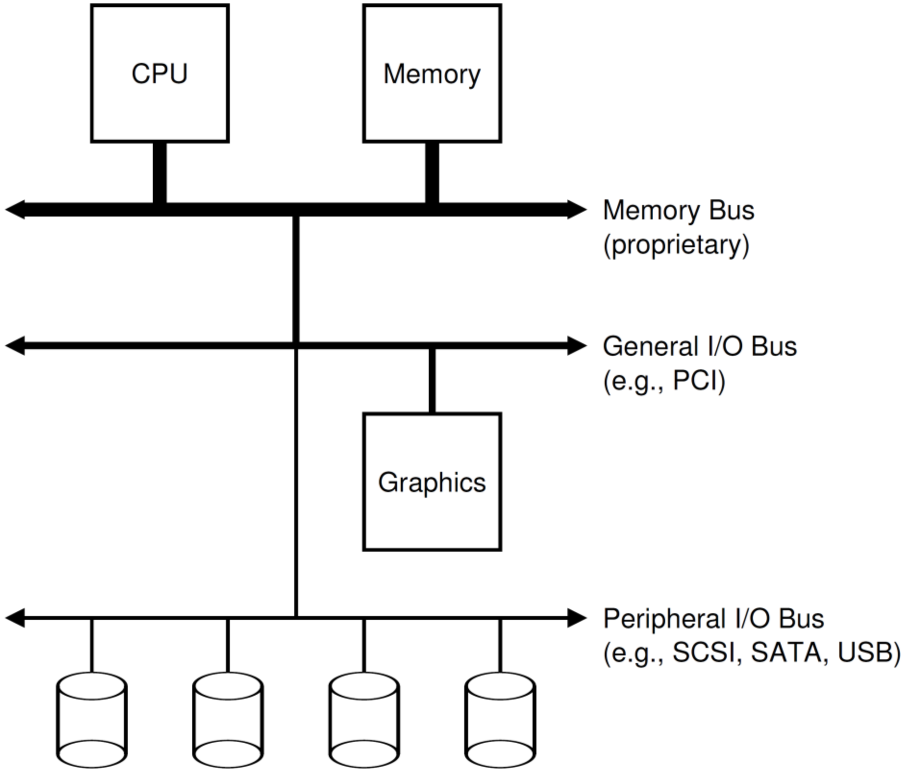
ITP 30002 Operating System

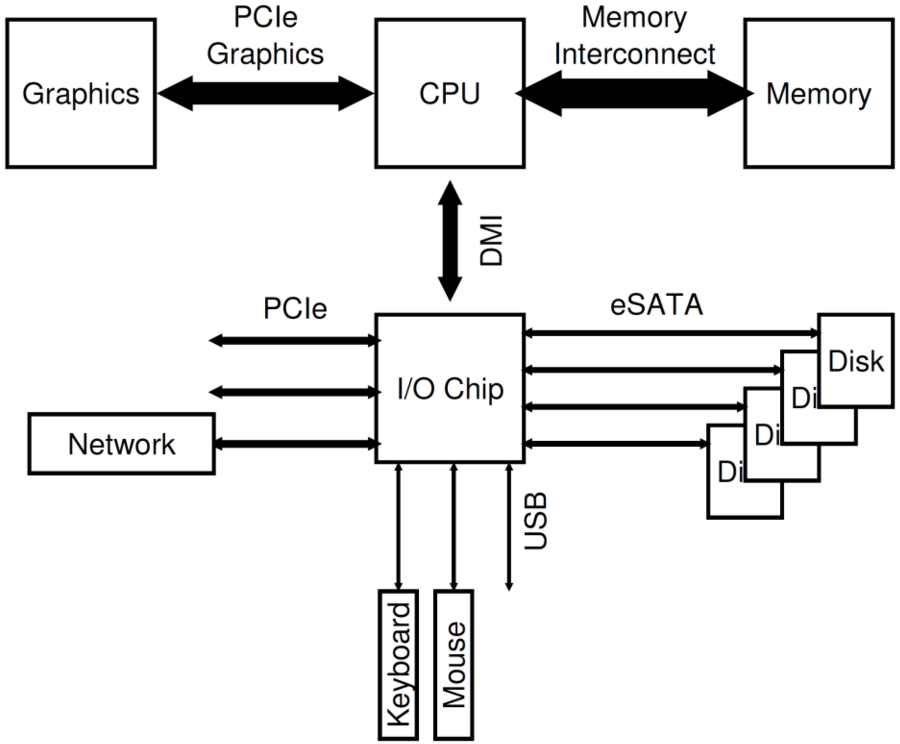
Input/Ouput Devices

Chapters 36

Shin Hong

How I/O devices are attached to system architecture





- nearer memory bus, fasterer devices are connected

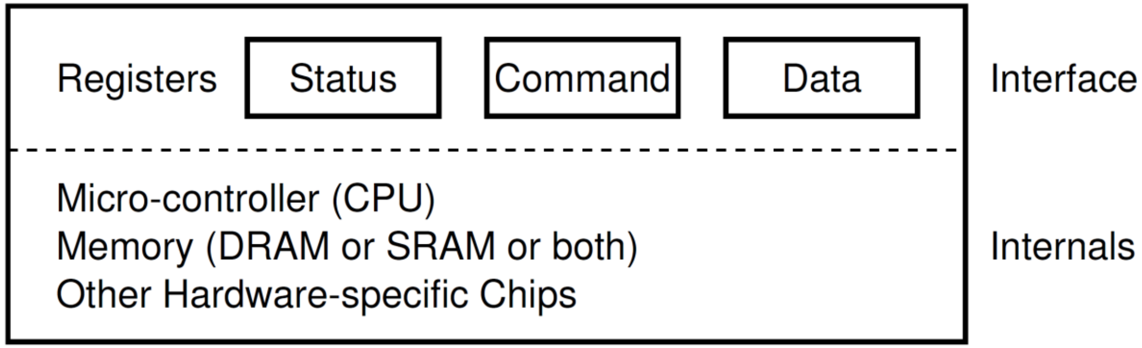
- Memory bus and graphic Bus
- I/O chip
  - Direct Media Interface

Method of Device Interaction

- via I/O instructions
  - CPU has previlaged instructions to send data to a specific device addressed by a port number
