Basic I/O Interface
Introduction

• This chapter outlines some of the basic methods of communications, both serial and parallel, between humans or machines and the microprocessor.

• We first introduce the basic I/O interface and discuss decoding for I/O devices.

• Then, we provide detail on parallel and serial interfacing, both of which have a variety of applications.
INTRO TO I/O INTERFACE

• I/O instructions (IN, INS, OUT, and OUTS) are explained.
• Also isolated (direct or I/O mapped I/O) and memory-mapped I/O, the basic input and output interfaces, and handshaking.
• Knowledge of these topics makes it easier to understand the connection and operation of the programmable interface components and I/O techniques.
The I/O Instructions

• One type of instruction transfers information to an I/O device (OUT).
• Another reads from an I/O device (IN).
• Instructions are also provided to transfer strings of data between memory and I/O.
  – INS and OUTS, found except the 8086/8088
• Instructions that transfer data between an I/O device and the microprocessor’s accumulator (AL, AX, or EAX) are called IN and OUT.
• The I/O address is stored in register DX as a 16-bit address or in the byte (p8) immediately following the opcode as an 8-bit address.
  – Intel calls the 8-bit form (p8) a fixed address because it is stored with the instruction, usually in a ROM
• The 16-bit address is called a variable address because it is stored in a DX, and then used to address the I/O device.
Other instructions that use DX to address I/O are the **INS** and **OUTS** instructions.

- **I/O ports are 8 bits in width.**
  - a **16-bit port** is actually **two** consecutive 8-bit **ports** being addressed
  - a **32-bit I/O port** is actually **four** 8-bit **ports**
• When data are transferred using **IN** or **OUT**, the I/O address, (**port number** or simply port), appears on the address bus.

• External I/O interface decodes the **port number** in the same manner as a memory address.
  – the 8-bit fixed **port number** (p8) appears on address bus connections A_7–A_0 with bits A_{15}–A_8 equal to 00000000_2
  – connections above A_{15} are **undefined** for I/O instruction
• The 16-bit variable port number (DX) appears on address connections $A_{15}–A_0$.
• The first 256 I/O port addresses (00H–FFH) are accessed by both fixed and variable I/O instructions.
  – any I/O address from 0100H to FFFFH is only accessed by the variable I/O address
• In a PC computer, all 16 address bus bits are decoded with locations 0000H–03FFH.
  – used for I/O inside the PC on the ISA (industry standard architecture) bus
• **INS** and **OUTS** instructions address an I/O device using the DX register.
  – but do not transfer data between accumulator and I/O device as do the IN/OUT instructions
  – Instead, they **transfer data** between memory and the I/O device
• Pentium 4 and Core2 operating in the 64-bit **mode** have the same I/O instructions.
• There are **no 64-bit I/O instructions** in the 64-bit mode.
  – most I/O is **still 8 bits** and likely will remain so
Isolated and Memory-Mapped I/O

• Two different methods of interfacing I/O: isolated I/O and memory-mapped I/O.
• In isolated I/O, the IN, INS, OUT, and OUTS transfer data between the microprocessor’s accumulator or memory and the I/O device.
• In memory-mapped I/O, any instruction that references memory can accomplish the transfer.
• The PC does not use memory-mapped I/O.
**Isolated I/O**

- The most common I/O transfer technique used in the Intel-based system is isolated I/O. 
  - *isolated* describes how I/O locations are isolated from memory in a separate I/O address space.
- Addresses for *isolated I/O devices*, called *ports*, are separate from memory.
- Because the ports are separate, the user can expand the memory to its full size without using any of *memory space* for I/O devices.
• A disadvantage of isolated I/O is that data transferred between I/O and microprocessor must be accessed by the **IN**, **INS**, **OUT**, and **OUTS** instructions.

• **Separate control signals** for the I/O space are developed (using **M/I/O** and **W/R** ), which indicate an I/O read (IORC) or an I/O write (RD) operation.

