Lecture 21: Virtual Memory, I/O Basics

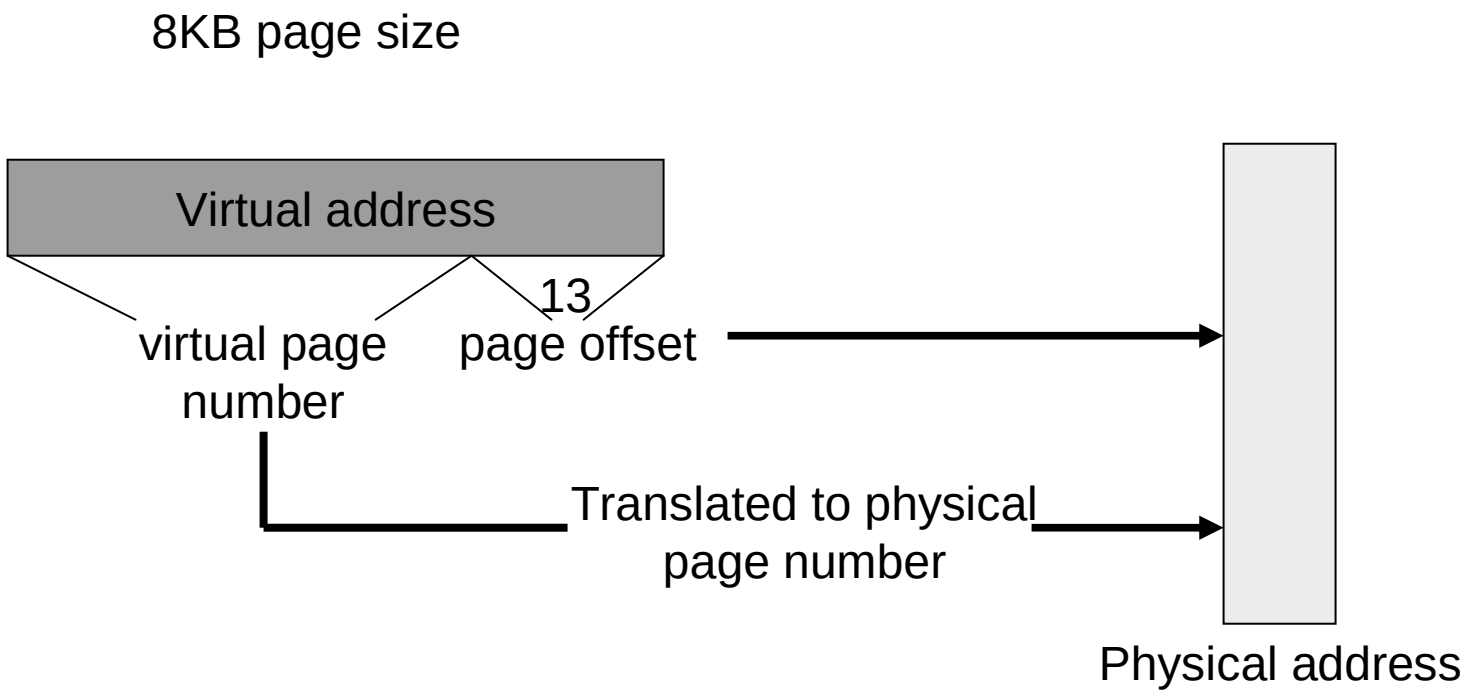
- Today's topics:
  - Virtual memory
  - I/O overview
- Reminder:
  - Assignment 8 due Tue 11/21

Virtual Memory

- Processes deal with virtual memory – they have the illusion that a very large address space is available to them
- There is only a limited amount of physical memory that is shared by all processes – a process places part of its virtual memory in this physical memory and the rest is stored on disk (called swap space)
- Thanks to locality, disk access is likely to be uncommon
- The hardware ensures that one process cannot access the memory of a different process

Address Translation

- The virtual and physical memory are broken up into pages



Memory Hierarchy Properties

- A virtual memory page can be placed anywhere in physical memory (fully-associative)
- Replacement is usually LRU (since the miss penalty is huge, we can invest some effort to minimize misses)
- A page table (indexed by virtual page number) is used for translating virtual to physical page number
- The page table is itself in memory

