Direct Memory Access (DMA)

- What is DMA?
- Why do we use it?
- Alternatives to using DMA.
  - CPU Polling + Data Transfer.
  - Interrupt Driven Requests + Data Transfer.
- DMA controllers - Protocol of Requesting the Bus.
- Transfers between IO and Memory.
- Transfers between IO devices.
- Transfers from Memory to Memory.
- DMA modes of operation - Burst vs. Interleaved Operation.

Direct Memory Access (DMA)

What is DMA?

- DMA or Direct Memory Access is a process whereby a slave device (a DMA controller) is able to take over control of the main CPU system buses, (address, data and control) and become a Bus Master able to initiate rapid transfers of data between the Memory and IO subsystems without the need for the CPU to become involved.

Why is it Used?

- It leads to greater performance because:-
- DMA controllers are more efficient at transferring blocks of data than a CPU trying to do the same thing using a program.
- For example, in a multi-tasking operating systems environment like LINUX or Windows, a thread requesting a data transfer from say a disk drive or network can be suspended by the Operating System (OS), and the DMA controller can perform the data transfer in the background.
- The OS could, while waiting for the transfer to take place, perform a context swap and run another thread that is not waiting for IO.

Direct Memory Access (DMA)

Why can't the CPU perform the transfer as efficiently as DMA?

- Consider how such transfers would have to take place under the control of a program.

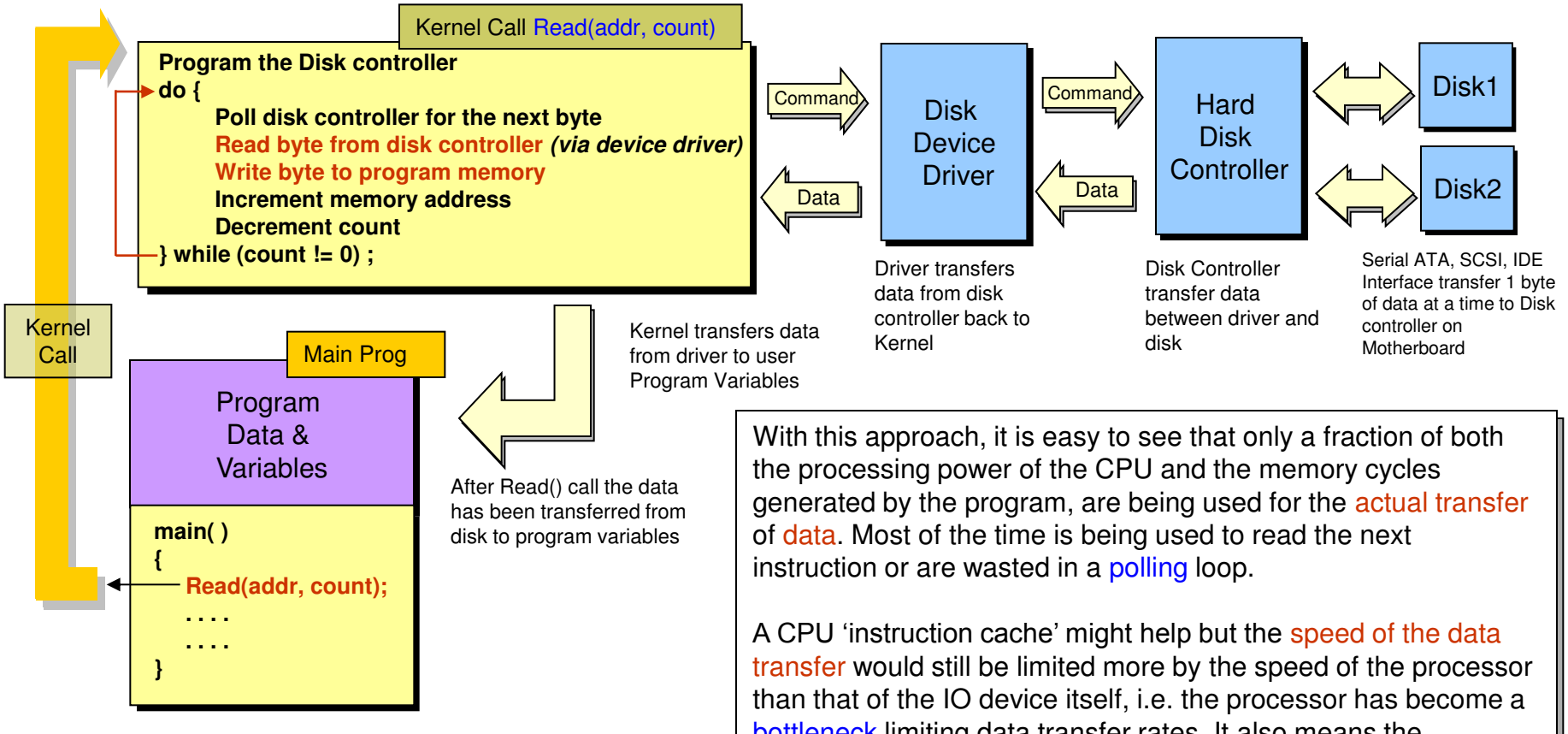
- Without a DMA controller there are two means a program could be written to transfer blocks of data between IO devices and memory.

