PROGRAMMING BOOK FOR MTK85SE, 8085 MICROPROCESSOR TRAINING KIT

Wichit Sirichote

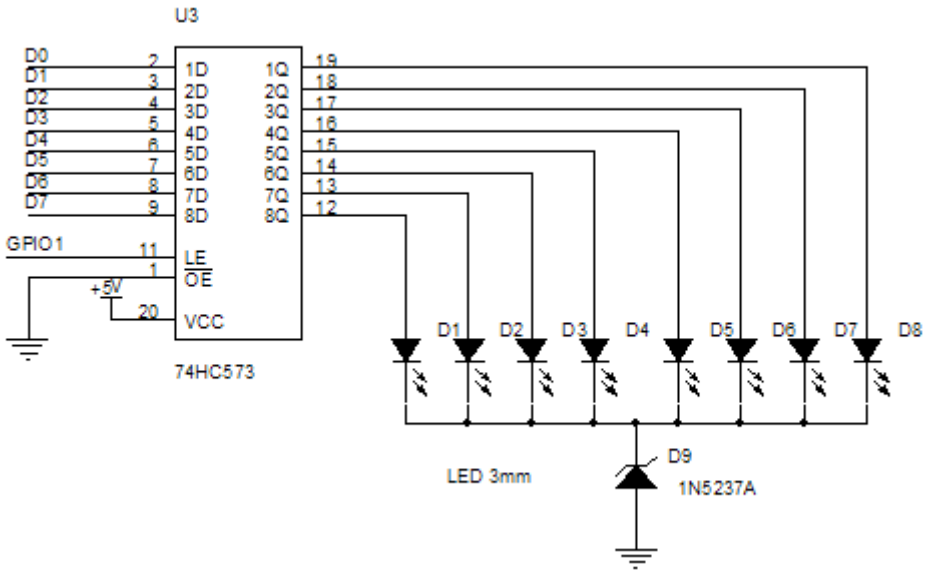
CONTENTS

Program 1: Writing data to output port

We can use debugging LED that connected to GPIO1 port to display the Accumulator content easily. This code will bring the internal 8-bit data in CPU to the real-world.

GPIO1 is 8-bit data flip-flop latch. The GPIO1 latch enable signal, LE is decoded at I/O space location 00.

The output Q1 to Q8 drives small LED. Logic '1' will make LED lit, logic '0' no light.



Let us see how the code brings the Accumulator content to the real-world.

```
Line    Addr Hex code       label Instruction

0001    0000                GPIO1 .EQU 00H
0002    0000
0003    8100                      .ORG 8100H
0004    8100
0005    8100 3E 01        START   MVI A,1
0006    8102 D3 00                OUT GPIO1
0007    8104 FF                   RST 7
0008    8105
0009    8105                      .END
tasm: Number of errors = 0
```

A constant is loaded to the Accumulator register A. Then write it to GPIO1 location. RST 7 will make the program control back to monitor program.

Procedure

1. Enter Hex code from location 8100 to 8104.

2. Press key HOME, then GO.

What is happening at GPIO1 LED?

Exercise

1. Modify the code to send another byte? And see the result on GPIO1 LED.

Summary

Using the LED indicator at the output port register is the simple method for program testing. The kit has 8-bit LED. The byte that needed to check must be loaded into the Accumulator then uses OUT 0, instruction to write it to the 8-bit binary display.

Program 2: Binary number counting

How the microprocessor count the 8-bit binary number?

```
Line   Addr Hex code    label   Instruction

0001   0000             GPIO1  .EQU 00H
0002   0000
0003   8100                     .ORG 8100H
0004   8100
0005   8100 3E 01       START   MVI A,1
0006   8102
0007   8102 D3 00       LOOP    OUT GPIO1
0008   8104 3C                  INR A
0009   8105 C3 02 81            JMP LOOP
0010   8108
0011   8108                     .END
tasm: Number of errors = 0
```

We modify Program 1 by inserting the instruction INR A and JMP LOOP.

INR A will increment the Accumulator by one.

JMP LOOP will let CPU jump back to address 8102.

Procedure

1. Enter Hex code from location 8100 to 8107.

2. Press key HOME, then STEP.

What is happening at GPIO1 LED when we STEP over location 8102?

We can see the counting in binary from 0000 0000 to 0000 0001, 0000 0010 and so on.

STEP key will let CPU execute single instruction. If we press key GO. What is happening?

All LED will look like turn on, like 1111 1111.

If we try another program by adding a small delay between incrementing.

```
Line    Addr Hex         label   Instruction

0001   0000             GPIO1    .EQU 00H
0002   0000
0003   8100                      .ORG 8100H
0004   8100
0005   8100 3E 01       START    MVI A,1
0006   8102
0007   8102 D3 00       LOOP1    OUT GPIO1
0008   8104 3C                   INR A
```

```
0009   8105 CD 0B 81             CALL DELAY
0010   8108 C3 02 81             JMP LOOP1
0011   810B
0012   810B 11 FF FF    DELAY    LXI D,-1
0013   810E 21 00 30             LXI H,3000H
0014   8111 19          LOOP2    DAD D
0015   8112 DA 11 81             JC  LOOP2
0016   8115 C9                   RET
0017   8116
0018   8116                      .END
tasm: Number of errors = 0
```

Enter the hex code and now press key HOME, GO.

We see that such small delay will make us see the binary counting.

Can you modify the code to make it run faster or slower? How?

Summary

The small delay is very useful subroutine for program debugging. Its functioning is to do counting until the initial value becomes zero.

Program 3: LED running

With a bit modification of Program 2, we can show the LED running light with new instruction easily.

```
Line    Addr Hex         label    Instruction

0001    0000             GPIO1    .EQU 00H
0002    0000
0003    8100                      .ORG 8100H
0004    8100
0005    8100 3E 01       START     MVI A,1
0006    8102
0007    8102 D3 00       LOOP1    OUT GPIO1
0008    8104 07                   RLC
0009    8105 CD 0B 81             CALL DELAY
0010    8108 C3 02 81             JMP LOOP1
0011    810B
0012    810B 11 FF FF    DELAY    LXI D,-1
0013    810E 21 00 20             LXI H,2000H
0014    8111 19          LOOP2    DAD D
0015    8112 DA 11 81             JC  LOOP2
0016    8115 C9                   RET
0017    8116
0018    8116                      .END
tasm: Number of errors = 0
```

Now, INR A instruction was replaced with RLC, Rotate left through carry.

Procedure

1. Enter hex code from location 8100 to 8115.

2. Press HOME, then GO.

Exercise

1. Modify the initial load value from 1 to any number. Test it.

2. Change running speed faster and slower.

Summary

8085 has left and right rotation instructions. We can learn theirs operation with the GPIO1 LED and with delay subroutine.

Program 4: Fill constant to RAM

We will use memory pointer and write the constant with a number of byte counted by loop counter.

```
Line    Addr Hex code          label   Instruction

0001   8100                     .ORG 8100H
0002   8100
0003   8100 21 00 90    START   LXI H,9000H     ; address pointer
0004   8103 06 08              MVI B,8          ; loop counter
0005   8105
0006   8105 3E 00              MVI A,0          ; constant
0007   8107 77          LOOP    MOV M,A         ; write to memory
0008   8108 23                  INX H           ; next address
0009   8109
0010   8109 05                  DCR B           ; decrement counter
0011   810A C2 07 81            JNZ LOOP        ; done?
0012   810D
0013   810D FF                  RST 7           ; jump to monitor
0014   810E
0015   810E
0016   810E                        .END
tasm: Number of errors = 0
```

The 8085 CPU uses HL as the 16-bit memory pointer (H for High address and L for Low address). The instruction MOV M,A will use HL as the pointer, register A content will copy to memory pointed to by HL. This is called indirect addressing.

Register B uses as the loop counter. The example will write 0 to memory from location 9000H to 9007H, 8 bytes.

Procedure

1. Enter hex code from location 8100 to 810D.

2. Write down the contents of memory location 9000 to 9007.

3. Press key HOME then GO.

4. Check and write down the contents of memory location 9000 to 9007.

Location	9000	9001	9002	9003	9004	9005	9006	9007
Before								
After								

Exercise

1. Modify code to fill constant FF to location 9000 to 90FF. Show the result.

Summary

Indirect addressing using HL register provides a flexible memory access for many applications.

Program 5: 16-Bit Binary addition

This program shows how to add multiple bytes using ADI and ACI instructions.

```
Line    Addr Hex          label   Instruction

0001   8100                      .ORG 8100H
0002   8100
0003   8100 21 00 90    START   LXI H,9000H
0004   8103 7E                   MOV A,M
0005   8104 C6 78                ADI 78H
0006   8106 77                   MOV M,A
0007   8107
0008   8107 23                   INX H
0009   8108 7E                   MOV A,M
0010   8109 CE 9A                ACI 9AH
0011   810B 77                   MOV M,A
0012   810C FF                   RST 7
0013   810D
0014   810D                      .END
tasm: Number of errors = 0
```

We have two 16-bit numbers to be added.

