Chapter 7 - Input / Output

Luis Tarrataca
luis.tarrataca@gmail.com

CEFET-RJ

Table of Contents I

1 Motivation

2 Generic I/O Module

3 Generic Peripheral Device

4 I/O Modules

Module Function

Control and Timing

Processor Communication

Device communication

Data buffering

Error detection

I/O Module structure

Programmed I/O

Table of Contents II

I/O Commands

Interrupt-Driven I/O

Interrupt Processing

Interruption-based design issues

Table of Contents I

5 DMA Module

6 Where to focus your study

7 References

Motivation

Computers need the ability to communicate with I/O sources:

- Input: keyboard, mouse, etc.
- Output: monitor, printer, etc.
- Different devices have different requirements:
    - Peripheral logic;
    - Peripheral speed;
    - Peripheral data protocol.

Generic I/O Module

How can we perform this type of communication between processor and external devices?

How can we perform this type of communication between processor and external devices?

Why not connect the peripherals directly to the bus? Any ideas?

Why not connect the peripherals directly to the bus? Any ideas?

Reason 1: There are a wide variety of peripherals:

- Each with its method of operation;
- Impractical to incorporate diverse set of logic within the processor;

Why not connect the peripherals directly to the bus? Any ideas?

Reason 2: Data transfer rates:

- Peripherals often operate much slower than processor speed or memory.
- Impractical to use the high-speed system bus to communicate directly;

Why not connect the peripherals directly to the bus? Any ideas?

Reason 3: Data transfer rates:

- On the other hand:
  - transfer rate may also be faster than that of the memory/processor;
  - mismatch would lead to inefficiencies if not managed properly.

Why not connect the peripherals directly to the bus? Any ideas?

Reason 4: Data formats:

- Peripherals often use different communication protocol:
  - data format not necessarily the same as the computer;
  - word length not necessarily the same as the computer;
- The processor would have to convert these back-and-forth...

Succinctly:

- Processor speed is usually faster than peripherals:
- No need to slow down the processor to do interaction with I/O sources!

What can be done to solve this problem? Any ideas?

What can be done to solve this problem? Any ideas?

Create a separate entity responsible for managing I/O sources:

- Manage the communication with the peripherals;
- Store the data input / output from and to peripherals;
- Detect errors.

This is known as the **I/O module**

Generic I/O Module

An I/O module is required:

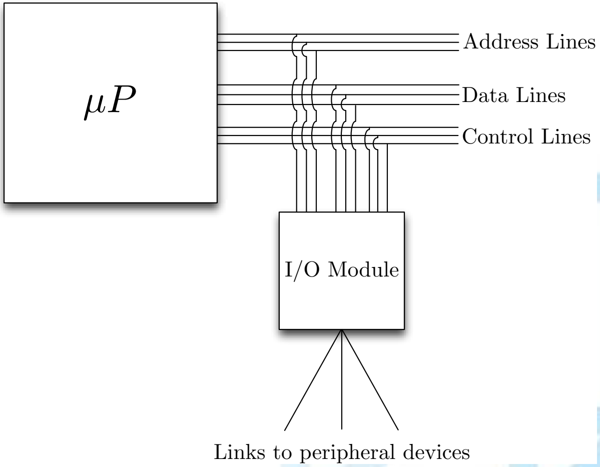


Figure: Generic Model of an I/O Module. (Source: (**?**))

Generic Peripheral Device

There is a wide assortment of external I/O devices:

- Means of exchanging data between environment and computer;
- External devices attaches to the computer by a link to an I/O module:
    - link is used to exchange control, status, and data;

We can broadly classify external devices into three categories:

- **Human readable:** Suitable for communicating with the computer user:
  - *E.g.:* video display terminals and printers.
- **Machine readable:** Suitable for communicating with equipment
  - *E.g.:* magnetic disks, tape systems, sensors and actuators
- **Communication:** Suitable for communicating with remote devices:
  - *E.g.:* wifi, modem

What do you think are the main components of an external device? Any ideas?

