Memory Management

- Introduction
- Memory Allocation
- Job Scheduling
- Job swapping
- Memory partion and fragmentation
- Protection

Introduction

Why Memory Management?

- one memory is shared by number of processes and need to:

1- Allocate part of memory to each process

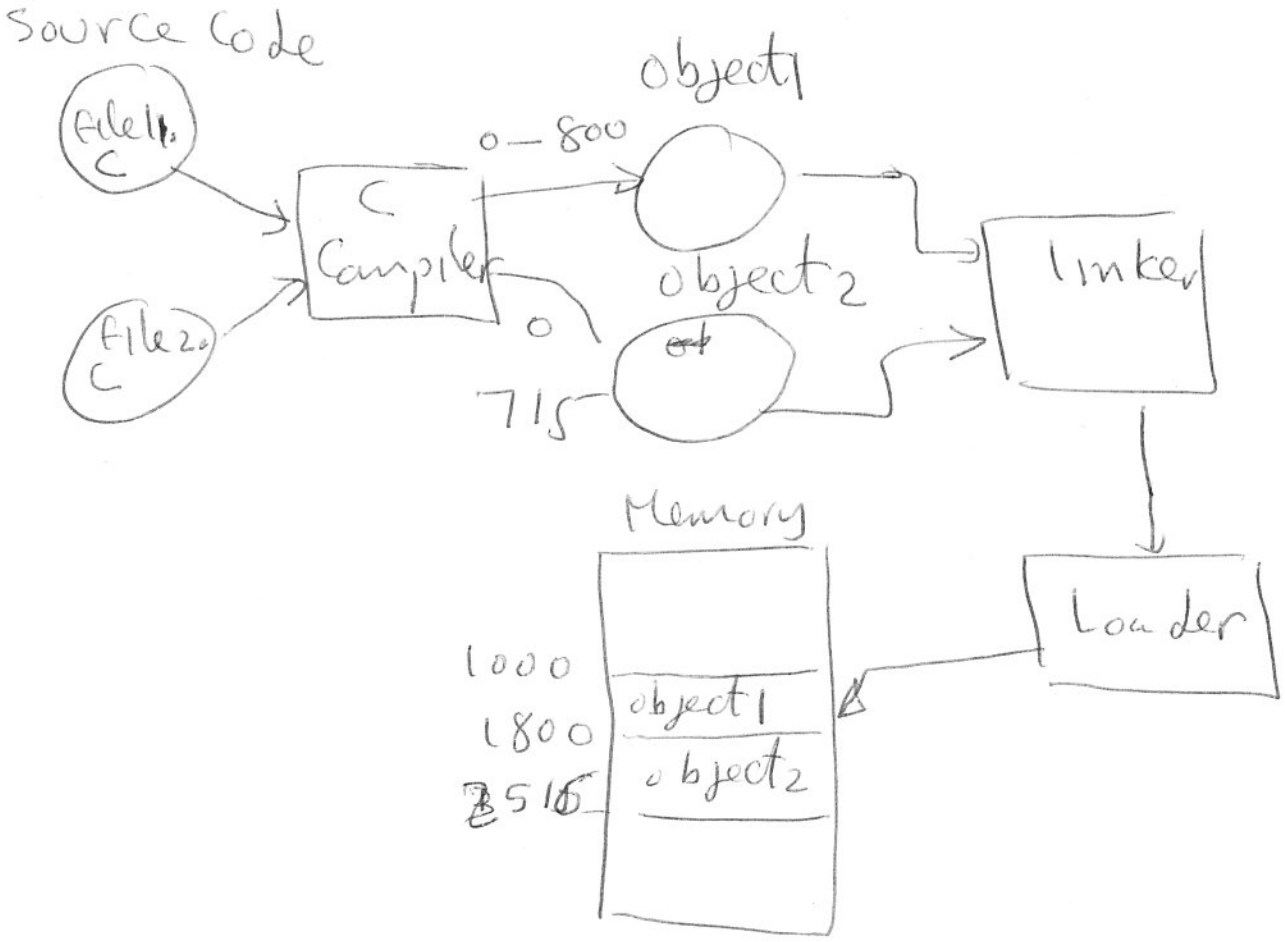
2- Schedule each process to part of memory

3- Replace and swap processes

4- Protect processes from each other

Mapping Programs To Memory
Use Load-and Link

Executable binary file reside in disk.
Must be transfered To memory (where?)



Linker recalculate addresses based on available memory address and binding or stecking one end to other.
object 1 starts at 1000 – ends at 1800, object 2 starts at 1801 ends at 1516

Memory Management gives Linker and Loader values of starting (new) address in memory (Allocation).

Memory Allocation with overlays (User)

- If a process has a Logical Address that should all reside in physical memory.

AND

- Physical memory size is not enough

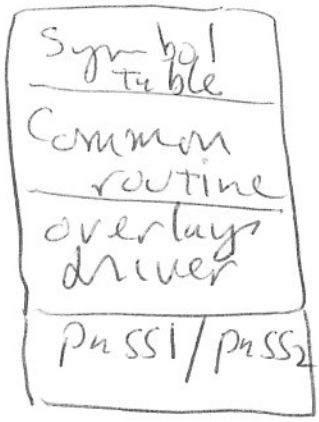
Memory Allocation with overlays could be used.

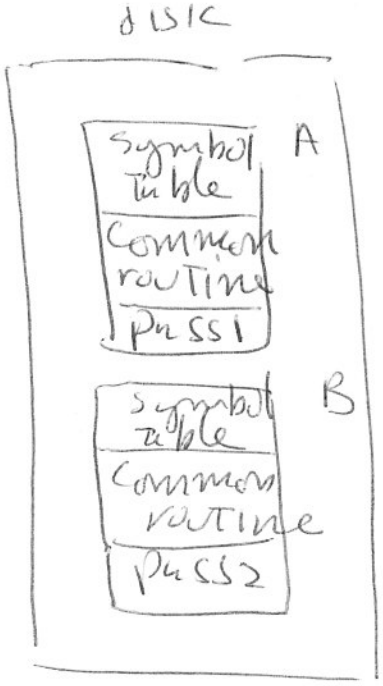
Concept of using overlays:-

Keep in memory only instructions and data that are needed at that given time (not all program is needed all the time).

Example: 2 Pass Assembler

overlay driver routine calls pass2 when 1st pass finish.





problems:

1- programmer has to manage memory

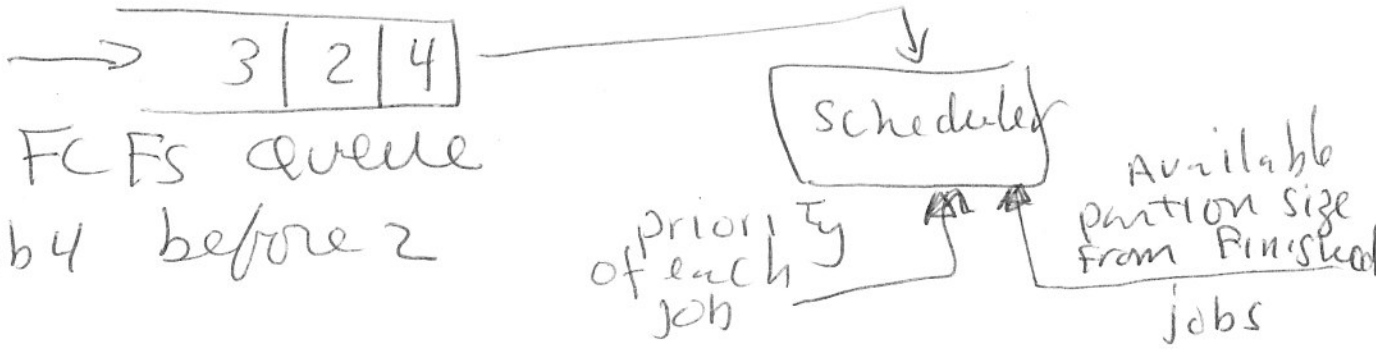
2- Need to know structure and behavior of programs before it is loaded.

Solution U.M.

Memory Management
Job Scheduling

Need: Job Queue and Scheduler

queue: (for jobs) as they enter the system using time stamp.



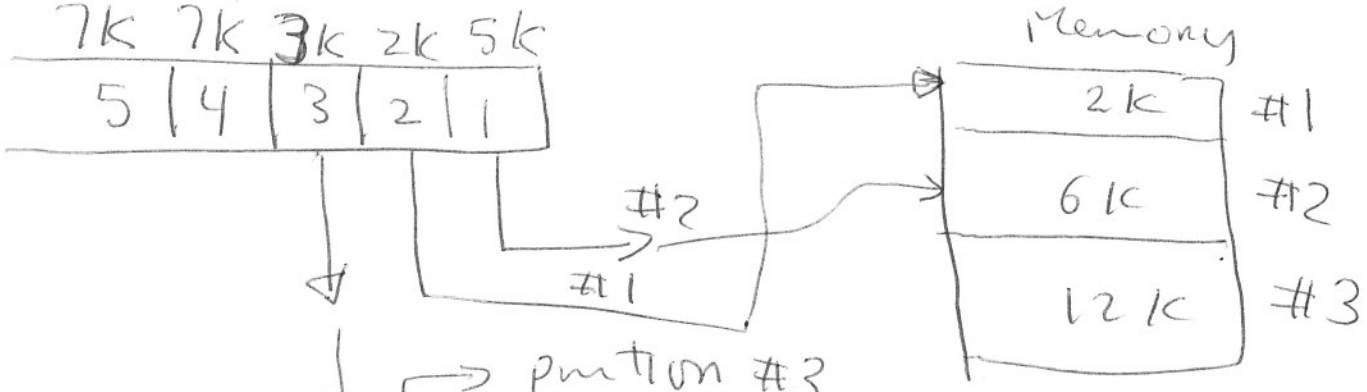
Example

Job #	size	arrival time
1	5K	0
2	2K	1
3	3K	2
4	7K	3
5	7K	4

Memory :

partion #	size
1	2K
2	6K
3	12K

1- Construct queue (Time of arrival)



partion #3 (12K) best available

or wait for job1 to finish and assign job 4 (7K) to partion #3

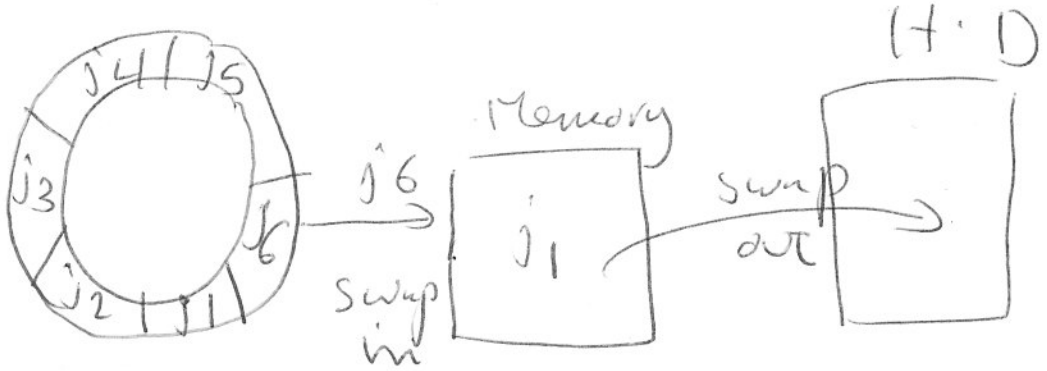
best fit Algorithm

performance of Algorithm is important

Memory Management
Job Swapping

Each job is given a time slice of cpu (quantum) and has to be swapped.

