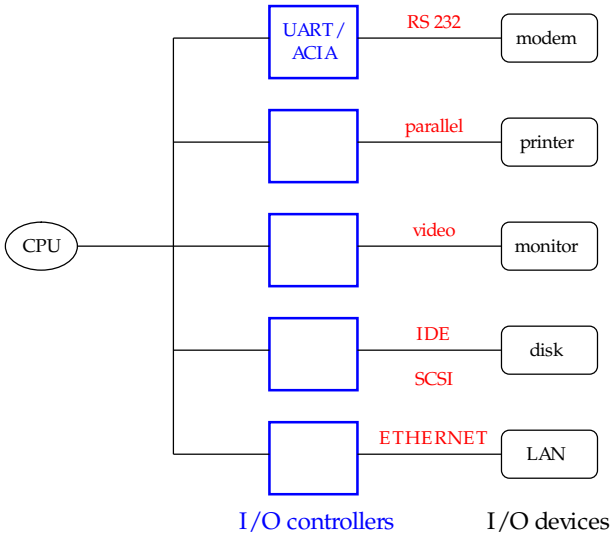
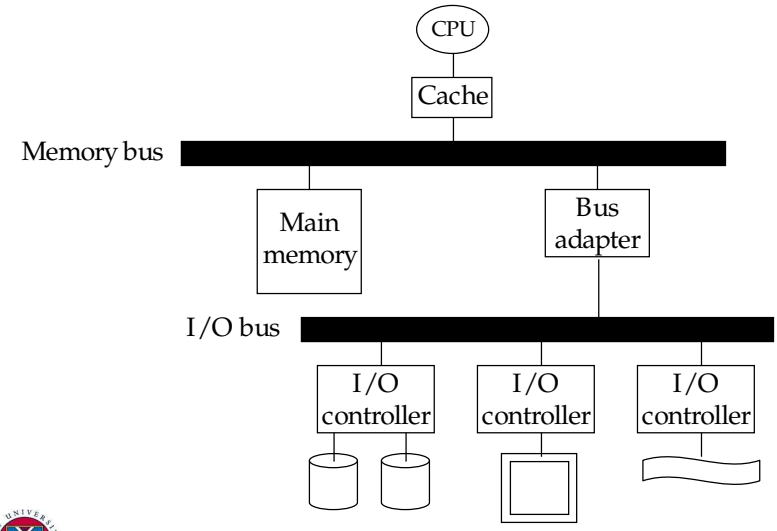
Lecture 14: I/O controllers & devices

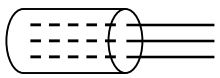


Computer system organization



Example: RS232 serial interface

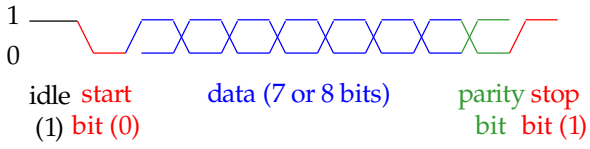
- I/O controller: UART (Universal Asynchronous Receiver Transmitter) or ACIA (Asynchronous Communications Interface Adapter)
- Used for modems and other serial devices
- Physical Implementation:
  - 2 signal wires (one for each direction) + ground reference



Tx data wire (CPU to I/O device)
Rx data wire (I/O device to CPU)
Ground (voltage reference) wire

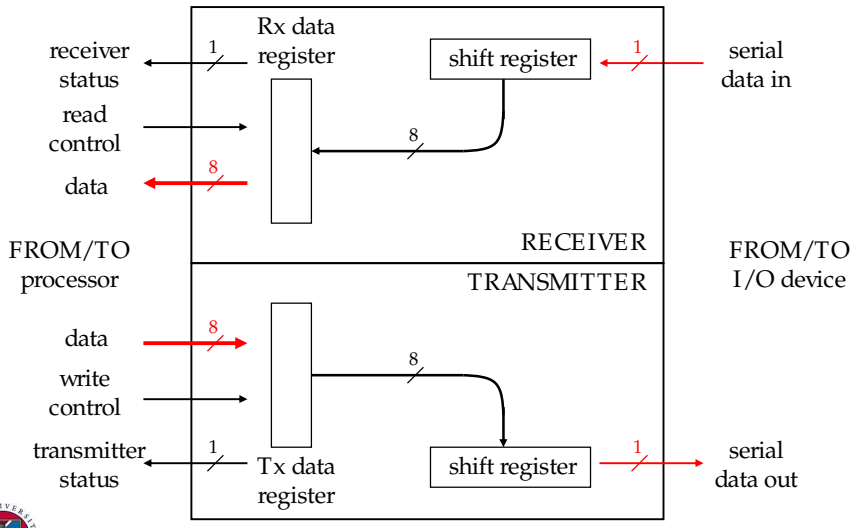
Serial transmission

- Encoding:
  - 1 character = 10 or 11 bits (including signaling)
  - Idle state is represented by a constant "1"



