

Topic 4 : I/O Devices cont...

Interfacing to I/O devices
Reference : N/A - Self contained.

C.Sc. 131: Systems Architecture

Techniques for Accessing I/O Devices

- Explicit read/write operations (isolated I/O).
 - Can provide optimised performance
- Or, use a technique called memory-mapped I/O.
 - no i/o instructions
 - each device has an associated memory address
 - when we manipulate that address we are actually manipulating the device

C.Sc. 131: Systems Architecture

Performance Mismatch

- There is a speed mismatch between I/O devices and computers.
- Problem: how to avoid slowing down the computer to the speed of the device.
- Solution: introduce a degree of autonomy into the device and allow the computer to get on with other tasks.

C.Sc. 131: Systems Architecture

Data and Status Registers

- In early machines there would be blocking read and write instructions.
- Computer would block until the transfer was complete.
- Since I/O devices are very slow in comparison to computers this is wasted time.

C.Sc. 131: Systems Architecture

Data and Status Registers ... contd.

- We can introduce some autonomy into the device by providing it with a data register capable of holding one character and a status register.
- The device sets the status register to signify that it has a character waiting.
 - cf Modem

C.Sc. 131: Systems Architecture

Data and Status Registers ... contd.

- Sequence of instructions to get a character.

```
start the device by clearing its status register
repeat
  test the status register
  until the status register has been set
  copy the character from the data register into the acc.
```

- **This doesn't take advantage of the device's autonomy.**

C.Sc. 131: Systems Architecture

Improving Efficiency...

- To take advantage of the device's autonomy we need to do some useful work inside the test loop.
- However, when the device has finished we want to find out as soon as possible.
- This means placing lots of tests for multiple devices throughout our code.
- The devices we wish to check may change.
- Since we want to find out about the device's completion ASAP most of the instructions we execute will be redundant.

C.Sc. 131: Systems Architecture

Interrupts

- The solution is to provide support in the form of interrupts.
- The status registers of all the devices are continually checked.
- If a status register changes, jump to instructions to deal with the interrupt.
- This sequence of instructions is called an interrupt service routine (ISR).

C.Sc. 131: Systems Architecture

Causes of Interrupts

- Example causes of interrupts include:
 - devices completing I/O.
 - clocks.
 - power failure.
 - illegal instructions and addresses.
 - data conditions (e.g. divide by zero).

C.Sc. 131: Systems Architecture

Implementing Interrupts

- Need to store the contents of the program counter so we can resume execution at the right point.
- Need to store some state, e.g. the accumulator.
- Need to determine what caused the interrupt.

C.Sc. 131: Systems Architecture

Implementing Interrupts

- Assign some storage to be associated with each interrupt (called the interrupt vector).
- When an interrupt occurs:
 - store the program counter
 - jump to the address specified
- The last instruction of the interrupt service routine can then be a jump through the stored address.

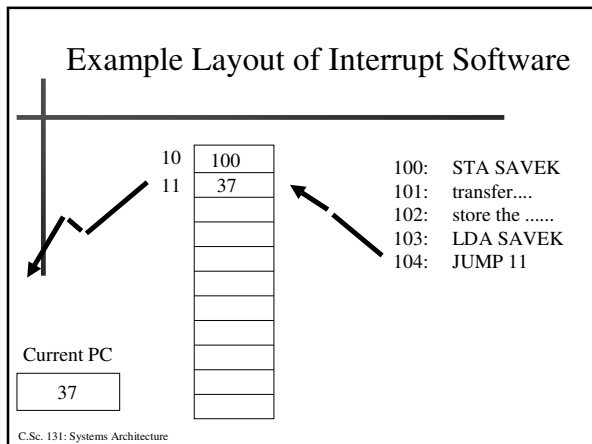
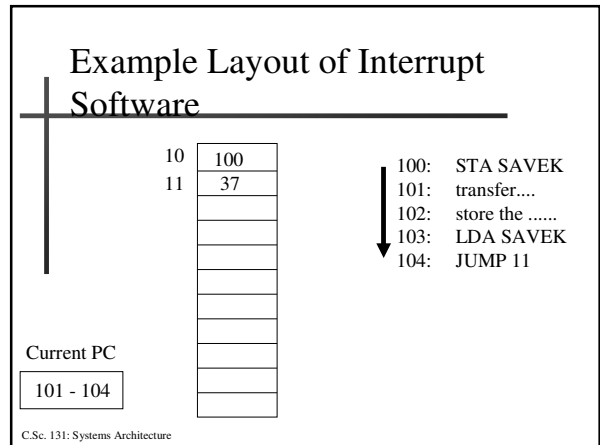
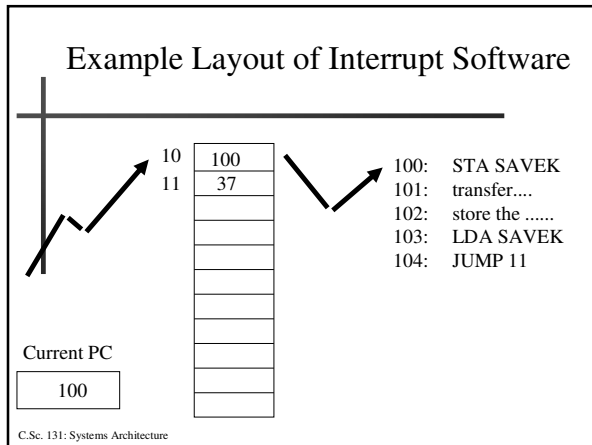
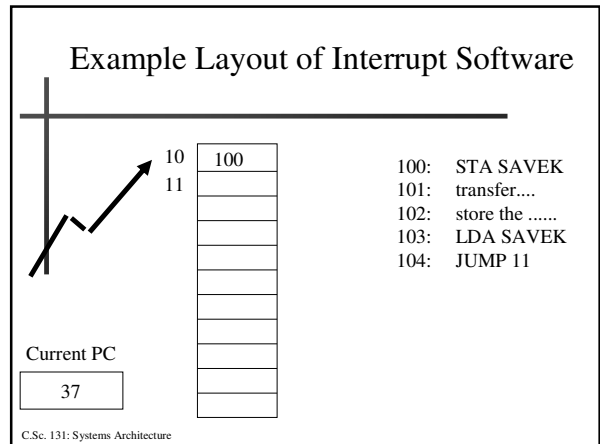
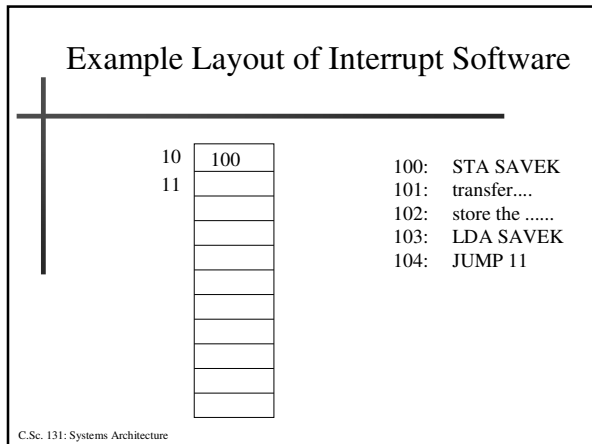
C.Sc. 131: Systems Architecture

