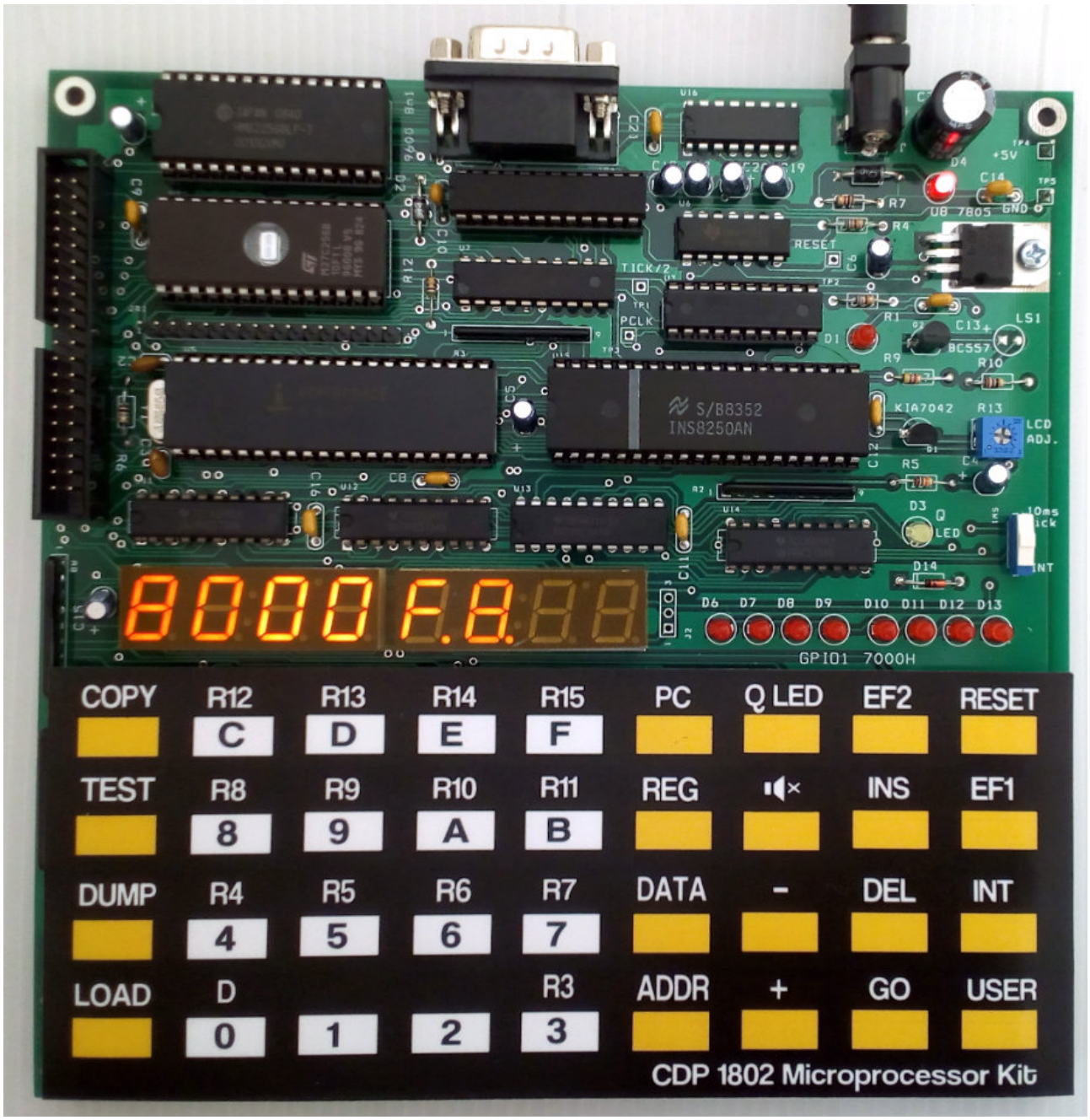
CDP1802 Microprocessor Kit
User's Manual