- Parity: for detection of transmission errors
  - odd → total number of "1"s (including parity bit) is odd
  - even → total number of "1"s is even

UART controller



Connecting CPU and I/O controllers

- Option 1: connect the I/O Tx and Rx registers directly into some special CPU I/O registers → not flexible
- Option 2: keep I/O registers in separate I/O controller and connect CPU to I/O controller through special I/O bus → expensive, not flexible

  I/O bus:
  - data lines (8 bits)
  - control lines (READ and WRITE signals),
  - address lines (a few bits) → each I/O controller is assigned a range of addresses for its registers

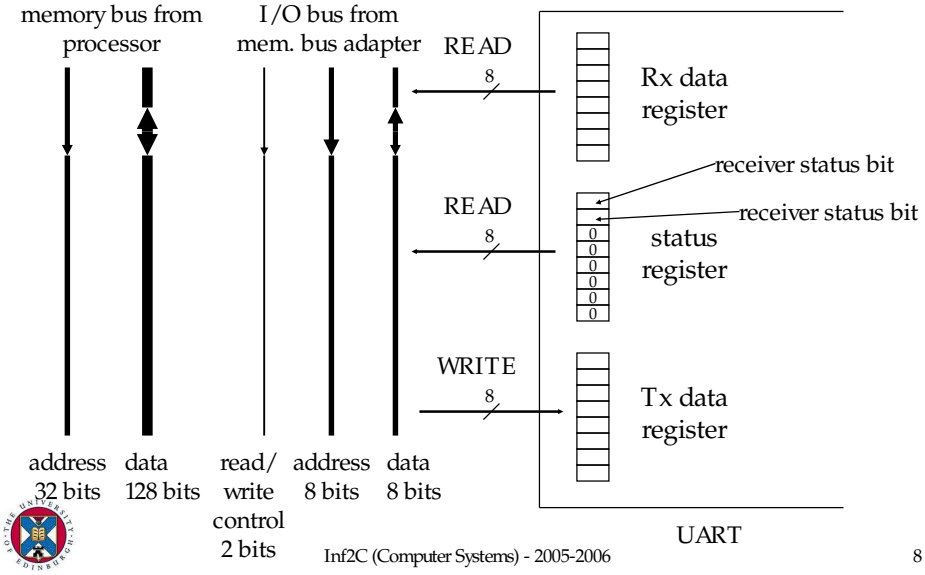
  Data is accessed through special I/O loads and stores

Connecting CPU and I/O controllers

- Option 3: keep I/O registers in I/O controller and connect CPU to I/O controller through memory bus

  Memory mapped I/O:
  - I/O controller registers (data and control) are mapped to a dedicated portion of memory and are accessed with regular load and store instructions
  - Takes bus bandwidth away from CPU-memory
- Option 4: connect I/O controllers to I/O bus and the I/O bus to the memory bus through a bus adapter
  - Off-load memory bus (multiple I/O devices appear as a single device to the memory bus)

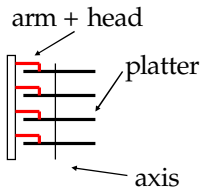
I/O via I/O or memory bus

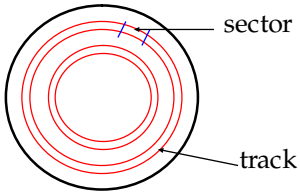


Polling and interrupt –based I/O

§ Option 1: Polling
  – User process calls OS at regular intervals to check status of I/O operation
  – Time-consuming
  – Used in embedded systems
§ Option 2: Interrupt
  – I/O controller interrupts user process to signal an I/O event
  – Used in general purpose systems

Disks





§ 100's tracks per surface for hard disk
  10's sectors per track
  512b – 2Kb per sector
§ Spinning speed:
  – 360 rpm for floppy disk
  – 5400–15000 rpm for hard disk

