  - Unix System V, Windows

Pentium Flat Model

- In practice segmentation cannot be switched off in 80386+ processors
- Operating systems that prefer not to use segmentation use the "flat model" where SS=ES=DS. CS has a different value because of the differences between code segments and data segments
- All segments have Base = 0 Limit = 4G

Pentium Address Translation Mechanism



Pentium Segmentation

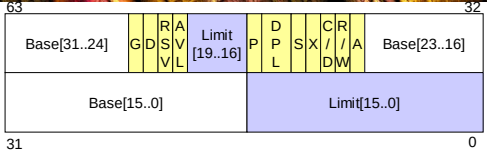
- Each virtual address is 16-bit segment and 32-bit offset
  - 2 bits of segment are protection mechanism (Descriptor Privilege Level)
  - 1 bit specifies Local or Global DT (Descriptor table)
  - Leaving 13 bits for DT index
  - So we can have 2 tables each with 8192 entries
- Unsegmented virtual memory 2^{32} = 4Gbytes
- Segmented $2^{32}*2^{16}$ 2^{46}=64 terabytes
  - Can be larger – depends on which process is active
  - Half (8K segments of 4Gbytes) is global
  - Half is local and distinct for each process

EA Computation

- Selector (segment register) layout:

- Index (Bits 3-15) 13-bit values allows 8,192 entries in a descriptor table
- Table Indicator (Bit 2) (1=Local DT, 0 = Global DT)
- Requested Privilege Level (Bits 0-1) must be <= privilege level stored in segment descriptor

Segment Descriptor Format



- Address computation

1. Selector indexes into GDT or LDT
2. Base Address is fetched from Descriptor
3. Offset is added to base address to determine "linear" 32-bit address
4. Linear address compared against Limit (in 4KB or 4MB pages) to determine if address is legal

Pentium Memory Management Parameters

Segment Descriptor (Segment Table Entry)

Base
Defines the starting address of the segment within the 4-GByte linear address space.

D/B bit
In a code segment, this is the D bit and indicates whether operands and addressing modes are 16 or 32 bits.

Descriptor Privilege Level (DPL)
Specifies the privilege level of the segment referred to by this segment descriptor.

Granularity bit (G)
Indicates whether the Limit field is to be interpreted in units by one byte or 4 KBytes.

Limit
Defines the size of the segment. The processor interprets the limit field in one of two ways, depending on the granularity bit: in units of one byte, up to a segment size limit of 1 MByte, or in units of 4 KBytes, up to a segment size limit of 4 GBytes.

S bit
Determines whether a given segment is a system segment or a code or data segment.

Segment Present bit (P)
Used for nonpaged systems. It indicates whether the segment is present in main memory. For paged systems, this bit is always set to 1.

Type
Distinguishes between various kinds of segments and indicates the access attributes.

Pentium Memory Management Parameters

Page Directory Entry and Page Table Entry

Accessed bit (A)
This bit is set to 1 by the processor in both levels of page tables when a read or write operation to the corresponding page occurs.

Dirty bit (D)
This bit is set to 1 by the processor when a write operation to the corresponding page occurs.

Page Frame Address
Provides the physical address of the page in memory if the present bit is set. Since page frames are aligned on 4K boundaries, the bottom 12 bits are 0, and only the top 20 bits are included in the entry. In a page directory, the address is that of a page table.

Page Cache Disable bit (PCD)
Indicates whether data from page may be cached.

Page Size bit (PS)
Indicates whether page size is 4 KByte or 4 MByte.

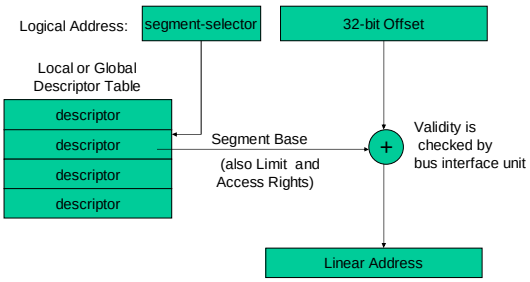
Page Write Through bit (PWT)
Indicates whether write-through or write-back caching policy will be used for data in the corresponding page.

Present bit (P)
Indicates whether the page table or page is in main memory.

Read/Write bit (RW)
For user-level pages, indicates whether the page is read-only access or read/write access for user-level programs.

User/Supervisor bit (US)
Indicates whether the page is available only to the operating system (supervisor level) or is available to both operating system and applications (user level).