Rev 1.0, October, 2018

CDP1802 MICROPROCESSOR KIT

CONTENTS

OVERVIEW.....4

FUNCTIONAL BLOCK DIAGRAM.....4

HARDWARE LAYOUT.....5

KEYBOARD LAYOUT.....7

HARDWARE FEATURES.....9

MONITOR PROGRAM FEATURES.....9

MEMORY AND I/O MAPS.....10

PLD DECODER.….....11

GETTING STARTED.....12

HOW TO ENTER PROGRAM USING HEX CODE.....14

AUTO LOAD Subroutine DELAY.....16

GPIO1 LED.....17

RS232C PORT.....18

DATA FRAME for UART COMMUNICATION.....18

CONNECTING KIT TO TERMINAL.....19

EXPANSION BUS HEADER.....22

10ms TICK GENERATOR.....23

CONNECTING LCD MODULE.....24

LOGIC PROBE POWER SUPPLY.....25

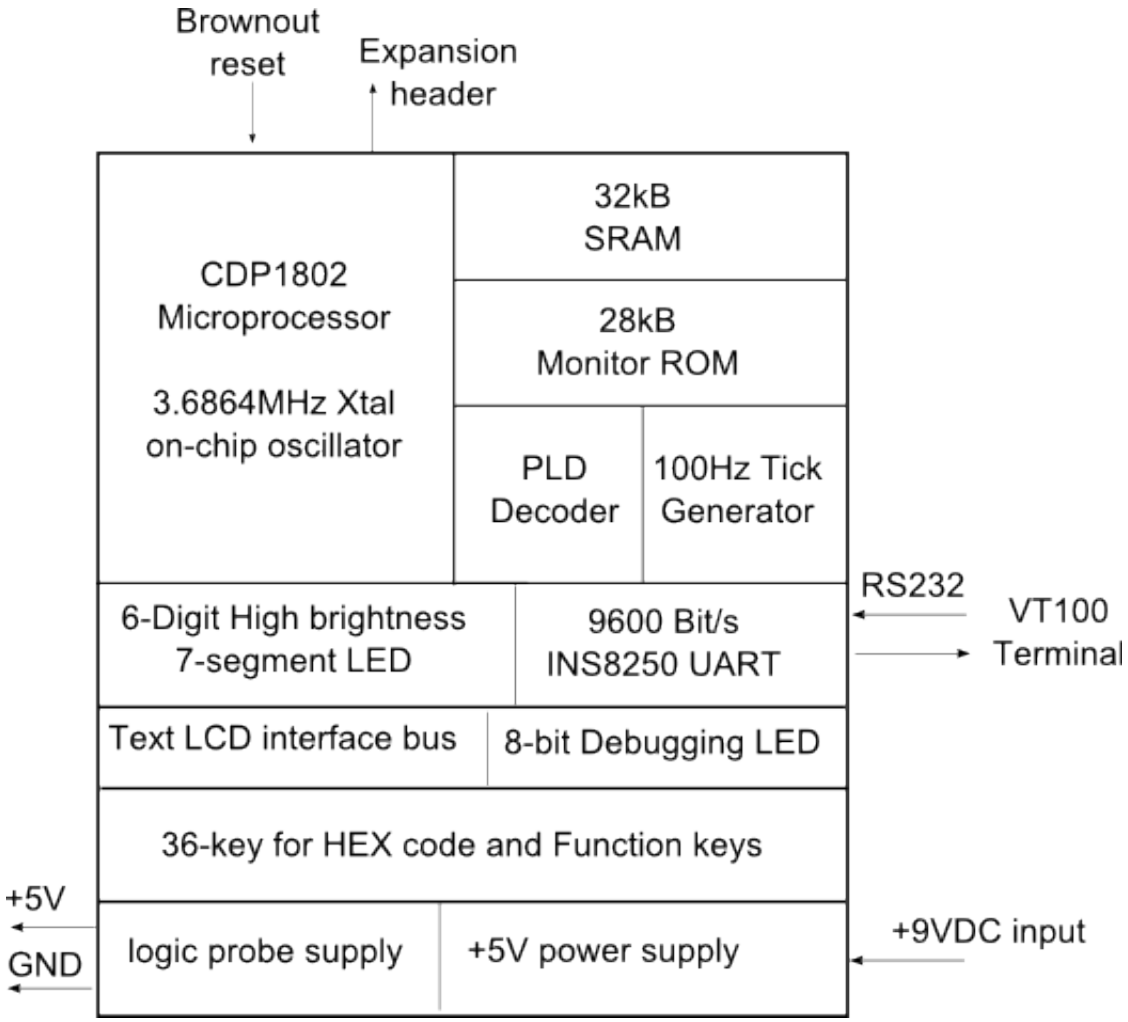
HARDWARE SCHEMATIC, BOM

MONITOR PROGRAM LISTINGS

OVERVIEW

The CDP1802 Microprocessor Kit is a new design single board educational computer. The kit is based on CDP1802 8-bit CMOS microprocessor. System memory are 28kB monitor ROM, 32kB user RAM and 4kB memory mapped I/O. The kit provides HEX keypad and 6-digit seven segment display. Students can enter CDP1802 instructions using HEX code to the memory and run it directly. The kit also provides 10ms tick generator, 9600 UART and expansion header.

