Virtual Memory

Introduction to Computer Systems

Recitation: March 22nd, 2010

Reminders and Agenda

- Reminders:
  - Shell lab due March 23rd (tomorrow)
    - Stuck? Need help?
      - Come to office hours: Sun-Thurs, 6-9pm Wean 5207 or
      - Email staff list
  - Malloc lab
    - Out tomorrow, due April 13th
  - Exam 2 on April 6th
- Agenda
  - Some shell lab hints
  - Today's topic: virtual memory
  - Style guidelines

Shell Lab due Tomorrow

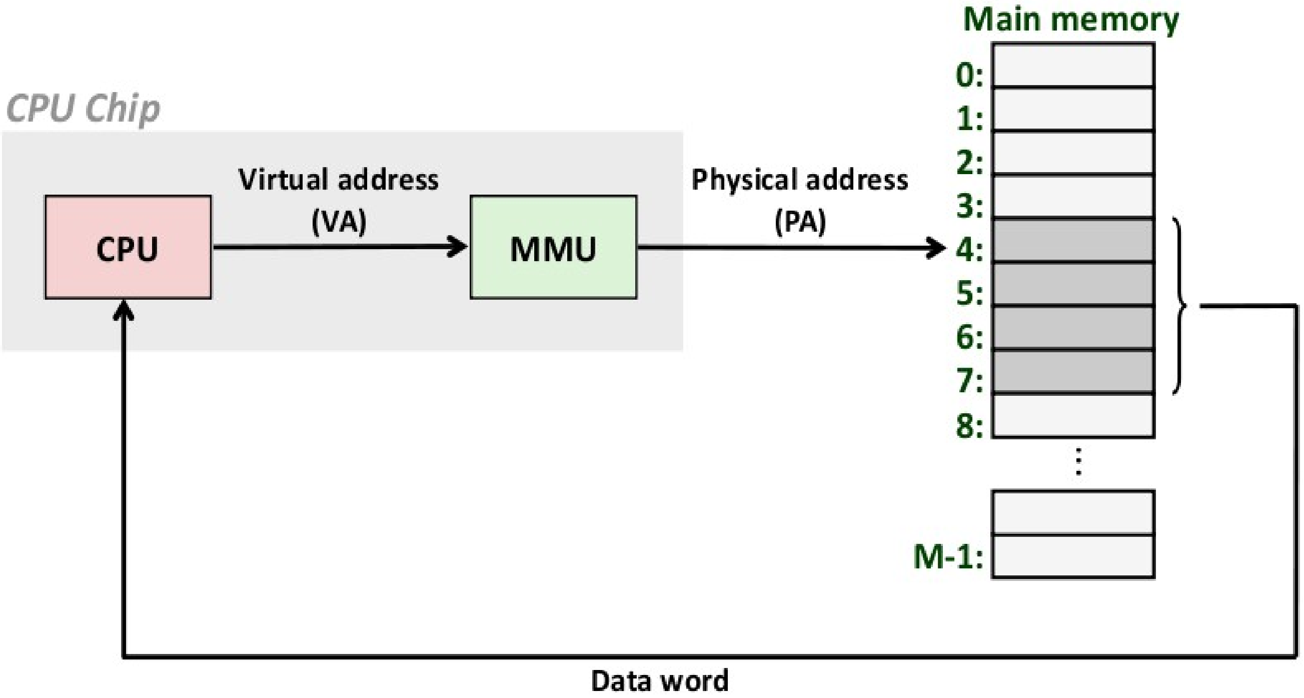
- Block signals when modifying data structures
- Write reentrant code
  - Reentrant code does not reference any shared data when called by multiple threads
- Avoiding race conditions
  - Unexpected behavior based on thread interleavings
  - Eliminating race conditions requires considering (and handling) every action that could happen at each point in your program
- I/O redirection
  - `int dup2(int oldfd, int newfd)` - function duplicates descriptor oldfd, assigns it to descriptor newfd, and returns newfd