1- Using round robin Scheduling



if high priority job arrives and partion is busy with a lower priority job, Memory manager can swap IN higher priority job and swap out lower priority job.

Context Switch Time:

Transfer of job out + Time to transfer job in < Time of quantum

Job Transfer Time is prop. to job size.

Memory Management
Partion size selection

Large partion size: only small number of jobs, but over head of swapping is reduced

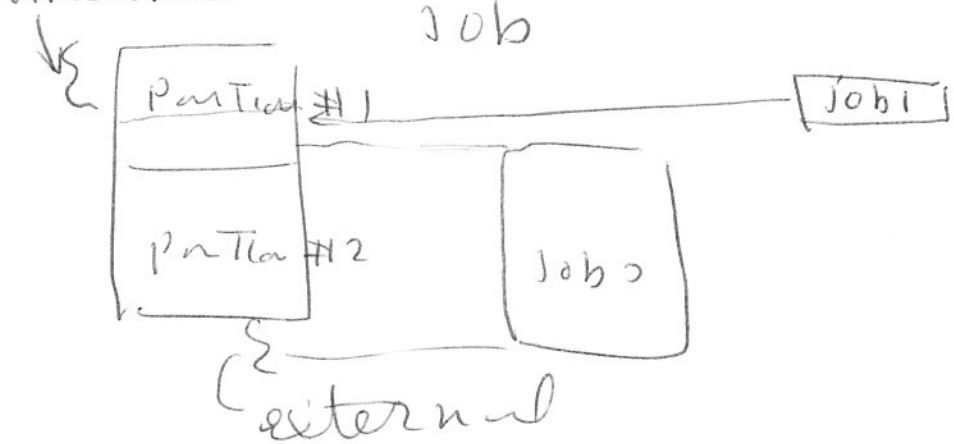
small partion size: high overhead of swapping

Fragmentation :-

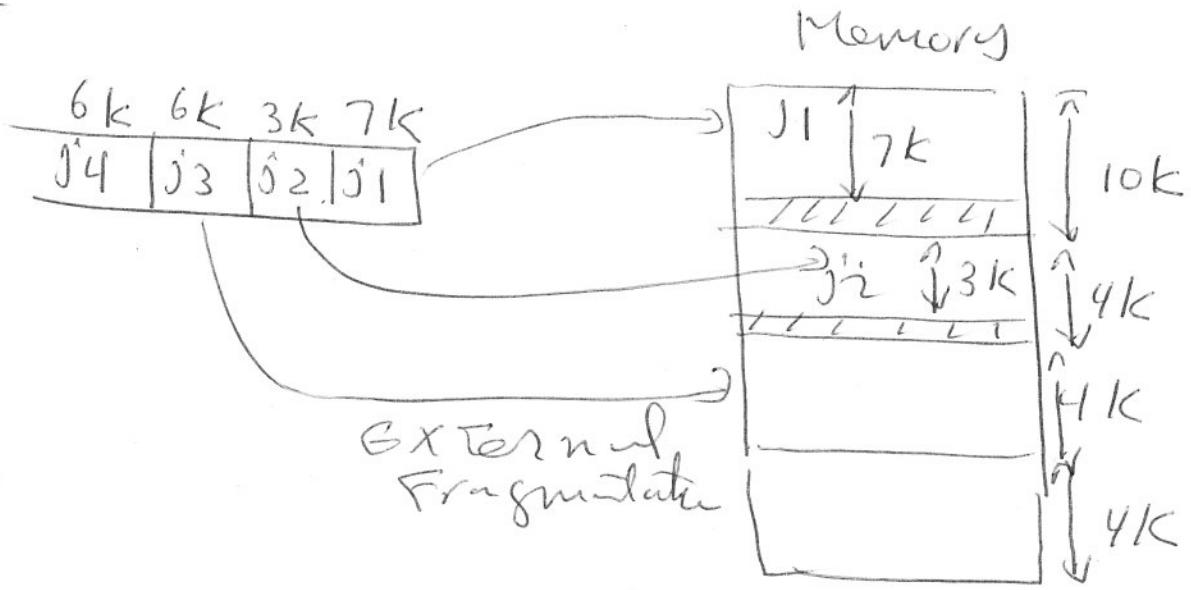
Two types :

1- Internal fragmentation if size of job < size of partion

2- External fragmentation if partion is too small for the job



Example



Solution To Fragmentation

Compaction

Move all jobs towards one end and holes towards the other end.

Example In the above

Can make larger portion to fill hole

Memory Management
Protection

One Program or Job should not interfere with other.

1- Need two registers for each process, one for lower bound and the other for upper bound.

If process exceeds lower or upper bounds, system should generate a Trap.

2- Use limit register.

If process logical address > limit

system should generate a Trap.

Physical Address = base + (logical) only if < limit

For simpler memory management use virtual memory

Virtual Memory

1- Memory Hierarchy

2- VM Concept and definitions

3- Mapping and organization

4- page size and Fragmentation

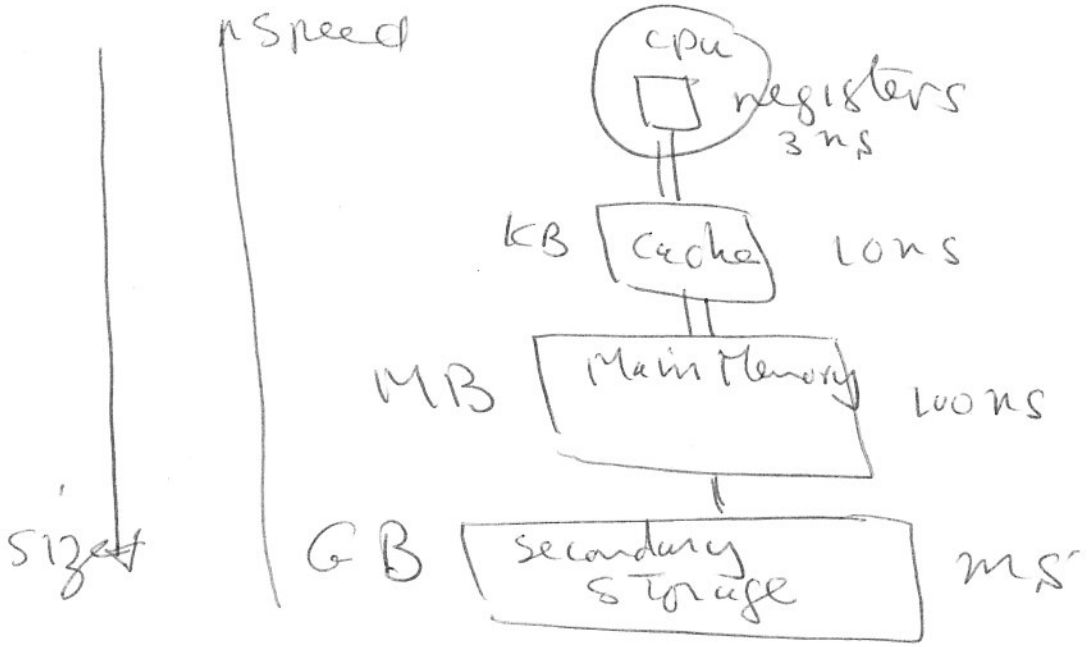
5- page replacement Algorithms

6- write policy

7- TLBs

8- Protection

Memory Hierarchy



Memory Hierarchy to bridge the gap between CPU speed / slow secondary storage
need Multi-Level

Virtual Memory Concept

It provides illusion of very large memory space, so a job that is larger than physical memory can use VM.

Advantages: - very effective utilization of physical memory

- improve performance at low cost secondary storage (HDD)
- Main advantage is it simplifies memory Management (allocation, protection, replacement, ...)

Definitions:-

page = UNIT of information to transfer between MM and DISK = block

Page Fault: when an access is not found in Main memory, system has to get it from DISK ≡ Miss

size of VM ≡ all processor address space.

In cache the size depends on address space.

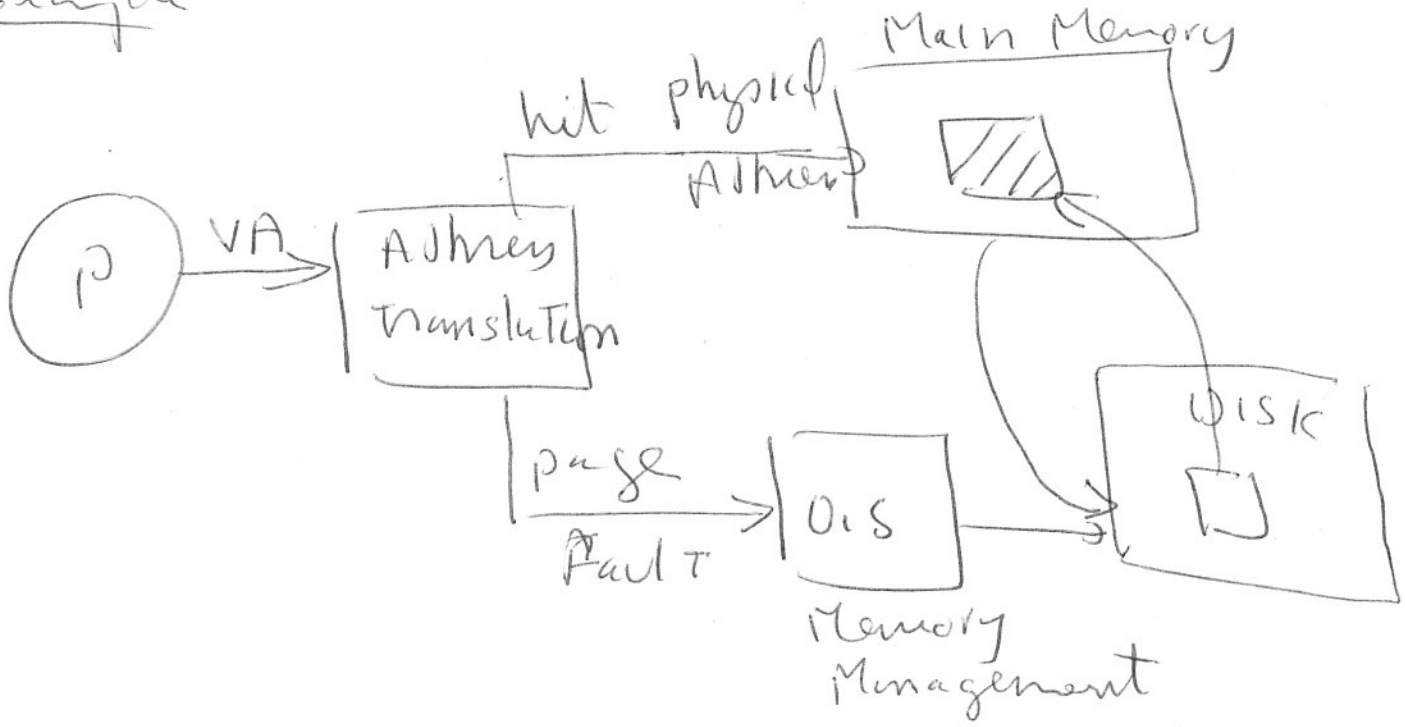
Replacement in VM = use software protocols

in cache use H/W controlled (LRU)