In very general terms:

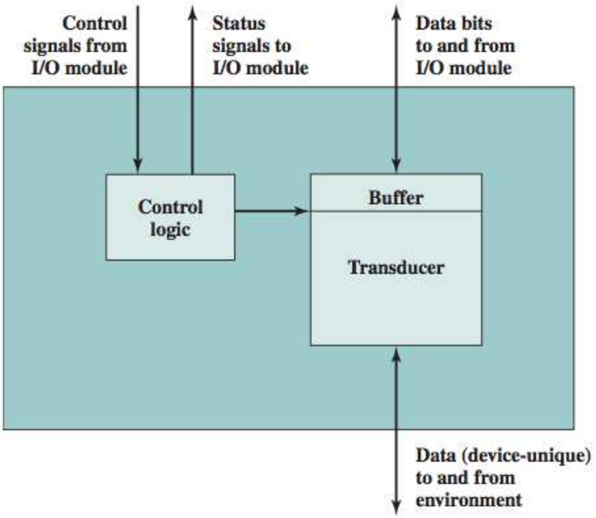

Control signals from I/O module
Status signals to I/O module
Data bits to and from I/O module
Control logic
Buffer
Transducer
Data (device-unique) to and from environment

Figure: Block Diagram of an External Device. (Source: (**?**))

Control Signals:

- To determine the function the device will perform, *e.g.:*
  - Send data to the I/O module;
  - Accept data from the I/O module;
  - Report status;
  - Perform function particular to the device (*e.g.:* position a disk head)

Data signals:

- Set of bits to be sent to or received from the I/O module;

Status signals:

- To indicate the state of the device, *e.g.:*
    - READY/NOT-READY to show whether the device is ready for data transfer

Control logic:

- Controls the device in response to direction from the I/O module;

Transducer:

- Converts data:
  - From electrical to other forms of energy during output;
  - And from other forms to electrical during input

Buffer:

- To hold data.

Module Function

Major functions of an I/O module fall into the following categories:

- Control and timing;
- Processor communication;
- Device communication;
- Data buffering;
- Error correction.

Control and Timing (1/4)

At any time the processor:

- May communicate with external devices in unpredictable patterns;
- This depends on the program's need for I/O;

This means that:

- Main memory and the system bus must be shared with the I/O function.
- Thus, the I/O function includes a control and timing requirement:
    - Coordinates flow of traffic between resources;

Control and Timing (2/4)

What do you think are the sequence of steps required for the processor to interact with the I/O module? Any ideas?

Control and Timing (3/4)

Interaction example between processor and I/O module:

1. Processor interrogates the I/O module to check device status;
2. I/O module returns the device status.
3. If the device is operational and ready to transmit:
    - Processor requests data transfer to I/O module;
    - I/O module issues a data transfer command to device ;
4. I/O module obtains data from device.
5. Data are transferred from the I/O module to the processor.

Control and Timing (4/4)

Each one of these interactions between processor and I/O module:

- involves one or more bus arbitrations

Preceding example illustrated that:

- I/O module must communicate with the processor and external device

What are the steps required to perform such a communication? Any ideas?

Processor Communication

Processor communication involves the following (1/3):

- **Command decoding:**
  - I/O module accepts commands from processor:
    - Control bus signals
  - *E.g.:* I/O module commands for a disk drive:
    - READ SECTOR;
    - WRITE SECTOR;
    - SEEK track number;
    - SCAN sector ID.

Processor Communication

Processor communication involves the following (2/3):

- **Data:**
  - Exchanged between the processor and the I/O module (data bus).
- **Status reporting:**
  - Important to know the status of the I/O module;
  - *E.g.:*
    - Processor asks to read data from I/O module;
    - But I/O module may not be ready;
    - This happens because it may still be working on previous I/O command;
    - This needs to be reported with a status signal;

Processor Communication

Processor communication involves the following (3/3):

- **Address recognition:**
  - Recall that each word of memory has an address;
  - The same is valid for each I/O device;
  - **I/O module has one unique address for each peripheral it controls**
    Remember from the laboratory?
    - Memory address for screen (FFFEh);
    - Memory address for keyboard (FFFFh);
    - Memory address for timer (FFF7h);

Device communication

I/O module must be able to perform device communication, involving:

- Commands;
- Status information;
- Data

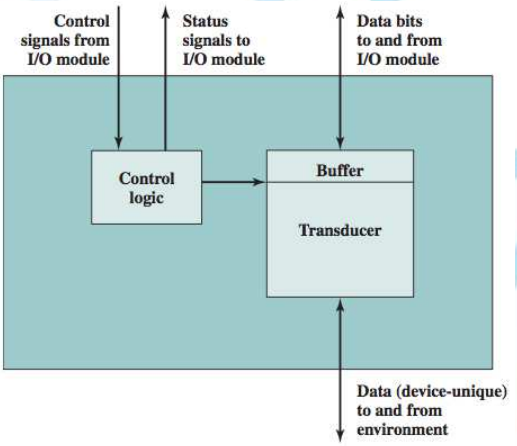

Control signals from I/O module
Status signals to I/O module
Data bits to and from I/O module
Control logic
Buffer
Transducer
Data (device-unique) to and from environment

Figure: Block Diagram of an External Device. (Source: (**?**))

Data buffering (1/4)

An essential task of an I/O module is data buffering:

- High transfer rate between main memory and processor;
- Whereas the rate is orders of magnitude lower for external devices.

Data buffering (2/4)

Main memory data are sent to an I/O module in a rapid burst:

- Data are buffered in the I/O module;
- Then sent to peripheral device at a data rate it can sustain;

Data buffering (3/4)

In the opposite direction (I/O module to main memory):

- Data are buffered from peripheral;
- This is done so as not to tie up the memory in a slow transfer operation.

Data buffering (4/4)

If the I/O device operates at a rate higher than the memory access rate:

- I/O module performs the needed buffering operation.

Error detection

I/O module is often responsible for error detection and reporting, *e.g.:*

- Mechanical and electrical malfunctions reported by the device:
  - *E.g.:* paper jam, bad disk track
- Unintentional bit changes during device transmission:
  - Remember error detection and correction? (Chapter 5)

I/O Module structure

Lets take a closer look at a generic I/O module.

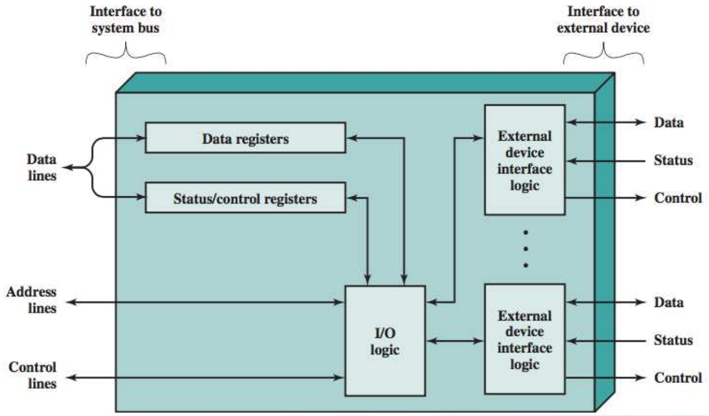


Figure: Block Diagram of an of an I/O Module. (Source: (**?**))

Organization of a generic I/O module (1/2):

- Module connects to computer through a set of signal lines;
- Data transferred to and from the module are buffered in data registers.
- Status registers provide status information:
  - Also function as control registers, to accept processor control info;

Organization of a generic I/O module (2/2):

- Logic within the module interacts with the processor via control lines:
  - Processor uses the control lines to issue commands to the I/O module;
  - Some of the control lines may be used by the I/O module
    - *E.g.* arbitration and status signals;
- Module must also be able to recognize and generate addresses:
  - for each device it controls
- I/O module contains logic specific for a set of interfaces;

I/O module allows processor to view peripherals in a simple way:

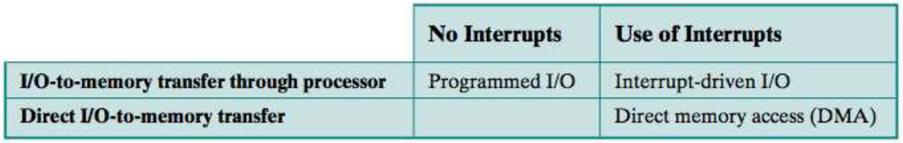
- Presents a high-level interface to the processor;
- Taking most of the I/O processing burden away from the processor;
- Also called an I/O processor =)

Now that we have an idea of the main components...