Segmented Addresses



Pentium Protection

- Protection bits give 4 levels of privilege
  - 0 most protected, 3 least
  - Usually level (ring) 3 for applications, ring 0 for O/S and kernel (levels 1 and 2 are rarely used)
  - Some instructions only work in level 0
- Recent developments
  - AMD x86-64 introduced a sort of super ring 0 ("ring –1") that allows virtualization systems to control operating systems running in ring 0

Pentium Paging

- Segmentation may be "disabled" (flat memory model)
  - In which case linear address space is used
- Two level page table lookup
  - First, page directory
    - 1024 entries max
    - Splits 4G linear memory into 1024 page groups of 4Mbyte
    - Each page table has 1024 entries corresponding to 4Kbyte pages
    - Can use one page directory for all processes, one per process or mixture
    - Page directory for current process always in memory
  - Use TLB holding 32 page table entries
  - Two page sizes available 4k or 4M

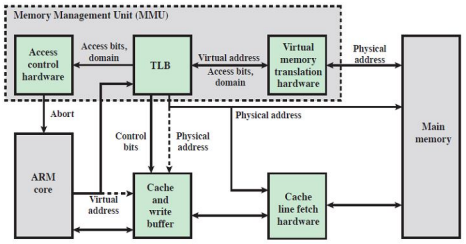
Large Pages

- With 4 KB pages a full 4 GB main memory uses 4MB of memory just for page tables
- With 4MB pages the page table size is reduced to 4KB
- The page size can vary by segment, so frequently used memory such as the OS kernel can reside in large pages
- Separate TLBs are used for small and large pages

ARM Memory Management

- Flexible system allows 1 or 2 levels of translation tables for virtual addresses
- TLB is cache of recently used page entries
  - If entry is available, TLB presents physical address directly to memory for read or write
  - Some ARM designs use a logical cache and others use a physical cache
  - With logical cache virtual addresses are presented both to TLB and cache
  - With physical cache TLB presents physical address to cache

ARM Memory Management Overview



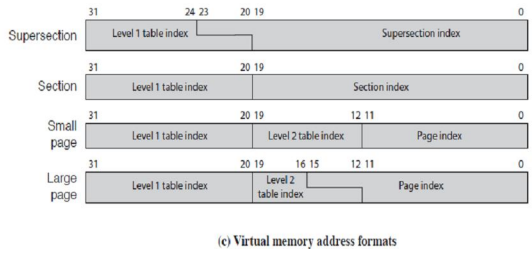
Virtual Address Translation

- ARM designs are intended for embedded systems with widely variable memory requirements
- ARM supports memory access based on sections or pages of main memory
  - Supersections – 16MB blocks (support optional)
  - Sections – 1MB blocks
  - Large pages – 64KB blocks
  - Small pages – 4KB blocks
- Sections and supersections allow mapping of large regions of memory using a single TLB entry and only one level of page table

Sections and Pages

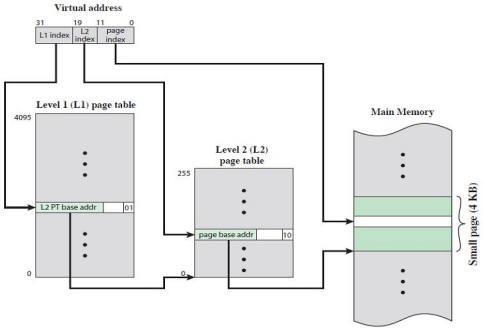
- Page translation tables have two levels:
  - First level table holds first-level descriptors that contain the base address and
    - translation properties for Sections and Supersections
    - translation properties and pointers to a second level table for Large pages and Small
  - Second level tables hold second-level descriptors, each containing the base address and translation properties for a Small pages or a Large page. Second-level tables are also referred to as Page tables.
- A section translation requires only a first level fetch

Memory Management Formats



(c) Virtual memory address formats

2-Level Virtual Address Translation (Small Pages)



Translation Flow for Small Page

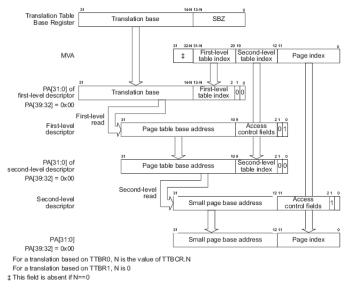


Figure B3-6 Small page address translation

Translation Flow for a Supersection

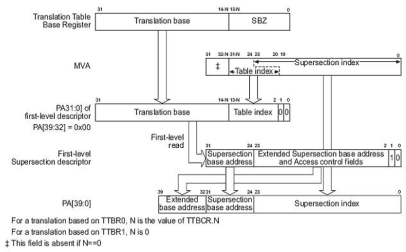
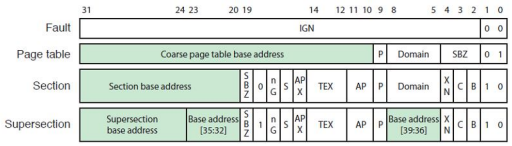


