

Rev 1.0, January 2022

Internet: <https://kswichit.net/8086/8086kit.htm>

8086 MICROPROCESSOR KIT

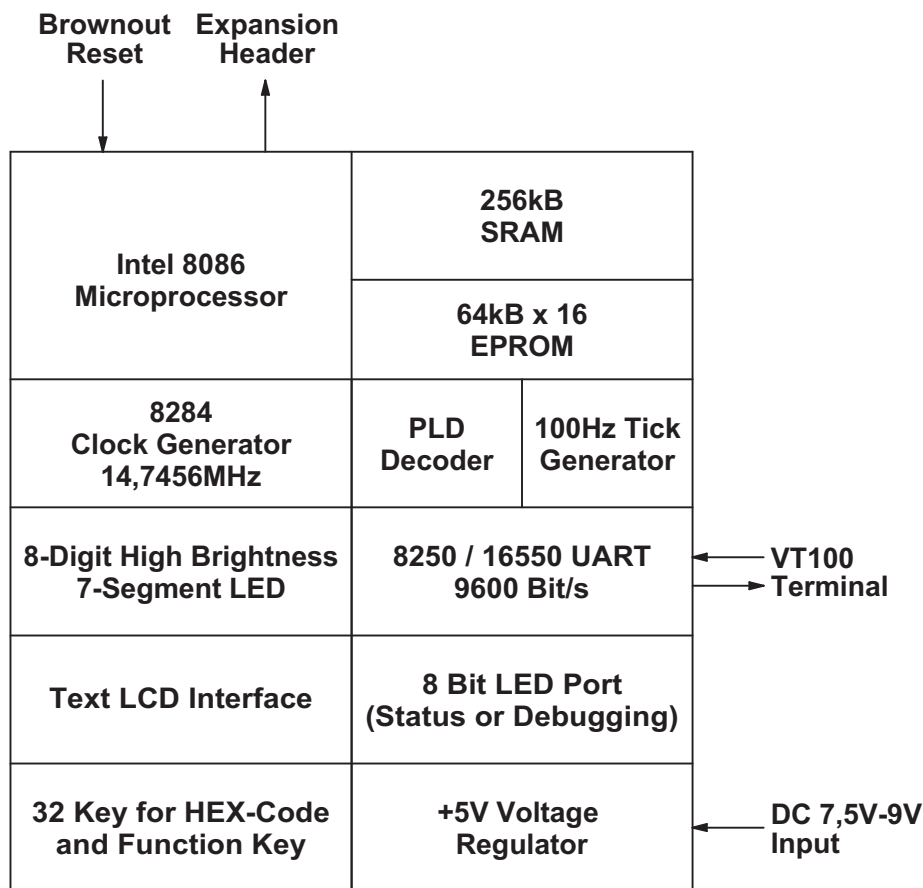
CONTENTS

OVERVIEW	1
FUNCTIONAL BLOCK DIAGRAM	1
HARDWARE AND MONITOR PROGRAM FEATURES	2
HARDWARE LAYOUT	3
KEYBOARD LAYOUT	4
MONITOR FUNCTION KEYS	4
MEMORY AND I/O MAP	5
PLD DECODER	5
GPIO1 LED	7
EXPANSION BUS HEADER	8
CONNECTING LCD MODULE	8
SPEAKER PORT	9
INTERRUPT TICK GENERATOR	9
DC POWER SUPPLY	11
POWER UP	11
DISPLAY USER REGISTERS	11
PROGRAM EXAMPLE 1: WRITE TO GPIO1 LED	12
PROGRAM EXAMPLE 2: RUNNING LIGHT ON GPIO1 LED	13
PROGRAM EXAMPLE 3: INTERRUPT	14
PROGRAM EXAMPLE 4: SPEAKER	15
RS232C PORT	15
CONNECTING 8086 KIT TO TERMINAL	15
MONITOR COMMANDS	17
LOGIC PROBE POWER SUPPLY	18
HARDWARE SCHEMATIC	19
PARTS LIST	23
MONITOR PROGRAM LISTING	25

OVERVIEW

The 8086 Microprocessor Kit is a single board computer designed for self-learning the 16-bit microprocessor operations. The CPU is 8086 with 16-bit memory interface. The monitor ROM provides functions for entering 8086 instruction code using onboard hex key, testing code running, single stepping and display user registers. Kit has 10ms tick generator for testing external interrupt operation. The kit also has RS232 port for 9600 terminal interface. The hex file generated from assembler can be download easily.

FUNCTIONAL BLOCK DIAGRAM



Notes

1. UART is 9600 bit/s controlled by the 8250 / 16550 asynchronous communication chip.
2. The kit has 8-bit LCD module interfacing bus.
3. 100Hz tick generator is for interrupt experiment.
4. Ports for display and keypad interfacing were built with discrete logic IC chips.
5. Memory and port decoders are made with Programmable Logic Device, PLD.

HARDWARE AND MONITOR PROGRAM FEATURES

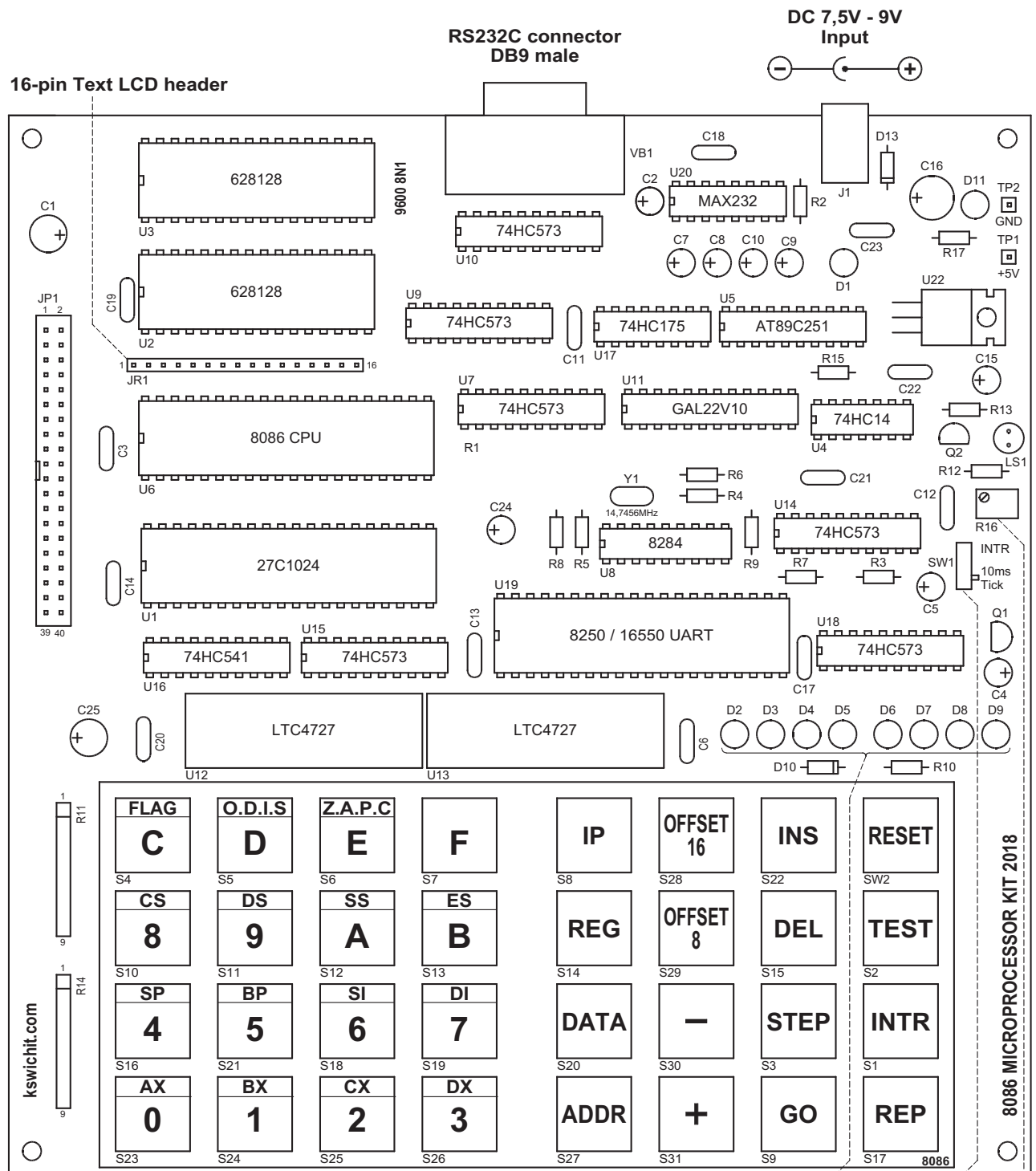
Hardware features

- CPU: 8086 16-bit microprocessor x86 compatible
- Oscillator: 8284, clock generator XTAL 14.7456MHz (4.9152MHz processor clock)
- Memory: 256kB RAM, 256kB EPROM
- Memory and I/O decoder chip: Programmable Logic Device GAL22V10
- Display: high brightness 6-digit 7-segment LED
- Keyboard: 32 keys
- RS232 port: 8250 / 16550 UART
- Debugging LED: 8-bit GPIO1 LED at location 00H
- Tick: 10ms tick produced by 89C2051 for time trigger experiment
- Text LCD interface: direct CPU bus interface, HD44780 compatible
- Brownout reset: KIA7042 reset chip for power brownout reset
- Power consumption: 500mA @7.5V - 9.0VDC AC adapter
- Expansion header: 40-pin header
- Mechanical dimension: 178x184 mm

Monitor program features

- Enter 8086 instructions using hex code directly
- Test code running with single step or break point
- User registers for status capturing
- Insert/delete byte
- 8/16 bit offset calculation
- Intel hex file downloading using RS232 port, 9600 bit/s 8n1

HARDWARE LAYOUT



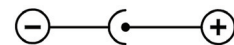
GPIO Led 8-bit binary display (address is 00H)

**Selector for 10ms
Tick or INTR key**

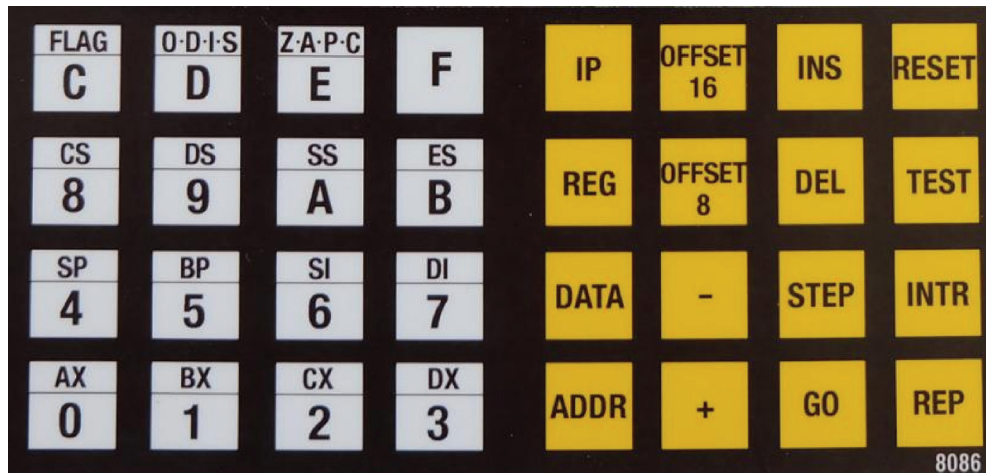
LCD contrast

Important Notes

1. Insert or remove the LCD module must be done when the kit is powered off!
2. AC adapter should provide 7.5 - +9VDC, higher voltage will cause the voltage regulator chip becomes hot. (+7.5V is recommended)
3. Under normal operation, the voltage regulator will be HOT on the heatsink



KEYBOARD LAYOUT



HEX keys Hexadecimal number 0 to F with associated user registers, AX, BX, CX, DX, SP, BP, SI, DI, CS, DS, SS, ES and flag displays.

