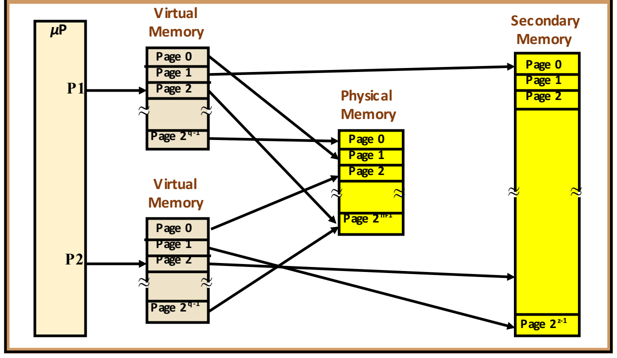
Virtual Memory Organization



On this Lesson

- Organization of virtual memory systems
- Paging – virtual to physical address translation
- Organization of page tables
- Page faults
- Translation Lookaside Buffer
- Multi-level page tables
- Segmentation
- Caches on virtual memory systems

Virtual Memory

- Allows the operating system to administer the transfer of data between main memory and secondary memory.
- Provides a mechanism for the operating system to protect data between users.
- Allows the execution of programs that are larger than the installed physical memory
- Facilitates multiprocessing

Memory Terms

Virtual Memory (Primary Memory)

- *Is the architectural memory, the memory space that can be accessed by fetch/load/store operations of a machine code program.*
- *It has **2^n** memory locations, where **n** is the number of bits needed to specify a memory address.*

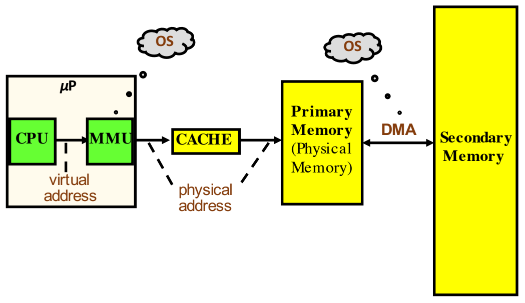
Physical Memory

- *Is the RAM memory physically installed in a computer.*
- *It has **2^m** memory locations, where **m** could be less than **n**.*

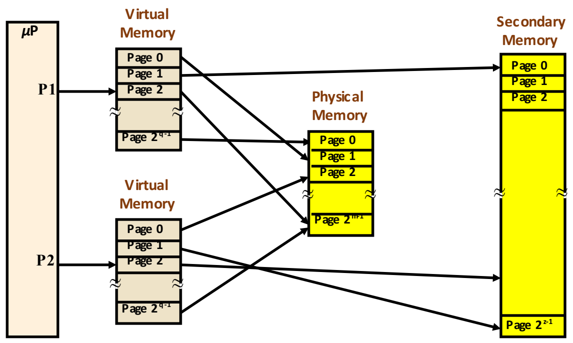
Secondary Memory

- *Is the memory where programs are permanently stored.*
- *It has **2^z** memory locations, where **z** is significantly larger than **n** or **m**.*
- *It is non-volatile*

Memory Spaces



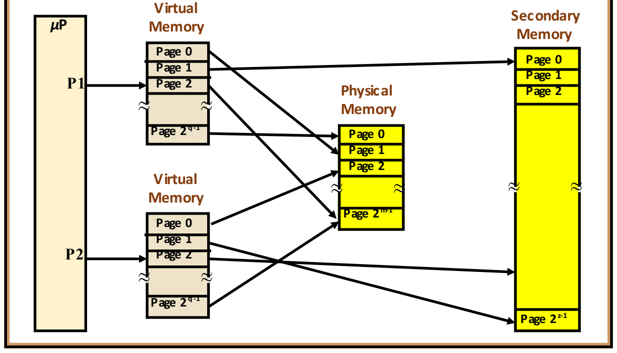
Virtual and Physical Spaces



Paging

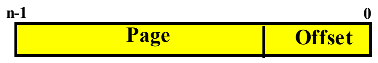
- Is the mechanism that facilitates the mapping of virtual addresses to physical addresses and the transfer of pages between physical memory and secondary memory.
- Mapping of virtual addresses to physical addresses is accomplished by mean of a page table that is **resident in primary memory**.
- Each fetch/load/store access involves two memory accesses, one to the page table and one to access the physical address of the fetch/load/store operation.
- Transfer of pages between secondary memory and physical memory is made on demand (when the pages are needed by a program).