Figure B3-5 Supersection address translation

First Level Descriptor Formats

- Each entry in the first level table corresponds to 1MB of virtual memory
- Entry a descriptor of how the associated 1MB range is mapped.Bits 0,1 and 18 determine the descriptor type



- [1:0]=00 unmapped address, access generates translation fault
- [1:0]=01 entry is physical address of level 2 page table
- [1:0]=10 and [18]=0 section descriptor
- [1:0]=10 and [18]=0 supersection descriptor
- Entries with [1:0] = 11 are reserved

ARM Memory Management Parameters

Access Permission (AP), Access Permission Extension (APX)
These bits control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, a Permission Fault is raised.

Bufferable (B) bit
Determines, with the TEX bits, how the write buffer is used for cacheable memory.

Cacheable (C) bit
Determines whether this memory region can be mapped through the cache.

Domain
Collection of memory regions. Access control can be applied on the basis of domain.

not Global (nG)
Determines whether the translation should be marked as global (0), or process specific (1).

Shared (S)
Determines whether the translation is for not-shared (0), or shared (1) memory.

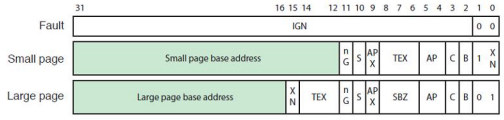
SBZ
Should be zero.

Type Extension (TEX)
These bits, together with the B and C bits, control accesses to the caches, how the write buffer is used, and if the memory region is shareable and therefore must be kept coherent.

Execute Never (XN)
Determines whether the region is executable (0) or not executable (1).

Second level descriptor formats

- Each entry in the 2nd level table corresponds to 4KB of virtual memory
- Entry is a descriptor of how the associated 4KB range is mapped.Bits 0,1 determine the descriptor type



- [1:0]=00 unmapped address, access generates translation fault
- [1:0]=01 Large page descriptor. Bits [31:16] of the descriptor are the base address of the Large page.
- [1:0]=10 or 11. Small page descriptor. Bits [31:12] of the descriptor are the base address of the Small page.
- In this descriptor format, bit [0] of the descriptor is the XN (execute never) bit.

Additional requirements for translation tables

- Additional requirements for the entries in a translation table apply:
  - to first-level translation tables when Supersection descriptors are used
  - to second-level translation tables when Large page descriptors are used.
- These requirements exist because the top four bits of the Supersection or Large page index region of the virtual address overlap with the bottom four bits of the table index.

Replication of Descriptors

- Considering the case of using Large page table descriptors in a second-level translation table, this overlap means that for any specific Large page, the bottom four bits of the second-level translation table entry might take any value from 0b0000 to 0b1111. Therefore, each of these sixteen index values must point to a separate copy of the same descriptor. This means that, in a second-level translation table, each Large page descriptor must:
  - occur first on a sixteen-word boundary
  - be repeated in 16 consecutive memory locations.
- For similar reasons, in a first-level translation table, each Supersection descriptor must also:
  - occur first on a sixteen-word boundary
  - be repeated in 16 consecutive memory locations.
- Second-level translation tables are 1KB in size, and must be aligned on a 1KB boundary. Each 32-bit entry in a table provides translation information for 4KB of memory. VMSAv7 supports two page sizes:
  - Large pages are 64KByte in size
  - Small pages are 4KByte in size.
- The required replication of Large page descriptors preserves this 4KB per entry relationship:
  - (4KBytes per entry) x (16 replicated entries) = 64KBytes = Large page size

Translation table base registers

- Three translation table registers describe the translation tables that are held in memory.
  - Translation Table Base Register 0 (TTBR0)
  - Translation Table Base Register 1 (TTBR1)
  - Translation Table Base Control Register (TTBCR)
- On a translation table walk, the most significant bits of the virtual address and the value of TTBCR.N determine whether TTBR0 or TTBR1 is used as the translation table base register. The value of TTBCR.N indicates a number of most significant bits of the virtual address and:
  - if either TTBCR.N is zero or the indicated bits of the MVA are zero, TTBR0 is used
  - otherwise TTBR1 is used.

Supersections

- Bits [31:24] of L1 entry contain 8 bit pointer to base of the 16MB section in main memory
- Page table entry replication is required
- Range of physical address space can expanded using bits [23:20] and [8:5]
  - Additional bits are interpreted as extending the size of physical memory by a factor of up to 2^8
  - Physical memory can therefore be as large as 256 times the memory available to one process