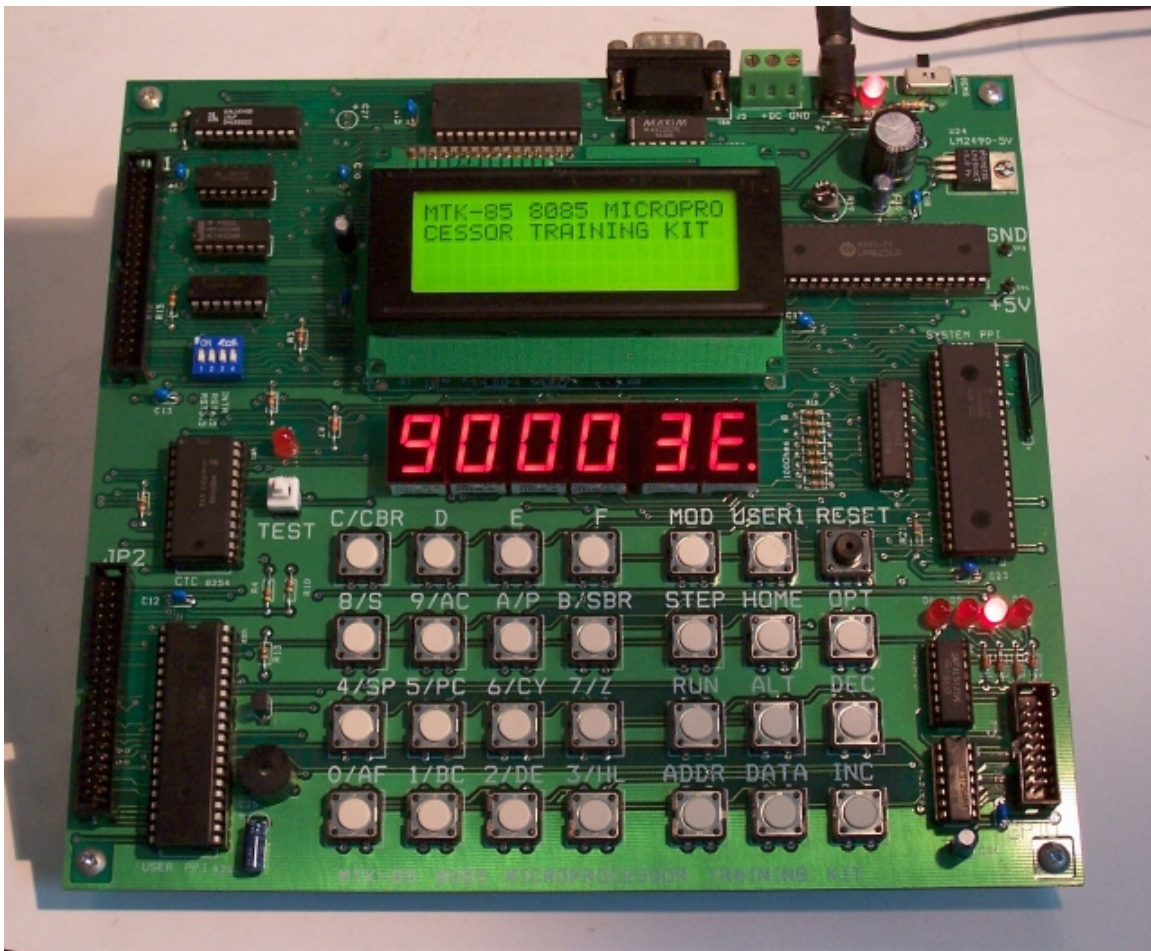


# User's Manual

## MTK-85

### 8085 Microprocessor Training Kit



Copyright © 2005 by Wichit Sirichote  
August 22, 2005

[WWW.MTK-85.COM](http://WWW.MTK-85.COM)

# Contents

Overview.....	3
Hardware Features – Software Features	
Getting Started.....	4
DC Adapter – LED Display and Keypad – RESET – ADDR – DATA – INC – DEC – HOME – ALT – RUN – STEP – MOD – Program #1 – Program #2 – Program #3	
Connecting Terminal.....	9
Command ‘A’ – Command ‘C’ – Command ‘D’ – Command ‘E’ – Command ‘F’ – Command ‘H’ – Command ‘I’ – Command ‘J’ – Command ‘K’ – Command ‘L’ – Command ‘M’ – Command ‘N’ – Command ‘Q’ – Command ‘R’ – Command ‘S’ – Command ‘W’ – Command ‘SPACE BAR’	
Hardware.....	16
CPU – Memory – GPIO – Programmable Port 8255 – Programmable Counter 8254 – Headers and Connectors – Interrupt Test Button – Technical Specifications – Monitor Call Number-NVRAM Bootable	
Appendix A	LCD Driver Routines
Appendix B	SCAN Keyboard and Display Subroutine
Appendix C	UART Driver Routines
Appendix D	Using NVRAM Bootable
Appendix E	Machine Code and Mnemonics of 8085 Instructions
Appendix F	Hardware Schematic

## Overview

The MTK-85 is a low-cost single board computer designed for self-learning the 8085 Microprocessor. The kit enables studying from low level programming with direct machine code entering to high level programming with PC tools easily. A nice feature, single-step running, helps students learn the operation of microprocessor instructions quickly and clearly. The user registers provide simple means to verify the code execution. Using a PC as the terminal, the MTK-85 can receive the Intel hex file and disassemble the machine code into 8085 instructions.

### Hardware Features:

- CPU: Mitsubishi M5M80C85AP-2 4MHz
- Memory: 32kB Monitor ROM and 32kB SRAM
- Simple I/O Port: 8-bit GPIO built with 74LS175 D-type F-F and 74LS126 Tri-state buffer
- Programmable Ports: two 8255 chips, system and user ports
- Programmable Counter: 8254
- UART: 16C550 compatible
  
- onboard I/O devices:
  - 6-digit seven segment LED,
  - 28-keypad,
  - 4-bit dot LED indicates status of GPIO,
  - magnetic buzzer,
  - Direct BUS interface 20x4 LCD (optional).
- Serial Interface: RS232C 9600bit/s 8-data bit no parity one stop bit
- +5V Power Supply: low-dropout voltage regulator with input protection
- 40-pin header for CPU bus
- 40-pin header for user port, 8255 and 8254
- 16-pin header for simple GPIO port
- onboard logic probe power supply
- Test button for single pulse generation to the interrupt pins
- Brown-out Protection
- No jumper settings

### Software Features:

- Enter the machine code in hexadecimal
- Single-step execution
- Examine and modify user registers
- Run user code with software breakpoint
- Built-in LCD drivers

- Download Intel Hex file
- Disassemble machine code into 8085 instructions
- Display user registers and disassemble the instruction after single-stepping
- NVRAM Bootable, runs user code when power up the board

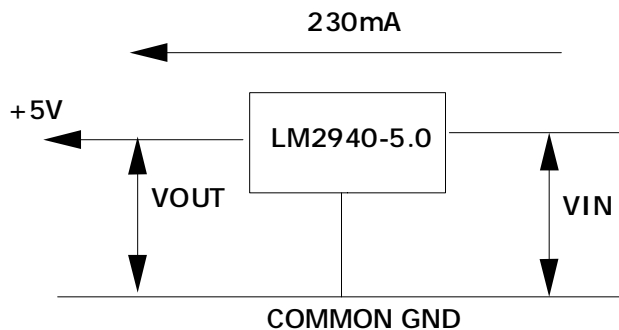
## Getting Started

### DC Adapter

The MTK-85 board requires DC power input to operate. The input voltage excepts from +6V to +12V. You may find any AC-to-DC adapter having DC jack with polarity as shown below.