  - Using Polling.
  - Using Interrupt Driven IO.

Direct Memory Access (DMA)

Consider Polling – a Software Bottleneck

- Consider the problem of transferring a block of data from a high speed device such as a disk drive (>50MB/sec) (or a network (>100MB/sec)) to memory one byte at a time under software control.
- The following algorithm gives an indication of what is required of the program.
- The user program starts by calling the kernel with a read request, which invokes the device driver to transfer the data from the hardware device into memory.



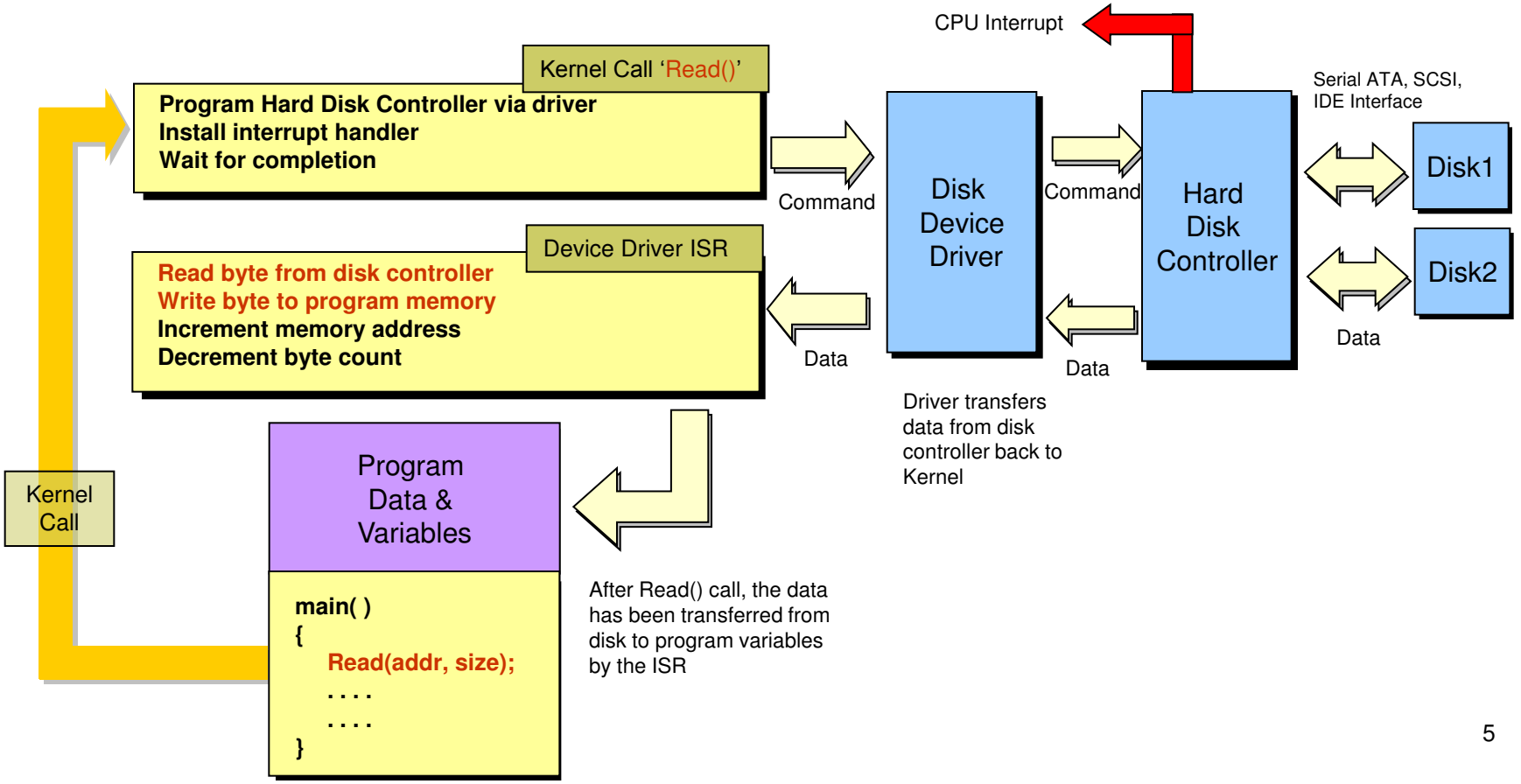
With this approach, it is easy to see that only a fraction of both the processing power of the CPU and the memory cycles generated by the program, are being used for the actual transfer of data. Most of the time is being used to read the next instruction or are wasted in a polling loop.

A CPU 'instruction cache' might help but the speed of the data transfer would still be limited more by the speed of the processor than that of the IO device itself, i.e. the processor has become a bottleneck limiting data transfer rates. It also means the processor isn't able to do anything else or communicate with other devices during this polled transfer.

