Preview

- Virtual Memory
- Virtual Memory with Paging
  - Page tables
  - Physical Address Mapping(MMU)
  - Page Table Structures
    - Hashed Page Tables
    - Inverted Page Tables
    - Multiple level page tables

Virtual Memory

Motivation

- Initial idea of executing a program is that the entire logical address space of a process must be in physical memory before the process can access (swapping)
- But since size of a program grows tremendously based on the complexity of modern software, and limited size of physical memory, initial idea is not possible

Virtual Memory

Overlay

- Split a program into pieces (overlays)
- Overlay 0 starts running first; when it finishes its job, it will call another overlay.
- Overlays were kept on the disk and swapped in and out of memory by OS.
- The work of splitting the program into pieces had to be done by programmer
- Splitting up large program into small pieces is time consuming

Virtual Memory

Virtual Memory

- Virtual memory is a technique that allows the execution of processes that may not be entirely in memory
- The OS need to keep tracking the parts of the program currently located in main memory and the rest on the disk.
- Virtual memory idea can be used in a multiprogramming system.

Virtual Memory

- Three approaches for managing virtual memory
  - Paging
  - Segmentation
  - Segmentation with paging

Virtual Memory with Paging

Paging

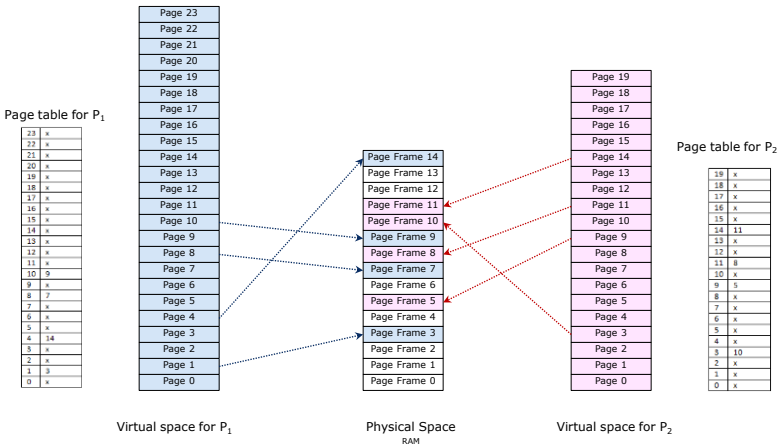
- Virtual address space is divided into units called **pages**.
- The corresponding units in the physical memory is called **page frames**.
  the size of a page = the size of a page frame
- Size of page is usually between 512 byte and 64KB

Virtual Memory with Paging

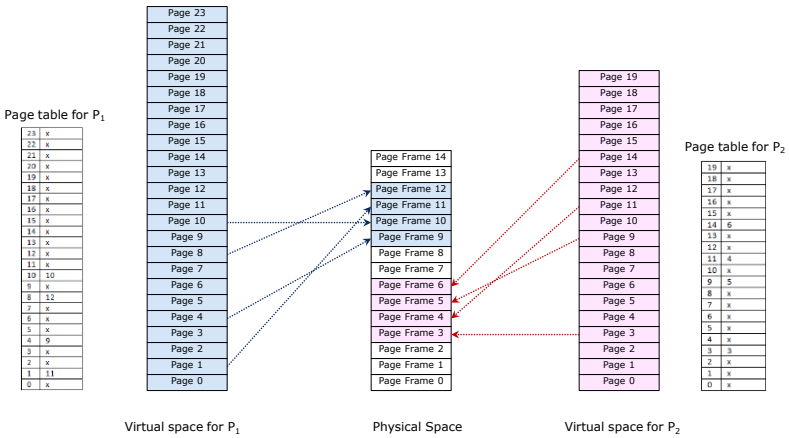
Mapping from Virtual Memory Address to Physical Memory Address

- PC save the address of next instruction (an virtual address in virtual space).
- When virtual memory is used, a virtual address (which is generated by program) do not go directly onto the memory bus.
- Instead, they goes to an MMU (Memory Management Unit) that maps the virtual addresses onto the physical memory addresses based on the memory map (page table).