Since the onboard voltage regulator is low-dropout, so dissipation of the voltage regulator will be  $(V_{in} - V_{out}) \times 230\text{mA}$ . The minimum voltage developed across  $V_{out}$  and  $V_{in}$  is 0.2V. So to provide +5VDC to the logic board, we can have  $V_{in}$  as low as say +6V. With such input, the dissipation of the voltage regulator will only be less than 250mW.



The common AC-to-DC adapters mostly have +12VDC output. You can use it also. The maximum current is 230mA at +12VDC input so we can use any adapters that provide at least 300mA.

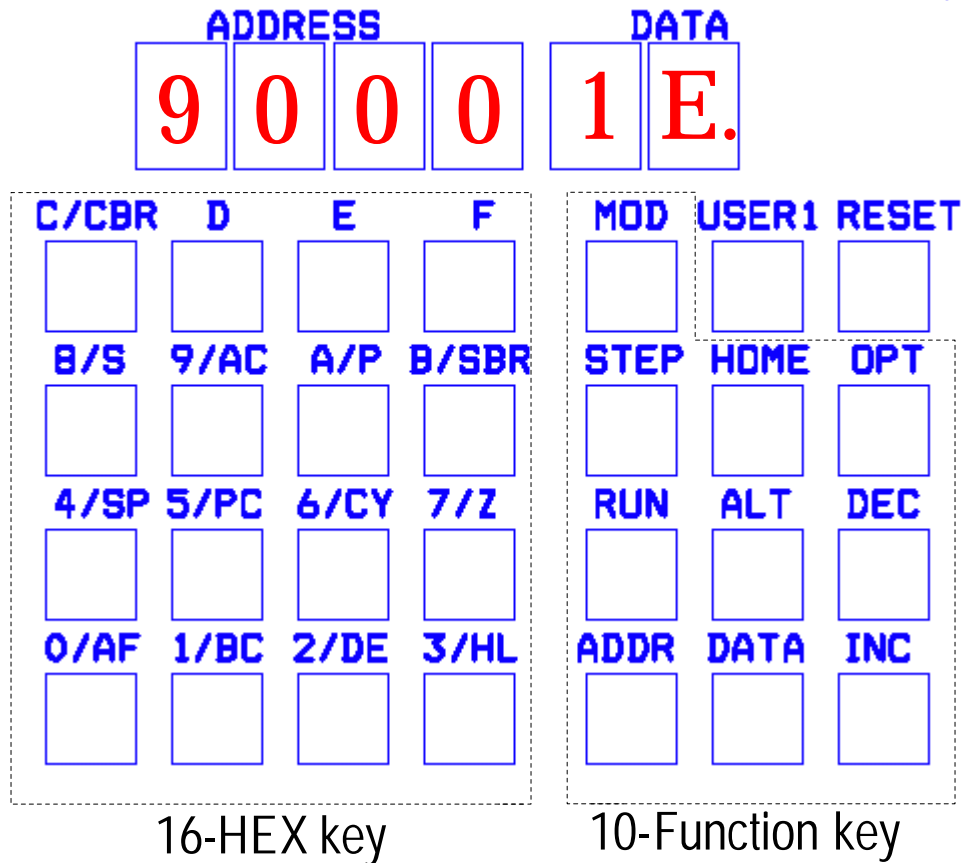
When power up the board, the cold message running text 8085 will show on 7-segment LED and the onboard dot LED will turn on.

## LED Display and Keypad

The MTK-85 has 6 digits 7-segment LED and 28 tact switches keypad.

Four digits labeled with text “ADDRESS” is for displaying the memory address and user registers contents. Two digits labeled with text “DATA” is for displaying the 8-bit data byte at address shown in the left-hand. The dot that shows 1E. indicating current mode is data entry. Typing Hex key will enter into data field.

The onboard keypad has two groups: the left-hand is 16-hex key and the right-hand is 10-function key. The hex key has alternate functions when used with ALT key.



Let's see the functions key as follows.

**RESET**

RESET is hardware reset. We can press reset to force the CPU begins execution the ROM monitor at address 0000H. The reset out signal which is active high also supplied to the UART and two 8255 PPI.

**ADDR**

ADDR changes current mode to ADDRESS entry mode. The dot indicator will move to digit4.

**DATA**

DATA changes current mode to DATA entry mode. The dot indicator will move to digit6.

**INC**

INC increments current address by one. The content of new address will show in data field LED.

**DEC**

DEC decrements current address by one. The content of new address will show in data field LED.

**HOME**

HOME brings home address back to current display. The home address is 8100H.

**ALT**

ALT enables alternate functions that used with HEX key. Below is the ALT function with HEX key.

0/AF displays user register AF. The Accumulator and Flag registers.

1/BC displays user register BC.

2/DE displays user register DE.

3/HL displays user register HL.

4/SP displays user register SP.

5/PC displays user register PC.

6/CY displays CARRY flag.

7/Z displays ZERO flag.

8/S displays SIGN flag.

9/AC displays half CARRY flag.

A/P displays PARITY flag.

B/SBR sets break address.

C/CBR clears break address.

**RUN**

RUN forces CPU to jump from monitor program to user program at current address.

**STEP**

STEP executes one instruction at address shown in current display.

**MOD**

MOD modifies the user registers. It was used together with ALT 0-5.

### Program #1

Let's learn how the keypad helps testing your first program.

```
9000 3C          main:   inr  a
9001 D300          out  0
9003 C30090       jmp  main
```

This simple program increments the content of Accumulator, send its content to the output port at location 00 then repeat it.

We see that the machine code has only 6 bytes i.e. 3C, D3, 00, C3, 00, 90.

Now enter the code into memory address 9000H.

Step 1 Press RESET, then press key 3 and key C. The 3C byte will enter to address 9000H.

Step 2 Press INC to increment address. Then repeat step1 until 90 byte was entered to address 9005H.

You can use INC or DEC to check the code, you can modify it easily in DATA entry mode.

We will begin set the value to user Accumulator beforehand. The purpose is to clear it to zero.

Press key ALT, 0/AF, display will show content of user Accumulator and Flag register.

Press key MOD, then key 0,0,0,0. The AF will be 0000.

Press key STEP, the display will show next instruction to be executed at address 9001. We can examine the content of AF by key ALT, 0/AF. We see that now Accumulator is 01.

Press key STEP again, the 01 will send to onboard GPIO. The next instruction, JMP 9000H will be executed.

We can press key STEP and see the value has changed on GPIO LED!

Instead of execution one instruction using single step, we can run the program without stopping for each instruction. We will try with key RUN.

### **Program #2**

This program shows how to use key RUN to force CPU jump from monitor program to user program.

```
9000 1E02          main:    mvi e,2
9002 CF           rst 1
9003 C30090       jmp main
```

This program has only 6 bytes i.e., 1E, 02, CF, C3, 00, 90. Enter the code, and press key HOME, RUN.

We will see the cold message repeat running on the display. RST 1 having machine code CF is the method that used to call built-in monitor functions. Register E is monitor call number.

To stop program#2 running, we must press RESET key.

### **Program #3**

We can test the program with software breakpoint. The instruction RTS 7 having machine code FF returns control back to monitor program and save the content of CPU registers to user registers. We can check the result in user registers easily.

Here is the program that adds two BCD numbers 19H and 02H. The result will be 21H.

```
9000 3E19          mvi a,19h
9002 0602          mvi b,2
9004 80           add b
9005 27           daa
9006 FF           rst 7
```

After enter the code, you can run it with key RUN. Check the result in Accumulator with ALT 0.