TLB

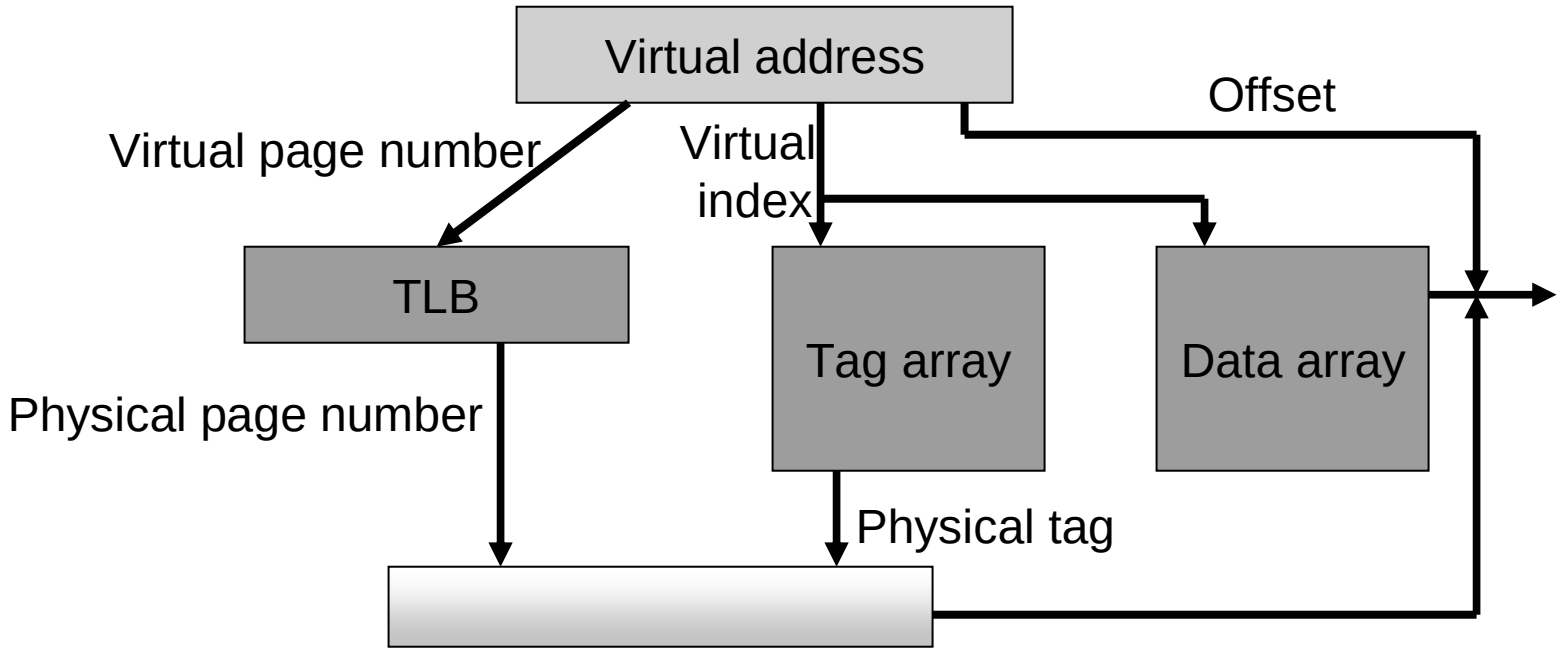
- Since the number of pages is very high, the page table capacity is too large to fit on chip
- A translation lookaside buffer (TLB) caches the virtual to physical page number translation for recent accesses
- A TLB miss requires us to access the page table, which may not even be found in the cache – two expensive memory look-ups to access one word of data!
- A large page size can increase the coverage of the TLB and reduce the capacity of the page table, but also increases memory wastage

TLB and Cache

- Is the cache indexed with virtual or physical address?
  - To index with a physical address, we will have to first look up the TLB, then the cache → longer access time
  - Multiple virtual addresses can map to the same physical address – must ensure that these different virtual addresses will map to the same location in cache – else, there will be two different copies of the same physical memory word

- Does the tag array store virtual or physical addresses?
  - Since multiple virtual addresses can map to the same physical address, a virtual tag comparison can flag a miss even if the correct physical memory word is present