Example Layout of Interrupt Software

```
10: 100 ; keyboard
11: 0 ; current PC will be written here

100: STA SAVEK ; save the accumulator
101: transfer the data register to the accumulator
102: store the accumulator in a suitable place
103: LDA SAVEK ; restore the accumulator
104: JUMP 11
```

C.Sc. 131: Systems Architecture



- ### Multiple Interrupts
- This scheme is ok except....
 - need to define what happens if an interrupt occurs when we are already handling an interrupt.
 - One solution is to just have a flag which says we are already busy with an interrupt and all further interrupts are ignored.
 - However, some interrupts we can't afford to ignore: e.g. clocks.
 - Therefore need a means of assigning priorities to interrupts.
- C.Sc. 131: Systems Architecture

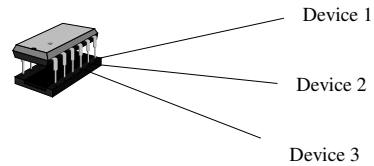
Adding Priorities to Our Interrupt Scheme

- We assign priorities to all sources of interrupts.
- High priority devices (e.g. clocks) can interrupt low priority devices.
- The main program runs at the lowest priority level.
- To tell the processor which priority it is running at, introduce a new register to hold the current priority.

C.Sc. 131: Systems Architecture

Connecting Interrupts to the Processor

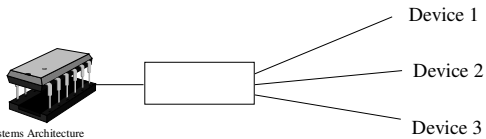
- Simplest approach is to have a separate input pin for each device which can generate an interrupt.



C.Sc. 131: Systems Architecture

Connecting Interrupts to the Processor

- A better (and more usual approach) is to have multiple devices connected to a single pin.
- When an interrupt occurs the processor either polls the devices or the interrupt includes a vector number.



C.Sc. 131: Systems Architecture

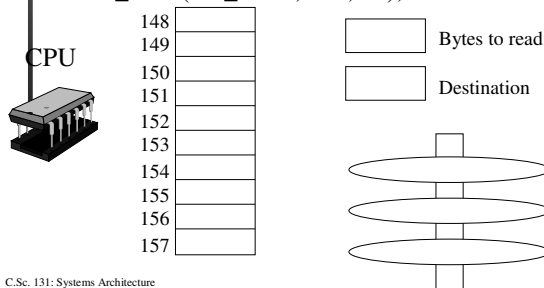
Fast I/O Devices

- Consider a disk which transfers several hundred thousand bytes a second.
- Each byte requires an interrupt.
- Solution is to use Direct Memory Access (DMA)
 - i/o operation specifies a section of memory and the number of bytes to read/write.
 - device reads/writes directly from/to this memory.
 - interrupt generated when complete transfer has been achieved.