In case of testing long program, the board also provides tool that helps inserting the RST 7 instruction to the specified location. This tool called set break point. Suppose we want to know the result after add b instruction. We want to set break point at location 9005. We can do by setting the address to 9005 with key ADDR 9,0,0,5. Then press ALT B, the display will show this address was set breakpoint.

Press HOME and RUN, check user AF with ALT 0, we see that after addition, the result in Accumulator is 1B. To clear this break address, press ALT C. The display will show

current address 9005. The code 27 will be restored back to address 9005. We can continue execution, press RUN, check result in AF again, we will get 21.

## Connecting Terminal

The ROM monitor contains powerful commands when using UART to connect a terminal. The UART drivers and serial commands are automatically configured when UART chip was inserted. Communication is based on 9600 bit/sec, 8 data bit, no parity and one stop bit. We can use PC running terminal emulation for VT100. The PC has HyperTerminal program that can emulate terminal VT100, say.

There is no need to switch between standalone mode and terminal mode. The monitor control code for both is working concurrently.

When press reset the prompt appears on screen.

```
MTK-85 8085 MICROPROCESSOR TRAINING KIT (? HELP)
9000>
```

Type ? for help menu listing.

```
MTK-85 8085 MICROPROCESSOR TRAINING KIT (? HELP)

A - ASCII code
C - clear watch variables
D - disassemble
E - edit memory
F - fill constant
H - hex dump
I - i/o address map
J - jump to user program
K - display user STACK
L - load Intel hex file
M - monitor call number
N - new location pointer
Q - quick home location
R - user register display
S - set value to user register
W - watch variables
SPACE BAR - single step
? - help menu

9000>
```

**Command 'A'** prints the hexadecimal code for printable ASCII characters.

**Command 'C'** clears the 16-byte watch variables. The monitor provides quick access to a 16-byte RAM for program testing. The watch variables use RAM space from F000-F00F.

```
9000>
F000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9000>
```

**Command 'D'** disassembles the machine code into 8085 instructions.

```
9000>disassemble...
9000 3E19      MVI      A,19
9002 0602      MVI      B,02
9004 80        ADD      B
9005 27        DAA
9006 FF       RST      7
9007 0D       DCR      C
9008 30       SIM
9009 FA6DFC   JM       FC6D
900C C8       RZ
900D 46       MOV      B,M
900E 49       MOV      C,C
900F 0B       DCX      B
9010 CEFD    ACI      FD
9012 89       ADC      C
9013 B6       ORA      M
9014 F64B    ORI      4B
9016>
```

**Command 'E'** examines and modify the data in memory. We can use this command to enter machine code. To view the content, uses Space key and to enter byte, press two digits. To quit just press ENTER.

```
9016>edit memory location = 9000
Enter to quit, SPACE key to view content

ADDR  DATA
9000  [3E]
9001  [19] 39
9002  [06]
9003  [02] 19
9004  [80]
9005  [27]
9006  [FF]
9016>
```

**Command 'F'** fills 8-bit constant to memory. The example shows filling byte 00 to address 9010-9020.

```
9016>Begin address = 9010 End address = 9020 Data = 00
9016>
```

**Command 'H'** dumps memory. The content of memory from current pointer 9010 to 908F will display in hexadecimal. The ASCII code for each byte will be displayed also. The dot will be displayed for nonprintable ASCII code.

```
9010>
9010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
9020 4D C2 97 CB DA DF A0 BE 9E 73 1A 34 E3 A6 83 4E M.....s.4...N
9030 97 47 81 CE C1 99 98 CB 14 ED 45 DE 35 6A 7C F1 .G.....E.5j|.
9040 F0 36 B2 69 CF 1D 90 90 70 F1 73 D8 C1 4F DF 56 .6.i....p.s..O.V
9050 A8 E2 30 84 76 AA C5 18 A7 84 C5 32 81 BF B9 03 ..0.v.....2....
9060 8A 13 8C FD 4A 82 B9 99 4E 24 33 9E EB 16 A8 0D ....J...N$3.....
9070 A9 31 CD B7 BB 4E 8D BE FF 5B 3C 8D EA 5E 4F 7F .1...N...[<..^O
9080 41 00 89 F3 54 BF EC BF E0 9F 72 CB 7D E8 34 7A A...T.....r.}.4z
9090>
```

**Command 'I'** displays onboard I/O address.

```
9090>
00H-0FH onboard 4-bit GPIO, D0-D3=output port
      D4-D7=input port

10H-13H 8255 system PPI, 10H=PORTA, 11H=PORTB, 12H=PORTC,
13H=CONTROL

20H-23H 8254 programmable counter, 20H=counter0, 21H= counter1
22H=counter2, 23H control register

30H-33H 8255 user PPI, 30H=PORTA, 31H=PORTB, 32H=PORTC,
33H=CONTROL

40H-47H C16550 UART registers
9090>
```

**Command 'J'** jumps from monitor program to user program. The example shows jump to address 9000. The user register displays results after running the code. The RST 7 returns control back to monitor program.

```
9090>jump to address [9006] = 9000
```

```
AF=5800 BC=19F4 DE=C256 HL=9504 SP=F098 PC=9007 S=0 Z=0 AC=0 P=0 CY=0
9090>
```

**Command 'K'** displays user STACK memory. The example below shows running instruction PUSH H.

We first check the user register with command 'r'. We see that TOP of STACK is F098. After pressing SPACE BAR for single step, the SP is now F096. We can see the content of STACK memory with command k. The content of HL was saved in STACK.

```
9000>press r for user register display
AF=5800 BC=19F4 DE=C256 HL=9504 SP=F098 PC=9000 S=0 Z=0 AC=0 P=0 CY=0
9000>press SPACE bar for single step
          9000 E5          PUSH      H
AF=5800 BC=19F4 DE=C256 HL=9504 SP=F096 PC=9001 S=0 Z=0 AC=0 P=0 CY=0
9000>press k for STACK display
ADDR  DATA
F096  [04]
F097  [95]
F098  [C0]
9000>
```

**Command 'L'** loads Intel Hex file to memory. The Assembler and C compiler for 8085 CPU can produce standard Intel Hex file. The hex file contains machine code represented by ASCII letters. The example below uses HyperTerminal to download the hex file. The hex file is ASCII text file. So with HyperTerminal we can go to Transfer, Send text file. Or press ALT T T, the file browser will open asking what text file to be sent. We can let it show only file with .hex extension by typing \*.hex. Then double clicks at the hex file.

