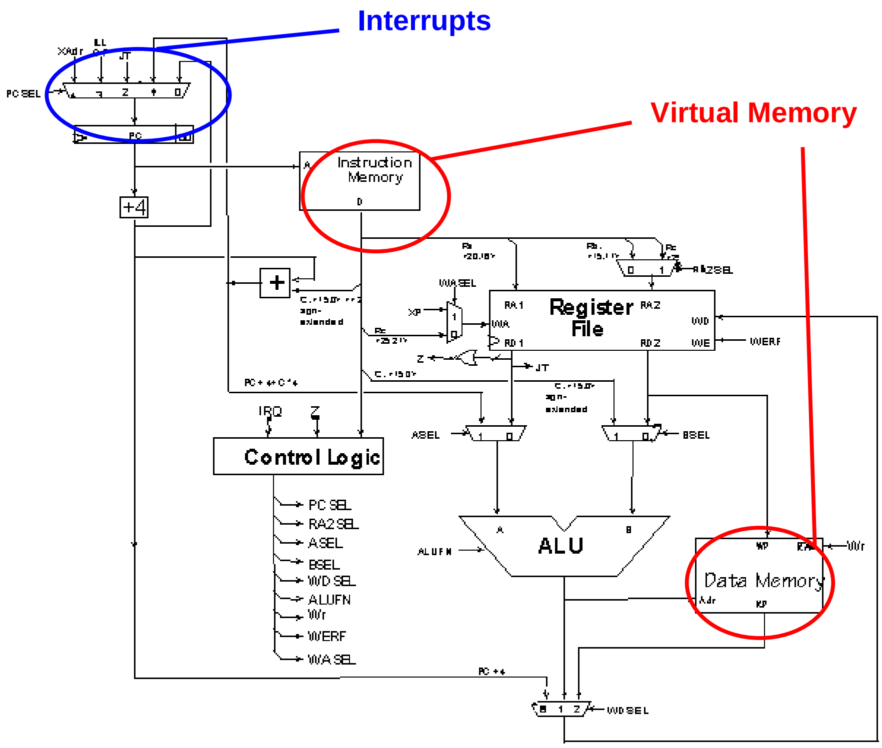
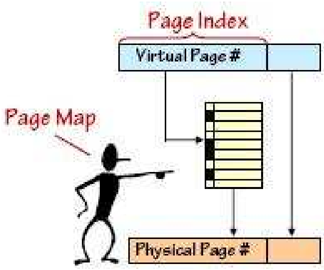
Daniel Leeds, April 26, 2006, Quiz Review




Understand this table:

| | OP | OPC | LD | ST | JMP | BEQ | BNE | LDR | Illop | IRQ |
|---|---|---|---|---|---|---|---|---|---|---|
| ALUFN | F(op) | F(op) | "+" | "+" | — | — | — | "A" | — | — |
| WERF | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| BSEL | 0 | 1 | 1 | 1 | — | — | — | — | — | — |
| WDSEL | 1 | 1 | 2 | — | 0 | 0 | 0 | 2 | 0 | 0 |
| WR | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| RA2SEL | 0 | — | — | 1 | — | — | — | — | — | — |
| PCSEL | 0 | 0 | 0 | 0 | 2 | Z ? 1 : 0 | Z ? 0 : 1 | 0 | 3 | 4 |
| ASEL | 0 | 0 | 0 | 0 | — | — | — | 1 | — | — |
| WASEL | 0 | 0 | 0 | — | 0 | 0 | 0 | 0 | 1 | 1 |




In Page Map (aka, Page Table, PTbl)
One entry per virtual page
Resident bit (R, also called "valid" bit) = 1 if page in physical memory
DIRTY bit = 1 if page contents have been changed since loaded into physical memory

Arithmetic

| | |
|---|---|
| $2^p$ | bytes per physical page |
| (v+p) | bits in virtual address |
| (m+p) | bits in physical address |
| $2^v$ | number of virtual pages |
| $2^m$ | number of physical pages |
| $(m+2)2^v$ | bits in page map |

