C Primer and Virtual Memory

15-213 / 18-213: Introduction to Computer Systems
10th Recitation, March 26th, 2012

Grant Skudlarek

Section H

Today

- **Shell Lab**
- Malloc Lab
- C Primer
- Virtual Memory

Shell Lab

- Due Thursday, March 29!!!

Today

- Shell Lab
- **Malloc Lab**
- C Primer
- Virtual Memory

Malloc Lab

- Lab goes out Thursday March 29
- Due Thursday April 12
- As always, read the documentation
- Start early (seriously)

Today

- Shell Lab
- Malloc Lab
- **C Primer**
- Virtual Memory

C Primer – Basics of C, style and conventions

- Saving you from malloc misery
- Basics
- Useful language features
- Debugging conventions
- The C Programming Language by Kernighan and Ritchie

Basics (just review)

- **Statically allocated arrays:**
  - `int prices[100];`
  - Getting rid of magic numbers:
    - `int prices[NUMITEMS];`
- **Dynamically allocated arrays:**
  - `int *prices2 = (int *) malloc(sizeof(int) * var);`
- **Which is valid:**
  - `prices2 = prices;`
  - `prices = prices2;`
- **The & operator:**
  - `&prices[1]` is the same as `prices+1`
- **Function Pointer:**
  - `int (*fun)();`
  - Pointer to function returning int

Pg 101 K&R

- `char **argv`
  - argv: pointer to a pointer to a char
- `int (*daytab)[13]`
  - daytab: pointer to array[13] of int
- `int *daytab[13]`
  - daytab: array[13] of pointer to int
- `char (*(*x())[])()`
  - x: function returning pointer to array[] of pointer to function returning char
- `char (*(*x[3])())[5]`
  - x: array[3] of pointer to function returning pointer to array[5] of char
- **Takeaway**
  - There is an algorithm to decode this (see K&R pg. 101)
  - Always use parenthesis!!
  - Typedef

Typedef

- **For convenience and readable code**
- **Example:**
  -
    ```
    typedef struct
    {
     int x;
     int y;
    } point;
    ```
- **Function Pointer example:**
  - `typedef int(*pt2Func)(int, int);`
  - `pt2Func` is a pointer to a function that takes 2 int arguments and returns an int

Macros

- **C Preprocessor looks at macros in the preprocessing step of compilation**
- **Use `#define` to avoid magic numbers:**
  - `#define TRIALS 100`
- **Function like macros – short and heavily used code snippets**
  - `#define GET_BYTE_ONE(x) ((x) & 0xff)`
  - `#define GET_BYTE_TWO(x) ( ( (x) >> 8) & 0xff)`
- **Also look at inline functions (example prototype):**
  - `inline int fun(int a, int b)`
  - Requests compiler to insert assembly of max wherever a call to max is made
- **Both useful for malloc**

Debugging

- **Using the `DEBUG` flag:**

  -
    ```
    #define DEBUG
     . . .
    #ifdef DEBUG
     . . . // debugging print statements, etc.
    #endif
    ```
- **Compiling (if you want to debug):**
  - `gcc -DDEBUG foo.c -o foo`
- **Using `assert`**
  - `assert(posvar > 0);`
  - `man 3 assert`
- **Compiling (if you want to turn off asserts):**
  - `gcc -DNDEBUG foo.c -o foo`

Other stuff

- **Usage messages**
  - Putting this in is a good habit – allows you to add features while keeping the user up to date
  - `man -h`
- **`fopen/fclose`**
  - Always error check!
- **Malloc**
  - Error check
  - Free everything you allocate
- **Global variables**
  - Namespace pollution
  - If you must, make them private:
    - `static int foo;`