C.Sc. 131: Systems Architecture

An Example of DMA

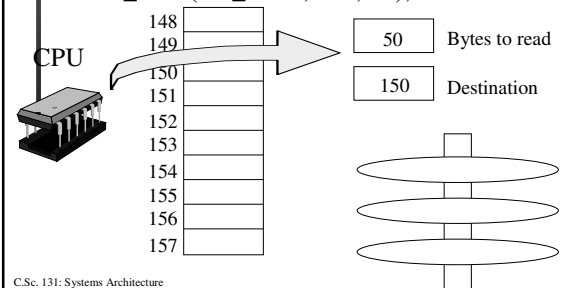
- `disk_read (file_name, 150, 50);`



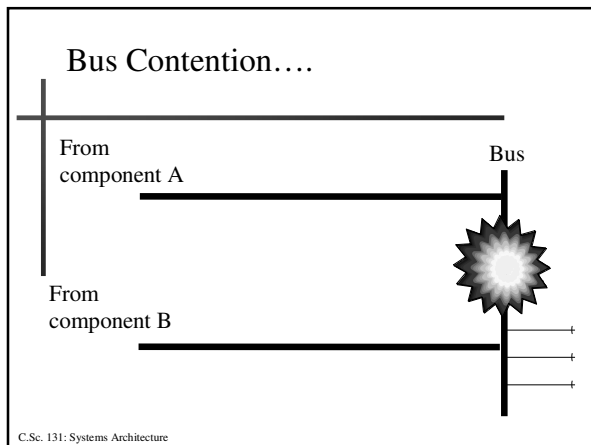
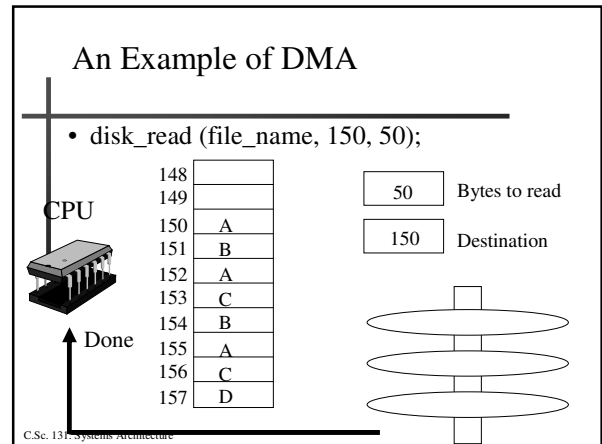
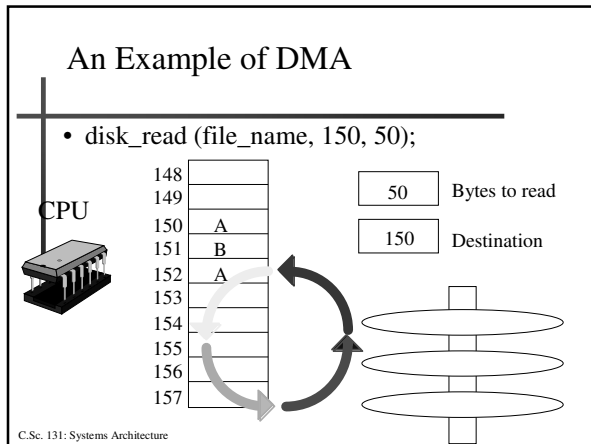
C.Sc. 131: Systems Architecture

An Example of DMA

- `disk_read (file_name, 150, 50);`



C.Sc. 131: Systems Architecture



Bus Contention

- A key issue with DMA is access to the system bus.
- Contention between the DMA device and the processor has to be addressed.
- Normal situation is that the DMA device either gains outright control (and the processor idles) or the device 'cycle steals' and always has priority for the bus.

C.S. 131: Systems Architecture

Device Controllers

- Controller must accept data from the system and control device to make it actually perform the i/o.
- The interface between a device controller and the device itself is very low-level.
 - cf keyboard controller – need to translate voltage to a seven bit ASCII code.
- The interface between the controller and the system is at a higher level - usually through special registers and interrupts.
 - cf a video controller could have a special cursor register and would generate an interrupt when its value changed

C.S. 131: Systems Architecture

An Example Controller

- The IBM PC floppy controller accepts 15 different commands such as READ, WRITE, RECALIBRATE etc.
- Commands and parameters are passed using the registers.
- While the command is being processed the CPU is free to do other work - an interrupt occurs when the command has been completed.

C.S. 131: Systems Architecture

Device Drivers

- Device drivers interface between device independent i/o software in the OS (e.g. `v_read(data)`) and Device Controllers.
- For example, if a device driver for a floppy disk is asked to read block n it must...
 - start the floppy motor.
 - work out where on the disk the block is.
 - move the arm to the appropriate position.
 - initiate the read.
 - stop the motor after the operation.

C.Sc. 131: Systems Architecture

Summary

- Recapped on the construction of magnetic disks.
- Problems of connecting i/o devices to a computer: the performance mismatch.
- Data/Status Registers and Polling
- The Interrupt approach.
- DMA for fast I/O devices.
- Device Controllers and Device Drivers

C.Sc. 131: Systems Architecture