CS 241 Section Week #12
(04/22/10)

Outline

- Virtual Memory
  - Why Virtual Memory
  - Virtual Memory Addressing
  - TLB (Translation Lookaside Buffer)
  - Multilevel Page Table
- Problems

Virtual Memory

Why Virtual Memory?

- Use main memory as a Cache for the Disk
  - Address space of a process can exceed physical memory size
  - Sum of address spaces of multiple processes can exceed physical memory

Why Virtual Memory?

- Use main memory as a Cache for the Disk
  - Address space of a process can exceed physical memory size
  - Sum of address spaces of multiple processes can exceed physical memory
- Simplify Memory Management
  - Multiple processes resident in main memory.
    - Each process with its own address space
  - Only "active" code and data is actually in memory

Why Virtual Memory?

- Use main memory as a Cache for the Disk
  - Address space of a process can exceed physical memory size
  - Sum of address spaces of multiple processes can exceed physical memory
- Simplify Memory Management
  - Multiple processes resident in main memory.
    - Each process with its own address space
  - Only “active” code and data is actually in memory
- Provide Protection
  - One process can’t interfere with another.
    - because they operate in different address spaces.
  - User process cannot access privileged information
    - different sections of address spaces have different permissions.

Principle of Locality

- Program and data references within a process tend to cluster

Principle of Locality

- Program and data references within a process tend to cluster
- Only a few pieces of a process will be needed over a short period of time (active data or code)

Principle of Locality

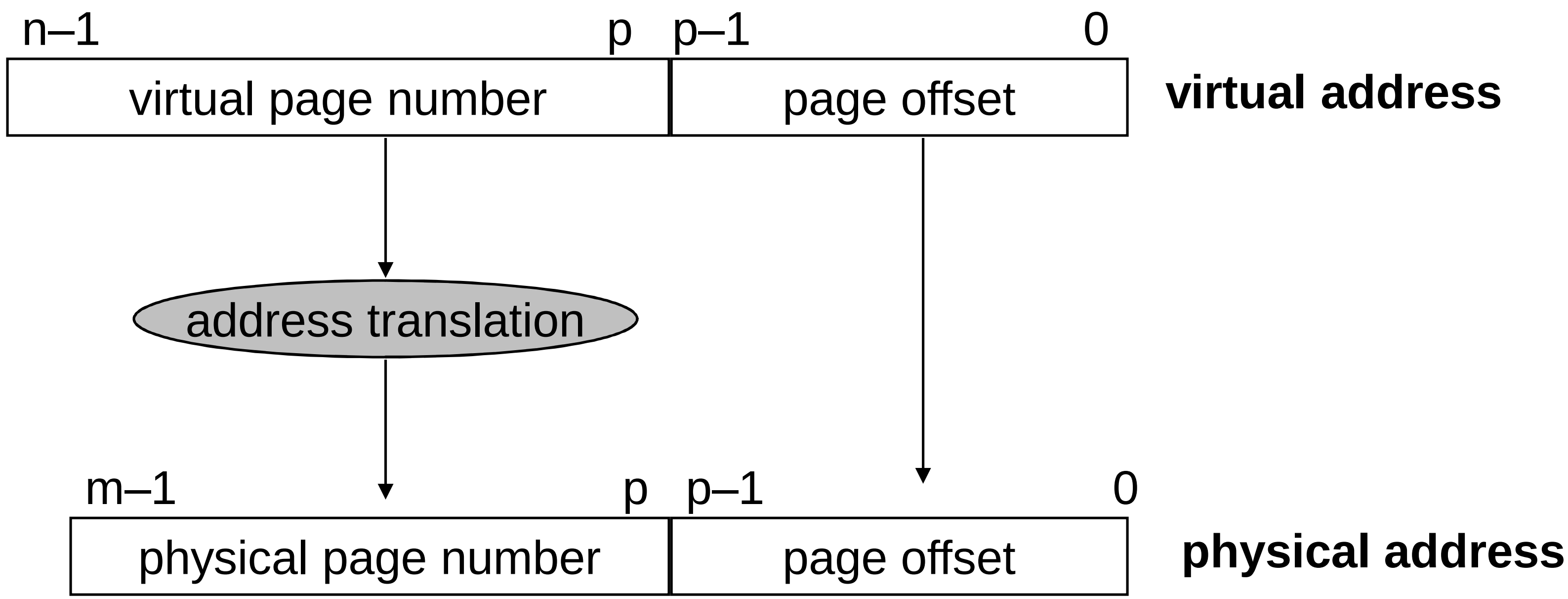
- Program and data references within a process tend to cluster
- Only a few pieces of a process will be needed over a short period of time (active data or code)
- Possible to make intelligent guesses about which pieces will be needed in the future