Today

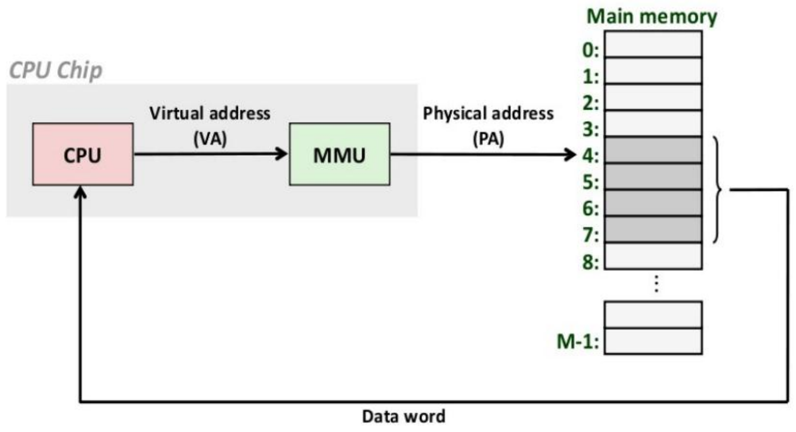
- Shell Lab
- Malloc Lab
- C Primer
- **Virtual Memory**

Virtual Memory Abstraction

- **Virtual memory is layer of indirection between processor and physical memory providing:**
  - Caching
    - Memory treated as cache for much larger disk
  - Memory management
    - Uniform address space eases allocation, linking, and loading
  - Memory protection
    - Prevent processes from interfering with each other by setting permission bits

Virtual Memory Implementation

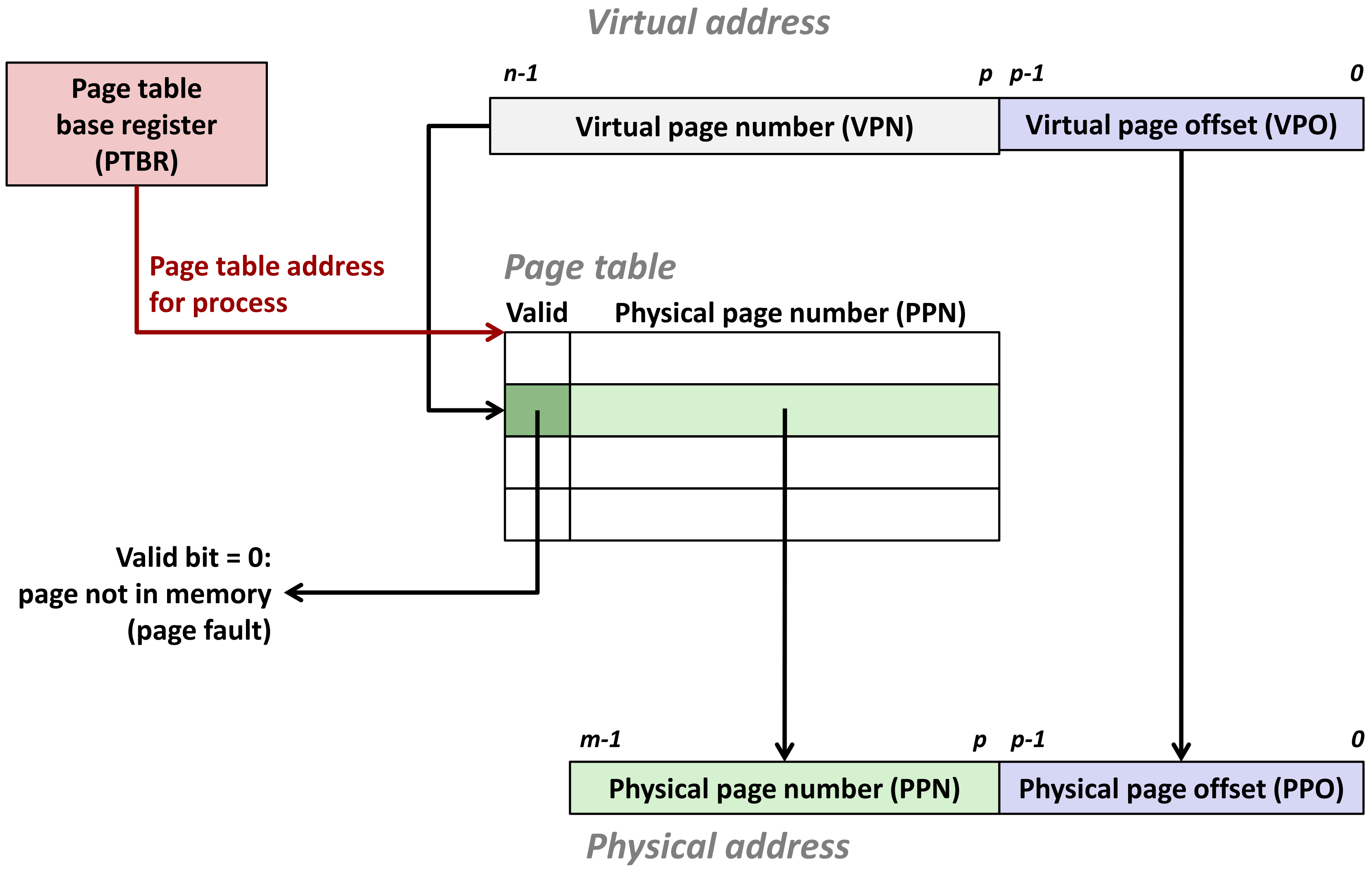
- **Virtual memory implemented by combination of hardware and software**
  - Operating system creates page tables
    - Page table is array of Page Table Entries (PTEs) that map virtual pages to physical pages
  - Hardware Memory Management Unit (MMU) performs address translation