3- VM mapping and organization using paging

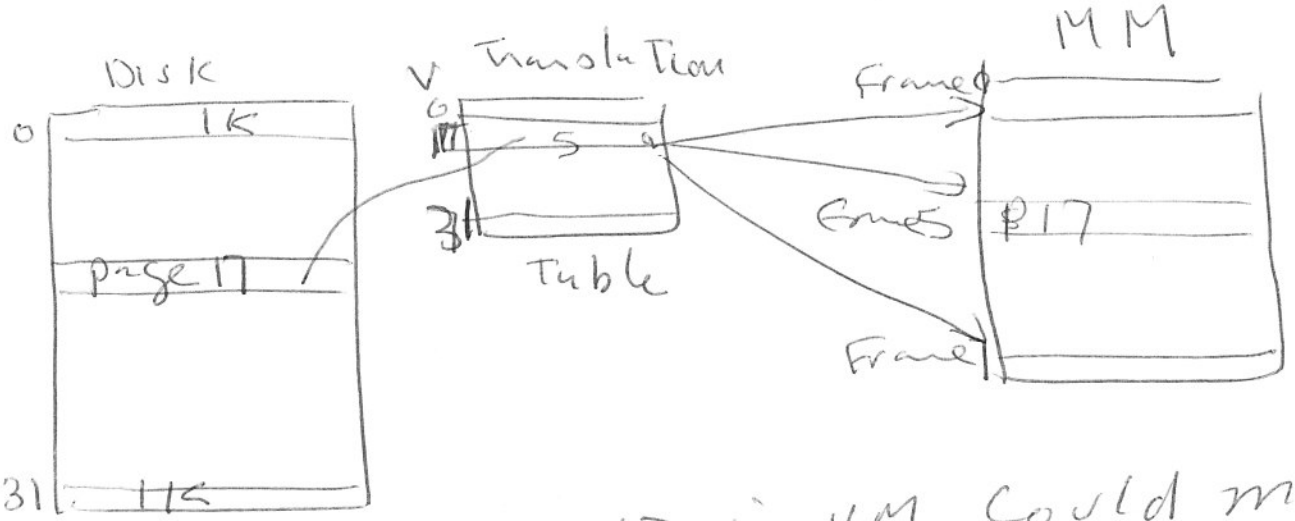
Mapping: because of high cost of page fault, mapping use fully Associative placement policy.

Example



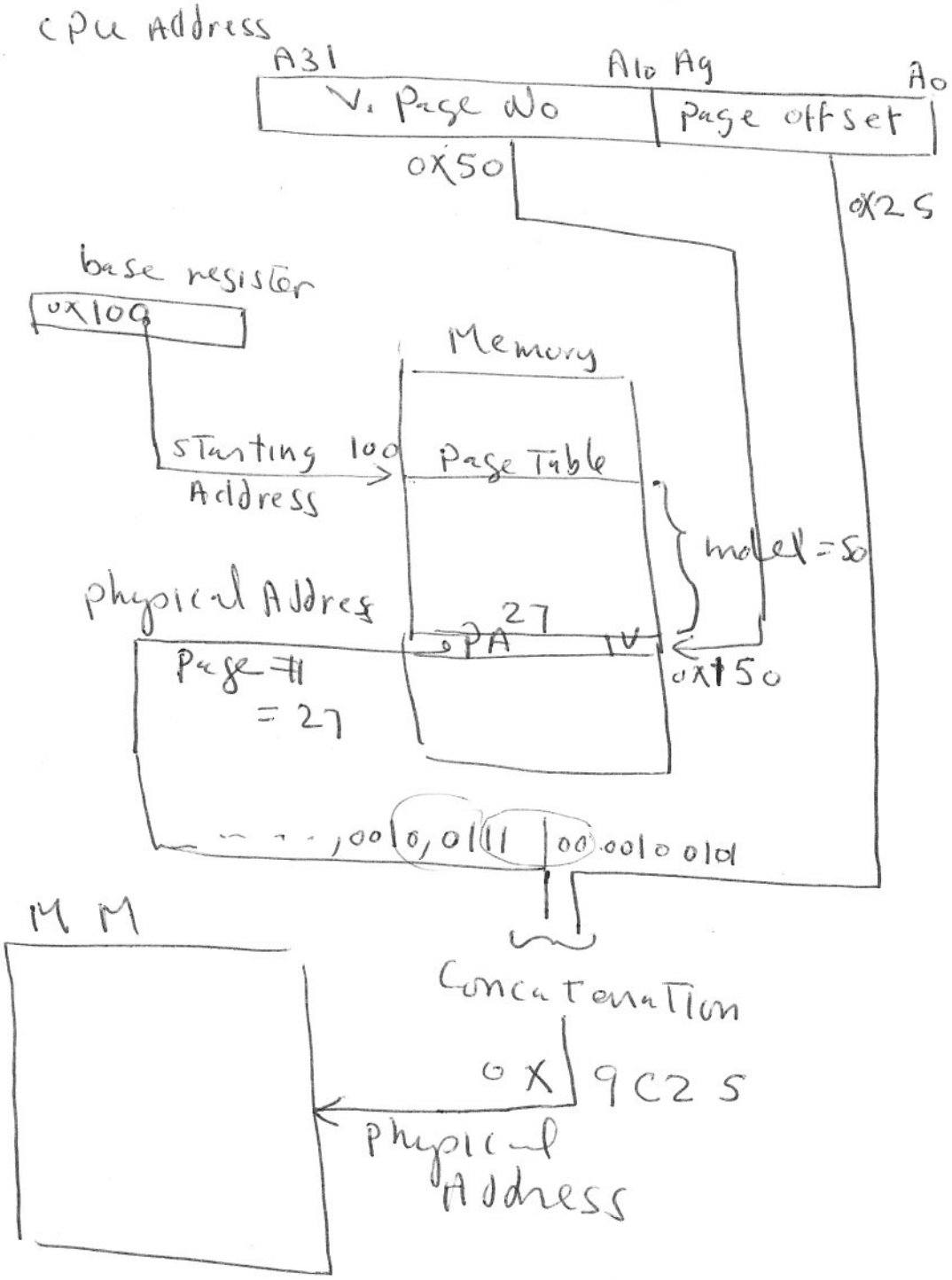
Example

If VM size = 32K, page size = 1K
Main Memory = 8K



page 17 in VM could map
To any MM frame
(fully Associative)

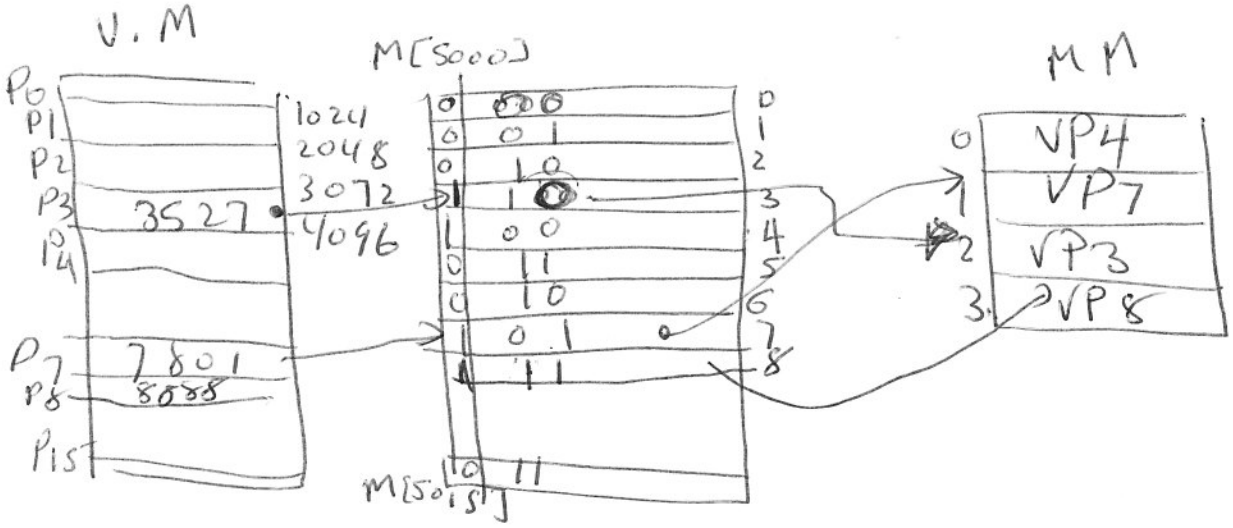
VM Organization



Example Assume that VM size = 16KB,
MM size = 4KB, page size = 1KB.

Base register = 5000 and content of
M[5000 – 5015] = 0, 1, 2, 6, 4, 3, 2, 5,
7, 2, 1, 1, 0, 3, 2, 0, 3

IF MSB in page Table is used for V bit,
find number of page faults for the
following VM accesses:
3527, 4327, 3880, 7801, 8888,
13582, 7527



3527 in V.M P#3, Translation Table shows
V=1 ⟶ physical page #2 hit
4327 in VM P#4 ⟶, valid bit=1
PP#0 hit
3880, hit in MM P0
7801 in VM P7 maps to MM P1
hit

Access	V.P #	P.P #1	hit/Fault
3527	3	2	hit
4327	4	0	hit
3880	3	2	hit
7801	7	1	hit
8888	8	3	hit
13582	13	—	miss
7527	7	1	hit

high hit rate

4- Optimal page size

- large page size reduces page table size
- Transferring large pages is more efficient (overhead is reduced)
- TLB entries are minimized, this reduces TLB misses

small pages could conserve storage and reduce internal fragmentation

Average size 4K - 8K, trend is to use large page size (DRAM size is increasing)

5 - Page Replacement Algorithms

- Need to minimize swapping
  (Page should stay in Main Memory for longest time)

1- First IN First out Algorithm:
on page Fault, replace the page that has been in memory for the longest time.

