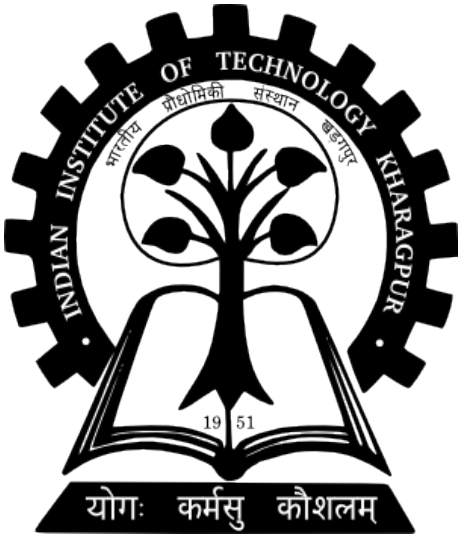


Indian Institute of Technology Kharagpur

Department of Computer Science and Engineering

CPU's I/O with Peripheral Devices

Hardik Soni
20CS30023

Assignment 2

Computer Organization and Architecture (CS31007)

September 18, 2024

Contents

1 Problem Statement

- Input/output is an important aspect of most computation tasks I/O must be done via programming the CPU.
- What is the architectural view of an i/o device for a programmer?
- How is the architectural view supported organisationally? Explain with the help of a diagram.
- Consider any simple i/o task (such as reading from a keyboard, reading or writing a block of data to the disk) and explain how that would be achieved in your framework.

2 Abstract

For the solution I have assumed that the CPU has direct access to motherboard's BIOS and RAM.

On modern architectures, peripherals are accessed via mapped memory addresses on a bus. It's more a way to address individual peripherals, of which memory (RAM/DDR) is just one type. For example, you might have 2GB of RAM at addresses 0x00000000.. 0x7fffffff. The bus controller (PCIe or whatever) knows which address ranges go to which peripheral.

Memory is usually special in that it can be cached, so individual reads/writes to memory tend not to translate directly to individual reads/writes to the RAM chips. Peripherals are marked as special - CPU accesses should go out to the peripheral exactly as written in your program.

The language you talk to peripherals with is pretty much ad-hoc depending on the device and the programmer's choice. The general theme is that the peripheral is mapped somewhere in memory (e.g 0x80000000 for a few KB as above), with individual bit of state and actions controlled by different words (usually 32 or 64bit).