The first number is stored in memory location 9000 (low byte) and 9001(high byte).

The second number is 16-bit immediate value. 9A78H.

We can enter the first number at location 9000 (low byte) and 9001(high byte).

Adding is done by ADI instruction for low byte and ACI for high byte.

Result will put back to location 9000 (low byte) and 9001(high byte).

Location	[9001]	[9000]
+	XX	YY
	9A	78

ADI, add with no carry will use for adding low order byte. For the next higher significant byte we will use ACI, add with carry flag to include carry bit if there is.

Procedure

1. Enter hex code from location 8100 to 810A.

2. Suppose the first number is 1ABBH. Enter it to location 9000(low byte) and 9001(high byte).

3. Compute it by hand, keep the result.

4. Now press key HOME, then GO.

5. Check the result that saved in RAM at location 9000(low byte) and 9001(high byte).

Program 6: BCD addition

By inserting DAA instruction after ADI and ACI instructions, we can add two 4-digit BCD numbers easily.

```
0001   8100                      .ORG 8100H
0002   8100
0003   8100 21 00 90    START    LXI H,9000H
0004   8103 7E                   MOV A,M
0005   8104 C6 99                ADI 99H
0006   8106 27                   DAA
0007   8107 77                   MOV M,A
0008   8108
0009   8108 23                   INX H
0010   8109 7E                   MOV A,M
0011   810A CE 19                ACI 19H
0012   810C 27                   DAA
0013   810D 77                   MOV M,A
0014   810E FF                   RST 7
0015   810F
0016   810F                      .END
tasm: Number of errors = 0
```

We see that the code for BCD addition is similar to binary addition, only there are DAA after ADI and ACI instruction.

Procedure

1. Enter hex code from location 8100 to 810E.

2. Suppose the first number is 1999H. Enter it to location 9000(low byte) and 9001(high byte).

3. Compute it by hand, keep the result.

4. Now press key HOME, then GO.

5. Check the result that saved in RAM at location 9000(low byte) and 9001(high byte).

Location	[9001]	[9000]
+	XX	YY
	19	99

Exercise

1. Try another BCD number that saved in RAM at 9000 and 9001.

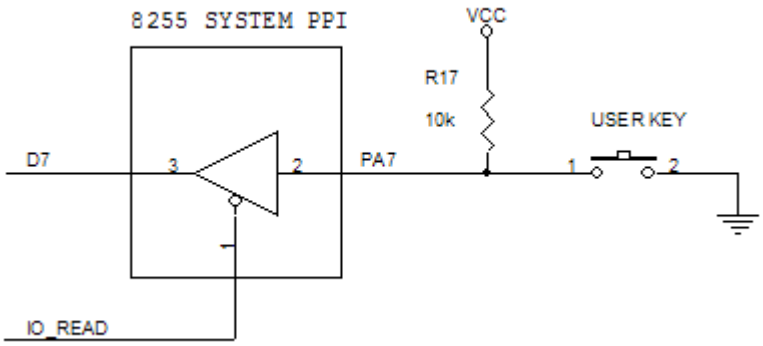
Summary

DAA can be used with binary adding instructions. It will adjust the result to BCD number automatically.

Program 7: Reading button press

We can test button press with USER key that tied to PA7 of the system PPI, 8255. PA7 is bit7 of PORTA. The input logic appears at PA7 is logic '1' with 10k pull-up resistor. When the button was pressed PA7 will short to GND, the logic will be '0'.

We can read this status by instruction IN PORT directly. The byte will be read to the Accumulator.



Our test program will read key press, then increment the byte at location 9000. We can see the incrementing by writing the content of location 9000 to GPIO1 LED.

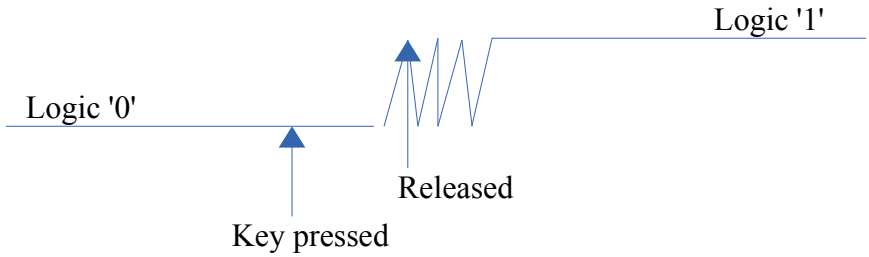
```
Line    Addr hex code     Label    Instruction

0001    0000              PORTA   .EQU 10H
0002    0000              GPIO1   .EQU 00H
0003    0000
0004    8100                      .ORG 8100H
0005    8100
0006    8100 21 00 90     START    LXI H,9000H
0007    8103
0008    8103 DB 10        PRESSED IN PORTA
0009    8105 E6 80                ANI 80H
0010    8107 CA 03 81             JZ PRESSED
0011    810A
0012    810A CD 22 81             CALL DEBOUNCE
0013    810D
0014    810D DB 10        READ    IN PORTA
0015    810F E6 80                ANI 80H
0016    8111 C2 0D 81             JNZ READ
0017    8114
0018    8114 CD 22 81             CALL DEBOUNCE
0019    8117 CD 1D 81             CALL WRITE_A
0020    811A
0021    811A C3 03 81             JMP PRESSED
0022    811D
0023    811D 34           WRITE_A INR M
0024    811E 7E                   MOV A,M
```

```
0025   811F D3 00               OUT GPIO1
0026   8121 C9                  RET
0027   8122
0028   8122 06 00       DEBOUNCE MVI B,0
0029   8124 05          DELAY1   DCR B
0030   8125 C2 24 81             JNZ DELAY1
0031   8128 C9                  RET
0032   8129
0033   8129                    .END
tasm: Number of errors = 0
```

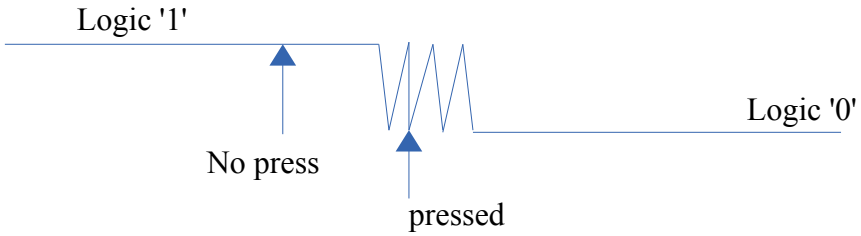
Yellow portions are key press checking. The first one is to wait if key has been pressed. The status of logic at PA7 is checking by logical AND instruction with 80H. Bit position of PA7 is bit 7. Thus we use 80H for bit testing. If logic is '0', Result from logical AND will give ZERO. Thus JZ PRESSED will repeat checking it until it is logic '1', or key has been released.

Logic appears at PA7 could be like this.



Tact switch is mechanical contact. It has elastic property, vibrate in a short period. The logic signal seen by CPU execution would be spike signal. Simple method to get stable logic readings is done by providing a short wait until the logic is stable. We thus insert small delay called DEBOUNCE subroutine.

The second yellow portion is now checking until key was pressed. Again we also get the same bouncing signal but in opposite direction.



So we wait until the logic is stable, then execute the desired task. Our task is to increment location 9000, then write it to GPIO1 LED.

Procedure

1. Enter hex code from location 8100 to 8128.

2. Now press key HOME, then GO.

3. Press USER key, what is happening at GPIO1 LED?

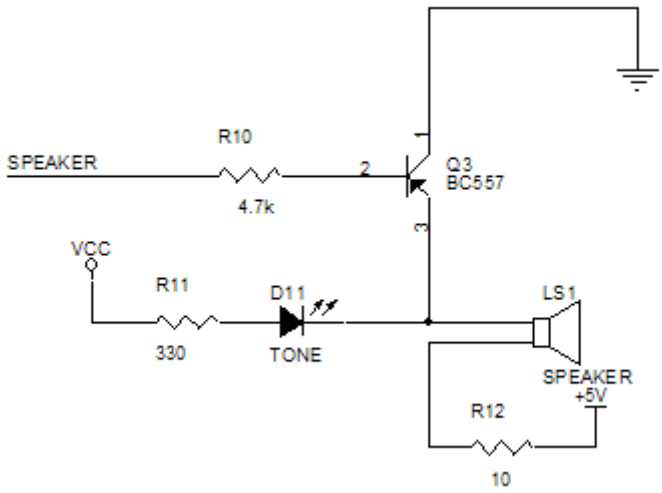
Exercise

1. Try reduce the delay period.

Program 8: Producing tone signal

The kit has one bit that drives a small speaker. We can make a tone signal and hear it from the loudspeaker. Q3 is PNP transistor switch. It can drive the speaker with logic input at SPEAKER signal.