Easy to implement (does not use locality)

2 - Least recently used LRU Algorithm

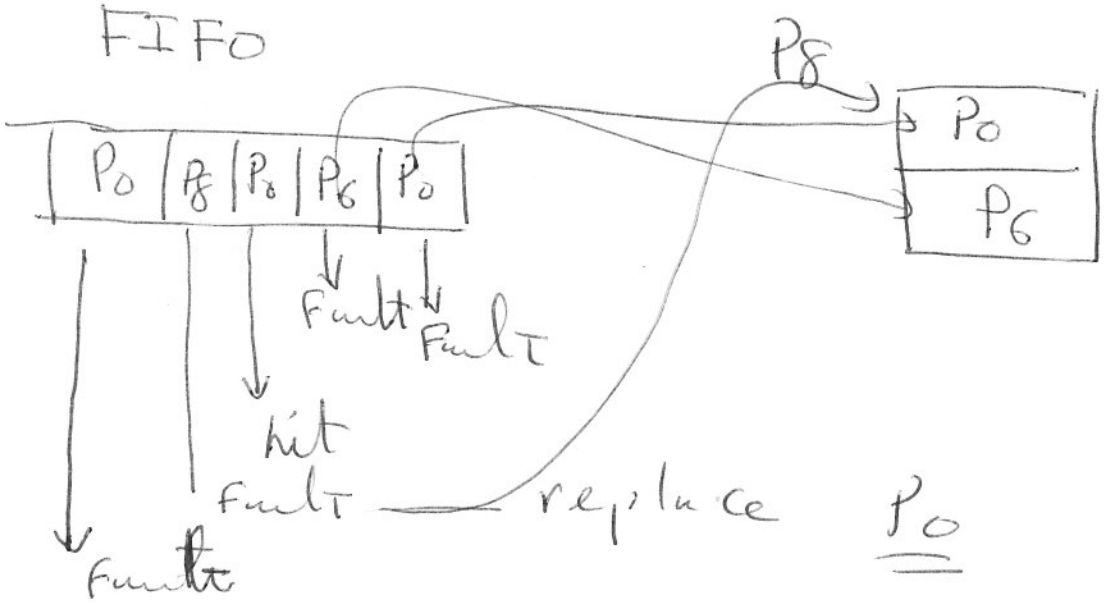
replace Page that is least recently used.

good performance and effeciant (locality)

Implementation: use a pointer each time a page is used, it moves pointer to next page. Replace the page that has the pointer

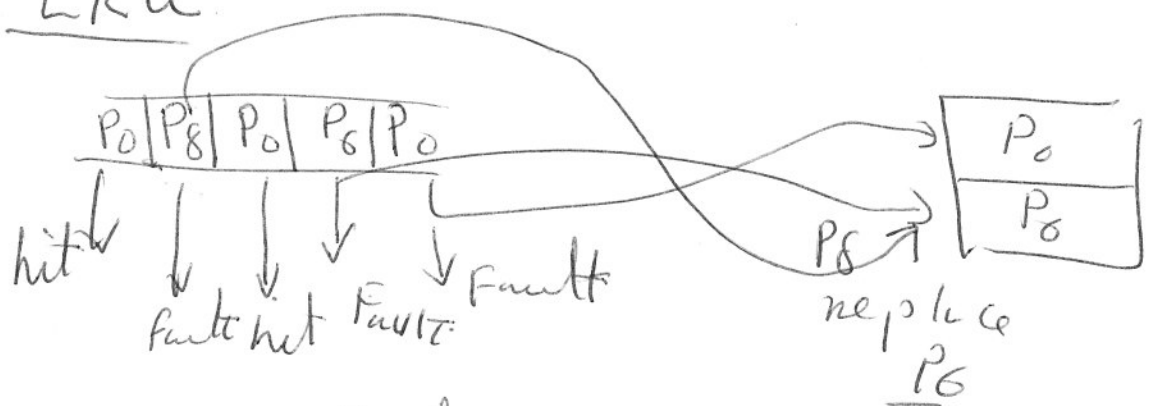
Example Find Page Fault for the following page accesses P_0, P_6, P_0, P_8, P_0 using FIFO and LRU algorithms, if MM consists of 2 pages

FIFO



Page Fault = 80%

LRU



Page Fault = 60%

6- Page Fetching Policies

1- Demand paging: Pages are loaded to memory on faults only

2- Prepaging: can go and fetch pages when anticipated for future references (cost and could replace active pages)

7- Write Policies

- H.Disk is very slow
- Always use write back policy to MM
- Need dirty bit in page table (M bit) for modified pages
- Modified pages have to be written back to H.D when replaced

FAST Address Translation using TLB

Translation Table is in Main Memory, Every access has to Do:-

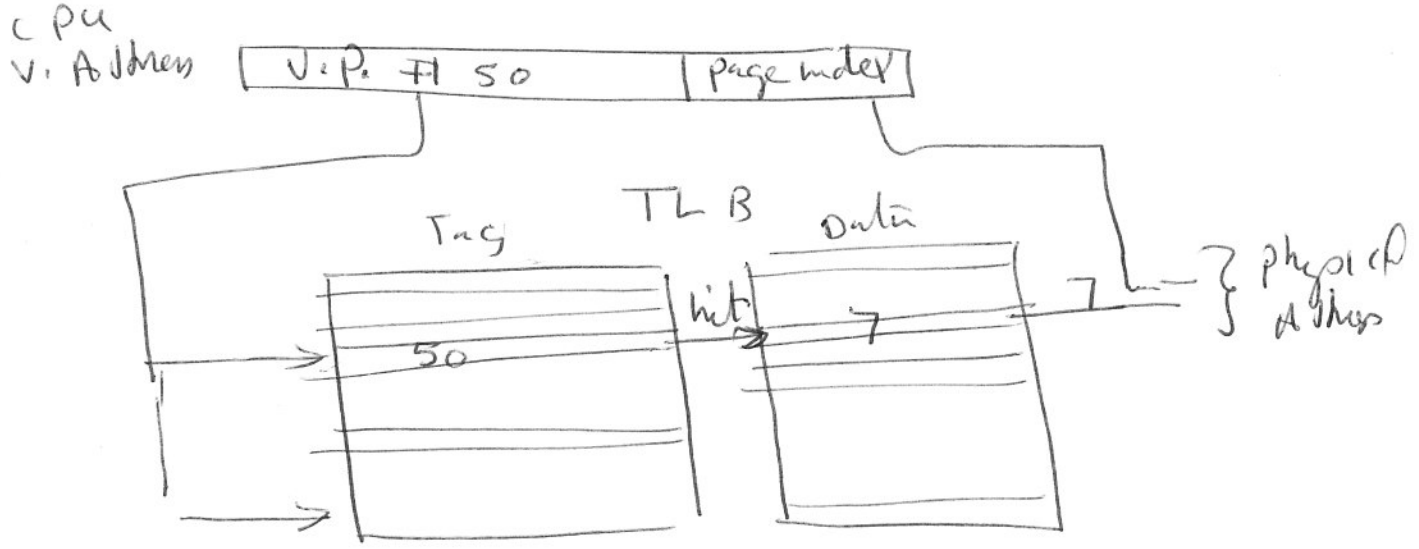
1- Memory Access for Translation Table

2- Access Main Memory on Page hit

total = 2 Memory Accesses (Slow)

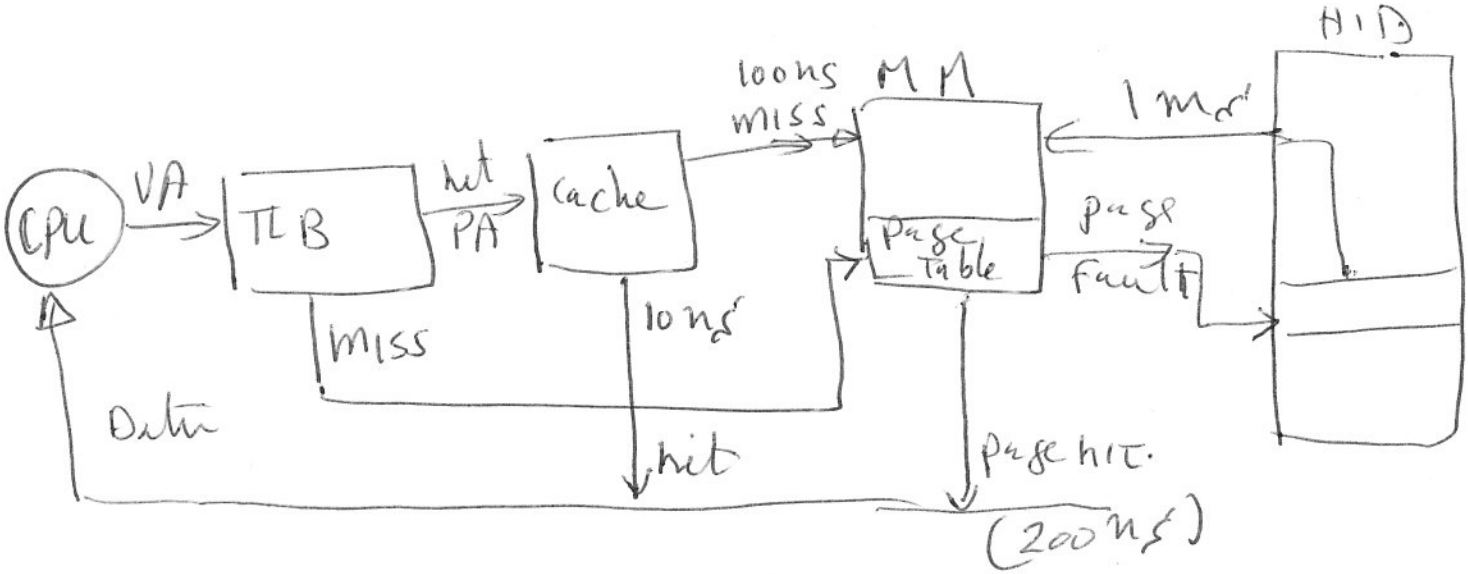
SOLUTION: Translation Lookaside Buffer
Cache address translation in CPU using Buffer