  - OS uses these instructions to send data and command to each device

- via memory instructions
  - Device registeres are mapped to specific memory addresses
  - OS uses memory read and write instructions to operate on the device

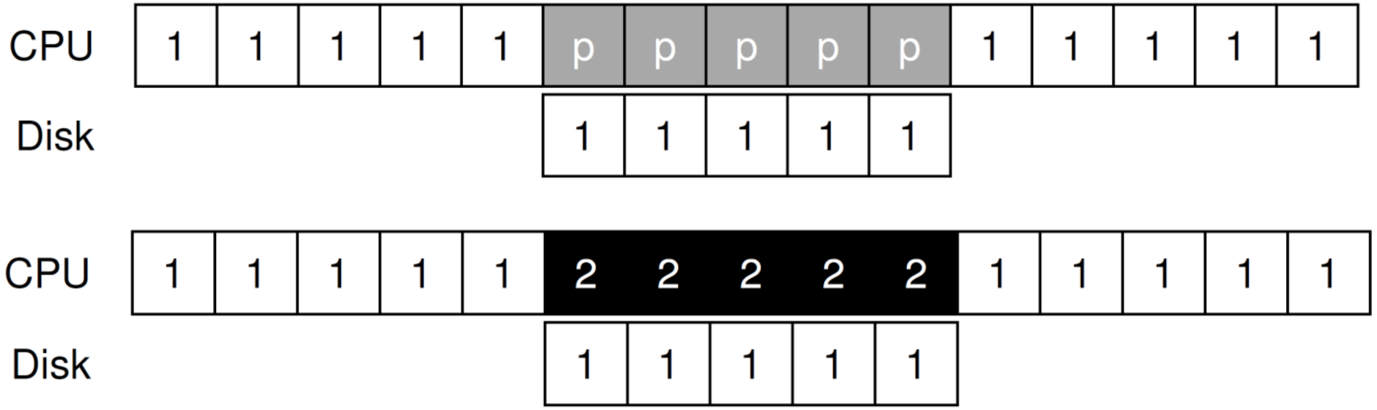
Interaction with Hardware Device - Polling



```
While (STATUS == BUSY)
    ; // wait until device is not busy        polling
Write data to DATA register
Write command to COMMAND register
    (starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request    polling
```

