DMA

Direct Memory Access

Types of I/O Devices

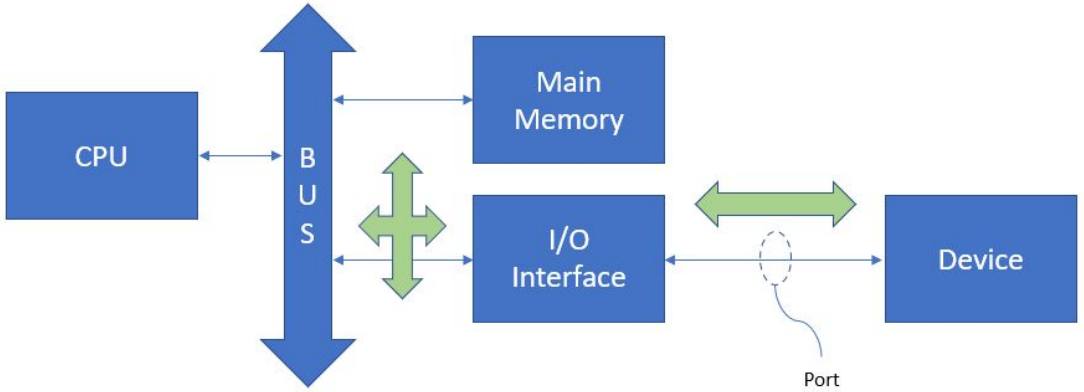
1. Character devices
    - E.g., keyboard, mouse
    - Tend to have lower data rates
    - Values (e.g., bytes) are transferred individually
    - Per-value interrupts are appropriate

Types of I/O Devices

1. Block devices
    - E.g., Drive (mass storage device), video device, PCM sound device
    - Tend to have higher data rates
    - Entire blocks of values are transferred as a unit
    - **Note:** Interrupts are not going away (when will the IRQ happen)?
    - Why are per-value interrupts a problem?

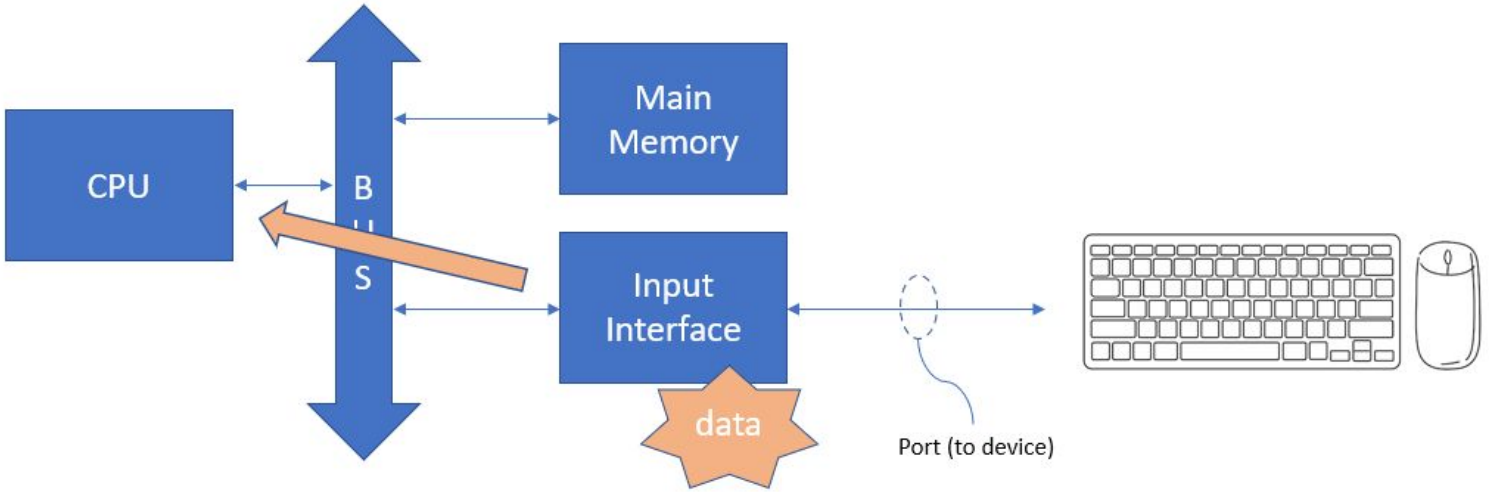
Approaches to Data Transfer

- Data is coming/going between the I/O interface and device
- As this happens, data also needs to be transferred from/to the I/O interface on the computer side (our focus in these slides)