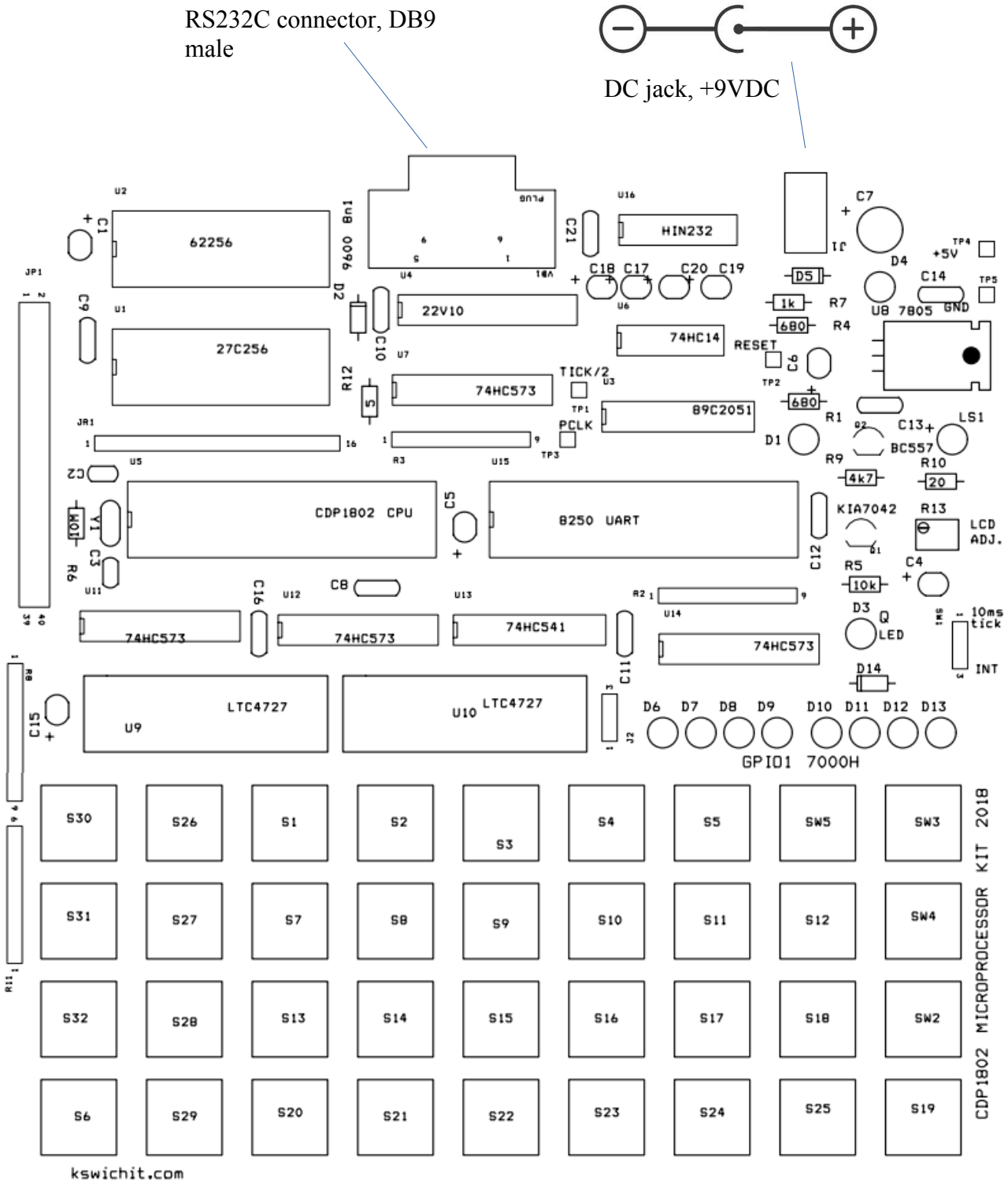
FUNCTIONAL BLOCK DIAGRAM



Notes

1. CDP1802 is CMOS microprocessor.
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.
6. Hardware UART is INS8250, 9600 bit/s.

HARDWARE LAYOUT



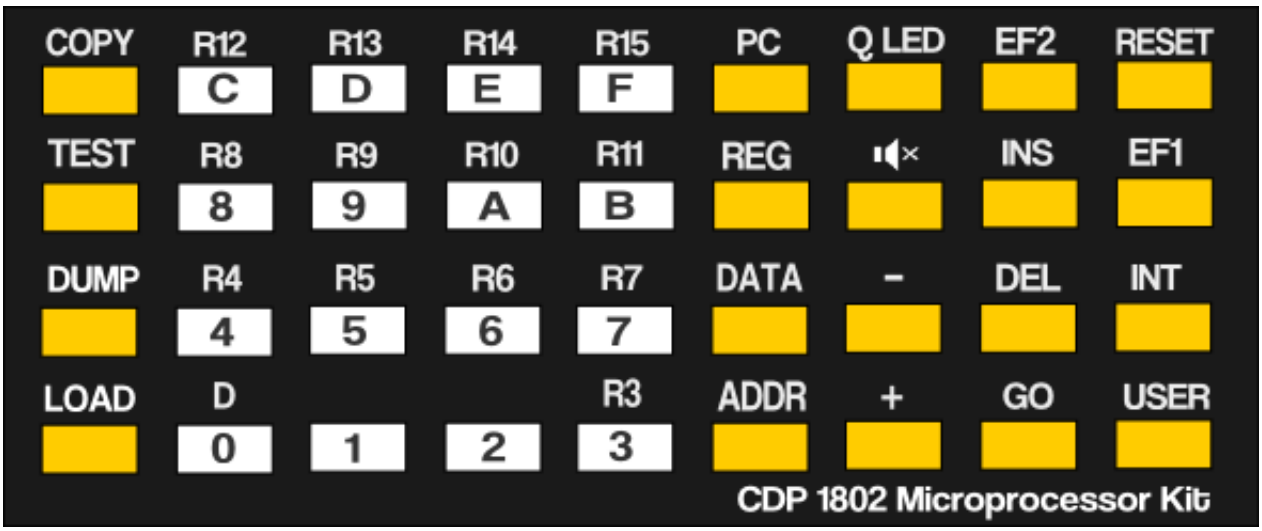
Safety Information

1. Plugging or removing the LCD module must be done when the kit is powered off!

2. AC adapter should provide approx. +9VDC, higher voltage will cause the voltage regulator chip becomes hot.

3. The kit has diode protection for wrong polarity of adapter jack. If the center pin is not the positive (+), the diode will be reverse bias, preventing wrong polarity feeding to voltage regulator.

KEYBOARD LAYOUT



HEX keys Hexadecimal number 0 to F with associated user registers, D, R3-R15 (use with key REG).