Interaction with Hardware Device - Interrupt

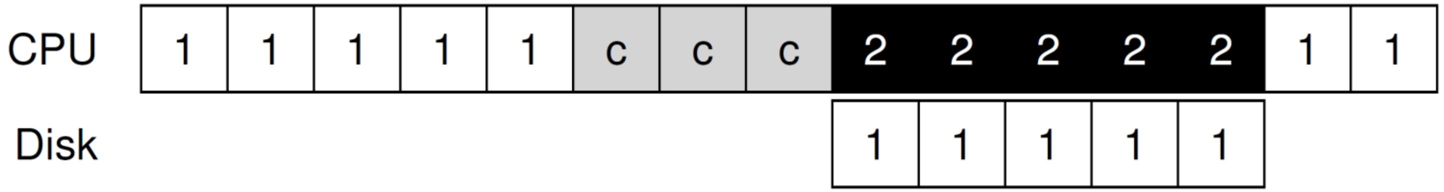
- polling vs. interrupt


CPU 1 1 1 1 1 p p p p p 1 1 1 1 1
Disk 1 1 1 1 1
CPU 1 1 1 1 1 2 2 2 2 2 1 1 1 1 1
Disk 1 1 1 1 1

- efficient use of interrupt
  - two-phase approach: conduct polling for a short time period first, and then use interrupt
  - coalescing: merge multiple messages and deliver them once with a single interrupt to limit the number of interrupts in a time unit

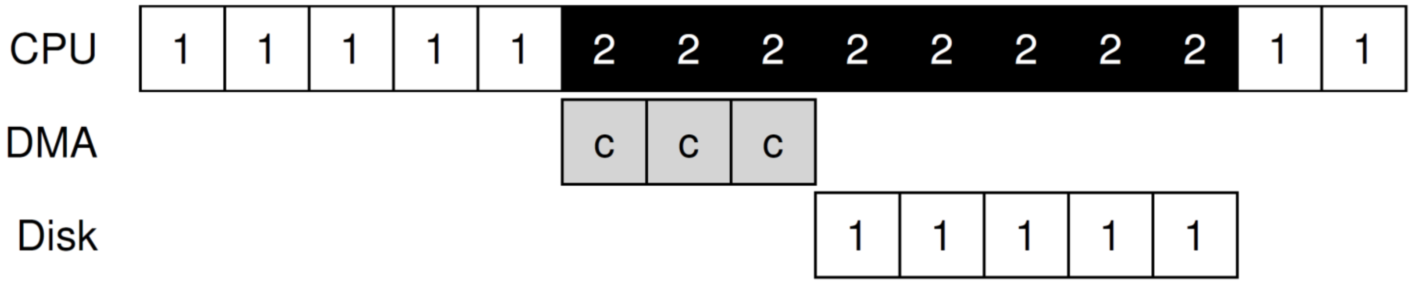
Moving Data

- Programmed I/O
  - CPU moves each value one-by-one to the data register
  - Device register access time is far longer than memory access time
  - Example. writing data to disk



CPU	1	1	1	1	1	c	c	c	2	2	2	2	2	1	1
Disk									1	1	1	1	1		

- Direct Memory Access (DMA)
  - CPU commands DMA to transfer data from memory to device registers
  - DMA raises an interrupt when it accomplishes the data transfer
  - Example. writing data to disk