Disk performance

- Total time of a disk operation is divided in two parts:
- Access time: time to get head into position to read/write data

  access time = seek time + rotational latency
  - Seek time: time to move head to appropriate track
  - Rotational latency: time to wait for appropriate sector to arrive underneath the head
- Transfer time: time to move data to/from disk

  transfer time = time to transfer 1 byte * number of bytes of data
  - Spinning speed and recording density
  - 500Kbit/s for floppy; 20-100Mbit/s for hard disk

Disk controllers

- Disk controller inside disk unit → responsible for all mechanical operation of disk + interface with CPU
- Disk interface standards
  - EIDE: simple, bus structure is similar to I/O or memory bus
  - SCSI: flexible, requires special bus controller to connect to I/O or memory bus
- I/O registers:
  - Exchange data and control between CPU and controller
  - Command register → tells controller what to do next

    e.g. Seek n

    Read Sector m

    Write Sector m

Using a disk controller

- Step 1: user program requests data from a file
- Step 2: OS file system determines sector(s) to be accessed
- Step 3: OS disk handler issues Seek command and CPU goes to work on some other process (multi-tasking)
- Step 4: I/O controller interrupts CPU to signal completion of seek
- Step 5: OS disk handler issues Read Sector command and CPU goes to work on some other process
- Step 6: I/O controller interrupts CPU to signal data ready
- Step 7: OS disk handler transfers data to/from disk
- Step 8: go to step 3 or 5 and repeat until all data transferred

Interrupt approach

- Program (through OS) has to issue individual commands to read every sector from disk
- Data transfer is very slow (~100µs for a 512 byte sector)
- Interrupt mechanism is very expensive
- Time taken by interrupt mechanism makes it difficult to synchronize head position with Read Sector command

- Solution: Direct Memory Access (DMA)

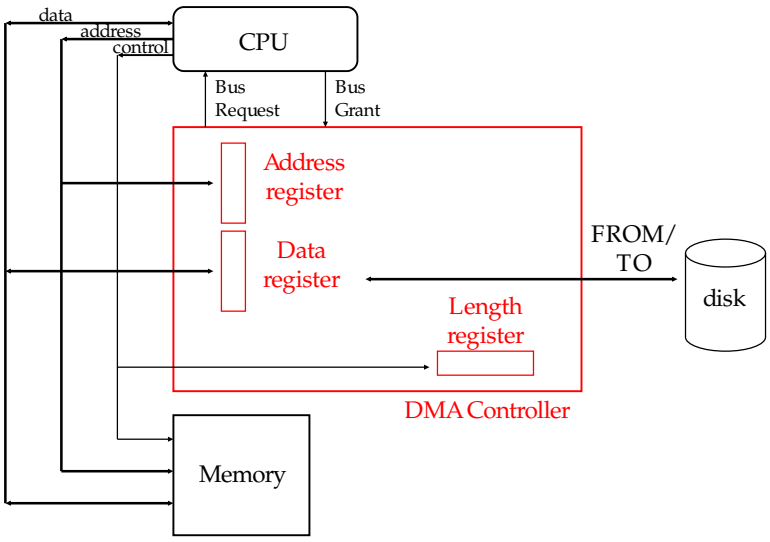
Direct memory access

- DMA controller: sits on the memory bus and can independently transfer data to/from memory from/to disk
- DMA registers:
  - Address register → position in memory of next data to be read/written
  - Data register → temporary storage for data to be transferred
  - Length register → number of bytes remaining to be transferred

DMA operation

- Step 1, 2: user program requests data, OS determines location of data on disk
- Step 3: OS disk handler issues Seek command and sets up DMA registers (address, length); CPU goes to work on another process
- Step 4, 5: I/O controller interrupts CPU, OS disk handler issues Read Sector command
- Step 6: I/O controller informs DMA controller that data is ready (no need to interrupt CPU)
- Step 7: DMA controller transfers data into memory; length register is decremented until all data is moved (advanced DMA controllers can access multiple tracks with a single operation)
- Step 8: DMA controller interrupts CPU to inform completion of DMA operation

DMA organisation



Bus arbitration

- DMA and CPU connect to memory bus → access must be somehow arbitrated to avoid conflicts
- Solution: additional logic (bus arbiter)
  - Authorizes CPU or DMA controller to access memory at any given time
- 2 new wires on memory bus:
  - Bus Request → asserted by DMA when it requires the bus
  - Bus Grant → asserted by the CPU when it is not using the bus and thus DMA can use it
  - In case of conflicting requests in the same cycle, CPU usually has priority