Computer interfacing (CI)

9. DMA peripheral

9.1 DMA

Delegating I/O Responsibility from the CPU

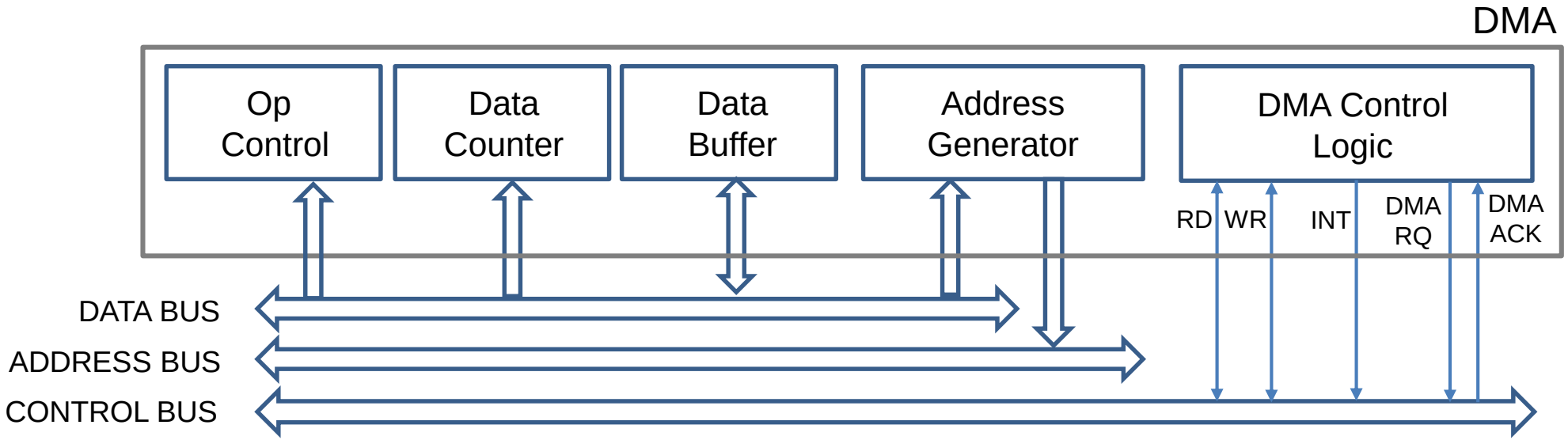
Interrupt-driven I/O relieves the CPU from waiting for every I/O event, increasing polling method's performance, but there are still many CPU cycles spent in transferring data.

Transferring a disk block of 2048 words, for instance, would require at least 2048 loads and 2048 stores, as well as the overhead for the interrupt.

Since I/O events so often involve block transfers, direct memory access (DMA) hardware is added to many computer systems to allow transfers of numbers of words without intervention by the CPU.

DMA is a specialized processor that transfers data between memory and an I/O device while the CPU goes on with other tasks. Thus, it is external to the CPU and must act as a master on the bus. The CPU first sets up the DMA registers, which contain a memory address and number of bytes to be transferred. Once the DMA transfer is complete, the controller interrupts the CPU. There may be multiple DMA devices in a computer system; for example, DMA is frequently part of the controller for an I/O device.

9.2 DMA operation



DMA function

- The processor sets up the DMA by supplying the peripheral device ID, the operation to perform on it, the memory address for the operation and the number of bytes to move.

- The DMA starts the operation on the device and arbitrates the access to the BUS (DMA RQ/ACK), supplies the memory address for the read or the write, interfaces the device and may contain some buffers to store the peripheral data while waiting to access the bus.

- Once the DMA transfer is complete an interrupt is generated.

9.2 DMA operation

Performance analysis example

(1) A microcontroller executes an instruction every 100ns. A peripheral provides data to the bus every 10us. We assume a single bus structure. We want to transfer 16KB of data using polling method. Calculate the time needed and the percentage of CPU used.