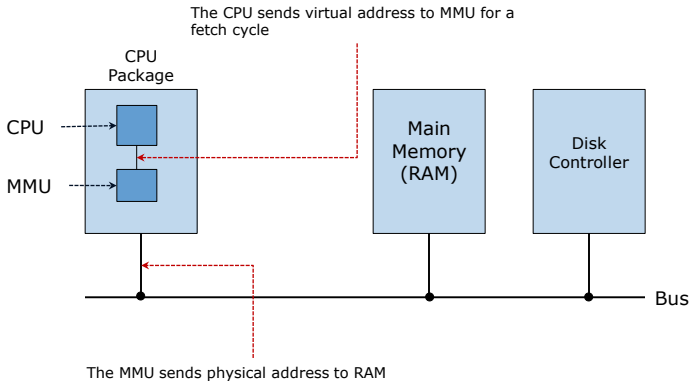
Virtual Memory with Paging



Virtual Memory with Paging



Virtual Memory with Paging



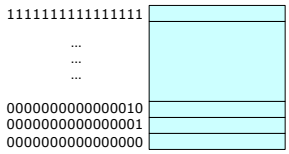
Virtual Memory with Paging

- 1 Byte = 8 bits
- 1 KB (kilo)= 2^{10} Bytes
- 1 MB (mega)= 2^{20} Bytes
- 1 GB (giga)= 2^{30} Bytes
- 1 TB (tera)= 2^{40} Bytes
- 1 PB (peta)= 2^{50} Bytes
- 1 EB (exa)= 2^{60} Bytes
- 1 ZB (zetta)= 2^{70} Bytes
- 1 YB (yotta)= 2^{80} Bytes

Virtual Memory with Paging

Example)

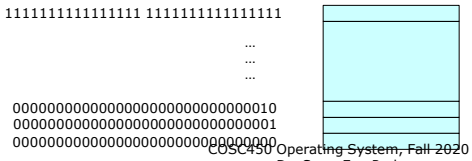
- A computer that can generate 16-bit address – this system support 64KB virtual space for a process ($0 \sim 2^{16}\text{-}1 = 0 \sim 64 \times 2^{10}$)
- A system is using the virtual memory with page size 4kB
- A computer has 32 KB memory
- There are 16 (64/4) virtual pages and 8 (32/4) page frames.



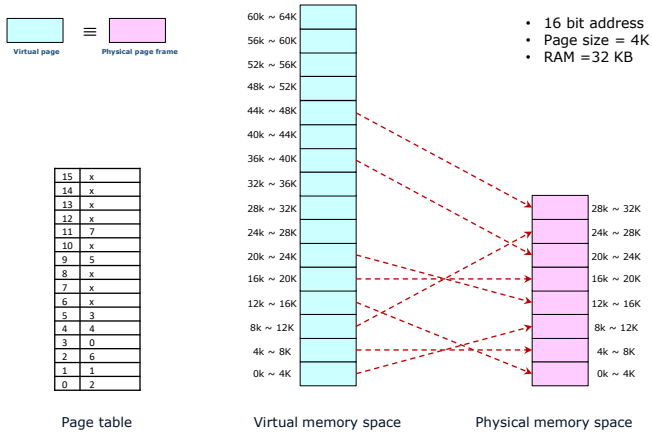
Virtual Memory with Paging

Example)

- A System supports
  - Generate 32bit address
  - Size of memory (RAM) = 1GB
  - Page size = 4 KB
- Since system generate 32bit address, system supports 4GB virtual space (0 ~ 2^{32}-1 = 0 ~ 4 $\times 2^{30}$) for a process
- Number of pages = possible virtual space/ a page size
  - 4GB/4KB=4$\times 2^{30}$/4$\times 2^{10}$ = 2^{20} =1048576 pages
- Number of page frames = size of memory / a page size
  - 1GB/4KB=1$\times 2^{30}$/4$\times 2^{10}$= 2^{18} = 262144 page frames.