• These signals indicate an **I/O port address**, which appears on the address bus, is used to **select** the I/O device.
## Table 1 Input/Output Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Data Width</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN AL, p8</td>
<td>8</td>
<td>A byte is input into AL from port p8</td>
</tr>
<tr>
<td>IN AX, p8</td>
<td>16</td>
<td>A word is input into AX from port p8</td>
</tr>
<tr>
<td>IN EAX, p8</td>
<td>32</td>
<td>A doubleword is input into EAX from port p8</td>
</tr>
<tr>
<td>IN AL, DX</td>
<td>8</td>
<td>A byte is input into AL from the port addressed by DX</td>
</tr>
<tr>
<td>IN AX, DX</td>
<td>16</td>
<td>A word is input into AX from the port addressed by DX</td>
</tr>
<tr>
<td>IN EAX, DX</td>
<td>32</td>
<td>A doubleword is input into EAX from the port addressed by DX</td>
</tr>
<tr>
<td>INSB</td>
<td>8</td>
<td>A byte is input from the port addressed by DI and stored into the extra segment memory location addressed by DI, then DI = DI ± 1</td>
</tr>
<tr>
<td>INSW</td>
<td>16</td>
<td>A word is input from the port addressed by DI and stored into the extra segment memory location addressed by DI, then DI = DI ± 2</td>
</tr>
<tr>
<td>INSD</td>
<td>32</td>
<td>A doubleword is input from the port addressed by DI and stored into the extra segment memory location addressed by DI, then DI = DI ± 4</td>
</tr>
<tr>
<td>OUT p8, AL</td>
<td>8</td>
<td>A byte is output from AL into port p8</td>
</tr>
<tr>
<td>OUT p8, AX</td>
<td>16</td>
<td>A word is output from AL into port p8</td>
</tr>
<tr>
<td>OUT p8, EAX</td>
<td>32</td>
<td>A doubleword is output from EAX into port p8</td>
</tr>
<tr>
<td>OUT DX, AL</td>
<td>8</td>
<td>A byte is output from AL into the port addressed by DX</td>
</tr>
<tr>
<td>OUT DX, AX</td>
<td>16</td>
<td>A word is output from AX into the port addressed by DX</td>
</tr>
<tr>
<td>OUT DX, EAX</td>
<td>32</td>
<td>A doubleword is output from EAX into the port addressed by DX</td>
</tr>
<tr>
<td>OUTSB</td>
<td>8</td>
<td>A byte is output from the data segment memory location addressed by SI into the port addressed by DX, then SI = SI ± 1</td>
</tr>
<tr>
<td>OUTSW</td>
<td>16</td>
<td>A word is output from the data segment memory location addressed by SI into the port addressed by DX, then SI = SI ± 2</td>
</tr>
<tr>
<td>OUTSD</td>
<td>32</td>
<td>A doubleword is output from the data segment memory location addressed by SI into the port addressed by DX, then SI = SI ± 4</td>
</tr>
</tbody>
</table>
Figure 1 The memory and I/O maps for the 8086/8088 microprocessors. (a) Isolated I/O. (b) Memory-mapped I/O.

- in the PC, isolated I/O ports are used to control peripheral devices
- an 8-bit port address is used to access devices located on the system board, such as the timer and keyboard interface
- a 16-bit port is used to access serial and parallel ports, video and disk drive systems
Memory-Mapped I/O

- Memory-mapped I/O does not use the IN, INS, OUT, or OUTS instructions.
- It uses any instruction that transfers data between the microprocessor and memory.
  – treated as a memory location in memory map
- Advantage is any memory transfer instruction can access the I/O device.
- Disadvantage is a portion of memory system is used as the I/O map.
  – reduces memory available to applications
the PC uses part of I/O map for dedicated functions, as shown here

I/O space between ports 0000H and 03FFH is normally reserved for the system and ISA bus

ports at 0400H–FFFFFH are generally available for user applications, main-board functions, and the PCI bus

80287 coprocessor uses 00F8H–00FFH, so Intel reserves I/O ports 00F0H–00FFH
Basic Input and Output Interfaces

- The basic **input device** is a set of three-state buffers.
- The basic **output device** is a set of data latches.
- The term **IN** refers to moving data *from the I/O device into* the microprocessor and
- The term **OUT** refers to moving data *out of the microprocessor to* the I/O device.
The Basic Input Interface

- **Three-state buffers** are used to construct the 8-bit input port depicted in Figure 3.
- External TTL data are connected to the inputs of the buffers.
  - buffer outputs connect to the data bus
- The circuit of allows the processor to read the contents of the eight switches that connect to any **8-bit** section of the data bus when the select signal becomes a logic 0.
Figure 3  The basic input interface illustrating the connection of eight switches. Note that the 74ALS244 is a three-state buffer that controls the application of the switch data to the data bus.
• When the IN instruction executes, contents of the switches copy to the AL register.
• This basic input circuit is not optional and must appear any time input data are interfaced to the microprocessor.
• Sometimes it appears as a discrete part of the circuit, as shown in Figure 3.
  – also built into a programmable I/O devices
• Sixteen- or 32-bit data can also be interfaced but is not nearly as common as 8-bit data.
The Basic Output Interface

• Receives data from the processor and usually must hold it for some external device.
  – latches or flip-flops, like buffers in the input device, are often built into the I/O device
• Fig 4 shows how eight light-emitting diodes (LEDs) connect to the processor through a set of eight data latches.
• The latch stores the number output by the microprocessor from the data bus so that the LEDs can be lit with any 8-bit binary number.
Figure 4  The basic output interface connected to a set of LED displays.
• Latches **hold** the data because when the processor executes an OUT, data are only present on the data bus for less than 1.0 \( \mu s \).
  – the viewer would never see the LEDs illuminate
• When the OUT executes, data from AL, AX, or EAX transfer to the latch via the data bus.
• Each time the OUT executes, the SEL signal activates, **capturing data** to the latch.
  – data are held until the next OUT
• When the output instruction is executed, data from the **AL register** appear on the LEDs.
Handshaking