MONITOR FUNCTION KEYS

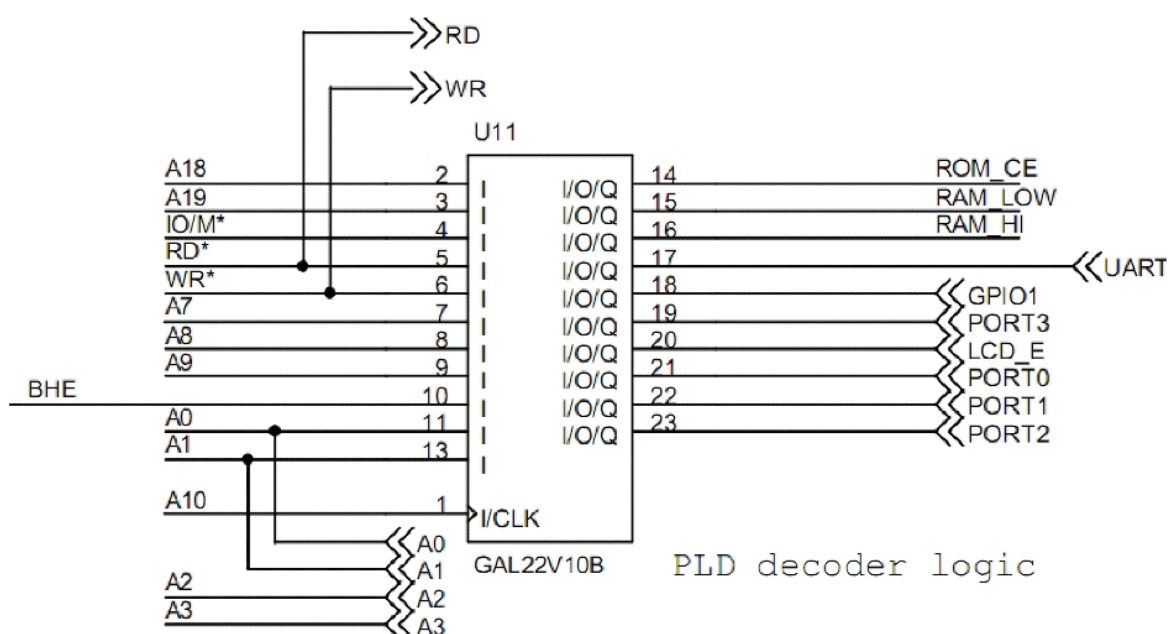
Key	Description
RESET	Reset CPU and start the monitor program.
TEST	Test the 10ms tick interrupt and speaker. SW1 must be 10ms position.
INTR	External interrupt signal. Interrupt vectors must be entered by user.
REP	Repeat function keys when press together.
REG	Set display user registers. Use with hex key for a given user register. Key D displays flag O, D, I, S (Overflow, Direction, Interrupt, Sign). Key E displays flag Z, A, P, C (Zero, Aux, Parity, Carry).
IP	Set display address to 400 after reset, set to current IP after service break point.
INS	Insert one byte and shift 512 bytes down.
DEL	Delete one byte and shift 512 bytes up.
STEP	Single step running, CPU registers will be saved to user registers.
GO	Jump from monitor program to user code.
OFFSET8	Compute 8-bit offset byte, used with key + and GO. Enter start address, destination with key +, key, GO will compute offset and write the offset byte.
OFFSET16	Compute 16-bit offset word, used with key + and GO. Enter start address, destination with +, key, GO will compute offset and write the offset word.

MEMORY AND I/O MAP

The kit provides 256kB RAM from location 00000H-3FFFFH and Monitor ROM from E0000H-FFFFFH. The I/O ports are located from 000H to 5FFH.

Address	Unit	Description	Device	Dev.No.
00000H - 3FFFFH	2 x 128kB RAM	static RAM	628128	U2, U3
E0000H - FFFFFH	128kB EPROM	monitor EPROM 64k x 16-bit	27C1024	U1
000H	GPIO1 (out)	8-bit LED Port	74HC573	U18
100H	PORT0 (in)	keypad grid input lines	74HC541	U16
180H	PORT1 (out)	keypad grid output lines 7 segment LED: common cathodes	74HC573	U15
200H	PORT2 (out)	7 segment LED: segments A .. DP	74HC573	U14
280H	PORT3 (out)	speaker (binary control with bit D3)	74HC175	U17
300H	UART base addr.	UART for RS232C port	8250 / 16550	U19
400H	USER PORT base address	user expansion port, 16 data-bits (D0-D15), 10 address bits (A0-A9)	Header 20x2	JP1
500H - 506H	LCD registers	LCD command write LCD command read LCD data write LCD data read	Any text LCD with HD44780 compatible controller	

PLD DECODER



Below PLD file is compiled with WinCupL version 5.30.4.

```

Name 8086KIT ;
PartNo 22V10 ;
Date 21/4/2561 ;
Revision 01 ;
Designer Engineer ;
Company home ;
Assembly None ;
Location ;
Device P22V10 ;

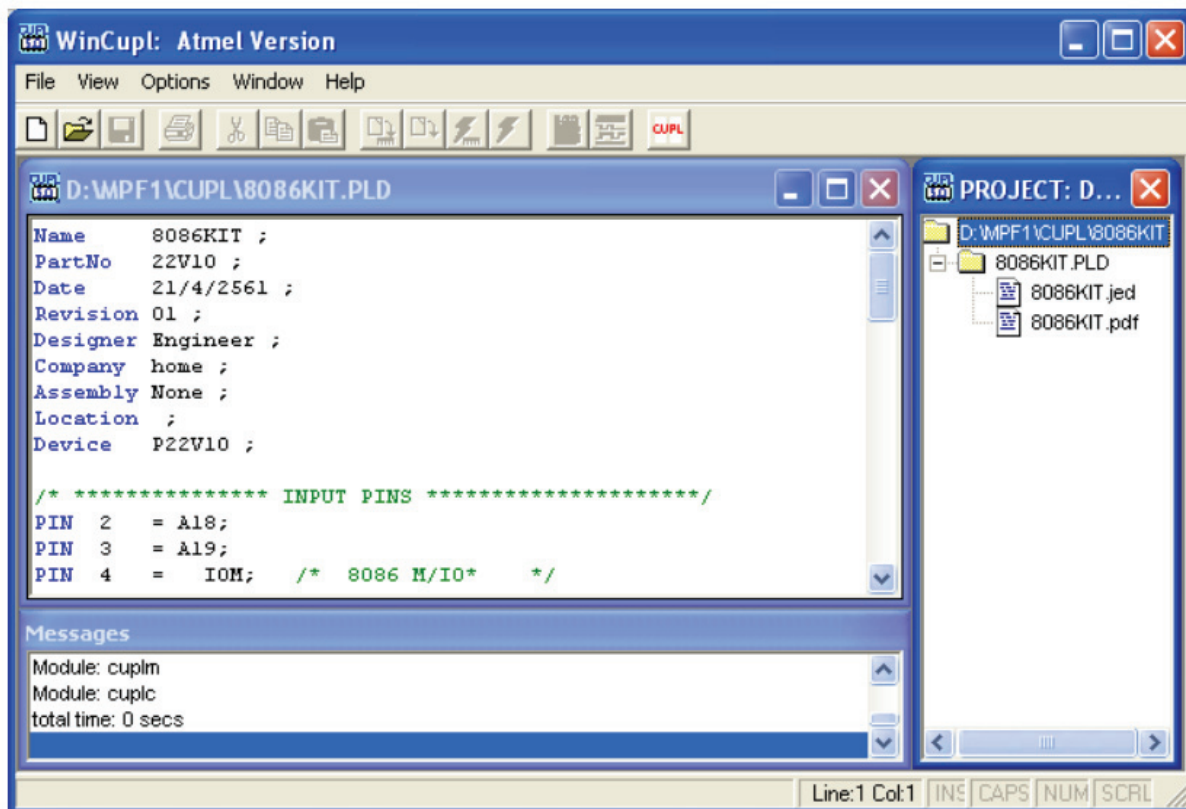
/* ***** INPUT PINS ***** */
PIN 2 = A18;
PIN 3 = A19;
PIN 4 = IOM; /* 8086 M/IO* */
PIN 5 = RD;
PIN 6 = WR;
PIN 7 = A7;
PIN 8 = A8;
PIN 9 = A9;
PIN 10 = BHE;
PIN 11 = A0;
PIN 13 = A1;
PIN 1 = A10;

/* ***** OUTPUT PINS ***** */
PIN 14 = ROMCE;
PIN 15 = RAM_LOW;
PIN 16 = RAM_HI;
PIN 17 = UART;
PIN 18 = GPIO1;
PIN 19 = PORT3;
PIN 20 = LCD_E;
PIN 21 = PORT0;
PIN 22 = PORT1;
PIN 23 = PORT2;

ROMCE    = !IOM # !A18 # !A19;
RAM_LOW  = !IOM # A18 # A19 # A0;
RAM_HI   = !IOM # A18 # A19 # BHE;
!GPIO1   = WR # IOM # A10 # A9 # A8 # A7 # A0;
PORT0    = RD # IOM # A10 # A9 # !A8 # A7 # A0;
!PORT1   = WR # IOM # A10 # A9 # !A8 # !A7 # A0;
!PORT2   = WR # IOM # A10 # !A9 # A8 # A7 # A0;
PORT3    = WR # IOM # A10 # !A9 # A8 # !A7 # A0;
UART     = IOM # A10 # !A9 # !A8 # A7 # A0;
!LCD_E   = (RD & WR) # IOM # !A10 # A9 # !A8 # A7;

```

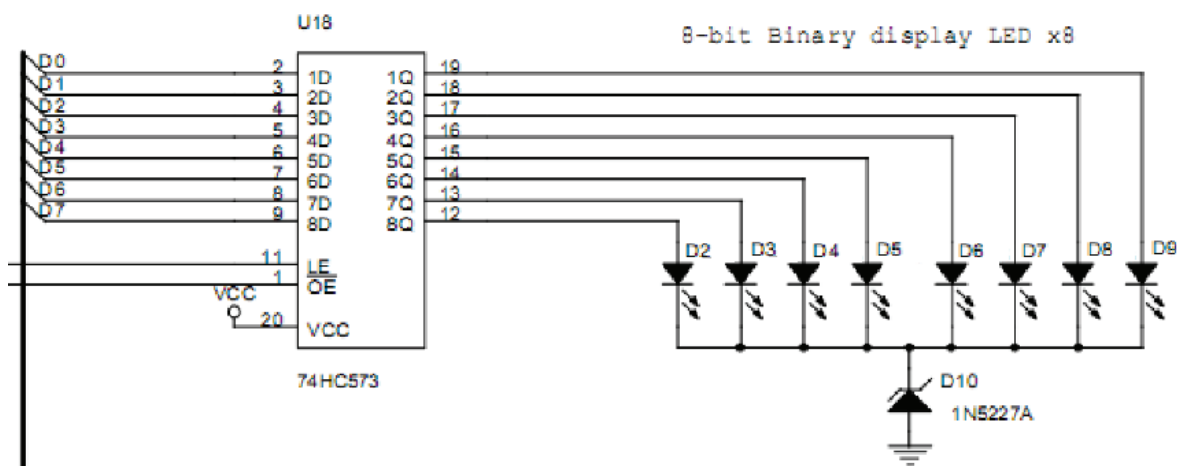
Note: # logical OR, & logical AND, ! Logical NOT



We can edit the PLD equation using WinCupL IDE directly. When compile it, the JEDEC file will be produced, 8086KIT.jed.