Principle of Locality

- Program and data references within a process tend to cluster
- Only a few pieces of a process will be needed over a short period of time (active data or code)
- Possible to make intelligent guesses about which pieces will be needed in the future
- This suggests that virtual memory may work efficiently

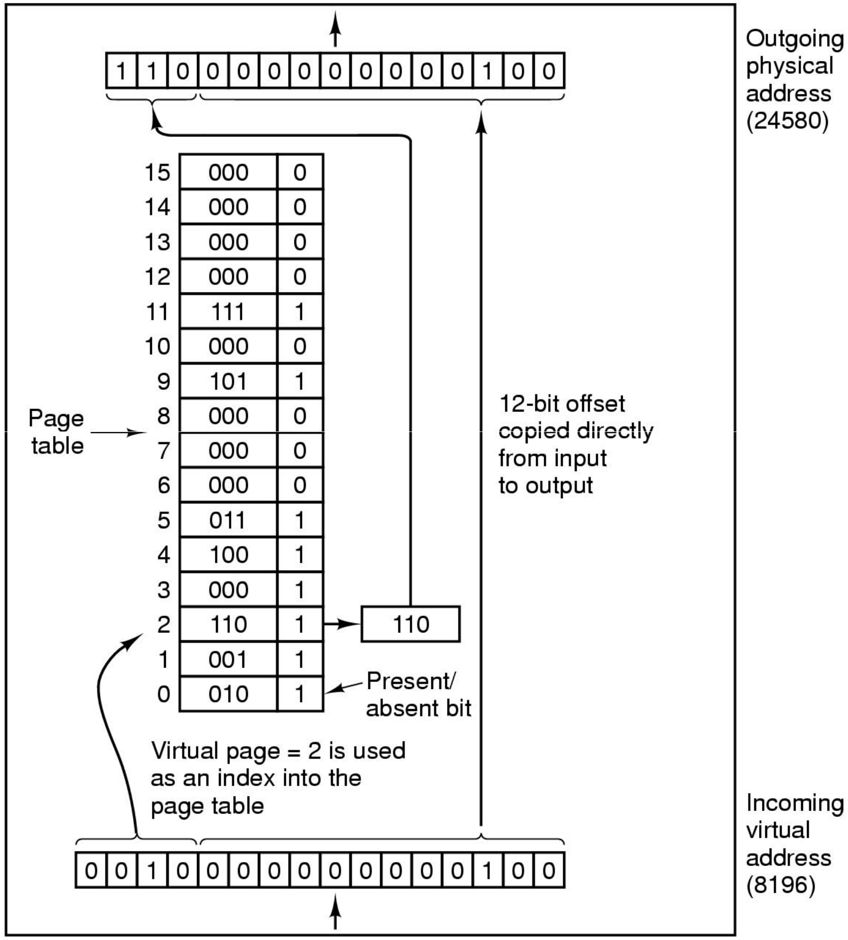
VM Address Translation

- Parameters
  - $P = 2^p$ = page size (bytes).
  - $N = 2^n$ = Virtual address limit
  - $M = 2^m$ = Physical address limit