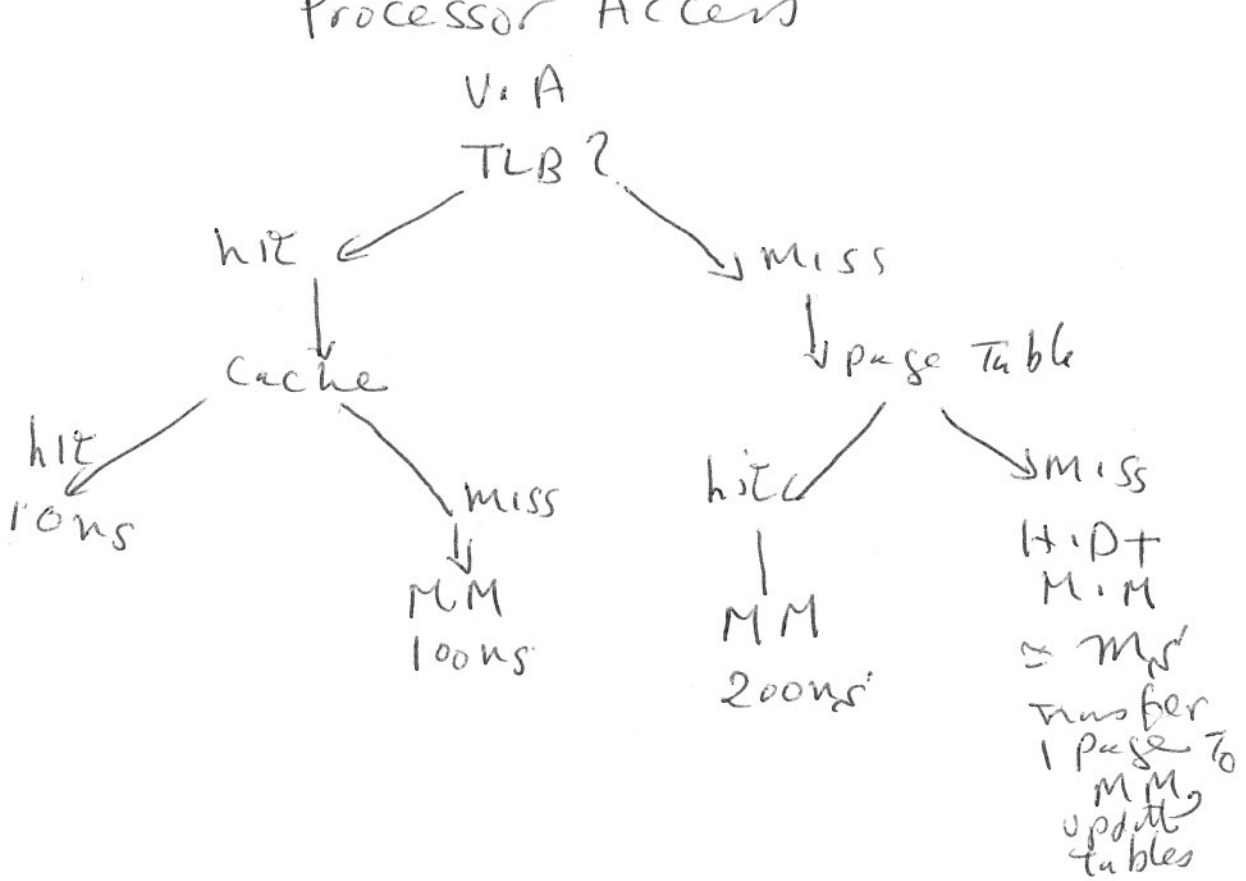
TLB organization



---

overall system organization for Memory Hierarchy





Example

Assume TLB with 2 locations, uses fully Ass. cache. Tag contents = 7, 3 and Data = 5, 6.

MSB of Data is used for valid bit.

Page Table uses ~~4KB MM~~ and Page size = 1KB, contains the following from the base:

0	1	2	6	4	3
2	5	7	2	1	
0	1	3	2	0	3

MSB is used for valid bit.

VM size = 16KB, MM = 4KB

Find overall performance of the following Accesses 3577, -- Assume cache time = 10ns, cache hit = 10ns, MM = 100ns H.D = 1MN / Bus B.W = 10 MB/s

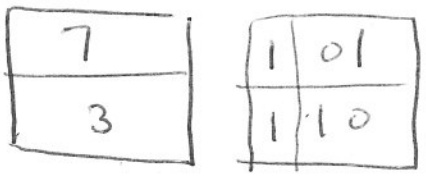
ASS#3 Assume TLB, Page Table, cache = 64 locations, all start from invalid state. Find Performance of

```
for(i=0; i<74000; i++)
    a[i] = 5*a[i];
```

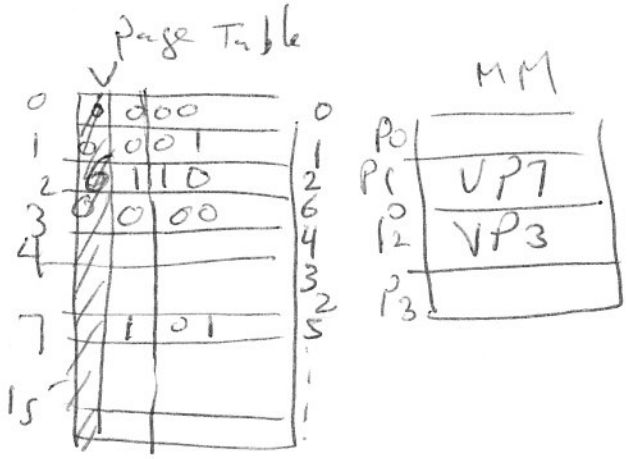
if a[0] = ϕ
cache BLK = 2 elem

Access 3,577 in VP 3 → 3072

offset = 3577 − 3072 = 505



→ TLB hit
PP = 2
physical Address
= 2048 + 505
= 2553



Ass #3 find performance of the following

code: for (i=0; i<4000; i++)
a[i] = 5 * a[i];

a[0] at address 0

using TLB of 2 loc, VM size = 16KB
Page size = 1KB, MM size = 4KB,
cache size = 256 Byte, direct mapped
Block size = 2B. each element of
a[i] = 1 byte.
cache speed = 10 ns, MM speed = 100 ns
HD Time = 1 ms, I/O bus BW = 10 MB/s
All caches, TLB, Page Table start from
invalid state.

Protection

Simpler with VM
VSems each page entry in Translation table to include Access rights.

Translation table

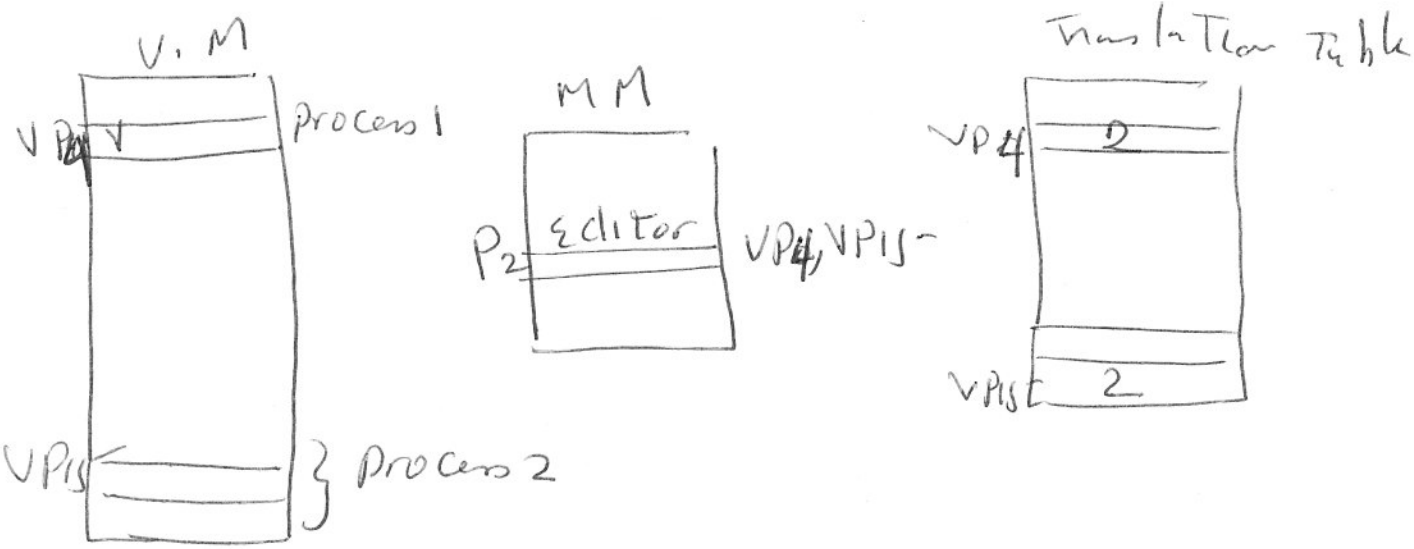
protection	V	M	physical Address

n bits for protection and Access rights

n bits — no access
n bits — read only
n bits — read/write
} user/system

Shared Pages

in VM, processes can share pages



V.M with Segmentation

1- Concept

User view memory as variable size segments with no ordering among segments (no $P_1, P_2, \ldots$)

Program is divided to segments of variable sizes (module1, . - modulen).

Logical Address space is divided to segments of variable size and each has (number and length)

2- Motivation

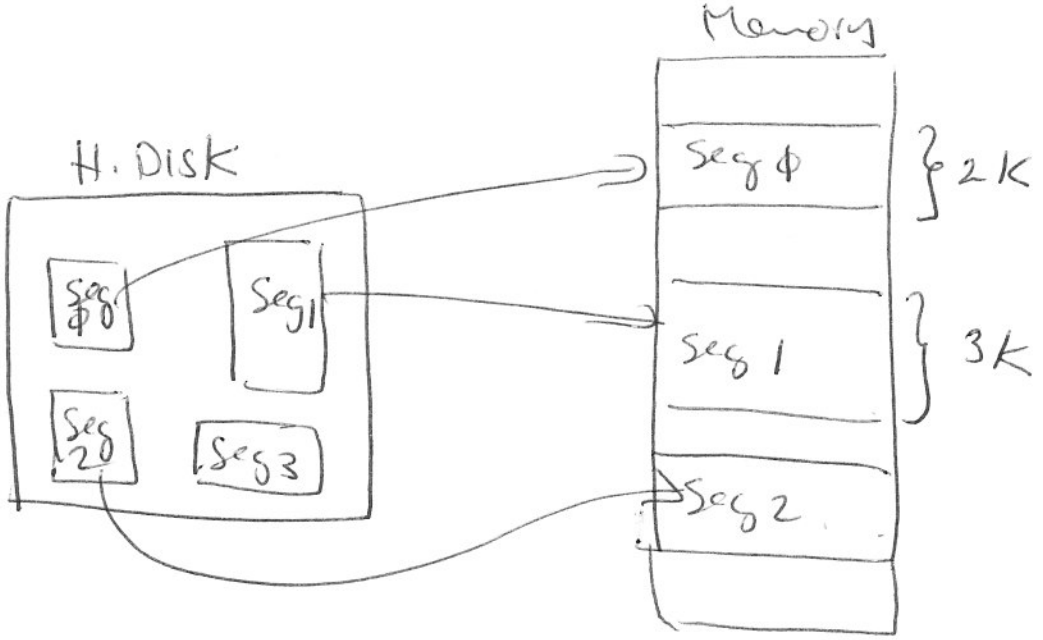
- Small segment table size
- reduce internal fragmentation (each fits module) no waste.
- Protection is simpler, it isolate code from data and have read only for code.