Direct Memory Access (DMA)

Interrupt Driven IO – Just as big a Software Bottleneck

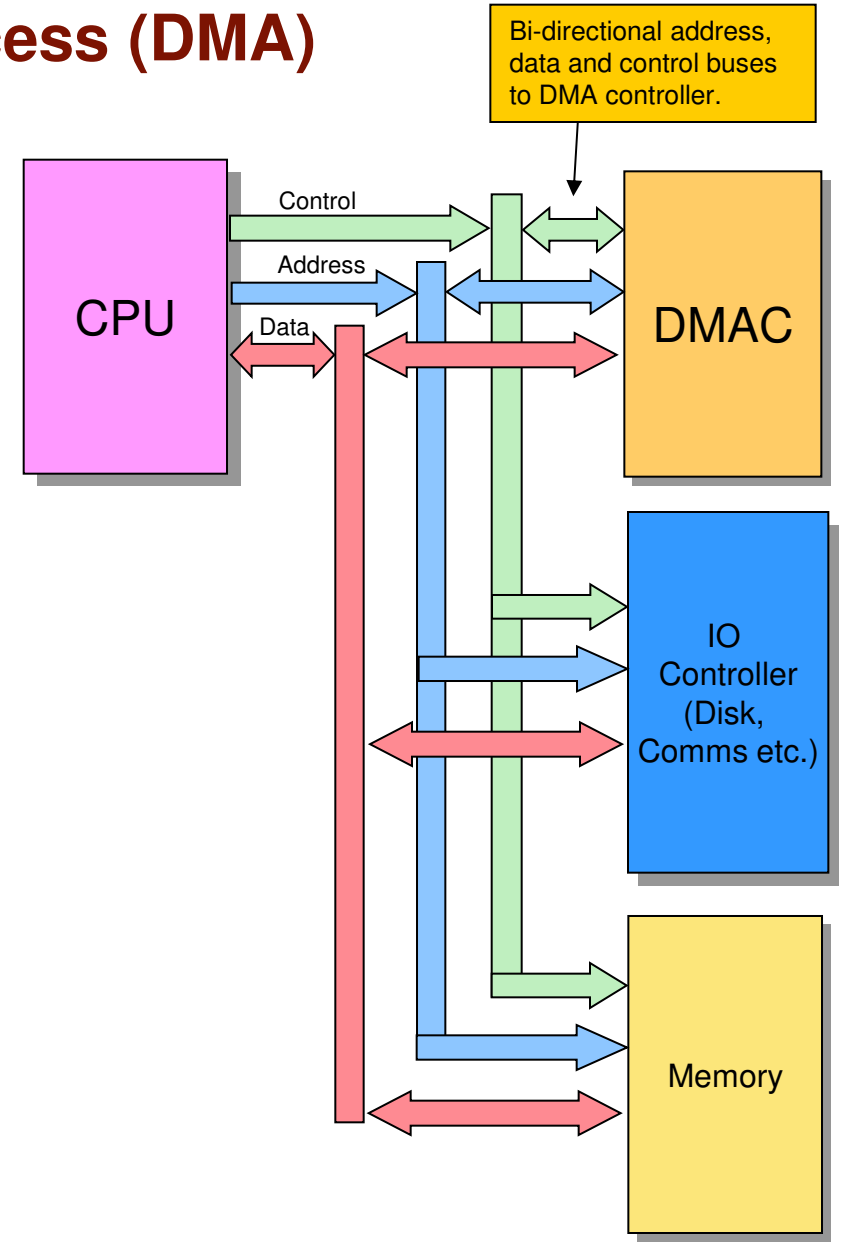
- Generating an Interrupt per item of data transfer is just as bad – in some ways worse!!
- Data still has to be transferred via software inside the interrupt service routine (ISR) for the device driver but now we have the overhead of calling the ISR for each byte/sector or whatever transferred. However, at least the CPU is free do other things between interrupts rather than polling.



Direct Memory Access (DMA)

How is DMA Performed?