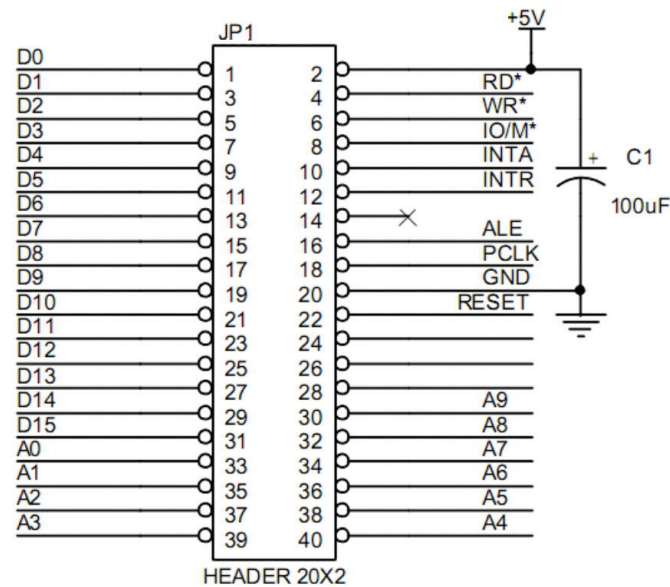
GPIO1 LED

Kit provides a useful 8-bit binary display. It can be used to debug the program or code running demonstration. This debug LED is located at output address at 000H. We can use instruction OUT 0,AL that writes 8-bit data in AL to the LED. Logic 1 makes LED ON and logic 0 makes LED OFF. The hex code for OUT 0,AL is E6, 00. Only two bytes and easy to remember! Anytime we need to check the 8-bit contents, we can copy it to register AL then use this instruction to write it to the GPIO1 LED easily.



EXPANSION BUS HEADER

JP1, 40-pin header provides CPU bus signals for expansion or I/O interfacing. Students may learn how to make the simple I/O port, interfacing to Analog-to-Digital Converter, interfacing to stepper motor or AC power circuits. Data bus is 16-bit, D0 to D15, address bus is 10-bit, A0 to A9.



CONNECTING LCD MODULE

JR1 is 16-pin header for connecting the text LCD module. R15 is a current limit resistor for back-light. R16 is trimmer for contrast adjustment. The LCD module is interfaced to the 8086 bus directly. The command and data registers are located in I/O space having address from 500H to 506H. Any text LCD with HD44780 compatible controller can be used.

LCD Register	I/O address
Command Write	500H
Command Read	504H
Data Write	502H
Data Read	506H

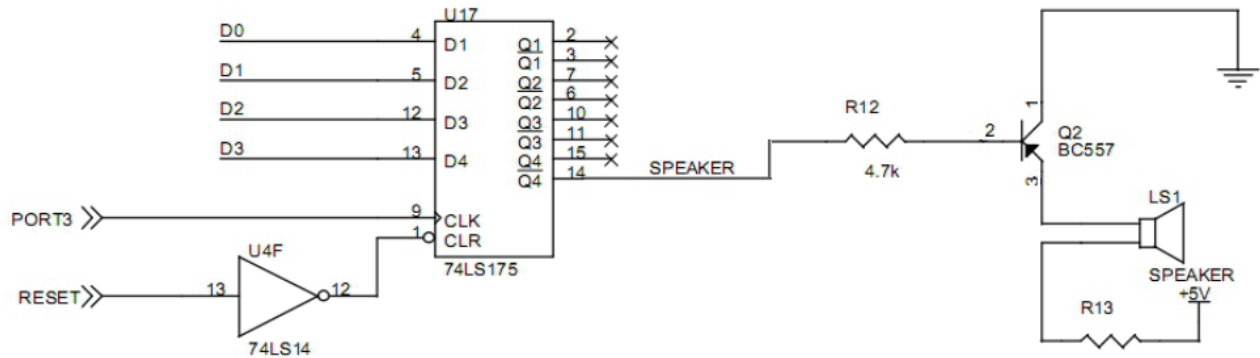
8086 kit signal names															
GND	+5V	CNTR	A1	A2	LCD_E	D0	D1	D2	D3	D4	D5	D6	D7	VBL	GND
names of the LCD module connector															
VSS	VDD	V0	RS	RW	E	D0	D1	D2	D3	D4	D5	D6	D7	A	K
JR1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
															16

Note: Insert or remove the LCD module only if the kit is powered off.



SPEAKER PORT

Kit has small speaker driven by digital output bit 3 of the PORT3. We can produce tone signal by pulsing this bit, ON and OFF. The small signal PNP transistor then drives the speaker load producing TONE. PORT3 is located at I/O space at 280H. We will use OUT DX,AL instruction to control it.



INTERRUPT TICK GENERATOR

```
; Source code listing of 10ms system tick timer for 8086 kit
; AT89C2051
; oscillator input 2.4576MHz
```

```
$mod52
```

```
DSEG AT 30h
```

```
CSEG AT 0
```

```
TICK: DS 1
```

```
JMP MAIN
```

```
ORG 00bh
```

```
TIMER0_VECTOR:
```

```
JMP TIMER0_SERVICE
```

```
org 100h
```

```
;-----initialize data-----
```

```
MAIN: MOV    R7,#0
      DJNZ   R7,$
      MOV    SP,#60h      ; set stack pointer to 60h
      MOV    P1,#0FFH
      MOV    TICK,#0
      MOV    TMOD,#1      ; set timer mode 1
      SETB   ET0          ; enable timer0 interrupt
      SETB   EA          ; enable all interrupt
```

```

        SETB  TR0          ; run timer0
        SJMP  $

; RELOAD FA2CH FOR 10MS INTERRUPT

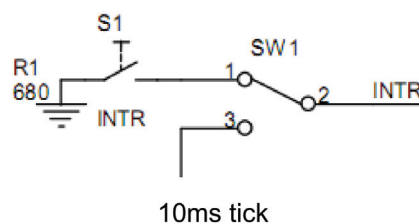
TIMER0_SERVICE:
    PUSH    ACC
    ORL     TH0,#0F8H      ; ORL TL0,#07DH ; computed value is F800H
    CLR     P3.7 ;         ; CPL P3.7
    NOP
    ; NOP
    ; NOP
    ; NOP
    SETB    P3.7
    CPL     P1.6           ; CHECK TICK FREQUENCY
    INC     TICK
    MOV     A,TICK
    CJNE    A,#100,SKIP
    MOV     TICK,#0
    CLR     P1.7
    MOV     R7,#30
    DJNZ    R7,$
    SETB    P1.7

SKIP:
READ_MORE:
    POP     ACC
    RETI

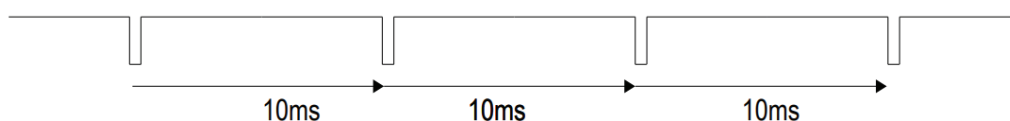
END

```

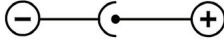
SW1 is a selector for interrupt source between key INTR key or 10ms tick produced by the 89C2051 microcontroller. Tick generator is software controlled using `timer0` interrupt in the 89C2051 chip. The active low tick signal is sent to P3.7. D1 LED blinking indicates tick running.



Tick is a 10ms periodic signal for triggering the 8086 INTR pin. When select SW1 to Tick, the 8086 CPU can be triggered by the external interrupt. The 100Hz tick or 10ms tick can be used to produce tasks that executed with multiple of tick.



DC POWER SUPPLY

Kit accepts DC power supply with DC voltage from +7.5V to +9.0V. It draws current approx. 500mA. However we can use +9VDC from any AC adapter. The example of AC adapter is shown below. The center pin is positive, the outer is GND. 



POWER UP

Connect DC power supply to the kit. When power up, we will see the cold boot message 8086.

8086

Press *IP* key, the display will show HOME location at 0400. The data field will show its random content.

0400 EA.

Keys +/- will increment / decrement address.

0401 FE.

DISPLAY USER REGISTERS

Before we write and test programs, let us see how to examine user registers. User registers are the memory block in RAM that used to save the contents of CPU registers after completed a given program running. We can examine the user registers for checking our code running then. Press key *REG*, then press key *0*, it will show 16-bit contents of register AX (AH:AL).

AX 0000

Press key *Reg*, 1, 2, 3, for BX, CX, and DX.

PROGRAM EXAMPLE 1: WRITE TO GPIO1 LED

Let us try to enter 8086 instructions using HEX CODE or machine code.

Address	Hex code	Label	Instruction	Comment
0400	B0 01	MAIN	MOV AL,1	load AL with 1
0402	E6 00		OUT 0,AL	write AL to GPIO @ 000H
0404	CC		INT 3	return to monitor @ 000CH

Analyze Program Example 1

Our test program has only three instructions.

The **1st** instruction is:

MOV AL,1 Load AL register with the 8-bit constant, 01. This instruction has two bytes hex code i.e. B0 and 01. B0 is instruction MOV AL,n and 01 is n.

The **2nd** instruction is:

OUT 0,AL Copy AL register to output port GPIO1 LED at location 000H.
The instruction's machine code is E6. The location of GPIO1 is 000H.

The **3rd** instruction is:

INT 3 Hex code CC jump back to monitor program at location 000CH.

This test program has only 5 bytes: B0, 01, E6, 00, CC. The first byte will be entered to location 0400. And the following bytes will be entered at 0401, 0402, 0403, and 0404. Let us see how to enter these codes into the memory.

Edit Program Example 1

Step 1 Press RESET, then IP. The display will show current memory address and its contents.



Shown the location 0400 has random data 2F. There are small dots at the data field indicating the active field, ready for modifying the hex contents.

Step 2 Press key B and key 0. The new hex code B0 will be entered to the location 0400.



Step 3 Press key + to increment the location from 0400 to 0401. Then enter hex value 1.



Repeat Step 3 until completed for the last location. We can verify the hex code with key + or key -. To change the display location, press key ADDR. The dots will move to address field. Any hex key pressed will change the display address.

Run Program Example 1

To run thr program, press first RESET, then IP. Now we can test above code running by pressing key GO. The last instruction INT 3 will return the CPU to monitor program to save CPU registers to user registers that we can examine the program operation.

What is happening?

The GPIO1 LED will be 0000 0001.

Can you change the display from 0000 0001 to another value? How? Test run again.

PROGRAM EXAMPLE 2: RUNNING LIGHT ON GPIO1 LED

Another example is LED running program.