Code:

```
 ...
i=0;
while( i<16192)
{
 while(!data_rdy());
 data[i++]=read_data();
}
...
```

Total time: Time= 16192*10 µs ≈ 162ms

CPU is 100% used during 162ms.

9.2 DMA operation

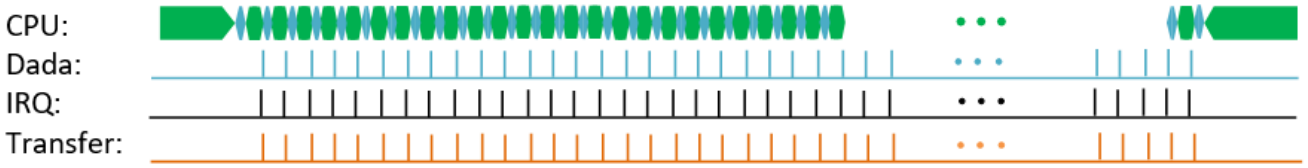
(2) Let's assume now that we will use an interrupt for every BYTE of data. Again, we are going to calculate the time needed and the percentage of CPU used.

Code:

```
...
Configure_ISR()
i=0;
done=FALSE;
...
```

```
interrupt ISR_Peripheral ()
{
        data[i++]=read_data();
        if(i==16192) done=TRUE;
        EOI();
}
```

Total time: Time= 16192 x 10 µs ≈ 162ms



In 162ms CPU could execute: 0,162s / (100 ns/instruction) = 1620000 instructions
As 16192 interruptions are generated, assuming that 30 instructions are needed per interrupt:

Total Interrupt instructions = 16192 interrupts x 30 instr/interr = 485700 instructions

So, data transfer spent: 485700 / 1620000 ≈ 30% of CPU time

9.2 DMA operation

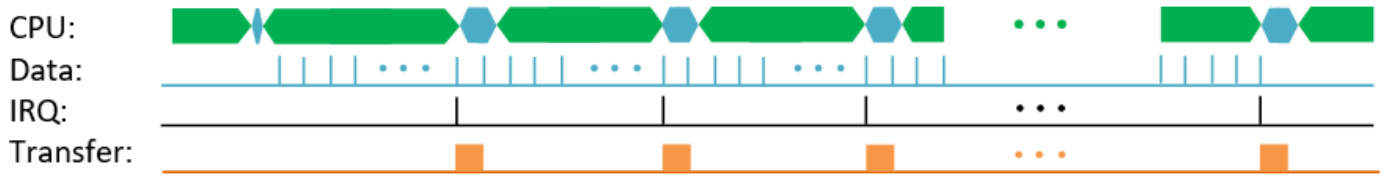
(3) Imagine now that we have an improved peripheral with an internal buffer of 64B that generates an interrupt every time the buffer is full.

Code:

```
...
Configure_ISR()
i=0;
done=FALSE;
...
```

```
interrupt ISR_Peripheral ()
{
        for (j=0; j<64; j++)
        {
          data[i++]=read_data();
        }
        if(i==16192) done=TRUE;
        EOI();
}
```

