

23-Oct-06 (1)



CSC2510 - Computer Organization

Lecture 8: I/O and Memory

Philip Leong

23-Oct-06 (4)



Other exceptions: privileged execution

- Privilege exception
 - Normal programs run in user mode which does not have certain privileges e.g. can only read certain memory locations, cannot directly access devices etc
 - Attempts to do something without permission leads to a privilege exception (i.e. an error)
 - Can execute a software interrupt to switch to supervisor mode and execute privileged instructions (with checking by OS) e.g. read a file

23-Oct-06 (2)



Exceptions

- So far, have talked about interrupts from I/O data transfers
- **Exception** is any event that causes an interruption to program flow, I/O interrupts are one example
- Others are
 - Recovery from errors e.g. memory has error checking codes which report an error which the OS could report back to the user

23-Oct-06 (5)



Interrupts in the OS

- OS responsible for coordinating all activities in computer
- Uses interrupts to
 - Perform I/O
 - Communicate with and control execution of user programs
- Interrupts allow OS to
 - Switch from one user to another in a multiuser system
 - Implement security and protection features
 - Coordinate I/O activities
- Most OSes e.g. Windows and Linux have both a supervisor and user mode

23-Oct-06 (3)



Other Exceptions: Debugging

- Debugging
 - Trace mode
 - Exception occurs after every instruction
 - Control passed to debugger
 - Breakpoints
 - Debugger places **trap** (also called **software interrupt**) at a given location
 - When program reaches that point, an interrupt occurs

23-Oct-06 (6)



Supervisor & User mode

- Bit in processor status register determines user/supervisor mode, set to supervisor after an interrupt
 - The return from interrupt can restore back to user mode
- E.g. multitasking – several user programs are run at the same time
 - OS uses time slicing to allocate time to all programs for a short period

23-Oct-06 (7)

Multitasking OS

OSINIT	Set interrupt vectors: Time-slice clock ← SCHEDULER Soft-wire interrupt ← OSSERVICES Keyboard interrupts ← IODATA :
OSSERVICES	Examine stack to determine requested operation. Call appropriate routine.
SCHEDULER	Save program state. Select a runnable process. Restore saved context of new process. Push new values for PS and PC on stack. Return from interrupt.

(a) OS initialization, services, and scheduler

23-Oct-06 (10)

Interrupt vector number

- For INTR, vector number sent to processor from I/O device via bus when interrupt request acknowledged
- For other exceptions, interrupt vector number assigned
- Interrupt vector number used to index the **Interrupt Descriptor Table** to get address of interrupt service routine

23-Oct-06 (8)

Multitasking OS

IOINIT	Set process status to Blocked. Initialize memory buffer address pointer and counter. Call device driver to initialize device and enable interrupts in the device interface. Return from subroutine.
IODATA	Poll devices to determine source of interrupt. Call appropriate driver. If END = 1, then set process status to Runnable. Return from interrupt.

(b) I/O routines

KBDINIT	Enable interrupts. Return from subroutine.
KBDATA	Check device status. If ready, then transfer character. If character = CR, then (set END = 1; Disable interrupts) else set END = 0. Return from subroutine.

(c) Keyboard driver

23-Oct-06 (11)

Pentium Interrupts

- A companion chip to the Pentium is called the Advanced Programmable Interrupt Controller (APIC)
 - I/O devices connected to this chip
 - Implements interrupt priority scheme
 - Sends vector number to processor
- Pentium's processor status register (EFLAGS)
 - IF=interrupt enable flag, TF=trap flag, IOPL=IO privilege level (4 levels)

15	14	13	12	11	10	9	8
					IOPL		
						IF	

23-Oct-06 (9)

Pentium Interrupts

- Two interrupt request lines
 - NMI Non maskable interrupt (always accepted by the processor)
 - INTR maskable interrupt (can be turned off by setting an interrupt enable bit in the processor status register)
- Other exceptions
 - Invalid opcode, division error, overflow, trace, breakpoint interrupts etc
- Any of these events cause an interrupt
 - Branch to the corresponding interrupt vector number

23-Oct-06 (12)