Operating system issues:
"OS Kernel" is a special, privileged process that oversees the other processes and handles real I/O devices
Each process has its own Process Control Block (PCB), which encapsulates its state
Scheduler() switches among user processes

```
struct MState {
  int Regs[31];                /* saved state of user's registers */
} User;

int N = 42;                    /* number of processes to schedule */
int Cur = 0;                   /* number of "active" process */

struct PCB {
  struct MState State;         /* processor state */
  Context PageMap;             /* VM map for process */
  int DPYNum;                  /* console/keyboard number */
} ProcTbl[N];                  /* one per process */

Scheduler() {
  ProcTbl[Cur].State = User;   /* save current user state */
  Cur = (Cur + 1)%N;           /* increment modulo N */
  User = ProcTbl[Cur].State;   /* make another process the current one
*/
}
```

**Fine print:**
Quiz is closed-book, no calculators; covers Building the Beta, Caches, Virtual Memory, OS issues (Virtual Machines) -- up to L18 (Virtual Machines)/R18 (this recitation)

**Practice, practice, practice:**
Follow "Previous terms" link from http://6004.csail.mit.edu, pick a semester (the more recent, the better), click on the "Announcements" page for the semester, and find the PDF for Quiz 4 and 5 solutions. **NOTE:** We covered material in different order this year, skipped over some subjects, and focused more on others. Do not worry about set associative caches; instead, worry about SVC's and questions like Problem 3 of today's tutorial.

**Another perspective on the material – Margaret Chong's Handbook:**
Follow "Handouts" link from http://6004.csail.mit.edu, click on handbook link near the bottom of the page.

**Handouts**
Make sure you understand as much as you can of the Unpipelined Beta diagram and Control Logic chart (provided on page 1).

Virtual Memory, revisited

Problem 1, Part G:

The table to the left shows the first 8 entries in the page map. Recall that the valid bit is 1 if the page is resident in physical memory and 0 if the page is on disk or hasn't been allocated.

| Virtual page | Valid bit | Physical page |
|---|---|---|
| 0 | 0 | 7 |
| 1 | 1 | 9 |
| 2 | 0 | 3 |
| 3 | 1 | 2 |
| 4 | 1 | 5 |
| 5 | 0 | 5 |
| 6 | 0 | 4 |
| 7 | 1 | 1 |

If there are 1024 ($2^{10}$) bytes per page, what is the physical address corresponding to the decimal virtual address 3956?

## OS issues

Problem 2:

```
. = VEC_RESET
        BR(I_Reset)  | on Reset (start-up)
. = VEC_II
        BR(I_IllOp)  | on Illegal Instruction
                     | (eg SVC)
. = VEC_CLK
        BR(I_Clk)    | On clock interrupt
. = VEC_KBD
        BR(I_Kbd)    | on Keyboard interrupt
. = VEC_MOUSE
        BR(I_BadInt) | on mouse interrupt
...
I_Reset:
        CMOVE(P0Stack, SP)
        CMOVE(P0Start, XP)
        JMP(XP)
```

Problem 1:

A

```
ReadKey_h() {
  int kdbnum = ProcTbl[Cur].DPYNum;
  while (BufferEmpty(kdbnum)) {
    /* busy wait loop */
  }
  User.Regs[0] =
ReadInputBuffer(kdbnum);
}
```

C

```
ReadKey_h() {
  int kdbnum = ProcTbl[Cur].DPYNum;
  if (BufferEmpty(kdbnum))
    User.Regs[XP] = User.Regs[XP] - 4;
  else
    User.Regs[0]=ReadInputBuffer(kdbnum);
}
```

D

```
ReadKey_h() {
  int kdbnum = ProcTbl[Cur].DPYNum;
  if (BufferEmpty(kdbnum)) {
    User.Regs[XP] = User.Regs[XP] - 4;
    Scheduler();
  } else
    User.Regs[0] = ReadInputBuffer(kdbnum);
}
```