How can we manage the communication between the μP and the I/O module?

- Any ideas?

Essentially, there are three techniques are possible for I/O operations:



	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Figure: I/O Techniques (Source: [**?**])

Lets have a look at each one of these =)

Programmed I/O

Data are exchanged between the processor and the I/O module:

1. Processor executes program controlling I/O operation;
   - *E.g.:* sensing device status, read/write command, data transfer.
2. Once the processor issues a command to the I/O module:
   - Processor must wait until the I/O operation is complete;
3. If the processor is faster than the I/O module:
   - Wasteful of processor time =’(

I/O Commands

To execute an I/O-related instruction:

- Processor issues an address:
    - Specifying the particular I/O module and external device;
- An I/O command which can be of the following type:
    - Control, Test, Read and Write

I/O commands the processor can issue are of the following type (1/2):

- **Control:** Used to activate a peripheral and tell it what to do:
  - *E.g.:* Rewind magnetic-tape; move to HD track;
- **Test:** test I/O module and its peripherals. *E.g.:*
  - Is the peripheral powered on?
  - Is the peripheral available for use?
  - Has the most recent I/O operation completed? Did any errors occur?

I/O commands the processor can issue are of the following type (2/2):

- **Read:** I/O module obtains a data item from the peripheral:
  - Placing the data in an internal buffer;
  - Processor requests I/O module to place data on bus;
- **Write:** I/O module writes a data item to the peripheral:
  - I/O module reads data from bus;
  - I/O module transmits data to peripheral;

In flowchart form:

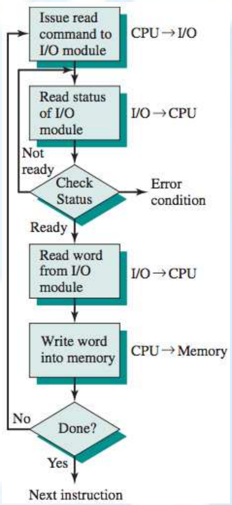


Figure: Programmed I/O technique for input of a block of data (Source: (**?**))

Can you characterize the system from an efficiency perspective?

Very wasteful! Recall that:

- Processor issues a command to the I/O module:
    - then waits for I/O operation to complete.
    - While waiting, processor repeatedly interrogates status of I/O module.
- If processor is faster than I/O module: wasteful of processor time.

Is it possible to do any better?

Interrupt-Driven I/O

What is the ideal scenario for processor performance?

- Do not wait for I/O module;
- Instead, continue processing other tasks;
- And be notified when I/O module has something for processor;

This is the concept of **interruption**, *I.e.*:

1. Ask for something from the I/O module;
2. Continue processing without waiting for the I/O module;
3. Be interrupted when the I/O module has something ready.

From the point of view of the **I/O module**:

4. Module waits for processor to request data:
5. When request is made:
   - When possible: module interacts with peripheral;
   - Once the data is completely buffered;
     - data are place on data bus;
6. An interrupt signal is sent to the processor over a control line;
7. Module becomes available for another I/O operation.

From the point of view of the processor (1/2):

1. A READ command is issued to I/O module;
2. Processor goes off to do something else;
3. Processor checks for interrupts at the end of each instruction cycle;

From the point of view of the processor (2/2):

4. When the interrupt from the I/O module occurs:
   - processor saves program context;
   - processor proceeds to read data from I/O module
   - processor stores data in memry;
5. Processor then restores previous program context;
6. Processes resumes execution of previous program

In flowchart form:

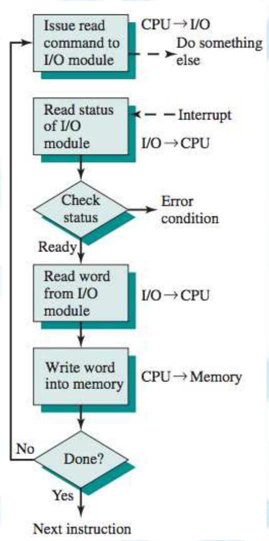


Figure: Interrupt-driven I/O (Source: (**?**))

Interrupt Processing

Lets take a closer look at the interruption-based strategy.

