18 I/O and Peripherals
Part 3: I/O Interfaces

CSC 230

Stallings: in chapter 7 (distributed)

M&H: in chapter 8 (distributed)

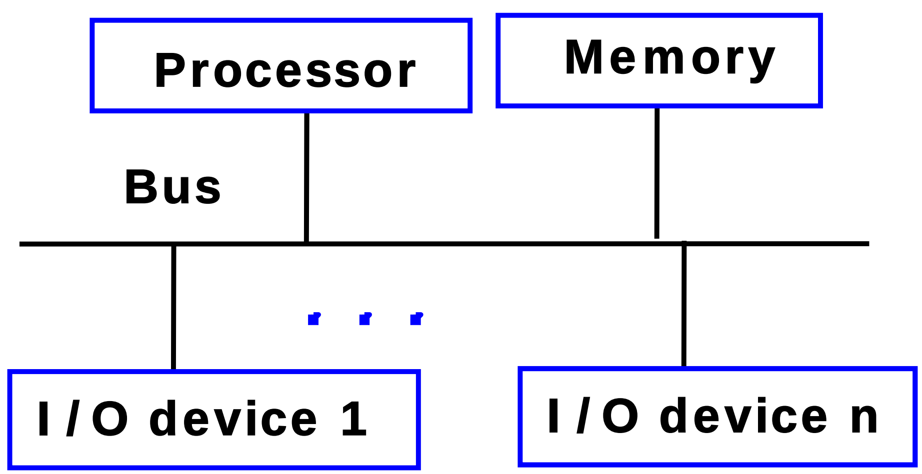
Input / Output: or general ability to exchange data

- **I/O refers to transmission of data**
  from one device (sender) to another (receiver)
- **I/O operations require controls**
  to coordinate the operation of the sender and receiver

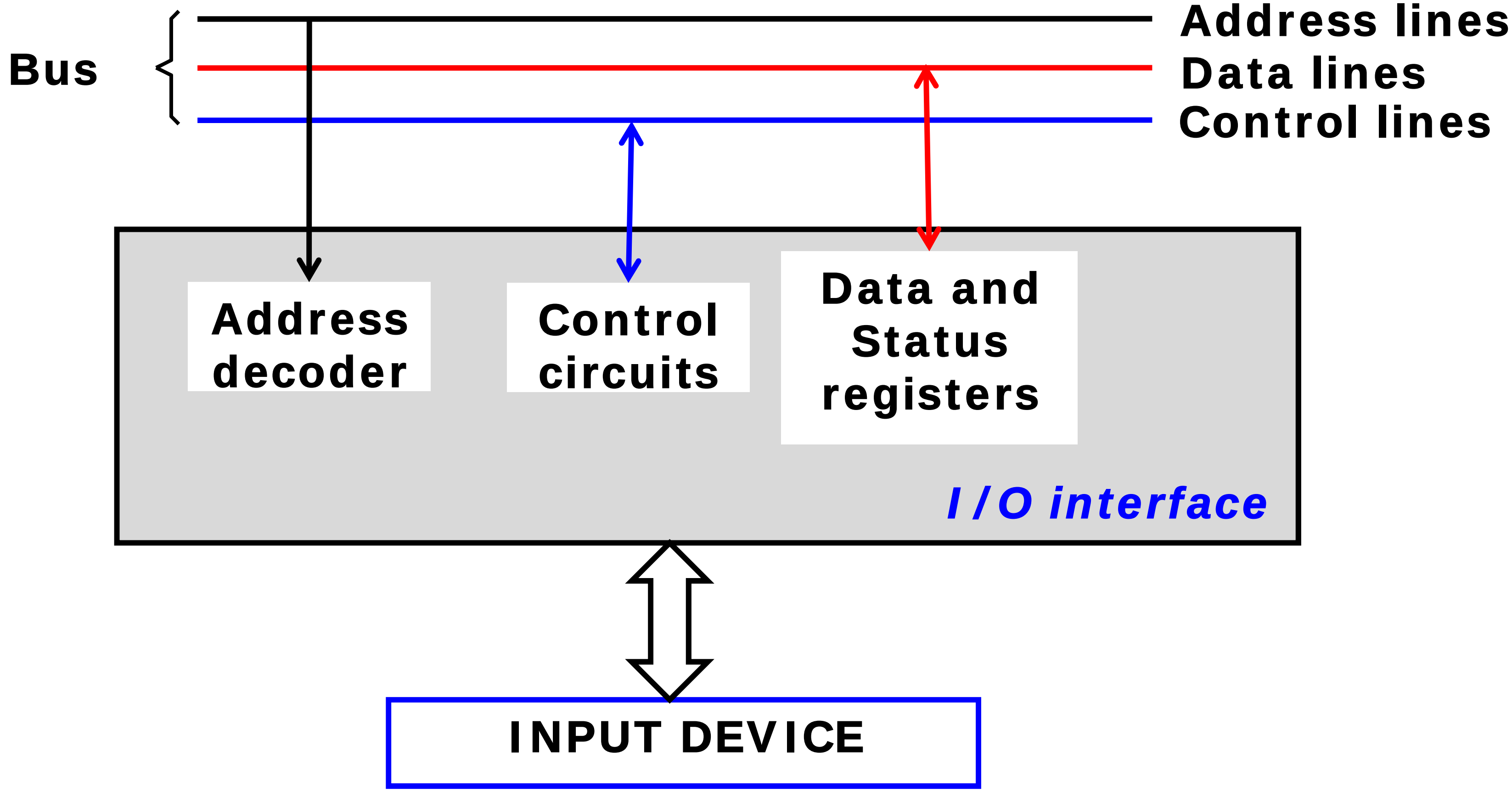
NEED:
- **DATA to transmit**
- **Control information**

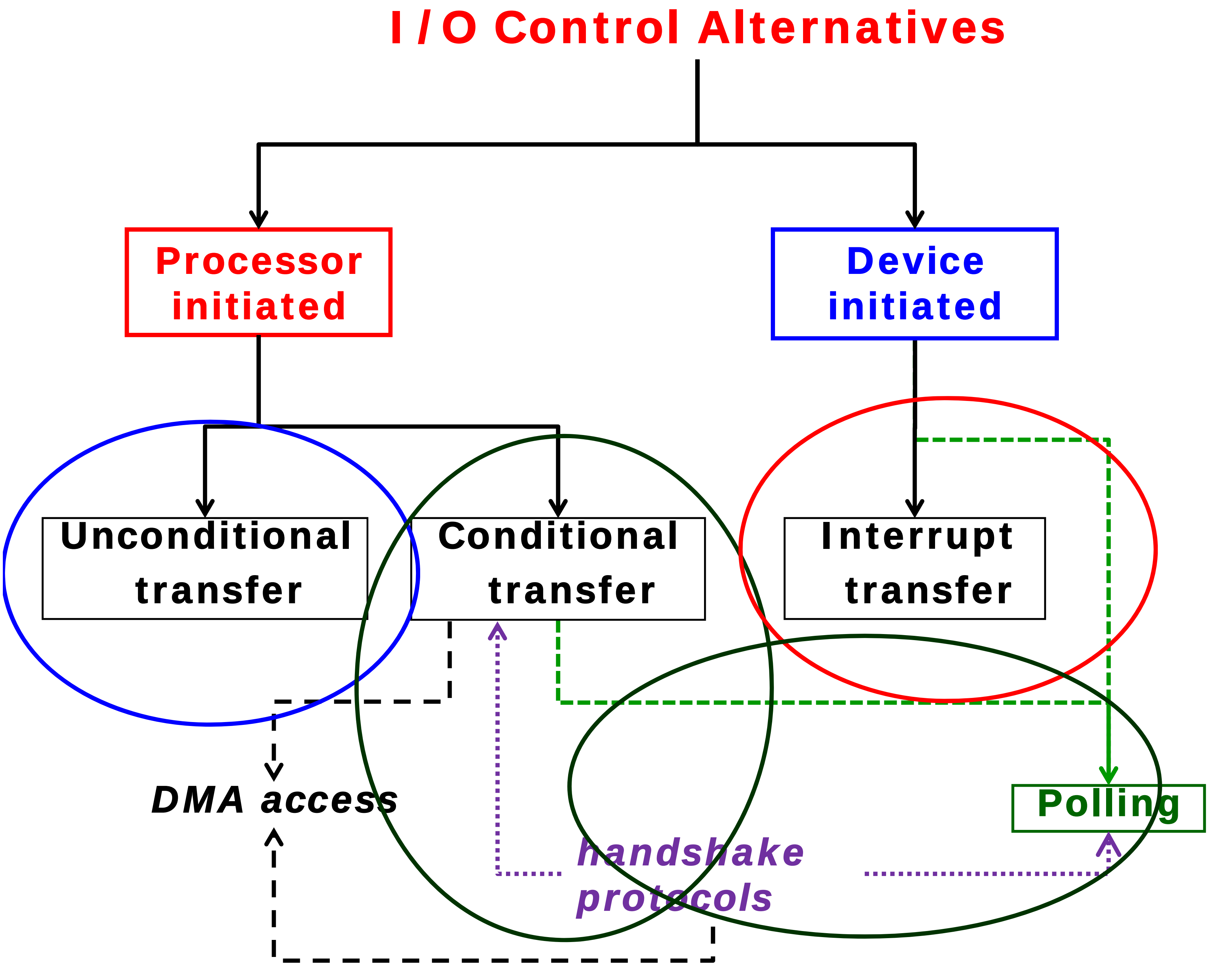
Typical information includes:

- ready to send
- clear to send
- data received
- errors
- etc.