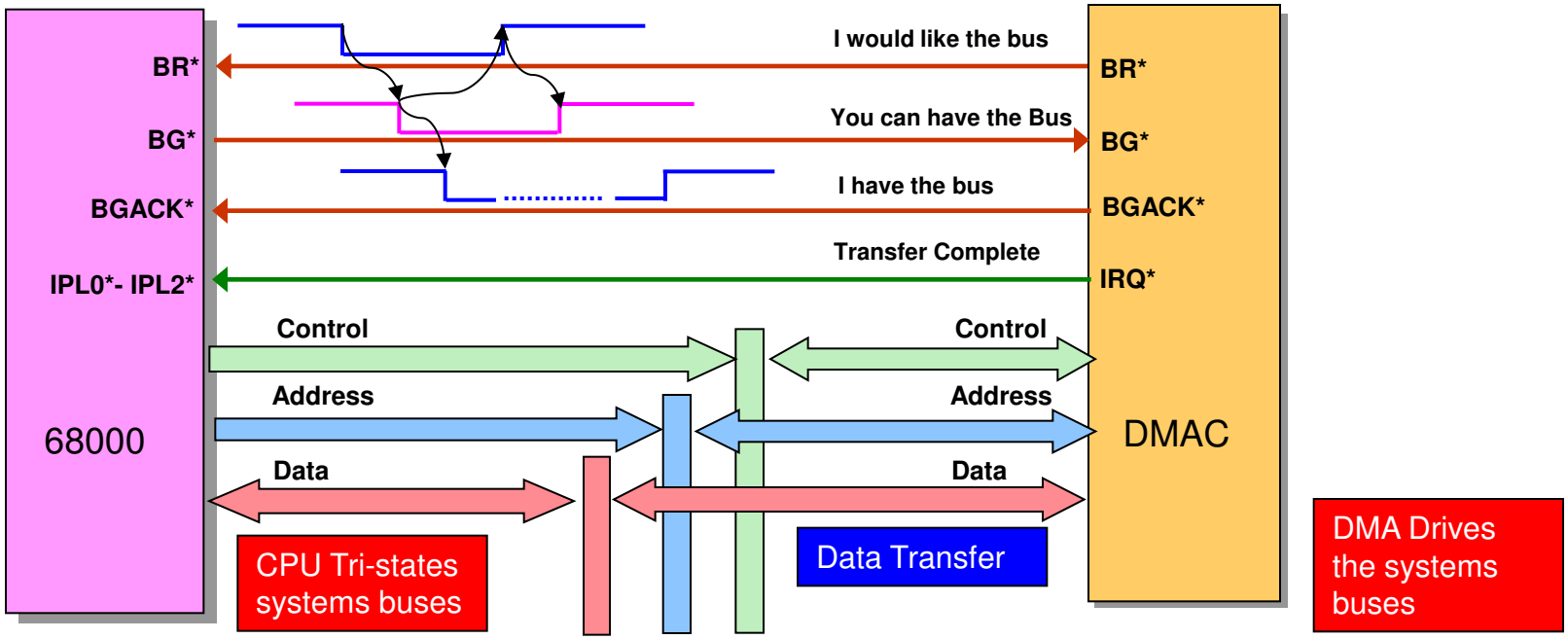
- DMA or Direct Memory access is performed by a dedicated hardware chip called, not surprisingly, a DMA controller (or DMAC).
- Such a chip is able to transfer data at high speed between memory and/or DMA capable peripheral (*such as a disk controller, network and other high bandwidth devices*) without the intervention of the CPU.
- A DMA controller is a programmable peripheral chip with registers which can be written to by the CPU to control the transfer.
- DMA controllers differ from most other peripherals in a system in that they are able to take over and drive the main system buses *[Address, Data and Control buses]* in order to carry out the data transfer.
- Notice how the address, and control buses of the DMAC are bi-directional (very different to other peripherals).
- The DMAC is able to emulate the exact signals, protocols and timing generated by the CPU.
- In modern high performance microcontrollers the DMA controller may be integrated into the same chip as the CPU.



Direct Memory Access (DMA)

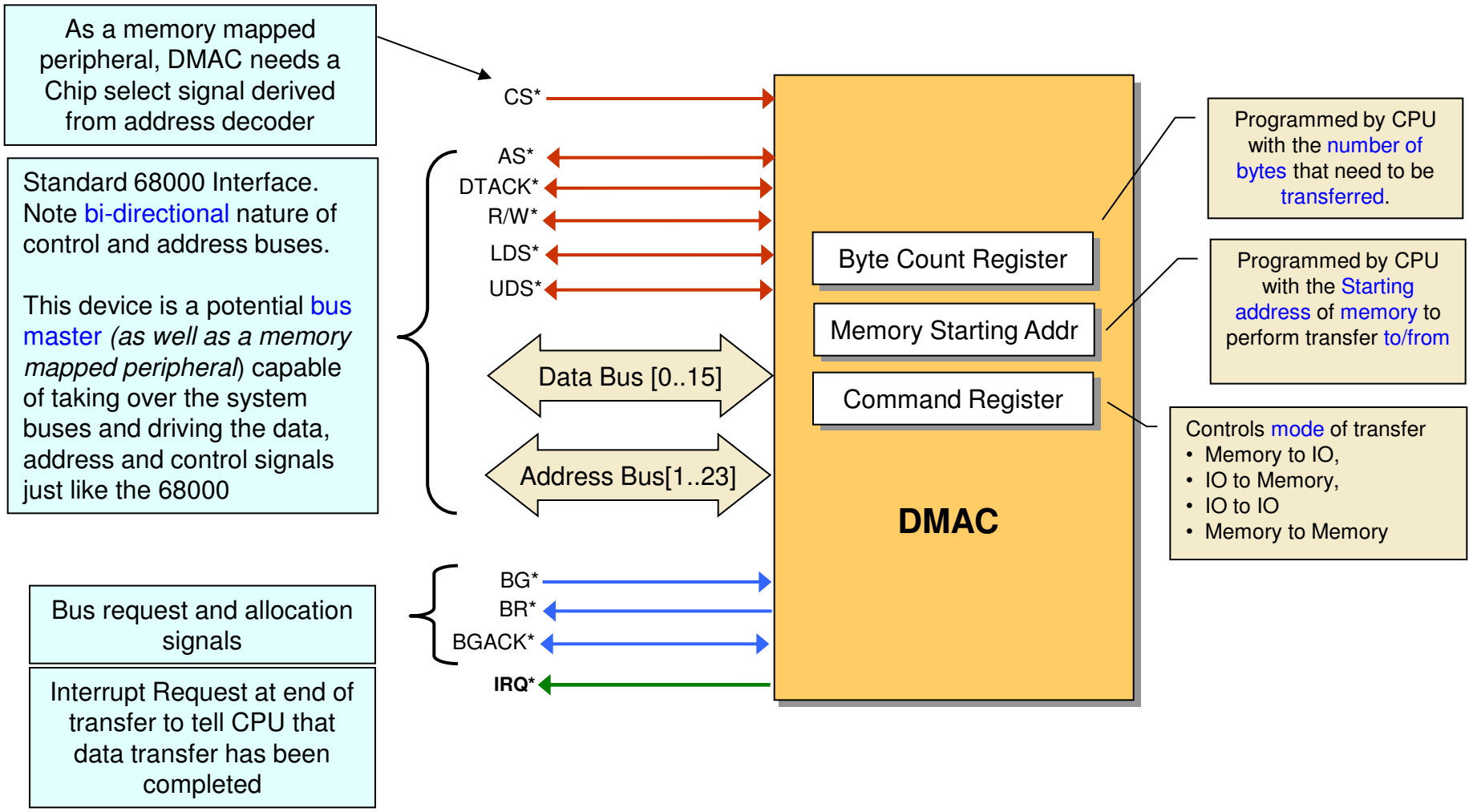
How does the DMAC control the Bus?