Address	Hex code	Label	Instruction	Comment
0400	B0 01	START	MOV AL,1	load AL with 1
0402	E6 00	LOOP	OUT 0,AL	write AL to GPIO @ 000H
0404	D0 C0		ROL AL,1	rotate Left AL
0406	E8 02 00		CALL DELAY	Call DELAY subroutine
0409	EB F7		JMP LOOP	Repeat LOOP forever
040B	B9 00 30	DELAY	MOV CX, 3000H	load CX with 3000H
040E	E2 FE		LOOP \$	repeat here
0410	C3		RET	return to main program

Now let us enter code, when complete, press key IP then GO.

What is happening? Can you change running speed? How?

Can you change direction? How?

PROGRAM EXAMPLE 3: INTERRUPT

This example code demos the 8086 maskable interrupt using the 10ms tick. The optional R1, data bus pull-up provides a simple interrupt type number FF when the 8086 chip acknowledges the INTR triggered. The kit thus gets the vectors from location 03FC (IP) and 03FE (CS).

			ORG 3FCH	; INTERRUPT NUMBER FF
03FC	00		DFB INTR & 0FFH	; IP = 0500
03FD	05		DFB INTR >> 8	
03FE	00		DFB 0	; CS = 0000
03FF	00		DFB 0	
0400			ORG 400H	
0400	FB	MAIN	STI	; ENABLE INTERRUPT
0401	EB FE		JMP \$	; JMP HERE
0500			ORG 500H	
0500	FE C4	INTR	INC AH	
0502	80 FC 64		CMP AH,100	
0505	75 06		JNZ SKIP	
0507	B4 00		MOV AH,0	
0509	FE C0		INC AL	
050B	E6 00		OUT 0,AL	
050D	CF	SKIP	IRET	

We see that the interrupt service vectors 0500 are stored at 03FC and 03FD. The code segment register CS 0000 are stored at 03FE and 03FF. Main program, just enable the interrupt with STI instruction. Then JUMP here, waiting for interrupt. The service routine uses AH for 100 tick counting. AL is incremented every one second. We see the counting at GPIO1 LED easily.

Set switch SW1 to 10ms Tick position.

Let us enter the hex code and test run. What is happening? Can you change the counting rate? How?

PROGRAM EXAMPLE 4: SPEAKER

```
ORG 400H

0400 BA 80 02      START  MOV DX,PORT3      ; load DX with 280

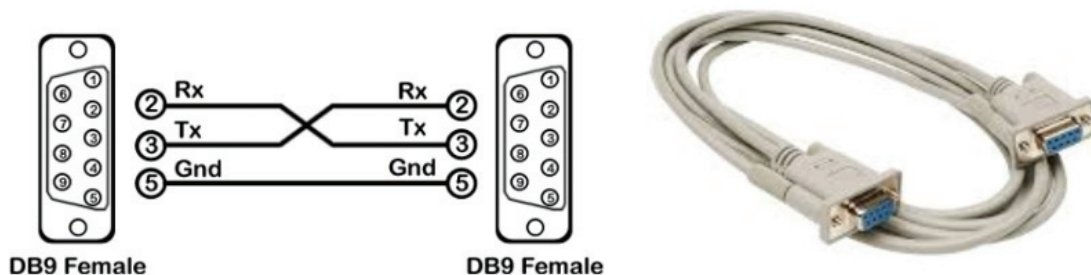
0403 B0 00          TONE   MOV AL,0          ; load AL with 0
0405 EE              OUT DX,AL              ; write AL to PORT3
0406 E8 08 00        CALL DELAY             ; call delay
0409 B0 08              MOV AL,8            ; load AL with 8
040B EE              OUT DX,AL              ; write AL to PORT3
040C E8 02 00        CALL DELAY             ; call delay
040F EB F2              JMP TONE             ; repeat forever

0411 B9 50 00      DELAY  MOV CX,50H         ; load CX with 50
0414 E2 FE              LOOP $               ; repeat here
0416 C3              RET                    ; return
```

Above program produces TONE signal. Let us enter the hex code to memory from address 400H to 416H. Press IP, then GO. What is happening? Can you change the frequency higher or lower? How?

RS232C PORT

The RS232C port is for serial communication. We can use a cross cable or null modem cable to connect between the kit and terminal. The connector for both sides are DB9 female.

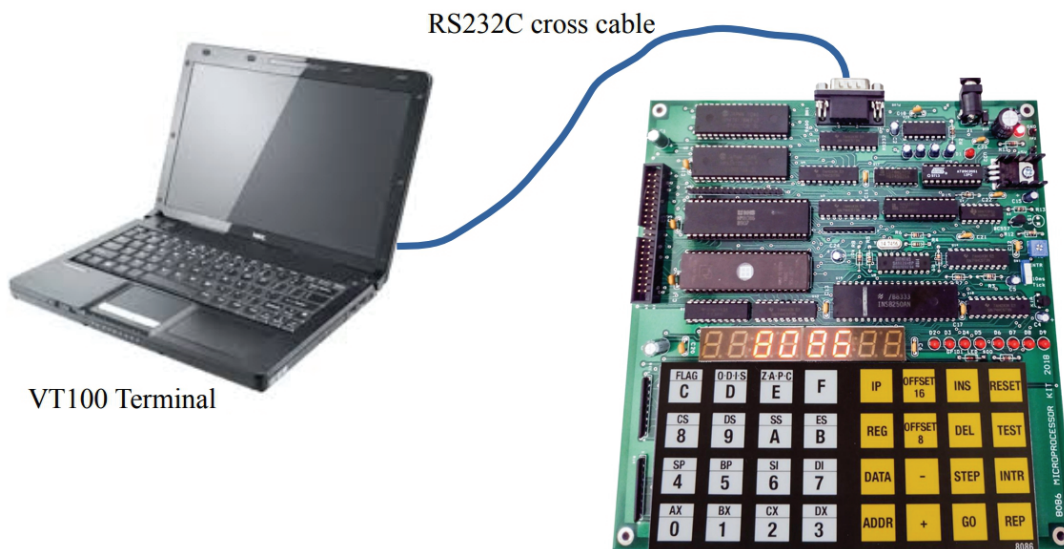


CONNECTING 8086 KIT TO TERMINAL

The example shows connecting a PC with COM1 port to the RS232C port of the 8086 kit. PC without COM port, we may use the USB-RS232 adapter.

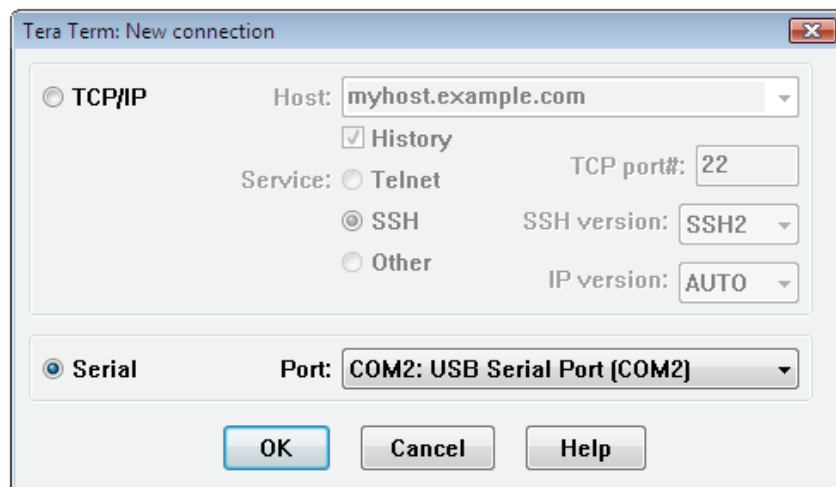


Download free terminal program, Teraterm from URL <http://tssh2.sourceforge.jp/index.html.en> .

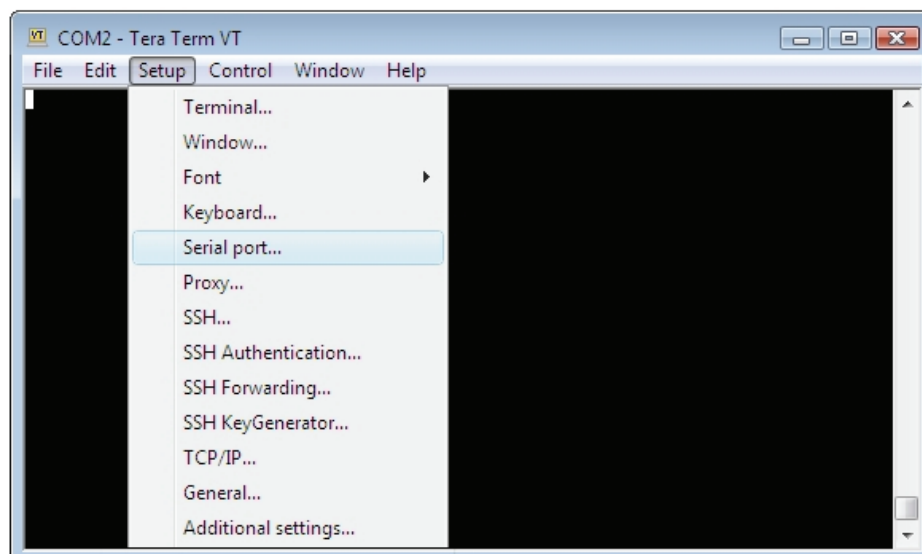


To download Intel hex file that generated from the assembler or C compiler, set serial port speed to 9600 bit/s, 8-data bit, no parity, no flow control, one stop bit.

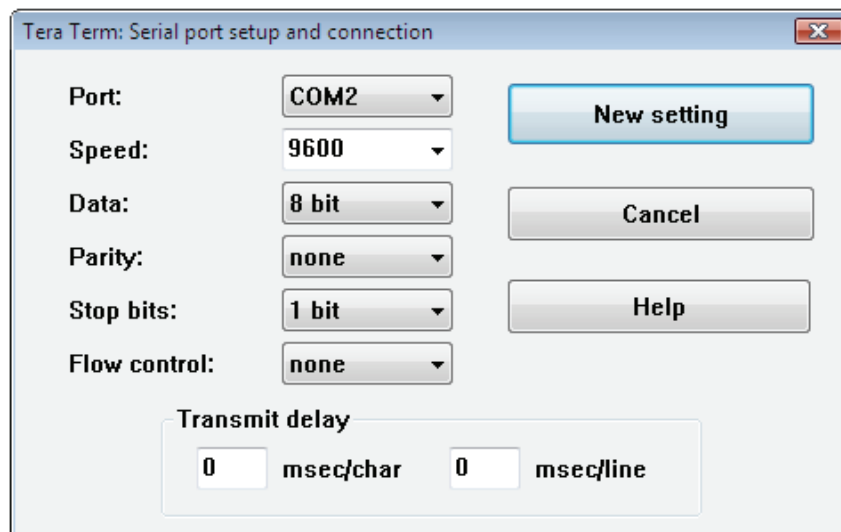
Step 1 Run TeraTerm, then click at Serial connection.



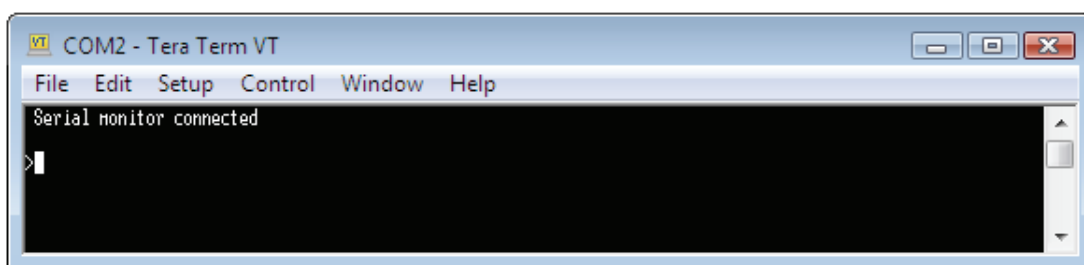
Step 2 Click Setup>Serial port



Step 3 Select your communication port and set serial port parameters 9600 8n1 as shown below.

