INF-1100
I/O

Åge Kvalnes

University of Tromsø, Norway

November 7, 2014

Input/Output

Input:

- Transfer data into the computer.
- Example devices: Keyboard, disk, mouse, etc.

Output:

- Transfer data from the computer and to some external device.
- Example devices: Disk, floppy, USB stick, monitor, etc.

I/O devices

I/O devices contain two components:

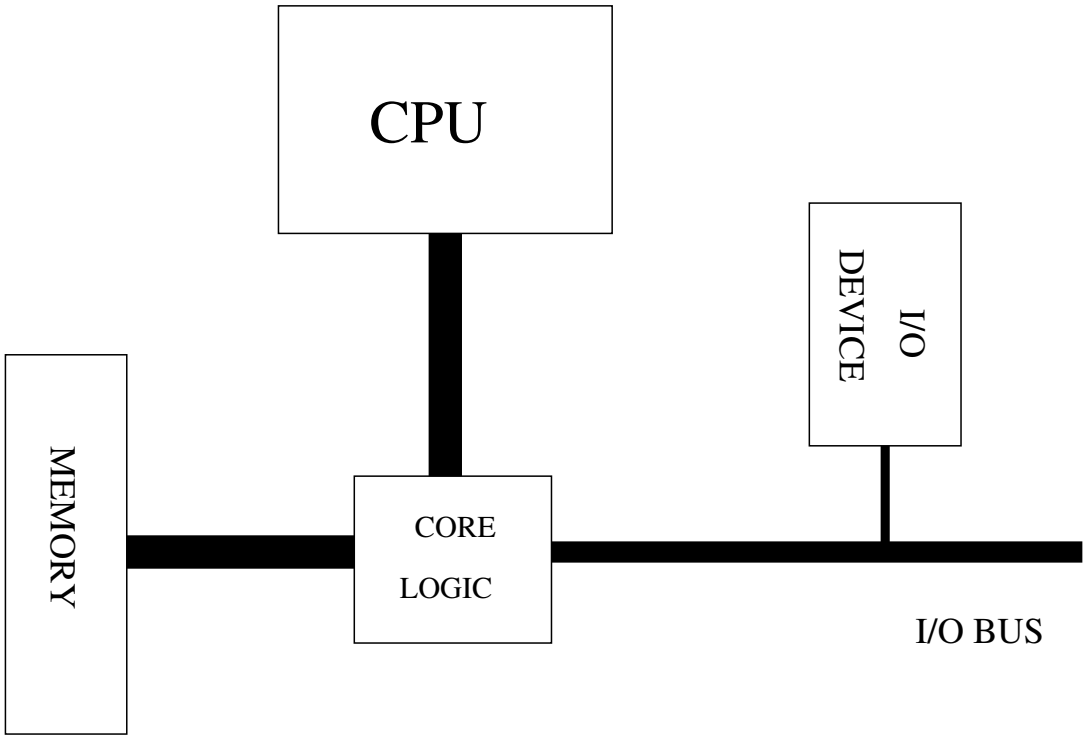
I/O controller:

- Functions as a proxy between the CPU and the actual device electronics (++).
- Exports a register-based programming interface.
  - CPU writes to the registers to tell the device what to do.
  - CPU reads from the registers to check whether a task has been performed.

Device electronics:

- Performs the actual operations.
  - Sense mouse movements.
  - Store/retrieve data from disk.
  - Etc.