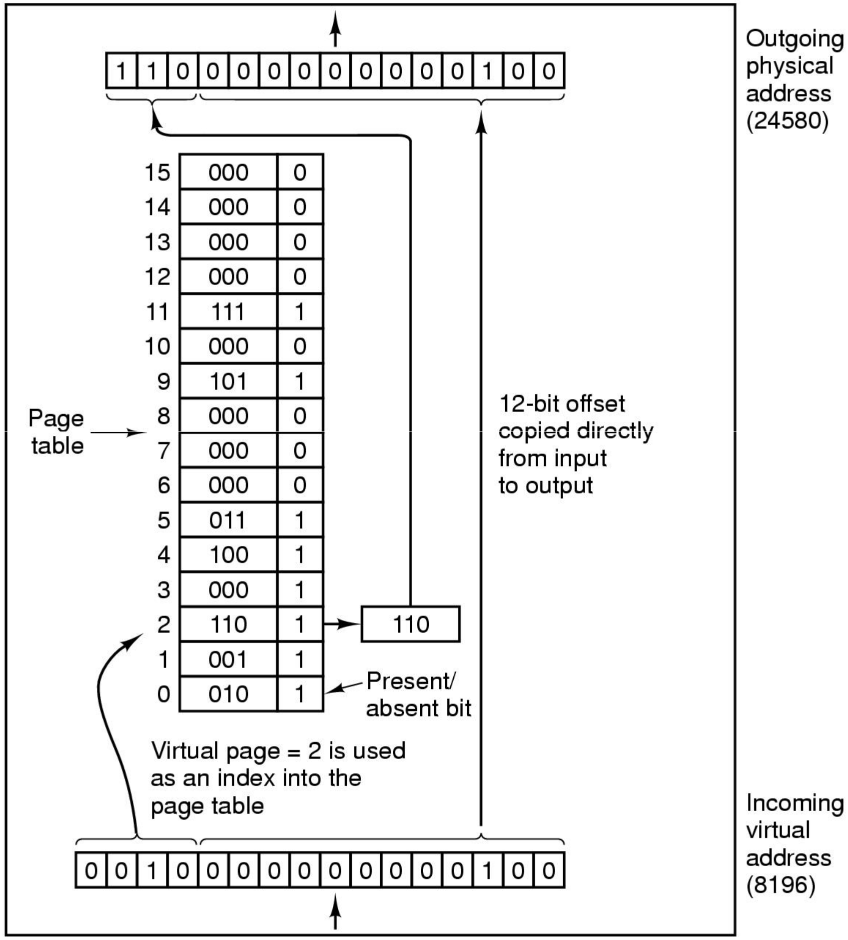
Page offset bits don't change as a result of translation

Page Table



- Keeps track of what pages are in memory

Page Table



- Keeps track of what pages are in memory
- Provides a mapping from virtual address to physical address

Handling a Page Fault

- Page fault
  - Look for an empty page in RAM
    - May need to write a page to disk and free it

Handling a Page Fault

- Page fault
  - Look for an empty page in RAM
    - May need to write a page to disk and free it
  - Load the faulted page into that empty page

Handling a Page Fault

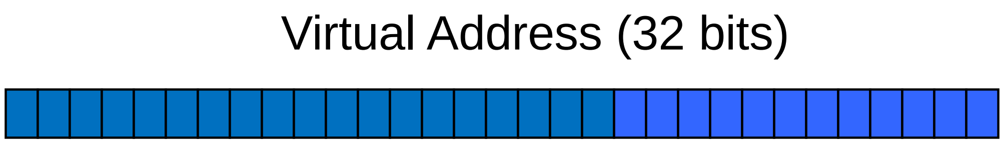
- Page fault
  - Look for an empty page in RAM
    - May need to write a page to disk and free it
  - Load the faulted page into that empty page
  - Modify the page table

Addressing

- 64MB RAM (2^26)

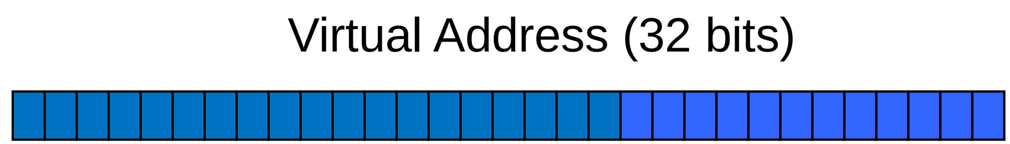
Addressing

- 64MB RAM (2^26)
- 2^32 (4GB) total memory



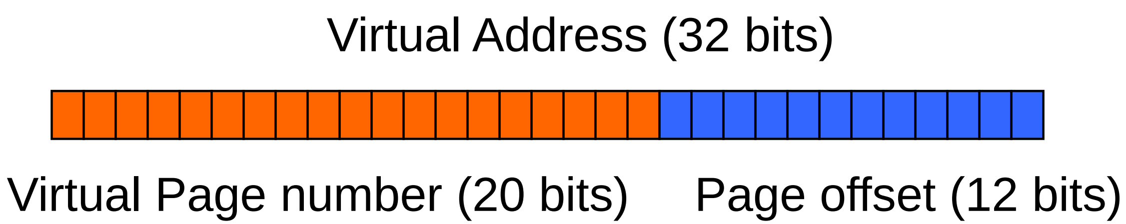
Addressing

- 64MB RAM (2^26)
- 2^32 (4GB) total virtual memory
- 4KB page size (2^12)