Total time: Time= 16192 x 10 µs ≈ 162ms

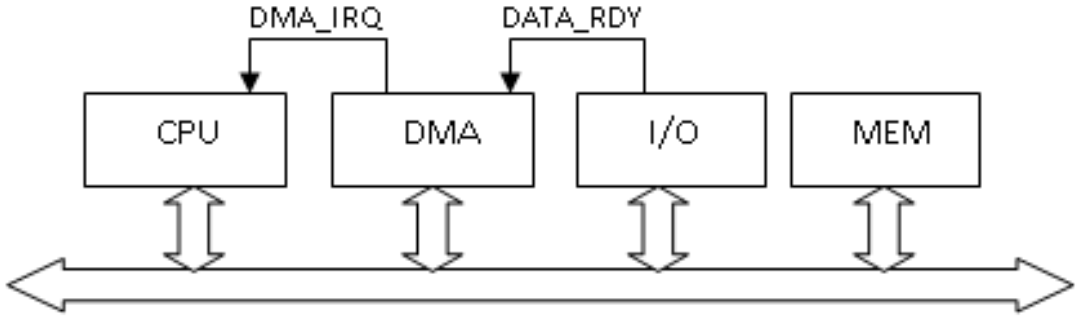


In 162ms CPU could execute: 0,162s / (100 ns/instruction) = 1620000 instructions.
As now we received 16192/64 = 256 interruptions, and assuming now 240 instructions per interrupt:
Total Interrupt instructions = 256 interr x 240 instr/interr = 61440 instructions

So, data transfer spent: 61440 / 1620000 ≈ 4% of CPU time

9.2 DMA operation

(4) Finally, we are going to use a DMA. It will master the BUS in synchronous mode, and will transfer one byte per operation.



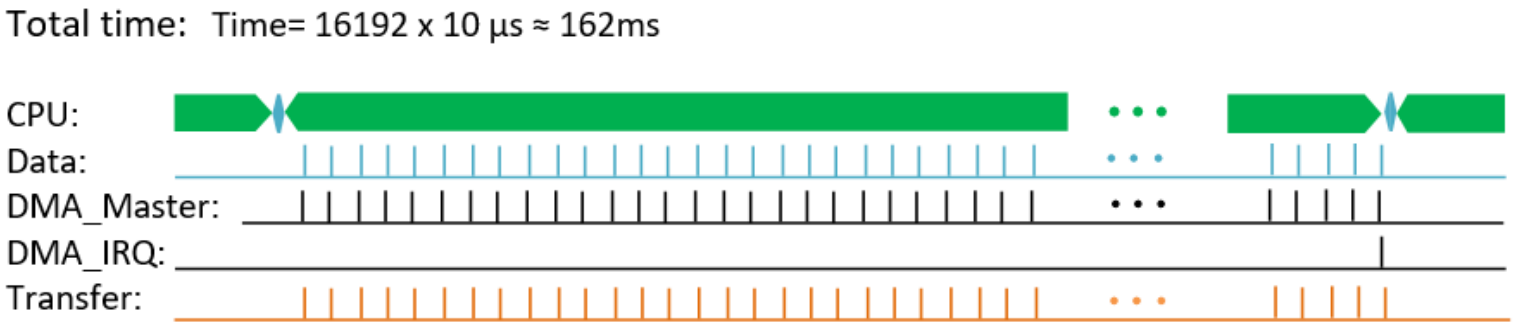
Code:

```
...
DMA_COUNT=16192;
DMA_ADDR = &data;
DMA_PERIF = &I/O;
DMA_MODE= Master|Sincron|IRQ;
done = FALSE;
DMA_ON ();
...
```

```
interrupt DMA ()
{
  done=TRUE;
  EOI();
}
```

9.2 DMA operation

(4) Let's calculate the time needed and the percentage of CPU used.



In 162ms CPU could execute: 0,162s / (100 ns/instruction) = 1620000 instructions.
In this time, we will have 16192 DMA bus operations.

Assuming the worst case: every time DMA took the bus, the CPU needed the bus and has been paused, we've lost:

Total DMA = 16192 dma_master x 1 instr = 16192 instructions.

Transfer need 16192 instruccions, plus DMA configuration and interrupt (only 1) attention:

Data transfer time spent: 16192+K / 1620000 ≈ 1% of CPU time
(this is the worst case, register operations or cache access will improve it)

9.3 DMA discussion

There are several DMA possible modes:

- Cycle stealing †: the DMA takes control of the bus for a single transfer when CPU doesn't access it (ALU ops, cache access)
- Bus mastering: the DMA takes control of the BUS to perform the transfers, blocking access of other devices to the bus.