- The protocol used by a DMAC to take control of the bus in a 68000 based system, is quite straight forward and only three signals are involved.
  - Bus Request (BR*): Asserted by the DMAC to request current bus master vacate the bus at the end of the current bus cycle.
  - Bus Grant (BG*): Asserted by the CPU once it has disconnected itself from the bus and tri-stated all address, data and control lines (AS, R/W, UDS, LDS etc).
  - Bus Grant Acknowledge (BGACK*): Asserted by the DMAC to indicate it has taken control of the bus.
- When the slave has finished, it relinquishes control back to the 68000 by negating BGACK*. The CPU then takes control of the bus and resumes processing.
- A DMAC can be programmed to interrupt the CPU when it has finished to indicate completion of the transfer (*the CPU will be unaware that it has taken place otherwise*).



Direct Memory Access (DMA)

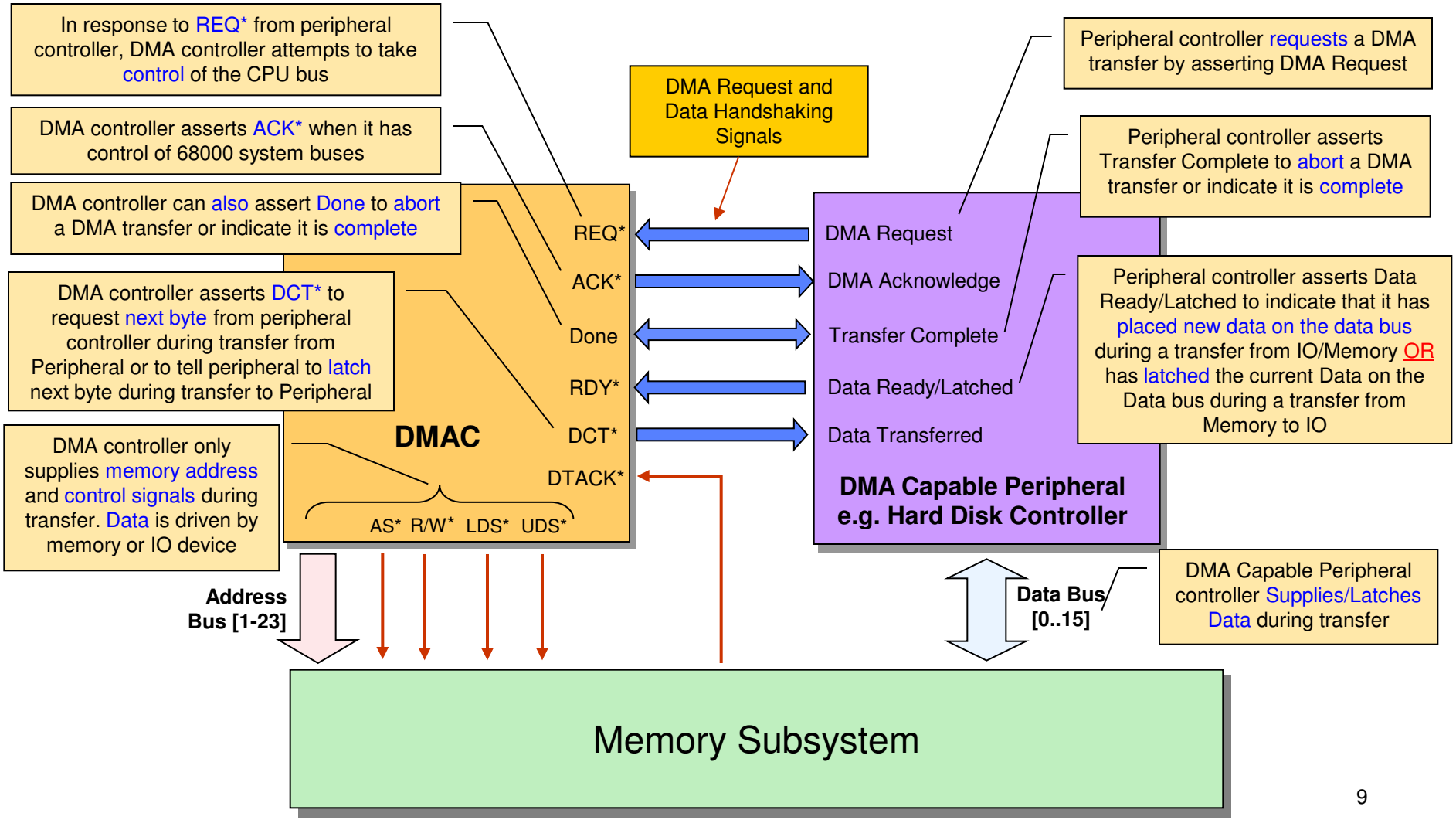
- Architecture of a simple 68000 compatible DMAC