Disadvantages

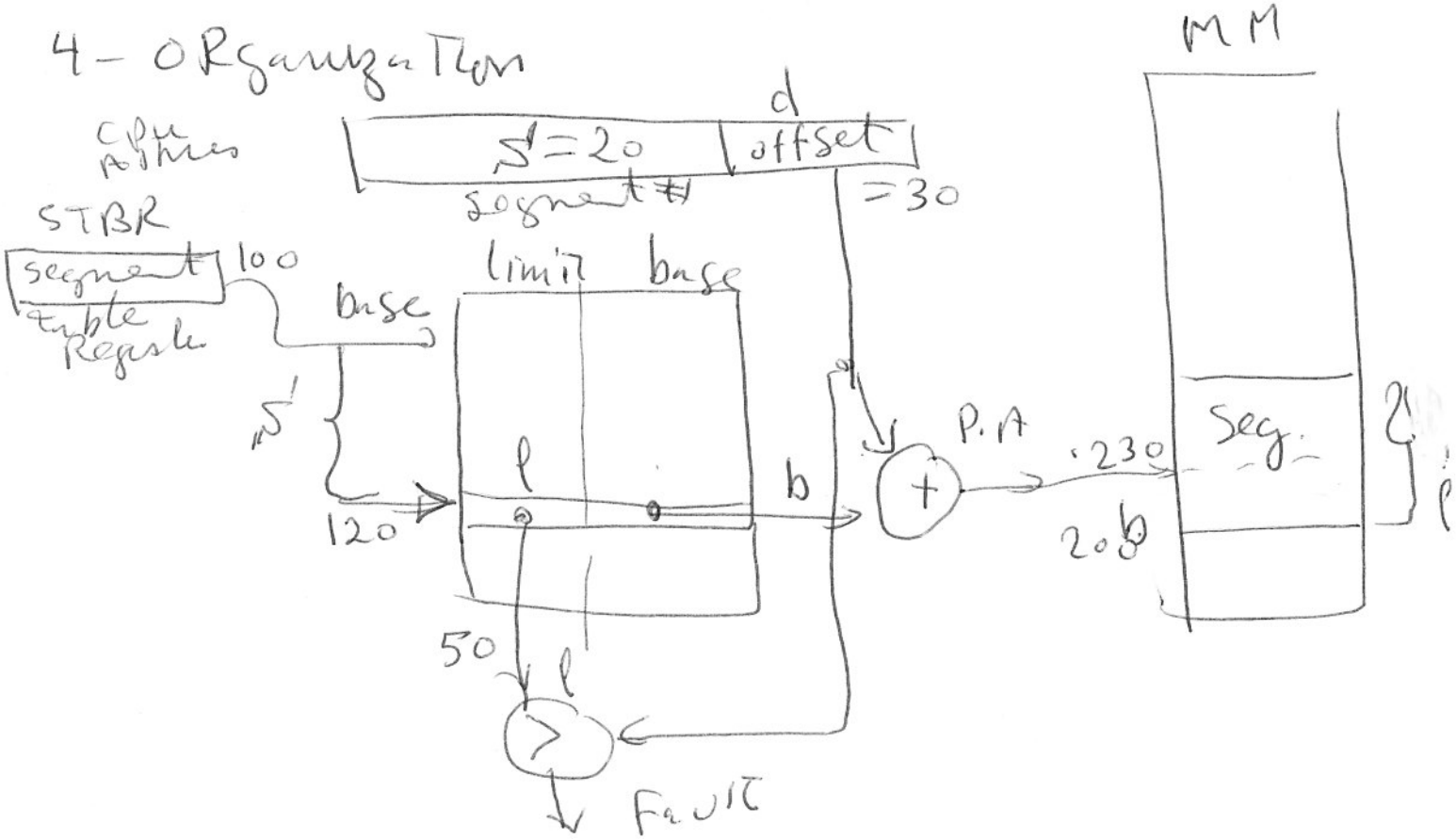
- External fragmentation: Large module size causes external fragmentation
- All segment is in or out (# pages)

3-Mapping

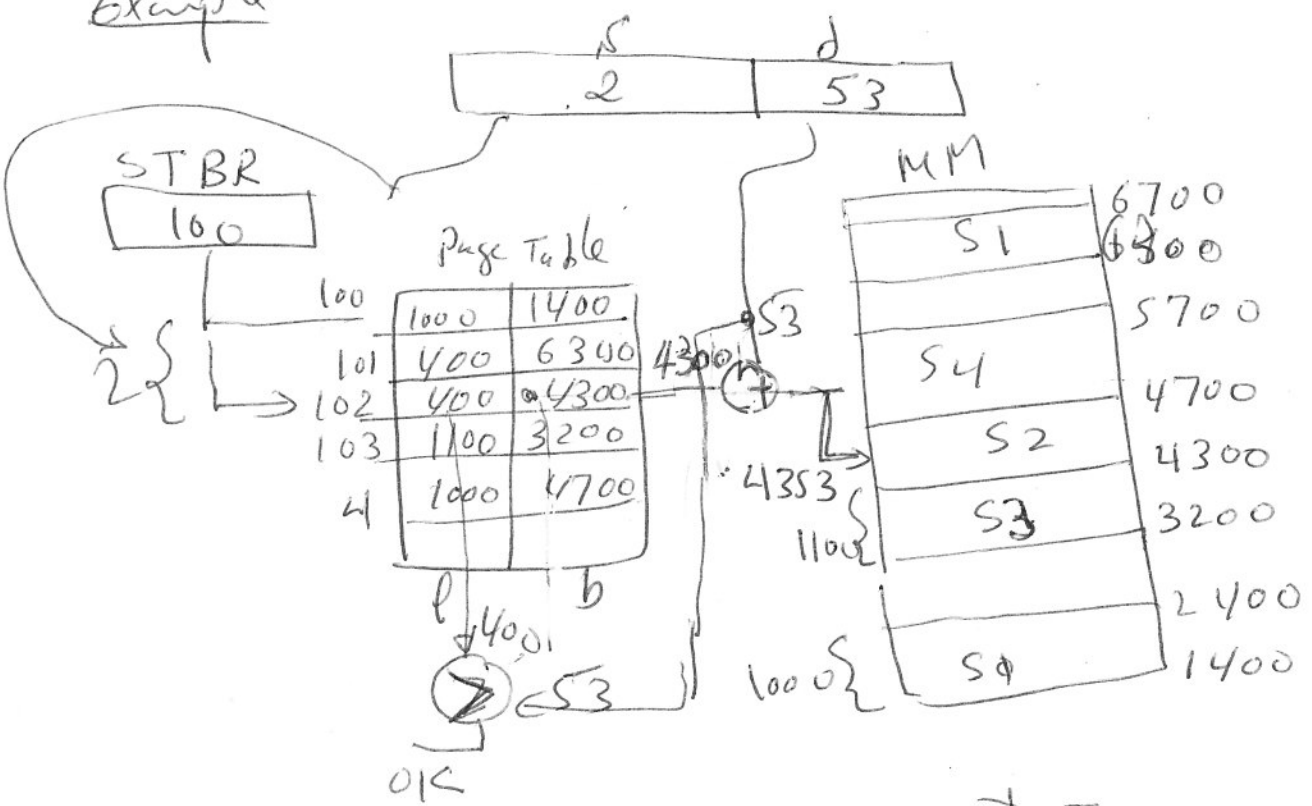
Any segment can map to any memory location that have enough space



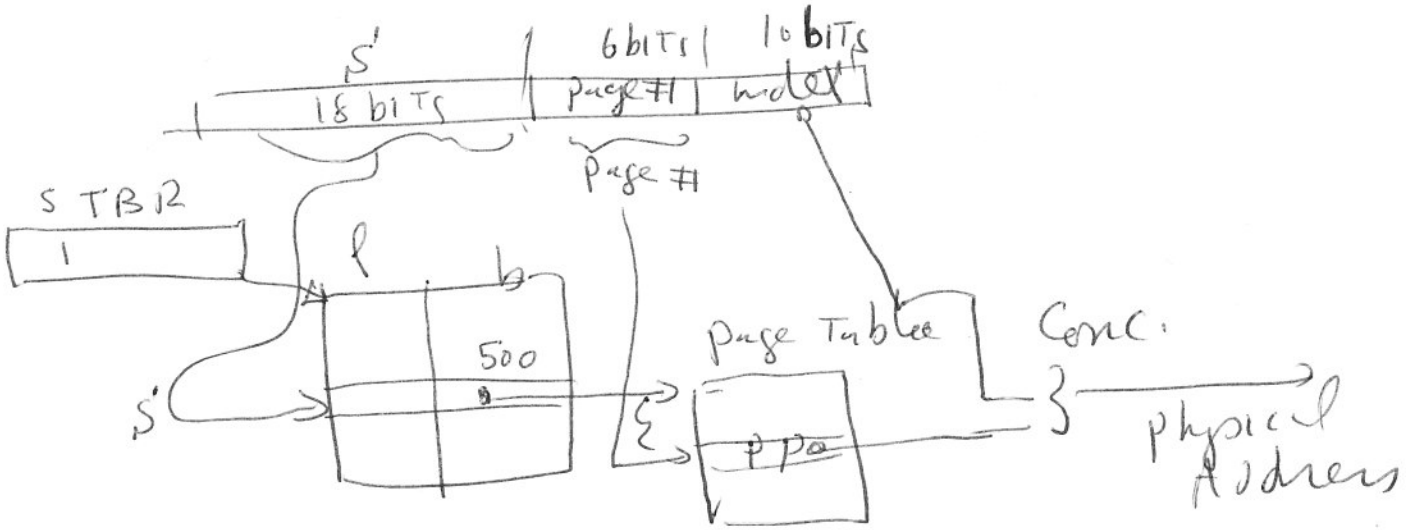
4- Organization



Example



Paged segmentation



7.11 Exercises

7.1 [10] <§7.2> Describe the general characteristics of a program that would exhibit very little temporal and spatial locality with regard to data accesses. Provide an example program (pseudocode is fine).