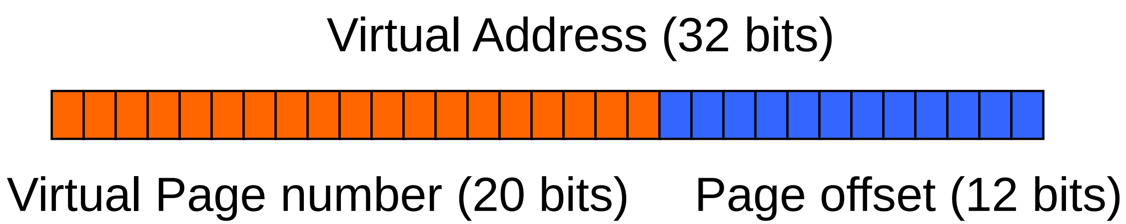
Addressing

- 64MB RAM (2^26)
- 2^32 (4GB) total memory
- 4KB page size (2^12)
- So we need 2^12 for the offset, we can use the remainder bits for the page



Addressing

- 64MB RAM (2^26)
- 2^32 (4GB) total memory
- 4KB page size (2^12)
- So we need 2^12 for the offset, we can use the remainder bits for the page
  - 20 bits, we have 2^20 pages (1M pages)



Address Conversion

- That 20bit page address can be optimized in a variety of ways
  - Translation Look-aside Buffer

Translation Lookaside Buffer (TLB)

- Each virtual memory reference can cause two physical memory accesses
  - One to fetch the page table
  - One to fetch the data

Translation Lookaside Buffer (TLB)

- Each virtual memory reference can cause two physical memory accesses
  - One to fetch the page table
  - One to fetch the data
- To overcome this problem a high-speed cache is set up for page table entries

Translation Lookaside Buffer (TLB)

- Each virtual memory reference can cause two physical memory accesses
  - One to fetch the page table
  - One to fetch the data
- To overcome this problem a high-speed cache is set up for page table entries
- Contains page table entries that have been most recently used (a cache for page table)

Translation Lookaside Buffer (TLB)

