Memory Management

Week 07 – Tutorial

Virtual Memory

Team

Instructors

Giancarlo Succi

Teaching Assistants

Nikita Lozhnikov (also Tutorial Instructor)

Manuel Rodriguez

Shokhista Ergasheva

Problem 3.5

- What is the difference between a physical address and a virtual address?

Problem 3.5 – Solution (1/5)

Long answer:

- Physical addresses:
  - The addresses that are available in main memory unit
- Virtual Addresses:
  - The addresses generated by the CPU while running the application program

See TB page 195

Problem 3.5 – Solution (2/5)

- Physical addresses:
  - The physical memory is not enough to accommodate all the programs that are to be executed, so only the application that is currently being executed is put to the physical memory
- Virtual Addresses:
  - As the physical memory is not enough to accommodate all the programs, CPU generates virtual addresses to all the programs during compile time. These virtual addresses are mapped to the physical address during run time or execution time

Problem 3.5 – Solution (3/5)

- Physical addresses:
  - Refer to real memory location, they physically exist. These addresses are loaded into the memory address register
- Virtual Addresses:
  - Are not real memory location. An illusion is created to the user that each user program is assigned to a memory location

Problem 3.5 – Solution (4/5)

- Physical addresses:
  - User cannot access these addresses. The MMU unit accesses data from these address locations through memory bus
- Virtual Addresses:
  - User can access virtual address locations

Problem 3.5 – Solution (5/5)

- Short answer:

  Real memory uses physical addresses. These are the numbers that the memory chips react to on the bus. Virtual addresses are the logical addresses that refer to a process' address space. Thus a machine with a 32-bit word can generate virtual addresses up to 4 GB regardless of whether the machine has more or less memory than 4 GB.

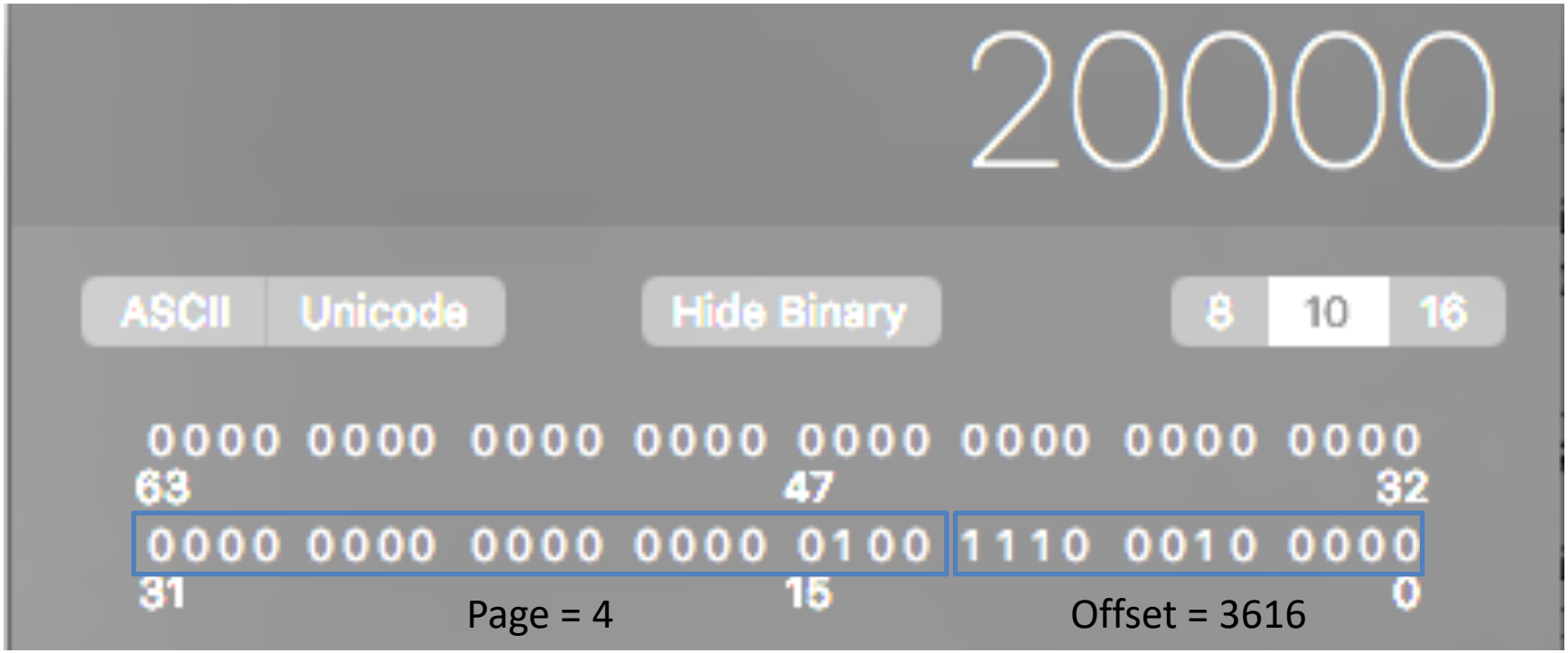
Problem 3.6

- 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:
  A. 20000
  B. 32768
  C. 60000

Problem 3.6 – Solution (1/4)

- Step 1. Page offset
  - Let's take page size of 4 KB first. We must be able to access every single address (every byte, actually) within a page. To do this, we need 4 KB ($4096 = 2^{12}$) of offsets.
  - It means that 12 bits of an address will be given for the offset