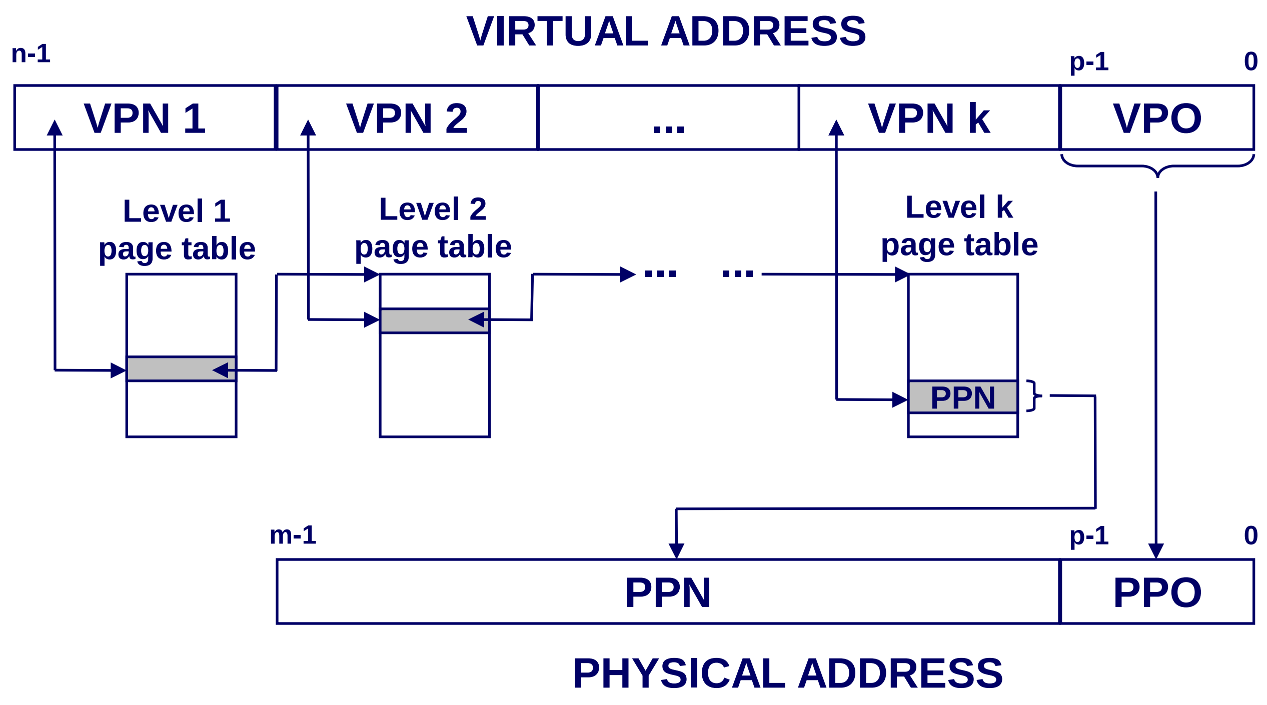
Address Translation and Lookup

- **On memory access (e.g., mov 0xdeadbeef, %eax)**
  - CPU sends virtual address to MMU
  - MMU uses virtual address to index into in-memory page tables
  - Cache/memory returns PTE to MMU
  - MMU constructs physical address and sends to mem/cache
  - Cache/memory returns requested data word to CPU

Recall: Address Translation With a Page Table


Virtual address
n-1
p
p-1
0
Virtual page number (VPN)
Virtual page offset (VPO)
Page table base register (PTBR)
Page table address for process
Page table
Valid
Physical page number (PPN)
Valid bit = 0: page not in memory (page fault)
m-1
p
p-1
0