3 Programmed I/O

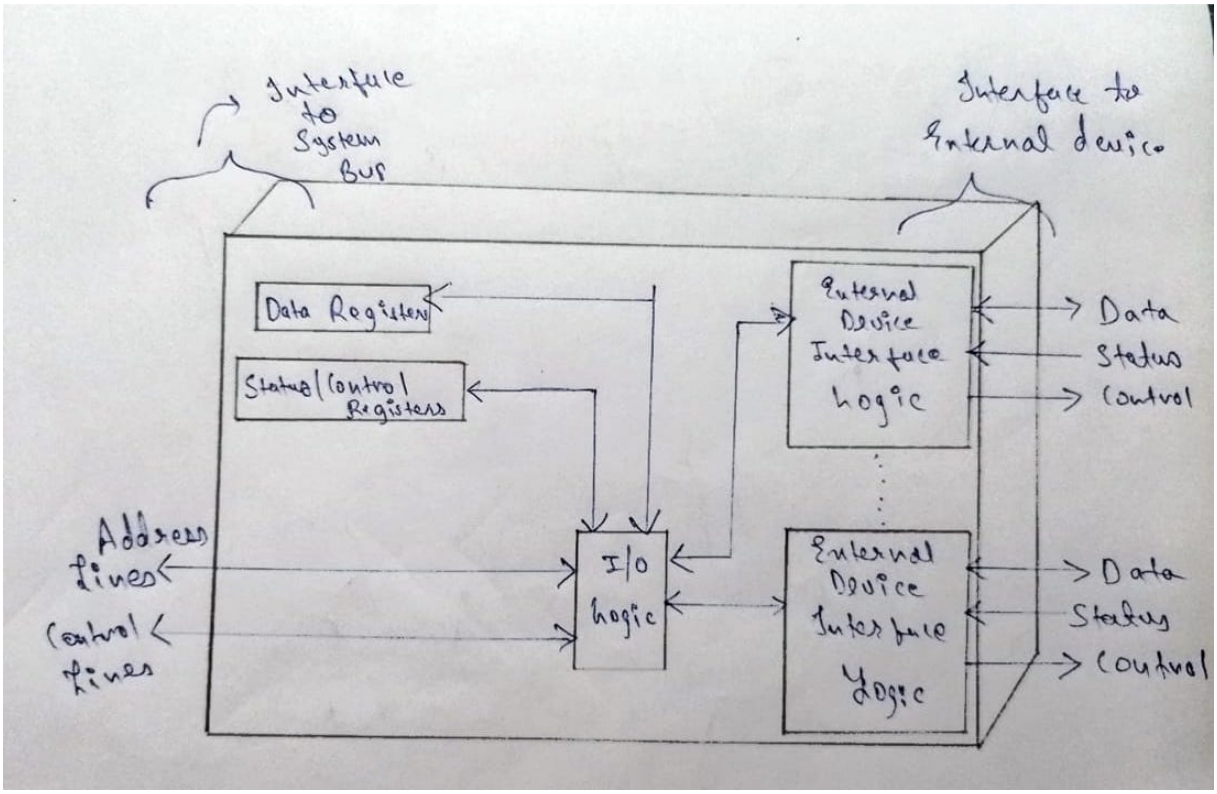
3.1 Architectural View

3.1.1 What is the architectural view of an i/o device for a programmer?

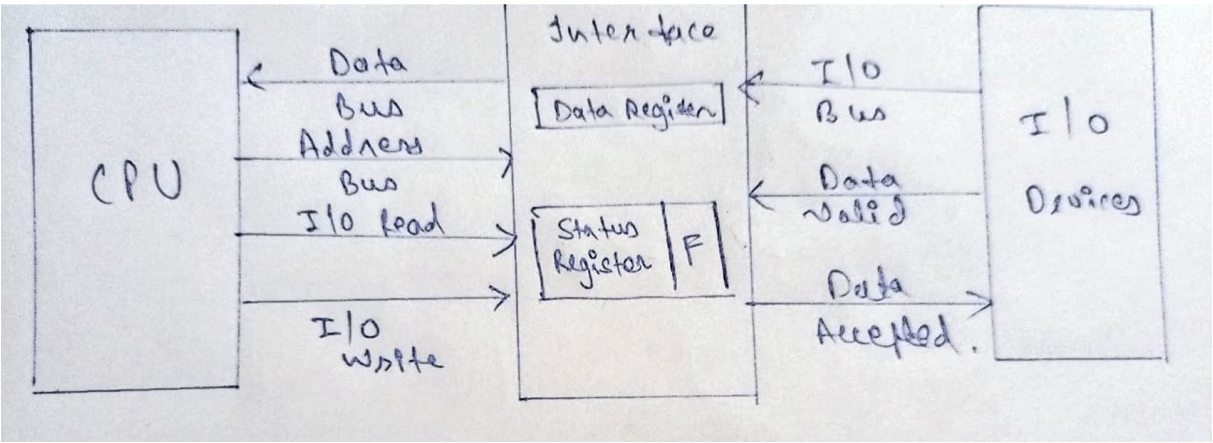
Each I/O Module is assigned a set of appropriate bits of I/O status registers. The Bus Structure for each I/O Device is designed using certain number of registers. It is further important to note that the I/O Module does not notify the Processor about whether or not it has performed the desired task. Moreover, it's the processors' responsibility to periodically check the status of the I/O module till it finds that the I/O has successfully completed the desired task.