Cache and TLB Pipeline

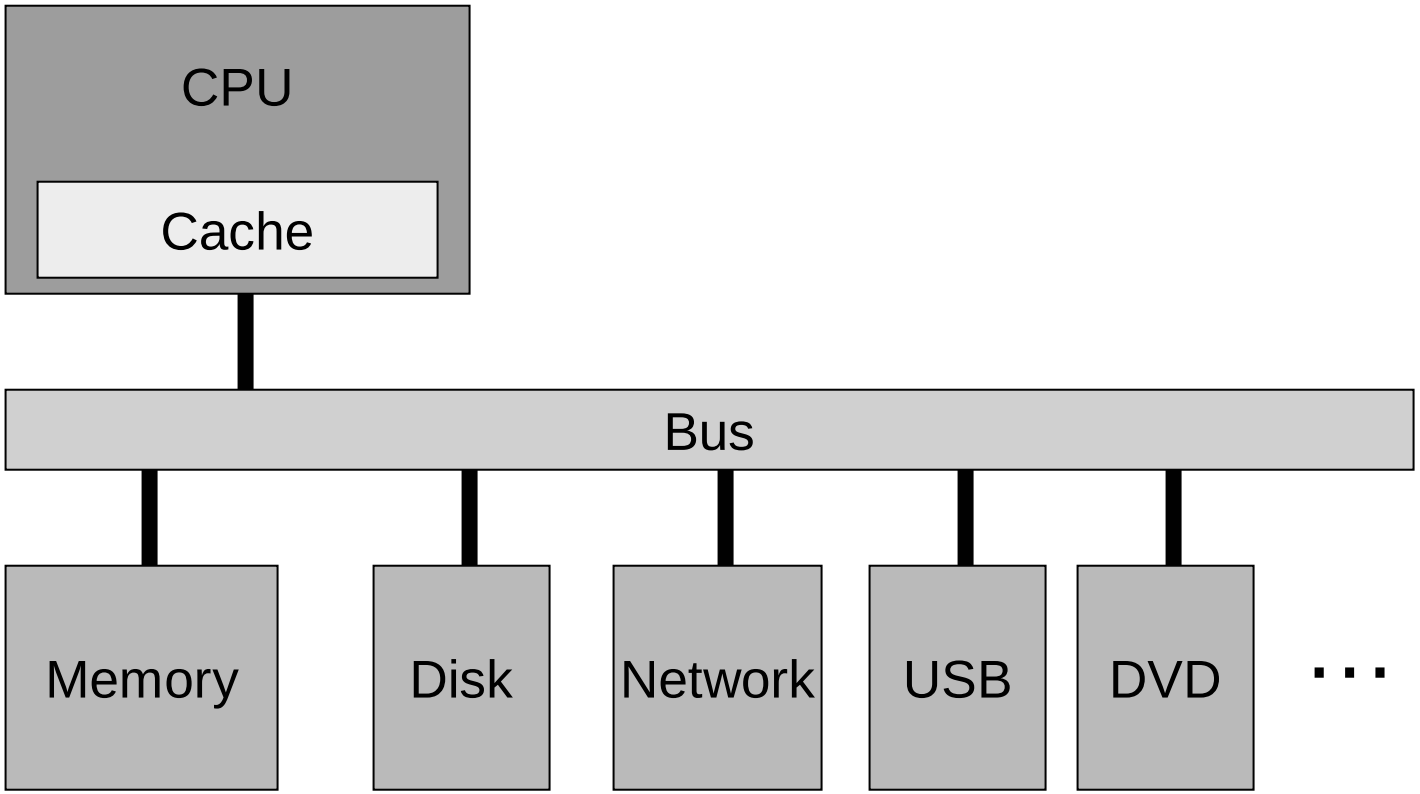


Virtually Indexed; Physically Tagged Cache

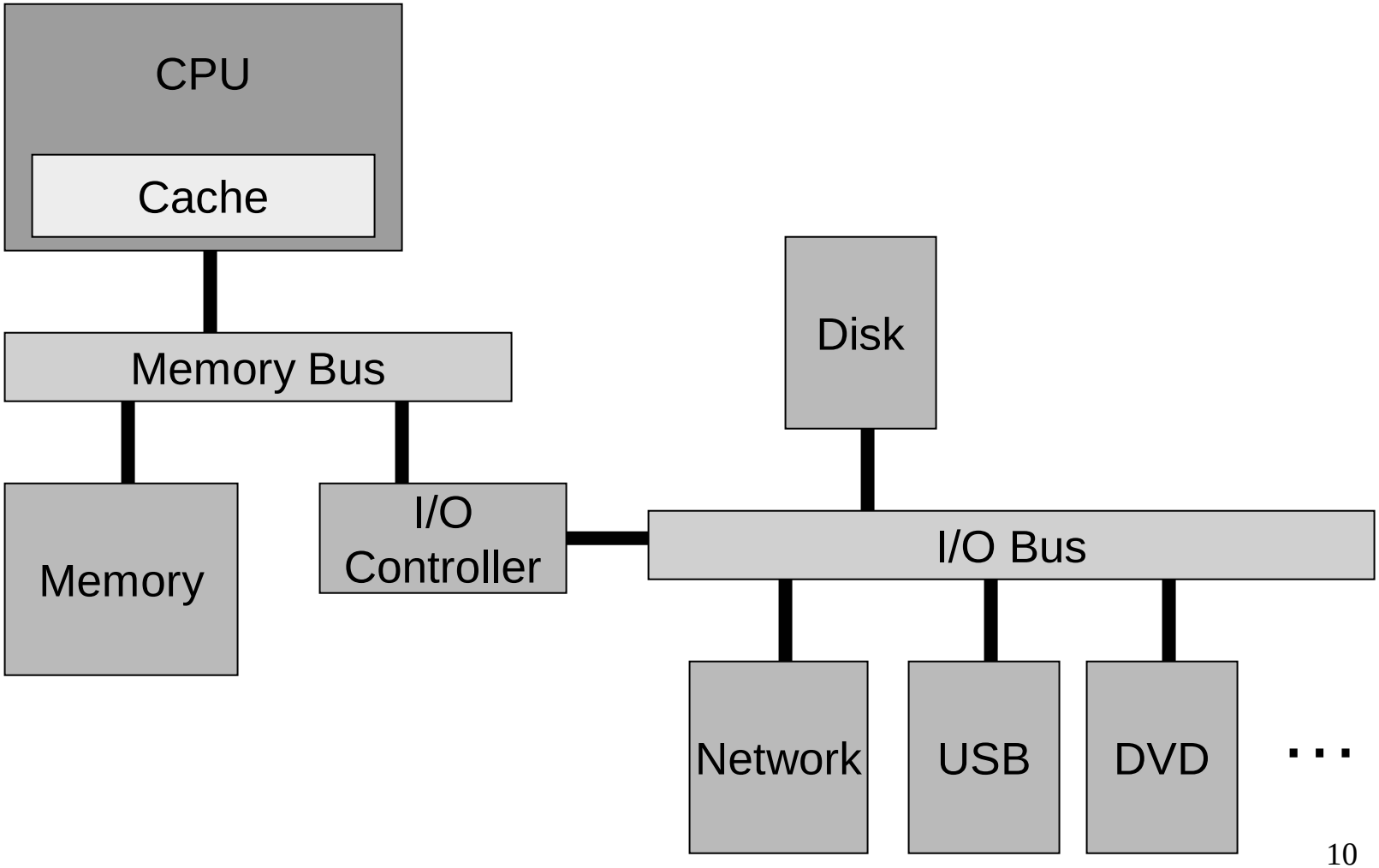
Bad Events

- Consider the longest latency possible for a load instruction:
  - TLB miss: must look up page table to find translation for v.page P
  - Calculate the virtual memory address for the page table entry that has the translation for page P – let's say, this is v.page Q
  - TLB miss for v.page Q: will require navigation of a hierarchical page table (let's ignore this case for now and assume we have succeeded in finding the physical memory location (R) for page Q)
  - Access memory location R (find this either in L1, L2, or memory)
  - We now have the translation for v.page P – put this into the TLB
  - We now have a TLB hit and know the physical page number – this allows us to do tag comparison and check the L1 cache for a hit
  - If there's a miss in L1, check L2 – if that misses, check in memory
  - At any point, if the page table entry claims that the page is on disk, flag a page fault – the OS then copies the page from disk to memory and the hardware resumes what it was doing before the page fault … phew!