- Single data operation (a word is moved in every DMA access)
- Multiple data operation (or burst, the DMA moves blocks of data)

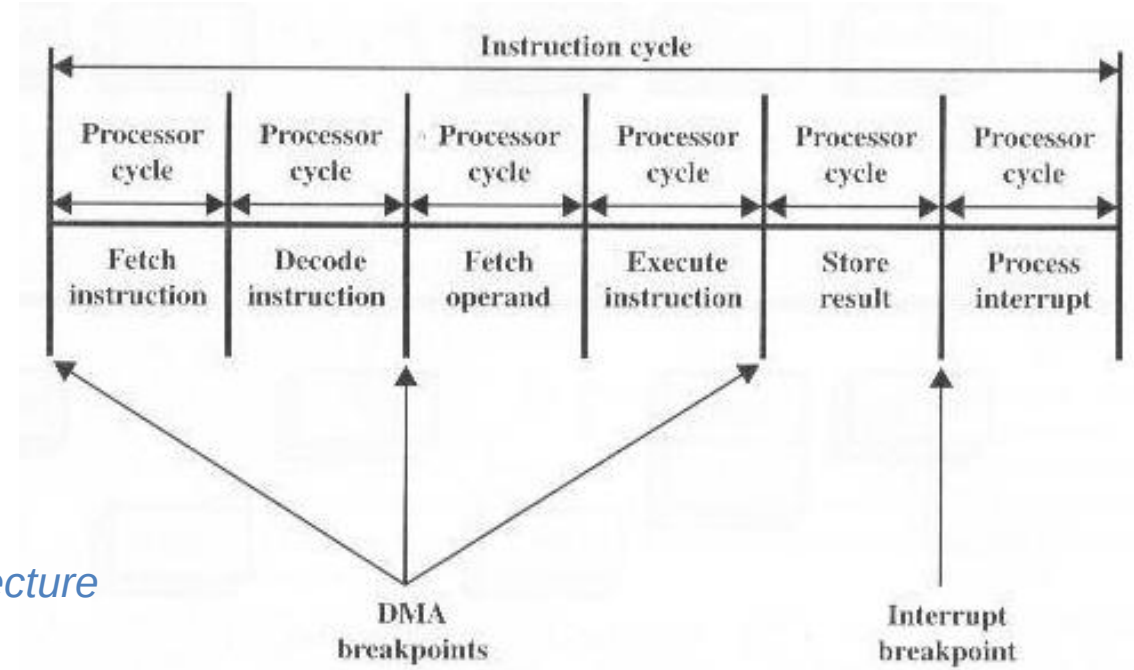


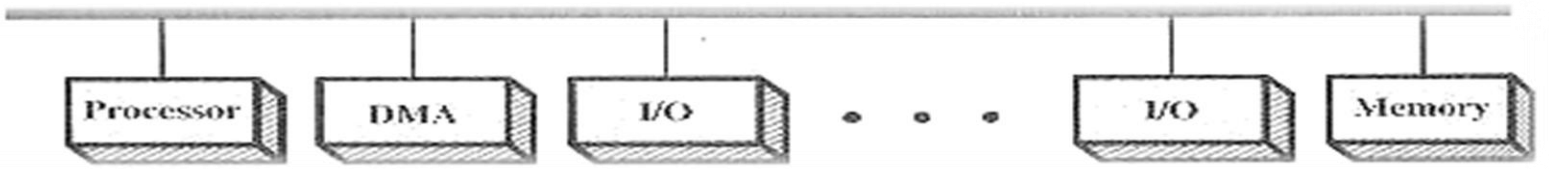
Figure 7.13 DMA and Interrupt Breakpoints during an Instruction Cycle