CPU control keys

RESET Reset the CPU, the CDP1802 will JUMP to location 0000.

INT Make INTRPT pin to logic low, for experimenting with interrupt process

EF1 Make EF1 pin to logic low, for testing EF1 conditional branch

EF2 Make EF2 pin to logic low, for testing EF2 conditional branch

EF3 and EF4 are available on expansion header.

Monitor function keys

PC Return current address to location 8000.

DATA Set entry mode of hex keys to Data field

ADDR Set entry mode of hex keys to Address field

GO Jump from monitor program to user code, R0 will be user program counter

- Decrement current display address by one

+ Increment current display address by one

Q LED Test Q output bit, blinking yellow LED

INS Insert byte, after current byte +512 locations will be shifted down

DEL Delete current byte, 512 locations will be shifted up

USER User key for monitor program customization

TEST Test 10ms tick, SW1 when set to 10ms tick, the gpio1 LED will be counting at 10Hz rate

COPY Copy block of memory, use with key + and GO, sequence will be START, END, DESTINATION, then GO

DUMP Dump memory contents to 9600 terminal, will need RS232 cross cable and PC running terminal emulator

LOAD Load Intel HEX file, 1ms character and line delay will be needed.

HARDWARE FEATURES

Hardware features:

- CPU: Intersil CDP1802 CMOS Microprocessor @3.6864MHz clock
- Memory: 32kB SRAM, 28kB EPROM, 4kB memory mapped I/O
- Memory and I/O Decoder chip: Programmable Logic Device GAL22V10D
- Display: high brightness 6-digit 7-segment LED
- Keyboard: 36 keys
- RS232 port: INS8250 UART, 9600 bit/s 8n1
- Debugging LED: 8-bit GPIO1 LED at location $7000
- Q LED: high brightness yellow color dot LED for Q output bit
- Tick: 10ms tick produced by 89C2051 for time trigger experiment
- Text LCD interface: direct CPU bus interface text LCD
- Brownout reset: KIA7042 reset chip for power brownout reset
- Expansion header: 40-pin header

MONITOR PROGRAM FEATURES

MONITOR program features:

- Simple hex code entering
- Insert and Delete byte
- User registers: D, R3-R15, used for saving CPU registers after BREAK
- Copy block of memory
- Intel HEX file downloading
- Memory dump
- Beep ON/OFF
- TEST 10ms tick

MEMORY AND I/O MAPS

The kit provides two spaces of memory, i.e. 1) 32kB RAM, 2) 28 monitor ROM.

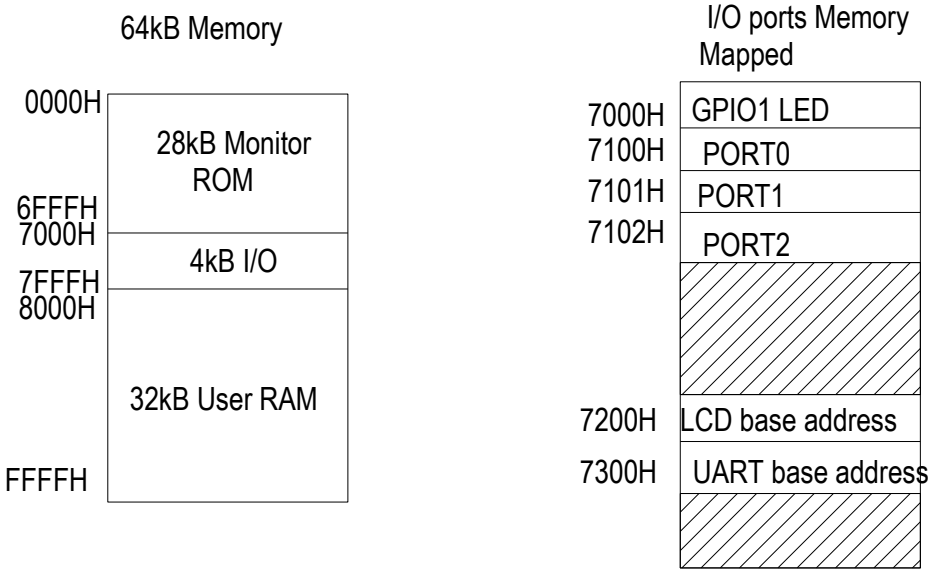
I/O space uses 4kB Memory mapped. These I/O locations can be accessed using memory READ/WRITE instructions directly.