Physical page number (PPN)
Physical page offset (PPO)
Physical address

Translating with a k-level Page Table



x86 Example Setup

- **Page size 4KB (2^12 Bytes)**
- **Addresses: 32 bits (12 bit VPO, 20 bit VPN)**
- **Consider a one-level page table with:**
  - Base address: 0x01000000
  - 4-byte PTEs
    - 4KB aligned (i.e., lowest 12 bits are zero)
    - Lowest 3 bits used as permissions
      - Bit 0: Present?
      - Bit 1: Writeable?
      - Bit 2: UserAccessible?
- **How big overall?**
  - 2^20 indicies, so 4MB

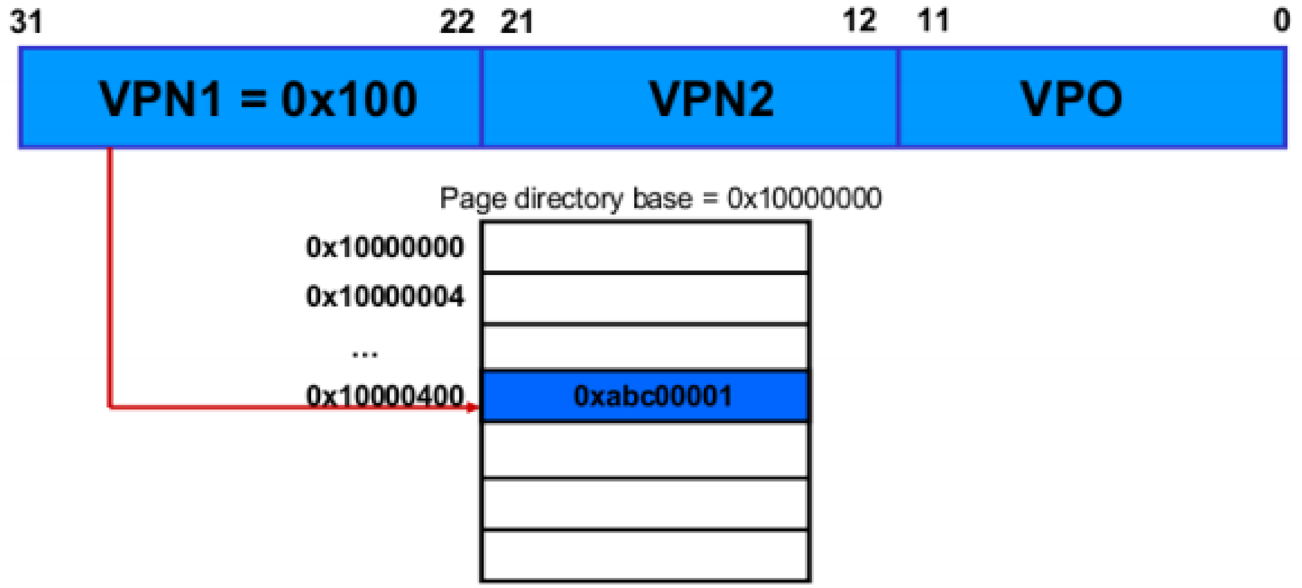
Example

- Given the setup from the previous slide, what are the VPN (index), PPO, and VPO of address: 0xdeadbeef?

