Addressing I/O Devices

- I/O as basic features of a computer
  - communicate with outside world
  - support mass data storage
  - critical role in embedded systems

Single external bus
- used by most modern computers
- each I/O device responds to a unique address placed by the processor
- conducts either a read or write operation

In “Memory-mapped I/O”, I/O devices and memory share the same address space
- an machine instruction that can access memory, can also transfer data to/from I/O

Single Bus Structure

```
Move, DATAIN, R0
Move R0, DATAOUT
```
Some processors use special I/O instructions and a separate address for I/O devices
  – (e.g. Intel processors)

I/O devices operate at speed vastly slower than the processor
  – e.g. reading a character from the keyboard happens when on is ready and only one is read

“Programmed-controlled I/O”
  – processor repeatedly checks synchronization flags within the Status register

I/O Interface for an Input Device
Interrupts

• Interrupt handling is an important function of the operating systems
  – coordinating I/O transfer
  – also enables transfer of control between programs due to an external event
  – very important in control applications, “real-time processing”

• An I/O device alerts the processor when it is ready by sending a hardware signal, “interrupt”
  – one control line is the “Interrupt Request Line”
  – allows the processor to perform other functions vice continuously checking external status

Transfer of Control Through the Use of Interrupts

Program 1
COMPUTE routine

Program 1
PRINT routine

Interrupt occurs here
Interrupts 2

- The processor used an “Interrupt-Acknowledge” signal to inform a device that its request has been recognized.

- An “Interrupt Service Routine” is executed in response to an interrupt request:
  - “Interrupt Latency” is a delay in which current program status information is temporarily stored.
  - Earlier processor with few registers saved/restored all registers automatically upon interrupt signal.
  - At a minimum, the contents of the PC and Processor Status Register is saved/restored automatically:
    - Register data is saved/restored by the Interrupt Service Routine.
  - To minimize service overhead, some computers have two levels of interrupt:
    - One saves all registers, the other does not.
    - Devices may use either depending on the required response time.
  - Another approach is to provide a dual set of registers.
Interrupts 3

• The arrival of an interrupt request from an external device causes the processor to suspend program execution
  – may not be desirable or efficient to do so during any particular process
  – all computers provide the capability to enable and (re)enable interrupts
  – e.g. disable printer interrupts after a print job completes or during execution of a critical routine

• The interrupt-request signal is active during the execution of the service routine until accesss to the interrupting device is instructed

• essential that the interrupt-request signal does not cause successive interrupts

• three basic solutions:
  – use a disable instruction as the fist command in the service routine
  – automatically disable interrupts before starting the service routine via a “interrupt mask” within the Status Register
  – have the interrupt handling circuit in the process respond on to the leading edge of the interrupt, “edge-trigger”
Interrupts 4

1. Device raises an interrupt request.
2. Processor interrupts the program currently being executed
3. Disable interrupts
4. Inform the device that its request is recognized, in response the device disables the request signal
5. The Service Routine performs the requested action
6. Resume execution of the interrupted program; enable interrupts

Several devices may independently generate interrupts

1. How can the processor recognize the device requesting an interrupt?
2. How can the processor obtain the starting address of the appropriate service routine?
3. Should a device be allowed to interrupt the processor while another interrupt is being serviced?
4. How should two or more simultaneous requests be handled?

Various ways of resolving these problems
- the specific method is an critical consideration in determining suitability for a given application
Handling Multiple Devices

- Device identification via an open collector bus
  - the bus is normally at a high-voltage, INTR, and inverted within the CPU to a 0 state
  - a device signals an interrupt by grounding the circuit and causing INTR to go to a 1 state
  - an Interrupt Request (IRQ) bit within the status register of the requesting device is set
  - the processor polls each device on the bus to check the IRQ bit

- In “Vectored Interrupts”, the requesting device identifies itself directly to the processor
  - by sending a special code over the bus
  - the code identifies the starting address of the service routine either by full or partial address
    - the service routine or a pointer to it (“the interrupt vector”) must always be loaded at the same memory address
  - Sequence of activities:
    - device generates interrupt
    - CPU completes execution of the current instruction and saves its status
    - CPU generates an interrupt acknowledgment signal, by activating the INTA line
    - the device sends the vector code on the data lines and turns off the INTR signal
Handling Multiple Devices 2