Direct Memory Access (DMA)

Interface between DMAC, Peripheral Controller and Memory

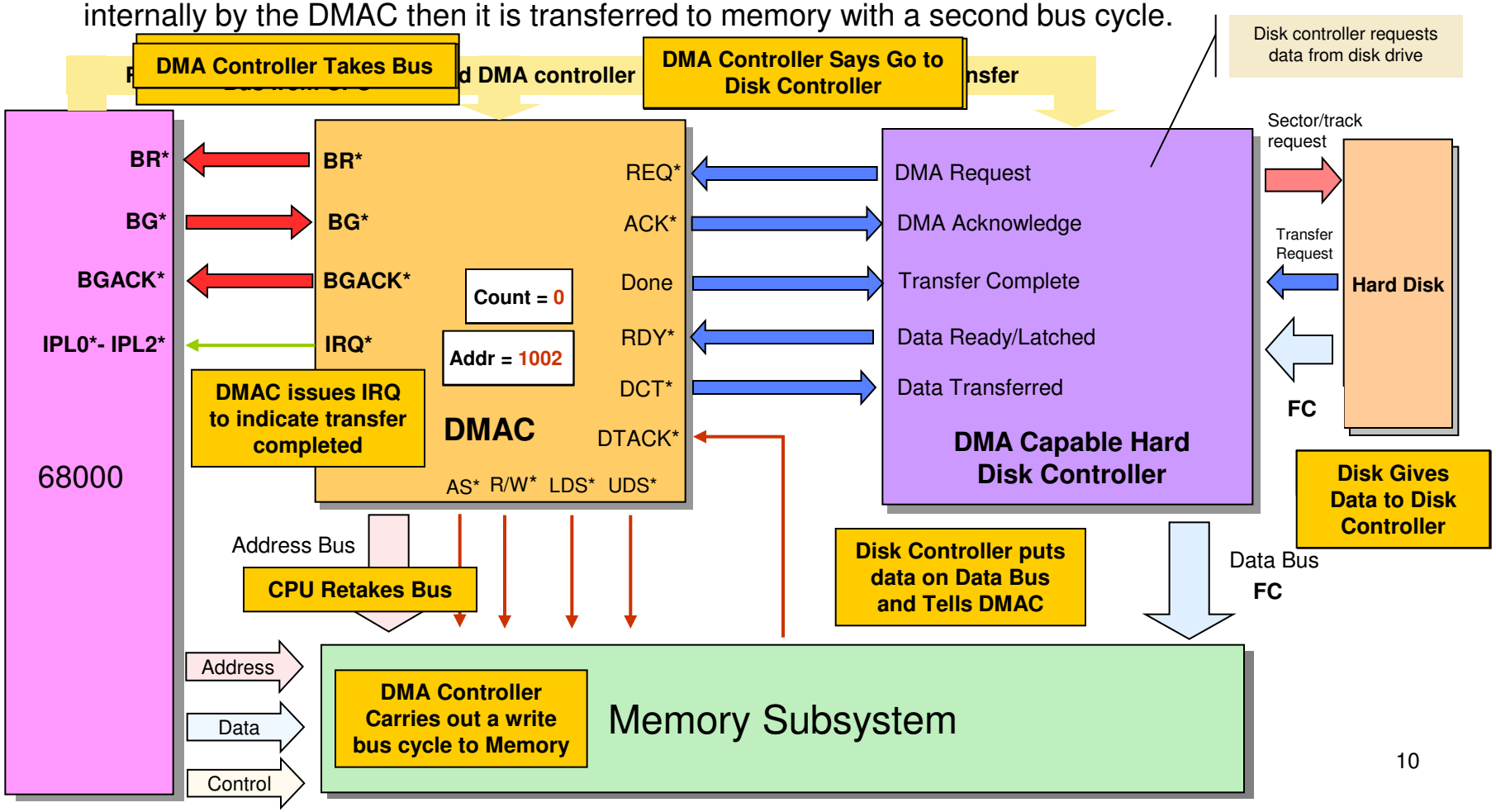
- Note DMAC can be used to transfer data from Memory to Peripheral or from Peripheral to Memory