Exceptions

- Pentium pushes processor status register, current segment register (CS) and instruction point (EIP) onto processor stack pointed to by ESP
- For abnormal execution condition exceptions, pushes a code with the cause of the exception
- Clears corresponding interrupt-enable flag if appropriate so no further interrupts from that source are enabled (e.g. trace interrupt shouldn't cause further interrupts)
- Fetches ISR address from Interrupt Descriptor Table by indexing with the interrupt vector number, loads into EIP and jumps to it (ISR ends with an IRET instruction which restores EIP, CS and the processor status)

23-Oct-06 (13)

Example: read one line from keyboard

```

Main program
MOV EOL,0
MOV BL,4
OR CONTR_OL,BL      Set KEN to enable keyboard interrupts.
STI                  Set interrupt flag in processor register.
:
:
:
In interrupt-service routine
READ  PUSH  EAX          Save register EAX on stack.
      PUSH  EBX          Save register EBX on stack.
      MOV   EAX,PNTR     Load address pointer.
      MOV   BL,DATAIN    Get input character.
      MOV   [EAX],BL     Store character.
      INC   DWORD PTR [EAX] Increment PNTR.
      CMP   BL,0DH       Check if character is CR.
      JNE   RTRN         RTRN
      MOV   BL,4
      XOR   CONTR_OL,BL  Clear bit KEN.
      MOV   EOL,1        Set EOL flag.
RTRN  POP   EBX          Restore register EBX.
      POP   EAX          Restore register EAX.
      IRET
    
```

23-Oct-06 (16)

Simplified Computer System with DMA

- DMA controller connects network to bus
- Disk controller has 2 independent DMA channels to 2 independent disks
 - Start DMA transfer by writing address and word count information to disk controller
 - DMA controller operates independently of processor (processor does other things, perhaps runs a different program in a multitasking system). Memory accesses by processor and DMA controller perhaps prioritised.
 - DMA completes and sets done and IRQ bit
- DMA accesses can be interwoven with processor accesses (**cycle stealing**) or take over the bus (called **block or burst mode**)
 - **Arbitration** is required to control DMA controllers and processors accessing memory at the same time

23-Oct-06 (14)

Direct memory access

- For transferring large blocks of data at high speed
- A **DMA controller** is a special control circuit which is part of the I/O device interface
 - Controls accesses to the memory (provides memory address, data and control signals)
 - Does so without intervention from the processor
 - Processor controls it i.e. says how much data to transfer, starting address etc
 - Processor and DMA controller run independently, processor can continue running other programs. When DMA finishes, it can return to program that requested the DMA

23-Oct-06 (17)

Bus Arbitration

- Device allowed to initiate data transfers on bus transactions is called **bus master**
- After using the bus, bus master relinquishes control of the bus so others can acquire it
- Bus arbitration is required to decide who is master when more than one requests bus at the same time. Two types
 - **Centralized**
 - **Distributed**

23-Oct-06 (15)

DMA registers (Simplified)

The diagram shows three registers for a DMA interface:

- Status and control:** A 32-bit register with bits 31 and 30 labeled 'IRQ' and 'IE'. Bits 1 and 0 are labeled 'Done' and 'R/W'.
- Starting address:** A register for the starting memory address.
- Word count:** A register for the number of words to transfer.

Figure 4.18. Registers in a DMA interface.

- R/W bit determines direction of transfer

23-Oct-06 (18)

Centralized Arbitration

- In this example processor has bus arbitration circuitry
- Processor is normally bus master
- DMA controller indicates it wants to be bus master by activating /BR (signal on /BR is the OR of all signals on that line – open drain)
 - Processor activates BG1 indicating that they may use bus when it finishes and
 - BG1 can become master and block BG2. Otherwise it can propagate the grant signal to BG2
 - Assume DMA controller 2 wants to become master, it doesn't propagate the grant signal and waits until /BBSY goes high
 - Now DMA controller 2 is the bus master and brings /BBSY low until it is ready to release the bus
- Arbiter circuit ensures only one request is granted at any time and enforces a priority scheme. Rotating priority is also possible.