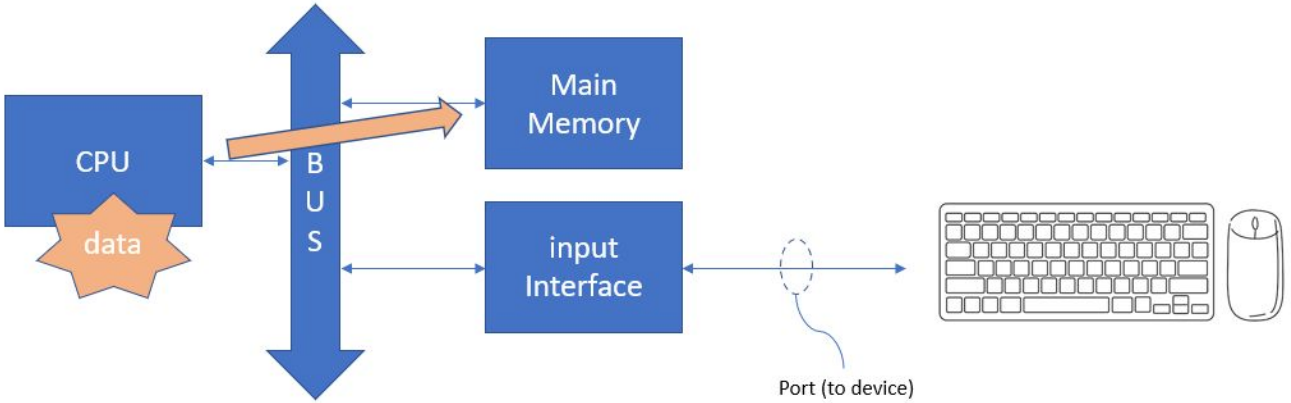
Approaches to Data Transfer

- E.g., once data has arrived from an input device ...
  - How is it obtained by the CPU?



Approaches to Data Transfer

- E.g., once data has arrived from an input device ...
- How is it obtained by the CPU?
- How is it transferred to main memory for later use?



Approaches to I/O

- There are three main approaches to I/O
  1. Programmed I/O
  2. Interrupt Driven I/O
  3. DMA (Direct Memory Access) I/O
- **Note:** Approaches to I/O are not mutually exclusive
  - More than one can be used in a single system

Approaches to I/O - Programmed I/O

- Main (non-ISR) algorithm manages I/O directly
- Continually polls interface (no IRQs)
  - Transfers data when interface ready
- Requires least-sophisticated hardware
- Has highest software overhead
  - Example: Busy wait
  - Risk of over/under-run

Approaches to I/O - Interrupts

- Install ISR that will be automatically CPU-invoked upon IRQ (if not masked)
  - Freeing up the logic of the main algorithm while increasing responsiveness
- ISR transfers data between interface and main memory (e.g., to/from a buffer)
  - The main algorithm only interacts with data in main memory
- Requires more sophisticated hardware
- Has much lower software overhead (even with ISR-vectoring overhead)

Approaches to I/O - Interrupts

- Are there any drawbacks to interrupt-driven I/O?
  - It's better than polling, but the CPU is still involved in every I/O read/write operation. An ISR must be invoked for each one, which can be expensive – even if the CPU is just copying data from an I/O register to memory or vice versa!
  - This is too slow for high-speed devices which transfer large blocks of data at a time,
    - Example: hard drives
  - DMA (direct memory access) solves this problem by allowing direct I/O interface to memory transfers, and vice versa

Approaches to I/O - DMA

- DMA (direct memory access) solves this problem by allowing direct I/O interface to memory transfers, and vice versa
    - Can go from device to memory **or** memory to device
    - Requires more sophisticated hardware
- A DMA controller manages the transfer, and becomes bus master as necessary to carry out the transfer
- Note that the CPU is still involved, in that it must initiate the DMA transfer. Additionally, it is typically notified of transfer completion via an interrupt

DMA - Main Issues

- I/O controller must be provided with:
  - The operation to perform
  - A pointer to the main memory buffer
  - The size of the block
- I/O controller must maintain additional registers:
  - Index register (pointer into main memory)
  - Count register (number of bytes transferred to/from main memory)
- I/O controller must be able to drive the bus!
  - CPU is continually using the bus for transfers to/from main memory!
  - Simpler I/O interfaces are connected to the system bus via a DMA controller