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*



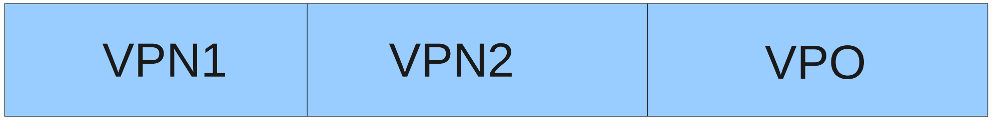
Address Translation and Lookup

- 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

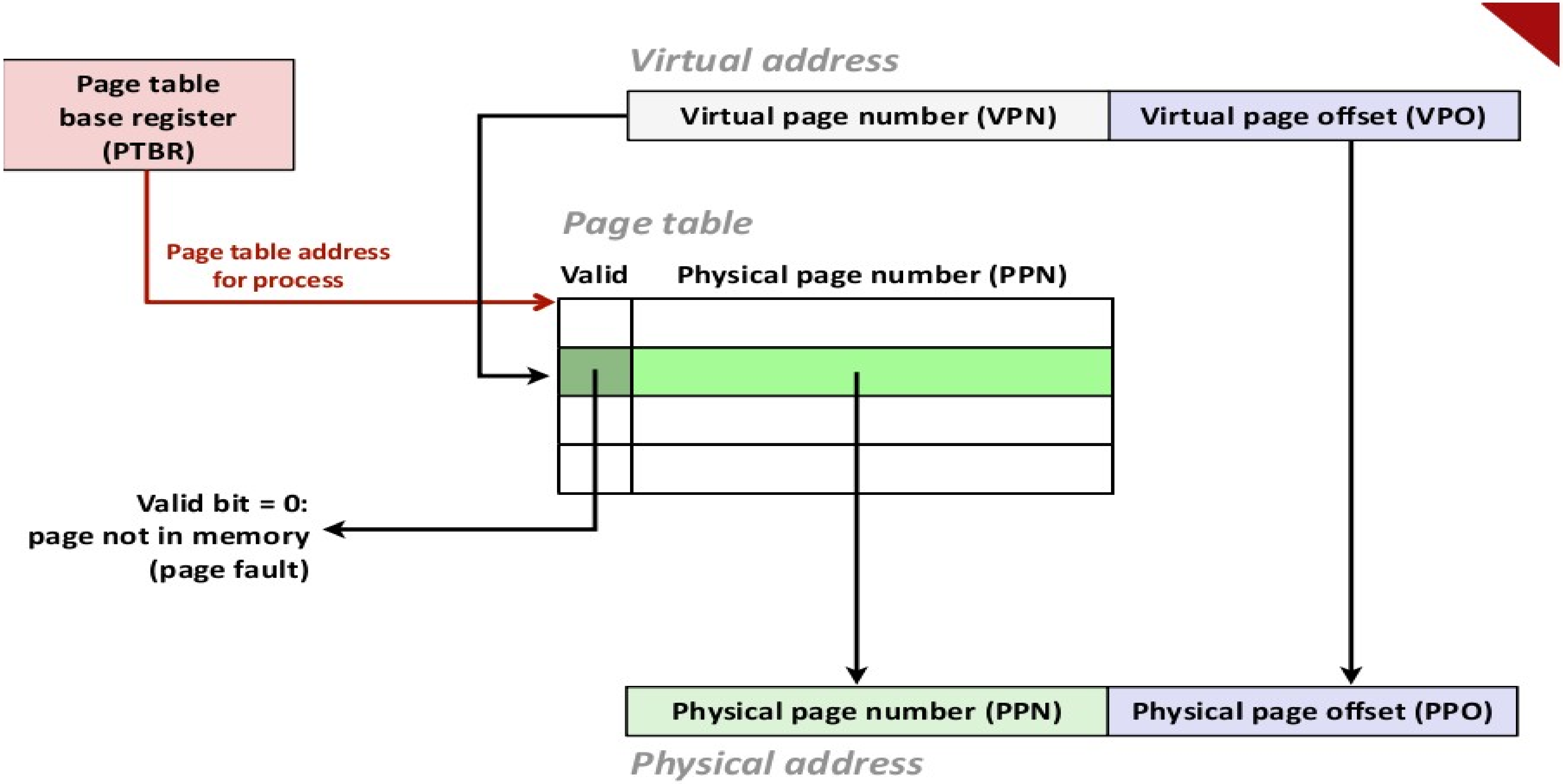
Virtual Address as Index & Offset

- Virtual address encodes:
  - Virtual Page Number (VPN) – index into page table
  - Virtual Page Offset – Byte offset from start of page frame

- In multi-level page tables VA encodes:
  - Multiple VPNs: Index into page table/directory
  - Virtual Page Offset – same as above