Example

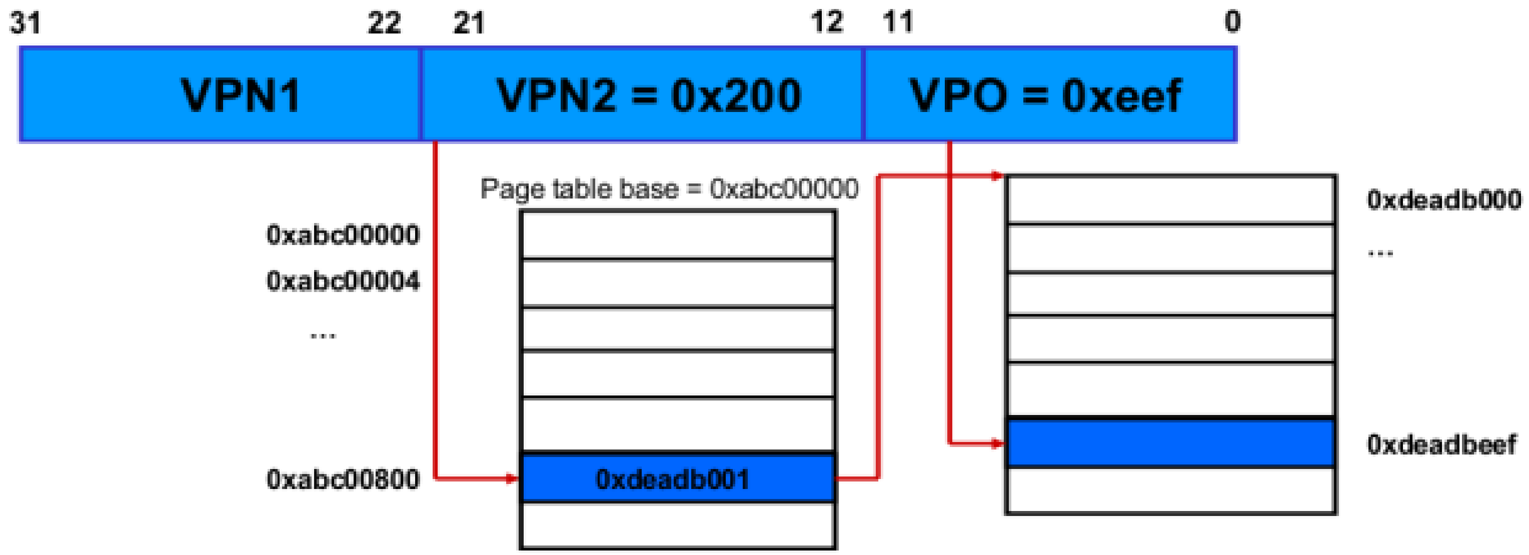
- **Answers:**
  - VPN (index) = 0xdeadb (1101 1110 1010 1101 1011)
  - VPO = PPO = 0xeef
- **Consider a page table entry in our example PT:**
  - Location of PTE = base + (size * index)
    - 0x0137ab6c = base + 4 * index
  - PTE: 0x98765007
  - Physical address: 0x98765eef

Example: 2 level page table



Use the first VPN to index into the page directory. This gives the address of the start of the page table.

2-level page table – cont'd



Use the second VPN to index into the page table. This gives the address of the start of the page frame. Add the offset to obtain the location in physical memory.

Questions?