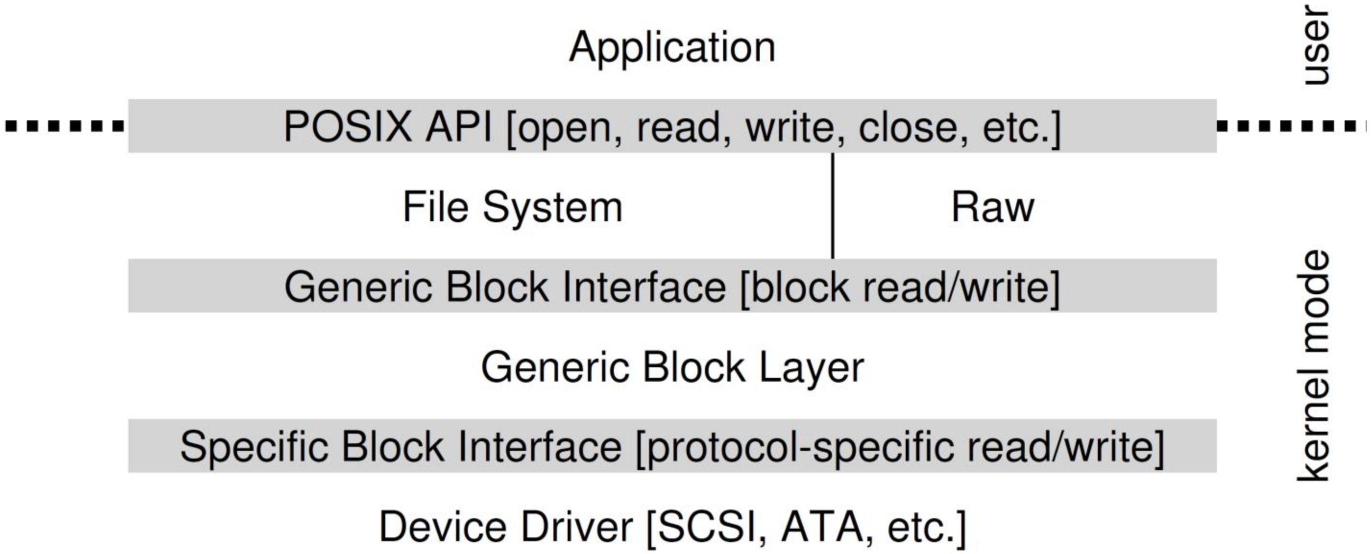


CPU	1	1	1	1	1	2	2	2	2	2	2	2	2	1	1
DMA						c	c	c							
Disk									1	1	1	1	1		

Device Driver

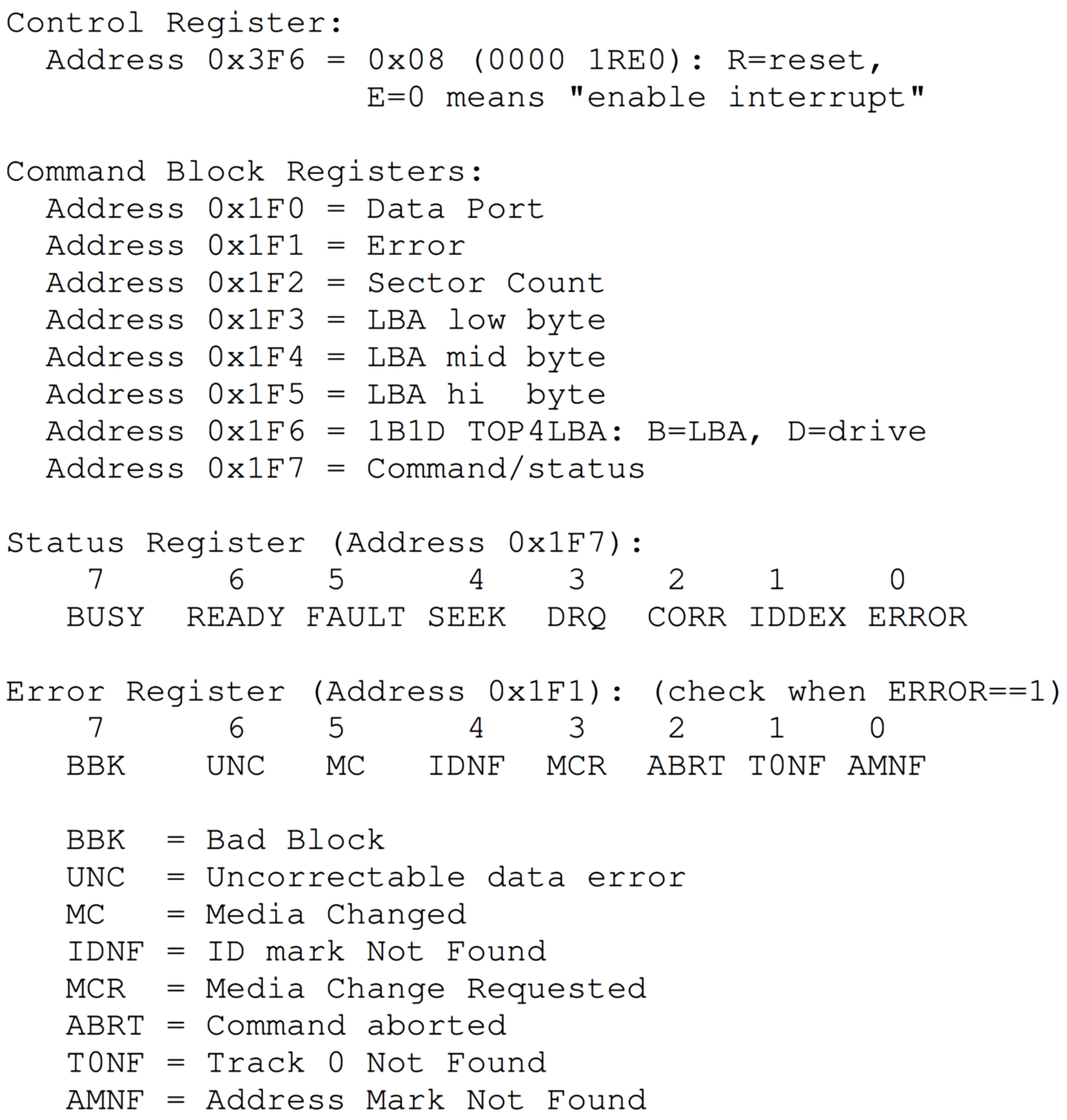
- A device driver is a kernel module that encapsulates hardware details and provices interface for OS to control the device
    - the devices of the same kind share the same interface
    - for Linux, device driver modules take around 70% of kernel code