I / O interface for an input device




I / O Control Alternatives
Processor initiated
Device initiated
Unconditional transfer
Conditional transfer
Interrupt transfer
DMA access
Polling
handshake protocols

I/O Control Alternatives (1): Processor initiated

Processor (CPU) initiates transfer (program controlled)

unconditional
device must always accept

Examples:
- **displaying results on LEDs**
- **reading a set of switches**

conditional
CPU checks if device accepts

Hand-shaking

Interchange of control information between the processor and a device to ensure *both are ready* for an I/O transfer

I/O Control Alternatives (2): Device initiated

Device initiates transfer

Interrupt driven

- **device sends an interrupt signal**
  ➔ it tells the processor to transfer data to/from the device

Polling (programmed)

- **device posts a flag signal in some location to state that data is ready to be transferred**
- **the processor checks it periodically**

What is Polling (aka Programmed I/O)?

Sampling the status of an external device which is repeatedly checked for readiness (by checking some posted signal)

Polling an I/O device can be seen as either a form of:

1. processor- initiated transfer
2. or device initiated transfer

VIEW 1:

A. the program/processor decides when the transfer should take place
B. processor periodically "polls" each device in turn and checks status before continuing with transfer

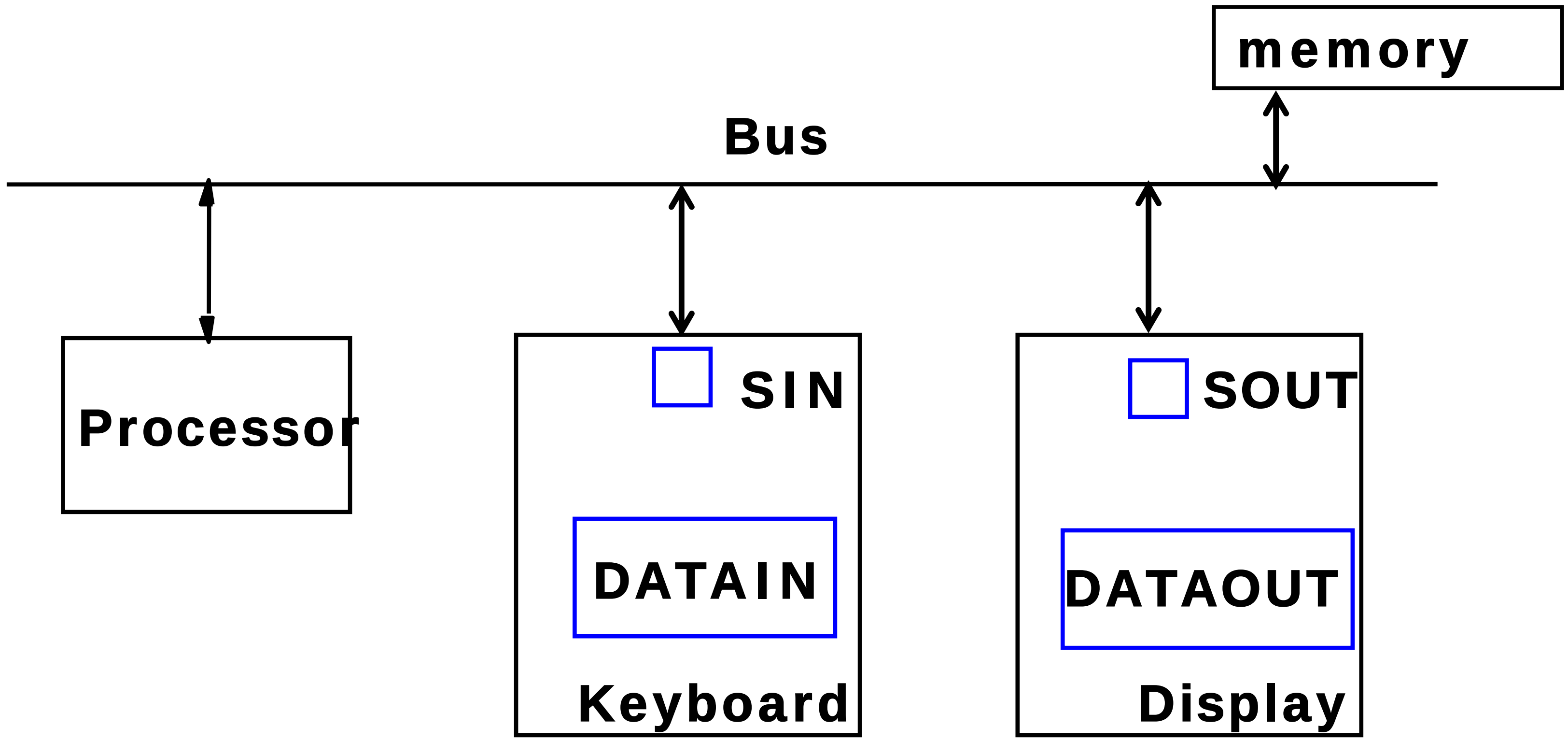
- conditional or unconditional
- e.g. is printer on?

VIEW 2:

I. device posts a "flag" to denote that it needs to transfer data
II. processor periodically "polls" each device in turn and checks flags
III. transfers are processed in turn by devices

Basic I/O operations: basic first example

- Task is to read characters from a keyboard input and display on screen
- Rates of data transfers are very different and different from clock speed