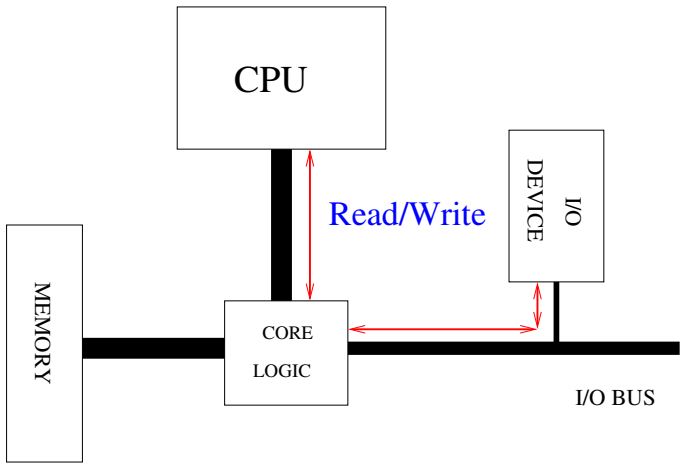
Connecting I/O devices



Accessing I/O device registers: Memory mappped I/O

Memory mapped I/O

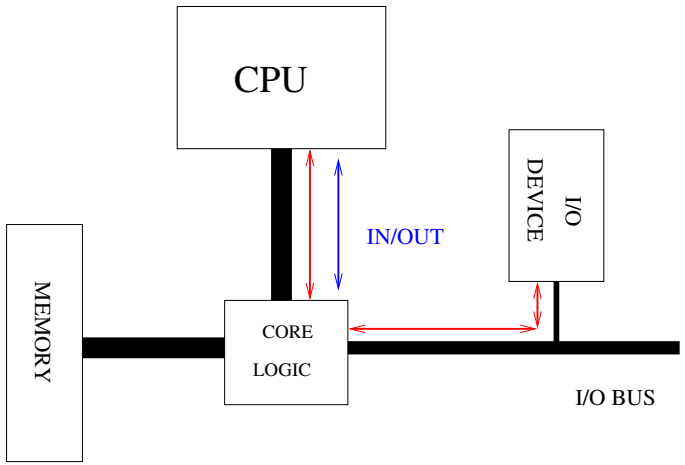
- I/O device registers are mapped into the normal address space.
- Normal load/store instructions can be used for communication.
  - Pro: No extra logic required in the CPU.
  - Con: Reduces effective size of normal address space.



Accessing I/O device registers: I/O instructions

I/O instructions:

- Special instructions are used for communication with I/O device.
  - Pro: Does not reduce effective size of normal address space.
  - Con: Requires special logic in the CPU. Requires extra signal line from CPU.



Synchronous vs Asynchronous

I/O devices generally operate at a much lower speed than the CPU:

- CPU: $3 - 15$ billion instructions per second.
- Disk: 320MB per second.
  - Assuming data is read/written via 16 bit register, the CPU must read/write at a rate of $160M$ ops/s $\Rightarrow$ approx once every $18 - 100$ cycle.
- Network: 125MB per second.
  - Assuming data is read/written via 16 bit register, the CPU must read/write at a rate of $62M$ ops/s $\Rightarrow$ approx once every $48 - 241$ cycle.

In practice, I/O devices are not designed to accept or emit data at a specific rate. Rather, they work in an *asynchronous* fashion (i.e. not in lockstep with the CPU).

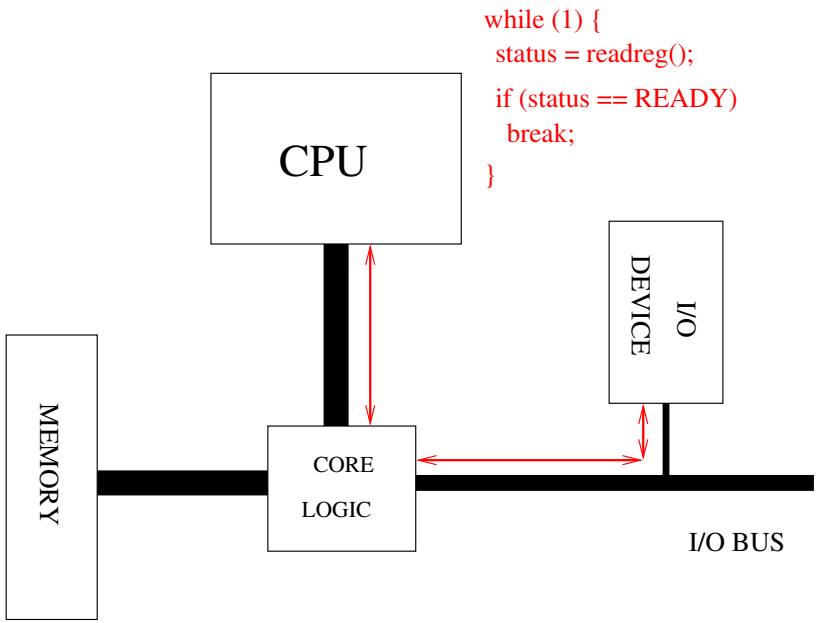
A synchronization protocol is used to tell the CPU when it can read data from or write data to the I/O device.