Direct Memory Access (DMA)

Interface between DMAC, Peripheral Controller and Memory

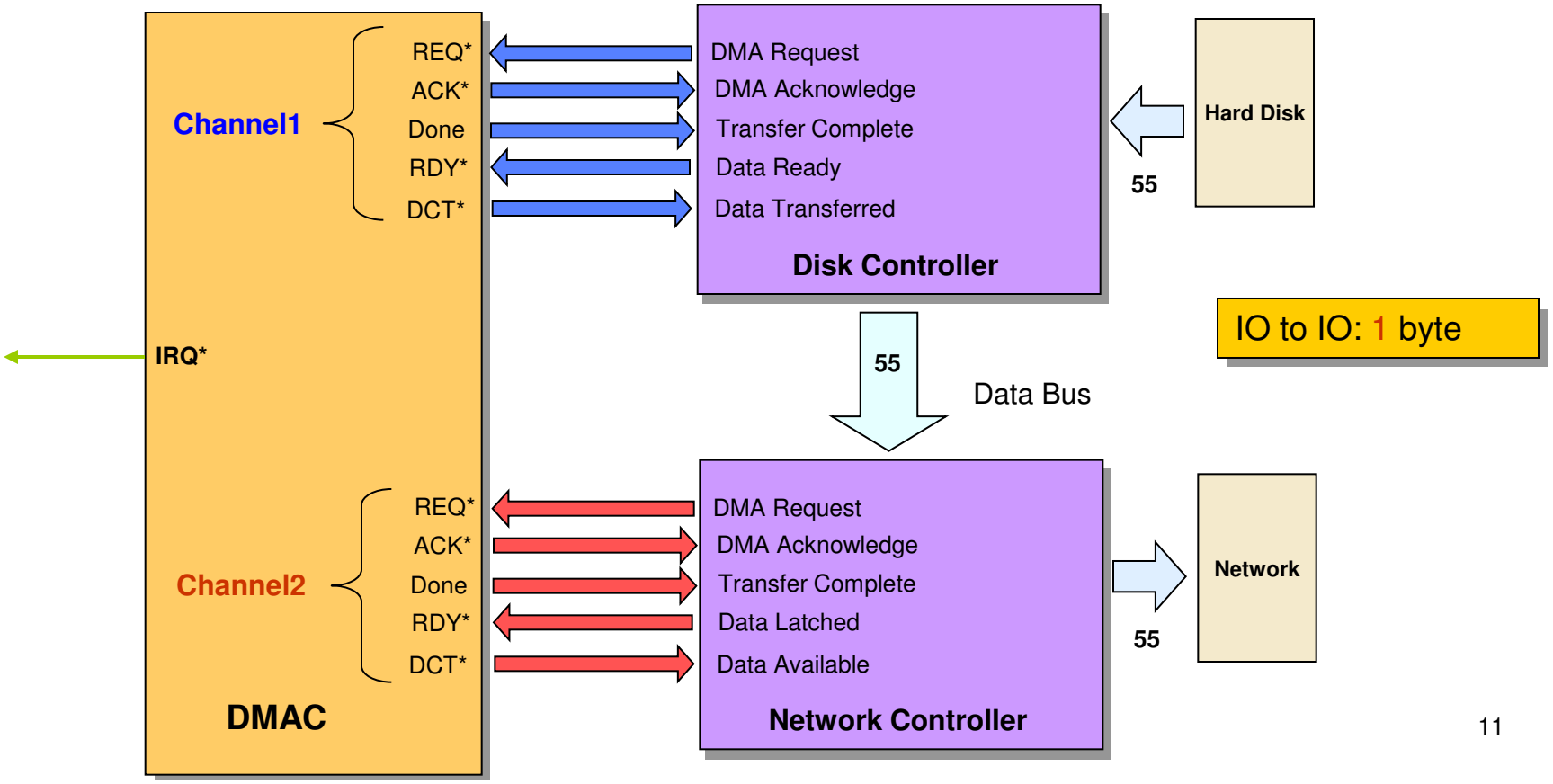
- Note DMA can be used to transfer data from Memory to Peripheral or from Peripheral to Memory
- Flyby DMA (demonstrated in the animation below) allows the data to be read from the IO and transferred to the Memory in one bus cycle, this is not always supported by all controllers but is the fastest way to do it; if not a read of the IO is done and a temporary copy of the data is made internally by the DMAC then it is transferred to memory with a second bus cycle.