One Level Page Table



- For x86-32, %cr3 is page table base register
- Addr length = 32 bits with 20 bit VPNs, 12 bit VPOs

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

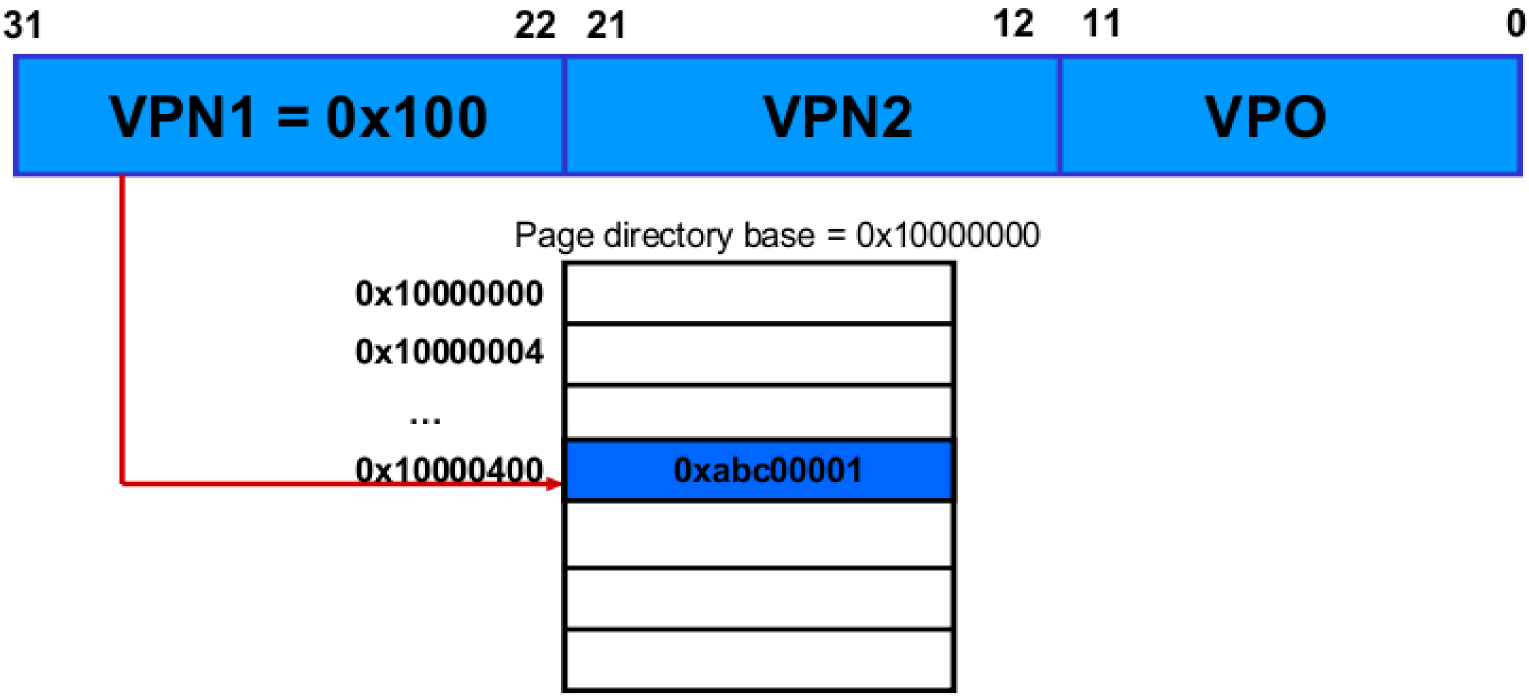
Example

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

Example

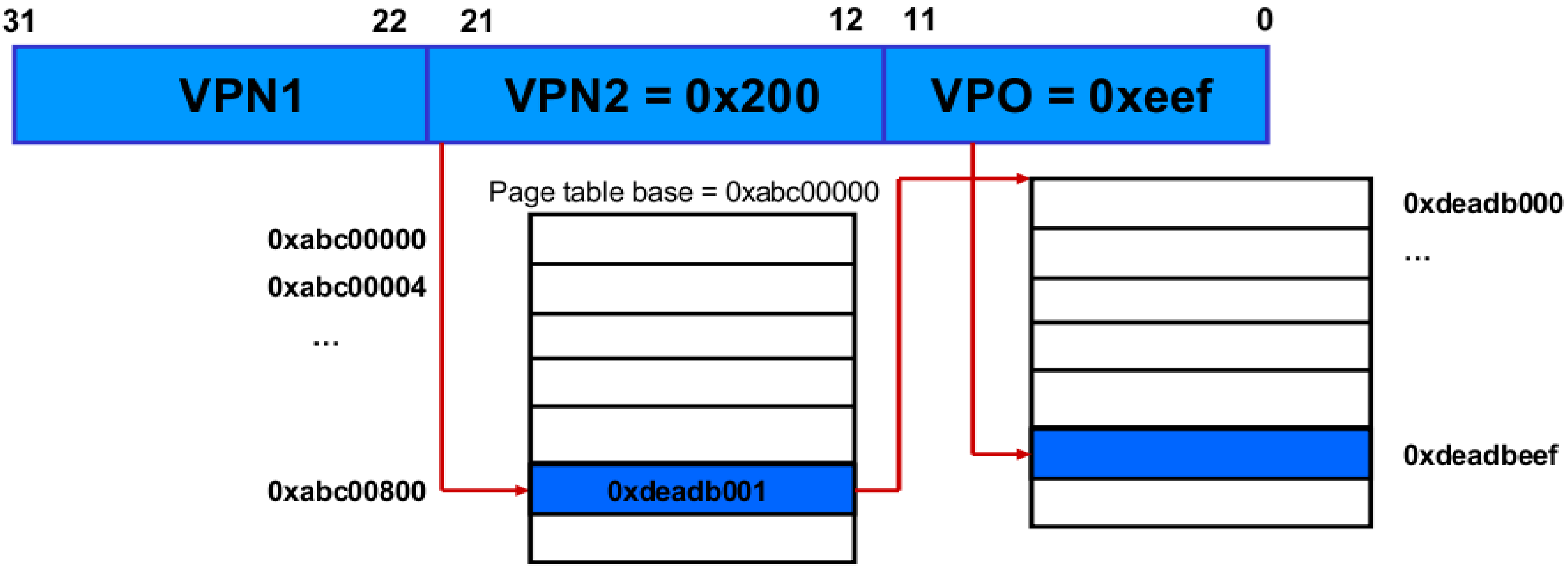
- Answers:
  - VPN (index) = 0xdeadb (1101 1110 1010 1101 1011)
  - VPO = PPO = 0xeef
- Consider a page table entry in our example PT:
  - Location of PTE = base + (size * index)
    - 0x0137ab6c = base + 4 * index
  - PTE: 0x98765007
  - Physical address: 0x98765eef

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.