DMA Controller Registers

- A typical DMA controller has, minimally:
  - A control register (e.g. can control whether operation is read or write)
  - A status register
  - An address register
  - A data count register
  - 1-more data registers for buffering the data to be transferred

Steps to Initiate DMA transfer

- The Software:
  1. Software prepares a main memory buffer
  2. Software requests an I/O operation
  3. Software then leaves the block to be transferred “in the background” without CPU involvement
  4. An IRQ typically signals that the operation has completed
     - The data requested is now in the main memory buffer

Steps to Initiate DMA transfer

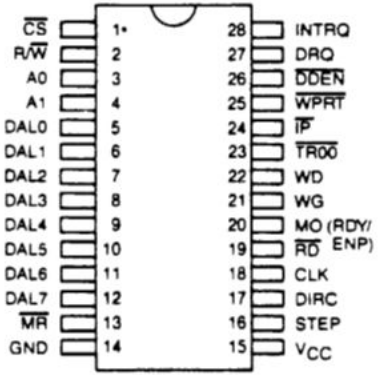
- The CPU:
  - Programs the DMA controller for read or write
  - Loads the address register with the start address of the transfer
  - Loads the data count register with the number of bytes to be transferred
- **Note:** At this point the CPU will no longer be directly involved as the DMA controller will communicate directly with memory for the data transfer

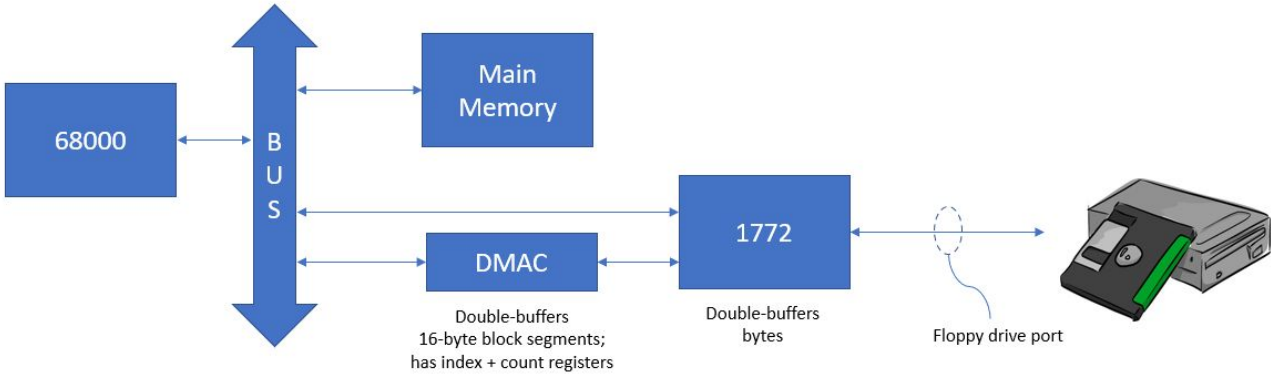
Steps to Initiate DMA transfer

- The DMA controller periodically requests the bus. When the bus is granted, the DMA controller uses it to transfer data.
    - The controller may use “cycle stealing” to transfer a small number of bytes at a time, or, if it has larger buffers, it may transfer the data in bursts.