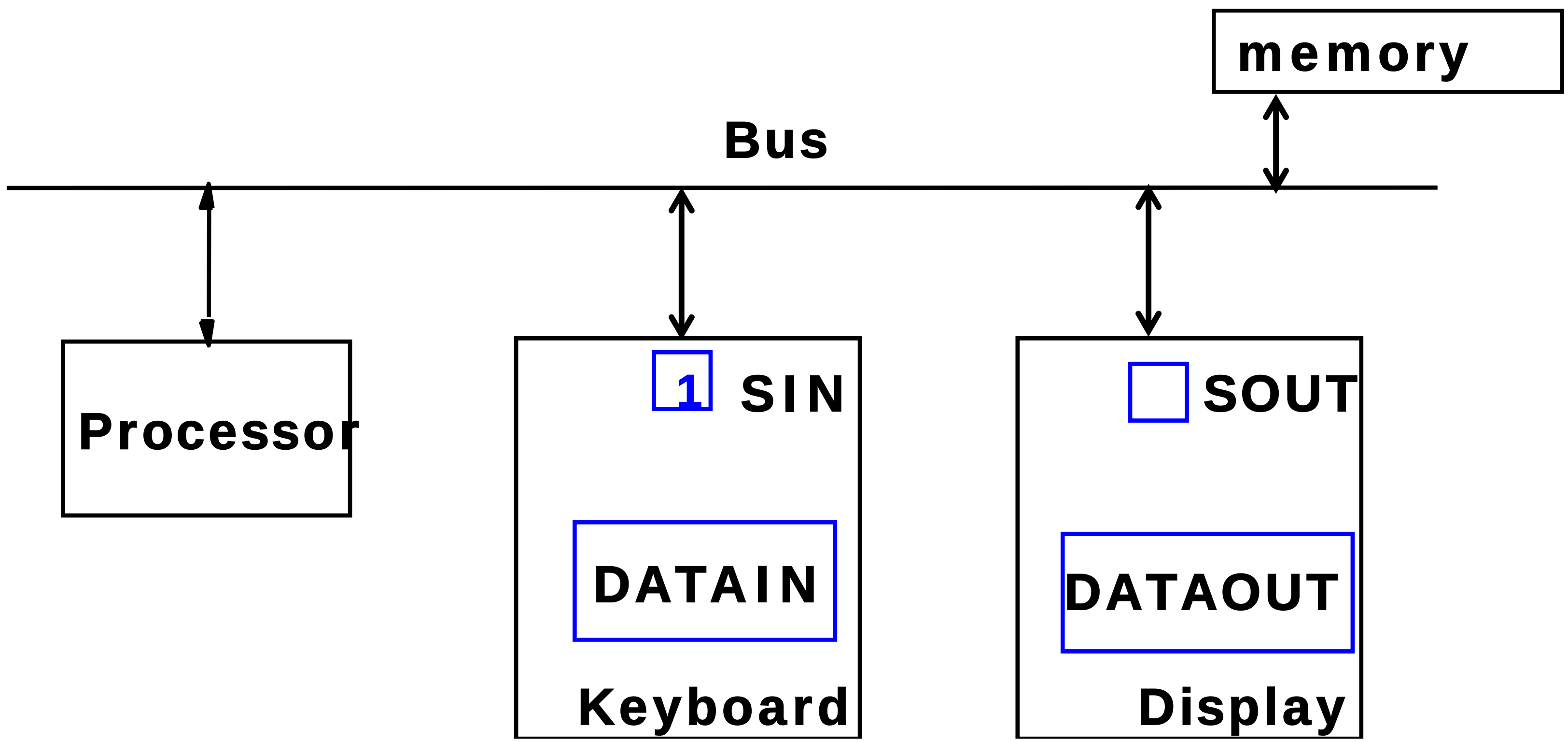


First steps in basic example

1. **Keyboard key struck**
   ➔ character stored in DATAIN ➔ set SIN to 1
2. **Program in processor monitors SIN**
   ➔ If/when SIN = 1, then copy DATAIN to memory ➔ clear SIN
   (Polling protocol example)

Similarly for output

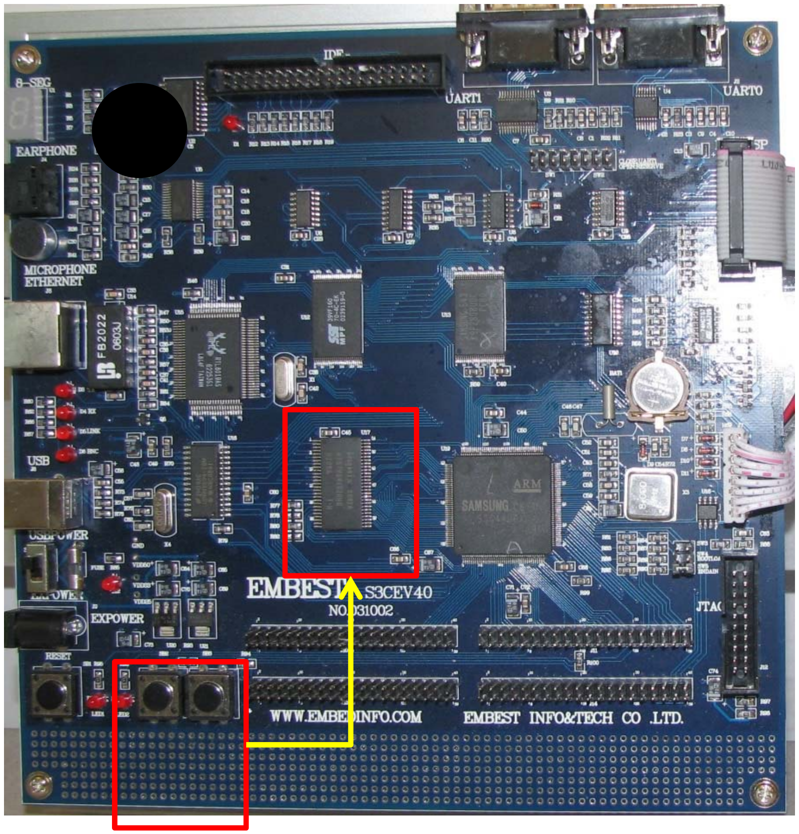
It can be very slow and cumbersome!

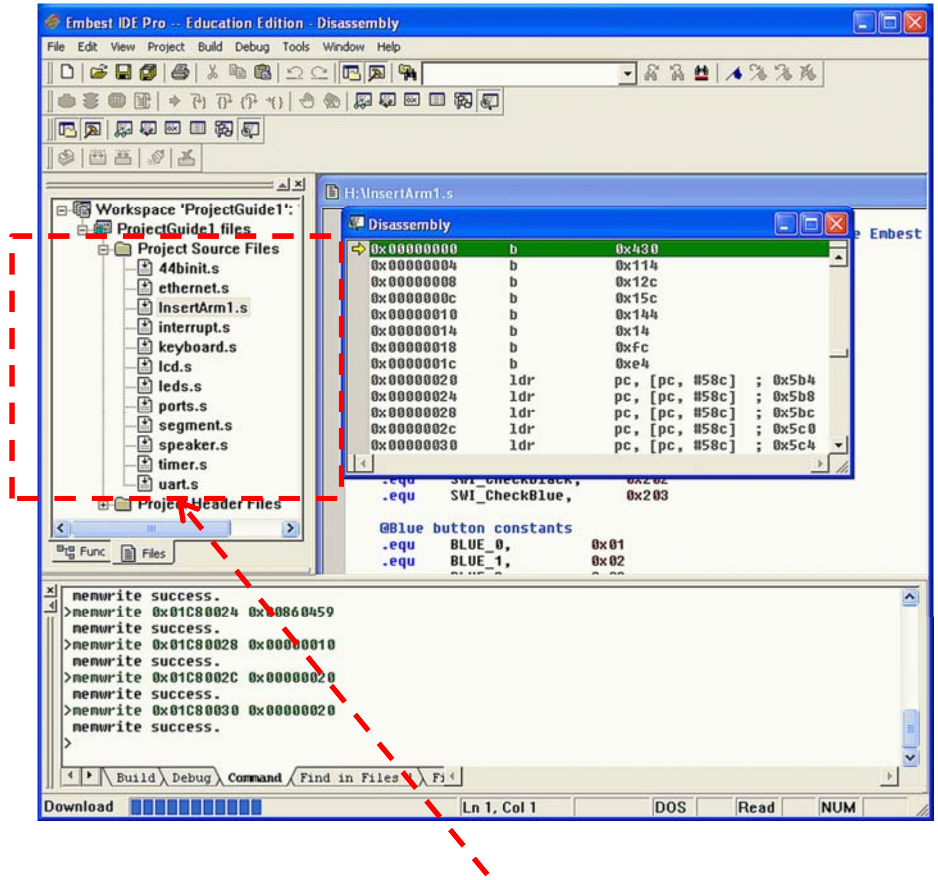


Consider the situation in Assignment 3

- ❑ **What happens when an LED is turned on?**
  - ➔ **issue SWI and action happens (no negotiation)**
- ❑ **What exactly happens when a button is pressed?**
  1. **time when button is pressed**
     - ➔ **capture/detection** ***(all under the hood for you)***
  2. **time when you check on the button with SWI**
     - ➔ **polling**
  3. **time when your code acts on result**
     - ➔ **action**

What happens in the project with SWI, the buttons, etc?