- E.g. disk device drivers and file system stack


Application
user
POSIX API [open, read, write, close, etc.]
File System
Raw
Generic Block Interface [block read/write]
Generic Block Layer
Specific Block Interface [protocol-specific read/write]
Device Driver [SCSI, ATA, etc.]
kernel mode

Example – IDE Disk Driver

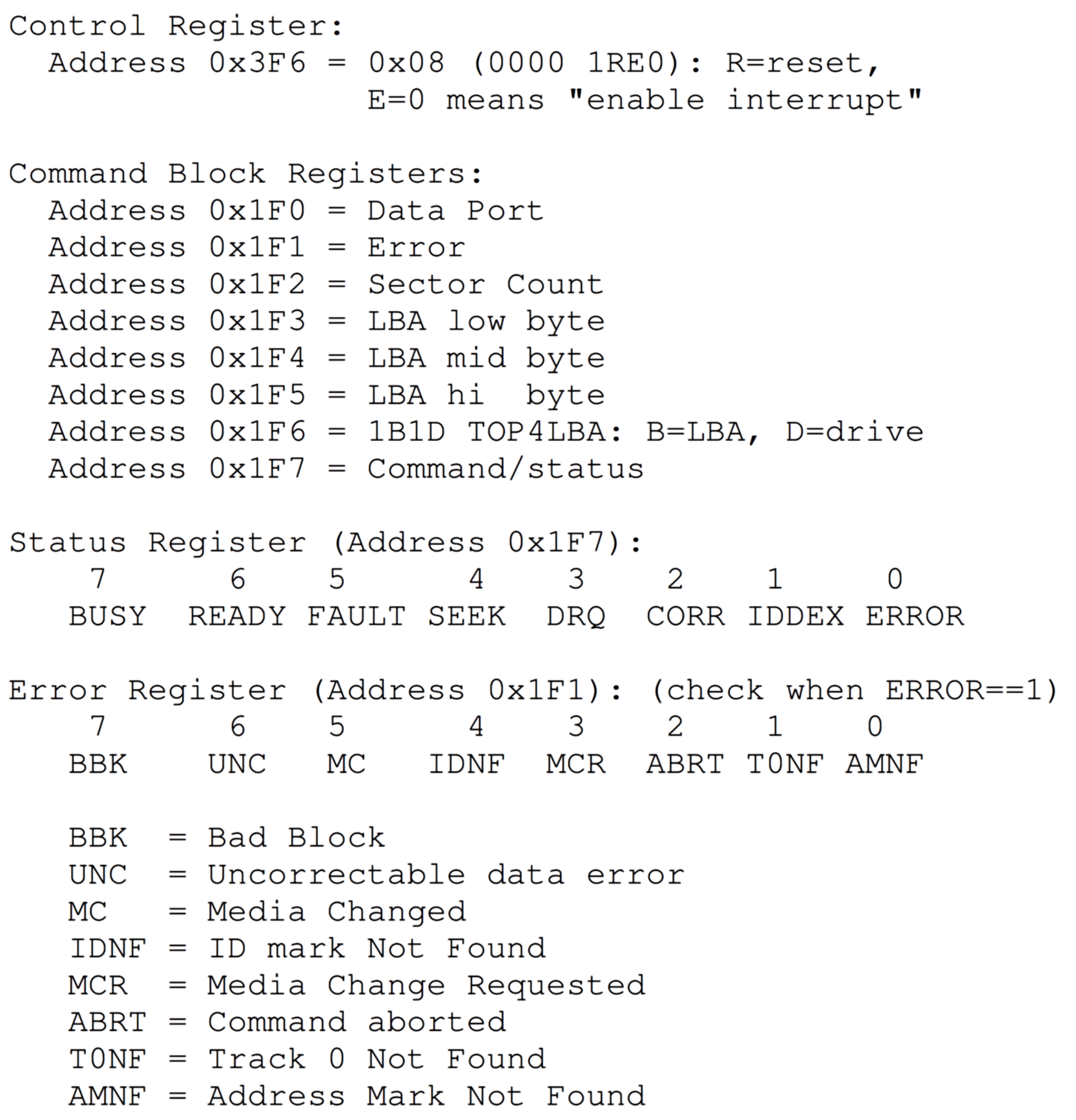
- Four interface registers
  - Control
  - Command block
  - Status
  - Error
- Each register can be read and written by the I/O instructions by its I/O address



```
Control Register:
  Address 0x3F6 = 0x08 (0000 1RE0): R=reset,
                  E=0 means "enable interrupt"

Command Block Registers:
  Address 0x1F0 = Data Port
  Address 0x1F1 = Error
  Address 0x1F2 = Sector Count
  Address 0x1F3 = LBA low byte
  Address 0x1F4 = LBA mid byte
  Address 0x1F5 = LBA hi  byte
  Address 0x1F6 = 1B1D TOP4LBA: B=LBA, D=drive
  Address 0x1F7 = Command/status

Status Register (Address 0x1F7):
    7      6     5      4    3    2    1     0
   BUSY  READY FAULT SEEK  DRQ  CORR IDDEX ERROR

Error Register (Address 0x1F1): (check when ERROR==1)
    7      6     5      4    3    2    1    0
   BBK    UNC   MC    IDNF  MCR  ABRT T0NF AMNF

   BBK  = Bad Block
   UNC  = Uncorrectable data error
   MC   = Media Changed
   IDNF = ID mark Not Found
   MCR  = Media Change Requested
   ABRT = Command aborted
   T0NF = Track 0 Not Found