Virtual Memory Address Translation



Virtual Address

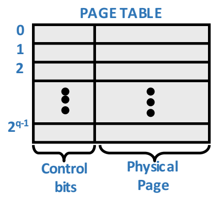
- A virtual address is broken into two fields:



  - *Page refers to a specific block of bytes in memory (virtual or physical).*
  - *Offset specifies a byte within a page.*
- The number of bits of the offset and the page depends on the size of the pages.
- For a 2^k-byte page there are 2^{n-k} (2^q) pages. Thus the Offset field is **k** bits and the Page field is **n-k** bits.
- For a 32-bit address and 4K-byte page there are
  - *2^{20} (2^{32-12}) pages and 2^{12} bytes per page*
  - *12 bits for the Offset field and 20 bits for the Page field*

Page Table

- An array with an entry for each page of the virtual space



- Each entry has a series of bits known as control bits and the physical page number corresponding to the virtual page if it resides in physical memory.
- The page table is constructed and managed by the operating system.

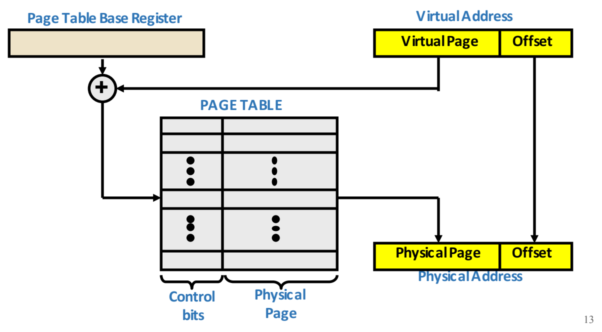
Control Bits

Valid	Indicates if the virtual page is in physical memory.
Dirty	Indicates if the corresponding physical page has been written while in physical memory.
Read	Indicates if the page can be read by the program that is trying to access it.
Write	Indicates if the page can be written by the program that is trying to access it.
Execute	Indicates if the code store in the page can be executed by the program that is trying to access it.

Page Table Base Register

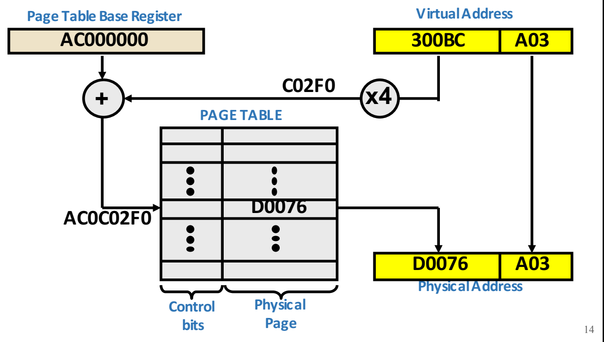
- Each program (process) assumes that it has the whole virtual memory space (2^{32} bytes in case of the 32-bit ARM architecture)
- Each process has its own page table in primary memory
- The location where the first entry of the page table is located in memory is specified by a **Page Table Base Register (PTBR)**
- The **Page Table Base Register** is managed under a privileged operation mode by the operating system (writing to this register is a privileged operation)

Virtual to Physical Address Translation



13

Example: 2^{32}-byte virtual memory, 4K-byte page, 4-byte page entry



14

Another Example: 2^{16}-byte virtual memory, 4K-byte page, 2-byte page entry, 8-page physical memory

In which physical address is the page table entry for virtual page 4 located ?

If PTBR = 1010100000011100:

= 4x2 + 1010100000011100
= 1000 + 1010100000011100
= 1010100000100100

If PTBR = 0000000000000000:

= 4x2 + 0000000000000000
= 1000 + 0000000000000000
= 0000000000001000

Page Table

	V	D	R	W	E	Physical Page
0	1	0	1	1	1	0110
1	1	1	0	1	0	0111
2	0	0	0	1	1	0001
3	1	1	0	1	0	0011
4	0	0	0	1	0	0001
5	0	1	1	1	1	0001
6	0	0	1	1	0	0001
7	1	1	1	1	1	0101
8	0	0	1	0	0	0001
9	1	1	1	0	0	0010
10	0	1	0	0	1	0001
11	0	1	1	0	0	0001
12	0	1	0	1	1	0001
13	1	0	1	0	0	0000
14	0	1	1	0	0	0001
15	0	0	0	0	0	0001

V - Valid
D - Dirty
R - Read
W - Write
E - Execute

Another Example: 2^{16}-byte virtual memory, 4K-byte page, 2-byte page entry, 8-page physical memory

For which virtual pages an access will result in a page fault?