(1) Button press ➔ bit stored in board SDRAM

➔ Aynchronously and independently of any user code

➔ Done by built-in project code provided

➔ In between instructions as capture of true interrupt signal

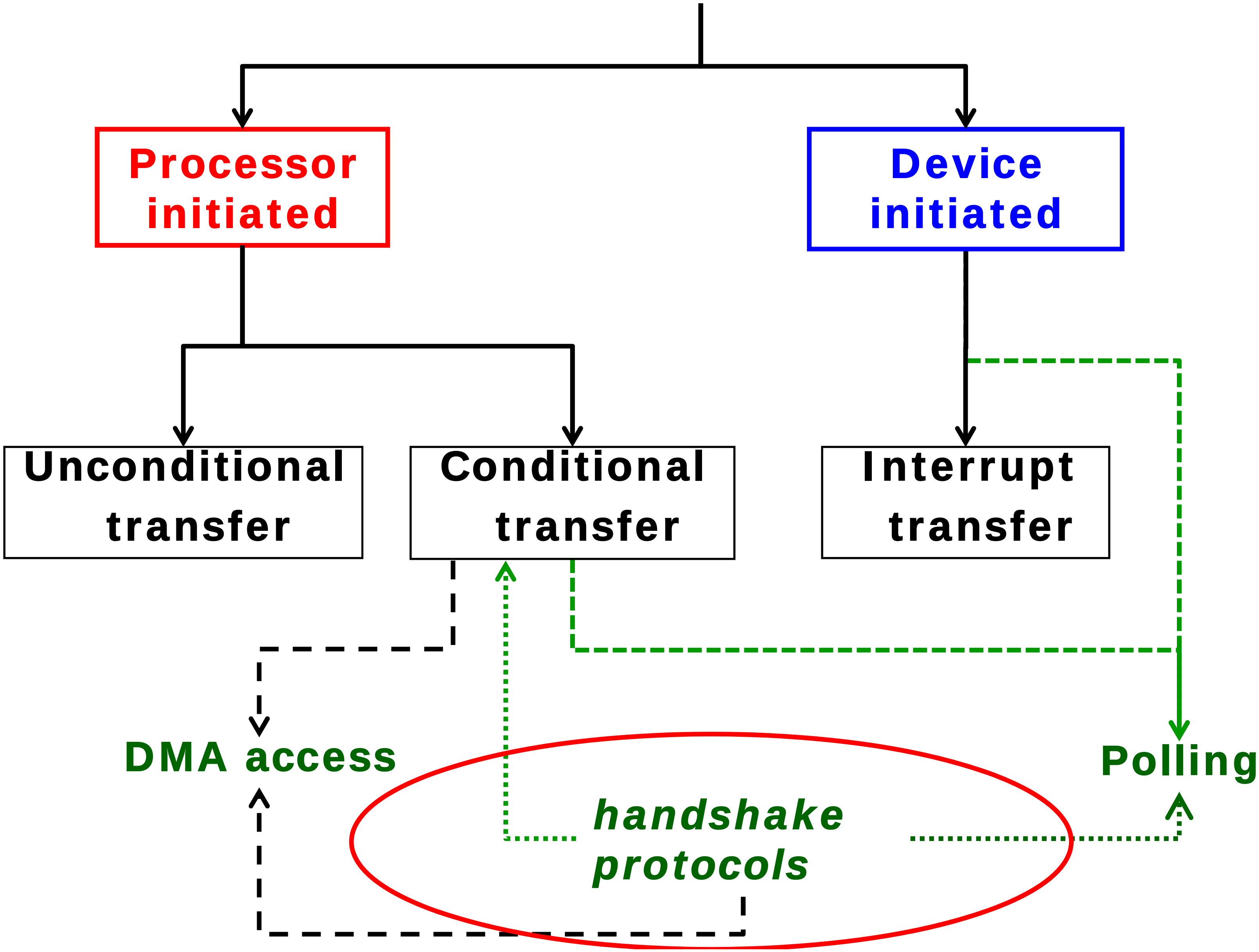
What happens in the project with SWI, the buttons, etc?

- ➔ Bit remains there until an appropriate SWI instruction issued by user code "polls" it

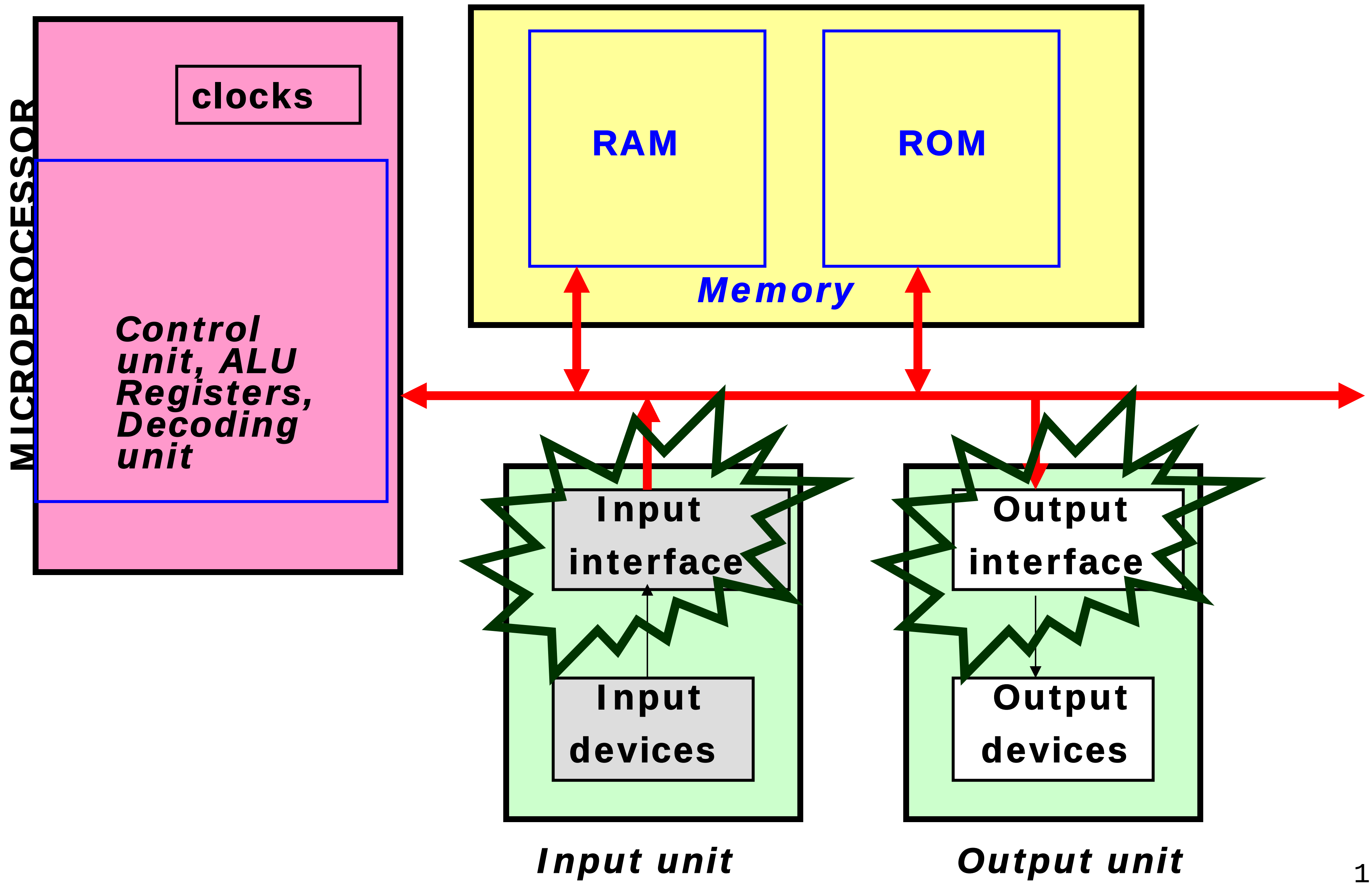
- ➔ Really fast polling basically is similar to capturing an interrupt

- ➔ In assignment 3, you are doing polling

I / O Control Alternatives: focus on handshake protocols



Organization and focus on Interfaces



Peripherals and Interfaces

(1) ***Many*** **categories of peripheral devices and** ***many*** **types of devices within each category**

- ✓ **Printers vs monitors vs disks vs CD ROM ➔ categories**
- ✓ **Which type of printer or which type of CD ROM? ➔ types**

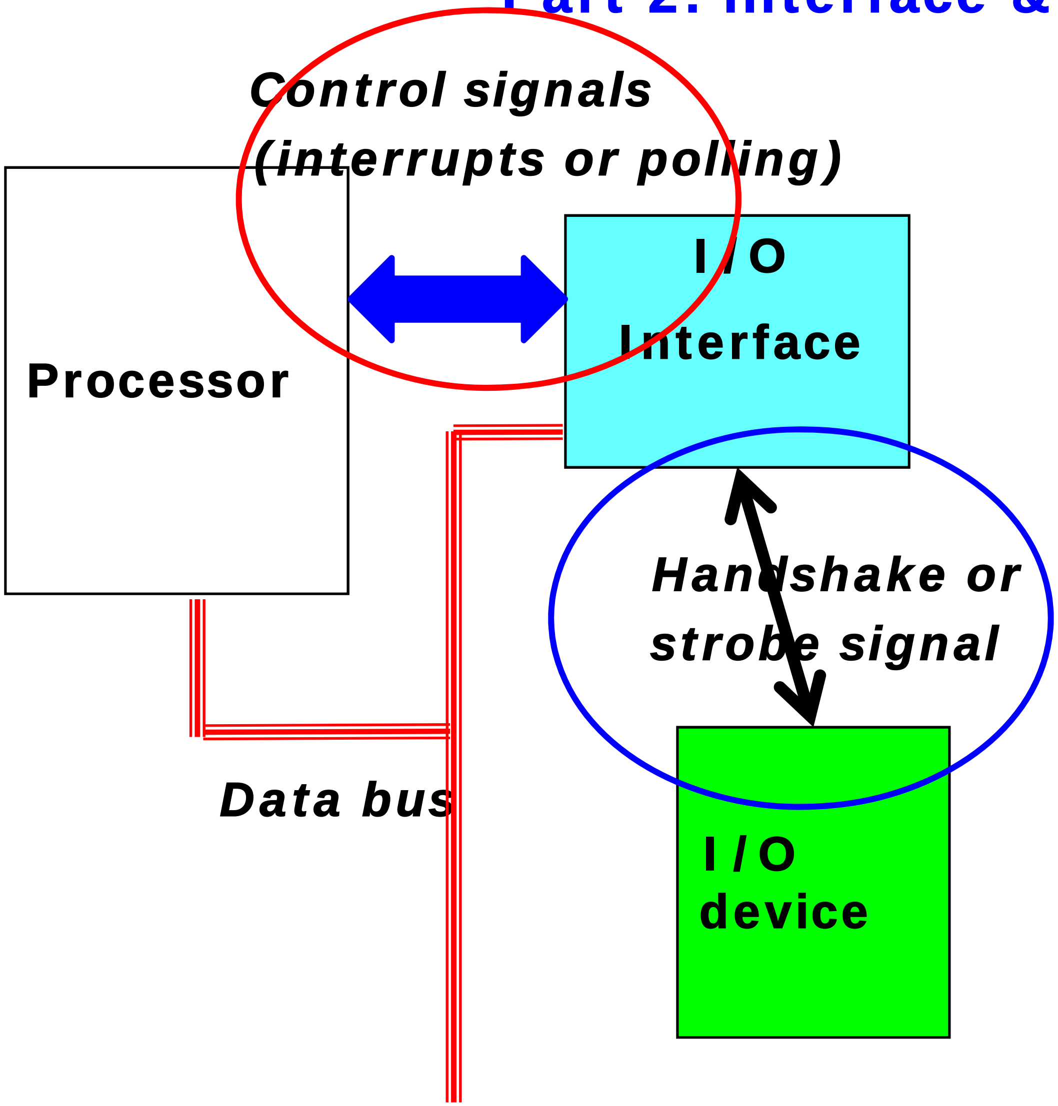
(2) Speed of peripherals very different from speed of CPU