W. Stallings, Computer organization and architecture

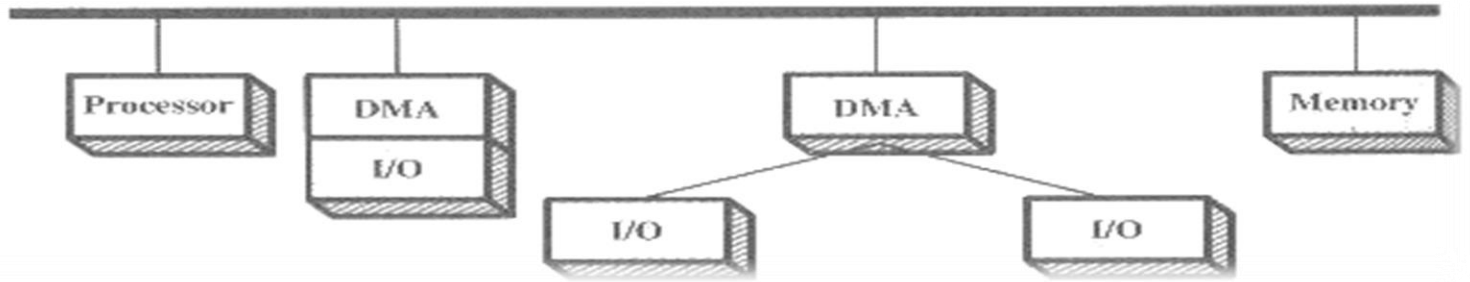
9.3 DMA discussion

At bus architecture level:

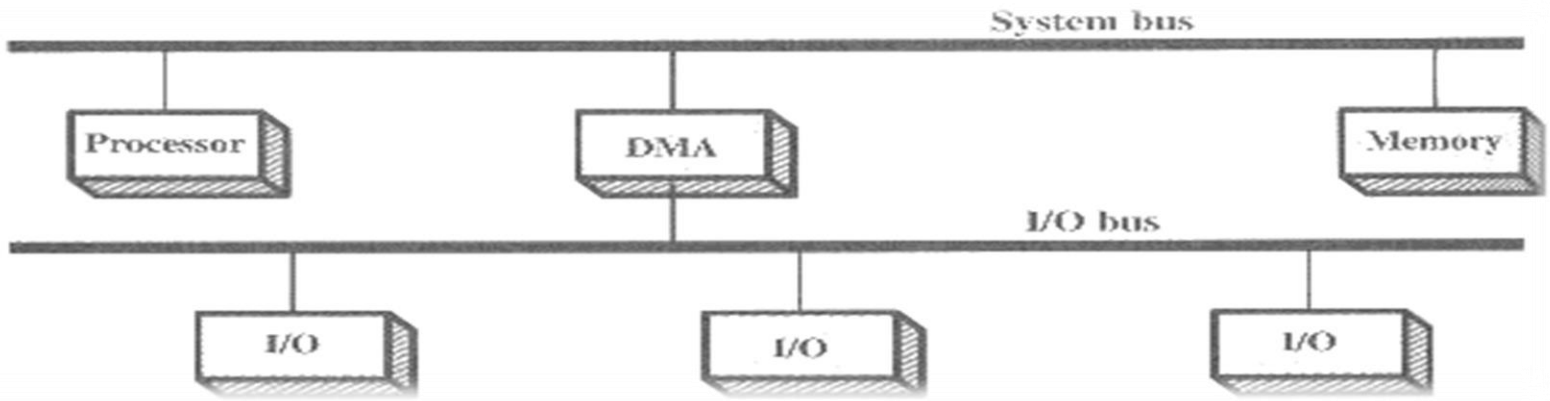
Detached DMA



Integrated DMA with peripheral.