- Interruption triggers a number of events
    - Processor, hardware and software.
- Automatically, we can pose a series of questions:
    - What happens to the program that is executing?
    - What happens to the processor?
    - How is the interruption processed?

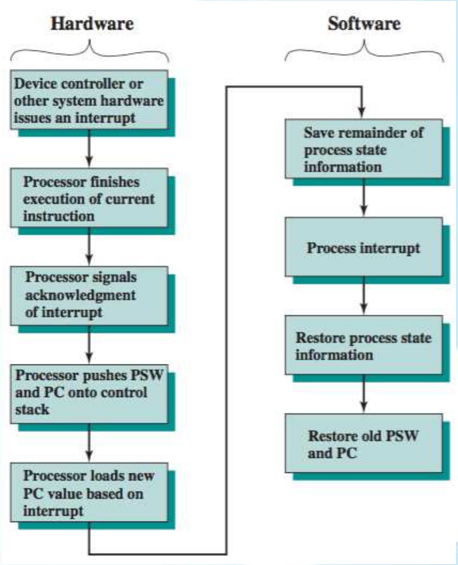

Hardware
Software
Device controller or other system hardware issues an interrupt
Processor finishes execution of current instruction
Processor signals acknowledgment of interrupt
Processor pushes PSW and PC onto control stack
Processor loads new PC value based on interrupt
Save remainder of process state information
Process interrupt
Restore process state information
Restore old PSW and PC

Figure: Simple Interrupt Processing (Source: (**?**))

When an I/O device completes an I/O operation (1/4):

1. Device issues an interrupt signal to the processor.
2. Processor finishes execution of the current instruction
   - before responding to the interrupt;
3. Processor tests for an interrupt:
   - determines if there is one;
   - if one exists, sends an acknowledgement signal to peripheral;
     - acknowledgment allows the device to remove its interrupt signal.

When an I/O device completes an I/O operation (2/4):

4. Processor needs to transfer control to the interrupt routine;
   - This is done by saving the program context:
     - processor status word;
     - program counter;
5. Processor then loads the program counter associated with the interrupt-handling routine.

When an I/O device completes an I/O operation (3/4):

6. Interruption routine may use the registers:
   - This means that these registers need to be saved;
   - Remember Push and Pops from the laboratory?
7. Typically, the interrupt handler will begin by saving all registers on the stack;
8. Interrupt handler next processes the interrupt

When an I/O device completes an I/O operation (4/4):

9. When interrupt processing is complete:
   - the saved registers need to retrieved from the stack and restored;
10. Final act is to restore the PSW and program counter
    - Next instruction to be executed will be from the previously interrupted program.

Recall the instruction cycle:

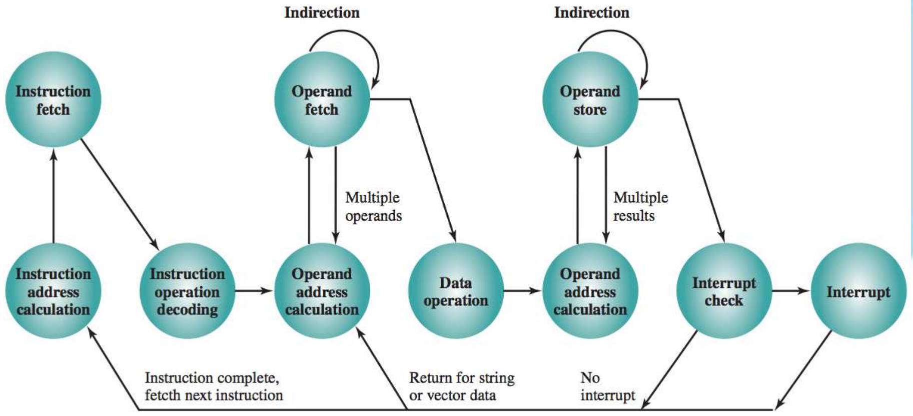

Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetcth next instruction
Return for string or vector data
No interrupt

Figure: Instruction Cycle State Diagram (Source: (**?**))

Interruption-based design issues

Almost invariably there will be multiple I/O modules...

How does the processor determine which device issued the interrupt?