Answer:
2, 4, 5, 6, 8, 10, 11, 12, 14, 15

Page Table

	V	D	R	W	E	Physical Page
0	1	0	1	1	1	0110
1	1	1	0	1	0	0111
2	0	0	0	1	1	0001
3	1	1	0	1	0	0011
4	0	0	0	1	0	0001
5	0	1	1	1	1	0001
6	0	0	1	1	0	0001
7	1	1	1	1	1	0101
8	0	0	1	0	0	0001
9	1	1	1	0	0	0010
10	0	1	0	0	1	0001
11	0	1	1	0	0	0001
12	0	1	0	1	1	0001
13	1	0	1	0	0	0000
14	0	1	1	0	0	0001
15	0	0	0	0	0	0001

V - Valid
D - Dirty
R - Read
W - Write
E - Execute

Another Example: 2^{16}-byte virtual memory, 4K-byte page, 2-byte page entry, 8-page physical memory

From which physical pages is the process allowed to read?

Answer:
6, 5, 2, 0

Page Table

	V	D	R	W	E	Physical Page
0	1	0	1	1	1	0110
1	1	1	0	1	0	0111
2	0	0	0	1	1	0001
3	1	1	0	1	0	0011
4	0	0	0	1	0	0001
5	0	1	1	1	1	0001
6	0	0	1	1	0	0001
7	1	1	1	1	1	0101
8	0	0	1	0	0	0001
9	1	1	1	0	0	0010
10	0	1	0	0	1	0001
11	0	1	1	0	0	0001
12	0	1	0	1	1	0001
13	1	0	1	0	0	0000
14	0	1	1	0	0	0001
15	0	0	0	0	0	0001

V - Valid
D - Dirty
R - Read
W - Write
E - Execute

Another Example: 2^{16}-byte virtual memory, 4K-byte page, 2-byte page entry, 8-page physical memory

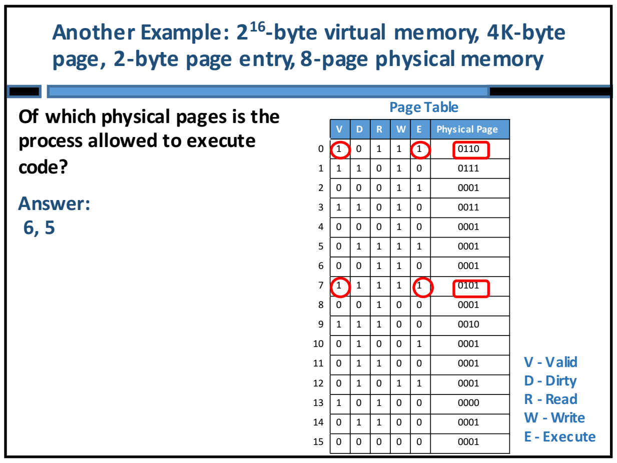
To which physical pages is the process allowed to write?

Answer:
6, 7, 3, 5

Page Table

	V	D	R	W	E	Physical Page
0	1	0	1	1	1	0110
1	1	1	0	1	0	0111
2	0	0	0	1	1	0001
3	1	1	0	1	0	0011
4	0	0	0	1	0	0001
5	0	1	1	1	1	0001
6	0	0	1	1	0	0001
7	1	1	1	1	1	0101
8	0	0	1	0	0	0001
9	1	1	1	0	0	0010
10	0	1	0	0	1	0001
11	0	1	1	0	0	0001
12	0	1	0	1	1	0001
13	1	0	1	0	0	0000
14	0	1	1	0	0	0001
15	0	0	0	0	0	0001

V - valid
D - Dirty
R - Read
W - Write
E - Execute



Another Example: 2^{16}-byte virtual memory, 4K-byte page, 2-byte page entry, 8-page physical memory

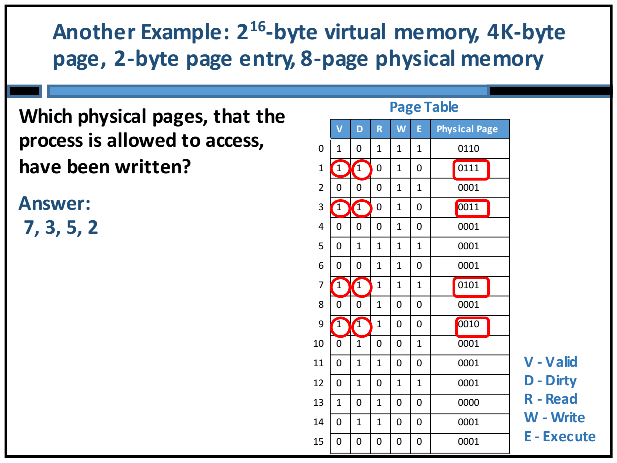
Of which physical pages is the process allowed to execute code?