The onboard dot LED will run to indicate downloading is on going. When completed, the report will show number of byte received and print checksum error. If no error it will show 0 errors.

```
9080>load Intel hex file...000005 bytes loaded 0 errors
9080>
```

**Command 'M'** shows monitor call number. Some of common subroutines can be called through RST 1 with function number preloaded in register E.

```
9080>
see input parameters in user manual

1Enn MVI E,function_number
CF   RST 1
```

```
00 - demo
01 - delay
02 - cold_boot
03 - scan
04 - cin
05 - cout
06 - put_str
07 - init_lcd
08 - lcd_ready
09 - clear_lcd
0A - goto_xy
0B - put_str_lcd
0C - put_ch_lcd
0D - demo2
```

```
9080>
```

**Command ‘N’** sets new location pointer at prompt. The example sets new pointer to E000 and press ‘d’ to disassemble.

```
9080>new location = e000
E000>disassemble...

E000 53          MOV      D,E
E001 32DE2A     STA      2ADE
E004 62          MOV      H,D
E005 25          DCR      H
E006 C9          RET
E007 1C          INR      E
E008 43          MOV      B,E
E009 CC4A05     CZ       054A
E00C 2655       MVI      H,55
E00E 67          MOV      H,A
E00F 04          INR      B
E010 F3          DI
E011 0E25       MVI      C,25
E013 54          MOV      D,H
E014 AF          XRA      A
E015 C3F0C3     JMP      C3F0

E018>
```

**Command ‘Q’** sets location pointer at prompt to 9000 and sets user PC to 9000.

```
9000>
AF=5800 BC=19F4 DE=C256 HL=9504 SP=F096 PC=9000 S=0 Z=0 AC=0 P=0 CY=0
9000>
```

**Command 'R'** displays user registers content.

```
9000>
AF=5800 BC=19F4 DE=C256 HL=9504 SP=F098 PC=9000 S=0 Z=0 AC=0 P=0 CY=0
9000>
```

**Command 'S'** sets value to user registers.

```
9013>set value to user register (enter A for AF) ?
AF=0404 ff00
9013>
AF=FF00 BC=19F4 DE=0434 HL=0534 SP=F096 PC=9006 S=0 Z=0 AC=0 P=0 CY=0
9013>
```

**Command 'W'** prints watch variables.

```
9013>
F000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

**Command 'SPACEBAR'** executes the instruction at address in user PC. The instruction will show on screen with user registers result after execution.

Suppose we write a program shown below.

```
org 9000h
xra a

loop: out 0
mov h,a
inr h
push h
pop d
mov a,d
jmp loop

end
```

Then translate it to machine code file using the Assembler program. Download hex file.

```
9000>load Intel hex file...000011 bytes loaded 0 errors
9000>disassemble...
9000 AF          XRA      A
9001 D300       OUT      00
9003 67         MOV      H,A
9004 24         INR      H
```

```

9005 E5      PUSH   H
9006 D1      POP    D
9007 7A      MOV    A,D
9008 C30190  JMP    9001
900B 00      NOP
900C 00      NOP
900D 00      NOP
900E 00      NOP
900F 00      NOP
9010 00      NOP
9011 00      NOP
9012 00      NOP

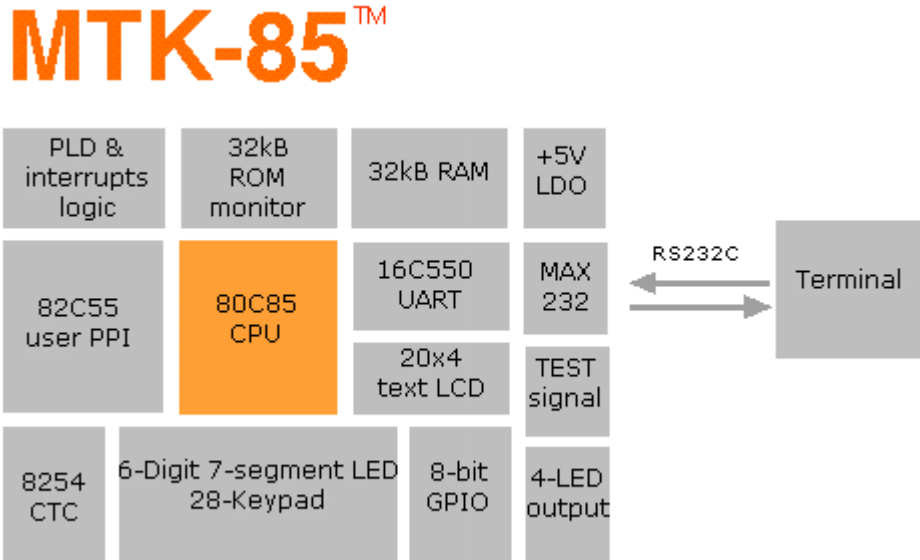
9013>print user register with command r
AF=5800 BC=19F4 DE=C256 HL=1234 SP=F098 PC=9000 S=0 Z=0 AC=0 P=0 CY=0
9013>press SPACE key to execute instruction at 9000, we see A=00
      9000 AF      XRA    A
AF=0044 BC=19F4 DE=C256 HL=1234 SP=F098 PC=9001 S=0 Z=1 AC=0 P=1 CY=0
9013>press SPACE key, the content of A will send to GPIO
      9001 D300    OUT    00
AF=0044 BC=19F4 DE=C256 HL=1234 SP=F098 PC=9003 S=0 Z=1 AC=0 P=1 CY=0
9013>press SPACE key, the content of A will copy to H
      9003 67      MOV    H,A
AF=0044 BC=19F4 DE=C256 HL=0034 SP=F098 PC=9004 S=0 Z=1 AC=0 P=1 CY=0
9013> press SPACE key, the content of H will increment by 1
      9004 24      INR    H
AF=0000 BC=19F4 DE=C256 HL=0134 SP=F098 PC=9005 S=0 Z=0 AC=0 P=0 CY=0
9013> press SPACE key, the content SP will decrement by 2
      9005 E5      PUSH   H
AF=0000 BC=19F4 DE=C256 HL=0134 SP=F096 PC=9006 S=0 Z=0 AC=0 P=0 CY=0
9013> press K, to see the content of STACK memory
ADDR  DATA
F096  [34]
F097  [01]
F098  [C0]