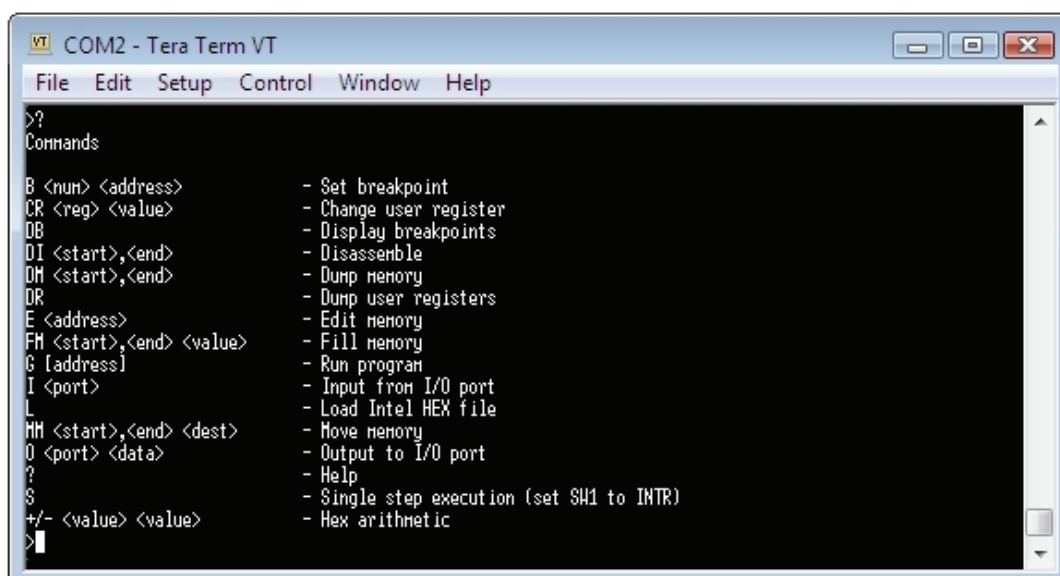


Step 4 Press ENTER key on terminal. The kit will connect terminal automatically. The monitor prompt is >. To get back to onboard monitor, press key RESET on 8086 kit keypad.



MONITOR COMMANDS

Press key ? for help menu with monitor commands.



To get back to onboard monitor, press key RESET.

Table of monitor commands

B <num> <address>	- Set breakpoint
CR <reg> <value>	- Change user register
DB	- Display breakpoints
DI <start>,<end>	- Disassemble
DM <start>,<end>	- Dump memory
DR	- Dump user registers
E <address>	- Edit memory
FM <start>,<end> <value>	- Fill memory
G [address]	- Run program
I <port>	- Input from I/O port
L	- Load Intel HEX file
MM <start>,<end> <dest>	- Move memory
O <port> <data>	- Output to I/O port
?	- Help
S	- Single step execution (set SW1 to INTR)
+/- <value> <value>	- Hex arithmetic

Example: enter dm 1000,1100 for memory dump of address 1000H to 1100H:

The screenshot shows a Tera Term VT window titled 'COM2 - Tera Term VT'. The command prompt shows '>dm 1000,1100'. The output displays a memory dump from address 0000:1000 to 0000:1100. Each line shows the address, followed by two columns of hexadecimal values, and then a column of ASCII characters. The ASCII column shows various symbols and letters, including 'V', 'Z', 'P', 'X', 'E', 'a', 'u', 'R', 'L', 'c', 'H', 'H', and 'o'.

```

>dm 1000,1100
0000:1000 F9 BF F4 FF F7 BF FF FA FF FF F0 F8 FF FF F7 FF .....
0000:1010 2C 56 44 80 FF FE BF 97 FF 7E FA FF FD FB 6F BF ,V.....o.
0000:1020 00 22 00 C4 00 25 00 89 00 00 00 A0 00 00 00 00 .....Z.....
0000:1030 00 01 04 00 00 20 08 21 00 50 05 10 00 58 00 00 .....!P...X..
0000:1040 4F 00 F8 FF FD F9 DF FF FD FF F8 FE F7 EF FE FB 0.....
0000:1050 FF FF F5 78 FF FF DE DF FF F5 FF F9 9F FD FB 8D .....{.....
0000:1060 10 98 00 04 00 04 04 04 00 45 00 26 00 29 00 2E .....E.&)...
0000:1070 00 80 00 C2 00 41 80 61 00 00 00 78 00 82 00 00 .....A.a...x...
0000:1080 13 0B 12 59 B3 7D DF FF AF FB 77 FF FF FF FB FF ...V.)...u....
0000:1090 FF 3E EE EF FF FF FF F7 9F FF 7D FF FF AD BC BD .>.....}.A
0000:10A0 00 98 00 11 00 21 00 AA 00 18 00 E9 40 02 00 41 .....!.....R
0000:10B0 80 00 00 06 00 0E 00 81 00 80 00 52 00 0B 00 90 .....R.....
0000:10C0 EF FF EF FF FF FF FD 7F FF FF EC FB F7 DF 7B DF .....{.....
0000:10D0 77 F4 7F 7F 07 7F FD 0F AF BC FD 7F FF 7B CF 77 u.....{u
0000:10E0 00 22 40 00 20 0F 10 9F 04 C2 02 4C 00 63 00 6D ."E. ....L.c.H
0000:10F0 40 C0 41 1C 00 12 02 98 48 04 C8 20 02 EC 01 3F @.A....H.. ...?
0000:1100 AF FE EE EF FF E7 E3 FB FD 6F EC 7D FD DF F9 9C .....o.)....

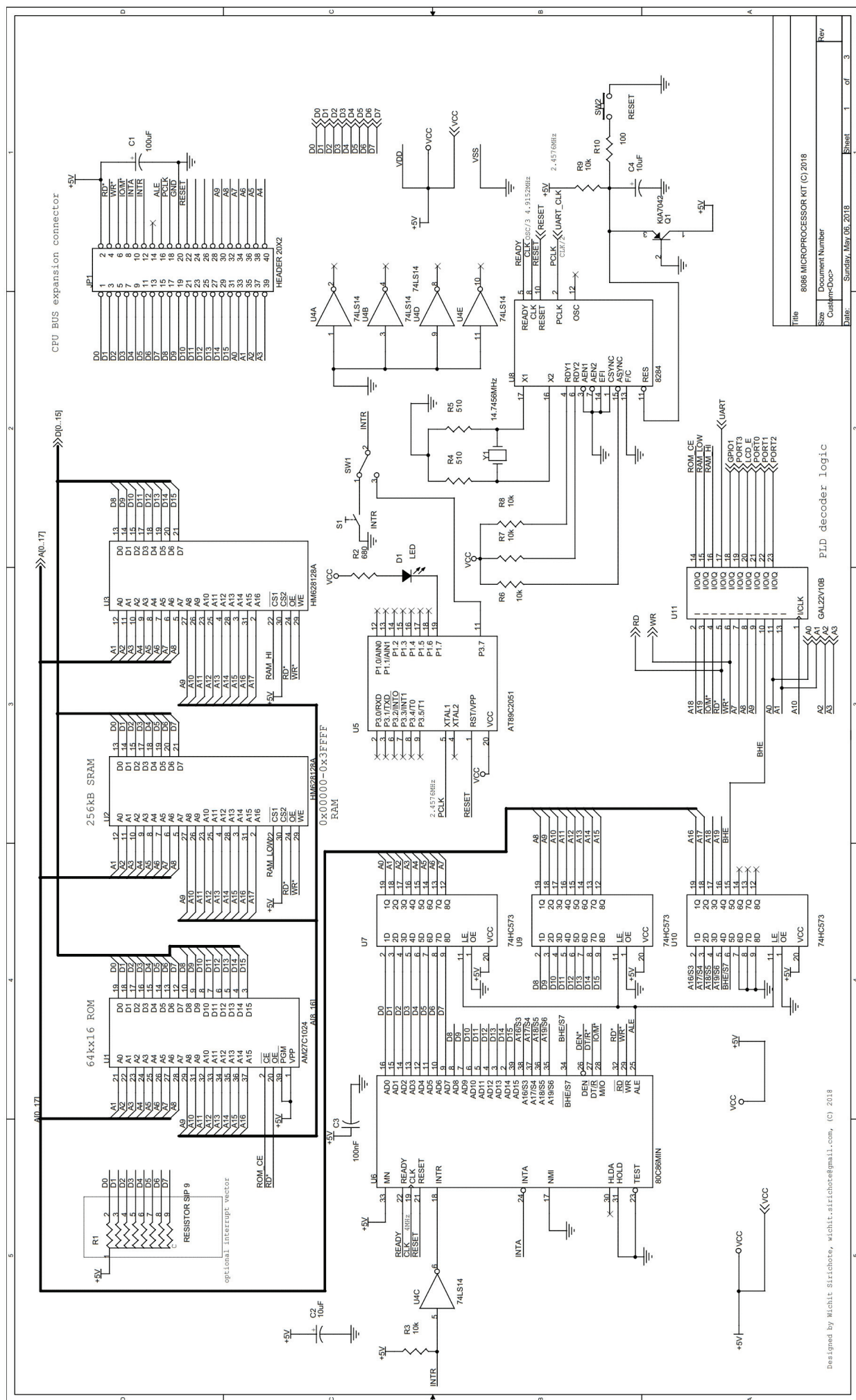
```