If multiple interrupts have occurred, how does the processor decide which one to process?

How does the processor determine which device issued the interrupt?

- **Multiple interrupt lines:**
  - Impractical to dedicate more than a few bus lines to interrupt lines;
  - Low number of interruption lines still means that we have to multiplex them.
- **Software poll:**
  - Processors calls an interrupt routine that polls each I/O module.
  - The I/O module responds positively if it set the interrupt.
  - Disadvantage: time consuming!

How does the processor determine which device issued the interrupt?

- **Daisy chain method:**
  - All I/O modules share a common interrupt request line;
  - When an interruption is detected an interrupt acknowledge (ACK) is sent;
  - ACK goes through the I/O modules until it gets to the requesting module;
  - Requesting module responds by placing a word on the data lines;
  - Word is either:
    - Address of the I/O module or;
    - Address of an adequate interruption handling technique.

If multiple interrupts have occurred, how does the processor decide which one to process?

- Multiple interrupt lines
  - Each line can have a predetermined priority;
  - Just choose the highest priority interrupt line.
- Software polling / Daisy chain:
  - The order with each the modules are polled determines the priority.

Interruption-based model is computationally more efficient, however:

- Processor has a ``test and service a device´´ rate:
    - This limits the I/O transfer rate...
    - And accessing main memory is expensive...
- Processor is tied up in managing an I/O transfer:
    - A number of instructions must be executed for each I/O transfer
    - This time could be spent doing something more useful: real processing!

Interrupt I/O is more efficient than programmed I/O:

- Eliminates needless waiting...

Despite the improvement, can you see any potential upgrade that can be performed with interrupt I/O?

Despite the improvement, can you see any potential upgrade that can be performed with interrupt I/O?

Interrupt I/O still consumes a lot of processor time:

- Data is exchanged between memory and I/O module...
- But this exchange still needs to go through the processor....
- **Processor spends time transferring data**
    - while it could be doing something more useful..

DMA module

Idea: Copy data directly to memory, bypassing processor:

- Memory accesses are performed by DMA module;
- Unburdens the processor;
- Combine with interruption scheme for optimum efficiency.

This strategy is called **Direct Memory Access (DMA)**

DMA involves an additional module on the system bus:

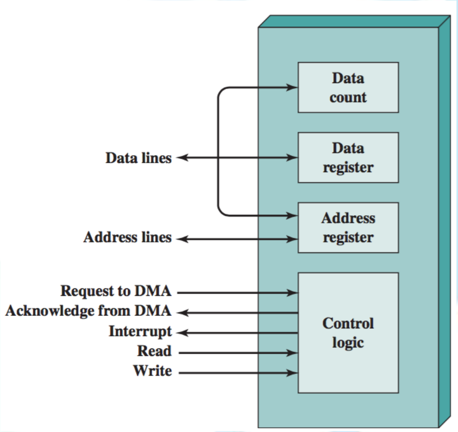


Figure: Typical DMA block diagram (Source: (**?**))

- Uses the bus only when the processor does not need to or...
- ...Forces the processor to suspend bus operations temporarily;

Processor issues a command to the DMA module:

- Command contains (1/2):
  - Whether a read or write is requested:
    - transmitted over the bus control lines;
  - Address of the I/O device involved
    - transmitted over the bus data lines;

Processor issues a command to the DMA module:

- Command contains (2/2):
  - Starting location in memory to read from or write to:
    - communicated on the data lines and...
    - stored by the DMA module in its address register;
  - Number of words to be read or written:
    - communicated via the data lines and stored in the data count register;

Processor then continues with other work, *i.e.:*

- I/O operation delegated to DMA module;
- DMA module transfers block of data:
  - bypassing the processor;
- When the transfer is complete:
  - DMA module sends interrupt signal;
- Processor is involved only at:
  - beginning of transfer;
  - end of transfer;

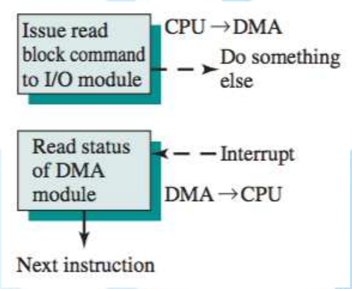

Issue read block command to I/O module
CPU → DMA
Do something else
Read status of DMA module
Interrupt
DMA → CPU
Next instruction

Figure: DMA-driven I/O (Source: (**?**))

Now that we know more about the DMA module:

How can the DMA module use the bus? Any ideas?