Problem 3.6 – Solution (2/4)



Step 2. Virtual address 20000, page size 4 KB

Problem 3.6 – Solution (3/4)

- Step 3. Calculating other values
  - As it is can be derived from the picture, for an 8 KB page size page number is 2 and offset is the same (3616)
  - Now try to solve the rest yourselves

Problem 3.6 – Solution (4/4)

Virtual address	Page, offset page size is 4 KB	Page, offset page size is 8 KB
20000	(4, 3616)	(2, 3616)
32768	(8, 0)	(4, 0)
60000	(14, 2656)	(7, 2656)

Problem 3.7 (1/2)

- Using the page table of Fig. 3-9 (next slide), give the physical address corresponding to each of the following virtual addresses:
  A. 20
  B. 4100
  C. 8300

Problem 3.7 (2/2)

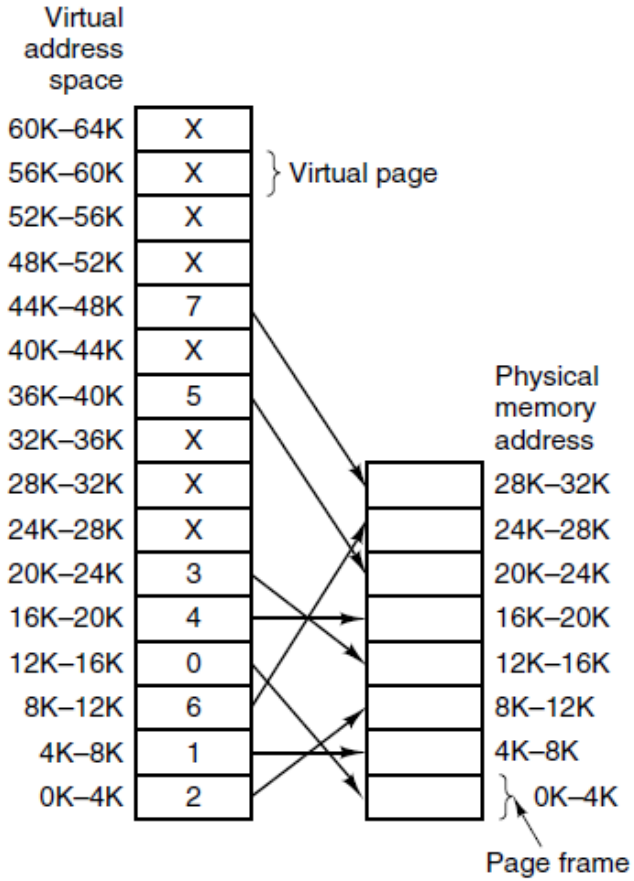


Figure 3-9. The relation between virtual addresses and physical memory addresses (notice X signs in virtual memory)

Problem 3.7 – Solution (1/2)

- Virtual address 20 belongs to virtual page 0 which maps to physical page 2.
- It means that physical addresses are starting from 8192
- So physical address corresponding to virtual address 20 is 8212
- No try to find other addresses

Problem 3.7 – Solution (2/2)

- Answer:
  A. 8212
  B. 4100
  C. 24684

Problem 3.11

- Question:
  Consider the following C program:

See TB page 206

```
int X[N];
int step = M;       /* M is some predefined constant */
for (int i = 0; i < N; i += step)
  X[i] = X[i] + 1;
```

- (a) If this program is run on a machine with a 4-KB page size and 64-entry TLB, what values of M and N will cause a TLB miss for every execution of the inner loop?
- (b) Would your answer in part (a) be different if the loop were repeated many times? Explain.

Problem 3.11 – Solution (1/3)

- Let's have a closer look on the memory structure:

X[0]	X[…]	X[1023]	X[1024]	X[…]	X[2047]	X[1024 * k + 0]	…
Page 0			Page 1			Page k	

- *Minimum int size* is 4 bytes which gives us 1024 integers per 4-KB memory page

Problem 3.11 – Solution (2/3)

- It means that after X[0] is accessed, accessing any of the succeeding elements up to X[1023] would not generate TLB fault
- However, if we try to access X[0], X[1024], X[2048] and so on, each time a TLB miss will occur which means that M should be at least 1024

Problem 3.11 – Solution (3/3)

b) M should still be at least 1024 to cause a TLB miss for every execution of the inner loop

- Assuming that such page replacement algorithm as FIFO is used, we need to fill the whole TLB and make one more reference to an absent page to make sure that the page 0 is not in the TLB at the beginning of each outer cycle
- It means that we need to have at least 65 pages. In this case N should be at least 64 * 1024 + 1

Problem 3.12

- The amount of disk space that must be available for page storage is related to:
  - the maximum number of processes, *n*
  - the number of bytes in the virtual address space, *v*
  - the number of bytes of RAM, *r*

  Give an expression for the worst-case disk-space requirements. How realistic is this amount?

Problem 3.12 – Solution (1/2)

- The total virtual address space for all the processes combined is ***n*v***, so this much storage is needed for pages
- However, an amount r can be in RAM, so the amount of disk storage required is only ***n*v - r***

Problem 3.12 – Solution (2/2)

- This amount is far more than is ever needed in practice because rarely will there be n processes actually running and even more rarely will all of them need the maximum allowed virtual memory

End

Week 07 – Tutorial