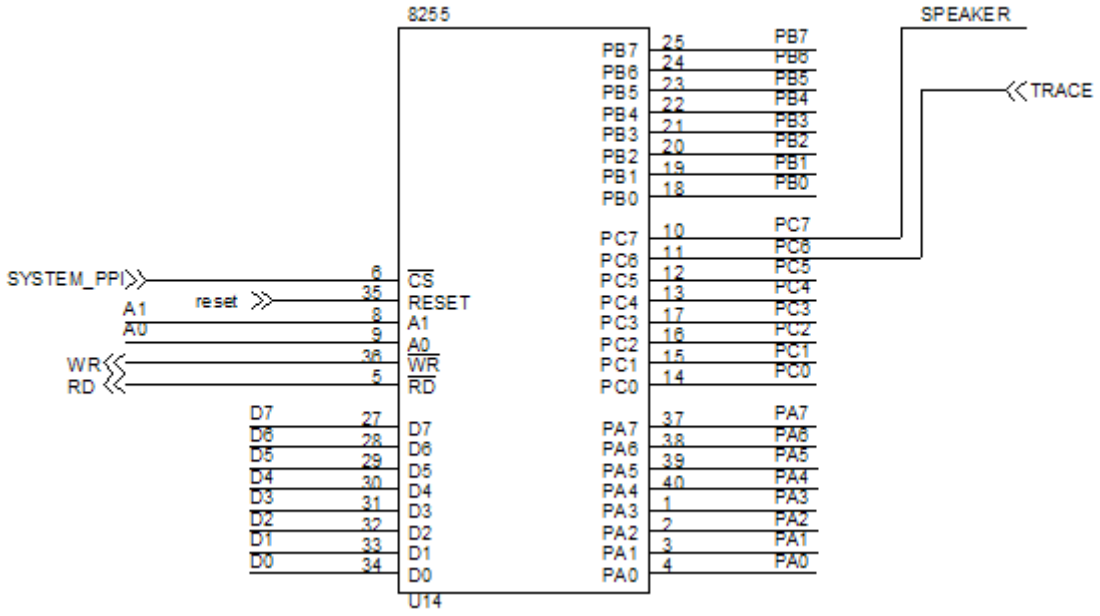
Logic low of SPEAKER signal will turn on Q3, and logic high will turn it off.



We can write a program that makes 50% duty cycle of tone signal.

```
0001   0000             PORTC   .EQU 12H
0002   0000
0003   8100                     .ORG 8100H
0004   8100
0005   8100 3E FF       START    MVI A,0FFH
0006   8102 32 00 90             STA 9000H
0007   8105
0008   8105 3A 00 90    TONE     LDA 9000H
0009   8108 EE 80                XRI 10000000B
0010   810A 32 00 90             STA 9000H
0011   810D D3 12                OUT PORTC
0012   810F
0013   810F 06 20                MVI B,20H
0014   8111 05          LOOP     DCR B
0015   8112 C2 11 81             JNZ LOOP
0016   8115
0017   8115 C3 05 81             JMP TONE
0018   8118
0019   8118                     .END
tasm: Number of errors = 0
```

Location 9000 stores a byte that will be sent to PORTC.
The data is loaded with FF. We see at the circuit of PORTC, the TRACE signal must be HIGH, to prevent TRAP signal to be activated.



Toggle PC7 is done by logical Exclusive OR instruction with a mask byte, 80H or 10000000b.

Small delay is done by register B.

Procedure

1. Enter hex code from location 8100 to 8117.

2. Now press key HOME, then GO.

3. Can you hear the tone signal?

Exercise

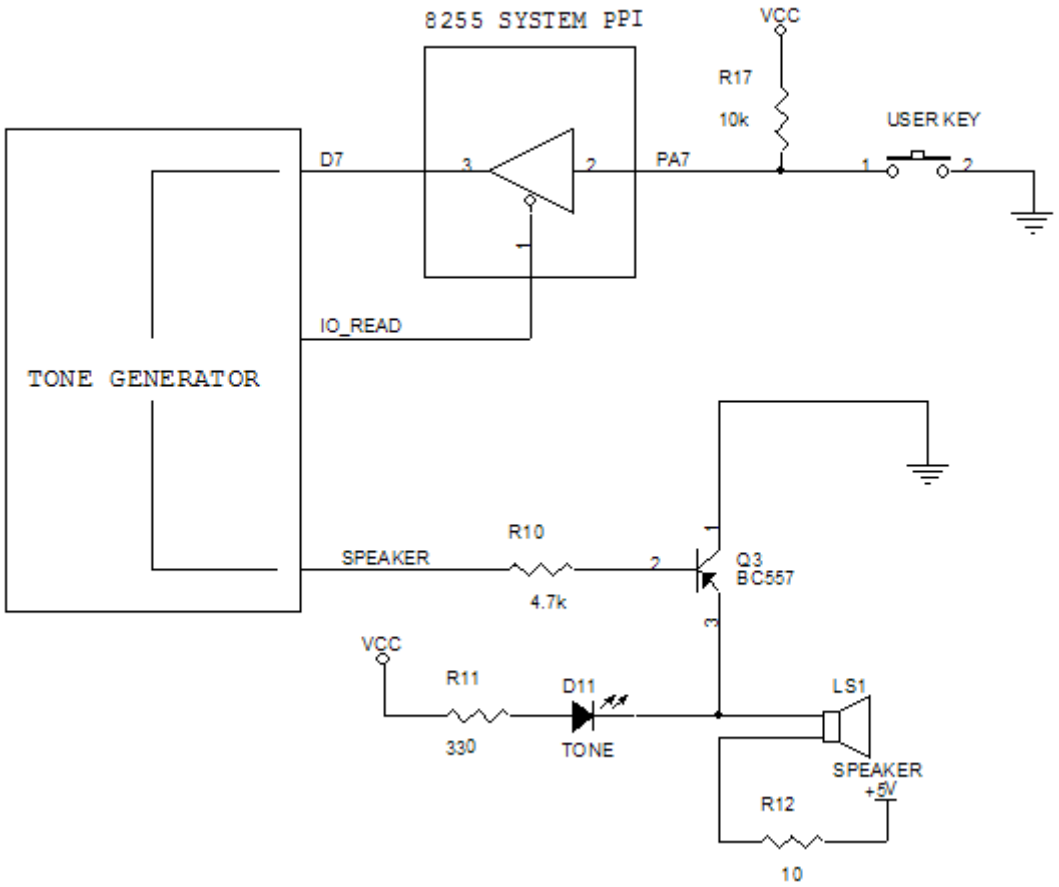
1. Modify the delay period to change the tone frequency.

Summary

The output bit at loudspeaker is one bit. So we can make a tone signal as the square wave signal easily. For some applications that produces purely sinusoidal waveform, like sine wave tone signal, we will need more output bits and may need Digital to Analog converter chip.

Program 9: Morse code keyer

Let us have some fun with Morse code keyer program. This program combines Program 7 and 8. It is a simple tone generator when we press key.




```
0001    0000              PORTA   .EQU 10H
0002    0000              PORTC   .EQU 12H
0003    0000
0004    8100                      .ORG 8100H
0005    8100
0006    8100 3E FF        START    MVI A,0FFH
0007    8102 32 00 90               STA 9000H
0008    8105
0009    8105 DB 10        PRESSED IN PORTA
0010    8107 E6 80                ANI 80H
0011    8109 C2 12 81             JNZ READ
0012    810C CD 1F 81             CALL TONE
0013    810F C3 05 81             JMP PRESSED
0014    8112
0015    8112
0016    8112 DB 10        READ     IN PORTA
0017    8114 E6 80                ANI 80H
0018    8116 C2 12 81             JNZ READ
```

```
0019   8119
0020   8119 CD 30 81            CALL DEBOUNCE
0021   811C C3 05 81            JMP PRESSED
0022   811F
0023   811F 3A 00 90    TONE    LDA 9000H
0024   8122 EE 80               XRI 10000000B
0025   8124 32 00 90            STA 9000H
0026   8127 D3 12               OUT PORTC
0027   8129
0028   8129 06 20               MVI B,20H
0029   812B 05          LOOP    DCR B
0030   812C C2 2B 81            JNZ LOOP
0031   812F C9                  RET
0032   8130
0033   8130
0034   8130 06 00       DEBOUNCE MVI B,0
0035   8132 05          DELAY1  DCR B
0036   8133 C2 32 81            JNZ DELAY1
0037   8136 C9                  RET
0038   8137
0039   8137                    .END
tasm: Number of errors = 0
```

We see that while the key has been pressed, tone signal will be produced.