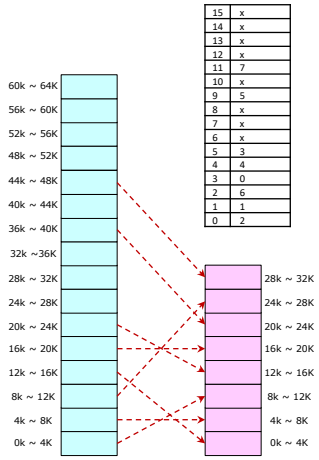


Virtual Memory with Paging



Virtual Memory with Paging

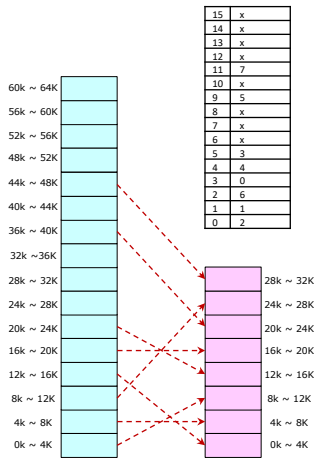
- **MOV REG, 0** ;;move content of address 0 to register.
  - virtual address 0 is send to MMU.
  - MMU check virtual address 0 is belongs to virtual page 0 (0 ~ 4095 (= 4×2^{10}).
  - MMU check memory map such that page 0 is map to page frame 2 (8K ~ 12K)
  - MMU send address 8192 (= 8×2^{10}) physical address to address bus.



Virtual Memory with Paging

MOV REG, 8900 ;;

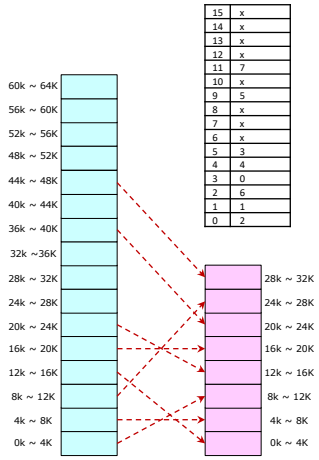
- Virtual address 8900 is send to MMU
- MMU calculate that address 8900 is belongs to virtual page 2 (=between 8 KB ~ 12 KB) (8192~12288)
- MMU check memory map such that virtual page 2 is map to physical frame 6 (24KB ~ 28 K: (24576 ~28672)
- MMU send address 24576 + (8900 - 8192) = 25287 physical address to address



Virtual Memory with Paging

MOV REG 24660

- Virtual address **24660** is send to MMU
- MMU calculate that address **24660** is belongs to virtual page 6 (=between 24 KB ~ 28 KB: 24576 ~28672)
- MMU check memory map such that virtual page 6 is not in physical space yet. (page fault)
- When a page fault occurs, operating system
  - Pick one of physical frame
  - Write back to disk and then
  - Fetch the page to the frame just freed.
  - Change the memory map.



Virtual Memory with Paging

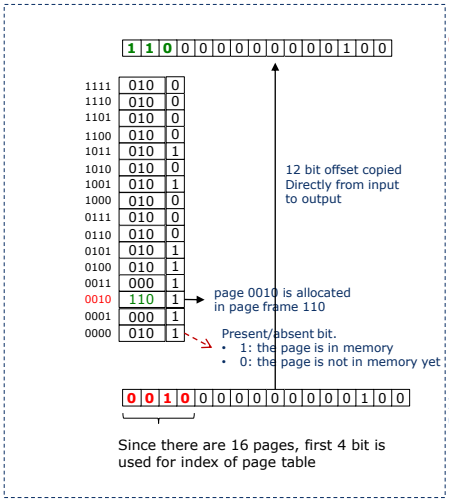
(Physical address mapping)

How to map virtual address into physical address by MMU with Previous Example

- Generate 16bit address
- Size of memory (RAM) = 32KB
- Page size = 4 KB
- #of pages = 16, #of page frame =8

- The incoming 16 bit address is split into 4-bit page number (since there are 16 pages) and 12bit offset (since size of page is 4KB).
- With 4 bits page number, 2^4 = 16 page number is available.
- The first 4-bits are used as an index into the page table (since 2^4 = 16 pages),
- With 12-bits offset, we can address 2^{12} = 4KB within a page.