Monitor ROM is placed from location 0000H to 6FFFH. RAM starts at 8000H to FFFFH

I/O ports are located from 7000H to 7FFFH.

GPIO1 LED is located at 7000H. User can use instruction that writes 8-bit data with 16-bit address easily, e.g.

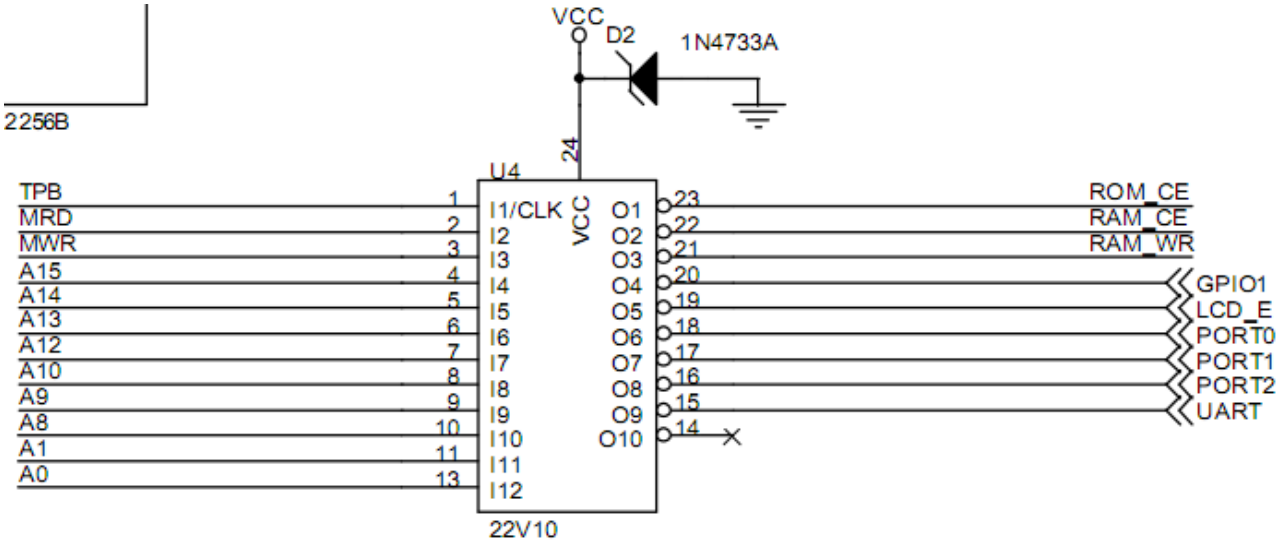
```
8000   F8 70 B4 F8    LOAD R4, GPIO1 ; LOAD R4 WITH GPIO1 LED LOCATION
8004   00 A4

8006   F8 01          LDI 1         ; LOAD D WITH 1
8008   54             STR R4      ; WRITE D TO GPIO1
8009   00             IDL           ;  BREAK
```



Note: For LCD and UART internal registers, check at the schematic and monitor source code.

PLD DECODER

The programmable logic device, GAL22V10D is memory and I/O decoder. The chip accepts addresses and control signals from CDP1802, I1 to I12 and provides output enable signals at O1 to O9.