Procedure

1. Enter hex code from location 8100 to 8136.

2. Now press key HOME, then GO.

3. Press key USER and try making Morse code?

Exercise

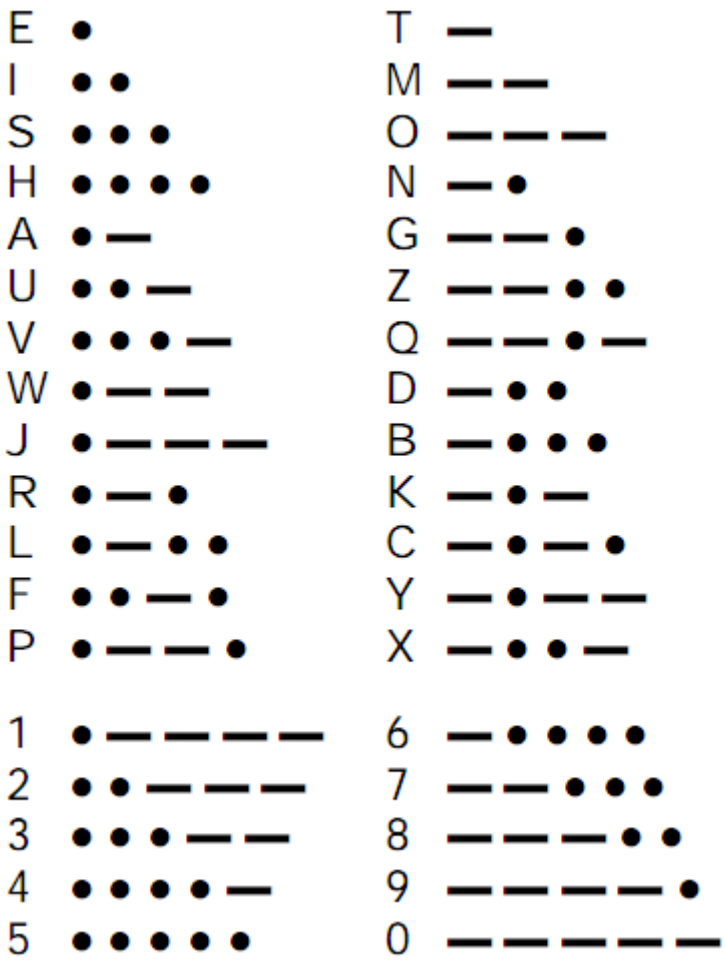
1. Modify the tone frequency and play again.

Summary

More complicated program can translate the text to Morse code. However it is more fun to learn Morse code and hit the key manually.

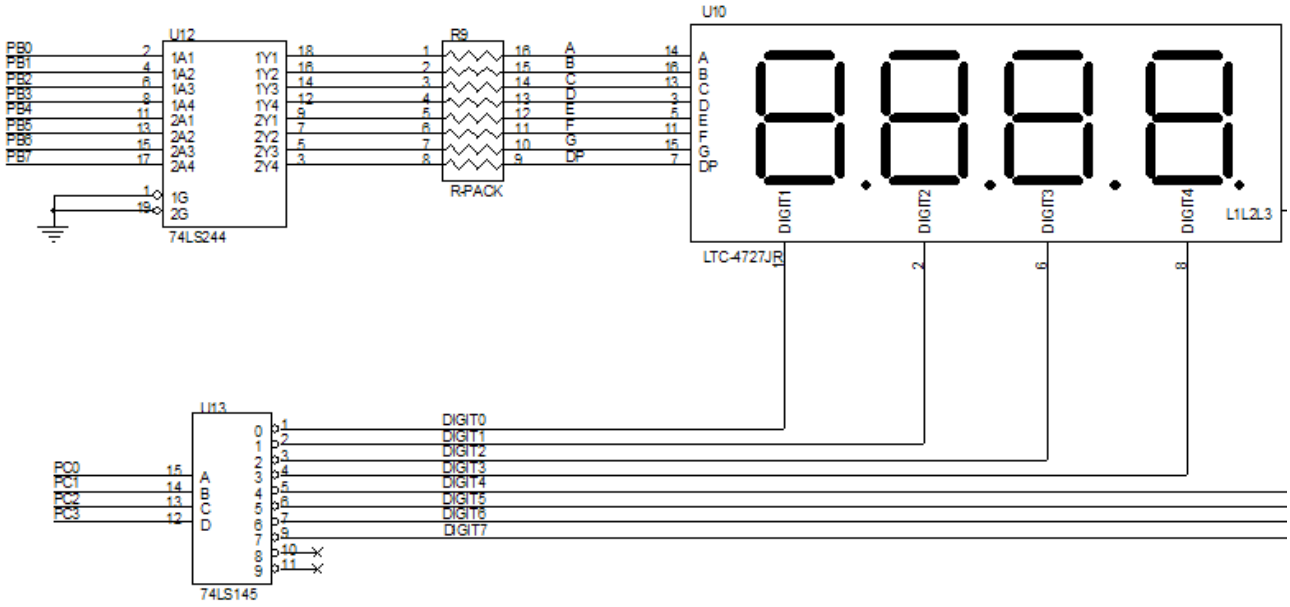
Here is the Morse code table for playing.

Source: http://rsgb.org/main/operating/morse/



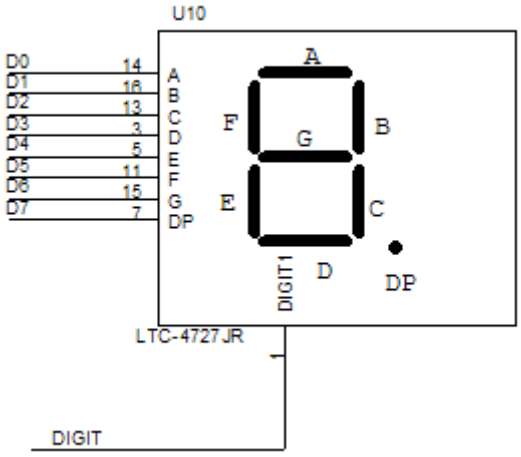
Program 10: 7-Segment display

The kit display is made with common cathode 7-segment LED, LTC472. Segment a, b, c, d, e, f, g, DP are driven by PORTB. All segments are common connected to all 8 digits (only 4-digit shown). U12 is segment driver with R9 for limiting driving current. To activate a given digit, the common cathode pin must be activated with logic LOW.



PORTC, PC0 to PC3 drives the 4-bit decoder, 74LS145 providing 8-digit for CC pin driver.

Segment designation for bit driven is shown below. For example, to display number '1', segment B and C must be '1'. The bit pattern will be 03. We can find the bit pattern for a given letter easily.



Such display is called multiplex display. At a given time, only one digit will be activated. However if we switch each digit with corresponding segment fast enough, we will see all digits with no blinking.

More features for this circuit, we can use PWM method to reduce the power consumption of the LED thus longer theirs life. We can adjust duty cycle for the driving signal.
Here is 50% duty cycle driving pulse.

Smaller duty cycle will reduce power consumption, reduce the LED brightness as well. Proper duty cycle will give nice display with less power.

The first example will display one digit with PWM method.

We will now test the code for single digit display.

```
0001   0000             SEGMENT.EQU 11H
0002   0000             DIGIT  .EQU 12H
0003   0000
0004   8100                    .ORG 8100H
0005   8100
0006   8100 3E 7F       START   MVI A,7FH
0007   8102 D3 11               OUT SEGMENT
0008   8104 3E F0               MVI A,0F0H
0009   8106 D3 12               OUT DIGIT
0010   8108
0011   8108 CD 15 81            CALL TIMEON
0012   810B
0013   810B 3E 00               MVI A,0
0014   810D D3 11               OUT SEGMENT
0015   810F
0016   810F CD 1C 81            CALL TIMEOFF
0017   8112
0018   8112 C3 00 81            JMP START
0019   8115
0020   8115
0021   8115 06 01       TIMEON  MVI B,1
0022   8117 05          LOOP1   DCR B
0023   8118 C2 17 81            JNZ LOOP1
0024   811B C9                  RET
0025   811C
0026   811C 06 64       TIMEOFF MVI B,100
0027   811E 05          LOOP2   DCR B
0028   811F C2 1E 81            JNZ LOOP2
0029   8122 C9                  RET
0030   8123
```

```
0031   8123                    .END
tasm: Number of errors = 0
```

Yellow portion will make the first digit to display number '8', code pattern is 7FH.

Time on subroutine makes a turn on period.

Then all segments will be turned off with Time off subroutine for no light period.

