

Virtual Memory (VM)

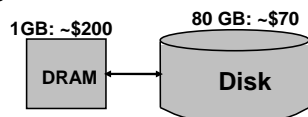
CIT 595
Spring 2008

Virtual Memory

- Gives a program an illusion that it has contiguous working memory
 - Even though main/physical memory may not be as large its address space
- Idea is to treat main memory like a cache
 - Store only a subset of program's address space
- Allows transparent memory management between main memory & permanent storage (disk)
 - Note that VM predates cache

Motivation for Virtual Memory

- Program Address Space
 - Process Address space is 0 to 2^{n-1} where n = machine size
 - > Full address space is quite large. E.g. 32-bit address (with 1 byte storage) = 4 GB
 - Making main memory as large as address space is too expensive



- Original Motivation by IBM in 1970
 - IBM S/370 machine size 31 bits vs. IBM S/360 had 24 bits
 - Compatibility of software between different machine sizes
 - Prevent explicit memory management

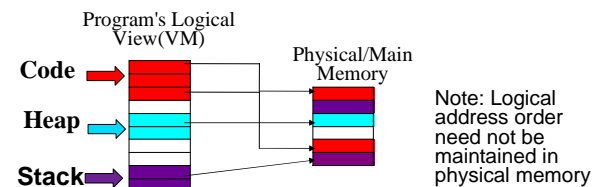
Big Picture: How VM works?

Virtual Address (VA)

- Is the address generated by your program
- Address range 0 to $2^n - 1$ where n is machine width

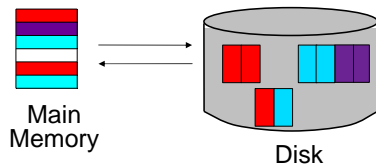
Physical Address (PA)

- Where the virtual address is physically stored in Main Memory (DRAM)
- Address ranges from 0 to $2^m - 1$ where $m < n$



Big Picture: How VM works?

- Use the disk as an extension to main memory
- Address mapping scheme tell us where the VA is actually located in main memory
 - Mapping need not preserve continuity of data in both physical memory and disk
 - Logical address is only from program's point of view
- Address translation from is done by Operating System (OS) + hardware



CIT 595

5

Uses of VM: Multiprogramming

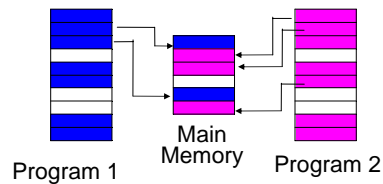
- A **process** is an instance of a computer program that is being executed
- **Multiple** processes can be resident in main memory
 - Many different active processes at same time
 - E.g. Word & Paint application
 - Same program is run in different processes i.e. multiple instances of the same program
 - Same instructions **but** data and program counters are different
- Only **active** part of the code and data of process is in main memory
 - Allocate more memory to process as needed

CIT 595

6

Uses of VM: Multiprogramming (contd..)

- Appearance to the user that different processes are being executed at the same time
- But in reality one process is executing
 - This known as time-sharing the processor
 - O.S decides which processes gets the CPU based on a scheduling algorithm (more on this in chp 8)



CIT 595

7

Uses for VM: Program Isolation/Protection

- VM creates an illusion i.e. each process thinks
 - It has 2^n address space
 - It has its own stack starting at address (say 0x3FFF)
- Map VA of a process to a PA that is separate from another process
 - Prevent processes from reading/writing each others memory

CIT 595

8

Data Size Transfer between Disk & Main Memory

- Due to Principle of Locality
 - Large chunks or pieces of data are transferred between Disk and Main Memory
- Unit of transfer is called a *page*
 - Analogous to *block* transfer between cache and main memory
 - However the page is much larger than block
 - E.g. Transfer 8-32 bytes vs. 2048-8192 bytes for most modern systems