Equivalent Circuit for an Open-Collector bus to Implement a Common Interrupt Line

CPU

\[ \text{INTR} \]

\[ \text{INTR}_1 \quad \text{INTR}_2 \quad \cdots \quad \text{INTR}_n \]

\[ V_{cc} \]

\[ R \]

Implementation of an Interrupt Priority Using Interrupt-Request and Acknowledge Lines

Interrupt Arbitration Circuit

CPU

\[ \text{INTA}_1 \quad \text{INTA}_2 \quad \cdots \quad \text{INTA}_p \]

\[ \text{INTR}_1 \quad \text{INTR}_2 \quad \cdots \quad \text{INTR}_p \]

Device 1

Device 2

\[ \text{INTA}_1 \quad \text{INTA}_2 \quad \cdots \quad \text{INTA}_p \]

Device p

CS 5515 - Computer Architecture

Copyright 1999 ©Roy M. Wnek. All rights reserved.
Handling Multiple Devices 3

• Handling interrupt service routines with interrupts disabled, may be problematic in real-time processing or with long service routines

• need a multi-level priority scheme for handling interrupts
  – assign each device a priority number under program control
  – while servicing an interrupt, accepts interrupts only from devices with higher priorities
  – disable interrupts from devices with lower priorities

• The processor decides which device to service when simultaneous interrupts are received
  – by priority in a multi-level priority system
  – by polling order in an open-collector bus system
  – by the way the device is connected in an interrupt-vector system

  e.g. a daisy-chain arrangement for the interrupt-acknowledgment signal, INTA, gives priority to the device which is electrically closest to the processor

  a priority group arrangement allows a device to be connected to several different priority levels
Interrupt Priority Schemes

(a) Daisy Chain

(b) Arrangement of Priority Groups
Interrupts 5

• “Exceptions” refers generically to all interrupts to an executing program
  – not just those generated by I/O devices
  – when errors or unusual conditions are detected by the processor (e.g., invalid OP code)
  – aids in debugging software programs as in a processor operating in “trace mode” or programmer-specified “breakpoints”
  – “Privilege Exception” protects the operating system from being corrupted by user programs
    • certain instructions can only be executed while in supervisor mode (e.g., change device interrupt priority level)

• An application program issues a “software interrupt” or “trap” to request support from the OS
  – the OS handles all I/O activities and other systems resources and returns control to the application
    • each software interrupt has its own interrupt vector
    • all software pertaining to a particular device be encapsulated in a “device driver” module
  – especially critical in “multitasking” where each application is given a “time slice” by the “scheduler”
  – programs or “Processes” can be in one of three states:
    • Running - currently executing
    • Runnable - ready for execution but waiting to be selected
    • Blocked - not ready to execute (e.g., waiting for I/O)
Direct Memory Access

- A special control unit allows transfer of blocks of data directly between an external device and main memory
  - without intervention of the processor

(a) Registers in a DMA Interface

<table>
<thead>
<tr>
<th>Status and Control</th>
<th>31</th>
<th>30</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRQ</td>
<td></td>
<td></td>
<td>R/W</td>
<td>Done</td>
</tr>
<tr>
<td>IE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) A Two Channel DMA Controller

- DMA Controller
- Disk Unit
- High Speed Printer
- Main Memory
- CPU
- Bus
- Disk Unit
- High Speed Printer
- Main Memory
- CPU

Copyright 1999 ©Roy M. Wnek. All rights reserved.
Direct Memory Access 2

• The DMA controllers performs the functions normally done by the CPU when accessing main memory
  – provides memory address and bus signals
  – automatically increments addresses when transferring blocks of data
  – operates under the control of a program executing within the processor
  – allow the processor to do other work

• To initiate block transfer of data, the processor send the following key data to the controller
  – the starting address
  – the number words in the block
  – function to be performed (read or write)

• The controller performs the transfer and signals the CPU when it is completed with an interrupt signal
  – a connection is needed for each DMA channel between the controller and one of the I/O devices
To transfer a block of data from main memory to the printer:

- 1. the application interrupts the OS and then waits
- 2. a routine in the OS writes the key data to the registers in the DMA channel assigned to the printer
- 3. the OS suspends the current program and may select another for execution
- 4. the DMA controller independently executes the specified function
   - uses its connection to the printer to determine if it is ready
   - when ready, the controller send a read request to memory and instructs the printer to receive data from the bus
   - then it waits for the printer to be ready again
   - when DMA transfer is complete, the Done bit within the DMA Status and Control Register is set to 1
   - if the IE bit is set, a interrupt request is sent to the CPU and the IRQ bit is set
- 5. when the processor recognized the completion interrupt, the OS returns the original program to the Runnable state
Bus Arbitration

• A conflict arises if the CPU and one or more DMA controllers attempt to use the bus at the same time to access memory.

• A “bus arbiter” circuit coordinates all devices requesting memory transfer:
  – Based on a priority system for gaining access to the bus.
  – The arbiter may be part of the CPU or a separate unit.
  – The device allowed to initiate bus transfers is “the Bus master”.

• “Cycle Stealing”:
  – Access by the CPU and DMA controller(s) is interwoven with top priority given to DMA transfers involving synchronous, high-speed peripherals.
  – Since the CPU originates most memory accesses, the DMA controller “steals” memory cycles from the CPU.

• “Block Mode”:
  – A DMA controller is given exclusive access to main memory to transfer a block of data without interruption.
Bus Arbitration 2

(a) Bus arbitration using a Daisy Chain

(b) Signal Sequence during Transfer of Bus Mastership
   (DMA Controller requests mastership)
Bus Arbitration 3

- Bus arbitration signals on an open collector circuits
  - Bus Request (BR) signal is a logical OR
  - Bus Busy (BBSY)
- Bus arbitration signals on standard circuit
  - Bus Grant (BG) indicates the a controller can use the bus if is not busy

Connection of a Device Interface Circuit to Priority Arbitration Lines

Start Arbitration

Interface circuit to device A
Bus Arbitration 4

- Details of timing vary significantly among processor buses

- Signals generated by the processor are always synchronized with the processor clock
  - a clock signal is often included on the bus for use by I/O device interfaces to control their own operations

- Several pairs of bus-request (BR$_1$..BR$_n$) and bus-grant (BG$_1$..BG$_n$) lines may be provided

  "Centralized arbitration"
  - may be arranged in multiple levels of priority
  - a single bus arbiter insures that only one device is set as the bus master using a fixed or rotating priority scheme

  "Distributed arbitration"
  - all devices have a role in bus arbitration
  - each device uses a Start Arbitration signal and a binary identification code on open collector lines
  - the bus master is determined by binary logical interaction between all requesting devices
  - the id number of a new bus master is returned on the identification code line (ARB$_1$..ARB$_n$)
• Primary function of the bus is to interconnect the process, main memory, & I/O devices

• Control sub-bus signals indicate
  – “mode” of data transfer (read or write); also the size of the transfer (word, byte, etc.)
  – timing of when the process or I/O devices may use the bus (synchronous or asynchronous)

Timing of an Input Transfer on a Synchronous Bus

Bus Clock

Address and mode information

Data

Bus cycle

\[ t_0 \quad t_1 \quad t_2 \]
Bus Timing

Synchronous Bus
- All devices derive timing from a common clock line that generates equally spaced pulses
- Clock pulse width must be greater than the max propagation delay and the decoding time
- Procedure for a read operation
  - \( t_0 \): processor sends a device address and set the mode; devices recognizes the mode
  - \( t_1 \): the devices sends data
  - \( t_2 \): the processor “strokes” the data lines; loads MDR
- Simple design but constrained to the slowest device
- No way for the processor to detect an response error

Asynchronous Bus
- Controlled by a “handshake” between the processor and the I/O device
- Clock line is replaced with Ready & Accept lines
- Procedure for a write operation
  - processor sends data; activates Ready
  - device recognizes Ready; reads data; activates Accept
  - processor waits for Accept before removing data from the bus
- Provides flexibility & reliability, but more complex circuitry