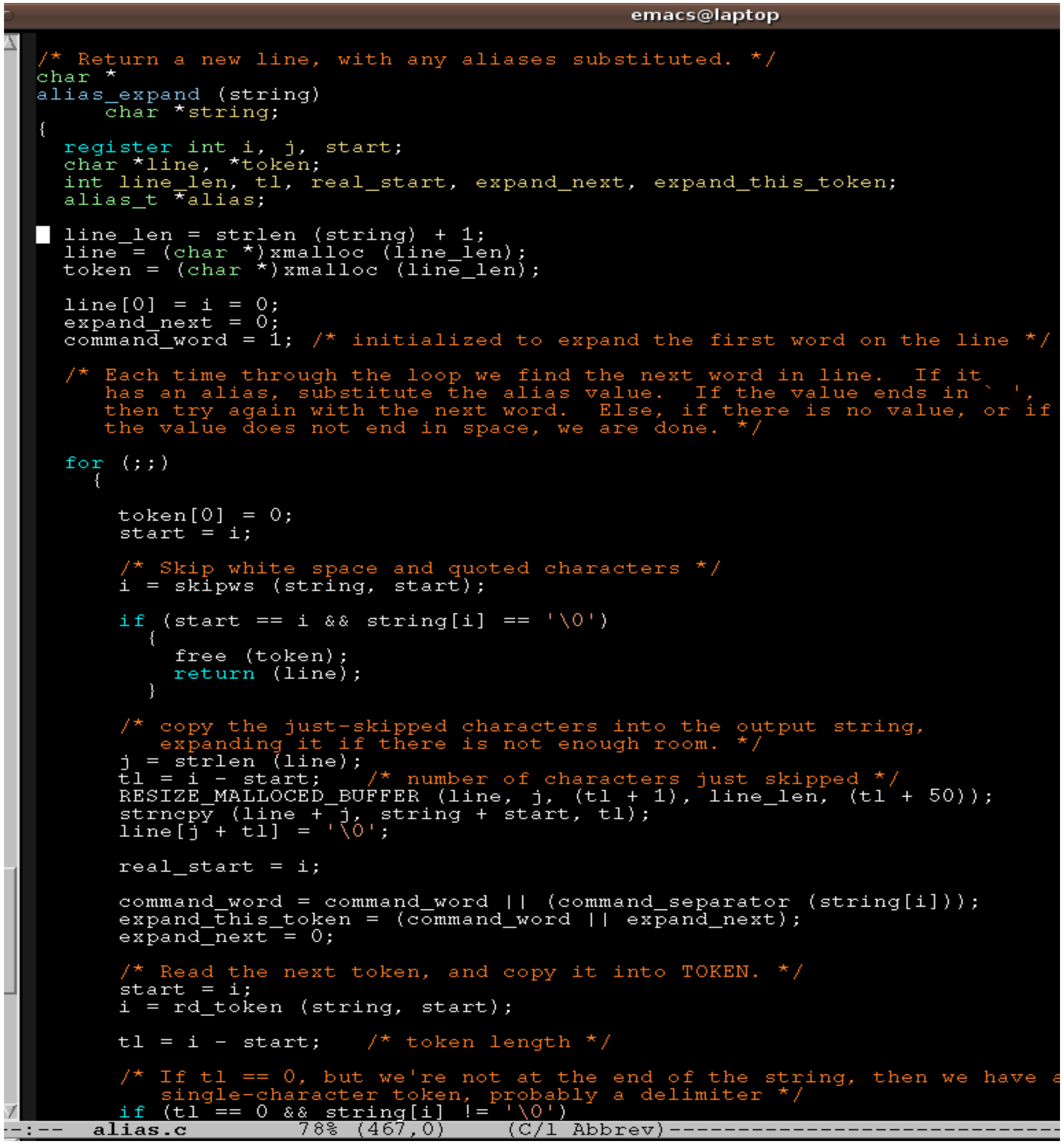
2-level page table – cont'd



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.