- Many I/O devices accept or release information *slower* than the microprocessor.
- A method of I/O control called *handshaking* or *polling*, synchronizes the I/O device with the microprocessor.
- An example is a parallel printer that prints a few hundred characters per second (CPS).
- The processor can send data much *faster*. – a way to slow the microprocessor down to match speeds with the printer must be developed.
• Fig 5 illustrates typical input and output connections found on a printer.
  – data transfers via data connections (D₇–D₀)
• ASCII data are placed on D₇–D₀, and a pulse is then applied to the STB connection.
  – BUSY indicates the printer is busy
  – STB is a clock pulse used to send data to printer
• The **strobe signal** sends or clocks the data into the printer so that they can be printed.
  – as the printer receives data, it places logic 1 on the BUSY pin, indicating it is printing data
FIGURE 5  The DB25 connector found on computers and the Centronics 36-pin connector found on printers for the Centronics parallel printer interface.
• The software **polls** or tests the **BUSY** pin to decide whether the printer is busy.
  – If the **printer** is **busy**, the **processor** waits
  – if not, the next **ASCII** character goes to the printer
• This process of interrogating the printer, or any asynchronous device like a printer, is called **handshaking** or **polling**.
Input Devices

• Input devices are already TTL and compatible, and can be connected to the microprocessor and its interfacing components.
  – or they are switch-based

• Switch-based devices are either open or connected; These are not TTL levels.
  – TTL levels are a logic 0 (0.0 V–0.8 V)
  – or a logic 1 (2.0 V–5.0 V)

• Using switch-based device as TTL-compatible input requires conditioning applied.
• Fig 6 shows a **toggle switch** properly connected to function as an input device.

• A **pull-up resistor** ensures when the switch is open, the output signal is a logic 1.
  – when the switch is closed, it connects to ground, producing a valid logic 0 level

• A **standard range** of values for **pull-up resistors** is between 1K Ohm and 10K Ohm.
Figure 6  A single-pole, single-throw switch interfaced as a TTL device.
• Mechanical switch contacts physically **bounce** when they are closed,
  – which can create a problem if a switch is used as a clocking signal for a digital circuit
• To prevent problems with bounces, one of the circuits shown in Fig 7 can be used.
  – the **first** is a classic textbook **bounce eliminator**
  – **second** is a more practical version of the same
• The first version costs more to construct
  – the second costs requires no pull-up resistors and two inverters instead of two NAND gates
Figure 7 Debouncing switch contacts: (a) conventional debouncing and (b) practical debouncing.

- as the Q input from the switch becomes a logic 0, it changes the state of the flip-flop
- if the contact bounces away from the Q input, the flip-flop remembers, no change occurs, and thus no bounce
Output Devices

- Output devices are more diverse than input devices, but many are interfaced in a uniform manner.
- Before an output device can be interfaced, we must understand voltages and currents from the microprocessor or TTL interface.
- Voltages are TTL-compatible from the microprocessor of the interfacing element.
  - logic 0 = 0.0 V to 0.4 V
  - logic 1 = 2.4 V to 5.0 V
• Currents for a processor and many interfacing components are less than for standard TTL.
  – Logic 0 = 0.0 to 2.0 mA
  – logic 1 = 0.0 to 400 µA

• Fig 11–8 shows how to interface a simple LED to a microprocessor peripheral pin.
  – a transistor driver is used in 11–8(a)
  – a TTL inverter is used in 11–8(b)

• The TTL inverter (standard version) provides up to 16 mA of current at a logic 0 level
  – more than enough to drive a standard LED
Figure 8  Interfacing an LED: (a) using a transistor and (b) using an inverter.
• TTL input signal has minimum value of 2.4 V
• Drop across emitter-base junction is 0.7 V.
• The difference is 1.7 V
  – the voltage drop across the resistor
• The value of the resistor is 1.7 V ÷ 0.1 mA or 17K Ω.
  – as 17K Ω is not a standard value, an 18K Ω resistor is chosen
• In 8(a), we elected to use a switching transistor in place of the TTL buffer.
  – 2N2222 is a good low-cost, general-purpose switching transistor with a minimum gain of 100
  – collector current is 10 mA; so base current will be 1/100 of collector current of 0.1 mA
• To determine the value of the base current–limiting resistor, use the 0.1 mA base current and a voltage drop of 1.7 V across the base current–limiting resistor.
• Suppose we need to interface a 12 V DC 1A motor to the microprocessor.
• We cannot use a TTL inverter:
  – 12 V signal would burn out the inverter
  – current far exceeds 16 mA inverter maximum
• We cannot use a 2N2222 transistor:
  – maximum current is 250 mA to 500 mA, depending on the package style chosen
• The solution is to use a **Darlington-pair**, such as a TIP120.
  – costs 25¢, can handle 4A current with heat sink
• Fig 9 illustrates a motor connected to the **Darlington-pair** with a minimum current gain of 7000 and a **maximum current of 4A**.

• Value of the bias resistor is calculated exactly the same as the one used in the LED driver.

• The current through the resistor is $1.0 \ A \div 7000$, or about 0.143 mA.

• Voltage drop is 0.9 V because of the two diode drops (base/emitter junctions).

• The value of the bias resistor is $0.9 \ V \div 0.143 \ mA$ or **6.29K Ω**.
Figure 9  A DC motor interfaced to a system by using a Darlington-pair.