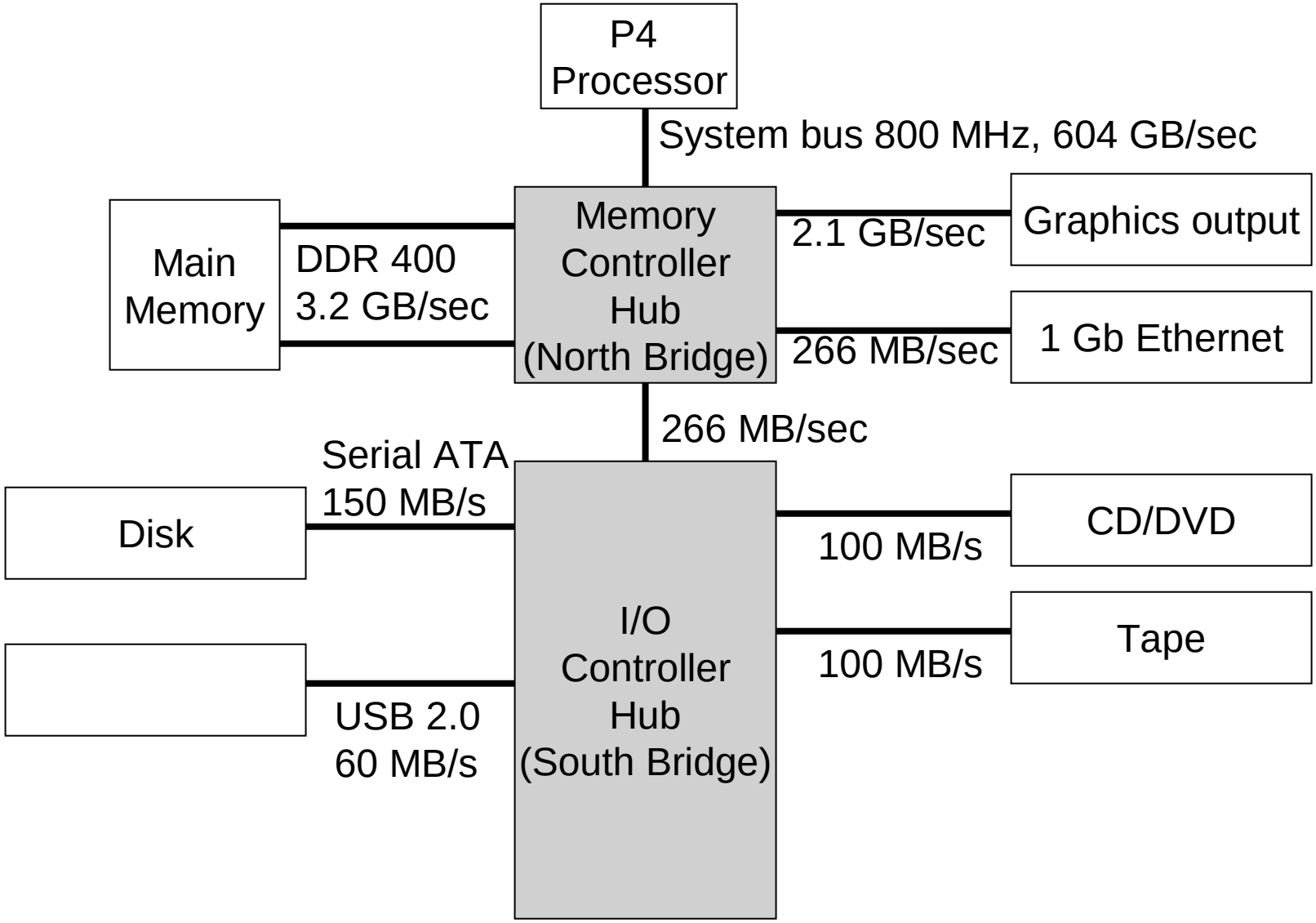
Input/Output



I/O Hierarchy



Intel Example



Bus Design

- The bus is a shared resource – any device can send data on the bus (after first arbitrating for it) and all other devices can read this data off the bus
- The address/control signals on the bus specify the intended receiver of the message
- The length of the bus determines its speed (hence, a hierarchy makes sense)
- Buses can be synchronous (a clock determines when each operation must happen) or asynchronous (a handshaking protocol is used to co-ordinate operations)

Memory-Mapped I/O

- Each I/O device has its own special address range
  - The CPU issues commands such as these:
    sw [some-data] [some-address]
  - Usually, memory services these requests… if the address is in the I/O range, memory ignores it
  - The data is written into some register in the appropriate I/O device – this serves as the command to the device

Polling Vs. Interrupt-Driven

- When the I/O device is ready to respond, it can send an interrupt to the CPU; the CPU stops what it was doing; the OS examines the interrupt and then reads the data produced by the I/O device (and usually stores into memory)

- In the polling approach, the CPU (OS) periodically checks the status of the I/O device and if the device is ready with data, the OS reads it

Direct Memory Access (DMA)

- Consider a disk read example: a block in disk is being read into memory

- For each word, the CPU does a
  lw [destination register] [I/O device address] and a
  sw [data in above register] [memory-address]

- This would take up too much of the CPU's time – hence, the task is off-loaded to the DMA controller – the CPU informs the DMA of the range of addresses to be copied and the DMA lets the CPU know when it is done

Title

- Bullet