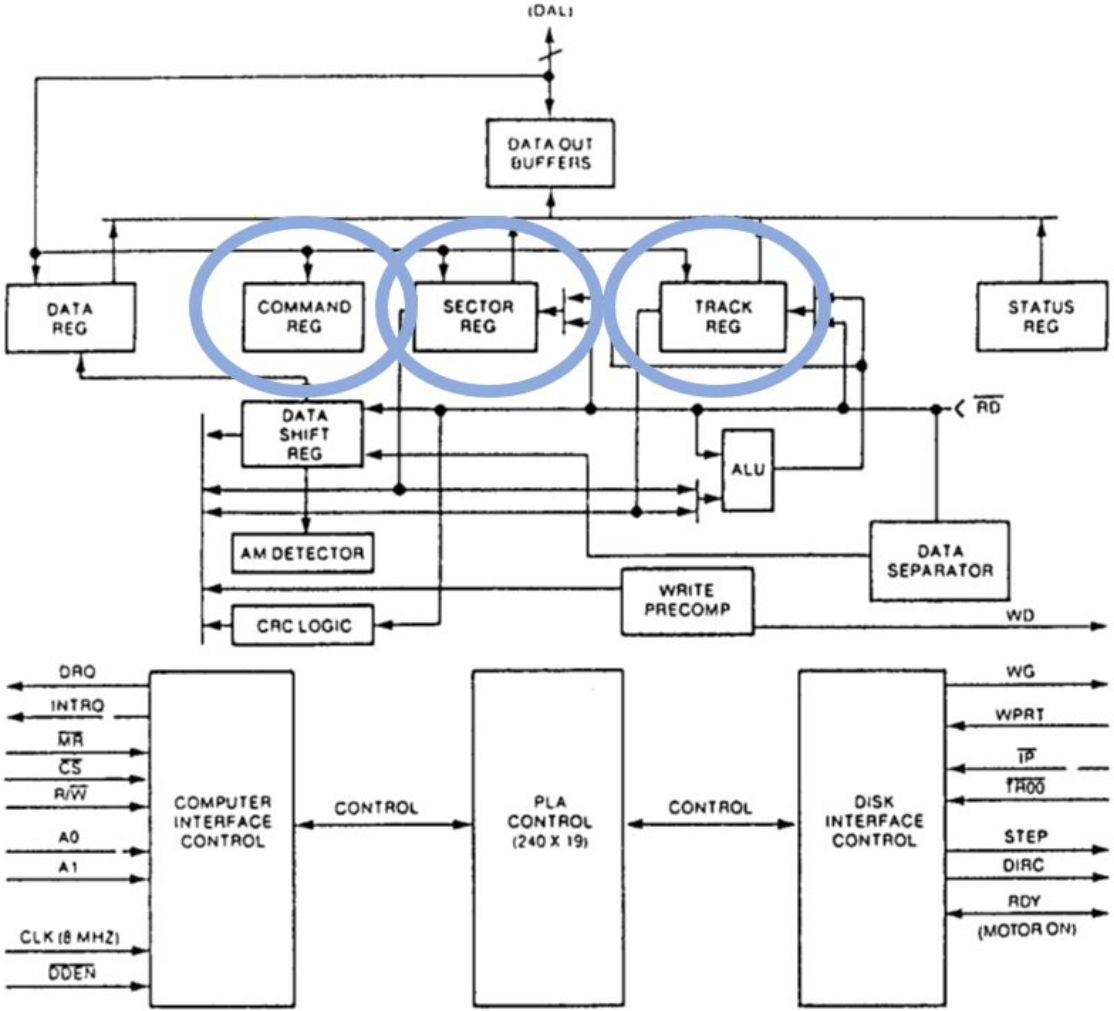
Simple Example - FDC on Atari ST

- Floppy drive controller:
  - The WD1772
  - Connected to system bus via the DMA controller