Procedure

1. Enter hex code from location 8100 to 8122.

2. Now press key HOME, then GO.

3. Observe the display, see the brightness of the LED.

4 Adjust duty cycle and observe the brightness again.

Exercise

1. Try with another pattern and another digit.

To display all 8 digits, we will use scanning method.

```
0001   0000             SEGMENT .EQU 11H
0002   0000             DIGIT   .EQU 12H
0003   0000
0004   8100                    .ORG 8100H
0005   8100
0006   8100 21 26 81    START   LXI H,TEXT
0007   8103 CD 09 81            CALL SCAN
0008   8106 C3 00 81            JMP START
0009   8109
0010   8109
0011   8109             ;SCAN DISPLAY
0012   8109             ;ENTRY: HL
0013   8109
0014   8109 0E 08       SCAN    MVI C,8
0015   810B 1E 00               MVI E,0
0016   810D
0017   810D 7B          SCAN1   MOV A,E
0018   810E F6 F0               ORI 0F0H
0019   8110 D3 12               OUT DIGIT
0020   8112
0021   8112 7E                  MOV A,M
0022   8113 D3 11               OUT SEGMENT
0023   8115
0024   8115 06 01       TIMEON  MVI B,1
0025   8117 05          LOOP1   DCR B
0026   8118 C2 17 81            JNZ LOOP1
0027   811B
0028   811B 3E 00               MVI A,0
```

```
0029   811D D3 11               OUT SEGMENT
0030   811F
0031   811F 1C                  INR E
0032   8120 23                  INX H
0033   8121
0034   8121 0D                  DCR C
0035   8122 C2 0D 81            JNZ SCAN1
0036   8125 C9                  RET
0037   8126
0038   8126
0039   8126 3F          TEXT    .BYTE 3FH  ; '0'
0040   8127 06                  .BYTE 06H  ; '1'
0041   8128 5B                  .BYTE 5BH  ; '2'
0042   8129 4F                  .BYTE 4FH  ; '3'
0043   812A 66                  .BYTE 66H  ; '4'
0044   812B 6D                  .BYTE 6DH  ; '5'
0045   812C 7D                  .BYTE 7DH  ; '6'
0046   812D 07                  .BYTE 07H  ; '7'
0047   812E
0048   812E 7F                  .BYTE 7FH  ; '8'
0049   812F 6F                  .BYTE 6FH  ; '9'
0050   8130 77                  .BYTE 77H  ; 'A'
0051   8131 7C                  .BYTE 7CH  ; 'b'
0052   8132 39                  .BYTE 39H  ; 'C'
0053   8133 5E                  .BYTE 5EH  ; 'd'
0054   8134 79                  .BYTE 79H  ; 'E'
0055   8135 71                  .BYTE 71H  ; 'F'
0056   8136
0057   8136                .END
tasm: Number of errors = 0
```

Subroutine scan uses HL as the buffer display pointer.

Register C is number of digit to be scanned.

Register E is digit scan code, which is 0 to 7 for 8-digit.

The example uses 8-byte from 8126 to 812D to be a display buffer.

Procedure

1. Enter hex code from location 8100 to 8135.

2. Now press key HOME, then GO.

3. Observe the display, see the brightness of the LED. And the number being displayed.

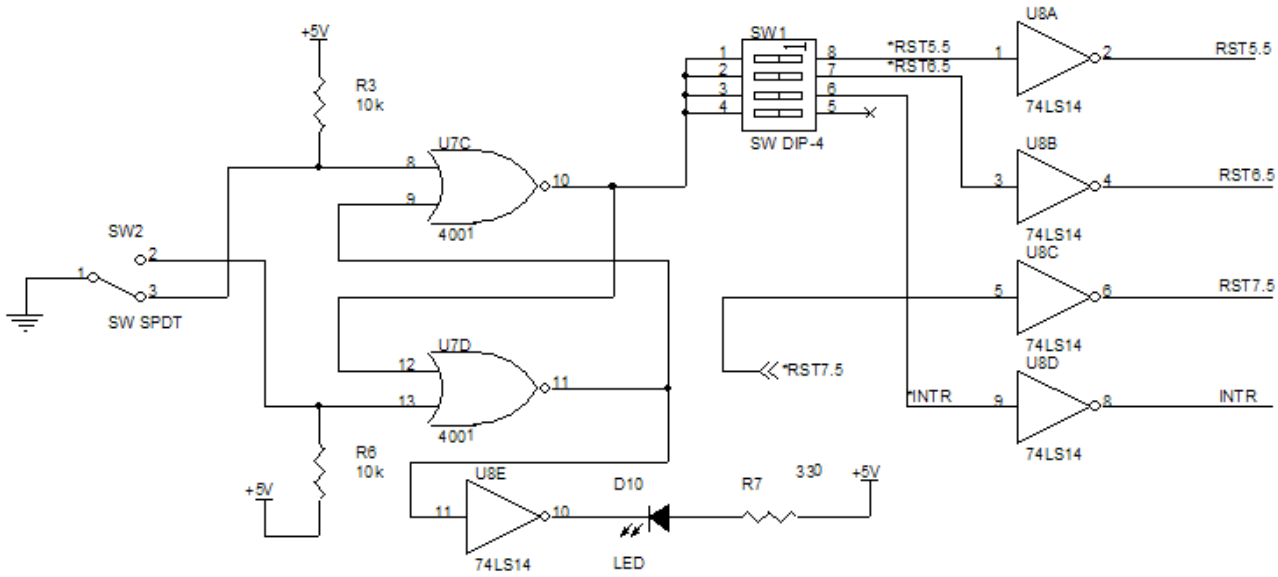
4. Change HL to 812E, test it again. See any change?

Exercise

1. Can you show 'HELLO JO' on the display, how?

Program 11: Hardware interrupt

The kit provides test button that produces single pulse for interrupt experiment. SW2 is push button, when press and release, the single pulse will send to interrupt pins.



For this test we will select RST6.5, so SW1 must set DIP switch position 2 ON.

```
0001    8034                    .ORG 8034H
0002    8034 C3 06 81           JMP SERVICE_RST6.5
0003    8037
0004    8100                    .ORG 8100H
0005    8100
0006    8100 F3          MAIN    DI
0007    8101 3E 0D               MVI A,1101B
0008    8103 30                  SIM
0009    8104 FB                  EI
0010    8105
0011    8105 FF                  RST 7
0012    8106
0013    8106             SERVICE_RST6.5
0014    8106
0015    8106 F5                  PUSH PSW
0016    8107
0017    8107 3E 34               MVI A,34H
0018    8109 D3 00               OUT 0
0019    810B
0020    810B F1                  POP PSW
0021    810C FB                  EI
0022    810D C9                  RET
0023    810E
0024    810E                    .END
0025    810E
0026    810E
```

```
tasm: Number of errors = 0
```

The interrupt vector for RST6.5 is located at 0034H. The kit relocate this vector to RAM space at location 8034H. User can enter JUMP instruction at 8034H to the service routine for RST6.5 easily.

Since the RST6.5 is a maskable, so to enable it, we must set the mask bit for RST6.5 to '0'. And use instruction SIM to to set it. That is all for main code then the control jump back to monitor program.

When we press test button, the CPU will jump to RST6.5 vector and jump to 8106. The service routine will write a byte 34H to GPIO1 LED. We can see the binary 34H on the GPIO1 LED directly.

Procedure

1. Enter hex code at 8034-8036 for JUMP instruction.

2. Enter hex code from 8100 to 810D.

3. Now press key HOME, then GO.

3. Observe the display, push the test button to produce interrupt signal for RST6.5 pin.

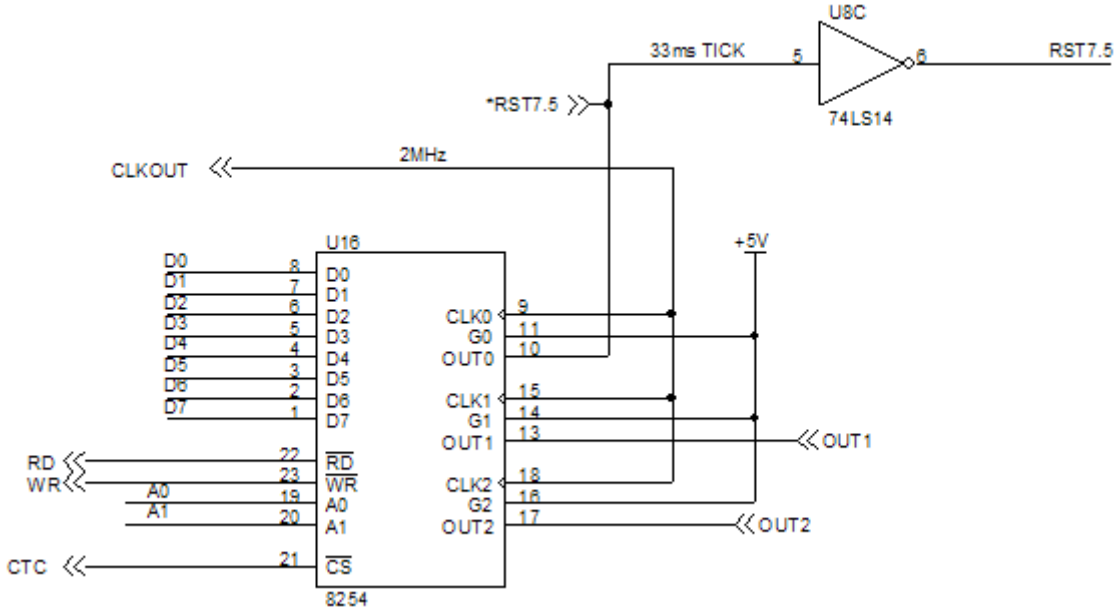
4. Press RESET, observe when push the test button again.