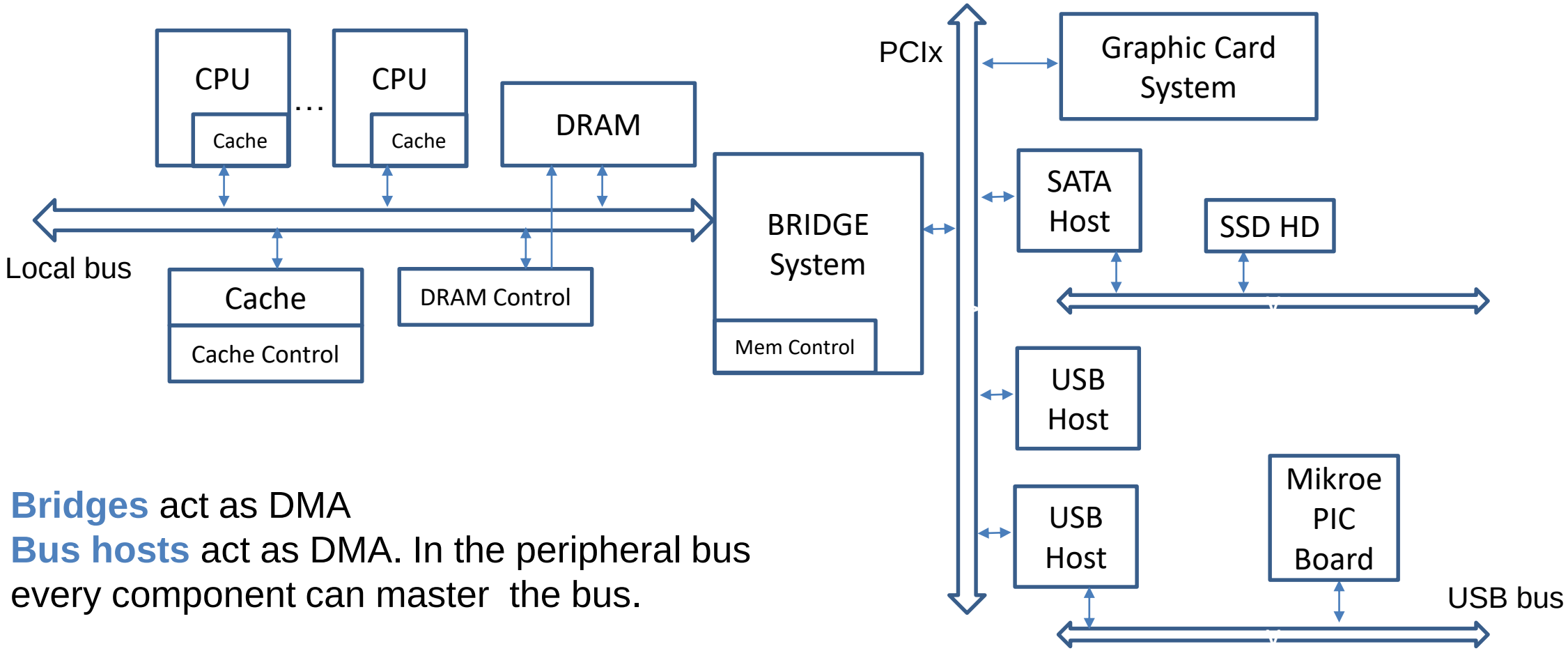


DMA mastering peripheral bus.