Virtual Memory with Paging
(Physical address mapping)



Outgoing Physical address = 110000000000100 = 24580

- 16 bit virtual address
- 4 KB page size
- 32 KB Memory space
- Number of Pages = = $2^{16}/(4\times2^{10})= 2^4 = 16$
- Number of page frames = $(32\times2^{10})/ (4\times2^{10})= 2^3 = 8$

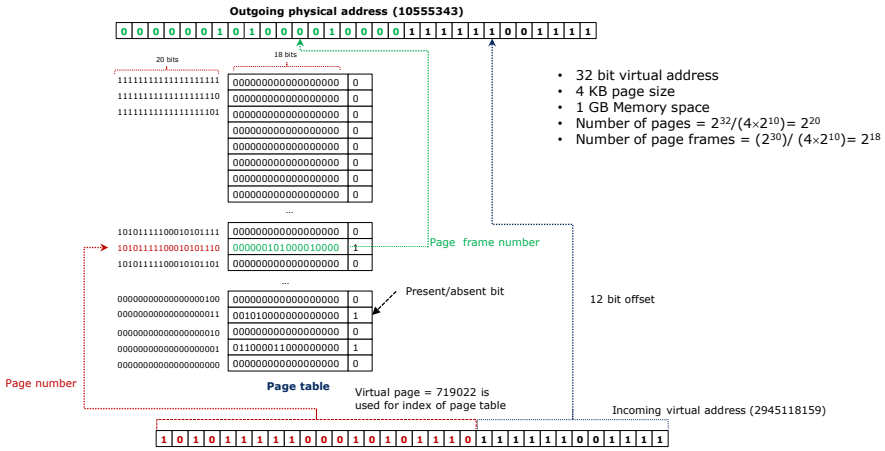
Incoming virtual address = 8196 = 0010000000000100

Virtual Memory with Paging
(Physical address mapping)

Example)

- A system generate 32 bit virtual address.
- Size of Memory: 1 GB (Giga Byte)
- Size of Page: 4 KB (Kilo Byte)
  - Size of virtual space = 2^{32}
  - Number of pages = 2^{32} / $(4 \times 2^{10})= 2^{20}$
  - Page table have 2^{20} entries
  - Number of page frames = $(2^{30})/(4 \times 2^{10})= 2^{18}$
- If a system generate an virtual address 10101111100010101110111111001111, first 20 bit will be used for index of page table, rest 12 bit (page size = 4KB = 2^{12}) will used for offset.
- Since there are 2^{18} page frames (RAM), 18 bit must be reserved for saving page frame number in page table entry.

Virtual Memory with Paging
(Physical address mapping)



Virtual Memory with Paging

Ex) A modern computer generate 32 bit virtual address with 4 KB page size. Assume that 2GB RAM is installed in the system.

Since system generate 32 bit virtual address

- 2^{32} bit = 2^{22} KB ($2^{10} \times 2^{22}$ = 4GB) virtual space (maximum size of a process)
- Number of pages = virtual space / size of a page
  = 2^{22} (KB)/ 4 (KB)= 2^{20} pages
- Since there are 2^{20} pages, there are 2^{20} entries in a page table for a process.

Since size of RAM is 2GB

- Number of page frames = size of RAM / size of a page = 2GB/4KB
  = $(2 \times 2^{30})/ (4 \times 2^{10})= 2^{19}$ page frames
- Since there are 2^{19} page frames, 19 bit is used to save page frame number in each entries in page table.

Page Table

- The purpose of the page table is to map virtual pages onto page frame.
- Two major issues using page table
  - The page table can be extremely large
  - The mapping must be fast

Page Table

The page table can be extremely large

Ex) A modern computer with virtual address 32 – bit with 4 KB page size

- 2^{32} bit = 2^{22} KB ($2^{10} \times 2^{22}$) virtual space
- 2^{22} / 4 = 2^{20} =1048576 pages
- Need 1048576 entries in the page table
- Since each process has its own virtual space, each process need its own page table.!!!

Page Table

The mapping must be fast

- Build a single table consisting of an array of fast hardware registers, with one entry for each virtual page, indexed by virtual page number. --- **very expensive**!!
- The page table can be entirely in main memory – all the hardware needs is a single register that point to the start of the page table --- need one or more memory references to read page table entries.