23-Oct-06 (19)

Centralised bus arbitration

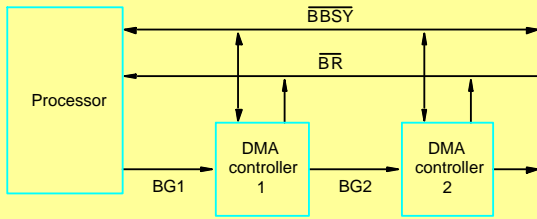


Figure 4.20. A simple arrangement for bus arbitration using a daisy chain

23-Oct-06 (22)

Buses

- Processor, main memory and I/O devices are interconnected by a common bus
 - Provides communications path for transfer of data
 - Includes signals to handle interrupts and arbitration
- **Bus protocol**
 - Set of rules for transferring data on a bus

23-Oct-06 (20)

Centralized Bus Mastering

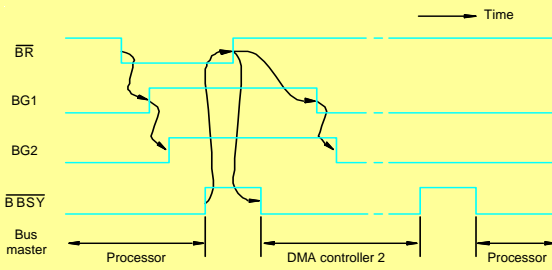


Figure 4.21. Sequence of signals during transfer of bus mastership for the devices in Figure 4.20.

23-Oct-06 (23)

Bus lines

- Three types
 - Data
 - Address
 - Control
 - specify whether read or write, usually a single R/W signal, read when 1 write when 0
 - specify operand size
- Normally processor is master and device being addressed is the slave

23-Oct-06 (21)

Distributed arbitration

- Not covered

23-Oct-06 (24)

Synchronous Bus

- Common clock used to derive timing
- Read: t_0 master puts address and command on bus. Slave responds at t_1 , data is strobed into master at t_2

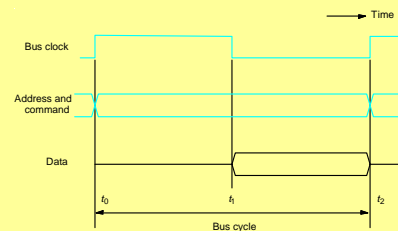


Figure 4.23. Timing of an input transfer on a synchronous bus.

23-Oct-06 (25)

Propagation delay

- Signals take time to propagate. t_1-t_0 must be longer than longest propagation delay between 2 devices on bus including all decoding of address and command signals.

23-Oct-06 (28)

Interface Circuits

- To add a peripheral to a microprocessor, need to design an **interface circuit**
 - Involves hooking up whatever is required to the processor's bus
- I/O device involves the following
 - Storage buffer
 - Status flags that can be accessed by the processor
 - Address decoding circuitry
 - Appropriate timing signals
 - Format conversions e.g. serial to parallel

23-Oct-06 (26)

Multiple-cycle transfers

- In previous example, slave must respond by t_2-t_0 , usually a control signal is used to inform master that the slave data is ready
- Thus slave could be of arbitrary speed
- Master can abort operation if no slave-ready signal is received
- IA-32 uses this scheme

Figure 4.25. An input transfer using multiple clock cycles.

23-Oct-06 (29)

Asynchronous bus (output)

- Slave strobos data in after Master-ready is asserted

Figure 4.27. Handshake control of data transfer during an output operation.

23-Oct-06 (27)

Asynchronous bus (input)

- Handshake between master and slave
- t_0 master places address and command on bus
- t_1 master asserts Master-ready
- t_2 slave puts data on bus and asserts Slave-ready
- t_3 master sees that Slave-ready asserted, strobos in the data and releases Master-ready
- t_4 master can now change address/command
- t_5 slave removes Slave-ready
- Skew is an important issue

Figure 4.26. Handshake control of data transfer during an input operation.

23-Oct-06 (30)

Interfacing circuits and Buses

- Not covered

23-Oct-06 (31)



Memory (Ch 5)

- Maximum size of memory determined by addressing scheme
 - E.g. 16-bit addresses can only address $2^{16} = 65536$ memory locations
 - Most machines byte addressable but retrieve/store data in words
 - $1K = 2^{10}$
 - $1M = 2^{20}$
 - $1G = 2^{30}$
 - $1T = 2^{40}$

23-Oct-06 (34)