7.2 [10] <§7.2> Describe the general characteristics of a program that would exhibit very high amounts of temporal locality but very little spatial locality with regard to data accesses. Provide an example program (pseudocode is fine).

7.3 [10] <§7.2> Describe the general characteristics of a program that would exhibit very little temporal locality but very high amounts of spatial locality with regard to data accesses. Provide an example program (pseudocode is fine).

7.4 [10] <§7.2> Describe the general characteristics of a program that would exhibit very little temporal and spatial locality with regard to instruction fetches. Provide an example program (pseudocode is fine).

7.5 [10] <§7.2> Describe the general characteristics of a program that would exhibit very high amounts of temporal locality but very little spatial locality with regard to instruction fetches. Provide an example program (pseudocode is fine).

7.6 [10] <§7.2> Describe the general characteristics of a program that would exhibit very little temporal locality but very high amounts of spatial locality with regard to instruction fetches. Provide an example program (pseudocode is fine).

7.7 [10] <§7.2> Here is a series of address references given as word addresses: 1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17. Assuming a direct-mapped cache with 16 one-word blocks that is initially empty, label each reference in the list as a hit or a miss and show the final contents of the cache.

7.8 [10] <§7.2> Using the series of references given in Exercise 7.7, show the hits and misses and final cache contents for a direct-mapped cache with four-word blocks and a *total size* of 16 words.

7.9 [10] <§7.2> Compute the total number of bits required to implement the cache in Figure 7.10 on page 557. This number is different from the size of the cache, which usually refers to the number of bytes of data stored in the cache. The number of bits needed to implement the cache represents the total amount of memory needed for storing all of the data, tags, and valid bits.

7.10 [10] <§7.2> Find a method to eliminate the AND gate on the valid bit in Figure 7.7 on page 549. (Hint: You need to change the comparison.)

7.11 [10] <§7.2> Consider a memory hierarchy using one of the three organizations for main memory shown in Figure 7.13 on page 561. Assume that the cache block size is 16 words, that the width of organization b of the figure is four words, and that the number of banks in organization c is four. If the main memory latency for a new access is 10 cycles and the transfer time is 1 cycle, what are the miss penalties for each of these organizations?

7.12 [10] <§7.2> {Ex. 7.11} Suppose a processor with a 16-word block size has an effective miss rate per instruction of 0.5%. Assume that the CPI without cache misses is 1.2. Using the memories described in Figure 7.13 on page 561 and Exercise 7.11, how much faster is this processor when using the wide memory than when using narrow or interleaved memories?

7.13 [15] <§7.2> Cache C1 is direct-mapped with 16 one-word blocks. Cache C2 is direct-mapped with 4 four-word blocks. Assume that the miss penalty for C1 is 8 clock cycles and the miss penalty for C2 is 11 clock cycles. Assuming that the caches are initially empty, find a reference string for which C2 has a lower miss rate but spends more cycles on cache misses than C1. Use word addresses.

7.14 [15] <§7.2> For the caches in Exercise 7.13, find a series of references for which C2 has more misses than C1. Use word addresses.

In More Depth

Average Memory Access Time

To capture the fact that the time to access data for both hits and misses affects performance, designers often use average memory access time (AMAT) as a way to examine alternative cache designs. Average memory access time is the average time to access memory considering both hits and misses and the frequency of different accesses; it is equal to the following:

$$\text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}$$

AMAT is useful as a figure of merit for different cache systems.

7.15 [5] <§7.2> Find the AMAT for a machine with a 2-ns clock, a miss penalty of 20 clock cycles, a miss rate of 0.05 misses per instruction, and a cache access time (including hit detection) of 1 clock cycle. Assume that the read and write miss penalties are the same and ignore other write stalls.

7.16 [5] <§7.2> {Ex. 7.15} Suppose we can improve the miss rate to [illegible] misses per reference by doubling the cache size. This causes the cache access time to increase to 1.2 clock cycles. Using the AMAT as a metric, determine if this is a good trade-off.

7.17 [10] <§7.2> [Ex. 7.16} If the cache access time determines the processor's clock cycle time, which is often the case, AMAT may not correctly indicate whether one cache organization is better than another. If the machine's clock cycle time must be changed to match that of a cache, is this a good trade-off? Assume the machines are identical except for the cache rate and the number of cache miss cycles; assume 1.5 references per instruction and a CPI without cache misses of 2. The miss penalty is 20 cycles for both machines.

7.18 [10] <§§7.2, B.5> You have been given 18 32K × 8-bit SRAMs to build an instruction cache for a processor with a 32-bit address. What is the largest size (i.e., the largest size of the data storage area in bytes) direct-mapped instruction cache that you can build with one-word (32-bit) blocks? Show the breakdown of the address into its cache access components (for an example, see Figure 7.8) and describe how the various SRAM chips will be used. (Hint: You may not need all of them.)

7.19 [10] <§§7.2, B.5> This exercise is similar to Exercise 7.18, except that this time you decide to build a direct-mapped cache with four-word blocks as in Figure 7.10. Once again show the breakdown of the address and describe how the chips are used.

7.20 [10] <§7.3> Using the series of references given in Exercise 7.7, show the hits and misses and final cache contents for a two-way set-associative cache with one-word blocks and a *total size* of 16 words. Assume LRU replacement.

7.21 [10] <§7.3> Using the series of references given in Exercise 7.7, show the hits and misses and final cache contents for a fully associative cache with one-word blocks and a *total size* of 16 words. Assume LRU replacement.

7.22 [10] <§7.3> Using the series of references given in Exercise 7.7, show the hits and misses and final cache contents for a fully associative cache with four-word blocks and a *total size* of 16 words. Assume LRU replacement.

7.23 [5] <§7.3> Associativity usually improves the miss ratio, but not always. Give a short series of address references for which a two-way set-associative cache with LRU replacement would experience more misses than a direct-mapped cache of the same size.

7.24 [15] <§7.3> Suppose a computer's address size is k bits (using byte addressing), the cache size is S bytes, the block size is B bytes, and the cache is A-way set-associative. Assume that B is a power of two, so $B = 2^b$. Figure out what

7.30 [10] <§§7.3, B.5> This exercise is similar to Exercise 7.18, except that this time you decide to build a three-way set-associative cache with one-word blocks. Once again show the breakdown of the address (see Figure 7.19 for an example of a four-way set-associative cache) and describe how the chips are used. Note that each SRAM will only perform a single read per cache access.

7.31 [5] <§§7.2–7.4> Rank each of the possible event combinations appearing in the example on page 595 according to how frequently you think they would occur.

7.32 [15] <§7.4> Consider a virtual memory system with the following properties:

- 40-bit virtual byte address
- 16-KB pages
- 36-bit physical byte address

What is the total size of the page table for each process on this machine, assuming that the valid, protection, dirty, and use bits take a total of 4 bits and that all the virtual pages are in use? (Assume that disk addresses are not stored in the page table.)

7.33 [15] <§7.4> Assume that the virtual memory system of Exercise 7.32 is implemented with a two-way set-associative TLB with a total of 256 TLB entries. Show the virtual-to-physical mapping with a figure like Figure 7.25 on page 593. Make sure to label the width of all fields and signals.

7.34 [15] <§7.3> Assume that the cache for the system described in Exercise 7.32 is two-way set associative and has eight-word blocks and a total size of 16 KB. Show the cache organization and access using the same format as Figure 7.19 on page 574.

7.35 [15] <§7.4> Page tables require fairly large amounts of memory (as described in the elaboration on page 587), even if most of the entries are invalid. One solution is to use a hierarchy of page tables. The virtual page number, as described in Figure 7.21 on page 582, can be broken up into two pieces, a "page table number" and a "page table offset." The page table number can be used to index a first-level page table that provides a physical address for a second-level page table, assuming it resides in memory (if not, a first-level page fault will occur and the page table itself will need to be brought in from disk). The page table offset is used to index into the second-level page table to retrieve the physical page number. One obvious way to arrange such a scheme is to have the second-level page tables occupy exactly one page of memory. Assuming a 32-bit virtual address space with 4-KB pages and 4 bytes per page table entry, how many bytes will each program need to use to store the first

7-7 set of references:

1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Given as word addresses.

Assume direct mapped cache with 16 ~~word~~ blocks (each 1 word). Find hit, miss

Mapping = Address modulo # of blocks

Address	Map	hit/miss
1	1	miss
4	4	miss
8	8	miss
5	5	miss
20	4	miss
17	1	miss
19	3	miss
56	8	miss
9	9	miss
11	11	miss
4	4	miss (20)
43	11	miss (11)
5	5	hit
6	6	miss
9	9	hit
17	1	hit

3 hit / 16

7-8 Same references if block size = 4 words and cache size = 16 words.

blocks = 4

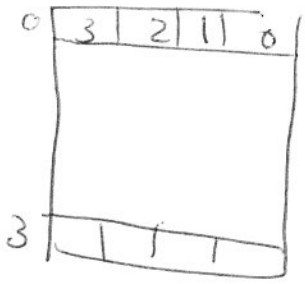
mapping

memory block number = Address / 4

mapped memory block # modulo 4

Address	Memory block #	Cache block	hit/miss
1	0	0	miss
4	1	1	miss
8	2	2	miss
5	1	1	hit –
20	5	1	miss
17	4	0	miss (0)
19	4	0	hit (4) –
56	14	2	miss (2)
9	2	2	miss (14)
11	2	2	hit (2) –
4	1	1	miss (20)
43	10	2	miss (2)
5	1	1	hit –
6	1	1	hit –
9	2	2	miss (10)
17	4	0	hit (4) –

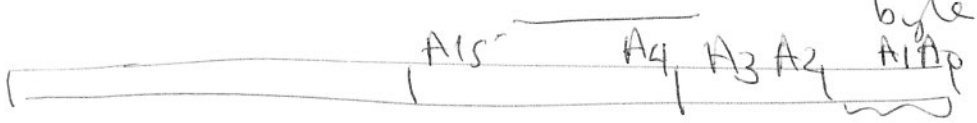
block #



6 hits

7-9 Find cache size (data, Tag, valid bit)

has 4K entries, block = four words = 16 byte.



byte select = $\log_2$ 4 byte = 2 bits

Word select in block = $\log_2$ 4 = 2 bits

Index = $\log_2$ 4K = 12 bits

Tag = 32 − 12 − 2 − 2 = 16 bits + 1 bit valid

= 17 bits × 4K = 2 byte × 4K + 4K bit

Data = 16B × 4K = 64KB

Total = 72 KB + 4K bit valid

7-15

2ns clock cycle, miss penalty of 20 cycles

miss rate = .05

find average time / instruction.