CIT 595

9

Virtual Memory Technique: Paging

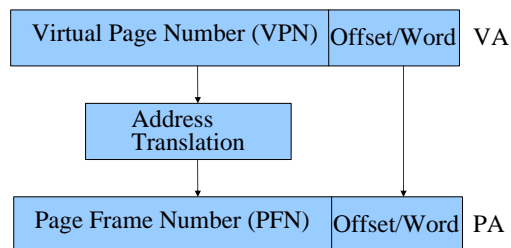
- Allocate physical memory to processes in fixed size chunks called *page frame*
- VA space is divided into pages of equal size
 - Page size is also same as the frame size
 - Each page has same number of words (similar to cache)
- Entire address space required by a process need not be in memory at once
- Pages allocated to a process do not need to be stored contiguously either on disk or in main memory
 - In both cases O.S. keeps a data structures to track actual locations

CIT 595

10

Paging: Address Translation Overview

Virtual Address (VA) is translated into Physical Address (PA)

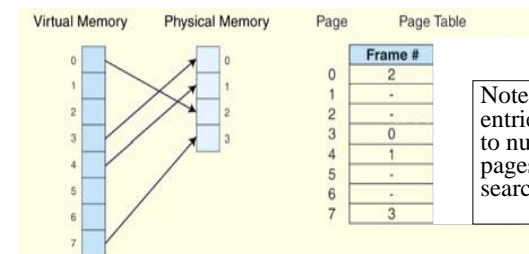


CIT 595

11

Address Translation using Page Table

- Location of each page, is maintained in a data structure called a *page table*
 - Updated by Operating System (OS)
 - Placed in memory at a known location
 - Virtual Page number is used to index the page to find which frame in Physical Memory is the data located



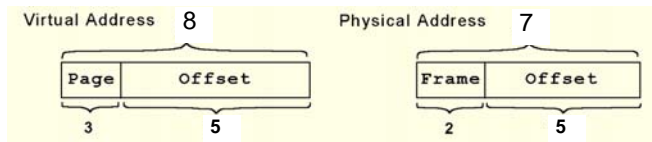
Note: Page Table entries are equal to number of pages to minimize search time

CIT 5

12

Example

- A system has a virtual address space of 2^8 and a physical address space of 2^7 . Further assume that each page has 32 words
 - A virtual address has 8 bits
 - Of the 8 bits, 5 bits are used for offset ($2^5 = 32$ words)
 - Remaining 3 bits are used for Virtual Page Number (VPN)
 - Since physical memory address is 7 bits, 2 bits are used for Page Frame Number (PFN)

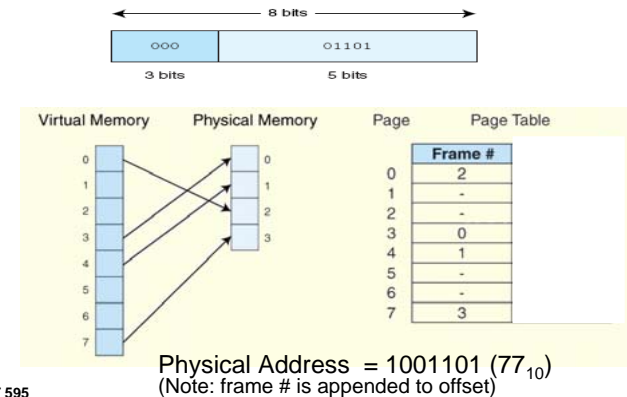


CIT 595

13

Example (contd..)

- Virtual Address 13 is produced by the processor for a process, what is the Physical Address?