Mixed Bus: Asynchronous but paced by a clock
Handshake control of Data Transfer

During an Input Operation

Address and mode information

Ready

Accept

Data

Bus cycle

During an Output Operation

Address and mode information

Data

Ready

Accept

CS 5515 - Computer Architecture

Copyright 1999 ©Roy M. Wnek. All rights reserved.
Interface Circuits

- Circuitry required to transfer data between the compute bus and I/O devices
  - bus side: must be standardized
  - device side: dependent on the function and design of the device
  - serial (transfers one bit at a time) or parallel (transfers one or bytes at a time)

- Functions of an I/O Interface
  1) Provide a storage buffer
  2) Contain status flags that can be accessed by the processor to determine whether the buffer is
     - full (for input)
     - empty (for output)
  3) Contain address-decoding circuitry to determine when it is being addressed by the processor
  4) Generates the appropriate timing signals required by the bus control scheme
     - e.g. in Distributed bus arbitration
  5) Perform an format conversion necessary to transfer data between the bus and the device
     - e.g. parallel-serial conversion
Interface

Parallel Interface

- Typically used with keyboards, printers and portable storage devices
- Keyboards contain mechanical switches which are normally open
  - Pressing a key closes circuit, generating an ASCII character
  - Debouncing circuits or software prevents multiple characters from being unintentionally generated
- Printers are typically controlled via handshake signals
  - Valid: interface is sending new data
  - Idle: printer is ready to receive new data

Serial Interface

- Contain a circuit to communicate in parallel on the bus side and serial on the device side
  - This transformation is normally done with shift registers
- Typically used with external modems, remote terminals, and some mice
- Double buffering allows additional data to be moved while the processor is handling previous data
Parallel Connections

Keyboard Connected to Processor

Processor

Data
Address
R/W
Ready
Accept

Input Interface

DATAIN
SIN

Encoder and debouncing circuit

Data
Accept

Keyboard

Printer Connected to Processor

Processor

Data
Address
R/W
Ready
Accept

Input Interface

DATAIN
SIN

Printer

Data
Valid
Accept
Bidirectional 8 bit Parallel Interface

- DATAIN
- DATAOUT
- Chip and register select
- Status and control
- CS, RS2, RS1, RS0, R/W, Ready, Accept, INTR

P0 → P7
Serial Interface

Input Shift Register

DATAIN

Output Shift Register

DATAOUT

Chip and register select

Status and control

Serial input

Serial output

Receive clock

Transmission clock

D7

D0

CS
RS2
RS1
RS0
R/W
Ready
Accept
INTR
I/O Interface Standards

- More important to standardize I/O interfaces than computer buses
  - allow peripheral devices to be used with multiple types of computers
  - computer bus are tied closely to the basic architecture
  - very wide range of peripheral designs and performances

- Centronix is a very common parallel interface standard for printers
  - defined by Centronix Corp

- RS-232-C is a very common serial interface standard
  - Defined by the Electronics Industry Association (EIA)

- SCSI (Small Computer System Interface) is a common bus-oriented standard for connecting mass storage devices and scanners
  - multiple devices can share the same bus
  - parallel data transfer
  - multiple versions: SCSI I, SCSI II, Wide SCSI
Universal Serial Bus (USB)

• Intel-promoted standard

• Communication occurs via bi-directional pipes
  – data transfers at full-speed (12 Mb/sec.) or low-speed (1.5 Mb/sec.)

• Control is a host-based token, with bus access guaranteed
  – the USB host handles most protocol complexity, generates the tokens, and manages the logical connection from the host to function
  – design asymmetry allows the design of extremely simple and low-cost USB-compliant peripherals

• Data flows on three logical levels:
  1. Client software to function
     • transport agent for data transfers moving between client software and function endpoints
     • envisioned as a bundle of logical data flow "pipes." (one logical endpoint for each client software-function pipe)
Universal Serial Bus (USB) 2

2. USB driver to device
   • devices are managed by USB system software
   • "Endpoint 0" is required of every device for control transfers
   • Up to 16 other endpoints are optionally available for each device

3. USB physical level
   • provides limited power distribution and allows hot insertion and removal

Host

- Client Software manages interfaces
- USB Systems Software manages devices
- USB Bus Interface Host Controller/SIE

Peripheral

- Function Collect of Interfaces
- USB Logical Device Collection of Endpoints
- USB Bus Interface Peripheral Controller/SIE