– The Darlington-pair must use a heat sink because of the amount of current
– the diode must be present to prevent the Darlington-pair from being destroyed by inductive kickback
I/O PORT ADDRESS DECODING

• Very similar to memory address decoding, especially for memory-mapped I/O devices.
• The difference between memory decoding and isolated I/O decoding is the number of address pins connected to the decoder.
• In the personal computer system, we always decode all 16 bits of the I/O port address.
Decoding 8-Bit I/O Port Addresses

• Fixed I/O instruction uses an 8-bit I/O port address that on $A_{15} - A_0$ as $0000H–00FFH$.
  – we often decode only address connections $A_7–A_0$ for an 8-bit I/O port address

• The DX register can also address I/O ports $00H–FFH$.

• If the address is decoded as an 8-bit address, we can never include I/O devices using a 16-bit address.
  – the PC never uses or decodes an 8-bit address
• Figure 10 shows a 74ALS138 decoder that decodes 8-bit I/O ports F0H - F7H.
  – identical to a memory address decoder except we only connect address bits \( A_7 - A_0 \) to the inputs of the decoder

• Figure 11 shows the PLD version, using a GAL22V10 (a low-cost device) for this decoder.

• The PLD is a better decoder circuit because the number of integrated circuits has been reduced to one device.
Figure 10  A port decoder that decodes 8-bit I/O ports. This decoder generates active low outputs for ports F0H–F7H.
Figure 11  A PLD that generates part selection signals
Decoding 16-Bit I/O Port Addresses

• PC systems typically use 16-bit I/O addresses.
  – 16-bit addresses rare in embedded systems
• The difference between decoding an 8-bit and a 16-bit I/O address is that eight additional address lines (A_{15}–A_8) must be decoded.
• Figure 12 illustrates a circuit that contains a PLD and a 4-input NAND gate used to decode I/O ports EFF8H–EFFFH.
• PLD generates address strobes for I/O ports
Figure 12  A PLD that **decodes** 16-bit I/O ports EFF8H through EFFFH.
**8- and 16-Bit Wide I/O Ports**

- Data transferred to an 8-bit I/O device exist in one of the I/O banks in a **16-bit processor** such as 80386SX.
- The I/O system on such a microprocessor contains **two 8-bit memory banks**.
- Fig 13 shows separate I/O banks for a 16-bit system such as 80386SX.
- Because two I/O banks exist, any 8-bit I/O write requires a **separate write**.
Figure 13  The I/O banks found in the 8086, 80186, 80286, and 80386SX.
• I/O reads don’t require separate strobes.
  – as with memory, the processor reads only the byte it expects and ignores the other byte
  – a read can cause problems when an I/O device responds incorrectly to a read operation
• Fig 14 shows a system with two different 8-bit output devices, located at 40H and 41H.
• These are 8-bit devices and appear in different I/O banks.
  – thus, separate I/O write signals are generated to clock a pair of latches that capture port data
Figure 14  An I/O port decoder that selects ports 40H and 41H for output data.

- all I/O ports use 8-bit addresses
- ports 40H & 41H can be addressed as separate 8-bit ports
- or as one 16-bit port
• Fig 15 shows a 16-bit device connected to function at 8-bit addresses 64H & 65H.
• The PLD decoder does not have a connection for address bits BLE ($A_0$) and BHE because the signals don’t apply to 16-bit-wide devices.
• The program for the PLD, illustrated in Example 5, shows how the enable signals are generated for the three-state buffers (74HCT244) used as input devices.
Figure 15  A **16-bit-wide** port decoded at I/O addresses 64H and 65H.
32-Bit-Wide I/O Ports

• May eventually become common because of newer buses found in computer systems.
• The EISA system bus supports 32-bit I/O as well as the VESA local and current PCI bus.
  – not many I/O devices are 32 bits in width
• Fig 16 shows a 32-bit input port for 80386DX - 80486DX microprocessor.
• The circuit uses a single PLD to decode the I/O ports and four 74HCT244 buffers to connect the I/O data to the data bus.
Figure 16  A 32-bit-wide port decoded at 70H through 73H for the 80486DX microprocessor.

- I/O ports decoded by this interface are the 8-bit ports 70H–73H
- When writing to access this port, it is crucial to use the address 70H for 32-bit input
- as instruction IN EAX, 70H
With the Pentium–Core2 and their 64-bit data buses, I/O ports appear in various banks, as determined by the I/O port address.

A 32-bit I/O access in the Pentium system can appear in any four consecutive I/O banks.

- 32-bit ports 0100H–0103H appear in banks 0–3

I/O address range must begin at a location where the rightmost two bits are zeros.

- 0100H–0103H is allowable
- 0101H–0104H is not
• Widest I/O transfers are **32 bits**, and there are no I/O instructions to support **64-bit transfers**.
  – true for Pentium 4 or Core2 in the 64-bit mode
• Suppose we need to interface a 16-bit-wide output port at I/O address 2000H and 2001H.
  – interface is illustrated in Figure 17
  – PLD program is listed in Example 7
• The problem that can arise is when the I/O port spans across a **64-bit boundary**.
  – example, a 16-bit- port at 2007H and 2008H
Figure 17 A Pentium 4 interfaced to a 16-bit-wide I/O port at port addresses 2000H and 2001H.
THE PROGRAMMABLE PERIPHERAL

• 82C55 programmable peripheral interface (PPI) is a popular, low-cost interface component found in many applications.
• The PPI has 24 pins for I/O, programmable in groups of 12 pins and groups that operate in three distinct modes of operation.
• 82C55 can interface any TTL-compatible I/O device to the microprocessor.
• The 82C55 (CMOS version) requires **wait states** if operated with a processor using higher than an 8 MHz clock.
  – also provides at least **2.5 mA of sink** (logic 0) current at each output, a **maximum of 4.0 mA**
• Because I/O devices are **inherently slow**, **wait states** used during I/O transfers do not impact significantly upon the **speed** of the system.
• The 82C55 still finds application even in the latest Core2-based computer system.
• **82C55** is used for interface to the keyboard and parallel printer port in many PCs.
  – found as a function within an interfacing chip set
  – also controls the timer and reads data from the keyboard interface

• An experimentation board is available that plugs into the parallel port of a PC, to allow access to an 8255 located on the board.

