

8255 PPI 7-Segment Test



```

explain: /*****
*   MDE-Win8086 EXPERIMENT PROGRAM   *
*   FILENAME : FND.C                 *
*   PROCESSOR : I8086                 *
*   8255 TEST                         *
*****/
#include    "mde8086.h"

int  data[11] = { 0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92,
                  0x82, 0xf8, 0x80, 0x90, 0x00 };

void  wait(long del)
{
    while( del-- );
}

void  main(void)
{
    int  *data1;

    outportb( PPI1_CR, 0x80 );
    outportb( PPI1_B, 0xf0 );
    outportb( PPI1_C, 0x00 );

    do {
        data1 = data;

        while( *data1 != 0x00 ){
            outportb( PPI1_A, *data1 );
            wait(30000);
            data1++;
        }
    }while(1);
}

```

This C program is written for the **Intel 8086 microprocessor** using the **MDE-Win8086** environment and is designed to test the **Intel 8255 Programmable Peripheral Interface (PPI)**. The program displays digits on a **7-segment LED display** by sending appropriate binary values through **Port A** of the 8255.

Program Overview:

```
yaml
```

```
FILENAME : FND.C
PROCESSOR : I8086
FEATURE : 8255 PPI test using 7-segment display
```

💡 Key Components:

1. `data[]` Array:

```
c
int data[11] = { 0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92,
                0x82, 0xf8, 0x80, 0x90, 0x00 };
```

- This holds **hex codes for 7-segment display digits 0-9**.
- Each hex value corresponds to the binary pattern needed to light up the segments for that digit.
- The last value `0x00` is used as a **sentinel** to end the loop.

2. `wait(long del)` Function:

```
c
void wait(long del) {
    while(del--);
}
```

- Creates a **delay** by looping. Simple software delay to control display duration.

3. `main()` Function:

```
c
void main(void)
{
    int *data1;
```

- `data1` is a pointer used to iterate over the `data[]` array.

⚙️ I/O Port Configuration:

```
c
outportb(PPI1_CR, 0x80); // Control word: set mode for 8255
outportb(PPI1_B, 0xf0); // Write to Port B
outportb(PPI1_C, 0x00); // Write to Port C
```

- `PPI1_CR` (**Control Register**) is set to `0x80`:
 - This configures the 8255 with:
 - Port A: output (used for 7-segment data)

- Port B: output
- Port C: output or input (depending on configuration bits)
- Mode 0 (basic input/output mode)
- `PPI1_B` and `PPI1_C` are also initialized. Their values depend on the specific circuit and are not actively used later.

Main Loop:

```
c
do {
    data1 = data;

    while( *data1 != 0x00 ) {
        outportb( PPI1_A, *data1 ); // Send digit to Port A
        wait(30000);                // Delay for visibility
        data1++;                    // Next digit
    }
} while(1);
```

- **Infinite loop** continuously cycles through the digits 0 to 9.
- For each digit:
 - Sends the corresponding segment pattern to **Port A** (connected to 7-segment display).
 - Waits so the digit stays visible.
 - Moves to the next digit.

Summary:

Feature	Explanation
Platform	Intel 8086 (using MDE-Win8086)
Peripherals	8255 PPI (Programmable Peripheral Interface)
Purpose	Display digits 0–9 repeatedly on a 7-segment display
Technique	Use PPI Port A for output, delay loop for visibility
Educational Use	Microprocessor interfacing lab or simulation environment

Let me know if you'd like a schematic diagram or a simulation of how this connects to a 7-segment display.



how to Display digits 0–9 repeatedly on a 7-segment display


To **display digits 0–9 repeatedly on a 7-segment display**, you need to:

Step-by-Step Process (for Microcontroller or Microprocessor like 8086 + 8255):

1. Understand the 7-Segment Display

A **7-segment display** has 7 LEDs (segments) labeled **a to g**, plus an optional dot (**dp**). You can turn on specific segments to form digits:

Digit	Segments On	Common Cathode Hex Code
0	a b c d e f	0xC0
1	b c	0xF9
2	a b g e d	0xA4
3	a b c d g	0xB0
4	f g b c	0x99
5	a f g c d	0x92
6	a f e d c g	0x82
7	a b c	0xF8
8	All	0x80
9	a b c d f g	0x90

 These values are for **Common Cathode** displays. For **Common Anode**, invert the bits.

2. Connect the Display to the Microprocessor (e.g., via 8255 PPI)

- Connect each segment (a-g) to the **output pins** of **Port A** on the **8255**.
- Connect **common cathode** to ground (for common cathode displays).

3. Write Code to Output the Segment Patterns

Here's a basic logic in **pseudo/C code** (as used in 8086 systems):

```

c

int data[10] = { 0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90 };

void wait(long t) {
    while(t--); // Software delay
}

void main() {
    int i;

    // Initialize 8255 control word to set Port A as output
    outportb(PPI1_CR, 0x80); // Mode 0, all ports output
    while(1) {
        for(i = 0; i < 10; i++) {
            outportb(PPI1_A, data[i]); // Output to 7-segment
            wait(30000);               // Wait for visibility
        }
    }
}

```

4. Infinite Loop for Repeating Digits 0–9

The loop sends each digit pattern to the 7-segment, waits, and moves to the next.