Logical Comm. Flow
Actual Comm. Flow
Standard I/O Interfaces

Processor Bus

Processor
Memory

Parallel interface

Printer

Terminal or Modem

Serial interface

SCSI controller

SCSI Bus

Disk controller
Disk 1
Disk2

CD-ROM controller
CD-ROM drive

Tape controller
Tape drive

Centronix interface standard
RS-232C interface standard
Addressing I/O Devices

• I/O as basic features of a computer
  – communicate with outside world
  – support mass data storage
  – critical role in embedded systems

Single external bus
  – used by most modern computers
  – each I/O device responds to a unique address placed by the processor
  – conducts either a read or write operation

• In “Memory-mapped I/O”, I/O devices and memory share the same address space
  – an machine instruction that can access memory, can also transfer data to/from I/O

Single Bus Structure

```
Move, DATAIN, R0
Move R0, DATAOUT
```
Addressing I/O Devices 2

- Some processors use special I/O instructions and a separate address for I/O devices
  - (e.g. Intel processors)

- I/O devices operate at speed vastly slower than the processor
  - e.g. reading a character from the keyboard happens when on is ready and only one is read

- “Programmed-controlled I/O”
  - processor repeatedly checks synchronization flags within the Status register

I/O Interface for an Input Device

```
Bus

Address lines

Data lines

Control lines

Address Decoder

Control Circuits

IE SOUT SIN

DATAOUT

DATAIN

Data and Status Registers

I/O Interface

Input Device
```
Interrupts

- Interrupt handling is an import function of the operating systems
  - coordinating I/O transfer
  - also enables transfer of control between programs due to an external event
  - very important in control applications, “real-time processing”

- An I/O device alerts the processor when it is ready by sending a hardware signal, “interrupt”
  - one control line is the “Interrupt Request Line”
  - allows the processor to perform other functions vice continuously checking external status

Transfer of Control Through the Use of Interrupts

Interrupt occurs here

Program 1
COMPUTE routine

Program 1
PRINT routine
Interrupts 2

• The processor used an “Interrupt-Acknowledge” signal to inform a device that its request has been recognized

• An “Interrupt Service Routine” is executed in response to an interrupt request
  – “Interrupt Latency” is a delay in which current program status information is temporarily stored
  – earlier processor with few registers saved/restored all registers automatically upon interrupt signal
  – at a minimum the contents of the PC and Processor Status Register is saved/restored automatically
    • register data is saved/restored by the Interrupt Service Routine
  – to minimize service overhead, some computer have two levels of interrupt
    • one saves all registers, the other does not
    • devices may use either depending on the required response time
  – another approach is to provide a dual set of registers
Interrupts 3

- The arrival of an interrupt request from an external device causes the processor to suspend program execution
  - may not be desirable or efficient to do so during any particular process
  - all computers provide the capability to enable and (re)enable interrupts
  - e.g. disable printer interrupts after a print job completes or during execution of a critical routine

- The interrupt-request signal is active during the execution of the service routine until accesss to the interrupting device is instructed
- essential that the interrupt-request signal does not cause successive interrupts

- three basic solutions:
  - use a disable instruction as the fist command in the service routine
  - automatically disable interrupts before starting the service routine via a “interrupt mask” within the Status Register
  - have the interrupt handling circuit in the process respond on to the leading edge of the interrupt, “edge-trigger”
Interrupts 4

1. Device raises an interrupt request.
2. Processor interrupts the program currently being executed
3. Disable interrupts
4. Inform the device that its request is recognized, in response the device disables the request signal
5. The Service Routine performs the requested action
6. Resume execution of the interrupted program; enable interrupts

- Several devices may independently generate interrupts
  1. How can the processor recognize the device requesting an interrupt?
  2. How can the processor obtain the starting address of the appropriate service routine?
  3. Should a device be allowed to interrupt the processor while another interrupt is being serviced?
  4. How should two or more simultaneous requests be handled?