• The 8255 is programmed in either assembly language or Visual C++ through drivers available with the board.
Basic Description of the 82C55

• Fig 18 shows pin-outs of the 82C55 in DIP and surface mount (flat pack) format.
• The three I/O ports (labeled A, B, and C) are programmed as groups.
  – group A connections consist of port A (PA\textsubscript{7}–PA\textsubscript{0}) and the upper half of port C (PC\textsubscript{7}–PC\textsubscript{4})
  – group B consists of port B (PB\textsubscript{7}–PB\textsubscript{0}) and the lower half of port C (PC\textsubscript{3}–PC\textsubscript{0})
• 82C55 is selected by its CS pin for programming and reading/writing to a port.
Figure 18  The pin-out of the 82C55 peripheral interface adapter (PPI).
• Table 2 shows I/O port assignments used for programming and access to the I/O ports.
• In the PC, a pair of 82C55s, or equivalents, are decoded at I/O ports 60H–63H and also at ports 378H–37BH.
• The 82C55 is a fairly simple device to interface to the microprocessor and program.
• For 82C55 to be read or written, the CS input must be logic 0 and the correct I/O address must be applied to the A₁ and A₀ pins.
• Remaining port address pins are don’t cares.
• Fig 19 shows an 82C55 connected to the 80386SX so it functions at 8-bit addresses C0H (port A), C2H (port B), C4H (port C), and C6H (command register).
  – this interface uses the low bank of the I/O map
• All 82C55 pins are direct connections to the 80386SX, except the CS pin. The pin is decoded/selected by a 74ALS138 decoder.
• A RESET to 82C55 sets up all ports as simple input ports using mode 0 operation.
  – initializes the device when the processor is reset
Figure 19  The 82C55 interfaced to the low bank of the 80386SX microprocessor.
• After a RESET, no other commands are needed, as long as it is used as an input device for all three ports.

• 82C55 is interfaced to the PC at port addresses 60H–63H for keyboard control.
  – also for controlling the speaker, timer, and other internal devices such as memory expansion

• It is also used for the parallel printer port at I/O ports 378H–37BH.
Programming the 82C55

- 82C55 is programmed through two internal command registers shown in Figure 20.
- Bit position 7 selects either command byte A or command byte B.
  - command byte A programs functions of group A and B
  - byte B sets (1) or resets (0) bits of port C only if the 82C55 is programmed in mode 1 or 2
- Group B (port B and the lower part of port C) are programmed as input or output pins.
Figure 20 The command byte of the command register in the 82C55. (a) Programs ports A, B, and C. (b) Sets or resets the bit indicated in the select a bit field.

- group B operates in mode 0 or mode 1
- Mode 0 is basic input/output mode that allows the pins of group B to be programmed as simple input and latched output connections
- Mode 1 operation is the strobed operation for group B connections
- data are transferred through port B
- handshaking signals are provided by port C
• **Group A** (port A and the upper part of port C) are programmed as input or output pins.

• **Group A** can operate in modes 0, 1, and 2.
  – mode 2 operation is a bidirectional mode of operation for port A

• If a 0 is placed in **bit position 7** of the command byte, command byte B is selected

• This allows any bit of **port C** to be set (1) or reset (0), if the 82C55 is operated in either mode 1 or 2.
  – otherwise, this byte is not used for **programming**
Mode 0 Operation

- Mode 0 operation causes 82C55 to function:
  - as a buffered input device
  - as a latched output device
- Fig 21 shows 82C55 connected to a set of eight seven-segment LED displays.
- These are standard LEDs.
  - the interface can be modified with a change in resistor values for an organic LED (OLED) display or high-brightness LEDs
Figure 21 An 8-digit LED display interfaced to the 8088 microprocessor through an 82C55 PIA.