Answer:
6, 5

Page Table

	V	D	R	W	E	Physical Page
0	1	0	1	1	1	0110
1	1	1	0	1	0	0111
2	0	0	0	1	1	0001
3	1	1	0	1	0	0011
4	0	0	0	1	0	0001
5	0	1	1	1	1	0001
6	0	0	1	1	0	0001
7	1	1	1	1	1	0101
8	0	0	1	0	0	0001
9	1	1	1	0	0	0010
10	0	1	0	0	1	0001
11	0	1	1	0	0	0001
12	0	1	0	1	1	0001
13	1	0	1	0	0	0000
14	0	1	1	0	0	0001
15	0	0	0	0	0	0001

V - Valid
D - Dirty
R - Read
W - Write
E - Execute



Another Example: 2^{16}-byte virtual memory, 4K-byte page, 2-byte page entry, 8-page physical memory

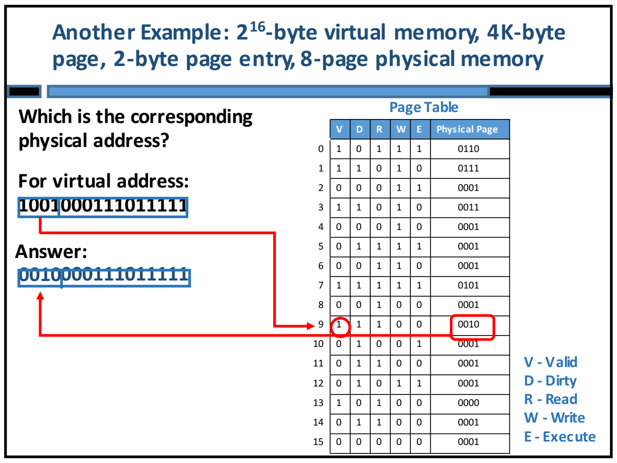
Which physical pages, that the process is allowed to access, have been written?

Answer:
7, 3, 5, 2

Page Table

	V	D	R	W	E	Physical Page
0	1	0	1	1	1	0110
1	1	1	0	1	0	0111
2	0	0	0	1	1	0001
3	1	1	0	1	0	0011
4	0	0	0	1	0	0001
5	0	1	1	1	1	0001
6	0	0	1	1	0	0001
7	1	1	1	1	1	0101
8	0	0	1	0	0	0001
9	1	1	1	0	0	0010
10	0	1	0	0	1	0001
11	0	1	1	0	0	0001
12	0	1	0	1	1	0001
13	1	0	1	0	0	0000
14	0	1	1	0	0	0001
15	0	0	0	0	0	0001

V - Valid
D - Dirty
R - Read
W - Write
E - Execute



Another Example: 2^{16}-byte virtual memory, 4K-byte page, 2-byte page entry, 8-page physical memory

Which is the corresponding physical address?

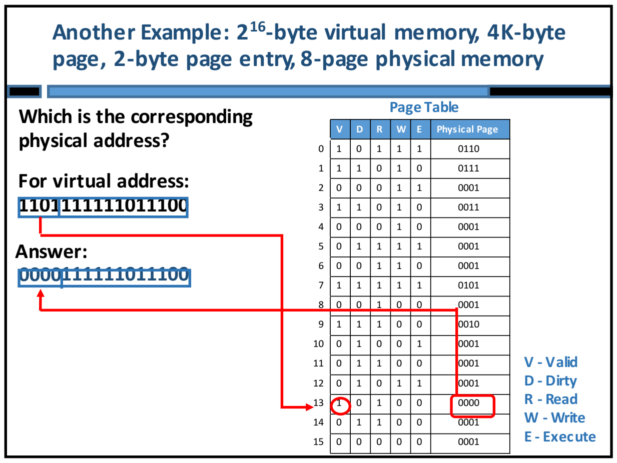
For virtual address:
1001000111011111

Answer:
0010000111011111

Page Table

	V	D	R	W	E	Physical Page
0	1	0	1	1	1	0110
1	1	1	0	1	0	0111
2	0	0	0	1	1	0001
3	1	1	0	1	0	0011
4	0	0	0	1	0	0001
5	0	1	1	1	1	0001
6	0	0	1	1	0	0001
7	1	1	1	1	1	0101
8	0	0	1	0	0	0001
9	1	1	1	0	0	0010
10	0	1	0	0	1	0001
11	0	1	1	0	0	0001
12	0	1	0	1	1	0001
13	1	0	1	0	0	0000
14	0	1	1	0	0	0001
15	0	0	0	0	0	0001

V - Valid
D - Dirty
R - Read
W - Write
E - Execute



Another Example: 2^{16}-byte virtual memory, 4K-byte page, 2-byte page entry, 8-page physical memory

Which is the corresponding physical address?