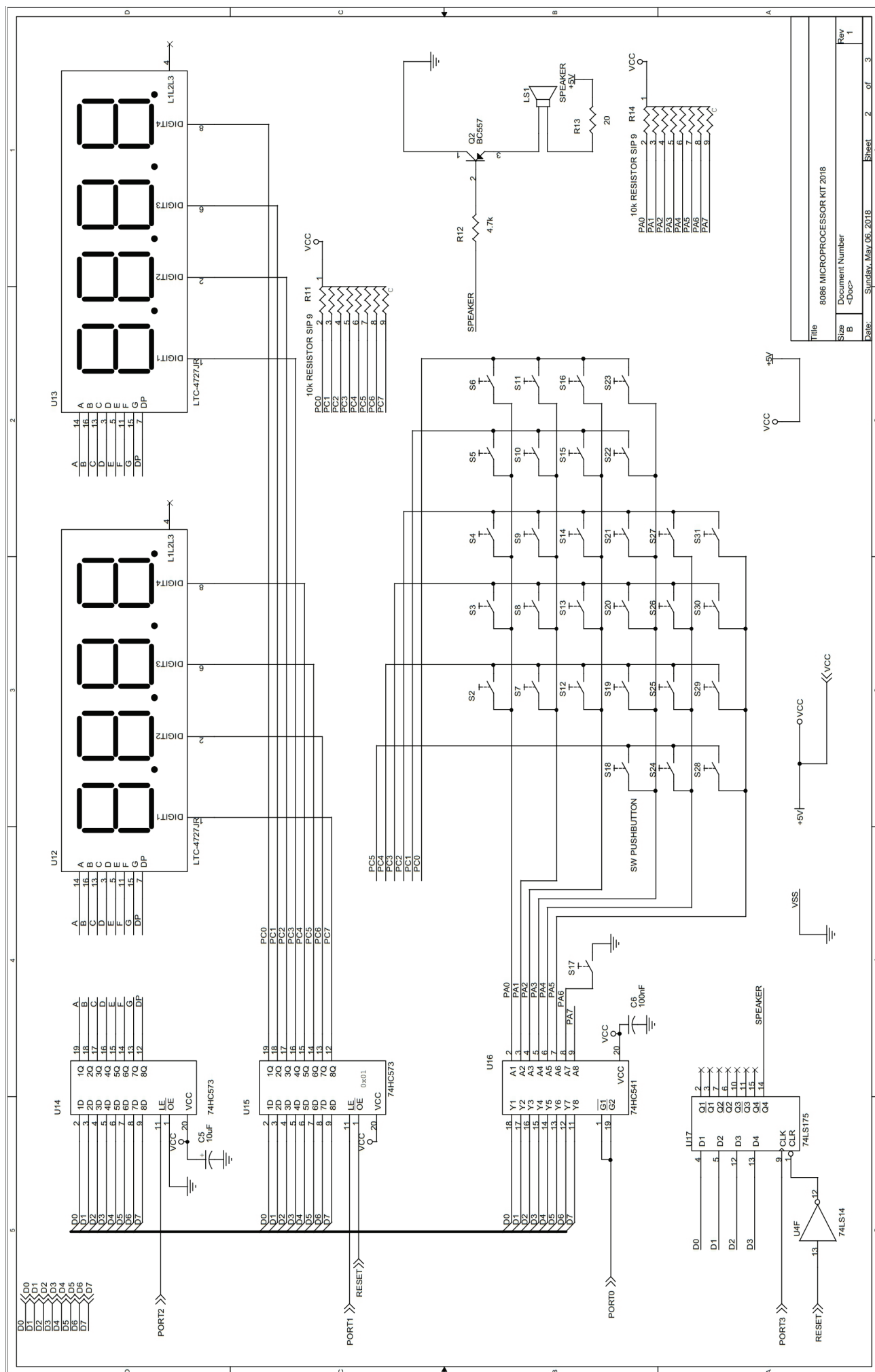
LOGIC PROBE POWER SUPPLY

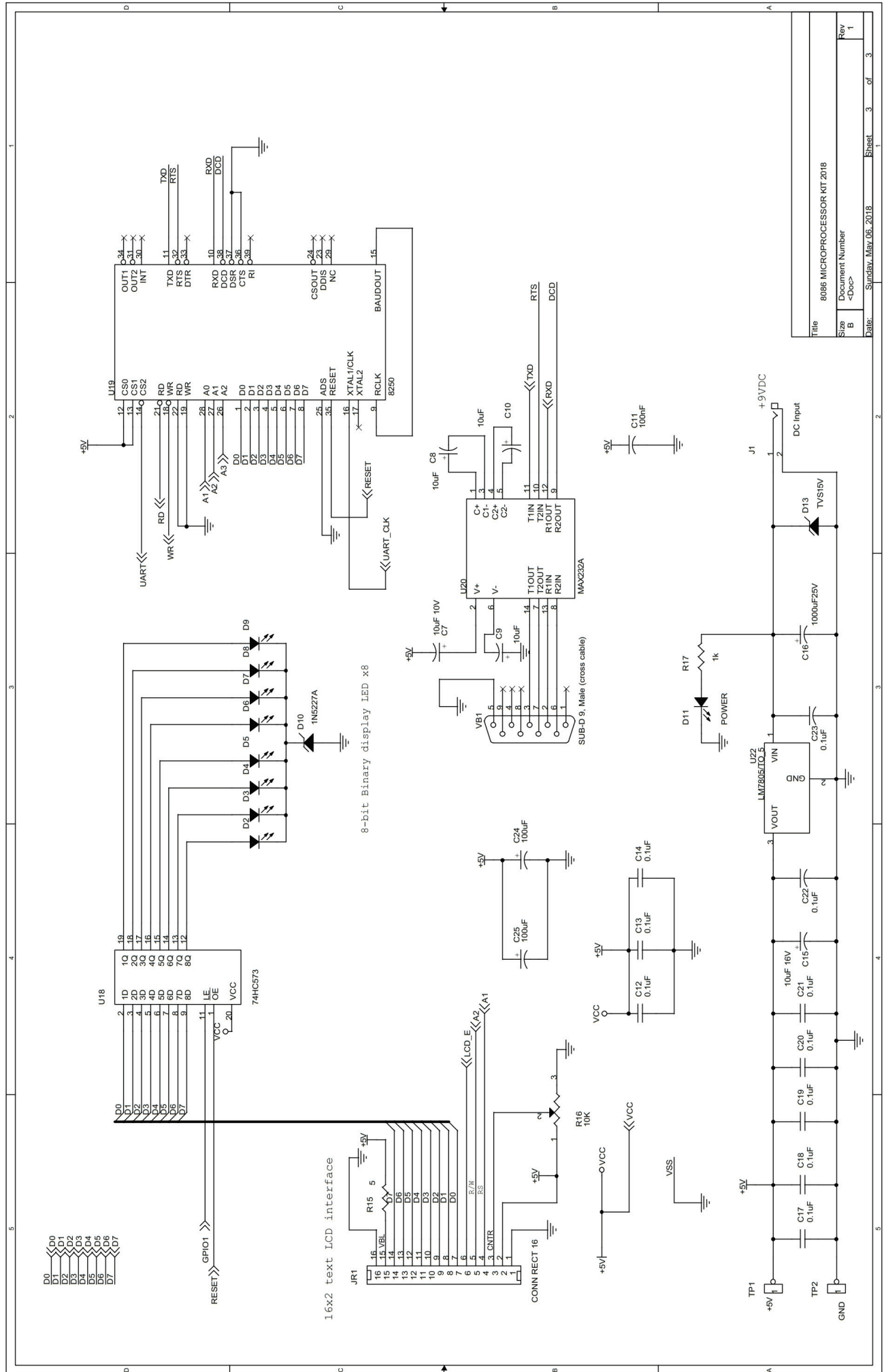
The kit provides test points TP1 (+5V) and TP2 (GND) for using the logic probe. Students may learn digital logic signals with logic probe easily. Red clip is for +5V and Black clip for GND.



HARDWARE SCHEMATIC







PARTS LIST

Semiconductors

U1	AM27C1024, 16-bit 256kB EPROM
U2, U3	HM628128A, 128kB SRAM
U4	74HC14, hex inverter
U5	AT89C2051, 8-bit tick generator
U6	8086, 16-bit Microprocessor
U7,U9,U10,U14,U15,U18	74HC573, 8-bit latch
U8	8284, clock generator
U11	GAL22V10, PLD
U12,U13	LTC-4727, 7-segment LED
U16	74HC541, 8-bit tristate buffer
U17	74LS175, 4-bit latch
U19	8250, 16550 UART
U20	HIN232, RS232 voltage converter
U22	LM7805, +5V voltage regulator

D1,D2,D3,D4,D5,D6,D7,D8, D9

LED, 3mm

D10	1N5227A
D11	POWER
D13	TVS15V
Q1	KIA7042,
Q2	BC557, PNP transistor

Resistors (all resistors are 1/8W +/-5%)

R1	10k Resistor SIP 9
R2	680
R3,R6,R7,R8,R9,R16	10K
R4,R5	510
R10	100
R14,R11	10k Resistor SIP 9
R12	4.7k
R13	20
R15	5
R17	1k

Capacitors

C1,C24,C25	100uF
C2,C4,C5,C8,C9,C10	10uF
C3,C6,C11	100nF
C7	10uF 10V
C12,C13,C14,C17,C18,C19,	0.1uF
C20,C21, C15	10uF 16V
C16	1000uF 25V
C22,C23	0.1uF

Additional parts

Y1	14.7456MHz
VB1	SUB-D 9, Male (cross cable)
JP1	HEADER 20X2
JR1	CONN RECT 16
J1	DC Input
LS1	SPEAKER
SW1	SW MAG-SPDT
SW2	RESET
S1	INTR
S2,S3,S4,S5,S6,S7,S8,S9, SW,	S10,S11,S12,S13,S14,S15,
S16,S17,S18,S19,S20,S21,	S22,S23,S24,S25,S26,S27,
S28,S29,S30,S31	PUSHBUTTON
TP1	+5V
TP2	GND