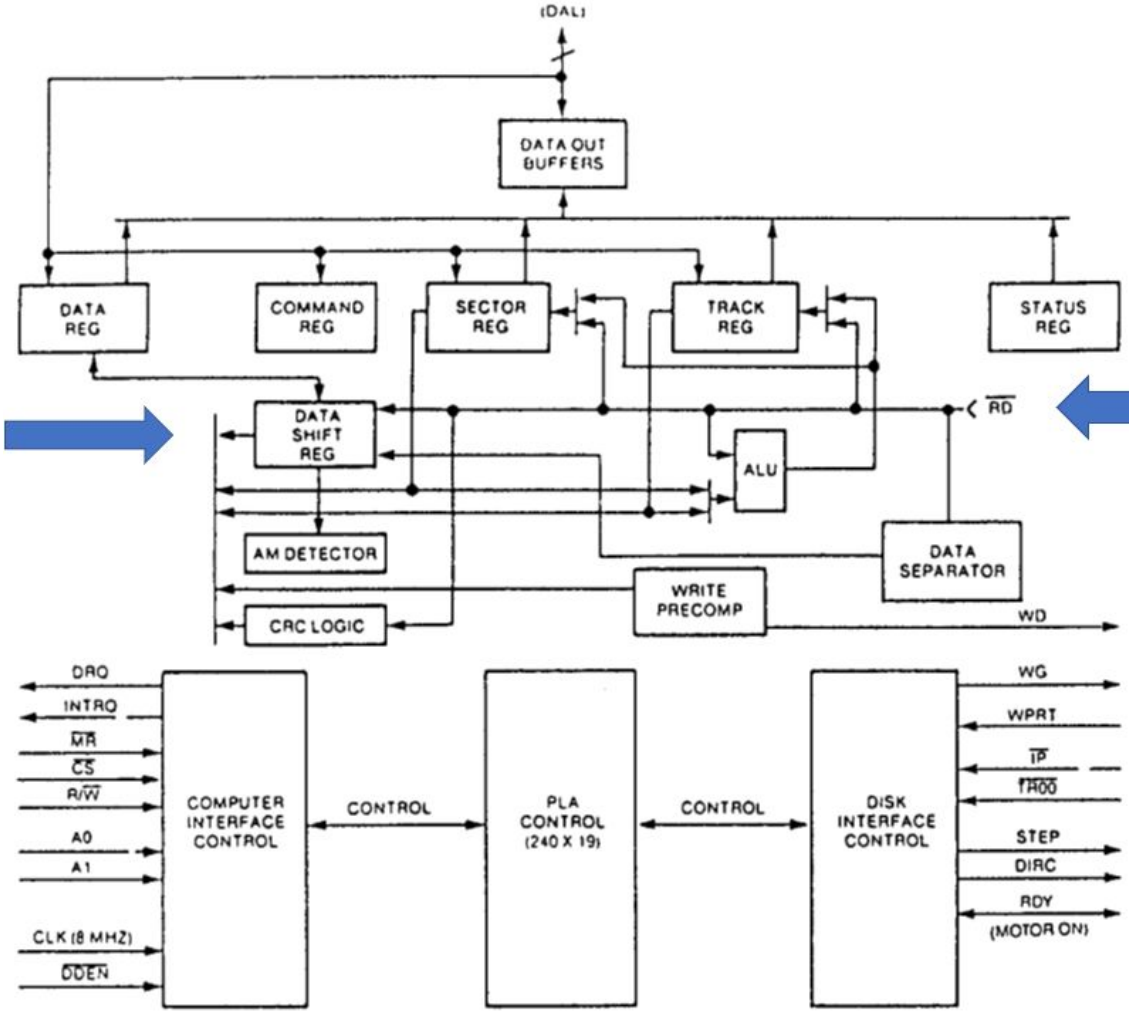


Block diagram of 1772 internals

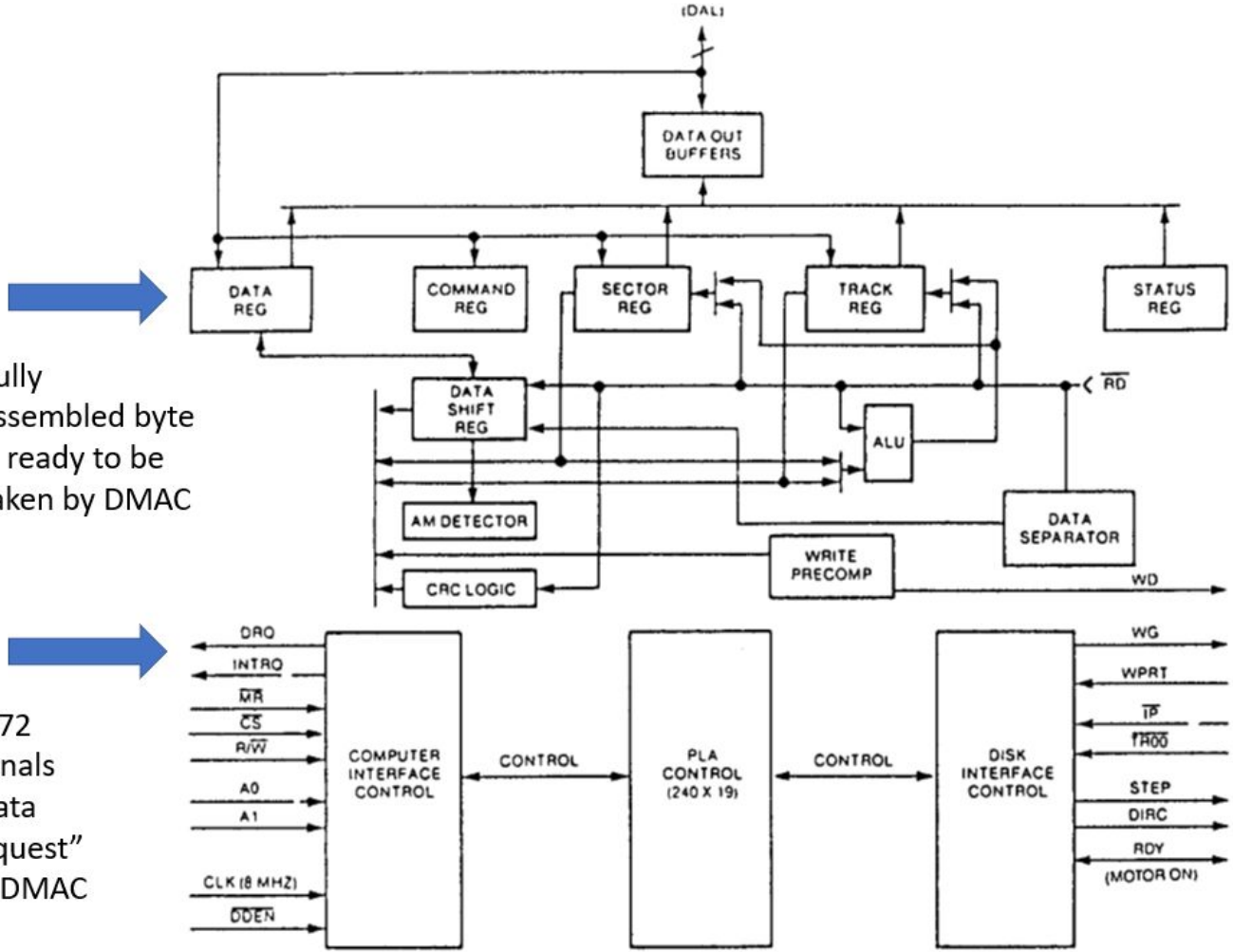


Initiate a read command via write to command register

(Block location on disk specified via track/sector registers)