The decoder logic was implemented with PLD equations.

```
!ROM_CE = !MRD & ADDRESS:[0000..6FFF];

!RAM_CE = ADDRESS:[8000..FFFF];

!RAM_WR = !MWR & ADDRESS:[8000..FFFF];

!GPIO1 = A15 # !A14 # !A13 # !A12 # A10 # A9 # A8 # A1 # A0 # MWR; /* 7000H */

PORT0 = A15 # !A14 # !A13 # !A12 # A10 # A9 # !A8 # A1 # A0 # MRD; /* 7100H */

!PORT1 = A15 # !A14 # !A13 # !A12 # A10 # A9 # !A8 # A1 # !A0 # MWR; /* 7101H */

!PORT2 = A15 # !A14 # !A13 # !A12 # A10 # A9 # !A8 # !A1 # A0 # MWR; /* 7102H */

!LCD_E = A15 # !A14 # !A13 # !A12 # A10 # !A9 # A8 # (MWR & MRD); /* 7200H */

UART = A15 # !A14 # !A13 # !A12 # A10 # !A9 # !A8; /* 7300H */
```

PLD equation was assembled and translated into JEDEC file using WinCupL. The JEDEC file will be used to program to the PLD chip.

GETTING STARTED

The kit accepts DC power supply with minimum voltage of +7.5V. It draws DC current approx. 180mA. 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.

If your adapter is adjustable output voltage, try with approx. +9V. Higher voltage will make higher power loss at the voltage regulator, 7805. Dropping voltage across 7805 is approx. +2V. To get +5VDC for the kit, we thus need DC input >+7.5V.

When power up, we will see the cold boot message 1802.

Press PC, the display will show HOME location at 8000. The data field will show its content.

HOW TO ENTER PROGRAM USING HEX CODE

Let us try enter HEX CODE of the example program to the memory and test it. We write the program with CDP1802 instructions.

Address	Hex code	Label	Instruction	comment
8000	F8 70 B4 F8 00 A4	MAIN	LOAD R4, 7000	Load R4 with 7000
8006	F8 01		LDI 1	Load D with 1
8008	54		STR R4	Write to 7000
8009	00		IDL	Wait for DMA or interrupt

Our test program has only four instructions.

The first instruction is

`LOAD R4,GPIO1`

Load R4 register with the 7000

This instruction has six bytes hex code i.e., F8, 70, B4, F8, 00, A4.

[To load 16-bit value to R4, CDP1802 will need:

LDI 70
PHI R4
LDI 00
PLO R4
]

The 2nd instruction is

`LDI 1` load immediate value 1, to D register. The instruction's machine code is F8. The immediate 8-bit is 01.

The 3rd instruction is STR R4, write register D to memory location pointed to by R4.

The last instruction is IDL, with hex code 00. It makes CPU to wait for DMA or interrupt process. We can use it to break here.

This test code has only 10 bytes, F8, 70, B4, F8, 00, A4, F8, 01, 54, 00.

The first byte will be entered to location 8000. And the following bytes will be entered at 8001, 8002, 8003, 8004, to the last byte at 8009.

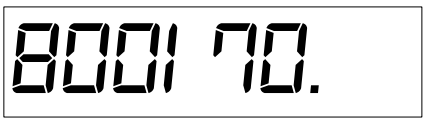
Let us see how to enter these codes into memory.

Step 1 Press RESET, PC, the display will show current memory address and its contents.

The location 8000 has data 00. There are small dots at the data field indicating the active field, ready for modifying the hex contents.

Step 2 Press key F and key 8. The new hex code F8 will be entered to the location 8000.

Step 3 Press key + to increment the location from 8000 to 8001. Then enter hex key 7, 0.



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.

To RUN the program, press PC then GO.

See what is happening at gpio1 LED?

Can you change the load value? How?

AUTO LOAD Subroutine DELAY

The kit provides automatically loaded DELAY subroutine on RESET. Every RESET will load the DELAY subroutine to the last page of user RAM.

```
F000   D0            RET_DELAY: SEP R0

F001   F8 64         DELAY:   LDI 100
F003   A6                     PLO R6

F004   F8 00         DELAY1:  LDI 0
F006   A7                     PLO R7
F007   27            DELAY2:  DEC R7
F008   87                     GLO R7
F009   3A 07                  BNZ DELAY2

F00B   26                 DEC R6
F00C   86                 GLO R6
F00D   3A 04              BNZ DELAY1

F00F   30 00              BR RET_DELAY
```

The subroutine provides delay for testing USER code.