9013> press SPACE key, DE will be loaded with top of STACK
      9006 D1      POP    D
AF=0000 BC=19F4 DE=0134 HL=0134 SP=F098 PC=9007 S=0 Z=0 AC=0 P=0 CY=0
9013> press SPACE key, the content of D will copy to A
      9007 7A      MOV    A,D
AF=0100 BC=19F4 DE=0134 HL=0134 SP=F098 PC=9008 S=0 Z=0 AC=0 P=0 CY=0
9013> press SPACE key, PC will be loaded with 9001
      9008 C30190  JMP    9001
AF=0100 BC=19F4 DE=0134 HL=0134 SP=F098 PC=9001 S=0 Z=0 AC=0 P=0 CY=0
9013> press SPACE key, the content of A will send to GPIO, see LED!
      9001 D300    OUT    00
AF=0100 BC=19F4 DE=0134 HL=0134 SP=F098 PC=9003 S=0 Z=0 AC=0 P=0 CY=0

```

## Hardware

A block diagram of the MTK-85 board is shown below. For complete hardware schematic, see Appendix D.



### CPU

The CPU is CMOS 80C85. The XTAL frequency is 4MHz. The reset signal is generated by simple RC circuit. The CPU is protected by brownout circuit. In case of power supply is dipped caused by AC supply voltage dropped. The brownout circuit detects VCC, if it is below threshold level, it will reset the CPU.

The brownout condition can emulate by using variable power supply. To test it, adjust the board VIN from 0-12V slowly and see the CPU can start running properly or not.

### Memory

The board has 64kB memory. The 32kB ROM monitor 27C256 is placed at address 0000-7FFFH. And the 32kB SRAM 62256, is placed at address 8000H-FFFFH.

Some of interrupt vectors are relocated to RAM, so user can write the jump instruction to the location of such interrupt service routine easily. Here is the location of interrupts.

8010H	RST 2
8018H	RST 3
8020H	RST 4
8028H	RST 5
802CH	RST 5.5

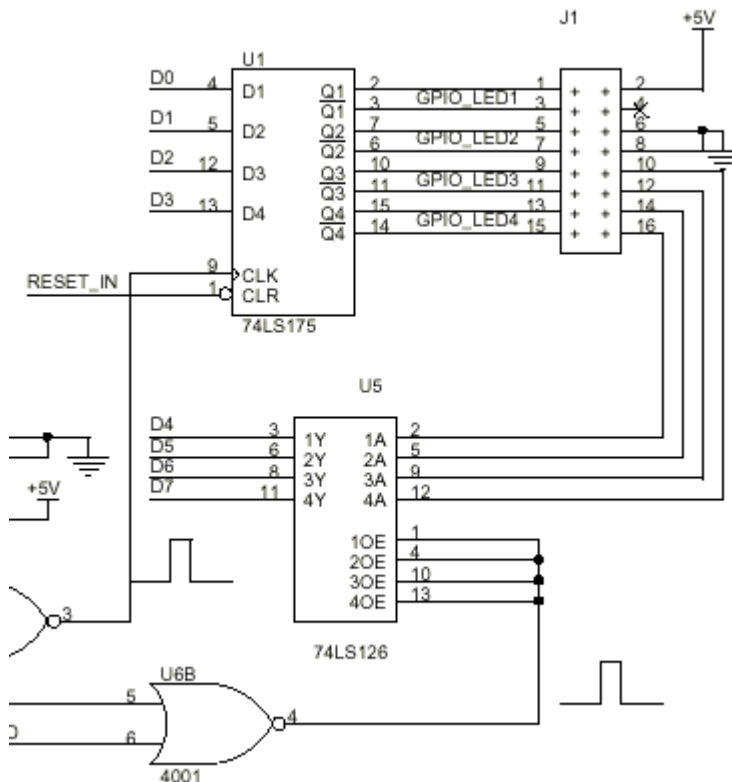
8030H	RST 6
8034H	RST 6.5
803CH	RST 7.5

Note:

1. RST 7 is used for software breakpoint.
2. RST 1 is used for monitor function call.
3. TRAP is used for hardware single-step.
4. RST 7.5 is tied to OUT0 of 8254 programmable counter.
5. Monitor program uses last page of RAM for data storage, STACK area, and monitor control functions. The space is from F000H to F098H.

## GPIO

GPIO provides 4-bit output port using D type F-F, 74LS175 and 4-bit input port using tri-state buffer, 74LS126. The address is 00 for both ports. The low-nibble D0-D3 is output port. The higher-nibble D4-D7 is input port. The signal from both ports appeared at J1, 16-pin header.



## System Programmable Port 8255

The I/O addresses of system port, 8255 are PORTA=10H, PORTB=11H, PORTC=12H and Control Port = 13H. Buzzer control pin is PORT C bit 7. To enable buzzer, write 7FH to PORTC.

## User Programmable Port 8255

The board provides a user programmable parallel port, 8255. The 8255's registers are mapped to I/O address from 30H to 33H.

30H	PORTA
31H	PORTB
32H	PORTC
33H	CONTROL PORT

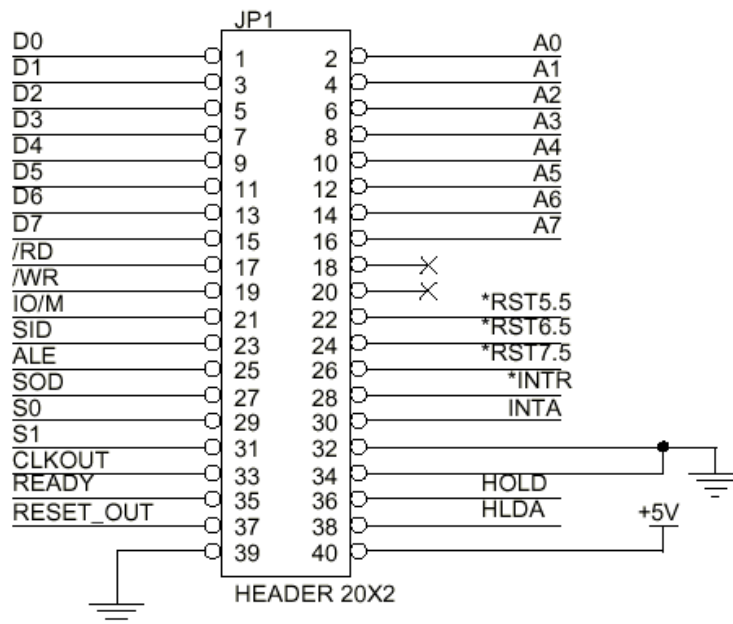
## Programmable Counter 8254

The programmable counter, 8254 was supplied with clock signal from CLOCKOUT or 2MHz for counter0 and counter1. The internal registers of 8254 are mapped to I/O space from 20H to 23H.

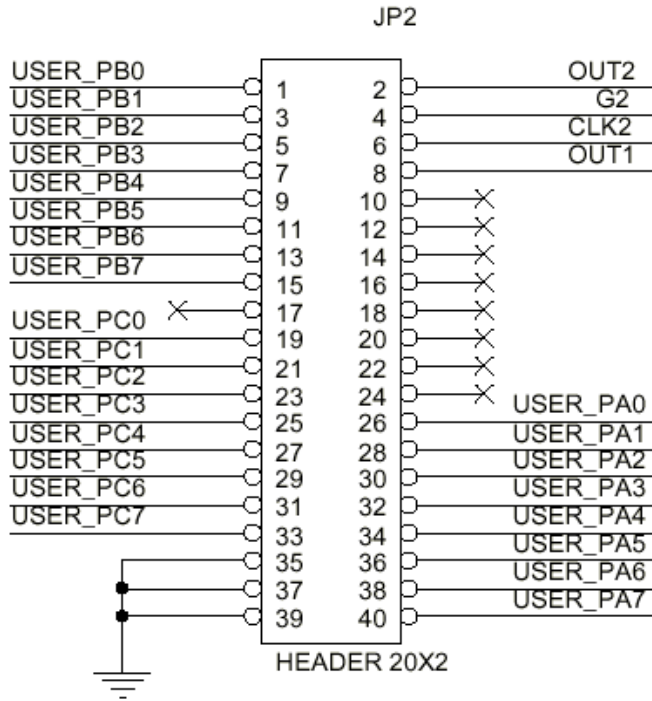
20H	COUNTER0
21H	COUNTER1
22H	COUNTER2
23H	CONTROL REGISTER

## Headers and Connectors

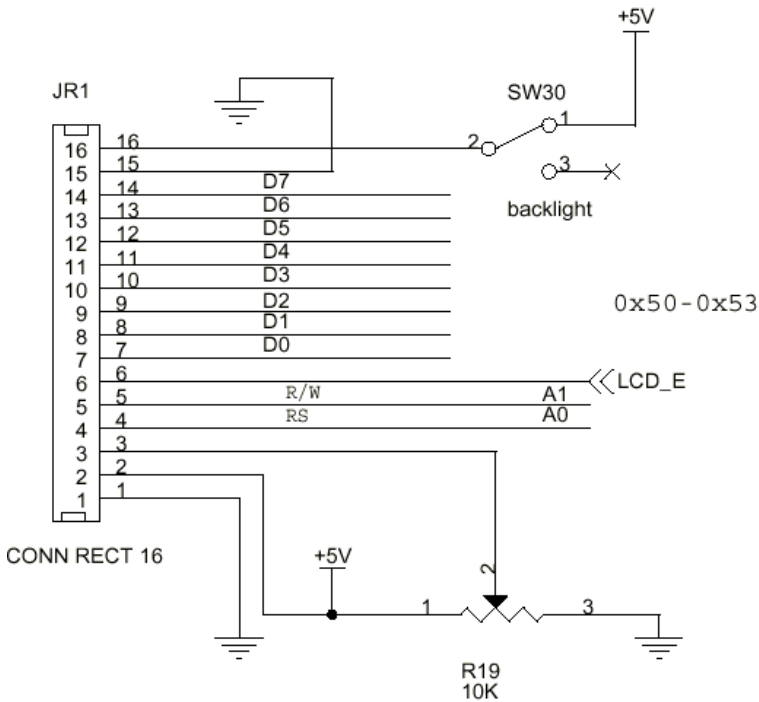
### CPU Header JP1



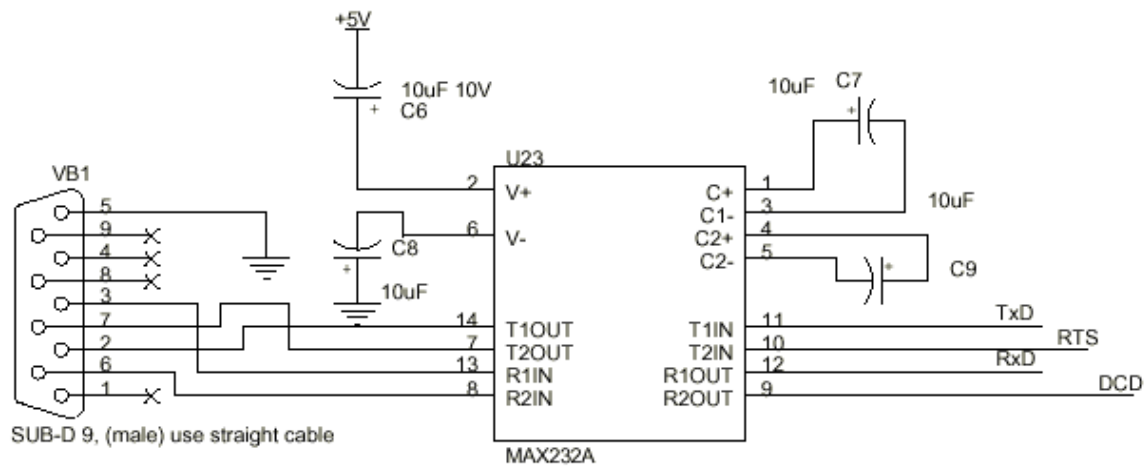
## User 8255 and 8254 Header JP1



## Onboard LCD Header JR1



## RS232C DB9 male connector VB1



## Interrupts Test Button

The interrupt test button provides a single positive pulse that tied to CPU hardware interrupt pins, RST5.5, RST6.5 and INTR. User can select the pulse to be triggered for each pin by dip switch SW1. The onboard LED, D4 indicates the pulse is activated when press Test button.

## Technical Specifications

CPU: CMOS 80C85 @4MHz

Memory: 64kB, 32kB 27C256, 32kB 62256

I/O port: Programmable Parallel port 8255x2, 8-bit GPIO

Counter: Programmable Counter 8254

UART: 16-byte FIFO TL16C550 compatible

ROM monitor: 8085.asm

Brownout Protection: KIA7045

Board Size: 8.90 x 8.13 inch

Weight: 320g (complete components except LCD)

DC Power Supply: AC-to-DC adapter 6V-12V 250mA

Power consumption: 2.76W (230mA @12VDC)

## Monitor Call Number

00 - demo

Scan 7-segment display with buffer display pointed by HL

Entry: HL

Exit: none

01 - delay  
Delay subroutine using register pair DE, D is outer loop delay, E is inner loop.  
Entry: DE  
Exit: none

02 - cold\_boot  
Display cold-boot message on 7-segment LED.  
Entry: none

03 - scan  
Scan keyboard and display one cycle.  
Entry: HL points the display buffer  
Exit: key = scan code -1 no key pressed

04 - cin  
Get character from console  
Entry: none  
Exit: A = character received

05 - cout  
Send character to console  
Entry: A = character to be sent  
Exit: none

06 - put\_str  
Print string to console, string is terminated by 0.  
Entry: HL  
Exit: none

07 - init\_lcd  
Initialize LCD module  
Entry: none  
Exit: none

08 - lcd\_ready  
Wait until LCD module is ready.  
Entry: none  
Exit: none

09 - clear\_lcd  
Clear LCD display  
Entry: none  
Exit: none

0A - goto\_xy  
Set cursor position of LCD  
Entry: HL, H = x, L = y

Exit: none

0B - put\_str\_lcd

Print string to LCD, string is terminated by 0

Entry: HL

Exit: none

0C - put\_ch\_lcd

Print character to LCD at current cursor position

Entry: A

Exit: none

0D - demo2

Running GPIO LED

Entry: none

Exit: none

## NVRAM Bootable

User can replace U3, SRAM with a Nonvolatile RAM for program storage, when the board is power off. A JMP instruction placed at 8000H will enable NVRAM bootable. The monitor program checks the location 8000H. If it has C3 (opcode of JMP instruction), it will jump to address 8000H. The feature allows application code to run easily. The monitor subroutines are still available for the application program.

To get back to monitor mode, user can press USER1 key while press RESET. The byte C3 at location 8000H will change to 00.

The sample code that demonstrates NVRAM Bootable is shown in Appendix D.

## Appendix A Onboard LCD Driver Routines