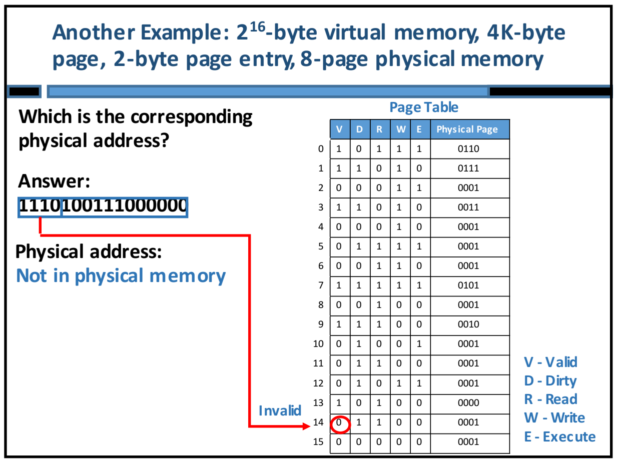
For virtual address:
1101111111011100

Answer:
0000111111011100

Page Table

	V	D	R	W	E	Physical Page
0	1	0	1	1	1	0110
1	1	1	0	1	0	0111
2	0	0	0	1	1	0001
3	1	1	0	1	0	0011
4	0	0	0	1	0	0001
5	0	1	1	1	1	0001
6	0	0	1	1	0	0001
7	1	1	1	1	1	0101
8	0	0	1	0	0	0001
9	1	1	1	0	0	0010
10	0	1	0	0	1	0001
11	0	1	1	0	0	0001
12	0	1	0	1	1	0001
13	1	0	1	0	0	0000
14	0	1	1	0	0	0001
15	0	0	0	0	0	0001

V - Valid
D - Dirty
R - Read
W - Write
E - Execute



Another Example: 2^{16}-byte virtual memory, 4K-byte page, 2-byte page entry, 8-page physical memory

Which is the corresponding physical address?

Answer:
1110100111000000

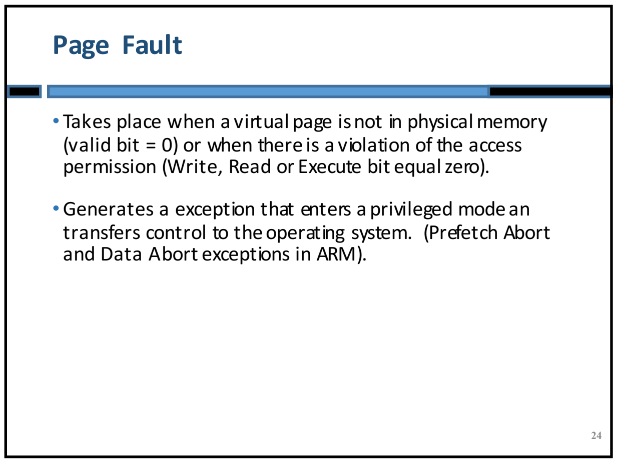
Physical address:
Not in physical memory

Invalid

Page Table

	V	D	R	W	E	Physical Page
0	1	0	1	1	1	0110
1	1	1	0	1	0	0111
2	0	0	0	1	1	0001
3	1	1	0	1	0	0011
4	0	0	0	1	0	0001
5	0	1	1	1	1	0001
6	0	0	1	1	0	0001
7	1	1	1	1	1	0101
8	0	0	1	0	0	0001
9	1	1	1	0	0	0010
10	0	1	0	0	1	0001
11	0	1	1	0	0	0001
12	0	1	0	1	1	0001
13	1	0	1	0	0	0000
14	0	1	1	0	0	0001
15	0	0	0	0	0	0001

V - Valid
D - Dirty
R - Read
W - Write
E - Execute



Page Fault

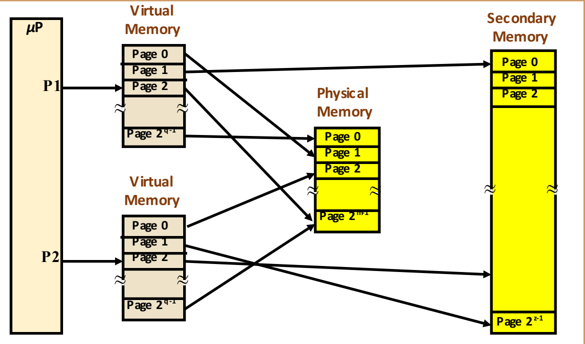
- Takes place when a virtual page is not in physical memory (valid bit = 0) or when there is a violation of the access permission (Write, Read or Execute bit equal zero).
- Generates a exception that enters a privileged mode an transfers control to the operating system. (Prefetch Abort and Data Abort exceptions in ARM).