Random Access Memory

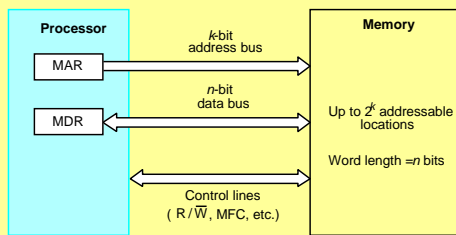
- RAM types
 - Static ram (SRAM) are fast but small
 - Dynamic RAMs (DRAM) store data as charge on a capacitor. This leaks away with time so must be refreshed. In return for this trouble, the density is much higher. The common type used today is called a synchronous DRAM as it uses a clock.
 - Double data rate SDRAM transfers data on both clock edges (normal circuits only operate on rising clock edge)
 - Memory modules are used to hold several SDRAM chips and are the standard type used in a computer's motherboard

23-Oct-06 (32)



Simplified view

- Data transfer takes place through the MAR (memory address register) and MDR (memory data register)

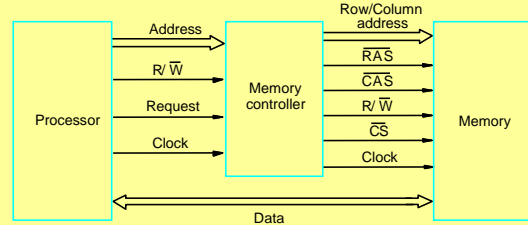


23-Oct-06 (35)



Memory Controller

- A **memory controller** is normally used to interface between DRAM and the processor. Dynamic RAMs have a slightly more complex interface as they multiplex signals in time to reduce pins
- SRAM interfaces are simpler and may not need a memory controller



23-Oct-06 (33)



Definitions

- **Memory access time** = time between start and finish of a memory request
- **Memory cycle time** = minimum delay between successive memory operations
- **Random access memory (RAM)**, any location with comparable access time (c.f. CD vs tape drive)
- Processor usually runs much faster than main memory
 - Small memories fast, large memories slow
 - Use a **cache memory** to store data that is likely to be used
- Processor memory is limited
 - Use **virtual memory** to increase apparent size of physical memory by moving unused sections of memory to disk (automatically). A translation between virtual and physical addresses is done by a **memory management unit**

23-Oct-06 (36)



Read only memories

- ROM
 - Value fixed and cannot be changed
- FLASH
 - Can be read and written. This is normally used for **nonvolatile** storage. Flash cards are made from flash chips.
- Need some way to **bootstrap** a computer as RAM is erased when power removed

23-Oct-06 (37)

Memory Hierarchy

- Used to produce fast, big and cheap memory
- L1, L2 usually SRAM
- Main mem is DRAM
- Relies on **locality of reference**

The diagram shows a vertical stack of memory components. At the top is the **Processor**, which contains **Registers** and a **Primary cache L1**. Below the processor is the **Secondary L2 cache**, followed by **Main memory**, and finally **Magnetic disk Secondary memory** at the bottom. Arrows indicate the flow of data between these levels. To the left of the stack, a downward arrow is labeled "Increasing size". To the right, two upward arrows are labeled "Increasing speed" and "Increasing cost per bit".

23-Oct-06 (40)

Mapping functions

- Direct mapped cache
- Block j of main memory maps to block $j \bmod 128$ of cache
- Cache hit occurs if **tag** matches desired address

The diagram illustrates a direct-mapped cache. On the right, a vertical stack of **Main memory** blocks is shown, including Block 0, Block 1, Block 127, Block 128, Block 129, Block 255, Block 256, Block 257, and Block 4095. On the left, a **Cache** is shown with three entries: **Block 0**, **Block 1**, and **Block 127**. Each entry has a **tag** field. An arrow points from the main memory stack to the cache, indicating the mapping function. Below the cache, a table shows the address format:

Tag	Block	Word
5	7	4

The text "Main memory address" is written below the table.

23-Oct-06 (38)

Locality of Reference

- Temporal (in time)**
 - When information item (instruction or data) is first needed, brought into cache where it will hopefully be used again
- Spatial (in space)**
 - Rather than a single word, fetch data from adjacent addresses as well.
 - Block refers to a set of contiguous addresses of given size (**cache block** also called **cache line**)
- Need to have a **replacement algorithm** to decide what to do when cache is full

23-Oct-06 (39)

Cache Usage

- Cache is transparent to the processor

The diagram shows three boxes: **Processor**, **Cache**, and **Main memory**. Bidirectional arrows connect the Processor to the Cache, and the Cache to Main memory, indicating the flow of data.