- ❑ **Need interface devices to synchronize data transfer between CPU and I/O devices ➔** ***hardware!***
- ❑ **Interface normally has** ***control registers, status register, data registers, latches, data direction registers, control circuitry, chip enable, pins to connect to data bus***
- ❑ **For mainframes, I/0 interfaces are computer systems in their own right (*****channel programming in IBM*****)**

Transfer synchronization: 2 parts

Part 1: processor & interface

Part 2: interface & device



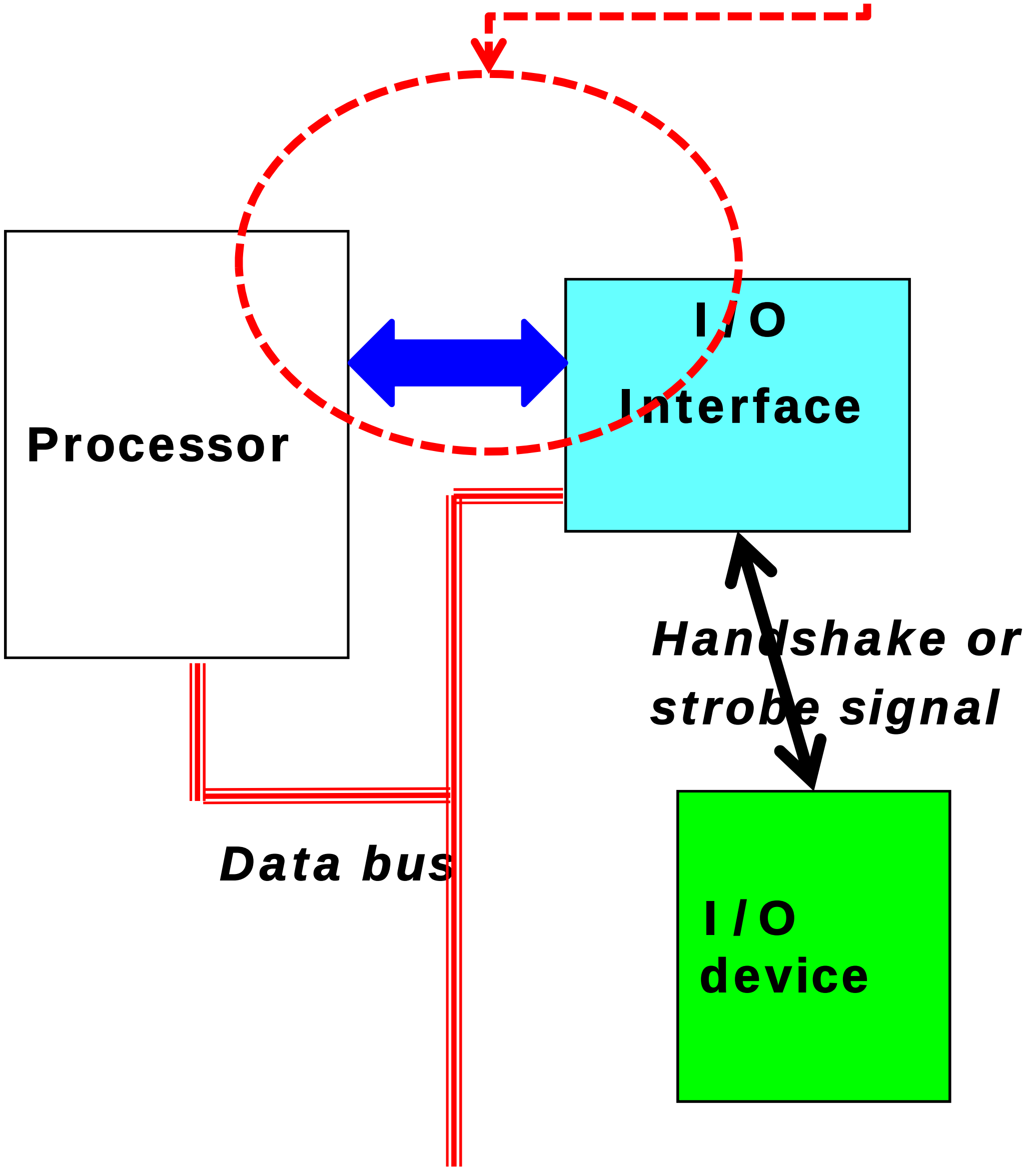
Consider that:

I. In I/O devices, often signals are mechanical or analog and need conversion to digital

II. Processor interacts *only* with interface

III. Must make sure data is valid at the correct moment in time

Transfer synchronization - part 1: processor & interface



Signals are controlled either with interrupts or polling

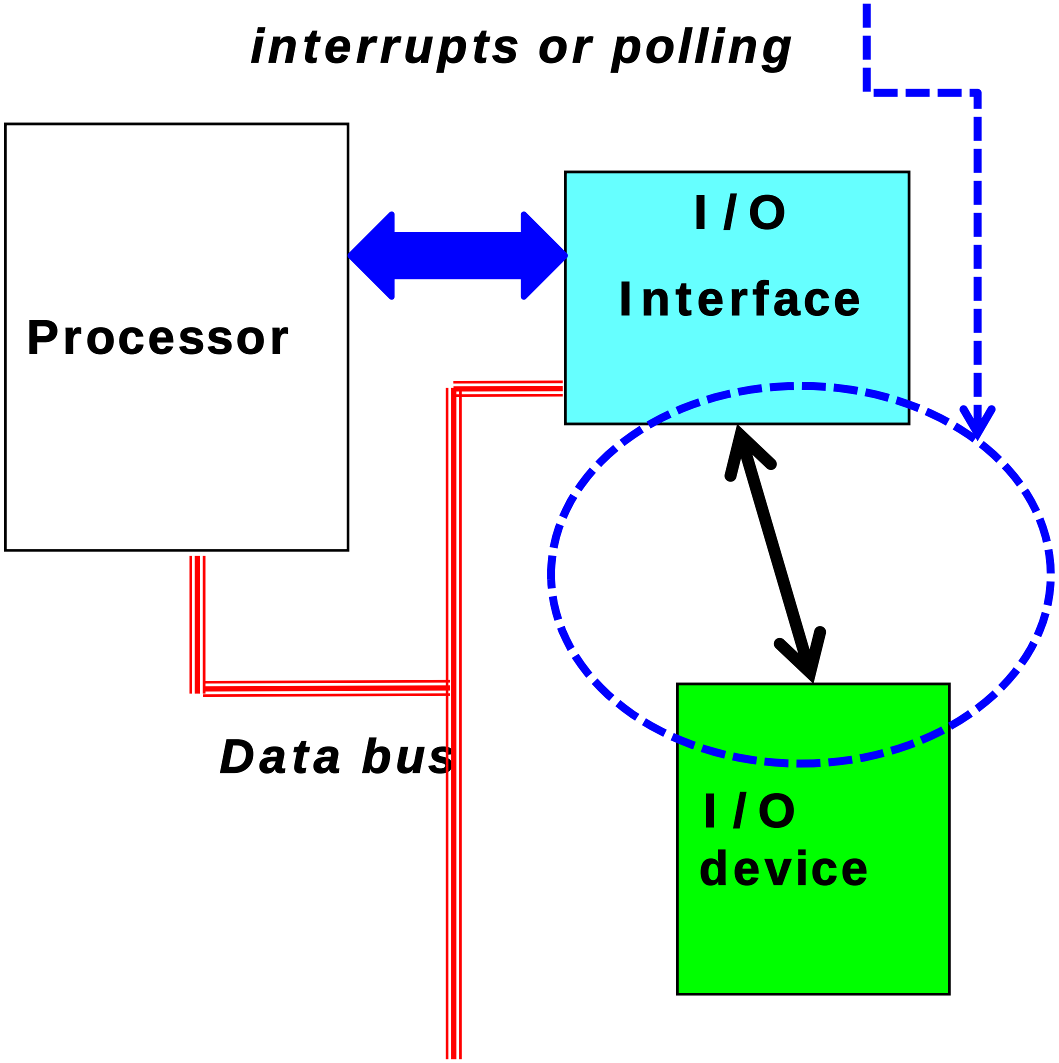
Two Methods for synchronization between a Processor and Interface Chips