```
;----- onboard LCD registers -----
command_write equ 50h
command_read  equ 52h
data_write    equ 51h
data_read     equ 53h
busy          equ 80h

;----- LCD driver routines -----
lcd_ready:   push psw

lcd_ready1:  in command_read
            ani busy
            jnz lcd_ready1  ; wait until lcd ready
            pop psw

            ret

clear_lcd:   call lcd_ready
            mvi a,1
            out command_write

exit_clear:  ret

init_lcd:   call lcd_ready
            mvi a,38h
            out command_write
            call lcd_ready
            mvi a, 0ch
            out command_write
            call clear_lcd

            ret

; print ASCII text on LCD
; entry: HL pointer with 0 for end of string

put_str_lcd: mov a,m      ; get A from [HL]
            cpi 0
            jnz put_str_lcd1
            ret

put_str_lcd1:

            call lcd_ready
            out data_write
            inx h
            jp put_str_lcd
```

```
; goto_xy set cursor location on lcd
; entry: HL: H = x, L = y
```

```
goto_xy:  call lcd_ready
          mov a,l
          cpi 0
          jnz goto_xy1
          mov a,h
          adi 80h
          out command_write
          ret
```

```
goto_xy1: cpi 1
          jnz goto_xy2
          mov a,h
          adi 0c0h
          out command_write
          ret
```

```
goto_xy2: cpi 2
          jnz goto_xy3
          mov a,h
          adi 094h
          out command_write
          ret
```

```
goto_xy3: cpi 3
          jnz goto_xy4
          mov a,h
          adi 0d4h
          out command_write
          ret
```

```
goto_xy4: ret
```

```
; put_ch_lcd put character to lcd
; entry: A
```

```
put_ch_lcd: call lcd_ready
            out data_write
            ret
```

## Appendix B Subroutine Scan keyboard and Display