- Various ways of resolving these problems
  - the specific method is an critical consideration in determining suitability for a given application
Handling Multiple Devices

- Device identification via an open collector bus
  - the bus is normally at a high-voltage, INTR, and inverted within the CPU to a 0 state
  - a device signals an interrupt by grounding the circuit and causing INTR to go to a 1 state
  - an Interrupt Request (IRQ) bit within the status register of the requesting device is set
  - the processor polls each device on the bus to check the IRQ bit

- In “Vectored Interrupts”, the requesting device identifies itself directly to the processor
  - by sending a special code over the bus
  - the code identifies the starting address of the service routine either by full or partial address
    - the service routine or a pointer to it (“the interrupt vector”) must always be loaded at the same memory address
  - Sequence of activities:
    - device generates interrupt
    - CPU completes execution of the current instruction and saves its status
    - CPU generates an interrupt acknowledgment signal, by activating the INTA line
    - the device sends the vector code on the data lines and turns off the INTR signal
Handling Multiple Devices 2

Equivalent Circuit for an Open-Collector bus to Implement a Common Interrupt Line

Implementation of an Interrupt Priority Using Interrupt-Request and Acknowledge Lines
• Handling interrupt service routines with interrupts disabled, may be problematic in real-time processing or with long service routines
• need a multi-level priority scheme for handling interrupts
  – assign each device a priority number under program control
  – while servicing an interrupt, accepts interrupts only from devices with higher priorities
  – disable interrupts from devices with lower priorities

• The processor decides which device to service when simultaneous interrupts are received
  – by priority in a multi-level priority system
  – by polling order in an open-collector bus system
  – by the way the device is connected in an interrupt-vector system
e.g. a daisy-chain arrangement for the interrupt-acknowledgment signal, INTA, gives priority to the device which is electrically closest to the processor
a priority group arrangement allows a device to be connected to several different priority levels
Interrupt Priority Schemes

(a) Daisy Chain

(b) Arrangement of Priority Groups
Interrupts 5

- “Exceptions” refers generically to all interrupts to an executing program
  - not just those generated by I/O devices
  - when errors or unusual conditions are detected by the processor (e.g. an invalid OP code)
  - aids in debugging software programs as in a processor operating in “trace mode” or programmer-specified “breakpoints”
  - “Privilege Exception” protects the operating system from being corrupted by user programs
    - certain instructions can only be executed while in supervisor mode (e.g. change device interrupt priority level)

- An application program issues a “software interrupt” or “trap” to request support from the OS
  - the OS handles all I/O activities and returns control to the application
    - each software interrupt has its own interrupt vector
    - all software pertaining to a particular device be encapsulated in a “device driver” module
  - especially critical in “multitasking” where each application is given a “time slice” by the “scheduler”
  - programs or “Processes” can be in one of three states:
    - Running - currently executing
    - Runnable - ready for execution but waiting to be selected
    - Blocked - not ready to execute, (e.g. waiting for I/O)
Direct Memory Access

• A special control unit allows transfer of blocks of data directly between an external device and main memory
  – without intervention of the processor

(a) Registers in a DMA Interface

<table>
<thead>
<tr>
<th>Status and Control</th>
<th>IRQ</th>
<th>IE</th>
<th>R/W</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word Count</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) A Two Channel DMA Controller

Diagram showing DMA Controller connected to Disk Unit, High Speed Printer, Main Memory, and CPU through the bus.
Direct Memory Access 2

• The DMA controllers performs the functions normally done by the CPU when accessing main memory
  – provides memory address and bus signals
  – automatically increments addresses when transferring blocks of data
  – operates under the control of a program executing within the processor
  – allow the processor to do other work

• To initiate block transfer of data, the processor send the following key data to the controller
  – the starting address
  – the number words in the block
  – function to be performed (read or write)

• The controller performs the transfer and signals the CPU when it is completed with an interrupt signal
  – a connection is needed for each DMA channel between the controller and one of the I/O devices
Direct Memory Access 3

- To transfer a block of data from main memory to the printer:
  - 1. the application interrupts the OS and then waits
  - 2. a routine in the OS writes the key data to the registers in the DMA channel assigned to the printer
  - 3. the OS suspends the current program and may select another for execution
  - 4. the DMA controller independently executes the specified function
     - uses its connection to the printer to determine if it is ready
     - when ready, the controller send a read request to memory and instructs the printer to receive data from the bus
     - then it waits for the printer to be ready again
     - when DMA transfer is complete, the Done bit within the DMA Status and Control Register is set to 1
     - if the IE bit is set, a interrupt request is sent to the CPU and the IRQ bit is set
  - 5. when the processor recognized the completion interrupt, the OS returns the original program to the Runnable state
Bus Arbitration

- A conflict arises if the CPU and one or more DMA controllers try to use the bus at the same time to access memory.