Exercise

1. Modify the code for testing RST5.5 interrupt pin.

Program 12: Timer interrupt

The kit provides 33ms tick generated from 8254 timer. The input clock 2MHz supplied to 8254 chip will be divided by 65536 producing 33ms tick. This tick signal is fed to RST7.5.



The monitor program initializes such tick signal at the beginning when reset the CPU. User can use this tick signal for many time trigger applications.

We will play with this tick by RST7.5 interrupt pin. The hardware ties RST7.5 to the output of 8254 directly. No jumper setting needed.

```
0001    0000
0002    803C                    .ORG 803CH
0003    803C C3 0D 81           JMP SERVICE_RST7.5
0004    803F
0005    8100                    .ORG 8100H
0006    8100
0007    8100 F3          MAIN    DI
0008    8101 3E 0B               MVI A,1011B
0009    8103 30                  SIM
0010    8104 FB                  EI
0011    8105
0012    8105 AF                  XRA A
0013    8106 32 00 90            STA SEC_33
0014    8109 32 01 90            STA SECOND
0015    810C
0016    810C FF                  RST 7
0017    810D
0018    810D             SERVICE_RST7.5
0019    810D
0020    810D F5                  PUSH PSW
0021    810E
0022    810E 3A 00 90            LDA SEC_33
0023    8111 3C                  INR A
0024    8112 32 00 90            STA SEC_33
```

```
0025   8115 FE 1E               CPI 30
0026   8117 C2 29 81            JNZ SKIP
0027   811A
0028   811A AF                 XRA A
0029   811B 32 00 90           STA SEC_33
0030   811E
0031   811E 3A 01 90           LDA SECOND
0032   8121 C6 01              ADI 1
0033   8123 27                 DAA
0034   8124 32 01 90           STA SECOND
0035   8127 D3 00              OUT 0
0036   8129
0037   8129             SKIP
0038   8129
0039   8129 F1                 POP PSW
0040   812A FB                 EI
0041   812B C9                 RET
0042   812C
0043   9000                    .ORG 9000H
0044   9000
0045   9000             SEC_33 .BLOCK 1
0046   9001             SECOND .BLOCK 1
0047   9002
0048   9002
0049   9002                   .END
0050   9002
0051   9002
tasm: Number of errors = 0
```

Main code initializes RST7.5 mask bit, clear two variables: SEC_33 and SECOND then enable interrupt and return to monitor program.

The RST7.5 service routine will be entered every 33ms or approx. 30 cycles/second.

Every entering, the SEC_33 is incremented by one. When it was equal to 30, time has elapsed for one second. We then increment the SEC variable and write it to GPIO1 LED. We can see the BCD counting up every one second on the GPIO1 LED.

Procedure

1. Enter hex code at 803C-803E for JUMP instruction.

2. Enter hex code from 8100 to 812B.

3. Now press key HOME, then GO.

3. Observe the display. Try pressing keypad as there is no timer interrupt.

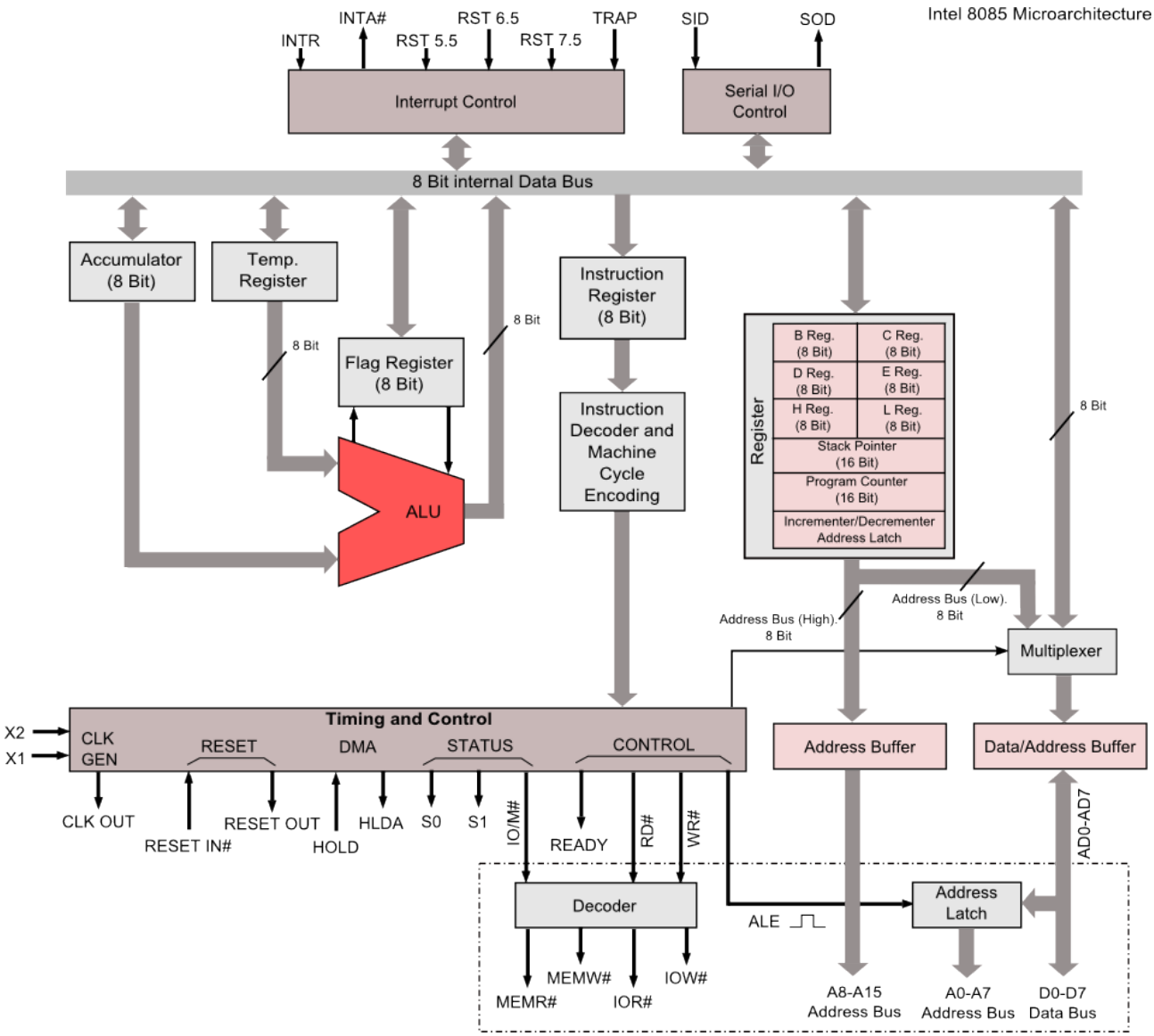
4. Press RESET, observe again.

Exercise

1. Change the update rate from one second to 3 seconds.

8085 Micro Architecture

Source: By Appaloosa - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=5217009



8085 Instruction Hex Code

MOVE, LOAD and STORE

```
40          MOV   B,B
41          MOV   B,C
42          MOV   B,D
43          MOV   B,E
44          MOV   B,H
45          MOV   B,L
46          MOV   B,M
47          MOV   B,A
48          MOV   C,B
49          MOV   C,C
4A          MOV   C,D
4B          MOV   C,E
4C          MOV   C,H
4D          MOV   C,L
4E          MOV   C,M
4F          MOV   C,A
50          MOV   D,B
51          MOV   D,C
52          MOV   D,D
53          MOV   D,E
54          MOV   D,H
55          MOV   D,L
56          MOV   D,M
57          MOV   D,A
58          MOV   E,B
59          MOV   E,C
5A          MOV   E,D
5B          MOV   E,E
5C          MOV   E,H
5D          MOV   E,L
6B          MOV   L,E
6C          MOV   L,H
6D          MOV   L,L
6E          MOV   L,M
6F          MOV   L,A
70          MOV   M,B
71          MOV   M,C
72          MOV   M,D
73          MOV   M,E
74          MOV   M,H
75          MOV   M,L
77          MOV   M,A
78          MOV   A,B
79          MOV   A,C
7A          MOV   A,D
7B          MOV   A,E
7C          MOV   A,H
7D          MOV   A,L
7E          MOV   A,M
7F          MOV   A,A

3E nn       MVI A,byte
06 nn       MVI B,byte
0E nn       MVI C,byte
16 nn       MVI D,byte
1E nn       MVI E,byte
26 nn       MVI H,byte
2E nn       MVI L,byte
36 nn       MVI M,byte

01 nnnn     LXI B,dble
11 nnnn     LXI D,dble
21 nnnn     LXI H,dble
31 nnnn     LXI SP,dble

02          STAX  B
12          STAX  D
0A          LDAX  B
1A          LDAX  D
32 nnnn     STA   adr
3A nnnn     LDA   adr
22 nnnn     SHLD  adr
2A nnnn     LHLD  adr
EB          XCHG
```