(1) Polling: a technique used to check periodically if a particular event has occurred

- Interface uses a status bit in its own status register to indicate whether it has valid data (stored in some data register in device itself)
- Processor keeps checking to test if status bit = 1
- Processor could be tied up doing polling and cannot do anything else while, for example, waiting to read a piece of data!
- Simple to implement though and perfectly fine if program needs the data to continue anyway

(2) Interrupts: interface chip sends a signal and interrupts the processor (later)

Transfer synchronization - part 2: interface & device



Signals are controlled either with a handshake protocol or some strobe signal

Two Main Methods for synchronization between Interface Chips and I/O Devices

(1) Brute force: ***interface chip simply passes voltage levels on input or output port pins***

- ✓ **okay when timing is not important (e.g. LED's)**

(2) Handshake protocol (the most commonly used):

exchange of control signals during a data transfer between interface and I/O device

- ❑ **When timing is critical**

  e.g. should not send characters to print to printer if it is still printing previous ones sent ➔ it takes longer physically to print than to send characters to print

- ❑ **A general form of handshake is used whether the transfer is synchronous or asynchronous**

Example of *Input* Handshaking Protocol
(between interface and device)

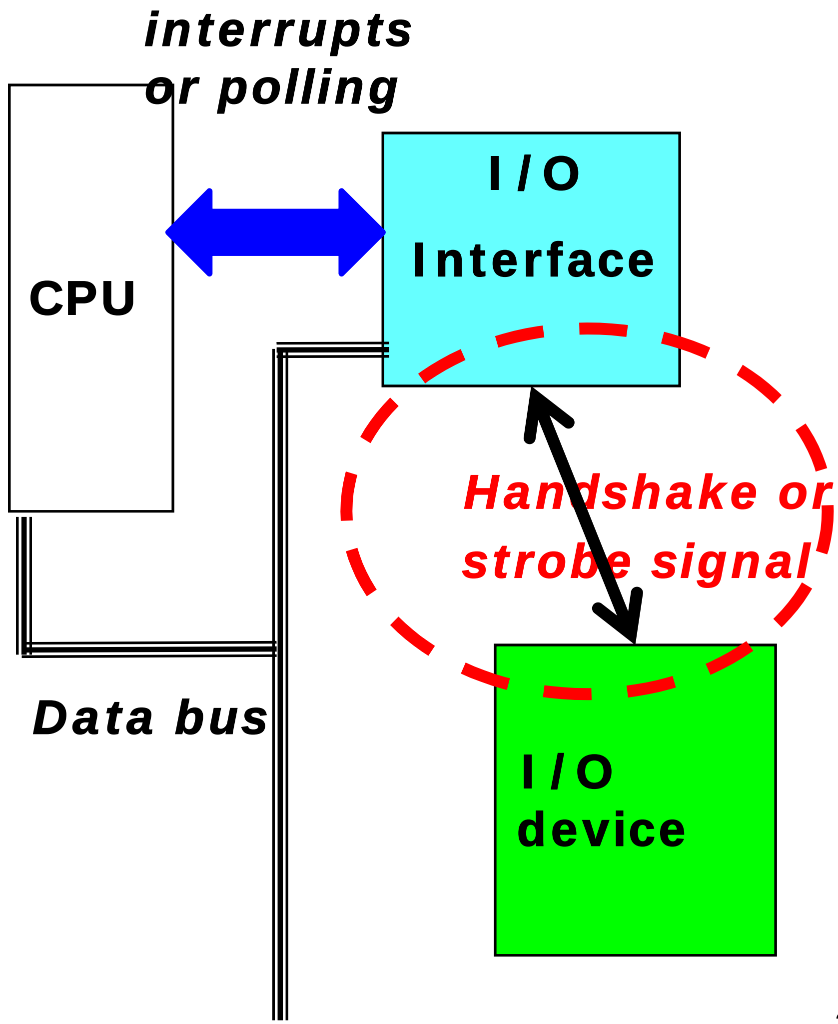
1. **Interface** asserts some signal, e.g. H1

   ➔ states intention to get a byte

2. **Input device** puts valid data on data port and asserts some signal, e.g. H2

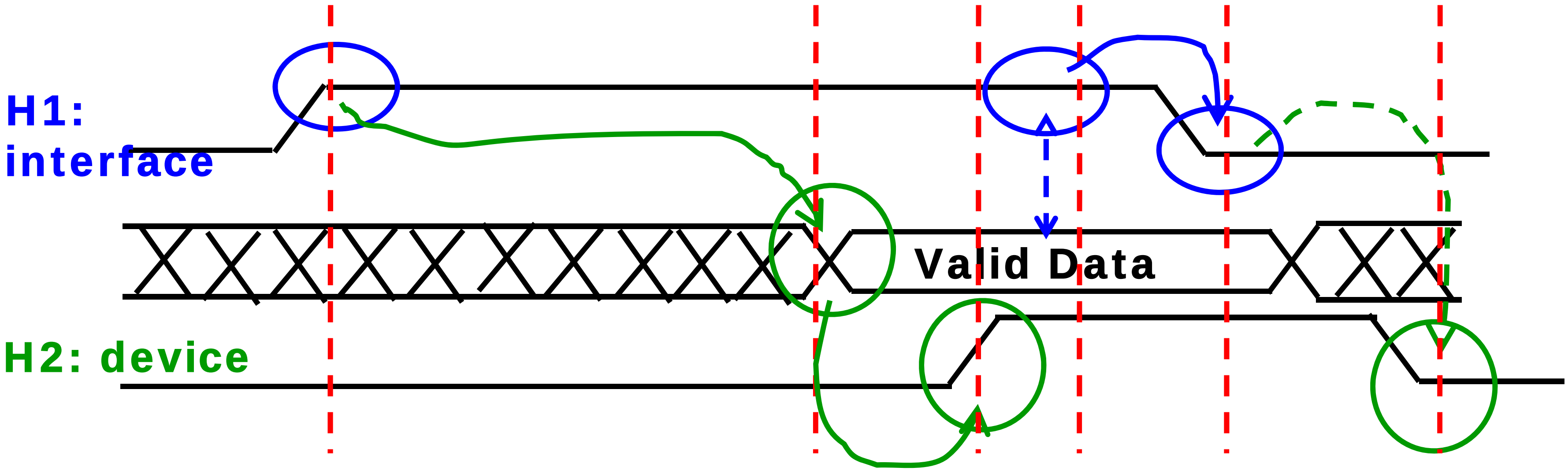
3. **Interface** latches data and de-asserts H1