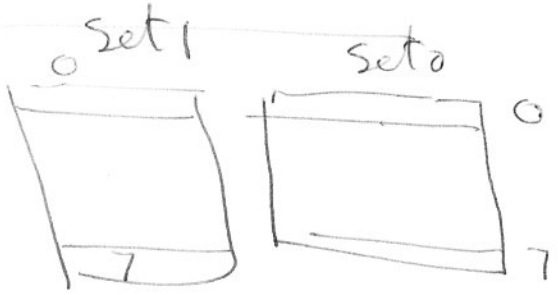
T = hit time + miss rate x miss penalty

= 2ns + .05 x 20 x 2ns

= 2 + 2 = 4ns

7-20 use same sequence, assume two way set associative, LRU, size = 16 word, block = 1 word

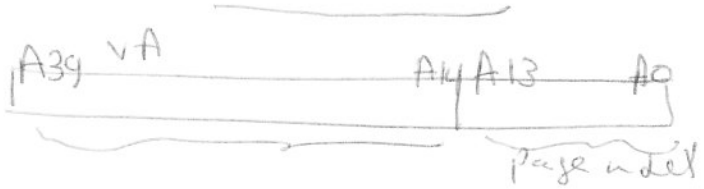
Address	cache set 0	hit/miss	cache set 1	hit/miss	
1	1	miss			
4	4	miss			
8	0	miss			
5	5	miss			
20			4	miss	
17			1	miss	
19	3	miss			
56			0	miss	
9	1	miss			→ LRU
11			3	miss	
4	4	hit			
43	3	miss			→ LRU
5	5	hit			
6	6	miss			
9	1	hit			
17			1	hit	



4 hits

7-32 Assume VM with:
- 40 bit VA
- 16KB pages
= 36 bit physical Address

Find size of page Table using 4 bits for valid & dirty & Protection



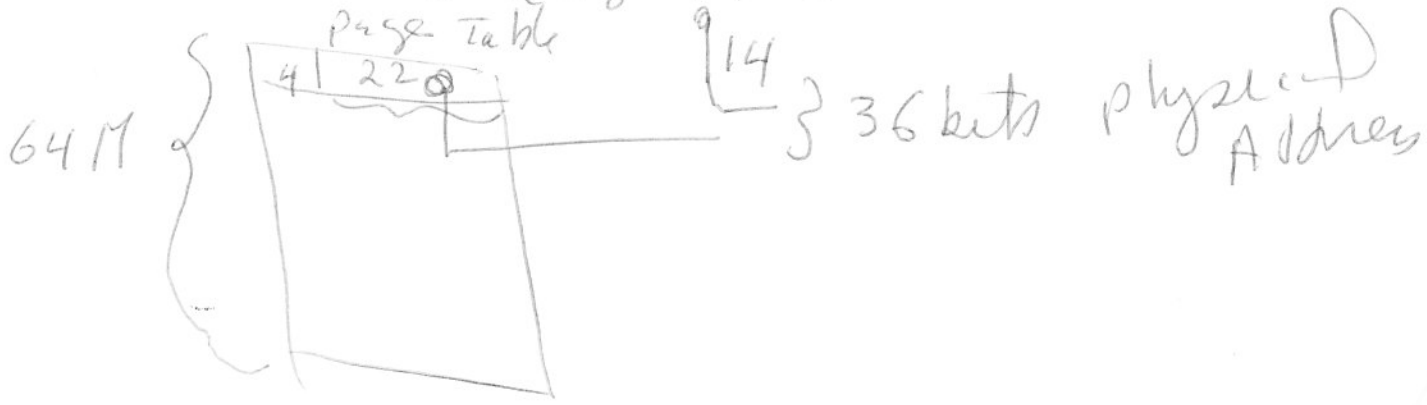
Page index = $\log_2$ 16KB = 14 bits

- size of VPage number in bits = 40 - 14 = 26 bits
- number of enteries in page Table
$= 2^{26}$ = 64 Million
- Each entry has = a physical page number
+4 bits

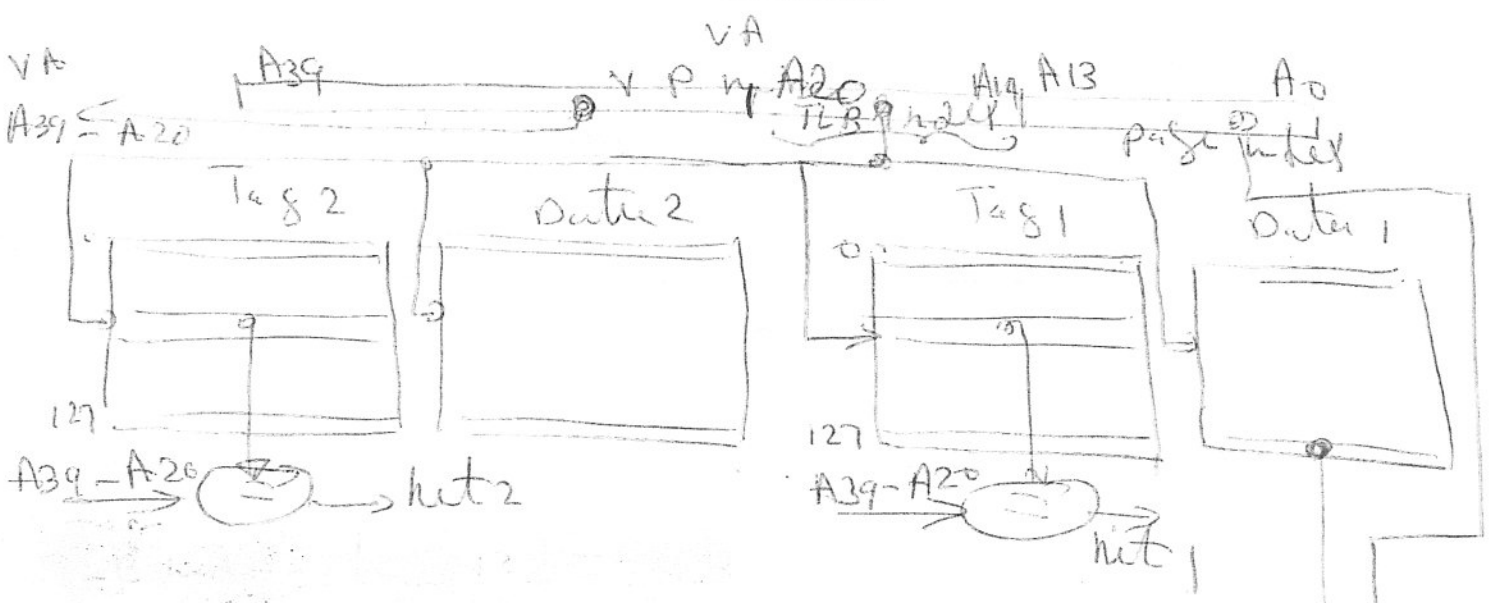
physical page number = 36 - 14 = 22 bits

Each entry has 22 + 4 = 26 bits

Total size of page Table = 64M X 26 bits
= 208 MB

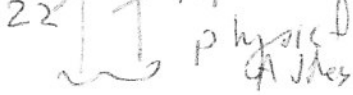


7-33. Design TLB with 256 entry & two way set associative for 40 bit VA, 36 bit physical Address, Page = 16KB

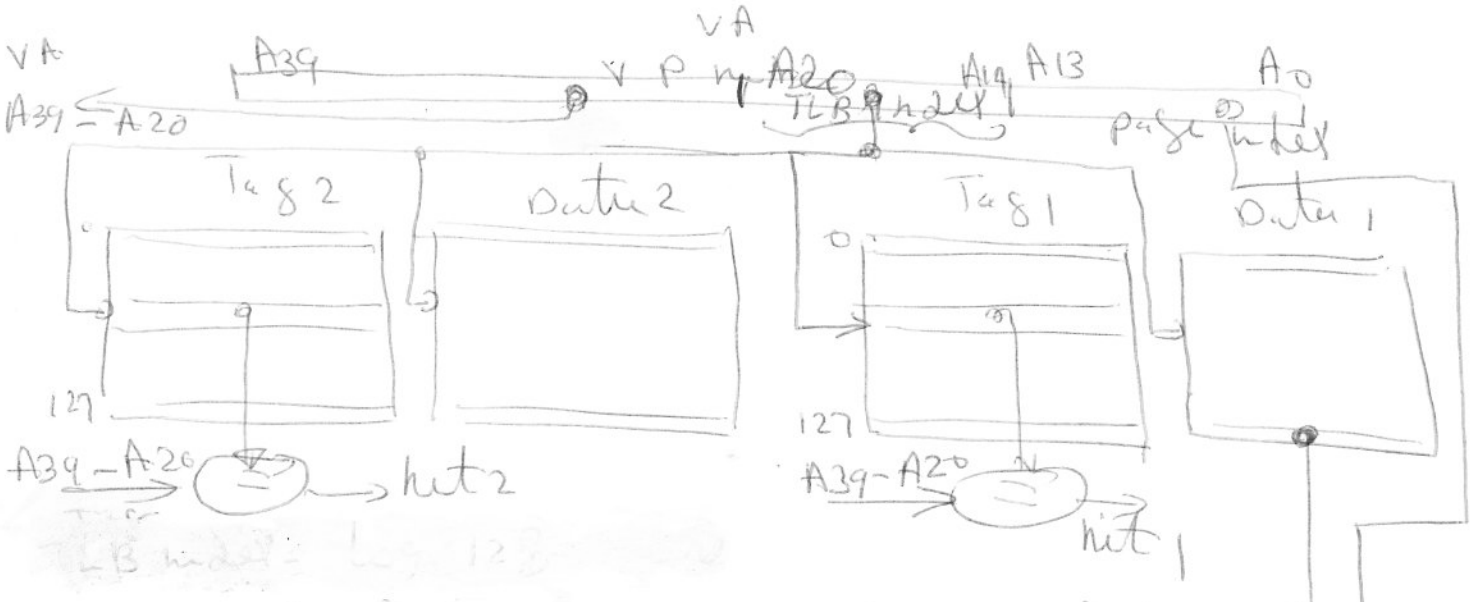


TLB index = $\log_2 128$ = 7 bits To index TLB

number of bits in TLB Data = physical Address bits
= 22 bits for physical page number



7-33. Design TLB with 256 entry & two way set associative for 40 bit VA, 36 bit physical Address, Page = 16KB



TLB index = $\log_2 128 = 7$ bits To index TLB

number of bits in TLB Data = physical Address bits
= 22 bits for physical page number