COMPARE

```
FE nn       CPI   byte
B8          CMP   B
B9          CMP   C
BA          CMP   D
BB          CMP   E
BC          CMP   H
BD          CMP   L
BE          CMP   M
BF          CMP   A
```

ROTATE

```
07          RLC
17          RAL
0F          RRC
1F          RAR
```

STACK

```
C5          PUSH  B
D5          PUSH  D
E5          PUSH  H
F5          PUSH  PSW

C1          POP   B
D1          POP   D
E1          POP   H
F1          POP   PSW
E3          XTHL
F9          SPHL
```

```
33          INX     SP
3B          DCX     SP
```

ARITHEMATICS

```
C6 nn       ADI     byte
CE nn       ACI     byte

80          ADD     B
81          ADD     C
82          ADD     D
83          ADD     E
84          ADD     H
85          ADD     L
86          ADD     M
87          ADD     A
88          ADC     B
89          ADC     C
8A          ADC     D

96          SUB     M
97          SUB     A
98          SBB     B
99          SBB     C
9A          SBB     D
9B          SBB     E
9C          SBB     H
9D          SBB     L
9E          SBB     M
9F          SBB     A

09          DAD     B
19          DAD     D
29          DAD     H
39          DAD     SP
```

CALL

```
CD nnnn     CALL    adr
DC nnnn     CC      adr
D4 nnnn     CNC     adr
CC nnnn     CZ      adr
C4 nnnn     CNZ     adr
F4 nnnn     CP      adr
FC nnnn     CM      adr
EC nnnn     CPE     adr
E4 nnnn     CPO     adr
```

RETURN

```
C9          RET
D8          RC
D0          RNC
C8          RZ
C0          RNZ
F0          RP
F8          RM
E8          RPE
E0          RPO
```

RESTART

```
C7          RST     0
CF          RST     1
D7          RST     2
DF          RST     3
E7          RST     4
EF          RST     5
FF          RST     7
```

SPECIALS

```
2F          CMA
37          STC
3F          CMC
27          DAA
```

INPUT/OUTPUT

```
DB nn       IN      byte
D3 nn       OUT     byte
```

INCREMENT/DECREMENT

```
04          INR     B
0C          INR     C
14          INR     D
1C          INR     E
24          INR     H
2C          INR     L
34          INR     M
3C          INR     A
03          INX     B
13          INX     D
23          INX     H
05          DCR     B
0D          DCR     C
15          DCR     D
1D          DCR     E
25          DCR     H
2D          DCR     L
35          DCR     M
3D          DCR     A
0B          DCX     B
1B          DCX     D
2B          DCX     H
```