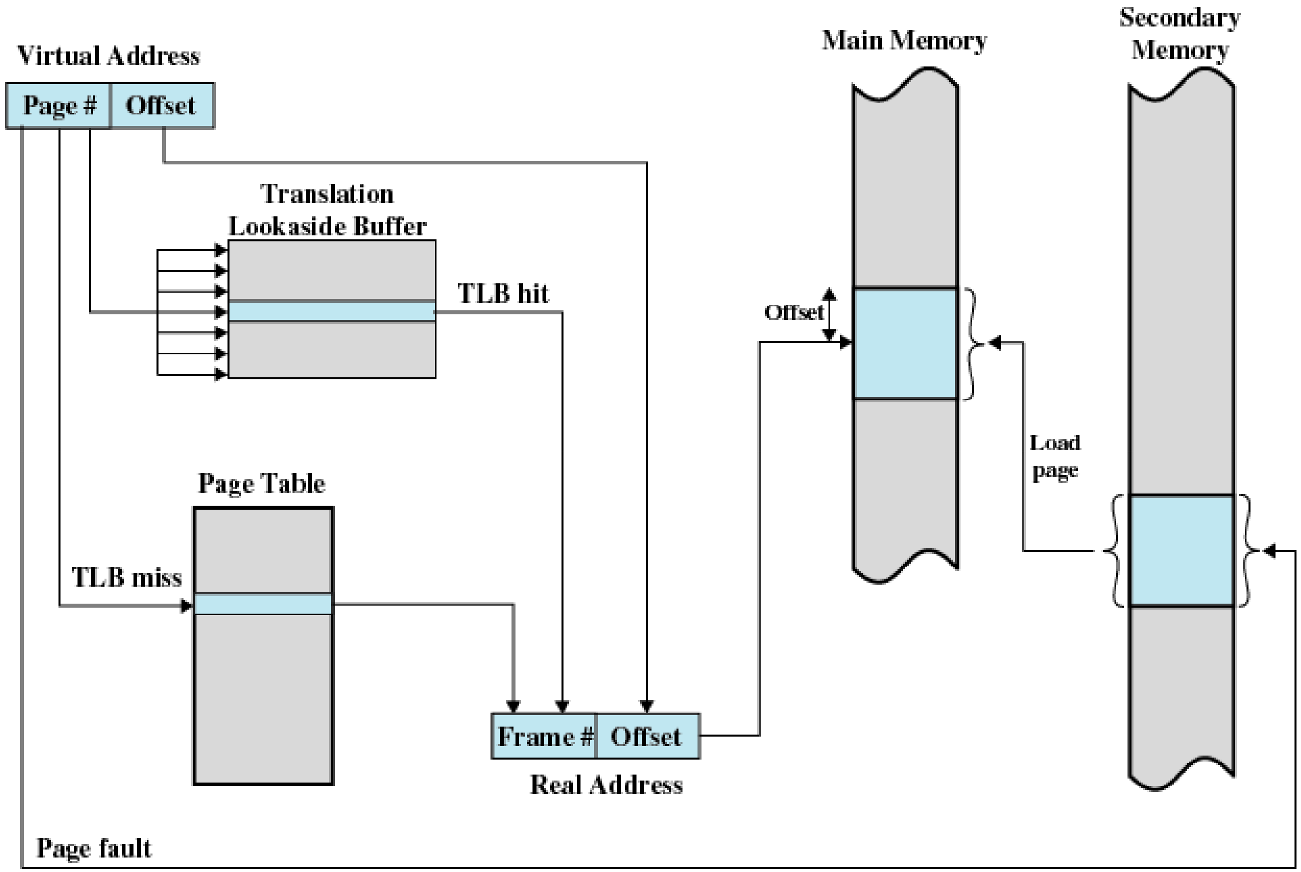


Figure 8.7 Use of a Translation Lookaside Buffer

Effective Access Time

- Effective Access time (EAT)
  - m – memory cycle, α - hit ratio, ε - TLB lookup time
  - $Eat = (m + \varepsilon)\alpha + (2m + \varepsilon)(1 - \alpha) = 2m + \varepsilon - m\alpha$

Multilevel Page Tables

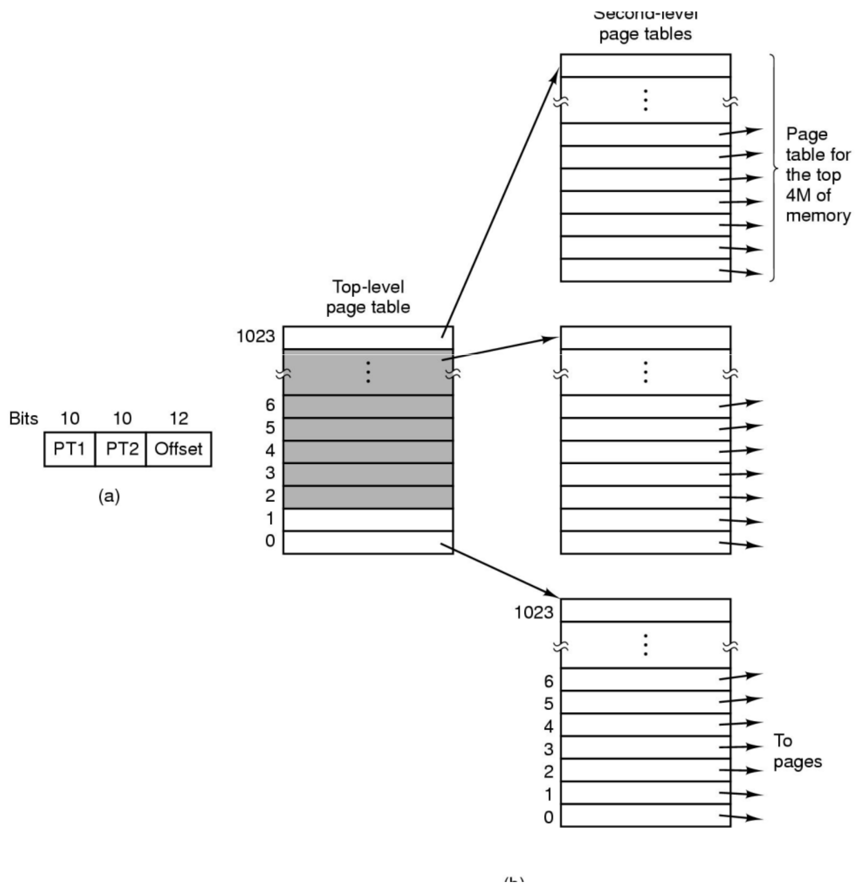
- Given:
  - 4KB (2^{12}) page size
  - 32-bit address space
  - 4-byte PTE