- ports A & B are programmed as (mode 0) simple latched output ports
- port A provides segment data inputs
- port B provides a means of selecting one display position at a time for multiplexing the displays
- the 82C55 is interfaced to an 8088 through a PLD so it functions at I/O port numbers 0700H–0703H
- PLD decodes the I/O address and develops the write strobe for the WR pin of the 82C55
• Resistor values in Fig 21 are chosen so the segment current is 80 mA.
  – required to produce average 10 mA current per segment as the displays are multiplexed
• A six-digit display uses a segment current of 60 mA for an average of 10 mA per segment.
• Peak anode current in an eight-digit display is 560 mA (seven segments × 80 mA).
  – average anode current is 80 mA
• In a six-digit display, peak current would be 420 mA (seven segments × 60 mA).
• In this display, the segment load resistor passes 80 mA current and has approximately 3.0 V across it.
• The value of the resistor is $3.0 \, \text{V} \div 180 \, \text{mA} = 37.5 \, \text{Ohm}$. The closest standard resistor value of 39 Ohm is used in Fig 21.
• Programming the 82C55 is accomplished by the short sequence of instructions listed in Example 9.
• Ports A and B are programmed as outputs.
An LCD Display Interfaced to the 82C55

- LCDs (liquid crystal displays) have replaced LED displays in many applications.
- Fig 22 shows an Optrex DMC-20481 LCD display interfaced to an 82C55.
  - DMC-20481 is a 4-line by 20-characters-per-line display that accepts ASCII code as input data
- It also accepts commands that initialize it and control its application.
Figure 22 The DMC-20481 LCD display interfaced to the 82C55.

*Current max is 480 mA, nominal 260 mA.*
• The data connections, which are attached to the **82C55 port A**, are used to input display data and to read information from the display.
• For a 4-bit interface, D_4–D_7 pins are used where the data must be formatted with the high nibble first, followed by the low nibble.
• A few newer **OLED devices** contain a serial interface that uses a single pin for the data.
• To program DMC-20481 it must first be initialized.
• This applies to any display using the HD44780 (Hitachi) display driver IC.
• The entire line of small display panels from Optrex and most other manufacturers is programmed in the same manner.
• To program DMC-20481 it must first be initialized.
  – this applies to any display using the HD44780 (Hitachi) display driver integrated circuit
Initialization is accomplished via the following:

1. Wait at least 15 ms after $V_{CC}$ rises to 5.0 V
2. Output the function set command (30H), and wait at least 4.1 ms
3. Output the function set command (30H) a second time, and wait at least 100 µs
4. Output the function set command (30H) a third time, and wait at least 40 µs
5. Output the function set command (38H) a fourth time, and wait at least 40 µs
6. Output 08H to disable the display, and wait at least 40 µs
– 7. Output a 01H to home the cursor and clear the display, and wait at least 1.64 ms
– 8. Output the enable display cursor off (0CH), and wait at least 40 µs
– 9. Output 06H to select auto-increment, shift the cursor, and wait at least 40 µs

• Software to accomplish the initialization of the LCD display is listed in Example 12.
• The time delays can also be obtained by using a timer in C++.
After initialization, **time delays** are no longer needed when sending data or many commands to the display.

The clear display command still needs a time delay as the **busy flag** is not used.

Instead of a **time delay**, the busy flag is tested to see whether the display has completed an operation.

The BUSY procedure tests the **LCD display** and only returns when the display has completed a prior instruction.
• Once the **BUSY** procedure is available, data can be sent to the display by writing another procedure called WRITE.

• The WRITE procedure uses **BUSY** to test before trying to write new data to the display.

• Example 14 shows the **WRITE** procedure, which transfers the **ASCII** character from the BL register to the current cursor position of the display.
• The only other procedure needed for a basic display is the **clear & home cursor procedure**, called **CLS**, shown in Example 15.

• This procedure uses the SEND macro from the initialization software to send the clear command to the display.

• With CLS and the procedures presented thus far, you can **display** any message on the display, **clear** it, **display** another message, and basically operate the display.
A Stepper Motor Interfaced to the 82C55.

- Another device often interfaced to a computer system is the **stepper motor**.
  - A digital motor because it is moved in discrete steps as it traverses through 360°.
- An inexpensive stepper motor is geared to move perhaps 15° per step.
- A more costly, high-precision stepper motor can be geared to 1° per step.
• In all cases, these steps are gained through many magnetic poles and/or gearing.

• Figure 23 shows a four-coil stepper motor that uses an armature with a single pole.
  – two coils are energized

• If less power is required, one coil may be energized at a time, causing the motor to step at 45°, 135°, 225°, and 315°.

• The motor is shown with the armature rotated to four discrete places, called full stepping.
  – accomplished by energizing the coils, as shown
Figure 23  The stepper motor showing full-step operation: (a) $45^\circ$  (b) $135^\circ$  (c) $225^\circ$  (d) $315^\circ$.
• The motor is driven by NPN Darlington amp pairs to provide a large current to each coil.
• A circuit that can drive this stepper motor is illustrated in Fig 24.
  – with the four coils shown in place
• This circuit uses the 82C55 to provide drive signals used to rotate the motor armature in either the right- or left-hand direction.
• A simple procedure that drives the motor is listed in Example 16 in both assembly language and as a function in C++.
Figure 24  A stepper motor interfaced to the 82C55. This illustration does not show the decoder.
Key Matrix Interface

• Keyboards come in a variety of sizes, from standard 101-key QWERTY keyboards to special keyboards that contain 4 to 16 keys.

• Fig 25 is a key matrix with 16 switches interfaced to ports A and B of an 82C55.
  – the switches are formed into a $4 \times 4$ matrix, but any matrix could be used, such as a $2 \times 8$

• The keys are organized into four rows and columns: $(\text{ROW}_0 - \text{ROW}_3) \ (\text{COL}_0 - \text{COL}_3)$
Figure 25 A $4 \times 4$ keyboard matrix connected to an 8088 microprocessor through the 82C55 PIA.