   AMNF = Address Mark Not Found
```

Example – IDE Disk Driver

- Protocol
  - Wait for device to be ready: wait until Status becomes Ready, not Busy
  - Write Sector Count, Logical Block Address of the sector to access, and Drive number
  - Write Read, or Write to Command block
  - Wait until Status is Ready and DRQ; write data to Data Port
  - Handle an interrupt for each sector transferred
  - After each operation, read Status and if its error bit is on, read Error



```
Control Register:
  Address 0x3F6 = 0x08 (0000 1RE0): R=reset,
                  E=0 means "enable interrupt"

Command Block Registers:
  Address 0x1F0 = Data Port
  Address 0x1F1 = Error
  Address 0x1F2 = Sector Count
  Address 0x1F3 = LBA low byte
  Address 0x1F4 = LBA mid byte
  Address 0x1F5 = LBA hi  byte
  Address 0x1F6 = 1B1D TOP4LBA: B=LBA, D=drive
  Address 0x1F7 = Command/status

Status Register (Address 0x1F7):
    7      6     5      4    3    2    1     0
   BUSY  READY FAULT SEEK  DRQ  CORR IDDEX ERROR

Error Register (Address 0x1F1): (check when ERROR==1)
    7      6     5      4    3    2    1    0
   BBK    UNC   MC    IDNF  MCR  ABRT T0NF AMNF

   BBK  = Bad Block
   UNC  = Uncorrectable data error
   MC   = Media Changed
   IDNF = ID mark Not Found
   MCR  = Media Change Requested
   ABRT = Command aborted
   T0NF = Track 0 Not Found
   AMNF = Address Mark Not Found
```

stem