PCB double side plate through hole

LED cover Clear RED color acrylic plastic

Keyboard sticker printable SVG file

MONITOR PROGRAM LISTING

```
0001 // MON86.C
0002 // Monitor program developed with Micro-c for 8086
0003 // written by Wichit Sirichote, wичit.sirichote@gmail.com
0004 // 13 May 2018
0005
0006 #include <8086io.h>
0007
0008 #define port0 0x100 // input port keypad
0009 #define port1 0x180 // keypad out, LED: common cathodes
0010 #define port2 0x200 // LED: segments
0011 #define port3 0x280 // BIT3: speaker
0012 #define UART 0x300 // 8250 UART
0013 #define user_io 0x400 // user expansion i/o port
0014 #define gpio1 0 // debug LED
0015 #define BUSY 0x80
0016 #define LCD_command_write 0x500
0017 #define LCD_command_read 0x504
0018 #define LCD_data_write 0x502
0019 #define LCD_data_read 0x506
0020
0021 char convert[16] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
0022                    0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};
0023 char buffer[8]; // display buffer
0024 char buffer2[128];
0025 unsigned char k, n;
0026 char i,j,u,q,o,x;
0027 unsigned int temp,temp2;
0028 int key;
0029 char hit;
0030 char auto_key;
0031 int temp16;
0032 int PC, save_PC, display_PC;
0033 int start,end,destination;
0034 char state;
0035 char *dptr;
0036 int user_flag, save_stack;
0037 int user_ip;
0038 int user_cs;
0039 int user_ds;
0040 int user_es;
0041 int user_ss;
```

```
0042  int user_sp;
0043  int user_ax;
0044  int user_bx;
0045  int user_cx;
0046  int user_dx;
0047  int user_bp;
0048  int user_si;
0049  int user_di;
0050  int flag_low,flag_high;
0051
0052  LcdReady()
0053  {
0054      while(in(LCD_command_read)&BUSY)
0055          continue; // wait until busy flag =0
0056  }
0057
0058  void clr_screen(void)
0059  {
0060      LcdReady();
0061      out(LCD_command_write,0x01);
0062  }
0063
0064  // if LCD connected, return 0, else return !=0
0065  char check_LCD()
0066  {
0067      i=0;
0068      while((in(LCD_command_read) & BUSY != 0) && ++i < 100)
0069          ;
0070      return (in(LCD_command_read) & BUSY);
0071  }
0072
0073  goto_xy(char x, char y)
0074  {
0075      LcdReady();
0076      switch(y){
0077          case 0 : out(LCD_command_write, 0x80 + x); break;
0078          case 1 : out(LCD_command_write, 0xC0 + x); break;
0079          case 2 : out(LCD_command_write, 0x94 + x); break;
0080          case 3 : out(LCD_command_write, 0xd4 + x); break;
0081      }
0082  }
```

```
0083
0084 InitLcd()
0085 {
0086     LcdReady();
0087     out(LCD_command_write, 0x38);
0088     LcdReady();
0089     out(LCD_command_write, 0x0c);
0090     clr_screen();
0091     goto_xy(0,0);
0092 }
0093
0094 char *Puts(char *str)
0095 {
0096     unsigned char i;
0097     for (i=0; str[i] != '\0'; i++){
0098         LcdReady();
0099         out(LCD_data_write, str[i]);
0100     }
0101     return str;
0102 }
0103
0104 void putch_lcd(char ch)
0105 {
0106     LcdReady();
0107     out(LCD_data_write, ch);
0108 }
0109
0110 void LCDWriteText(char *txt)
0111 {
0112     while(*txt)
0113         putch_lcd(*txt++);
0114 }
0115
0116 void LCDWriteConstText(const char *txt)
0117 {
0118     while(*txt)
0119         putch_lcd(*txt++);
0120 }
0121
0122 dot_address()
0123 {
```

```
0124     buffer[0] = buffer[0] & ~0x80;
0125     buffer[1] = buffer[1] & ~0x80;
0126     buffer[2] = 0; // could be 1MB physical location
0127     buffer[3] = buffer[3] | 0x80;
0128     buffer[4] = buffer[4] | 0x80;
0129     buffer[5] = buffer[5] | 0x80;
0130     buffer[6] = buffer[6] | 0x80;
0131     buffer[7] = 0;
0132 }
0133
0134 dot_data()
0135 {
0136     buffer[0] = buffer[0] | 0x80;
0137     buffer[1] = buffer[1] | 0x80;
0138     buffer[2] = 0;
0139     buffer[3] = buffer[3] & ~0x80;
0140     buffer[4] = buffer[4] & ~0x80;
0141     buffer[5] = buffer[5] & ~0x80;
0142     buffer[6] = buffer[6] & ~0x80;
0143     buffer[7] = 0;
0144 }
0145
0146 hex4(int h)
0147 {
0148     temp16 = h;
0149     buffer[3] = convert[temp16 & 0xf];
0150     temp16 >>= 4;
0151     buffer[4] = convert[temp16 & 0xf];
0152     temp16 >>= 4;
0153     buffer[5] = convert[temp16 & 0xf];
0154     temp16 >>= 4;
0155     buffer[6] = convert[temp16 & 0xf];
0156     buffer[7] = 0;
0157 }
0158
0159 hex4low(int h)
0160 {
0161     temp16=h;
0162     buffer[0] = convert[temp16 & 0xf];
0163     temp16 >>= 4;
0164     buffer[1] = convert[temp16 & 0xf];
```

```
0165     temp16 >>= 4;
0166     buffer[2] = convert[temp16 & 0xf];
0167     temp16 >>= 4;
0168     buffer[3] = convert[temp16 & 0xf];
0169 }
0170
0171 address_display()
0172 {
0173     temp16 = PC;
0174     hex4(temp16);
0175 }
0176
0177 data_display()
0178 {
0179     dptr = PC;
0180     n = *dptr;
0181     temp16 = n;
0182     buffer[0] = convert[temp16 & 0xf];
0183     temp16>>=4;
0184     buffer[1] = convert[temp16 & 0xf];
0185     buffer[2] = 0;
0186 }
0187
0188 read_memory()
0189 {
0190     address_display();
0191     data_display();
0192 }
0193
0194 key_address()
0195 {
0196     state = 1;
0197     read_memory();
0198     dot_address();
0199     hit = 0;
0200 }
0201
0202 key_data()
0203 {
0204     read_memory();
0205     dot_data();
```

```
0206     hit = 0;
0207     state = 2;
0208 }
0209
0210 key_PC()
0211 {
0212     PC = save_PC;
0213     key_data();
0214 }
0215
0216 key_plus()
0217 {
0218     if(state == 1 || state == 2)
0219     {
0220         PC++;
0221         display_PC = PC;
0222         read_memory();
0223         key_data();
0224     }
0225
0226     if(state == 4)
0227     {
0228         start = display_PC;
0229         state = 5;
0230         buffer[0] = 0x5e;
0231         hit = 0;
0232     }
0233
0234     if(state == 6)
0235     {
0236         start = display_PC;
0237         state = 7;
0238         buffer[0] = 0x5e;
0239         hit = 0;
0240     }
0241 }
0242
0243 key_minus()
0244 {
0245     if(state == 1 | state == 2)
0246     {
```

```
0247     PC--;
0248     display_PC = PC;
0249     read_memory();
0250     key_data();
0251 }
0252 }
0253
0254 data_hex()
0255 {
0256     dptr = PC;
0257     x = *dptr;
0258     if(hit == 0) x = 0;
0259     {
0260         hit = 1;
0261         x = x << 4;
0262         x = x|key;
0263         *dptr = x;
0264         read_memory();
0265         dot_data();
0266     }
0267 }
0268
0269 hex_address()
0270 {
0271     if(hit == 0) PC = 0;
0272     {
0273         hit = 1;
0274         PC <=< 4;
0275         PC |= key;
0276         read_memory();
0277         dot_address();
0278     }
0279 }
0280
0281 // insert word and shift 512 words down
0282 insert()
0283 {
0284     dptr = PC;
0285     for(j = 512; j > 0; j--)
0286     {
0287         *(dptr+j) = *(dptr+j-1);
```

```
0288     }
0289     *(dptr+1) = 0; // insert next byte
0290     PC++;
0291     read_memory();
0292     dot_data();
0293     state = 2;
0294 }
0295
0296 // delete current word and shift 512 words up
0297 cut_byte()
0298 {
0299     dptr=PC;
0300     for(j = 0; j < 512; j++)
0301     {
0302         *(dptr+j)=*(dptr+j+1);
0303     }
0304     read_memory();
0305     dot_data();
0306     state = 2;
0307 }
0308
0309 reg_ax()
0310 {
0311     temp16 = user_ax;
0312     hex4low(temp16);
0313     buffer[4] = 0;
0314     buffer[5] = 0x64;
0315     buffer[6] = 0x77;
0316     buffer[7] = 0;
0317 }
0318
0319 reg_bx()
0320 {
0321     temp16 = user_bx;
0322     hex4low(temp16);
0323     buffer[4] = 0;
0324     buffer[5] = 0x64;
0325     buffer[6] = 0x7c;
0326     buffer[7] = 0;
0327 }
0328 reg_cx()
```

```
0329  {
0330      temp16 = user_cx;
0331      hex4low(temp16);
0332      buffer[4] = 0;
0333      buffer[5] = 0x64;
0334      buffer[6] = 0x39;
0335      buffer[7] = 0;
0336  }
0337
0338  reg_dx()
0339  {
0340      temp16 = user_dx;
0341      hex4low(temp16);
0342      buffer[4] = 0;
0343      buffer[5] = 0x64;
0344      buffer[6] = 0x5e;
0345      buffer[7] = 0;
0346  }
0347
0348  reg_sp()
0349  {
0350      temp16 = user_sp;
0351      hex4low(temp16);
0352      buffer[4] = 0;
0353      buffer[5] = 0x73;
0354      buffer[6] = 0x6d;
0355      buffer[7] = 0;
0356  }
0357
0358  reg_bp()
0359  {
0360      temp16 = user_bp;
0361      hex4low(temp16);
0362      buffer[4] = 0;
0363      buffer[5] = 0x73;
0364      buffer[6] = 0x7c;
0365      buffer[7] = 0;
0366  }
0367
0368  reg_si()
0369  {
```

```
0370     temp16 = user_si;
0371     hex4low(temp16);
0372     buffer[4] = 0;
0373     buffer[5] = 0x30;
0374     buffer[6] = 0x6d;
0375     buffer[7] = 0;
0376 }
0377
0378 reg_di()
0379 {
0380     temp16 = user_di;
0381     hex4low(temp16);
0382     buffer[4] = 0;
0383     buffer[5] = 0x30;
0384     buffer[6] = 0x5e;
0385     buffer[7] = 0;
0386 }
0387
0388 reg_cs()
0389 {
0390     temp16 = user_cs;
0391     hex4low(temp16);
0392     buffer[4] = 0;
0393     buffer[5] = 0x6d;
0394     buffer[6] = 0x39;
0395     buffer[7] = 0;
0396 }
0397
0398 reg_ds()
0399 {
0400     temp16 = user_ds;
0401     hex4low(temp16);
0402     buffer[4] = 0;
0403     buffer[5] = 0x6d;
0404     buffer[6] = 0x5e;
0405     buffer[7] = 0;
0406 }
0407
0408 reg_ss()
0409 {
0410     temp16 = user_ss;
```

```
0411     hex4low(temp16);
0412     buffer[4] = 0;
0413     buffer[5] = 0x6d;
0414     buffer[6] = 0x6d;
0415     buffer[7] = 0;
0416 }
0417
0418 reg_es()
0419 {
0420     temp16 = user_es;
0421     hex4low(temp16);
0422     buffer[4] = 0;
0423     buffer[5] = 0x6d;
0424     buffer[6] = 0x79;
0425     buffer[7] = 0;
0426 }
0427
0428 reg_flag()
0429 {
0430     temp16 = user_flag;
0431     hex4low(temp16);
0432     buffer[4] = 0;
0433     buffer[5] = 0x38;
0434     buffer[6] = 0x71;
0435     buffer[7] = 0;
0436 }
0437
0438 reg_flag_low()
0439 {
0440     flag_low = 0;
0441     temp16 = user_flag;
0442     if(temp16 & 1) flag_low |= 1;
0443     if(temp16 & 4) flag_low |= 0x10;
0444     if(temp16 & 0x10) flag_low |= 0x100;
0445     if(temp16 & 0x40) flag_low |= 0x1000;
0446     hex4low(flag_low);
0447     buffer[4] = 0;
0448     buffer[5] = 0x38;
0449     buffer[6] = 0x71;
0450     buffer[7] = 0;
0451 }
```

```
0452
0453  reg_flag_high()
0454  {
0455      flag_high=0;
0456      temp16= user_flag;
0457      if(temp16 & 0x80)  flag_high |= 1;
0458      if(temp16 & 0x200) flag_high |= 0x10;
0459      if(temp16 & 0x400) flag_high |= 0x100;
0460      if(temp16 & 0x800) flag_high |= 0x1000;
0461      hex4low(flag_high);
0462      buffer[4] = 0;
0463      buffer[5] = 0x76;
0464      buffer[6] = 0x71;
0465      buffer[7] = 0;
0466  }
0467
0468  reg_display()
0469  {
0470      if(key == 0)    reg_ax();
0471      if(key == 1)    reg_bx();
0472      if(key == 2)    reg_cx();
0473      if(key == 3)    reg_dx();
0474      if(key == 4)    reg_sp();
0475      if(key == 5)    reg_bp();