Multilevel Page Tables

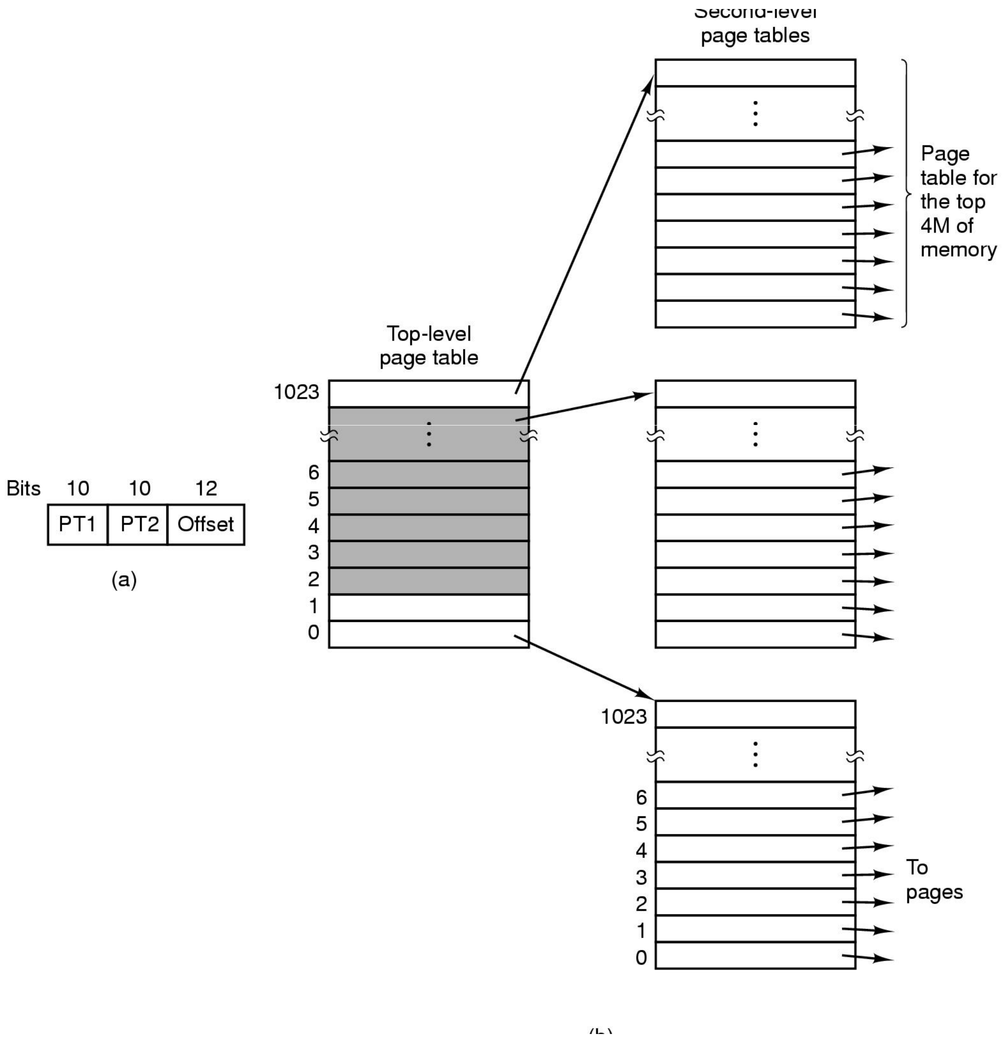
- Given:
  - 4KB (2^{12}) page size
  - 32-bit address space
  - 4-byte PTE
- Problem:
  - Would need a 4 MB page table!
    - 2^{20} *4 bytes

Multilevel Page Tables

- Given:
  - 4KB (2^{12}) page size
  - 32-bit address space
  - 4-byte PTE
- Problem:
  - Would need a 4 MB page table!
    - 2^{20} *4 bytes
- Common solution
  - multi-level page tables
  - e.g., 2-level table (P6)
    - Level 1 table: 1024 entries, each of which points to a Level 2 page table.
    - Level 2 table: 1024 entries, each of which points to a page



Summary: Multi-level Page Tables



- Instead of one large table, keep a tree of tables
  - Top-level table stores pointers to lower level page tables
- First *n* bits of the page number == index of the top-level page table
- Second *n* bits == index of the 2nd-level page table
- Etc.

Example: Two-level Page Table

- 32-bit address space (4GB)

Example: Two-level Page Table

- 32-bit address space (4GB)
- 12-bit page offset (4kB pages)

Example: Two-level Page Table

- 32-bit address space (4GB)
- 12-bit page offset (4kB pages)
- 20-bit page address
  - First 10 bits index the top-level page table
  - Second 10 bits index the 2nd-level page table
  - 10 bits == 1024 entries * 4 bytes == 4kB == 1 page

Example: Two-level Page Table

- 32-bit address space (4GB)
- 12-bit page offset (4kB pages)
- 20-bit page address
  - First 10 bits index the top-level page table
  - Second 10 bits index the 2nd-level page table
  - 10 bits == 1024 entries * 4 bytes == 4kB == 1 page
- Need three memory accesses to read a memory location

Why use multi-level page tables?