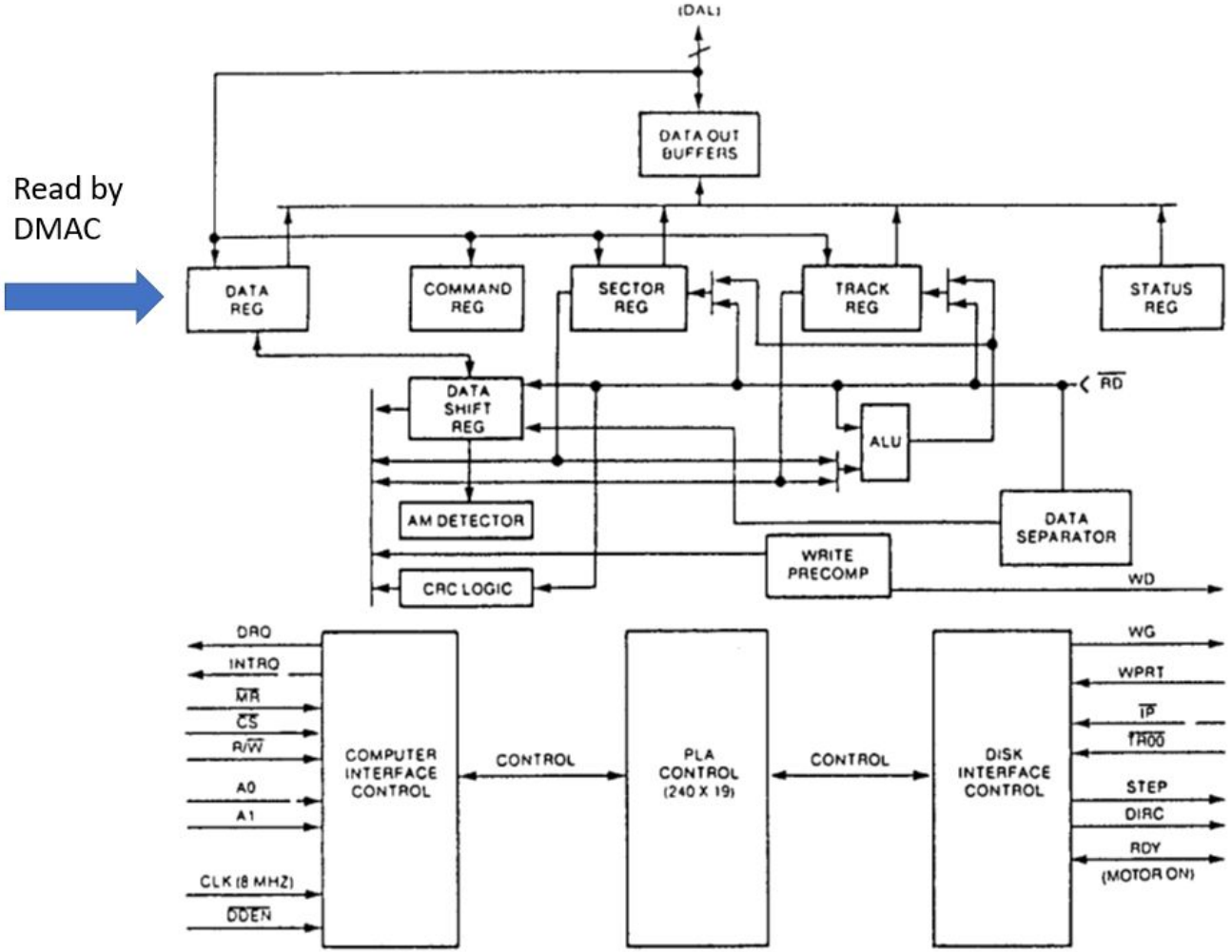
Bytes arrive serially over floppy port



Fully assembled byte is ready to be taken by DMAC

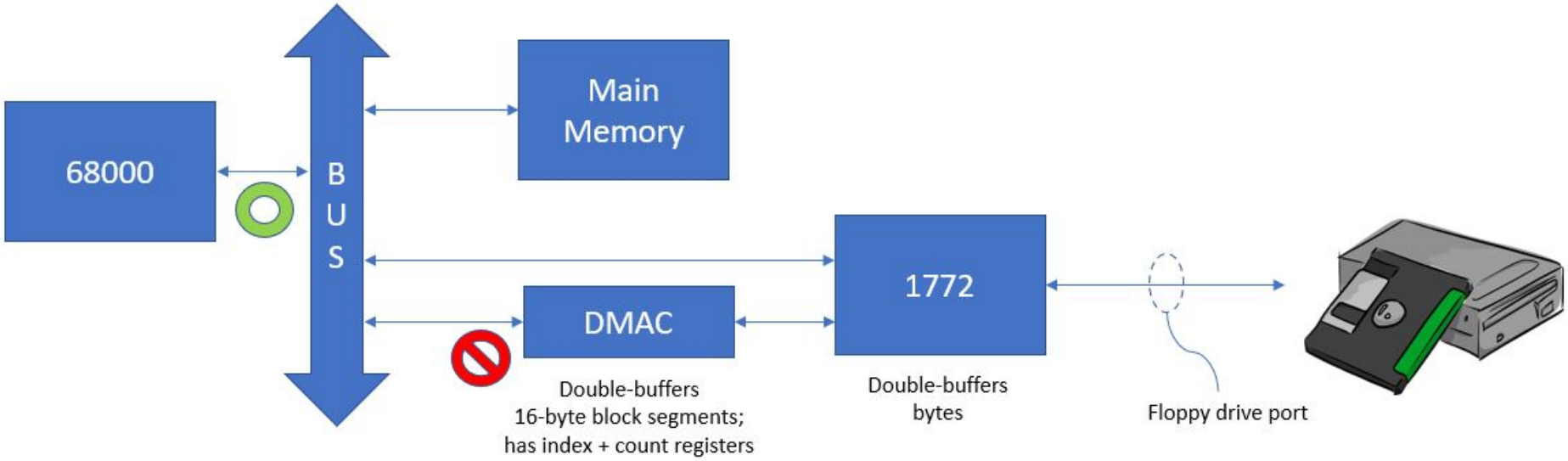
1772 signals "data request" to DMAC

Read by DMAC


(DAL)
DATA OUT BUFFERS
DATA REG
COMMAND REG
SECTOR REG
TRACK REG
STATUS REG
DATA SHIFT REG
ALU
RD
AM DETECTOR
DATA SEPARATOR
WRITE PRECOMP
CRC LOGIC
WD
DRQ
INTRQ
MR
CS
R/W
A0
A1
CLK (8 MHZ)
DDEN
COMPUTER INTERFACE CONTROL
CONTROL
PLA CONTROL (240 X 19)
CONTROL
DISK INTERFACE CONTROL
WG
WPRT
IP
TR00
STEP
DIRC
RDY
(MOTOR ON)