0476      if(key == 6)    reg_si();
0477      if(key == 7)    reg_di();
0478      if(key == 8)    reg_cs();
0479      if(key == 9)    reg_ds();
0480      if(key == 0xa)  reg_ss();
0481      if(key == 0xb)  reg_es();
0482      if(key == 0xc)  reg_flag();
0483      if(key == 0xd)  reg_flag_high();
0484      if(key == 0xe)  reg_flag_low();
0485  }
0486
0487  key_reg()
0488  {
0489      buffer[7] = 0;
0490      buffer[6] = 0x50;
0491      buffer[5] = 0x79;
0492      buffer[4] = 0x6f;
```

```
0493     buffer[3] = 0;
0494     buffer[2] = 0;
0495     buffer[1] = 0;
0496     buffer[0] = 0;
0497     state = 3; // register display state = 3 with hex key
0498 }
0499
0500 key_go()
0501 {
0502     if(state == 1 || state == 2)
0503     {
0504         asm
0505         {
0506             MOV AX,SP
0507             MOV save_stack,AX
0508             MOV AX,user_sp
0509             MOV SP,AX           ; LOAD SP WITH USER STACK
0510             MOV AX,user_cs
0511             PUSH AX
0512             MOV AX,PC ;user_ip
0513             PUSH AX
0514             MOV AX,user_di
0515             PUSH AX
0516             MOV AX,user_si
0517             PUSH AX
0518             MOV AX,user_bp
0519             PUSH AX
0520             MOV AX,user_dx
0521             PUSH AX
0522             MOV AX,user_cx
0523             PUSH AX
0524             MOV AX,user_bx
0525             PUSH AX
0526             MOV AX,user_ax
0527             PUSH AX
0528             POP AX
0529             POP BX
0530             POP CX
0531             POP DX
0532             POP BP
0533             POP SI
```

```
0534      POP DI
0535      RETF          ;JUMP TO USER PROGRAM WITH CS = 0000
0536  }
0537  }
0538
0539  if(state == 5)
0540  {
0541      destination = display_PC;
0542      temp = destination - (start + 3);
0543      hex4(temp);
0544      dptr = start + 1;
0545      *(dptr) = (char)temp & 0xff;
0546      *(dptr+1)= (char)(temp >>= 8);
0547      PC = start + 1;
0548      read_memory();
0549      state = 1;
0550  }
0551  if(state == 7)
0552  {
0553      destination = display_PC;
0554      temp = destination-(start + 2);
0555      hex4(temp);
0556      dptr = start + 1;
0557      *(dptr)= (char)(temp & 0xff);
0558      PC = start + 1;
0559      read_memory();
0560      state = 1;
0561  }
0562  }
0563
0564  service_break()
0565  {
0566      asm
0567      {
0568          MOV user_ax,AX
0569          MOV user_bx,BX
0570          MOV user_cx,CX
0571          MOV user_dx,DX
0572          MOV user_bp,BP
0573          MOV user_si,SI
0574          MOV user_di,DI
```

```
0575     POP AX
0576     MOV save_PC,AX
0577     MOV PC,AX
0578     POP AX
0579     MOV user_cs,AX
0580     POP AX
0581     MOV user_flag,AX
0582     MOV user_sp,SP
0583     CALL read_memory
0584     MOV AX,save_stack
0585     INC AX
0586     INC AX
0587     MOV SP,AX
0588     RET
0589 }
0590 }
0591
0592 key_step()
0593 {
0594     asm
0595     {
0596         MOV AX,SP
0597         MOV save_stack,AX
0598         MOV AX,user_sp
0599         MOV SP,AX                ; LOAD SP WITH USER STACK
0600         PUSHF
0601         MOV BP,SP
0602         OR >[BP],#$0100        ; set trap flag
0603         MOV AX,user_cs
0604         PUSH AX
0605         MOV AX,PC                ; user_ip
0606         PUSH AX
0607         MOV AX,user_di
0608         PUSH AX
0609         MOV AX,user_si
0610         PUSH AX
0611         MOV AX,user_bp
0612         PUSH AX
0613         MOV AX,user_dx
0614         PUSH AX
0615         MOV AX,user_cx
```

```
0616     PUSH AX
0617     MOV AX,user_bx
0618     PUSH AX
0619     MOV AX,user_ax
0620     PUSH AX
0621     POP AX
0622     POP BX
0623     POP CX
0624     POP DX
0625     POP BP
0626     POP SI
0627     POP DI
0628     IRET                                ; JUMP TO USER PROGRAM WITH IRET,
0629 }                                       ; TRAP FLAG WAS SET
0630 }
0631
0632 offset16()
0633 {
0634     buffer[0] = 0x6d;
0635     buffer[1] = 0;
0636     display_PC = PC;
0637     state = 4; // enter word address
0638     hit = 0;
0639 }
0640
0641 offset8()
0642 {
0643     buffer[0] = 0x6d;
0644     buffer[1] = 0;
0645     display_PC = PC;
0646     state = 6; // enter word address
0647     hit=0;
0648 }
0649
0650 word_enter()
0651 {
0652     if(hit == 0) display_PC=0;
0653     {
0654         hit = 1;
0655         display_PC <=<= 4;
0656         display_PC |= key;
```

```
0657     hex4(display_PC);
0658     dot_address();
0659 }
0660 }
0661
0662 key_test()
0663 {
0664     test_sound();
0665     enable();
0666     for(;;)
0667         ;
0668 }
0669
0670 key_exe()
0671 {
0672     if(key > 15)
0673     {
0674         if(key == 0x13) key_address();
0675         if(key == 0x12) key_data();
0676         if(key == 0x17) key_plus();
0677         if(key == 0x16) key_minus();
0678         if(key == 0x10) key_PC();
0679         if(key == 0x21) key_go();
0680         if(key == 0x11) key_reg();
0681         if(key == 0x18) insert();
0682         if(key == 0x19) cut_byte();
0683         if(key == 0x14) offset16();
0684         if(key == 0x15) offset8();
0685         if(key == 0x20) key_step();
0686         if(key == 0x22) key_test();
0687     }
0688     else
0689     {
0690         if(state == 1) hex_address();
0691         if(state == 2) data_hex();
0692         if(state == 3) reg_display();
0693         if(state == 4 || state == 5 || state == 6 || state== 7)
0694             word_enter();
0695     }
0696 }
0697
```

```
0698 // return internal code hex keys and function keys
0699 char key_code(char n)
0700 {
0701     if(n == 3) return 0;
0702     if(n == 0x22) return 1;
0703     if(n == 0x1c) return 2;
0704     if(n == 0x16) return 3;
0705     if(n == 2) return 4;
0706     if(n == 0xf) return 5;
0707     if(n == 0x21) return 6;
0708     if(n == 0x1b) return 7;
0709     if(n == 7) return 8;
0710     if(n == 1) return 9;
0711     if(n == 0x1a) return 0xa;
0712     if(n == 0x14) return 0xb;
0713     if(n == 0xc) return 0xc;
0714     if(n == 6) return 0xd;
0715     if(n == 0) return 0xe;
0716     if(n == 0x19) return 0xf;
0717     if(n == 0x13) return 0x10;
0718     if(n == 0xe) return 0x11;
0719     if(n == 0x15) return 0x12;
0720     if(n == 0x10) return 0x13;
0721     if(n == 0x23) return 0x14;
0722     if(n == 0x1d) return 0x15;
0723     if(n == 0x17) return 0x16;
0724     if(n == 0x11) return 0x17;
0725     if(n == 9) return 0x18;
0726     if(n == 8) return 0x19;
0727     if(n == 0x12) return 0x20;
0728     if(n == 0xd) return 0x21;
0729     if(n == 0x18) return 0x22;
0730     if(n == 0x30) return 0x23;
0731 }
0732
0733 chk_serial()
0734 {
0735     if(chkchr() == 0x0d)
0736     {
0737         out(port2, 0xff);
0738         out(port1, 0x00);
```

```
0739     asm {
0740         JMP $c000
0741     }
0742     // out(gpio1,0xaa);
0743 }
0744 }
0745
0746 delay(int d)
0747 {
0748     for(x = 0; x < d; x++)
0749         ;
0750 }
0751
0752 char scan()
0753 {
0754     k=1;
0755     u=0;
0756     key = -1; // if no key pressed key=-1
0757     q = 0;    // key code
0758     for(i = 0; i < 8; i++)
0759     {
0760         out(port1, ~k); // write digit
0761         out(port2, buffer[i]); // write segment
0762         // delay(10);
0763         if(buffer[i] != 0x06 && buffer[i] != 0x86)
0764             delay(10);
0765         else
0766             delay(1);
0767         out(port2, 0);
0768         o = in(port0); // read keypad
0769         for(n = 0; n < 6; n++)
0770         {
0771             if((o & 1) == 0)
0772                 key = q;
0773             else
0774                 q++;
0775             o >>= 1;
0776         }
0777         k <<= 1;
0778     }
0779     return key;
```

```
0780  }
0781
0782  // scan1 repeat scan display and keyboard
0783  void scan1()
0784  {
0785      temp2=0;
0786      while((scan() != -1) && ((in(port0) & 0x40) != 0))
0787      {
0788          chk_serial();
0789      }
0790      delay(10);
0791      while(scan() == -1)
0792      {
0793          chk_serial();
0794          auto_key = 0;
0795      }
0796      delay(10);
0797      key = scan();
0798      // check key in range
0799      if(key >=0 && key < 0x31)
0800      {
0801          key = key_code(key);
0802          key_exe();
0803          // out(gpio1,key);
0804      }
0805  }
0806
0807  service_timer()
0808  {
0809      out(gpio1,i++);
0810      asm
0811      {
0812          POP BP
0813          IRET
0814      }
0815  }
0816
0817  test_sound()
0818  {
0819      for(j = 0; j < 2000; j++)
0820      {
```

```
0821     out(port3,j);
0822     delay(1);
0823 }
0824 out(port3,0); // turn off
0825 }
0826
0827 insert_vectors()
0828 {
0829     asm
0830     {
0831         MOV AX,#service_break + 3
0832         MOV $000c,AX ; SERVICE ADDRESS FOR INT 3
0833         MOV AX,$F000 ; CS SEGMENT FOR MONITOR
0834                     ; ROM F000, RAM 0000
0835         MOV $000E,AX
0836         ; INSERT VECTOR FOR INT 1 TRAP
0837         MOV AX,#service_break+3
0838         MOV $0004,AX ; SERVICE ADDRESS FOR INT 2
0839         MOV AX,$F000 ; CS SEGMENT FOR MONITOR
0840                     ; ROM F000, RAM 0000
0841         MOV $0006,AX
0842         ; INSERT VECTOR FOR EXTERNAL INTR
0843         MOV AX,#service_timer
0844         MOV $3FC,AX
0845         MOV AX,$F000 ; CS SEGMENT FOR
0846                     ; ROM F000, RAM 0000
0847         MOV $3FE,AX
0848     }
0849 }
0850
0851 init_UART()
0852 {
0853     out(UART + 6, 0x83);
0854     out(UART + 0, 16); // 9600 8n1
0855     out(UART + 2, 0);
0856     out(UART + 6, 3);
0857     out(UART + 2, 0);
0858     out(UART + 8, 3);
0859 }
0860
0861
```

```
0862  main()
0863  {
0864      disable();      // disable interrupts
0865      out(port3, 0); // turn off speaker
0866      PC = 0x400;
0867      save_PC = 0x400;
0868      user_sp = 0xff00;
0869      user_cs = 0x0000;
0870      user_ds = 0x0000;
0871      user_ss = 0x0000;
0872      user_es = 0x0000;
0873      init_UART(); // 9600 8n1
0874      strcpy(buffer2, "\r\n8086 MICROPROCESSOR KIT 2018");
0875      putstr("\r\n8086 MICROPROCESSOR KIT 2018");
0876      if(check_LCD() == 0)
0877      {
0878          InitLcd();
0879          Puts("8086 MICROPROCESSOR KIT 2018");
0880          goto_xy(0, 1);
0881          Puts("256kB RAM, 256kB ROM, UART");
0882      }
0883      out(gpio1, 0); // turn off gpio1
0884      buffer[7] = 0;
0885      buffer[6] = 0;
0886      buffer[5] = convert[8];
0887      buffer[4] = convert[0];
0888      buffer[3] = convert[8];
0889      buffer[2] = convert[6];
0890      buffer[1] = 0;
0891      buffer[0] = 0;
0892      insert_vectors();
0893      while(1)
0894      {
0895          scan1();
0896      }
0897  }
```