When is I/O done? Polling

The CPU determines when it can read data from or write data to the device by continuously polling (reading) a device register.

Polling:

- The value of this register tells if the I/O device is ready to accept or emit data.
- Con: The result of the poll may be negative ⇒ CPU cycles are wasted.

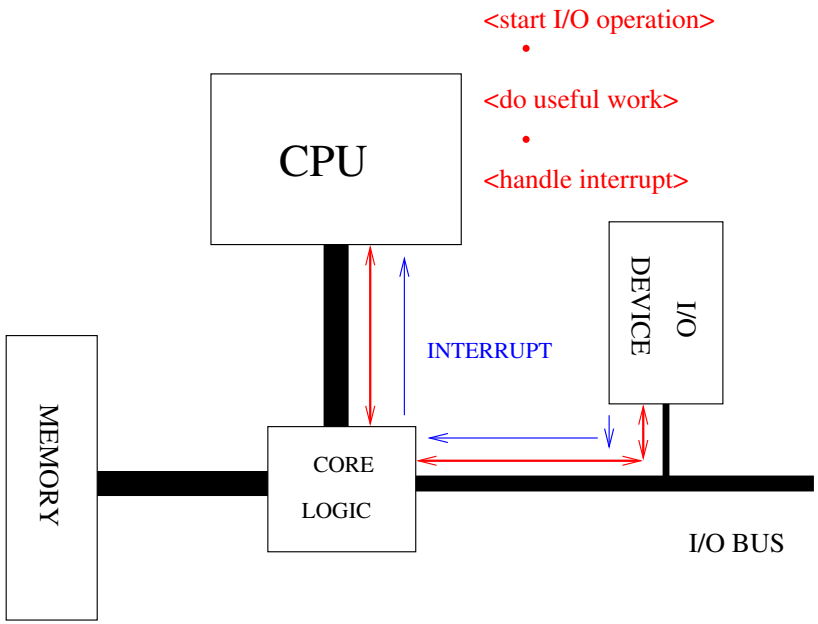


When is I/O done? Interrupts

The device informs the CPU that data can be read or written by sending an interrupt.

Interrupts:

- Pro: The CPU can perform other tasks while waiting for the I/O device to be ready.