- the 82C55 is decoded at I/O ports 50H–53H for an 8088
- port A is programmed as an input port to read the rows
- port B is programmed as an output port to select a column
- a flowchart of the software required to read a key from the keyboard matrix and debounce the key is illustrated in Fig 26
keys must be debounced, normally with a time delay of 10–20 ms

the software uses a procedure called SCAN to scan the keys and another called DELAY10 to waste 10 ms of time for debouncing

the main keyboard procedure is called KEY and appears in Example 17

the KEY procedure is generic, and can handle any configuration from a $1 \times 1$ matrix to an $8 \times 8$ matrix.
The **ShortDelay procedure** is needed as the computer changes port B at a very high rate.
- the time delay allows the data sent to port B to settle to their final state

This is not needed if **scan rate** (time between output instructions) does not exceed 30 KHz.
- if the scanning frequency is higher, the device generates radio interference

If so, the FCC will not approve application in any **accepted** system
- without certification the system cannot be sold
Mode 1 Strobed Input

• Causes port A and/or port B to function as latching input devices.
  – allows external data to be stored to the port until the microprocessor is ready to retrieve it

• Port C is used in mode 1 operation—not for data, but for control or handshaking signals.
  – to help operate either or both port A and B as strobed input ports

• Fig 27 shows how both ports are structured for mode 1 strobed input operation.
Figure 27  Strobed input operation (mode 1) of the 82C55. (a) Internal structure and (b) timing diagram.
Signal Definitions for Mode 1
Strobed Input

**STB**

- The *strobe* input loads data to the port latch, which holds the information until it is input to the microprocessor via the IN instruction.

**IBF**

- *Input buffer full* is an output indicating that the input *latch* contains information.
**INTR**

- **Interrupt request** is an output that requests an interrupt. The INTR pin becomes a logic 1 when STB returns to a logic 1. Cleared when data are input from the port by the processor.

**INTE**

- **Interrupt** enable signal is neither input nor output; it is an internal bit programmed via port PC₄ (port A) or PC₂ (port B) bit position.
PC_7, PC_6

• The port C pins 7 and 6 are general-purpose I/O pins that are available for any purpose.
Strobed Input Example

• An example of a strobed input device is a keyboard.

• The keyboard encoder **debounces** the key switches and provides a strobe signal whenever a key is depressed.
  – the data output contains ASCII-coded key code

• Figure 11–28 illustrates a keyboard connected to strobed input port A.
Figure 28 Using the 82C55 for strobed input operation of a keyboard.
Mode 1 Strobed Output

- Fig 11–29 shows the internal configuration and timing diagram of 82C55 when operated as a strobed output device under mode 1.
- Strobed output operation is similar to mode 0 output operation.
  - except control signals are included to provide handshaking.
- When data are written to a strobed output port, the output buffer full signal becomes logic 0 to indicate data are present in the port latch.
**Figure 29** Strobed output operation (mode 1) of the 82C55. (a) Internal structure and (b) timing diagram.
Signal Definitions for Mode 1
Strobed Output

OBF

- **Output buffer full** goes low whenever data are output (OUT) to the port A or B latch. The signal is set to logic 1 when the ACK pulse returns from the external device.
**ACK**

- The **acknowledge signal** causes the OBF pin to return to logic 1. The ACK signal is a response from an external device, indicating that it has received data from the 82C55 port.

**INTR**

- **Interrupt request** often interrupts the processor when the external device receives the data via the **ACK** signal. Qualified by the internal INTE (**interrupt enable**) bit.
INTE

- **Interrupt enable** is neither input nor output; it is an internal bit programmed to enable or disable the INTR pin. INTE A is programmed using PC\textsubscript{6} bit. INTE B is programmed using the PC\textsubscript{2} bit.

PC\textsubscript{4}, PC\textsubscript{5}

- Port C pins PC\textsubscript{4} and PC\textsubscript{5} are general-purpose I/O pins. The bit set and reset command is used to set or reset these two pins.
Strobed Output Example

• The printer interface demonstrates how to achieve strobed output synchronization between the printer and the 82C55.

• Figure 11–30 illustrates port B connected to a parallel printer, with eight data inputs for receiving ASCII-coded data, a DS (data strobe) input to strobe data into the printer, and an ACK output to acknowledge the receipt of the ASCII character.
Figure 30  The 82C55 connected to a parallel printer interface that illustrates the strobed output mode of operation for the 82C55.
Mode 2 Bidirectional Operation

- Mode 2 is allowed with group A only.
- Port A becomes bidirectional, allowing data transmit/receive over the same eight wires.
  - useful when interfacing two computers
- Also used for IEEE-488 parallel high-speed GPIB (general-purpose instrumentation bus) interface standard.
- Figure 31 shows internal structure and timing for mode 2 bidirectional operation.
Figure 31  Mode 2 operation of the 82C55. (a) Internal structure and (b) timing diagram.
Signal Definitions for Bidirectional Mode 2

**INTR**

- **Interrupt request** is an output used to interrupt the microprocessor for input and output conditions.