9.3 DMA discussion

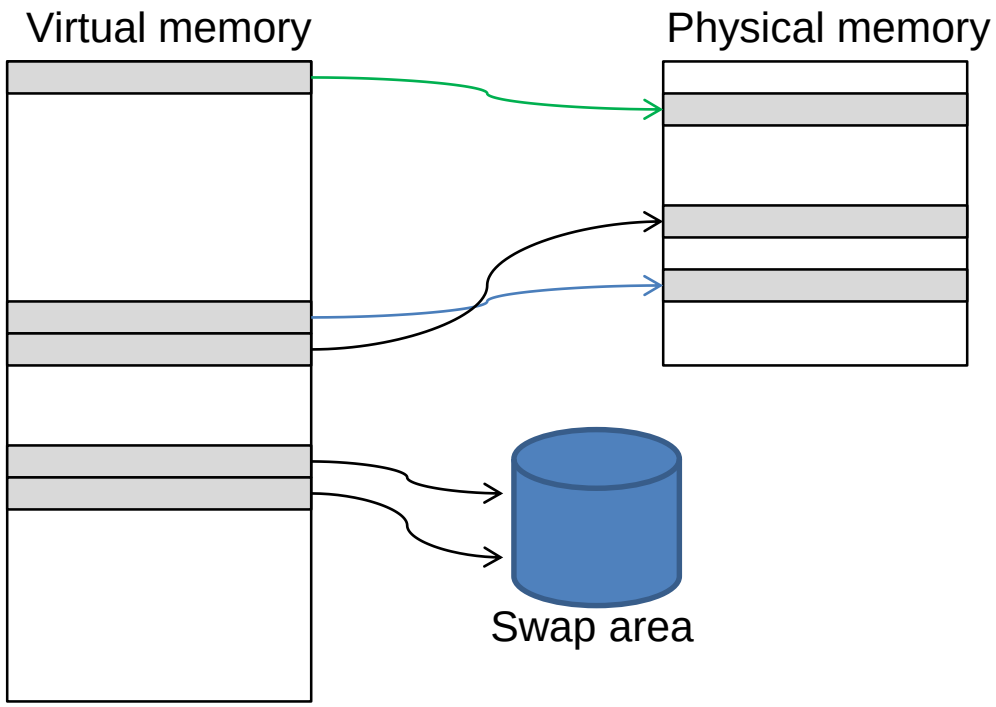
PC Example architecture



Bridges act as DMA
Bus hosts act as DMA. In the peripheral bus every component can master the bus.

9.4 DMA problems

Virtualization and Pagination



- When a transfer involves more than one page, they may have not been mapped in sequential pages in physical memory.
- If the DMA is transferring data to a page and it is moved to the swap or relocated, the DMA would be transferring to the wrong pages.
- As pages have physical and virtual addresses, DMA can be placed before o after the translation: either all DMA addresses must be translated or the programmer don't know the addresses.

9.4 DMA problems

Solution 1:

Force sizes of DMA transfers to 1 page.

Block the pages being used in a DMA transfer.

Solution 2:

Using *virtual* DMA (HW), the transfers can be programmed in virtual addresses and part of the page translation table must be connected to the DMA engine.