4. After appropriate delay, input device de-asserts H2



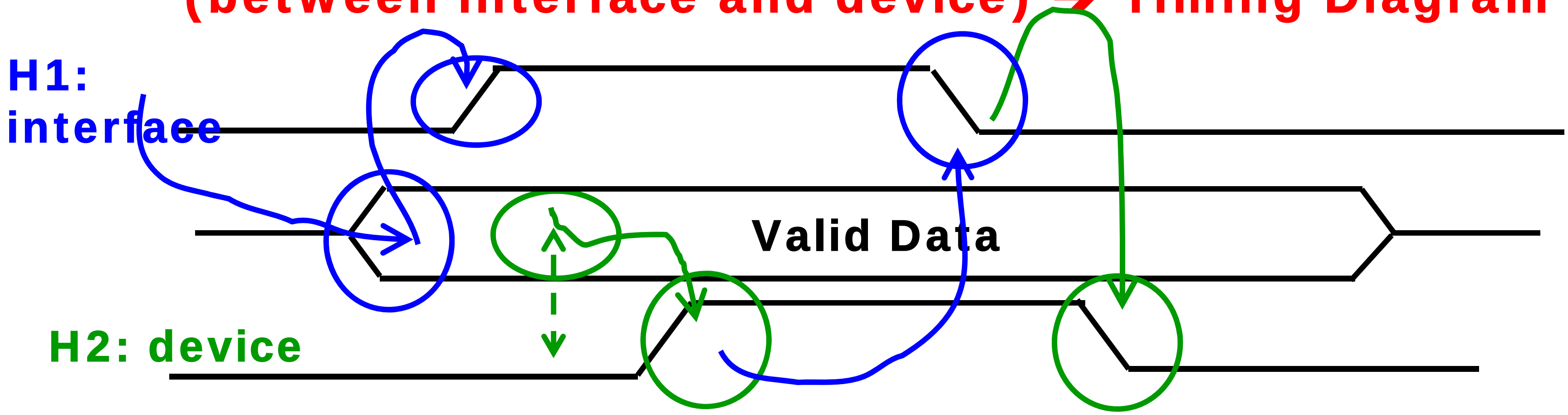
Example of *Input* Handshaking Protocol

(between interface and device) ➔ Timing Diagram



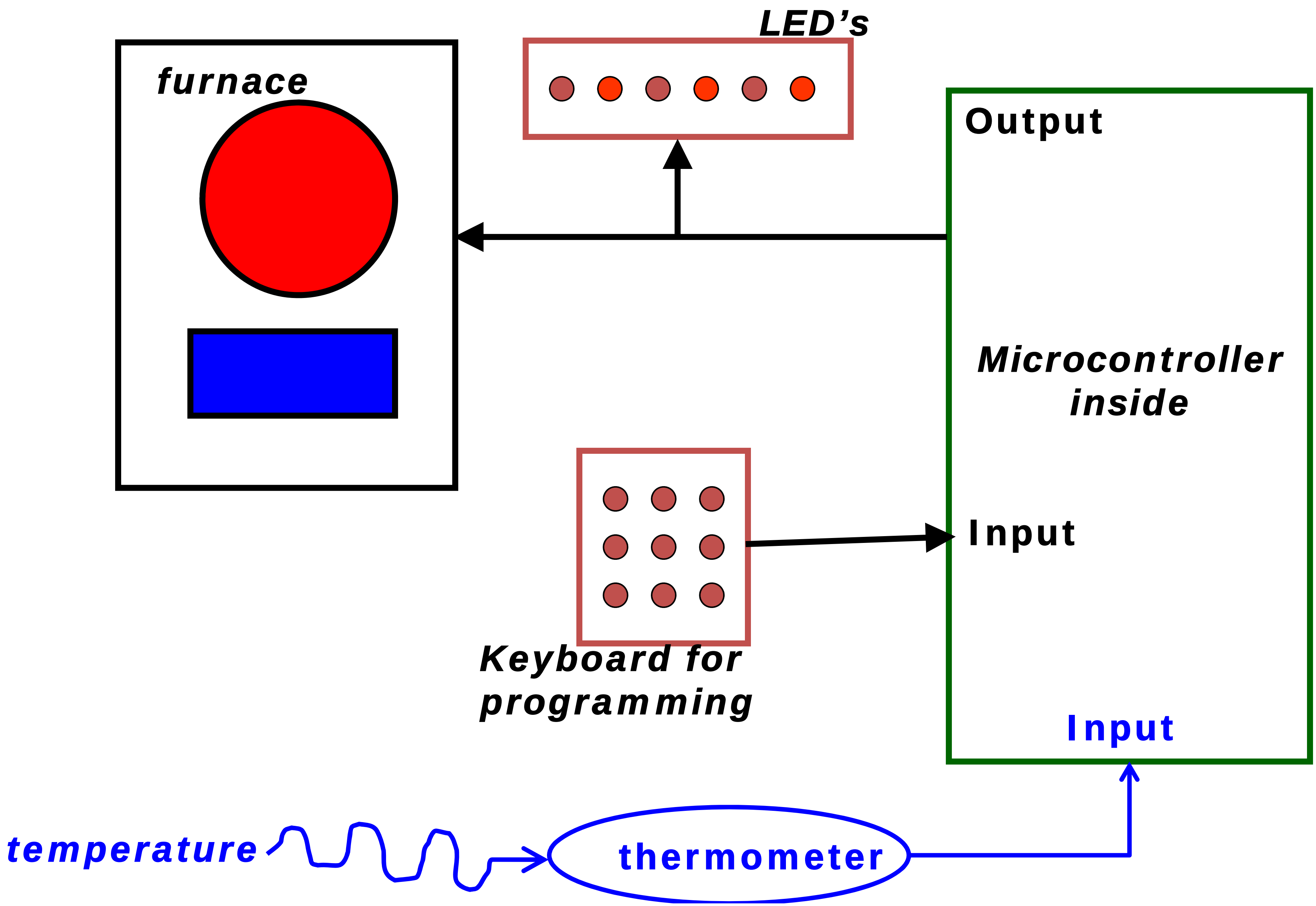
1. interface asserts a signal H1 ➔ states intention to get a byte
2. input device puts valid data on data port
3. input device asserts a signal H2
4. interface latches data
5. interface de-asserts H1
6. after appropriate delay, input device de-asserts H2

Example of *Output* Handshaking Protocol
(between interface and device) ➔ Timing Diagram

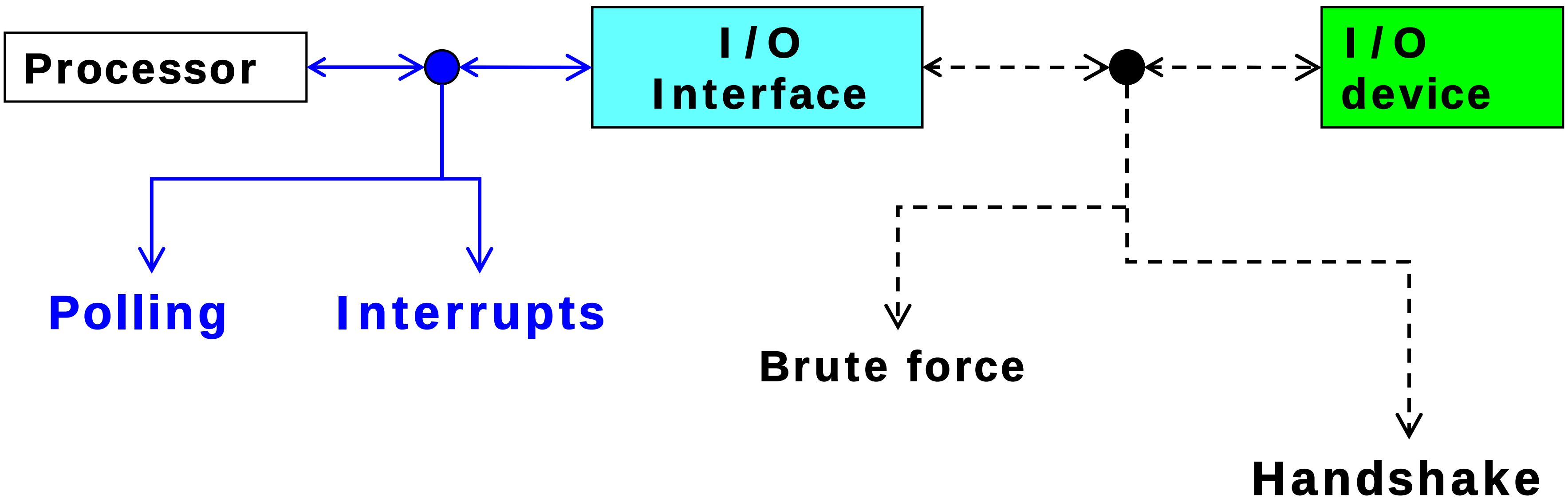


1. interface puts data on data port
2. interface asserts H1 ➔ states intention to output a byte
3. output device latches data
4. output device asserts H2 to acknowledge
5. interface de-asserts H1 ➔ done
6. output device de-asserts H2 ➔ done

Small Example of Embedded System: a programmable thermostat controller



Small Summary



I/O Software requirements: general summary

1. **Initialization**
   - set functions of ports
   - set direction of data flow
2. **Data input and output**
   - read or write from appropriate register in control register stack
3. **Software synchronization**
   - meet timing requirements of I/O devices