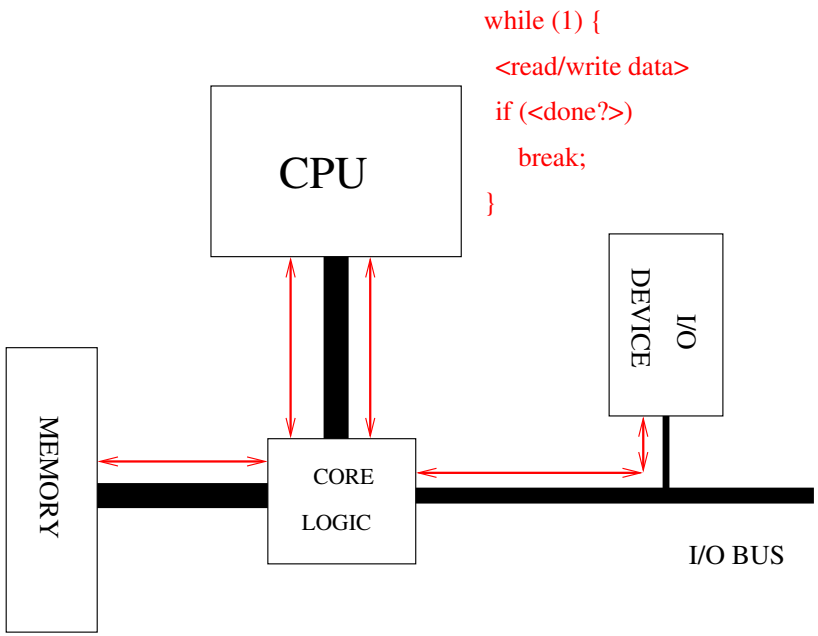


Who transfers data to/from memory?

Transferring data: Programmed I/O

Programmed I/O:

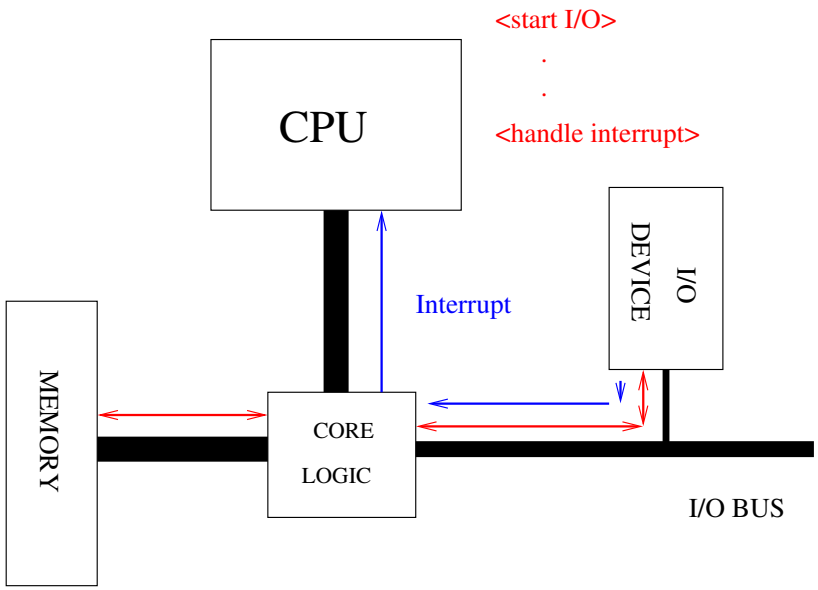
- The CPU transfers data between the I/O device and memory.
- Con: The CPU has other things to do.



Transferring data: Direct Memory Access

Direct Memory Access (DMA):

- The I/O device transfers data to/from memory.
- When transfer is done, CPU is signaled via interrupt.
- Pro: The CPU can do other things while data is transferred from/to memory.



Most modern I/O devices use/support DMA.

Wordlist

Problem: Read text from a set of files and insert words into a list. Words should not be repeated in the list. It should be possible to determine what files contain a specific word.

Interface:

- **void wordlist_lookup(list_t *wordlist, char *wordstring)**

C library file interface

- Create new or open existing file.
  - **FILE* fopen(char *path, char *mode);**
- Close open file.
  - **int fclose(FILE *file);**
- Read/Write data from/to file.
  - **size_t fread(void *buf, size_t size, size_t nmemb, FILE *file);**
  - **size_t fwrite(void *buf, size_t size, size_t nmemb, FILE *file);**
- Delete file, create directory, delete directory same as Unix interface.

C library uses OS interface, but maintains an internal buffer for each file to reduce the number of OS calls. OS is invoked when a call cannot be served from the buffer.

Operating system file interface (Unix)

- Create new or open existing file.
  - **int open(char *path, int flags, ...);**
- Close open file.
  - **int close(int file);**
- Read/Write data from/to file.
  - **ssize_t read(int file, void *buf, size_t nbytes);**
  - **ssize_t write(int file, void *buf, size_t nbytes);**
- Delete file.
  - **int unlink(char *path);**
- Create directory
  - **int mkdir(char *path, mode_t mode);**
- Delete directory.
  - **int rmdir(char *path);**