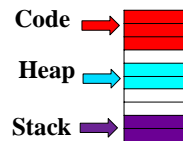
CIT 595

14

Paging in Multiprogramming Environment

- Each program's logical view is the same

Program's Logical View



- Problem: Different processes use the same virtual addresses
- Solution: Each process has its *own* page table
- OS responsible for *updating* page tables so that virtual address spaces of different processes do not collide

CIT 595

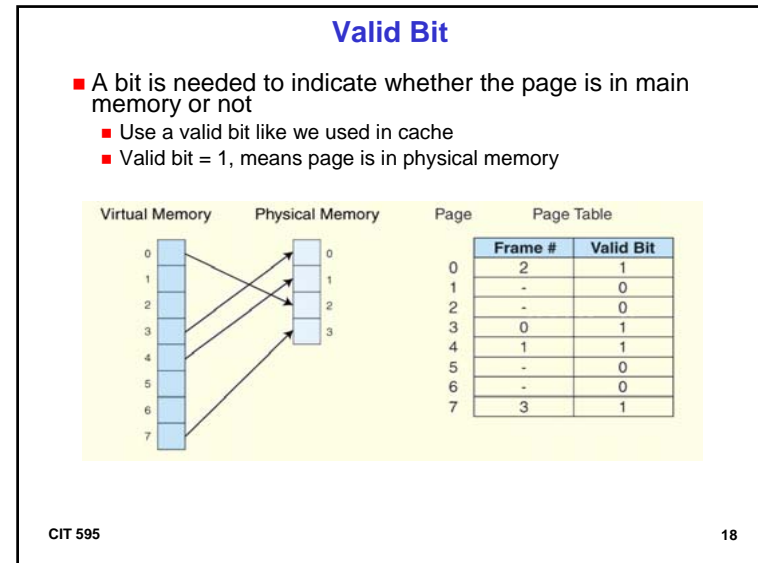
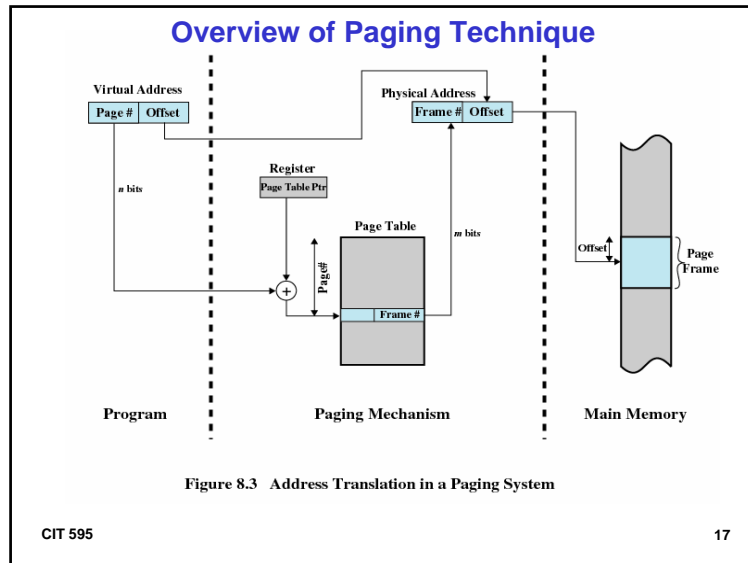
15

Paging in Multiprogramming Environment (contd..)

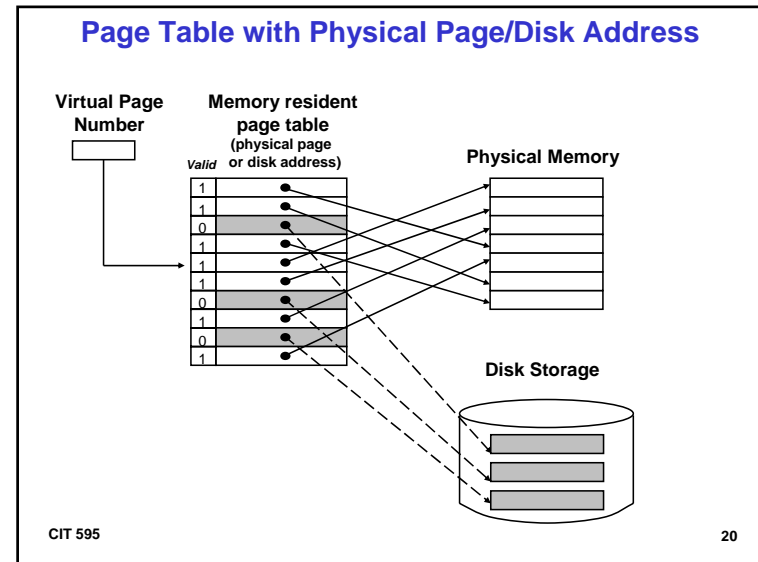
- Processes exist in memory time share CPU
 - OS switches between process
 - Known as context switch
- Save *state* of existing process before switch
 - Save PC, registers and page table
 - Store only the address where the first page table entry is located
- On switch, OS loads the state of the new process
 - Page table address is loaded into an internal register
 - Known as *Page Table Pointer*
 - Like PC register is used to load the value of PC