Software Synchronization Methods

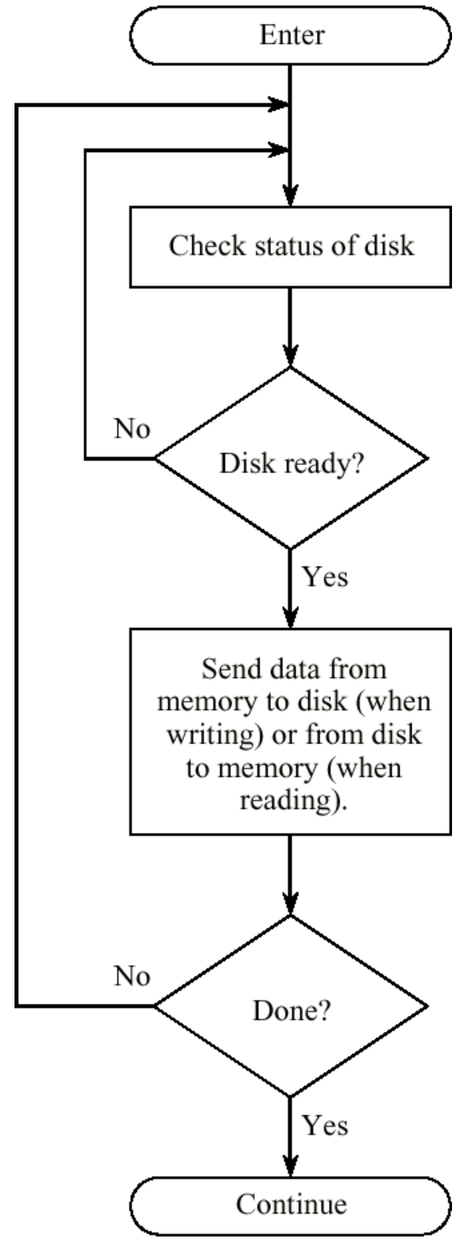
(a) Real Time

- use software delay to match SW & HW timing
  - e.g. output character to PORT B no faster than 10 characters per second (requires ~100ms between each output operation – must code a software delay)
- problems:
  - ✓ depends on clock frequency
  - ✓ overhead cycles ➔ errors ➔ timing is not exact (even using internal timer system)

(b) Polled I/O

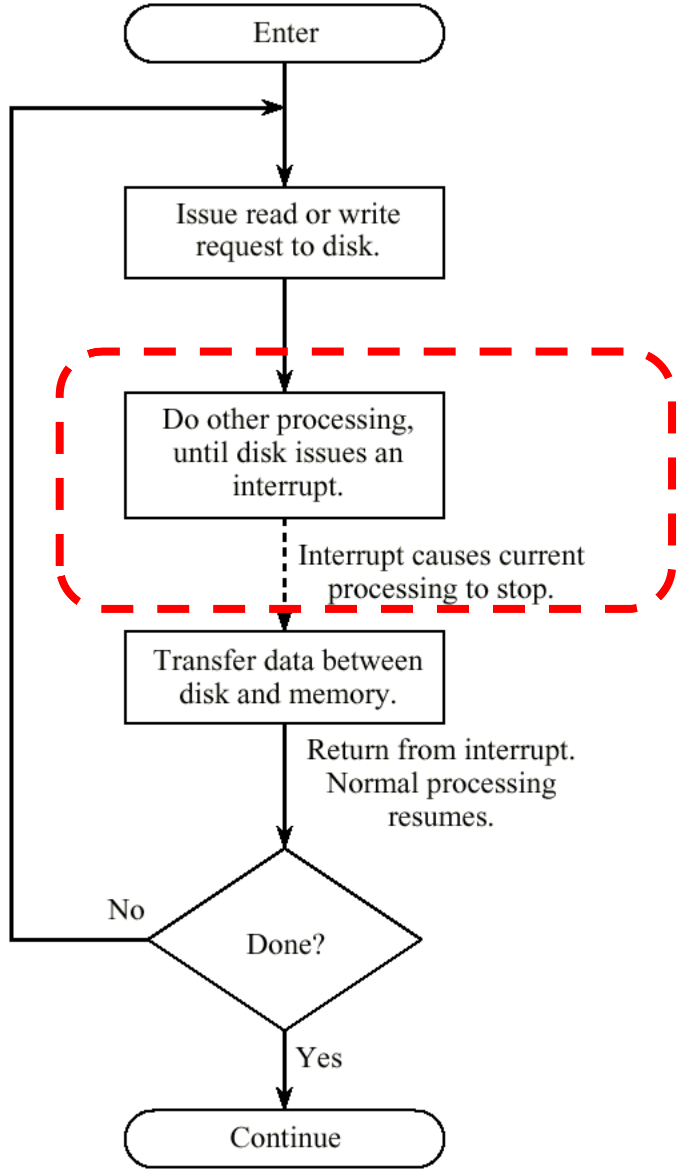
- strobe flag is a status bit to indicate data is available for input
- Hardware device must know which bit to assert

Programmed I/O (polling) Flowchart for a Disk Transfer



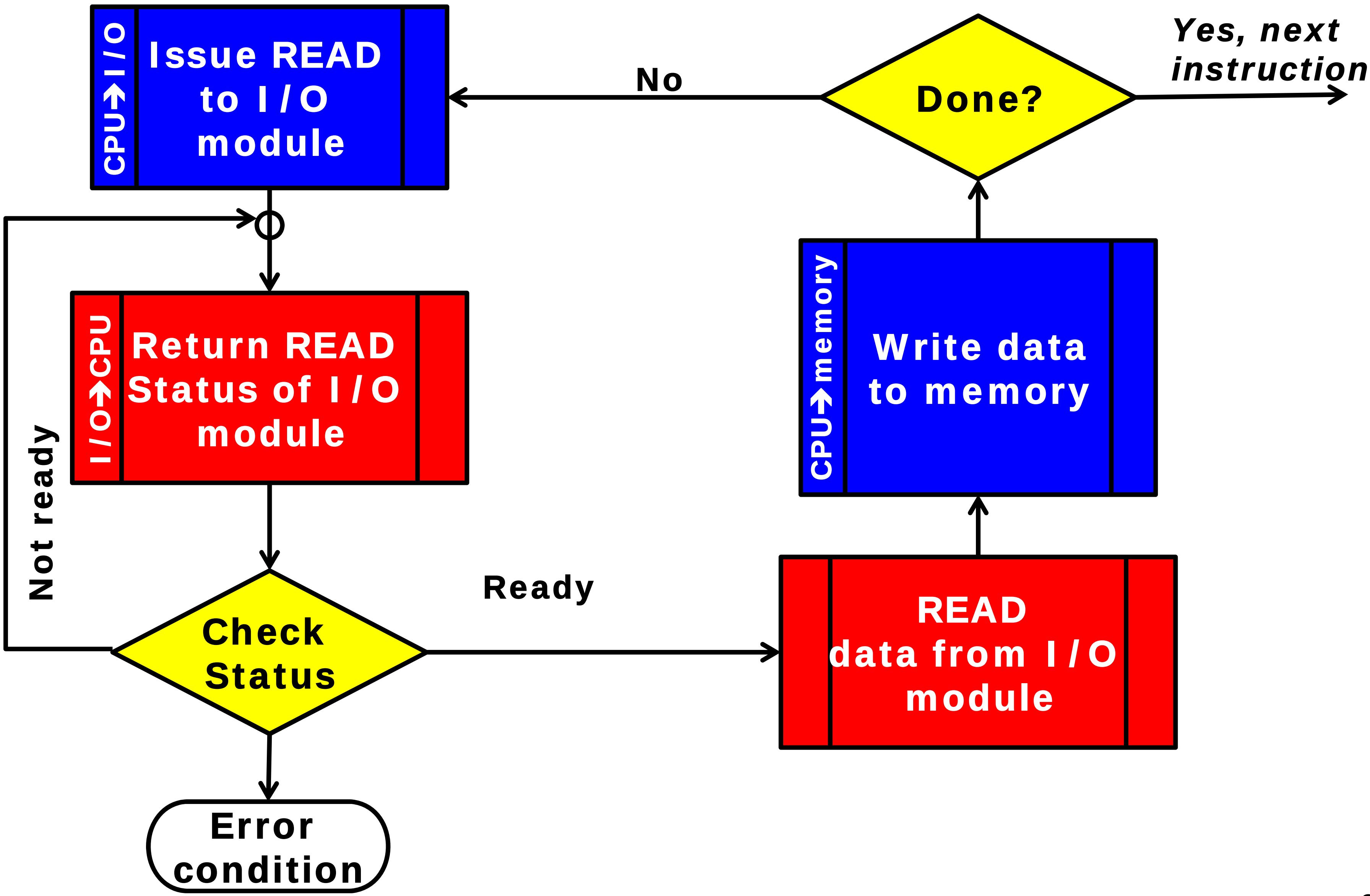
© 2007 M. Murdocca and V. Heuring

Interrupt Driven I/O Flowchart for a Disk Transfer



© 2007 M. Murdocca and V. Heuring

Programmed I/O (polling) for a Transfer



Interrupt-Driven I/O for a Transfer