- A “bus arbiter” circuit coordinates all devices requesting memory transfer:
  - based upon a priority system for gaining access to the bus
  - the arbiter may be part of the CPU or a separate unit
  - the device allowed to initiate bus transfers is “the Bus master”

- “Cycle Stealing”
  - access by the CPU and DMA controller(s) is interwoven with top priority given to DMA transfers involving synchronous, high-speed peripherals
  - since the CPU originates most memory accesses, the DMA controller in speaking “steals” memory cycles from CPU

- “Block Mode”
  - a DMA controller is given exclusive access to main memory to transfer a block of data without interruption
Bus Arbitration 2

(a) Bus arbitration using a Daisy Chain

Processor

DMA controller 1

DMA controller 2

BG1

BG2

BBSY

BR

(b) Signal Sequence during Transfer of Bus Mastership

(DMA Controller requests mastership)

BR

BG1

BG2

BBSY

Bus Master

Processor

DMA Controller 2

Processor
Bus Arbitration 3

- Bus arbitration signals on an open collector circuits
  - Bus Request (BR) signal is a logical OR
  - Bus Busy (BBSY)
- Bus arbitration signals on standard circuit
  - Bus Grant (BG) indicates the controller can use the bus if is not busy

Connection of a Device Interface Circuit to Priority Arbitration Lines

[Diagram showing ARB3, ARB2, ARB1, ARB0 and O.C. connections]
Bus Arbitration 4

• Details of timing vary significantly among processor buses

• Signals generated by the processor are always synchronized with the processor clock
  – a clock signal is often included on the bus for use by I/O device interfaces to control their own operations

• Several pairs of bus-request (BR₁..BRₙ) and bus-grant (BG₁..BGₙ) lines may be provided

  “Centralized arbitration”
  – may be arranged in multiple levels of priority
  – a single bus arbiter insures that only one device is set as the bus master using a fixed or rotating priority scheme

  “Distributed arbitration”
  – all devices have a role in bus arbitration
  – each device uses a Start Arbitration signal and a binary identification code on open collector lines
  – the bus master is determined by binary logical interaction between all requesting devices
  – the id number of a new bus master is returned on the identification code line (ARB₁..ARBₙ)
Processor Bus

- Primary function of the bus is to interconnect the process, main memory, & I/O devices

- Control sub-bus signals indicate
  - “mode” of data transfer (read or write); also the size of the transfer (word, byte, etc.)
  - timing of when the process or I/O devices may use the bus (synchronous or asynchronous)

Timing of an Input Transfer on a Synchronous Bus

<table>
<thead>
<tr>
<th>Bus Cycle</th>
<th>Bus Clock</th>
<th>Address and mode information</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright 1999 ©Roy M. Wnek. All rights reserved.
Bus Timing

Synchronous Bus
- All devices derive timing from a common clock line that generates equally spaced pulses
- Clock pulse width must be greater than the maximum propagation delay and the decoding time
- Procedure for a read operation
  - $t_0$: processor sends a device address and sets the mode; devices recognize the mode
  - $t_1$: the device sends data
  - $t_2$: the processor “strokes” the data lines; loads MDR
- Simple design but constrained to the slowest device
- No way for the processor to detect a response error

Asynchronous Bus
- Controlled by a “handshake” between the processor and the I/O device
- Clock line is replaced with Ready & Accept lines
- Procedure for a write operation
  - Processor sends data; activates Ready
  - Device recognizes Ready; reads data; activates Accept
  - Processor waits for Accept before removing data from the bus
- Provides flexibility & reliability, but more complex circuitry

Mixed Bus: Asynchronous but paced by a clock
Handshake control of Data Transfer

During an Input Operation
- Address and mode information
- Ready
- Accept
- Data

During an Output Operation
- Address and mode information
- Data
- Ready
- Accept
Interface Circuits