24

Operating System Intervention on a Page Fault

1. Context Switch - Stops the faulting process and let another process to run (changes the content of the Page Table Base Register).
2. Uses an algorithm to determine a victim in physical memory to place the faulting page.
3. If the victim has been written (dirty bit = 1), instructs an I/O port to initiate the transfer of the victim page to secondary memory.
4. After the victim is transferred, instructs and I/O port to initiate the transfer of the faulting page to physical memory into the space previously occupied by the victim page.
5. After the page is transferred, places the faulting process back into the execution queue.

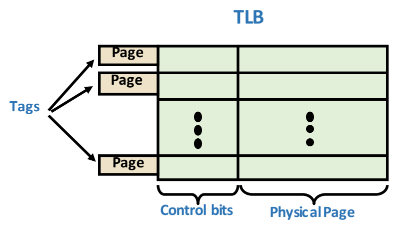
Virtual Memory
Reducing Translation Time



Reducing Memory Translation Time

- A CPU access to memory involves two memory access on a virtual memory system, one to the page table and one to the desired memory location in physical memory.
- On the best scenario (a hit on a cache) each access may take one cycle.
- The access time could be reduced by incorporating a mechanism that can construct the physical address without having to access the page table in physical memory.
- A small fully associative cache of virtual to physical page translations could do the job.
- Such cache is known as a Translation Lookaside Buffer (TLB).

Translation Lookaside Buffer (TLB)

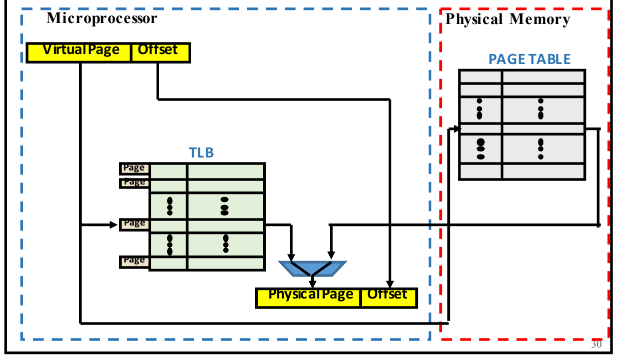


- Tags are virtual page numbers
- Resides in the microprocessor

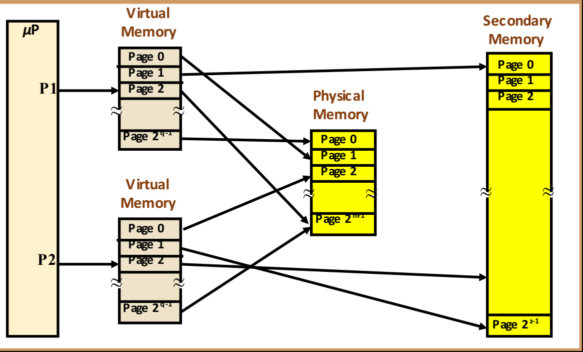
Translation Misses

- When an address translation results in a miss, the translation is made using the page table in physical memory.
- The resulting translation is placed on the TLB using a cache replacement algorithm.
- If the virtual page is not in physical memory, then a page fault is generated.

Address Translation with TLB



Virtual Memory
Multi-level Page Tables



Memory Space Demand by Page Tables

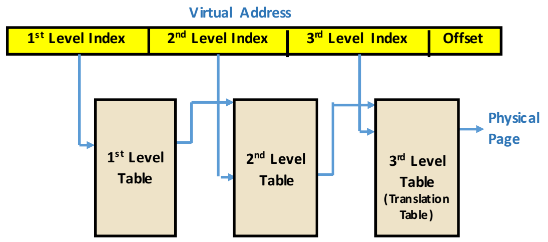
- Each process running on a CPU with virtual memory requires a page table.
- For a system with a virtual space of 2^{32} bytes and pages of 2^{10} bytes there are 2^{22} pages.
- If each page table entry is 4 bytes then, the space required for the page table is $2^{22} \times 2^{2} = 2^{24}$ bytes (16 Mbytes)
- If 50 processes run concurrently on the CPU, a memory space of 50 x 16 Mbytes is required, this is 800 Mbytes, which is close to one forth the total virtual space.

Alternatives for Reducing the Memory Space Required by Page Tables

- Multi-level page tables
- Segmentation

Multi-level Page Tables

- Various levels of address indirection are used to get to the page table that will provide the translation:
- The page field of the virtual address is broken into sub fields corresponding to the different page table levels and an offset.

