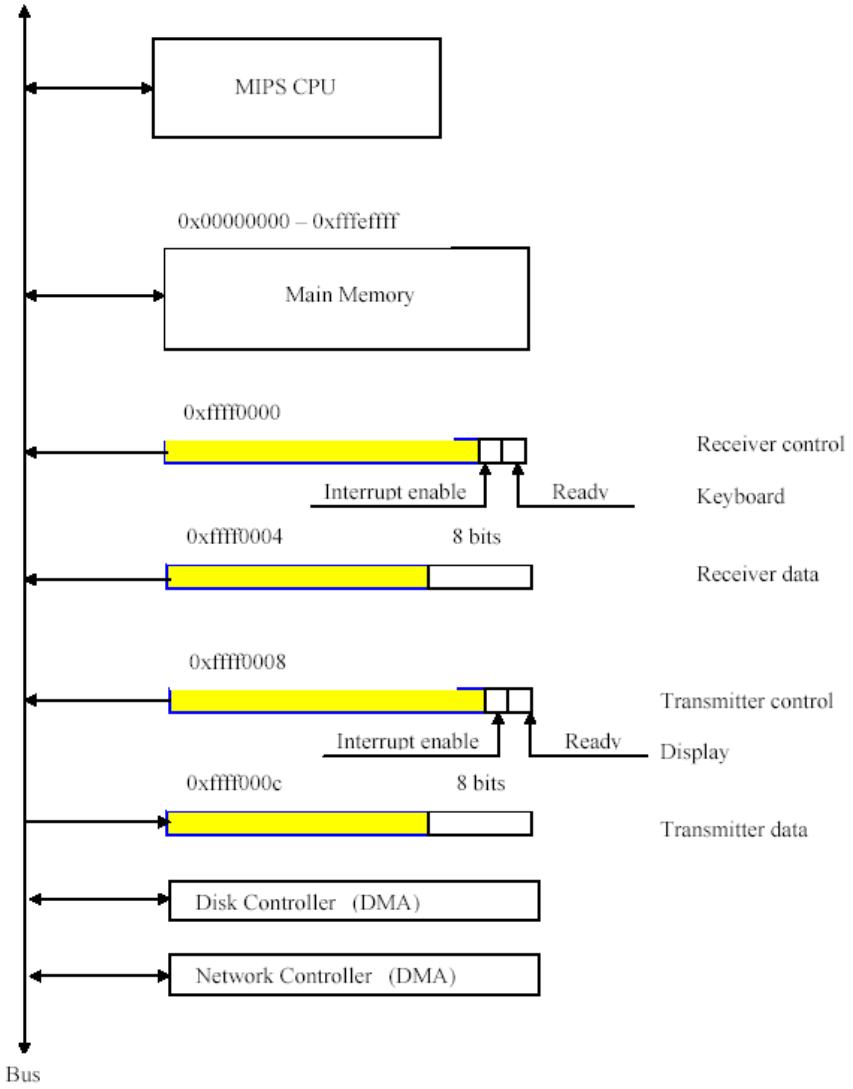
I/O Polling

- Direct program controlled I/O uses polling
  - Device status registers are repeatedly checked
  - CPU is kept busy by the polling operations
- Each device is assigned one or more registers
  - Status register
  - Control register
  - Data register
- Simple devices have fewer registers
  - Keyboard
  - Console display
- Complex devices have more registers

I/O Device Registers

Example memory-mapped system.



I/O Polling Example

Detecting key presses using polling:

```
# device register base address = 0xFFFF0000
          lui    $t3,0xFFFF    # KB status address
poll_CR:  lw     $t1,0($t3)    # read status register
          andi   $t1,$t1,1     # does LSB=1?
          beqz   $t1,poll_CR   # keep checking if not
          lw     $t0,4($t3)    # else read character into $t0
```

Time between keystrokes can be considerable

The status bit is automatically cleared when the data register is read

I/O Polling Example

Output to the console display using polling:

```
          lui   $t3,0xFFFF    # device reg base address
          li    $t0,'>'       # ASCII code for output char
Poll_XR:  lw    $t1,8($t3)    # Console Cntrl reg offset=8
          andi  $t1,$t1,1     # does LSB=1?
          beqz  $t1,poll_XR   # keep checking if not
          sw    $t0,12($t3)   # display the character
```

Status register is polled to see if the console is ready
Ready bit goes to 0 when data register is written
Ready bit goes back to 1 once the character has been displayed

Polling keeps the CPU busy
Using interrupts instead allows the CPU to do other useful work