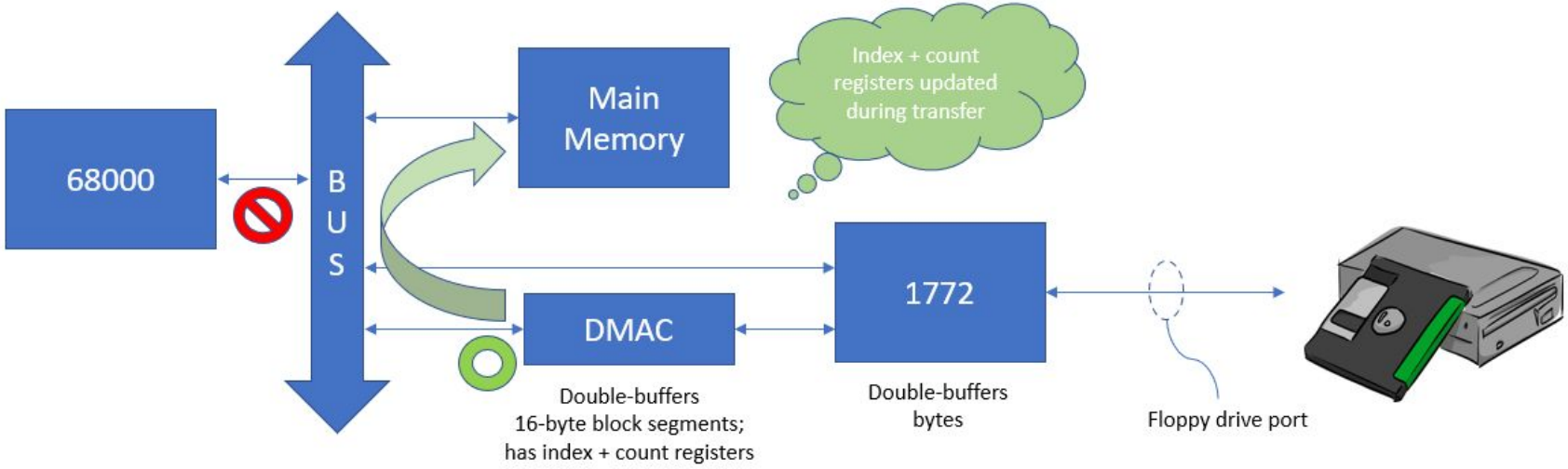
Simple Example - Atari ST's DMAC

- By default, the CPU is the "bus master"



Simple Example - Atari ST's DMAC

- DMAC periodically requests to take control of the system bus
- To transfer the next segment of the incoming block directly to memory



DMA Interfaces

- An I/O interface may have its own built-in DMA controller
- Alternatively, a distinct DMA controller may manage a set of I/O interfaces, which may not themselves be connected directly to the bus
- E.g. the 68000 supports DMA via its BR, BG and BGACK signals
- E.g. the ST has a DMA controller which manages the hard drive and the floppy drive(s)
    - Actually, the memory indexing part of the DMA functionality is performed by the MMU chip.

Bus Mastering

- The bus master (here, 68000) supports a handshaking protocol with the DMAC:
  - **B**us **R**equest ← from DMAC to 68000
  - **B**us **G**rant ← response from 68000 (when willing)
  - **B**us **G**rant **Ack**nowledge ← asserted by DMAC until transfer complete

Bus Mastering

- DMACs typically transfer data in one of two modes:
  - Burst
    - Upon being granted the bus, all buffered bytes are transferred one after the other
  - Cycle stealing
    - One value is transferred per bus grant
    - Allows interleaved use of bus by CPU and DMAC
    - Cycle-stealing can be used by a DMA-enabled video controller to continually scan a frame buffer as it generates a video signal

Final Notes

- Also to add:
  - IO interface ⟶ IO controller ⟶ **IO (co)processor (e.g., GPU)**
    - Start with basic IO interface providing little more than buffering and polling support
    - … ⟶ add interrupt capability (and PIC support)
    - … ⟶ add storage & transfer capability (maybe with separate DMAC support)
    - … ⟶ **add compute capability (+ more onboard memory and/or DMA capability)**