```
; subroutine scan keyboard and display
; entry: hl pointer to display buffer
; exit: key = scan code
;         -1 no key pressed
;
;
scan:   push h
        push b
        push d

        mvi c,6      ; for 6-digit LED
        mvi e,0      ; digit scan code appears at 4-to-10
                    ; decoder
        mvi d,0      ; key position
        mvi a,0ffh  ; put -1 to key
        sta key      ; key = -1

scan1:  mov a,e
        ori 0f0h    ; high nibble must be 1111
        out system_port_c ; active digit first
        mov a,m      ; load a with [hl]
        out system_port_b ; then turn segment on

        mvi b,0      ; delay for electron transition process
wait1:  dcr b
        jnz wait1

        in  system_port_a ; read input port

        mvi b,8      ; check all 8-row
shift_key: rar          ; rotate right through carry
        jc  next_key  ; if carry = 1 then no key
                    ; pressed

        push psw
        mov a,d
        sta key      ; save key position
        pop psw

next_key:
        inr d        ; next key position

        dcr b        ; until 8-bit was shifted
        jnz shift_key

        mvi a,0      ; clear a
        out system_port_b ; turn off led
```

```
    inr e           ; next digit scan code
    inx h           ; next location

    dcr c           ; next column
    jnz scan1

    pop d
    pop b
    pop h
    ret
```

```
;----- 8255 PPI system port I/O address -----
system_port_a:  equ 10h
system_port_b:  equ 11h
system_port_c:  equ 12h
system_port_control: equ 13h
```

## Appendix C UART Driver Routines

```
;----- 16C550 compatible UART I/O address -----
; e.g. UM8250B, 16C450, 16C550

uart_buffer: equ 40h
uart_line_status: equ 45h
uart_fifo:      equ 42h
uart_lcr:       equ 43h
uart_divisor_lsb: equ 40h
uart_divisor_msb: equ 41h
uart_scr:       equ 47h

; initialize 16C550 uart to 9600 8n1 with 2MHz clock
; 2MHz/13 = 153846Hz

init_uart:

    mvi a,83h
    out uart_lcr      ; set DLAB bit to access divider

    mvi a,13
    out uart_divisor_lsb
    mvi a,0
    out uart_divisor_msb ; 2MHz/13 = 153846 Hz
                        ; 153846Hz/16 = 9615Hz

    mvi a,7
    out uart_fifo      ; init fifo and clear all buffers
    mvi a,03h
    out uart_lcr      ; clar DLAB

; check uart line status, if the byte is FF then no uart
;
;
    xra a
    out uart_scr      ; check if there is uart
    in uart_scr
    cpi 0
    jz found
    xra a
    sta uart_found
    ret

found    mvi a,1
        sta uart_found
        ret

cout:    mov b,a          ; save a

cout1:   in uart_line_status
```

```

        ani 20h                ; transmitter ready?
        jz cout1

        mov a,b                ; restore a
        out uart_buffer
        ret

cin:    in uart_line_status
        ani 1                  ; data available?
        jz  cin
        in uart_buffer
        ret

; print string terminated by 0
; input: HL

put_str: mov a,m              ; get A from [HL]
         cpi 0
         jnz put_str1
         ret

put_str1: call cout
         inx h
         jp  put_str

```

## Appendix D Using NVRAM Bootable

```
; MTK-85 8085 Microprocessor Training Kit
; expl.asm
;
; Using 8254 to produce 30.52Hz interrupt signal at RST7.5
;
; The 8254 counter0 was loaded with 0000 by system monitor.
; The input clock to the 8254 is 2MHz, the OUT0 then
; produces
; 2MHz/65536 = 30.52Hz interrupt at RST7.5!
;
; CPU "8085.TBL" ;CPU TABLE
; HOF "INT8" ;HEX FORMAT
gpio equ 0

; enable NVRAM boot running

org 8000h
jmp start ; put instruction JMP to boot from
RAM

org 803ch ; interrupt vector of RST7.5
(relocated from 003CH)
jmp service_rst7.5

org 8100h

start: mvi a,11111011b ; enable rst7.5
sim ; set interrupt mask register
ei ; enable interrupt

jmp $ ; jump here

service_rst7.5:

lda count ; increment count
inr a
sta count
out gpio ; write to onboard LED
ei
ret