Style Guidelines

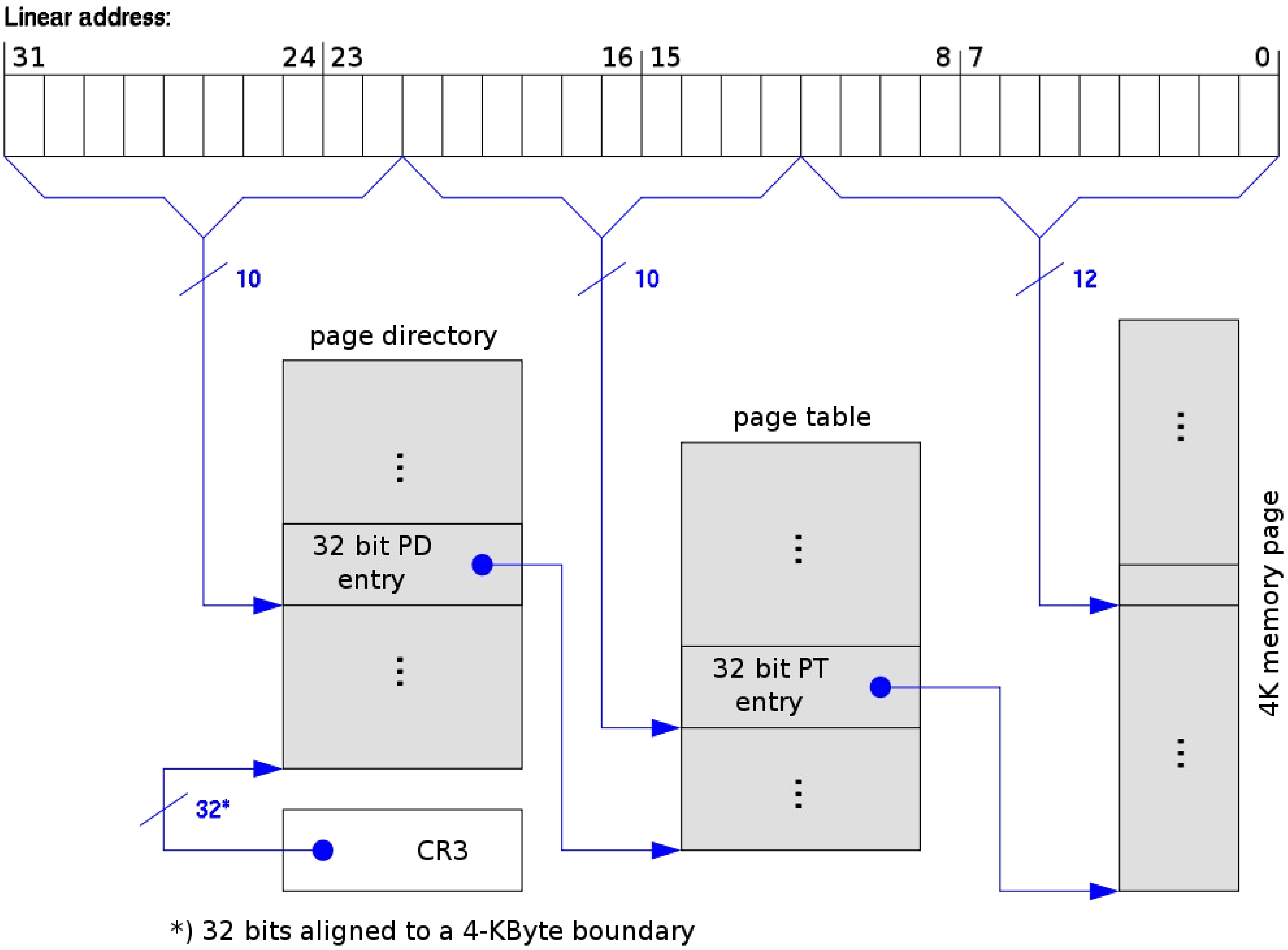
- Avoid lines longer than 80 characters
- Use meaningful variable names
  - Instead of `int x`, write `int counter`
- Modularize your code
  - Avoid duplicating code, use functions instead
- Indent code blocks, be consistent!
  - `> man indent`
- Add error checking code
  - Handle all possible errors
- Eliminate dead code
  - Code on branch that is always false
- Do not inline magic numbers
  - Instead #define them



```
/* Return a new line, with any aliases substituted. */
char *
alias_expand (string)
     char *string;
{
  register int i, j, start;
  char *line, *token;
  int line_len, tl, real_start, expand_next, expand_this_token;
  alias_t *alias;

  line_len = strlen (string) + 1;
  line = (char *)xmalloc (line_len);
  token = (char *)xmalloc (line_len);

  line[0] = i = 0;
  expand_next = 0;
  command_word = 1; /* initialized to expand the first word on the line */

  /* Each time through the loop we find the next word in line.  If it
     has an alias, substitute the alias value.  If the value ends in ` ',
     then try again with the next word.  Else, if there is no value, or if
     the value does not end in space, we are done. */

  for (;;)
    {

      token[0] = 0;
      start = i;

      /* Skip white space and quoted characters */
      i = skipws (string, start);

      if (start == i && string[i] == '\0')
        {
          free (token);
          return (line);
        }

      /* copy the just-skipped characters into the output string,
         expanding it if there is not enough room. */
      j = strlen (line);
      tl = i - start;    /* number of characters just skipped */
      RESIZE_MALLOCED_BUFFER (line, j, (tl + 1), line_len, (tl + 50));
      strncpy (line + j, string + start, tl);
      line[j + tl] = '\0';

      real_start = i;

      command_word = command_word || (command_separator (string[i]));
      expand_this_token = (command_word || expand_next);
      expand_next = 0;

      /* Read the next token, and copy it into TOKEN. */
      start = i;
      i = rd_token (string, start);

      tl = i - start;    /* token length */

      /* If tl == 0, but we're not at the end of the string, then we have
         single-character token, probably a delimiter */
      if (tl == 0 && string[i] != '\0')
--:--  alias.c        78% (467,0)     (C/l Abbrev)----------------------------
```

Questions?

- Shell lab due March 23rd (tomorrow)
  - Stuck? Need help?
    - Office hours: Sun-Thurs, 6-9pm Wean 5207
- Malloc lab
  - Out tomorrow, due April 13th
- Exam 2 on April 6th



** From Wikipedia, GNU License