CIT 595

16



- ### Page Fault
- If valid bit is 0, then page is not in physical memory
 - This is known as an occurrence of *page fault*
 - An exception is thrown and the OS intervenes
 - Finds page required from disk & loads it into main memory
 - May need replacement policy if all frames are occupied
 - OS also maintains a data structure to record where each virtual page is stored on disk
 - Can be part of the page table or a separate structure
- CIT 59519



Write Policy

- Modified data in physical memory must also be updated in the disk
- Disk access time is very slow
 - 5ms – 20ms compared to DRAM access time of 30-90ns
- Write-Back** policy is employed i.e. update disk only when the page is going to be replaced
- Dirty** (Modify) bit indicates if the page has been altered since it was last loaded into main memory
 - If dirty = 1, then write-back page to disk upon replacement

D	V	Page Frame # or Disk Addr
1	1	
0	1	
0	0	
0	1	
1	1	
1	1	

CIT 595

21

Sample OS code for VM

```
#define NUM_VIRTUAL_PAGES 1000
struct {
    union{
        int pageFrameNum;
        int diskAddr;
    }
    int isValid;
    int isDirty;
}PTE; //1 page table entry

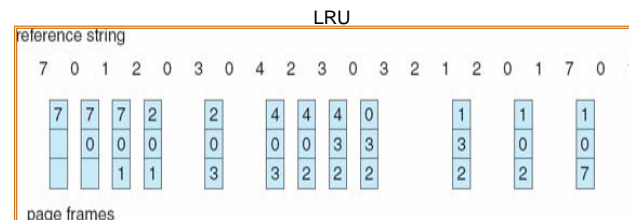
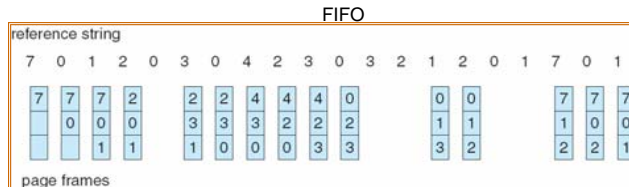
//Array of page table entries
struct PTE pageTable[NUM_VIRTUAL_PAGES];

//Lookup table for pageFrameNum
int addressTranslation(int vpNum){
    if(pageTable[vpNum].isValid)
        return pageTable[vpNum].pageFrameNum;
    else
        return -1;
}
```

CIT 595

22

Page Replacement (Just like cache block replacement)



- Unlike caches, replacement algorithm can be done in software

CIT 595

23

EAT Example

- Suppose a main memory access takes 200ns, the page fault rate is 1%, and it takes 10ms to load a page from disk.
- What is Effective Access Time?

$$EAT = 0.99(200\text{ns} + 200\text{ns}) + 0.01(10\text{ms}) = 100396\text{ns}$$