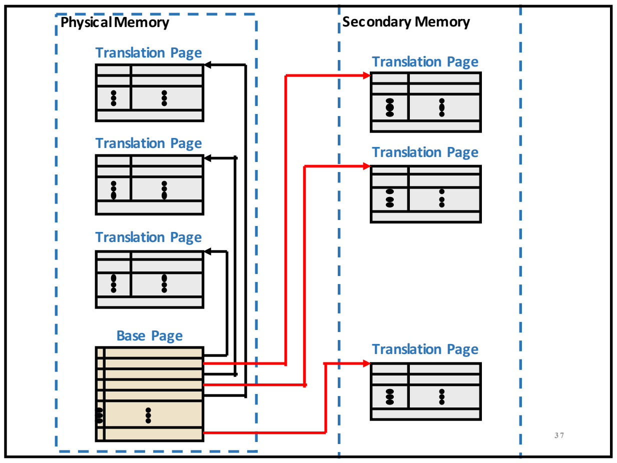


Two-level Page Tables

- Base Table – first level table of base addresses of second level tables (the beginning addresses of the a translation table).
- Translation Table - second level table that translates the virtual page to a physical page. Has control bits and physical page fields like a single-level page table.
- The Base Table of a process must reside in physical memory.
- At least one Translation Table must be in physical memory.
- Translation tables are brought to physical memory from secondary memory on demand (like on single-level page tables).

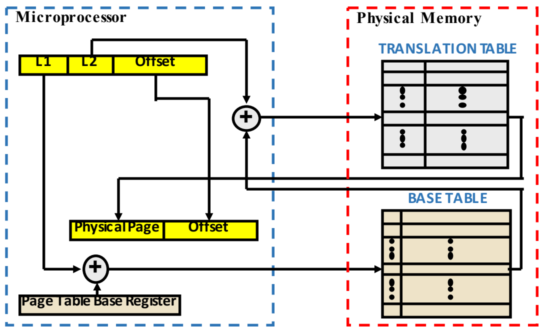
Page Faults on Two-level Page Tables

- Page faults may occur on accesses to either a Base Table or a Translation Table.
- A page fault on a Base Table requires a transfer of a Translation Table to physical memory.
- A page fault on a Translation Table requires the transfer of a page to main memory.



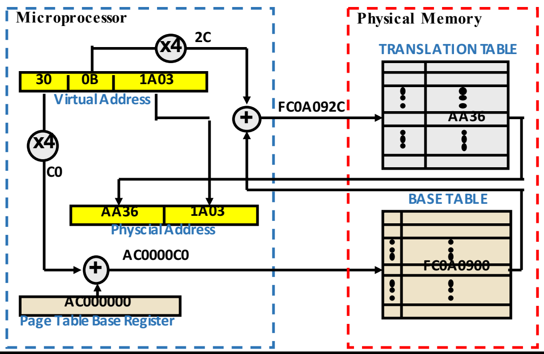
37

Two-Level Table Address Translation



38

Example: Two-level page tables, 2^{32}-byte virtual memory, 64K-byte page, 4-byte page entry



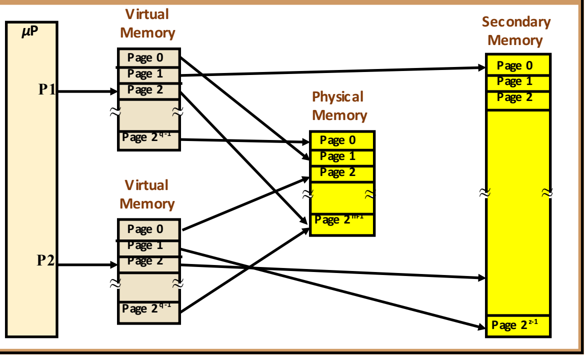
39

Memory Space Demand by Multi-level Page Tables

- Each process running on a CPU with virtual memory requires a Base Table and at least one Translation Table.
- For a system with a virtual space of 2^{32} bytes and pages of 2^{10} bytes there are 2^{22} pages.
- If the first and second level fields of the virtual address have 11 bits, and each page table entry is 4 bytes then, the space required for each page table is $2^{11} \times 2^{2} = 2^{13}$ bytes (8 Kbytes)
- If 50 processes run concurrently on the CPU, a minimum memory space of 800 Kbytes (50 x 2 x 8 Kbytes) is required. (1/1000 the size required for a single level table).