**OBF**

- **Output buffer full** is an output indicating the output buffer contains data for the bidirectional bus.
ACK

• **Acknowledge** is an input that enables the three-state buffers so that data can appear on port A. If ACK is logic 1, the output buffers of port A are at their high-impedance state.

STB

• The **strobe** input loads the port A input latch with external data from the bidirectional port A bus.
PC_0, PC_1, and PC_2

- These pins are general-purpose I/O pins in mode 2 controlled by the bit set and reset command.
IBF

- **Input buffer full** is an output used to signal that the input buffer contains data for the external bidirectional bus.

INTE

- **Interrupt enable** are internal bits (INTE1 & INTE2) that enable the INTR pin. The state of the INTR pin is controlled through port C bits PC₆ (INTE1) and PC₄ (INTE2).
The Bidirectional Bus

• The bidirectional bus is used by referencing port A with the IN and OUT instructions.
• To transmit data through the bidirectional bus, the program first tests to determine whether the output buffer is empty.
  – if so, data are sent to the output buffer via OUT
• The external circuitry also monitors the signal to decide whether the microprocessor has sent data to the bus.
To receive data through the bidirectional port A bus, IBF is tested with software to decide whether data have been strobed into the port.
- if IBF = 1, data is input using IN

The external interface sends data to the port by using the STB signal.
- the IBF signal becomes logic 1 and data at port A are held inside the port in a latch

When the IN executes, the IBF bit is cleared and data in the port are moved into AL.

See Example 21 for a procedure.
• The INTR (**interrupt request**) pin can be activated from both directions of data flow through the bus.

• If INTR is enabled by both INTE bits, the output and input buffers both cause interrupt requests.

• This occurs when data are **strobed** into the buffer using **STB** or when data are written using **OUT**.
82C55 Mode Summary

- Figure 11–32 shows a graphical summary of the three modes of operation for the 82C55.
- Mode 0 provides simple I/O.
- Mode 1 provides strobed I/O.
- Mode 2 provides bidirectional I/O.
- These modes are selected through the command register of the 82C55.
**Figure 32** A summary of the port connections for the 82C55 PIA.

<table>
<thead>
<tr>
<th>Port A</th>
<th>Mode 0</th>
<th>Mode 1</th>
<th>Mode 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IN</td>
<td>IN</td>
<td>I/O</td>
</tr>
<tr>
<td></td>
<td>OUT</td>
<td>OUT</td>
<td>Not used</td>
</tr>
<tr>
<td>Port B</td>
<td>IN</td>
<td>IN</td>
<td>I/O</td>
</tr>
<tr>
<td></td>
<td>STB_B</td>
<td>OBF_B</td>
<td>I/O</td>
</tr>
<tr>
<td></td>
<td>IBF_B</td>
<td>ACK_B</td>
<td>I/O</td>
</tr>
<tr>
<td></td>
<td>INTR_B</td>
<td>INTR_B</td>
<td>INTR_A</td>
</tr>
<tr>
<td></td>
<td>STB_A</td>
<td>STB_A</td>
<td>STB_A</td>
</tr>
<tr>
<td></td>
<td>IBF_A</td>
<td>IBF_A</td>
<td>IBF_A</td>
</tr>
<tr>
<td></td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
</tr>
<tr>
<td></td>
<td>ACK_A</td>
<td>ACK_A</td>
<td>ACK_A</td>
</tr>
<tr>
<td></td>
<td>OBF_A</td>
<td>OBF_A</td>
<td>OBF_A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port C</th>
<th>Mode 0</th>
<th>Mode 1</th>
<th>Mode 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IN</td>
<td>IN</td>
<td>INTR</td>
</tr>
<tr>
<td></td>
<td>OUT</td>
<td>OUT</td>
<td>STB</td>
</tr>
<tr>
<td></td>
<td>I/O</td>
<td>I/O</td>
<td>IBF</td>
</tr>
<tr>
<td></td>
<td>I/O</td>
<td>I/O</td>
<td>ACK</td>
</tr>
<tr>
<td></td>
<td>I/O</td>
<td>I/O</td>
<td>OBF</td>
</tr>
</tbody>
</table>
The Serial EEPROM Interface

- In Chapter 10, Figure 23, a serial EEPROM is illustrated, but at that point in the text, no I/O existed for an interface.
- Suppose that port C of an 82C55 is used for connection to this interface and software is needed to drive the interface.
- The PC₀ pin is programmed as an output to send data and as an input to receive data from the EEPROM.
Chapter 10, Figure 23  A serial EEPROM interface.

- this memory interface has two signal lines
- one is a serial clock (SCL); the other a bidirectional serial data line (SDA)
- not meant to replace system main memory
- it is fast enough for music or other low-speed data
• Refer to Chapter 10 Figure 24.
• The data format for the software for reading and writing data to the EEPROM is also illustrated in Example 22.
• This software is written in C with some assembly language, but it can also be developed in assembly language.
• I/O port address for the command register is 0x1203, and 0x1202 for the port C register.
  – the time delay should be 1.25 $\mu$s for a data rate of 400 KHz
Chapter 10, Figure 10–24 Data signals to the serial EEPROM for a read or a write.

- the serial data contains the address in the first byte as well as a device code of 1010, which represents the EEPROM
- other serial devices have different device codes
- this is followed by the memory location and the data in additional bytes
THE END