1. Issue Read Command to I/O Module(CPU -> I/O)
2. Read Status of I/O Module (I/O -> CPU)
3. Check Status:
   (a) **Ready**: Move to the Next Step in the Procedure.
   (b) **Not Ready**: Move to Step 2.
   (c) Else Report **Compilation Error**.
4. Read Word from I/O
5. Write to Memory
6. Check is status of procedure is Done, then End or **Wait** till the next Command.

3.2 Organisational Support



Organisation Implementation



Data Transfer from I/O to Device

How is the architectural view supported organisationally? Explain with the help of a diagram. The Buses are common for both I/O Peripheral devices and the Processor, implying that the same set of instructions work for both I/O and memory. The address translation is handled by the corresponding hardware. As shown above the Address Lines, Control Lines and the I/O buses work in common and are parallel respectively, if we refer to the Explanation of the Part 1, the Control Flow explained can be emulated as follows:-
Note:- The Status of the Process is determined by the Flag Bit F as mentioned in the Figure 2.

1. The Request I/O call is made by the I/O Peripheral Devices through the I/O Bus.
2. The System call is read and sent to the Processor via Data Bus and Decrypted and check if it can satisfy request or Not.
3. It then send the Request to device driver, and blocks some process if needed. It then Processes the Request, Issues commands to the Controller, and configure it to be in place still till interrupted otherwise.
4. The Processor(if finds appropriate I/O Device) issues Controller Commands and monitor whether or not the Issued process has been completed. And generate an interrupt appropriately, if needed.
5. It then understands the Interrupt generated via the Controller, Store the data in device-driver buffer. If an input, generate a control signal to unblock the device's driver.
6. Determine which I/O stands completed via the flag bit (F) explained earlier, and indicate further state change to the I/O Subsystem(if needed).
7. Transfer data (if appropriate) to process, return completion control signal or the error control signal as the case may be.
8. I/O Process Completed, Input Data Available and Stored in memory, or is output is displayed via available means of the I/O Device.

3.3 Example

3.3.1 Consider any simple i/o task (such as reading from a keyboard, reading or writing a block of data to the disk) and explain how that would be achieved in your framework.

We consider that our I/O Device is Keyboard, and we intend to write "CRM" via the device to the: The letters 'C', 'R' and 'M' are read sequentially, and then stored in memory in consecutive 1-bit addresses:-

- First the CPU issues the Read Command to the I/O Module for itself. We initiate the flow of instruction.
- The Keyboard makes the Request I/O call over the I/O Bus for reading from it. The system call is read, then transmitted to the processor via the data bus, where it is decrypted and checked to see if it can fulfill the request or not.
- The Interface receives these 2 control signals and alerts the keyboard to get ready to receives input from the user, in addition to this it set the status bit to false initially.
- The keyboard's device driver is then notified, and if necessary, a process is blocked. Once the keyboard receives an input from the User it sends the data to the Interface into the Data Reg and alerts the Interface that it has received the first input by sending true into the I/O bus from the Keyboard to the Interface.
- The Request is subsequently processed(CPU has been looping and checking if the status bit is set to true, once it sees this it understands that the keyboard has finished taking the first part of the input and transfers the data stored in the Data Reg to the CPU for further processing and to send it to the memory for storage), directives are sent to the Controller, and it is configured to remain in place until otherwise interrupted.The interface receives this and sets the Data Reg appropriately and sets the status bit to true.
- It then comprehends the interrupt caused by the controller and stores the data in the buffer of the device driver. Create a control signal if an input is detected, unblock the device's driver, and determine which I/O for character stands finished using the flag bit (F) described previously.
- Then, inform the I/O Subsystem of any subsequent status changes (if needed). Send data to the process (if necessary), and, if necessary, return the completion control signal or the error control signal.
- The I/O process for the Keyboard is complete, the input data is available and stored in memory, or the output is shown using the I/O device's various output methods.(not necessary for this case). Interface notifies the Keyboard that it no longer needs to take Inputs.