Direct Memory Access (DMA)

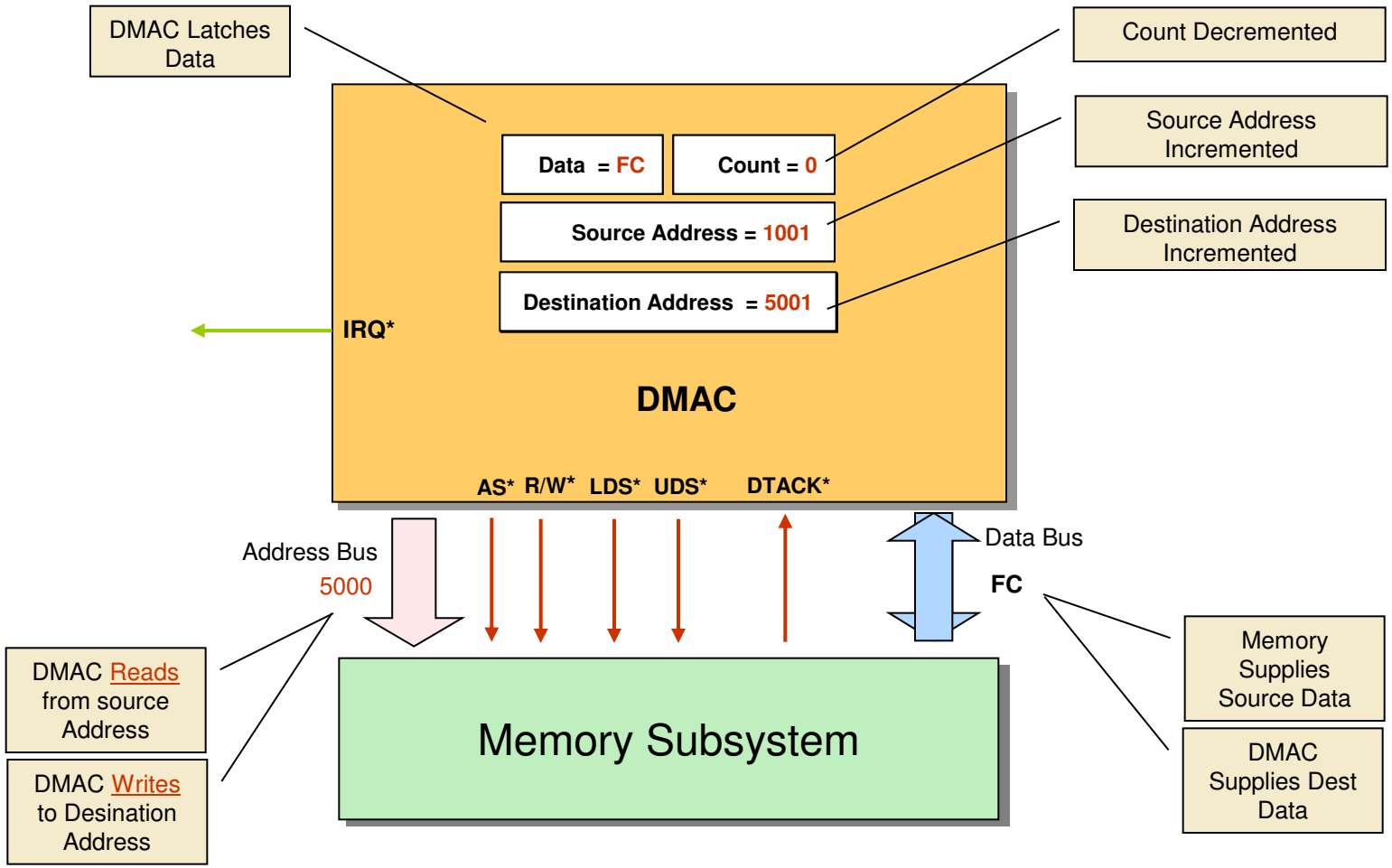
Peripheral to Peripheral Transfers using DMA

- If the DMA controller has two or more IO channels, then transfers between peripherals are possible also, e.g. direct from a Hard disk controller to a Network controller for example.
- Network controller has to be programmed by CPU to receive data via DMA, disk controller has to be programmed to supply data via a DMA and DMAC has to be programmed to transfer data from Channel 1 to Channel 2



Direct Memory Access (DMA)

Memory to Memory Transfers



Memory to Memory Transfers always takes 2 bus cycles

Direct Memory Access (DMA)

Modes of Data Transfer: Interleaved and Burst Transfers

- Most DMACs can be programmed to consume a variable percentage of the available memory bandwidth of the system in order to carry out their transfers. There are two modes
  - Burst
  - Interleaved
- In 'burst' mode a DMAC is programmed to work as fast as possible and may consume 100% of the available memory bandwidth in order to carry out the transfer of data. The CPU may be completely locked out until the completion of the transfer.
- This has a number of disadvantages.
  - The DMAC can upset the operation of a system, particularly a real-time system where the CPU may not be able to react to an event or interrupt if there is a DMA going on and thus may miss its response time.
  - In a multi-tasking system, the process initiating the DMA becomes super-high priority because it locks out other processes from running.
- For this reason most DMA controllers can be programmed to interleave their operation with that of the CPU, a form of "Cycle Stealing" whereby a byte is transferred while the CPU is not using the bus or maybe it alternates with it CPU-DMAC-CPU-DMAC etc. (The CPU still needs to be prevented from attempting to use the bus during the brief period)
- http://www.embedded.com/shared/printableArticle.jhtml?articleID=15300200

Direct Memory Access (DMA)

The Intel 8037 – 4 channel DMA