Hardware Requirements:

- 1x **Intel 8086 microprocessor** (or any microcontroller like Arduino/PIC)
- 1x **8255 Programmable Peripheral Interface**
- 1x **7-segment display**
- Resistors (current-limiting for LEDs)
- Breadboard and connecting wires

Summary:

Task	How
Send digit to display	Output hex code to 8255 Port A
Repeat digits	Loop through values 0–9
Delay between digits	Software delay using a loop
Hardware interface	8255 PPI connects to the 7-segment display

If you're using **Arduino**, I can also give you code for that. Let me know your platform.



clarification and explain: A 7-segment display has 7 LEDs (segments) labeled a to g, plus an optional dot (dp). You can turn on specific segments to form digits:

Digit	Segments On	Common Cathode Hex Code
0	a b c d e f	0xC0
1	b c	0xF9
2	a b g e d	0xA4
3	a b c d g	0xB0
4	f g b c	0x99
5	a f g c d	0x92
6	a f e d c g	0x82
7	a b c	0xF8
8	All	0x80
9	a b c d f g	0x90

 These values are for Common Cathode displays. For Common Anode, invert the bits.

Sure! Let's break this down and clarify it for better understanding:

What is a 7-Segment Display?

A **7-segment display** is an electronic component made of **7 LEDs**, arranged to form the shape of digits (0–9). Each LED is called a **segment**, and they are labeled as:

```
lua
-- a --
|      |
f      b
|      |
-- g --
|      |
e      c
|      |
-- d -- (optional dot = dp)
```

Each of these segments (a–g) can be **turned ON or OFF** individually to form numbers or some letters.

Types of 7-Segment Displays

1. Common Cathode (CC):

- All cathodes (negative terminals) of the LEDs are connected together to **GND**.
- To **turn ON a segment**, you apply **HIGH (1)** to its anode.
- But in hardware (like 8255), due to pull-up resistors or driving circuitry, logic **LOW (0)** is often used to **turn ON** a segment (we'll clarify this below).

2. Common Anode (CA):

- All anodes (positive terminals) are connected together to **+Vcc**.
- To **turn ON a segment**, you apply **LOW (0)** to the cathode.

✓ Segment Activation Using Binary / Hex Codes

Each segment corresponds to a bit in an 8-bit pattern. Here's the bit order typically used:

Bit No	Segment	Purpose
7	dp	Decimal Point
6	g	Center bar
5	f	Top left
4	e	Bottom left
3	d	Bottom
2	c	Bottom right
1	b	Top right
0	a	Top

So the pattern ``0b11000000`` (which is ``0xC0``) turns on segments ``a, b, c, d, e, f`` — which displays the digit **0**.

Example Breakdown for Digit `0` (Common Cathode)

To display `0`, you need segments **a b c d e f** ON and segment **g** OFF.

Segment	State
a	ON
b	ON
c	ON
d	ON
e	ON
f	ON
g	OFF
dp	OFF

So in binary:

``dp g f e d c b a = 1 1 0 0 0 0 0 0 = 0xC0``

Note: In **common cathode**, you may assume **0 = ON**, **1 = OFF** for simplicity depending on your hardware driver logic. That's why segment ON → 0.

Common Cathode vs. Common Anode

- **Common Cathode:**
 - ``0 = ON`, `1 = OFF``
 - So digit ``1`` = segments b and c ON = ``0b11111001 = 0xF9``
- **Common Anode:**
 - ``1 = ON`, `0 = OFF``
 - So you **invert** the bits of common cathode values.
 - e.g., for digit ``1``, ``0xF9`` (common cathode) → ``~0xF9 = 0x06`` (common anode)

✓ Summary Table for Digits (Common Cathode)

Digit	Segments ON	Binary	Hex
0	a b c d e f	<code>`11000000`</code>	<code>`0xC0`</code>
1	b c	<code>`11111001`</code>	<code>`0xF9`</code>
2	a b g e d	<code>`10100100`</code>	<code>`0xA4`</code>
3	a b c d g	<code>`10110000`</code>	<code>`0xB0`</code>
4	f g b c	<code>`10011001`</code>	<code>`0x99`</code>
5	a f g c d	<code>`10010010`</code>	<code>`0x92`</code>
6	a f e d c g	<code>`10000010`</code>	<code>`0x82`</code>
7	a b c	<code>`11111000`</code>	<code>`0xF8`</code>

Digit	Segments ON	Binary	Hex
8	a b c d e f g	`10000000`	`0x80`
9	a b c d f g	`10010000`	`0x90`

If you are using a microcontroller or 8255, you'd **send these hex values to the output port** connected to the display to show digits.

Let me know if you'd like a visual simulation or an Arduino version.