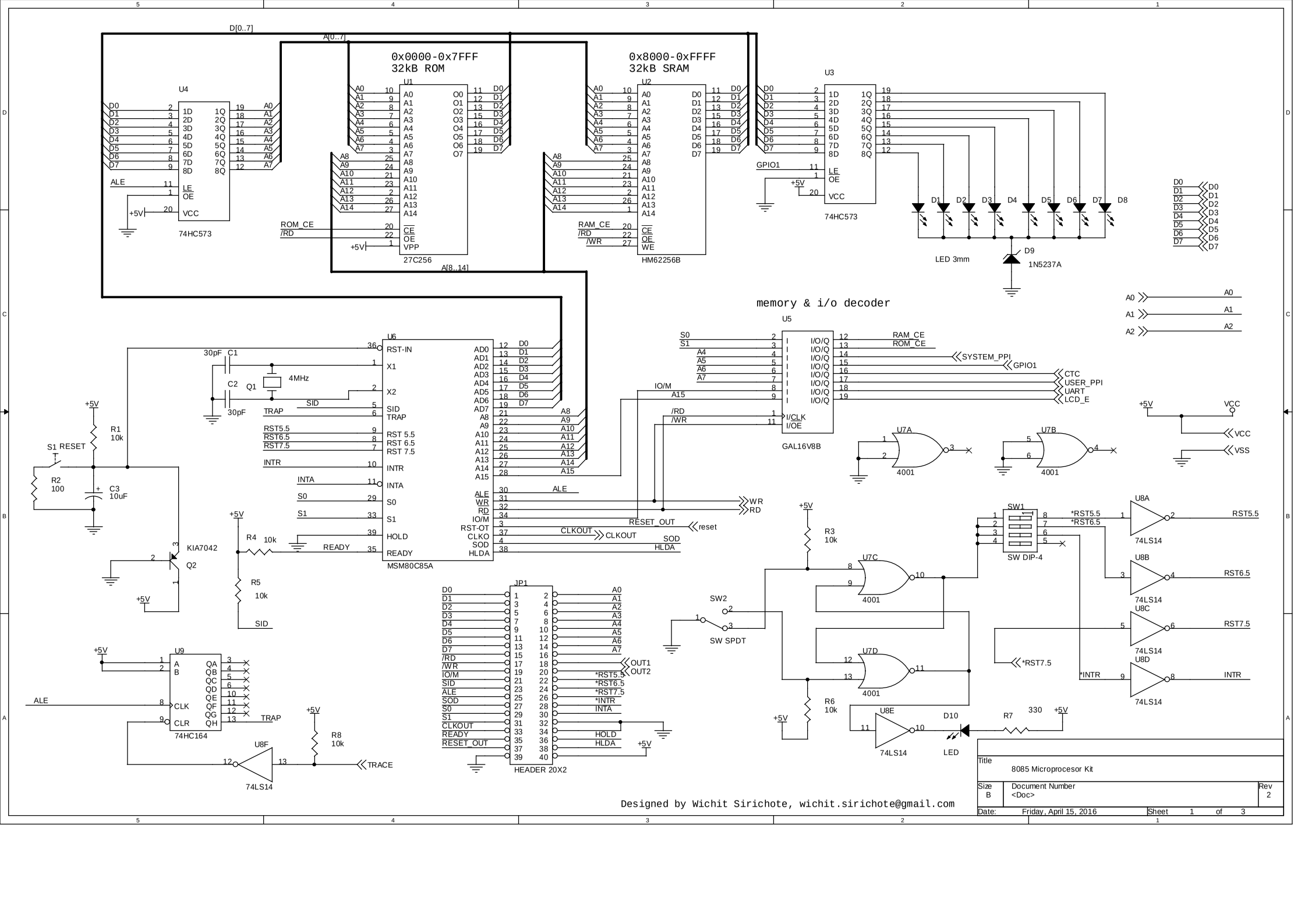
JUMP

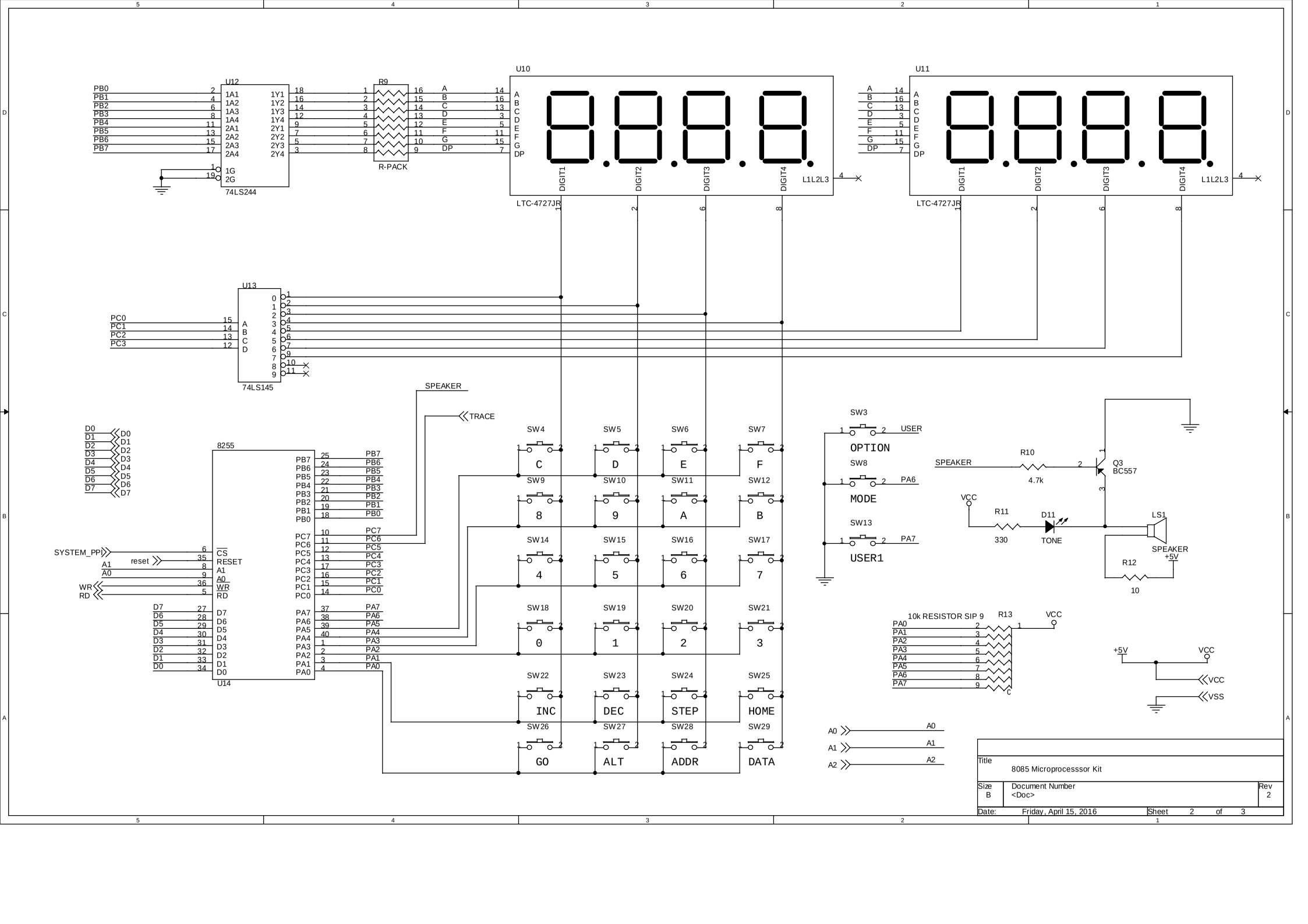
```
C3 nnnn     JMP     adr
```

```
DA nnnn     JC      adr
D2 nnnn     JNC     adr
CA nnnn     JZ      adr
C2 nnnn     JNZ     adr
F2 nnnn     JP      adr
FA nnnn     JM      adr
EA nnnn     JPE     adr
E2 nnnn     JPO     adr
E9           PCHL
```

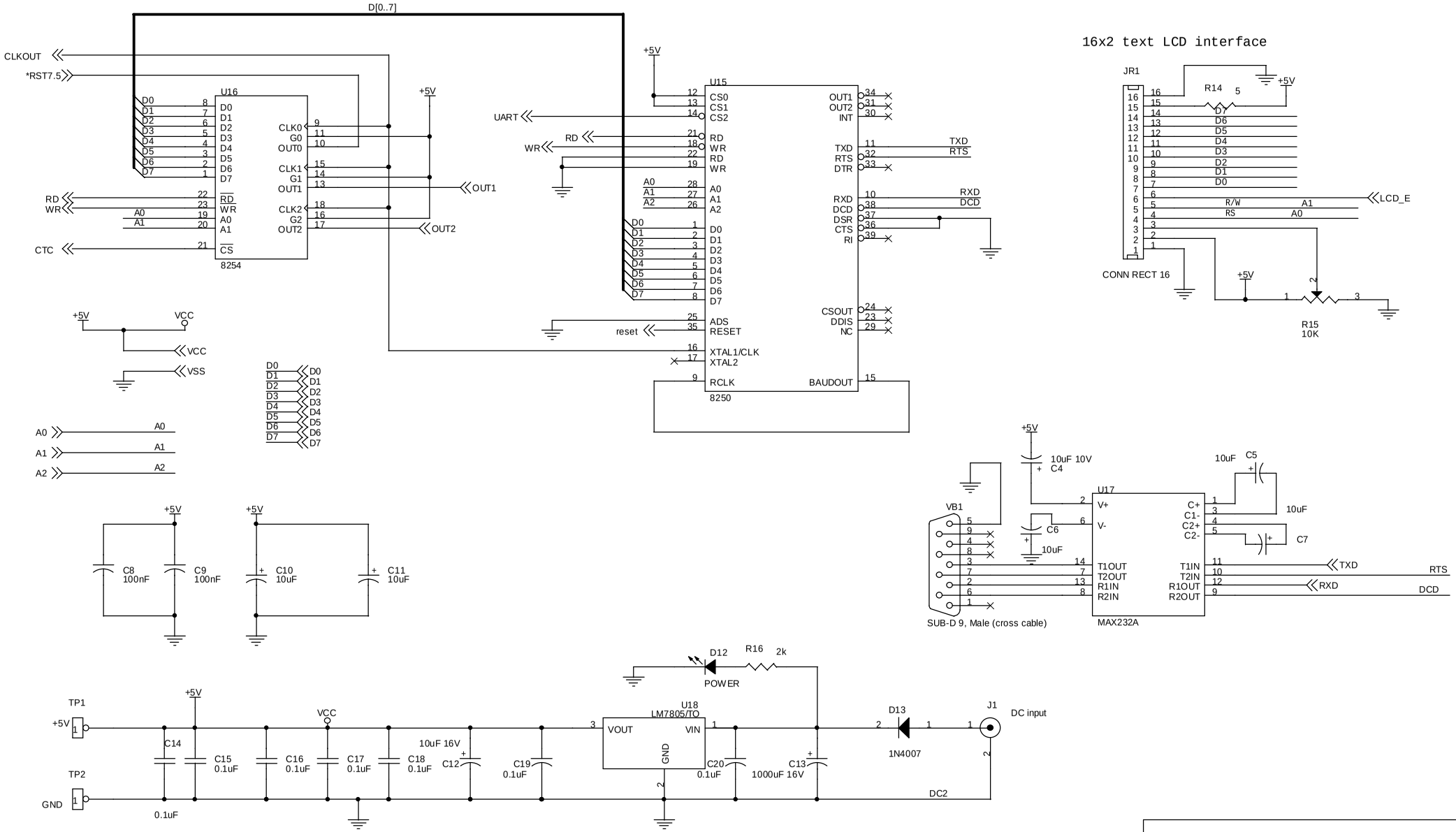
LOGICAL

```
E6 nn        ANI    byte
EE nn        XRI    byte
F6 nn        ORI    byte
A0           ANA    B
A1           ANA    C
A2           ANA    D
A3           ANA    E
A4           ANA    H
A5           ANA    L
A6           ANA    M
A7           ANA    A
A8           XRA    B
A9           XRA    C
AA           XRA    D
AB           XRA    E
AC           XRA    H
AD           XRA    L
AE           XRA    M
AF           XRA    A
B0           ORA    B
B1           ORA    C
B2           ORA    D
B3           ORA    E
B4           ORA    H
B5           ORA    L
B6           ORA    M
B7           ORA    A
```


D[0..7]
A[0..7]
0x0000-0x7FFF
32kB ROM
0x8000-0xFFFF
32kB SRAM
U1
27C256
U2
HM62256B
U3
74HC573
U4
74HC573
A[8..14]
ROM_CE
RAM_CE
/RD
/WR
GPIO1
LED 3mm
D9
1N5237A
memory & i/o decoder
U5
GAL16V8B
SYSTEM_PPI
CTC
USER_PPI
UART
LCD_E
U6
MSM80C85A
4MHz
30pF C1
30pF C2
Q1
R1 10k
S1 RESET
R2 100
C3 10uF
KIA7042
Q2
R4 10k
R5 10k
SID
TRAP
RST5.5
RST6.5
RST7.5
INTR
INTA
S0
S1
HOLD
READY
IO/M
A15
RESET_OUT
reset
CLKOUT
SOD
HLDA
WR
RD
ALE
JP1
HEADER 20X2
OUT1
OUT2
*RST5.5
*RST6.5
*RST7.5
*INTR
U7A
U7B
4001
U7C
U7D
R3 10k
R6 10k
SW1
SW DIP-4
SW2
SW SPDT
U8A
U8B
U8C
U8D
74LS14
U8E
U8F
D10
LED
R7 330
U9
74HC164
R8 10k
TRACE
VCC
VSS
Designed by Wichit Sirichote, wichit.sirichote@gmail.com
Title
8085 Microprocesor Kit
Size B
Document Number <Doc>
Rev 2
Date: Friday, April 15, 2016
Sheet 1 of 3



Title	8085 Microprocesssor Kit	
Size B	Document Number <Doc>	Rev 2
Date: Friday, April 15, 2016	Sheet 2 of 3	



Title	8085 Microprocessor Kit	
Size B	Document Number <Doc>	Rev 2
Date: Friday, April 15, 2016	Sheet 3 of 3	