Idea:

- Processor is potentially suspended just before it needs to use the bus;
- DMA module transfers one word and returns control to the processor

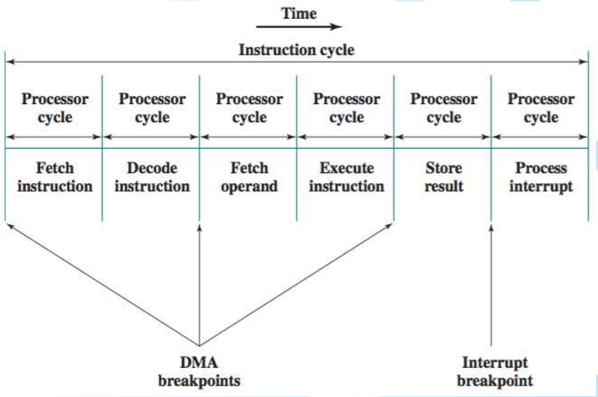


Figure: DMA and interrupt breakpoints during an instruction cycle (Source: (**?**))

Processor suspend procedure is not an interrupt:

- Processor does not save a context and do something else;
- Rather, the processor pauses for one bus cycle;
- Overall effect: processor executes more slowly;
- Still, DMA is more efficient than interrupt-driven or programmed I/O.

Now that we know more about the DMA module:

Where should the DMA module be placed? Any ideas?

DMA mechanism can be configured in a variety of ways (1/3):

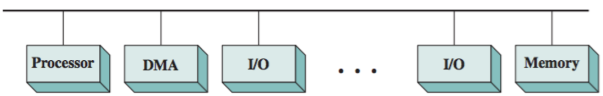


Figure: Single-bus, detached DMA (Source: (**?**))

In this example:

- **All modules share the same system bus;**
- DMA module uses programmed I/O to:
  - Exchange data between memory and an I/O module;
- Inexpensive but inefficient:
  - Each transfer of a word consumes multiple bus cycles.

DMA mechanism can be configured in a variety of ways (2/3):

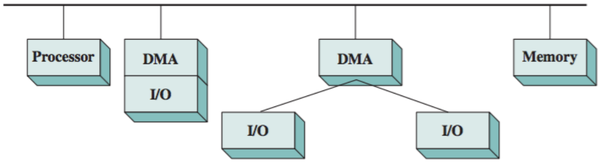


Figure: Single-bus, integrated DMA-I/O (Source: (**?**))

In this example:

- Number of bus cycles can be cut by integrating DMA and I/O;
- DMA module and I/O share a non-system bus connection.

DMA mechanism can be configured in a variety of ways (3/3):

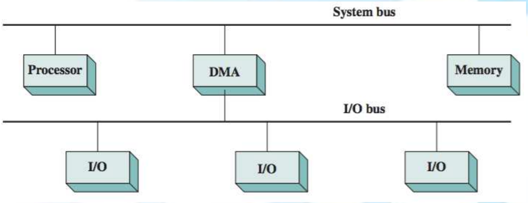


Figure: I/O bus (Source: (**?**))

In this example:

- I/O modules are connected to the DMA module using an I/O bus;
- Reduces the number of I/O interfaces in the DMA module to one;
- Providing for an easily expandable configuration.

In essence:

- DMA is a co-processor: unburdens CPU;
- I/O module also used to unburden CPU;

Where to focus your study

After this class you should be able to:

- Explain the use of I/O modules as part of a computer organization.
- Understand the difference between programmed I/O and interrupt-driven I/O and discuss their relative merits.
- Present an overview of the operation of direct memory access.

Less important to know how these solutions were implemented:

- details of specific programmable peripheral interfaces;
- details of specific DMA controllers;
- details of specific external interfaces.

Your focus should always be on the building blocks for developing a solution =)

References I

Stallings, W. (2015).
Computer Organization and Architecture.
Pearson Education.