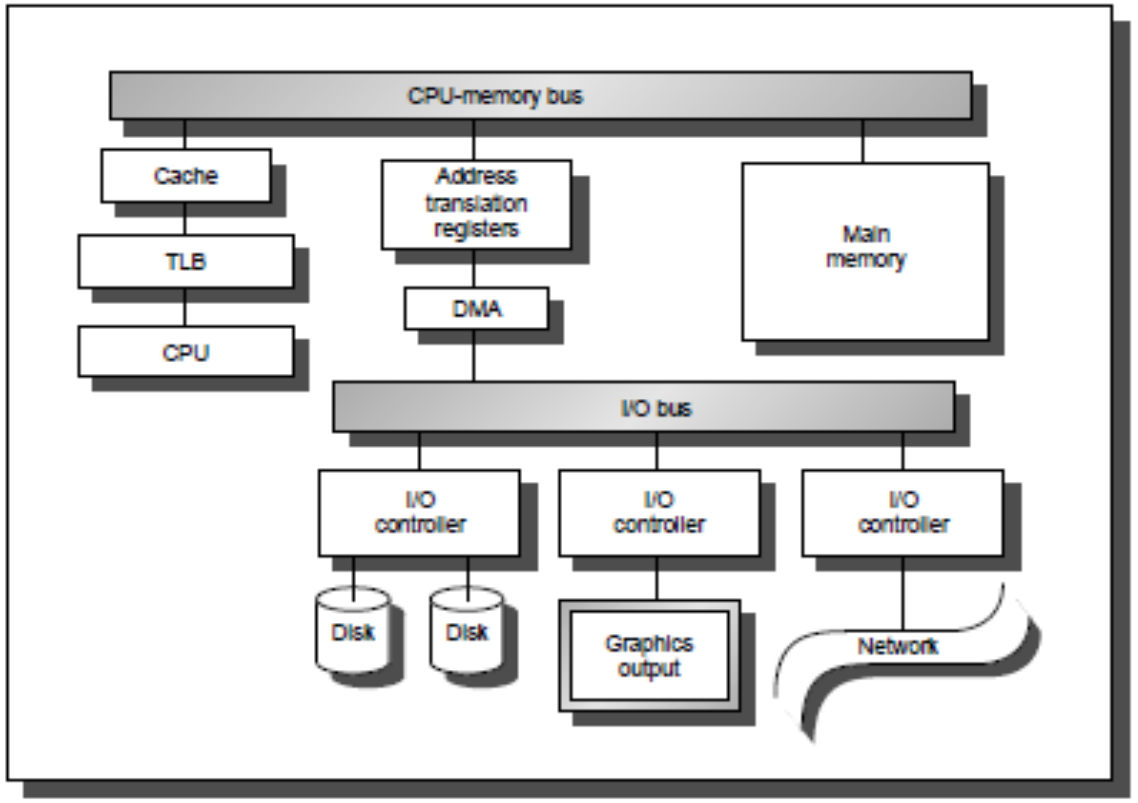
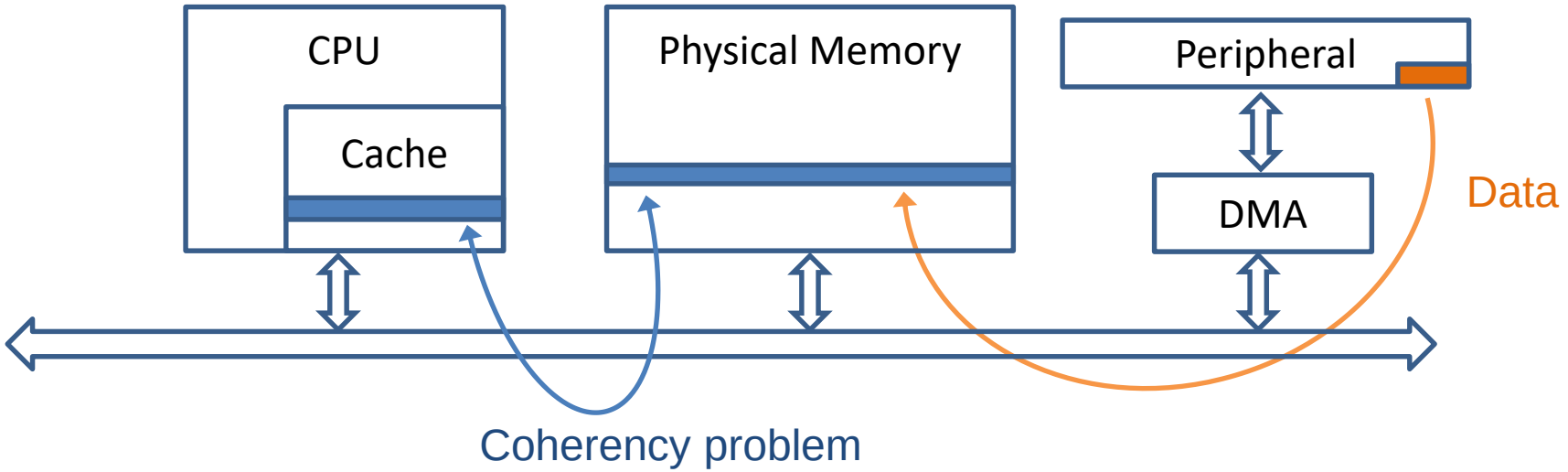


FIGURE 6.30 Virtual DMA requires a register for each page to be transferred in the DMA controller, showing the protection bits and the physical page corresponding to each virtual page.

9.4 DMA problems

Cache memory



Solution 1:

Route all the I/O activity through the cache: expensive and gives low performance because I/O data may not be the most useful to put in cache.

Solution 2:

Cache *flushing* by OS, when a page is read or written by the DMA the OS selectively invalidates the cache entries.

Solution 3:

Cache *flushing* by hardware, a specialized hardware invalidates the cache entries when a block is read or written by the DMA.