CIT 595

24

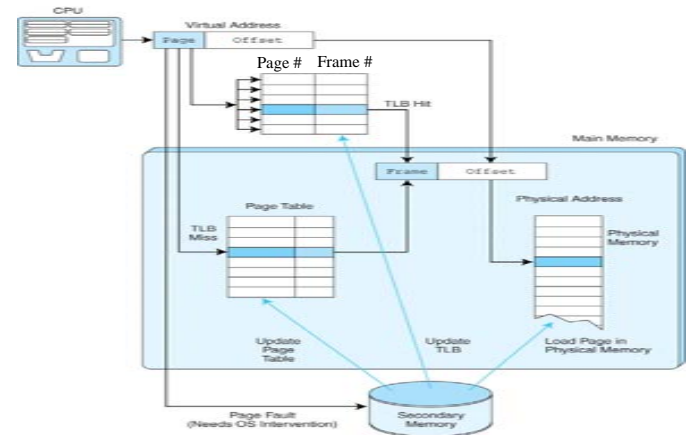
Accelerating Address Translation

- Each virtual memory reference causes 2 physical memory accesses
 1. fetch the page table
 2. fetch the data
- Solution: *Translation Lookaside Buffer* (TLB)
 - High-speed memory
 - TLB is made of SRAM technology
- TLB stores the *most frequently* used mappings
 - Stores page number and corresponding frame number
 - Fully associative in terms of mapping
 - SRAM is costly in terms of storage
 - Hence use it store the data that you most need

CIT 595

25

Using TLB



CIT 595

26

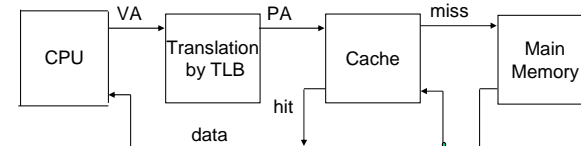
Modern Processor with Cache and Virtual Memory

- Caches indexed using physical/main memory address
- But processor (CPU) generates a logical/virtual address
- So then how does the whole memory system work?
- Problem: Require Address Translation before Cache lookup
 - Involve a memory access itself (page table lookup)
- Solution: We know that page table entries can also be cached
 - Hence use TLB to translate VA to PA and then use cache

CIT 595

27

Integrating Cache and VM

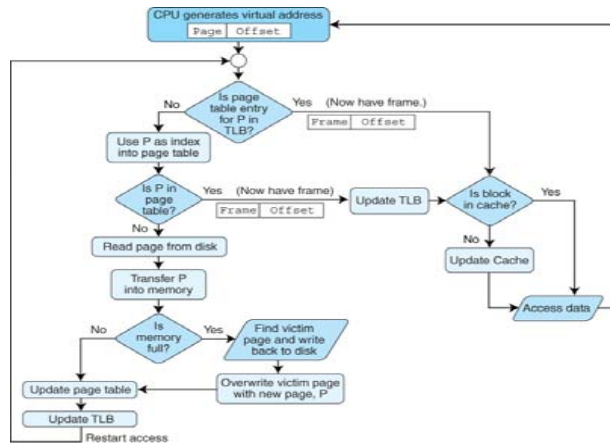


- Advantage of Integration:
 - Allows multiple processes to have blocks in cache at same time
 - Access time for data is faster
 - SRAM (TLB and Cache) vs. two DRAM (main memory) accesses

CIT 595

28

All Memory put together: Cache, TLB and Paging VM



CIT 595

29

Disadvantage of Paging Technique

- A process may not need the entire range of addresses contained within the page
- There may be many pages containing unused fragments of memory
 - Known as *internal fragmentation*
- Unused fragments could allow more processes to be existent in physical memory
 - Constraint due to fixed page size