- Split one large page table into many page-sized chunks
  - Typically 4 or 8 MB for a 32-bit address space

Why use multi-level page tables?

- Split one large page table into many page-sized chunks
  - Typically 4 or 8 MB for a 32-bit address space

- Advantage: less memory must be reserved for the page tables
  - Can swap out unused or not recently used tables

Why use multi-level page tables?

- Split one large page table into many page-sized chunks
  – Typically 4 or 8 MB for a 32-bit address space

- Advantage: less memory must be reserved for the page tables
  – Can swap out unused or not recently used tables

- Disadvantage: increased access time on TLB miss
  – *n+1* memory accesses for *n*-level page tables

Address Conversion

- That 20bit page address can be optimized in a variety of ways
  - Translation Look-aside Buffer
  - Multilevel Page Table
  - Inverted Page Table

Problems

Problem 1

For each of the following decimal virtual addresses, compute the virtual page number and offset for a 4 KB page and for an 8 KB page: 20000, 32768, 60000.

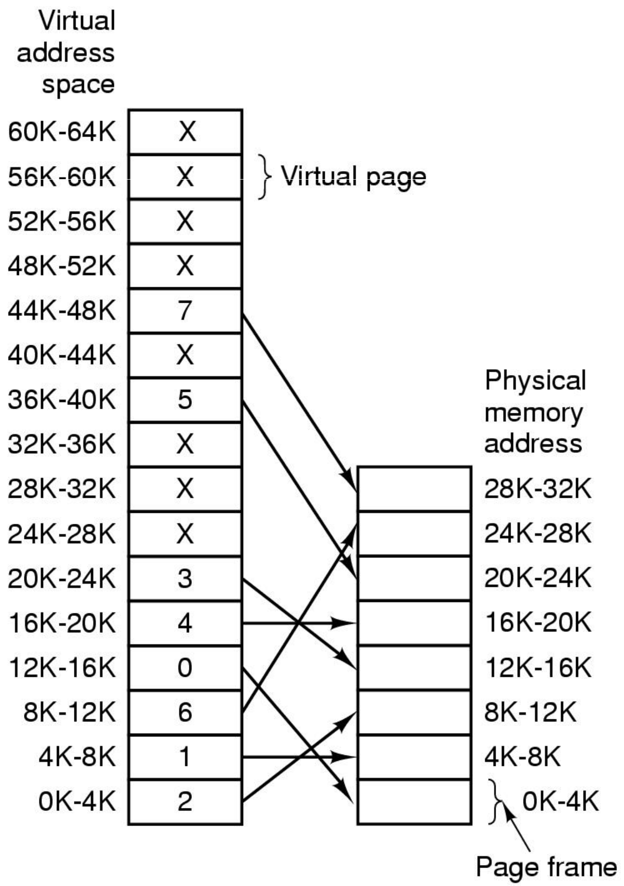
Problem 1 Solution

Address	**Page Number (4KB)**	**Offset (4KB)**	**Page Number (8KB)**	**Offset (8KB)**
20000	4	3616	2	3616
32768	8	0	4	0
60000	14	2656	7	2656

Problem 2

Consider the page table of the figure. Give the physical address corresponding to each of the following virtual addresses:

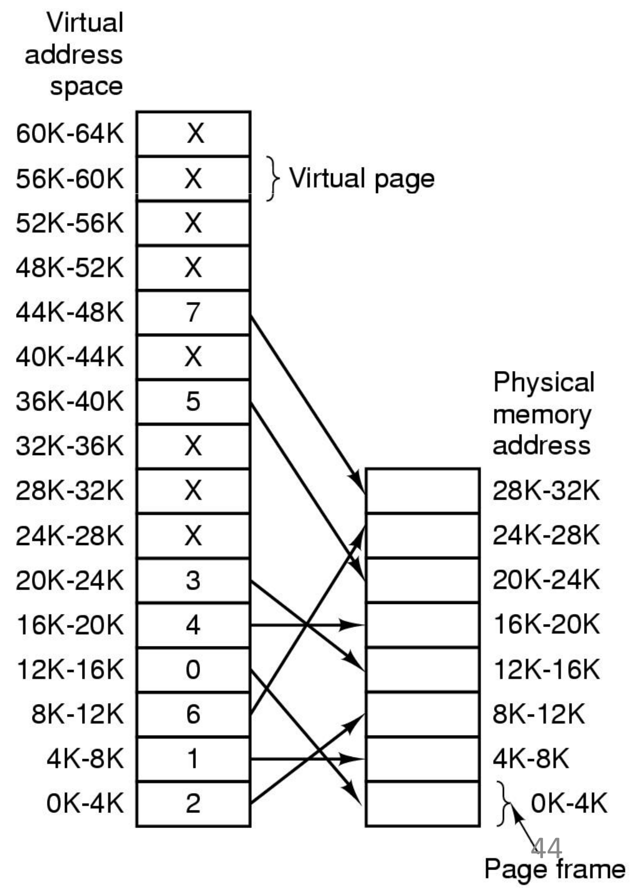
- 29
- 4100
- 8300



Problem 2 Solution

Consider the page table of the figure. Give the physical address corresponding to each of the following virtual addresses:

- 29
- 4100
- 8300



29: Physical address: 8K + 29 = 8221
4100: Physical address: 4K + (4100 – 4K) = 4100
8300: Physical address: 24K + (8300 – 8K) = 24684

Problem 3

A machine has 48 bit virtual addresses and 32 bit physical addresses. Pages are 8 KB. How many entries are needed for the page table?

Problem 3 Solution

A machine has 48 bit virtual addresses and 32 bit physical addresses. Pages are 8 KB. How many entries are needed for the page table?

Page size = 8 KB = 2^13 B

Offset = 13 bits

# of virtual pages = 2^(48 – 13) = 2^35 = # of entries in page table

Problem 4

Consider a machine such as the DEC Alpha 21064 which has 64 bit registers and manipulates 64-bit addresses.

If the page size is 8KB, how many bits of virtual page number are there?

If the page table used for translation from virtual to physical addresses were 8 bytes per entry, how much memory is required for the page table and is this amount of memory feasible?

Problem 4 Solution

Page size = 8 KB = 2^13 B
Offset = 13 bits
Bits for virtual page number = (64 – 13) = 51

# of page table entries = 2^51
Size of page table = 2^51 * 8 B =2^54 B = 2^24 GB

Problem 5

A computer with a 32-bit address uses a two-level page table. Virtual addresses are split into 9-bit top-level page table field, an 11 bit second-level page table field, and an offset.

How large are the pages and how many are there in the address space?

Problem 5 Solution

A computer with a 32-bit address uses a two-level page table. Virtual addresses are split into 9-bit top-level page table field, an 11 bit second-level page table field, and an offset.

How large are the pages and how many are there in the address space?

Offset = 32 – 9 – 11 = 12 bits

Page size = 2^12 B = 4 KB

Total number of pages possible = 2^9 * 2^11 = 2^20

Problem 6

Fill in the following table:

Virtual Address (bits)	Page Size	# of Page Frames	# of Virtual Pages	Offset Length (bits)	Addressable Physical Memory
16	256 B	2^2			
32	1 MB	2^4			
32	1 KB	2^8			
64	16 KB	2^20			
64	8 MB	2^16			

Problem 6 Solution

Fill in the following table:

Virtual Address (bits)	Page Size	# of Page Frames	# of Virtual Pages	Offset Length (bits)	Addressable Physical Memory
16	256 B = 2^8	2^2	2^8	8	2^10 = 1 KB
32	1 MB = 2^20	2^4	2^12	20	2^24 = 16 MB
32	1 KB = 2^10	2^8	2^22	10	2^18 = 256 KB
64	16 KB = 2^14	2^20	2^50	14	2^34 = 16 GB
64	8 MB = 2^23	2^16	2^41	23	2^39 = 512 GB

Problem 7

Fill in this table with the correct page evictions.

Physical memory contains 4 pages.

Page Accesses	0	1	2	3	4	1	3	4	4	5	3	1	2	0	4	5	4
Optimal	-	-	-	-	0	-	-	-	-	4	-	-	-	3	2	-	-
FIFO	-	-	-	-													
LRU	-	-	-	-													
LFU	-	-	-	-													
MRU	-	-	-	-													

Problem 7 Solution

Fill in this table with the correct page evictions.
Physical memory contains 4 pages.

Page Accesses	0	1	2	3	4	1	3	4	4	5	3	1	2	0	4	5	4
Optimal	-	-	-	-	0	-	-	-	-	4	-	-	-	3	2	-	-
FIFO	-	-	-	-	0	-	-	-	-	1	-	2	3	4	5	1	-
LRU	-	-	-	-	0	-	-	-	-	2	-	-	4	5	3	1	-
LFU	-	-	-	-	0	-	-	-	-	2	-	-	5	2	-	0	-
MRU	-	-	-	-	3	-	1	-	-	4	-	3	-	-	0	-	-