• Circuitry required to transfer data between the compute bus and I/O devices
  – bus side: must be standardized
  – device side: dependent on the function and design of the device
  – serial (transfers one bit at a time) or parallel (transfers one or bytes at a time)

• Functions of an I/O Interface
  1) Provide a storage buffer
  2) Contain status flags that can be accessed by the processor to determine whether the buffer is
     • full (for input)
     • empty (for output)
  3) Contain address-decoding circuitry to determine when it is being addressed by the processor
  4) Generates the appropriate timing signals required by the bus control scheme
     • e.g. in Distributed bus arbitration
  5) Perform an format conversion necessary to transfer data between the bus and the device
     • e.g. parallel-serial conversion
Interface

Parallel Interface

• Typically used with keyboards, printers and potable storage devices

• Keyboards contain mechanical switches which are normally open
  – Pressing a key closes circuit, generating an ASCII character
  – Debouncing circuits or software prevents multiple characters from being unintentionally generated

• Printers are typically controlled via handshake signals
  – Valid: interface is sending new data
  – Idle: printer is ready to receive new data

Serial Interface

• Contain a circuit to communicate in parallel on the bus side and serial on the device side
  – this transformation is normally done with shift registers

• Typically used with external modems, remote terminals, and some mice

• Double buffering allows additional data to be moved while the processor is handling previous data
Parallel Connections

Keyboard Connected to Processor

Processor

Data
Address
R/W
Ready
Accept

Input Interface

DATAIN
SIN

Encoder
and
debouncing
circuit

Data
Accept

Keyboard

Printer Connected to Processor

Processor

Data
Address
R/W
Ready
Accept

Input Interface

DATAIN
SIN

Printer

Data
Valid
Accept
Bidirectional 8 bit Parallel Interface

- DATAIN
- DATAOUT
- Chip and register select
- Status and control

Inputs:
- CS
- RS2
- RS1
- RS0
- R/W
- Ready
- Accept
- INTR

Outputs:
- P7
- P0
Serial Interface

Input Shift Register

Serial input

...DATAIN...

...DATAOUT...

Output Shift Register

Serial output

Chip and register select

CS
RS2
RS1
RS0
R/W
Ready
Accept

Status and control

Receiving clock
Transmission clock

D7
D0

CS 5515 - Computer Architecture

Copyright 1999 ©Roy M. Wnek. All rights reserved.
I/O Interface Standards

- More important to standardize I/O interfaces than computer buses
  - allow peripheral devices to be used with multiple types of computers
  - computer bus are tied closely to the basic architecture
  - very wide range of peripheral designs and performances

- Centronix is a very common parallel interface standard for printers
  - defined by Centronix Corp

- RS-232-C is a very common serial interface standard
  - Defined by the Electronics Industry Association (EIA)

- SCSI (Small Computer System Interface) is a common bus-oriented standard for connecting mass storage devices and scanners
  - multiple devices can share the same bus
  - parallel data transfer
  - multiple versions: SCSI I, SCSI II, Wide SCSI
Universal Serial Bus (USB)

- Intel-promoted standard

- Communication occurs via bi-directional pipes
  - data transfers at full-speed (12 Mb/sec.) or low-speed (1.5 Mb/sec.)

- Control is a host-based token, with bus access guaranteed
  - the USB host handles most protocol complexity, generates the tokens, and manages the logical connection from the host to function
  - design asymmetry allows the design of extremely simple and low-cost USB-compliant peripherals

- Data flows on three logical levels:
  1. Client software to function
     - transport agent for data transfers moving between client software and function endpoints
     - envisioned as a bundle of logical data flow "pipes." (one logical endpoint for each client software-function pipe)
2. USB driver to device
   • devices are managed by USB system software
   • "Endpoint 0" is required of every device for control transfers
   • Up to 16 other endpoints are optionally available for each device

3. USB physical level
   • provides limited power distribution and allows hot insertion and removal
Standard I/O Interfaces

Processor Bus

Processor
Memory
Parallel interface
Printer
Serial interface
Terminal or Modem

SCSI Bus

SCSI controller

Disk controller
Disk 1
Disk2

CD-ROM controller
CD-ROM drive

Tape controller
Tape drive

Centronix interface standard
RS-232C interface standard