CIT 595

30

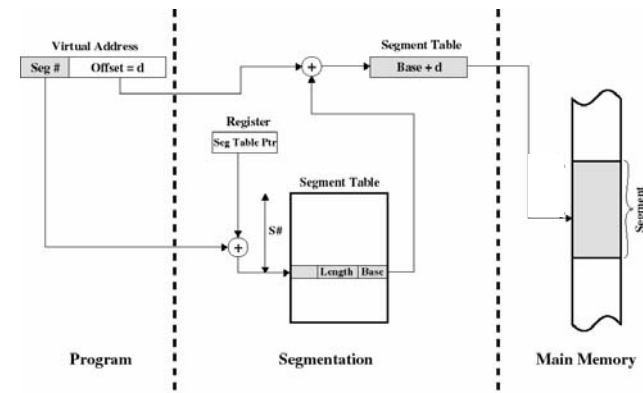
VM Technique: Segmentation

- VA space into variable-length units called *segments*
 - Physical Memory is not partitioned
- On a page fault, O.S looks for free chunk of memory large enough to fit the segment
- Each segment contains <base, bound> pair
 - *Base* address indicating where in memory it is located
 - *Bound* (or length) indicates the size of the segment
 - This information is stored in *segment table*
- VA to PA translation is the same except offset is added to the starting address
 - Instead of being appended like in paging

CIT 595

31

VM with Segmentation



CIT 595

32

Disadvantage of Segmentation

- Allocation and de-allocation cause free chunks in memory to become broken into small chunks
- End up with many small chunks but none larger to store an entire segment
 - Not contiguous enough to fit the entire segment
 - Known as *external fragmentation*
- Some point will need to defragment
 - Collect small free chunks into large one by moving around existing occupied chunk

Protection of Pages

- Prevent process from accessing another's memory
- Non multi-programming
 - Only OS access to system memory
 - E.g. Implementation in LC3
 - Memory Protection Register (MPR)
 - Processor Status Register (PSR[15]) to indicate Supervisor or User Mode
- Multi-programming
 - Add Read/Write/Execute protection bits in page table
 - Attempt to illegal access i.e. to execute or write to read-only
 - Causes exception and OS terminates program
- Protection can be limited though
 - Viruses and worms can easily exploit unchecked memory bounds in program
 - E.g. Overwrite return address of function with pointer to attack code

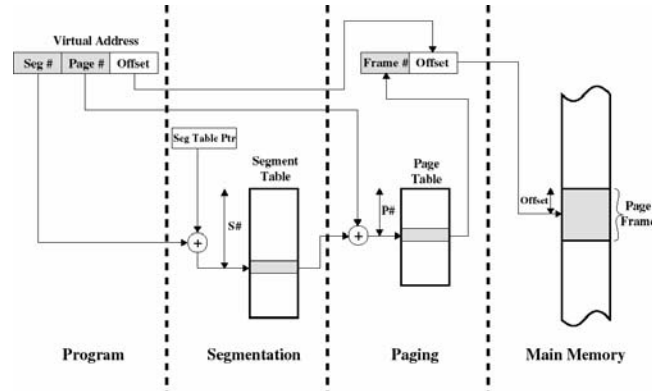
Sharing Pages

- Why share pages?
 - If we share the same code or data among different processes, it is sufficient to keep only one copy in main memory
 - E.g. Common subroutines are shared among processes
- Sharing is implemented by making shared pages read-only
- Sharing is easier to implement using segmentation
 - As segments semantically can represent defined portions to be shared
 - E.g. Code, Heap, Stack segments
 - Protection on the segmented regions then can be set accordingly
 - In paging, semantic portions can be across different page
 - Harder to set protection as you might want part of the data sharable and part to be exclusive

Paging Combined with Segmentation

- Modern systems employ combined paging and segmentation
- Take advantage of the best features of both
- Assign fixed-size pages within variable-sized segments
- Each segment has a page table
 - VA is broken into 3 fields segment, page, and offset

Paging Combined with Segmentation



CIT 595

37

Advantages of VM

- Enhances performance by providing greater memory capacity, without the expense of adding main memory
- Programmer does not have to worry about address space i.e. illusion of *very large memory*
- To a process it appears as if it “owns” machine
 - Has private address space
 - Unaffected by behaviour of other processes

CIT 595

38