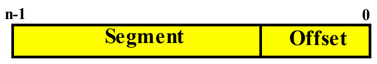
40

Virtual Memory Segmentation



Segmentation

- Programs are broken into segments of variable length.
- The virtual address is broken into **Segment** and **Offset** fields:

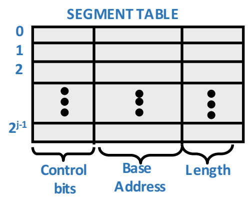


- A segment table is used to map virtual addresses to physical addresses.
  - *resides in physical memory.*
  - *constructed and managed by the operating system.*
- Each process has its own segment table

42

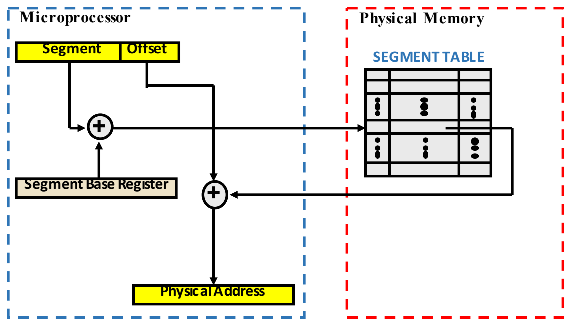
Segment Table

- An array with an entry for each segment of the virtual space:



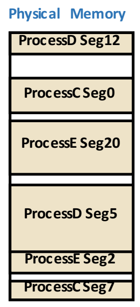
- Page fault occurs under any of the following circunstances:
  - *segment is not valid*
  - *violation of access permission*
  - *offset larger than the segment length*

Address Translation with Segment Table



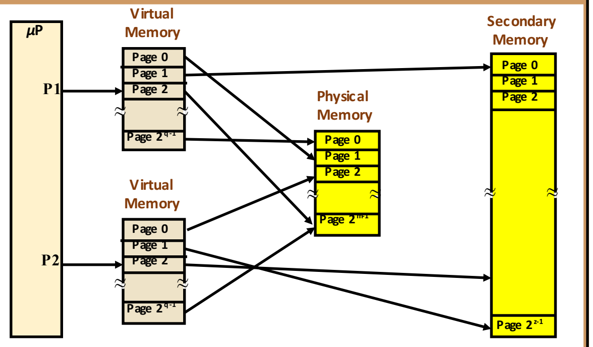
Fragmentation

- As segments are allocated and replaced into physical memory, blocks of free memory locations of varied sizes are created.
- New blocks that do not fit into a free block end up replacing an existing block.
- New blocks that are allocated in a free block that is larger end up increasing fragmentation.
- On physical memory, fragmentation causes an inefficient use the memory space.
- On a secondary memory like a hard disk, fragmentation increases access time due to an increase in travel time of the disc reader mechanism as it moves from track to track.



Virtual Memory

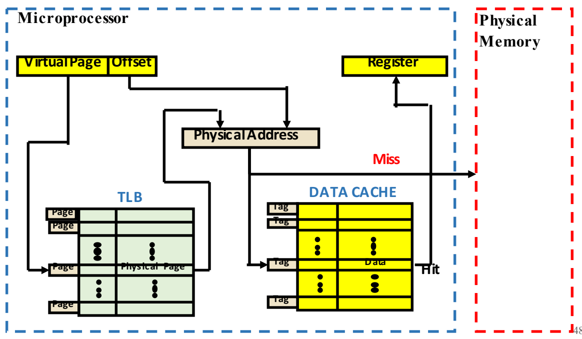
Caches on Virtual Memory Systems



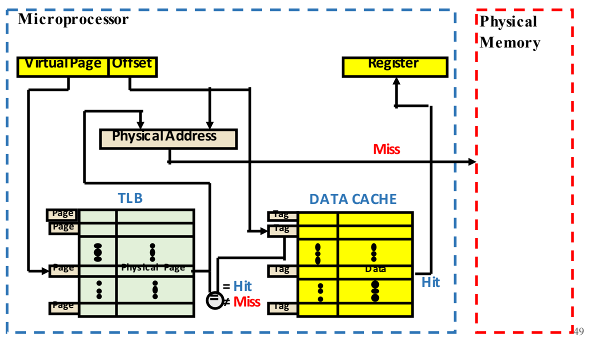
Caches on Virtual Memory Systems

- Physical Cache
  - *Use physical addresses*
  - *Total memory access can be a minimum of two cycles if a TLB produces the physical address*
- Virtually Indexed, Physically Tagged (VIPT) Cache
  - *Uses offset of virtual address as index*
  - *Uses physical address tags (physical pages)*
  - *Needs TLB to validate physical page*
  - *Total memory access can be a minimum of one cycle*

Physical Caches on Virtual Memory Systems



Virtually Indexed, Physically Tagged (VIPT) Caches on Virtual Memory Systems



Lesson Outcomes

- Understand the organization of virtual memory systems.
- Understand the process of virtual to physical address translation.
- Know how the operating system intervenes in a virtual memory system, in particular on a page fault.
- Know how multi-level page tables accomplish address translation.
- Know how address translation is accomplished on virtual memory systems that use segmentation.
- Understand how caches are integrated on a virtual memory system.