org 0e000h

count dfs 1 ; use RAM one byte for count
variable

end
```

## Appendix E Machine Code and Mnemonic of 8085 Instructions

---

<b>MOVE, LOAD and STORE</b>			6E	MOV	L, M
			6F	MOV	L, A
40	MOV	B, B	70	MOV	M, B
41	MOV	B, C	71	MOV	M, C
42	MOV	B, D	72	MOV	M, D
43	MOV	B, E	73	MOV	M, E
44	MOV	B, H	74	MOV	M, H
45	MOV	B, L	75	MOV	M, L
46	MOV	B, M	77	MOV	M, A
47	MOV	B, A	78	MOV	A, B
48	MOV	C, B	79	MOV	A, C
49	MOV	C, C	7A	MOV	A, D
4A	MOV	C, D	7B	MOV	A, E
4B	MOV	C, E	7C	MOV	A, H
4C	MOV	C, H	7D	MOV	A, L
4D	MOV	C, L	7E	MOV	A, M
4E	MOV	C, M	7F	MOV	A, A
4F	MOV	C, A			
50	MOV	D, B	3E nn	MVI	A, byte
51	MOV	D, C	06 nn	MVI	B, byte
52	MOV	D, D	0E nn	MVI	C, byte
53	MOV	D, E	16 nn	MVI	D, byte
54	MOV	D, H	1E nn	MVI	E, byte
55	MOV	D, L	26 nn	MVI	H, byte
56	MOV	D, M	2E nn	MVI	L, byte
57	MOV	D, A	36 nn	MVI	M, byte
58	MOV	E, B			
59	MOV	E, C	01 nnnn	LXI	B, dble
5A	MOV	E, D	11 nnnn	LXI	D, dble
5B	MOV	E, E	21 nnnn	LXI	H, dble
5C	MOV	E, H	31 nnnn	LXI	SP, dble
5D	MOV	E, L			
5E	MOV	E, M	02	STAX	B
5F	MOV	E, A	12	STAX	D
60	MOV	H, B	0A	LDAX	B
61	MOV	H, C	1A	LDAX	D
62	MOV	H, D	32 nnnn	STA	adr
63	MOV	H, E	3A nnnn	LDA	adr
64	MOV	H, H	22 nnnn	SHLD	adr
65	MOV	H, L	2A nnnn	LHLD	adr
66	MOV	H, M	EB	XCHG	
67	MOV	H, A			
68	MOV	L, B	<b>STACK</b>		
69	MOV	L, C			
6A	MOV	L, D	C5	PUSH	B
6B	MOV	L, E	D5	PUSH	D
6C	MOV	L, H	E5	PUSH	H
6D	MOV	L, L	F5	PUSH	PSW

C1	POP	B	09	DAD	B
D1	POP	D	19	DAD	D
E1	POP	H	29	DAD	H
F1	POP	PSW	39	DAD	SP
E3	XTHL				
F9	SPHL				
33	INX	SP			
3B	DCX	SP			

**ARITHMETICS**

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
8B	ADC	E
8C	ADC	H
8D	ADC	L
8E	ADC	M
8F	ADC	A

D6 nn	SUI	byte
DE nn	SBI	byte

90	SUB	B
91	SUB	C
92	SUB	D
93	SUB	E
94	SUB	H
95	SUB	L
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

**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

**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

F7	RST	6
FF	RST	7

### INPUT/OUTPUT

DB nn	IN	byte
D3 nn	OUT	byte

### 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	

### 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

### 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

### SPECIALS

2F	CMA
37	STC
3F	CMC
27	DAA

### RESTART

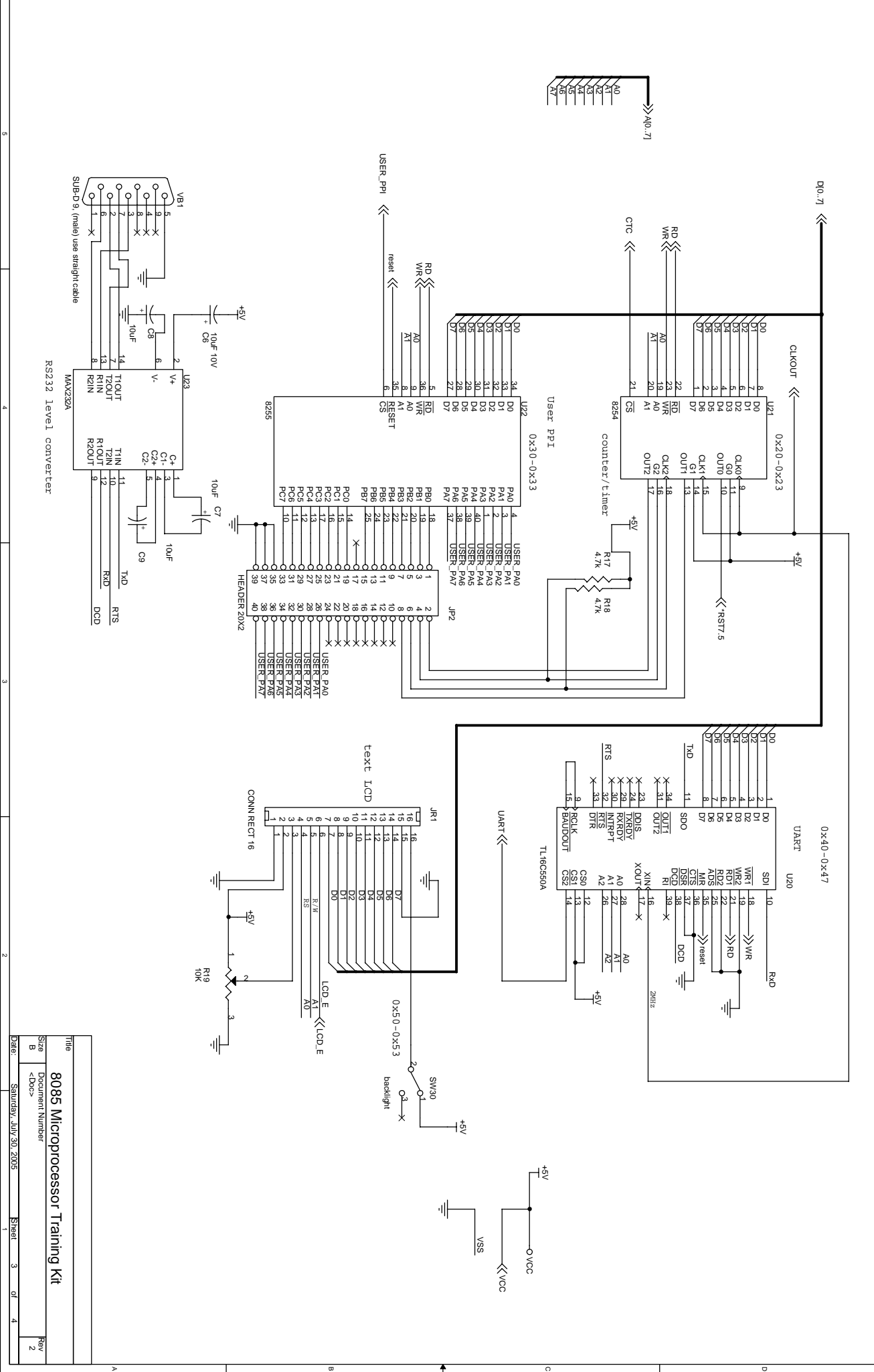
C7	RST	0
CF	RST	1
D7	RST	2
DF	RST	3
E7	RST	4
EF	RST	5

### CONTROL

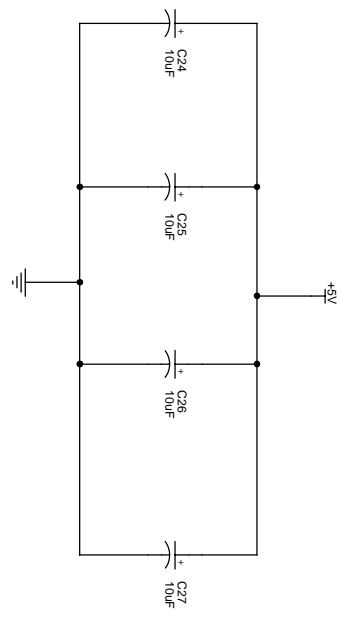
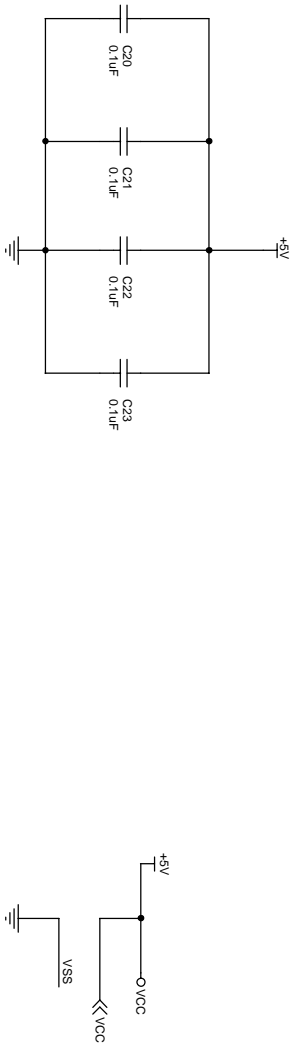
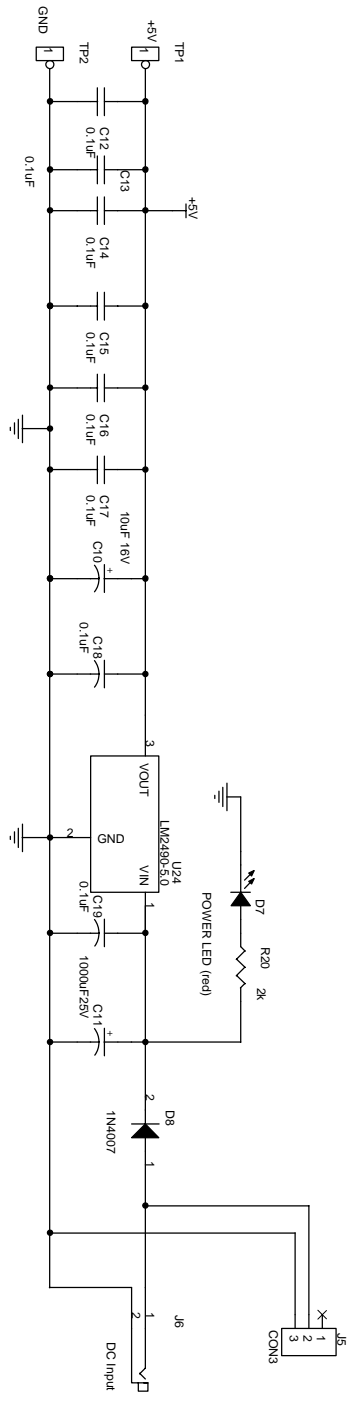
00	NOP
F3	DI
FB	EI
76	HLT
20	RIM
30	SIM







Title	8085 Microprocessor Training Kit
Size	Document Number
B	<Doc>
Date	Saturday, July 30, 2005
Sheet	3 of 4
Rev	2



Title		8085 Microprocessor Training Kit	
Size	Document Number	Rev	
B	<Doc>		
Date:	Saturday, July 30, 2005	Sheet	4 of 4