Let us see another example:

```
8000   F8 F0 B3 F8          LOAD R3, F001  ; delay
8004   01 A3

8006   7B            LOOP:  SEQ      ; set q bit
8007   D3                   SEP R3   ; delay
8008   7A                   REQ      ; reset q bit
8009   D3                   SEP R3   ; delay
800A   30 06                BR LOOP  ; repeat
```

The kit has Q LED that shows Q logic. Simple program, Q LED blinking will need R3 loaded with F001.

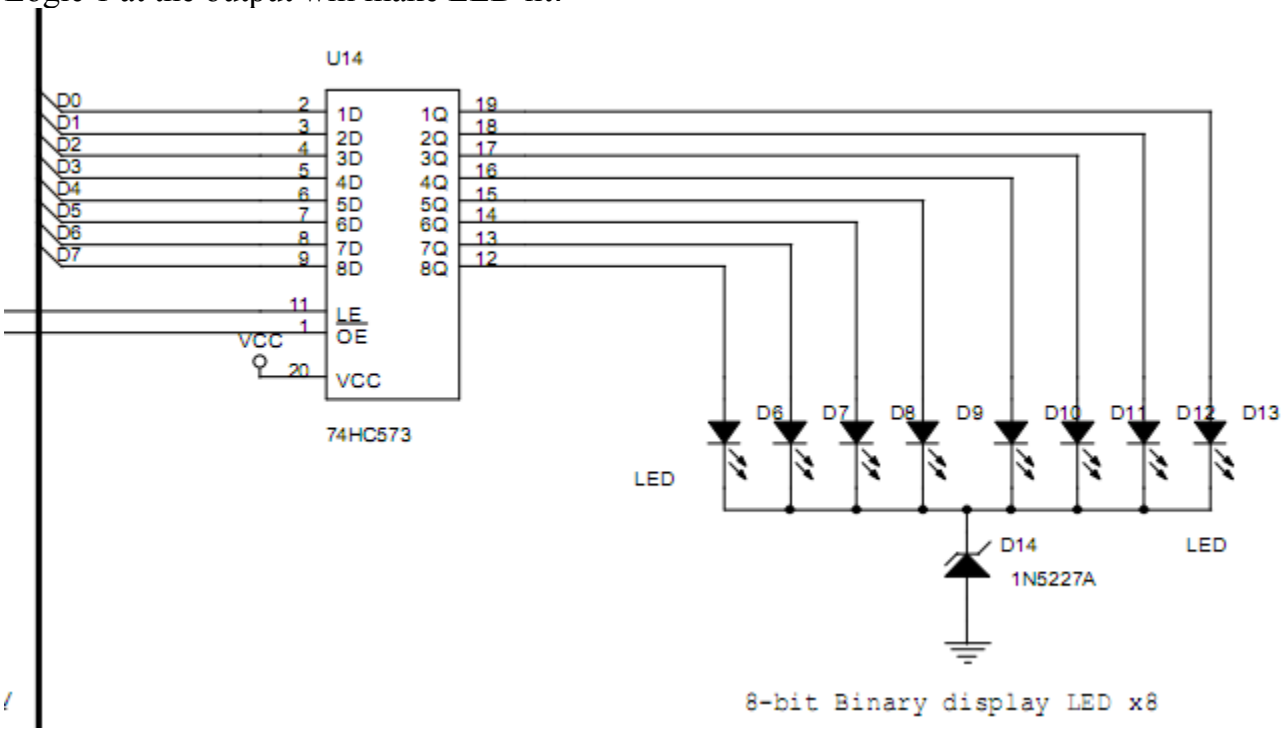
Let us try with hex key code entering. Press PC, then GO.

What is happening at Q LED?

We can change blinking rate easily at location F002. Try change it to 20. And press PC, GO.

GPIO1 LED

The kit provides a useful 8-bit binary display. It can be used to debug the program or code running demonstration. The I/O address is 7000. The output port is 8-bit data flip-flop. Logic 1 at the output will make LED lit.



Try below code for testing how to use gpio1 LED.

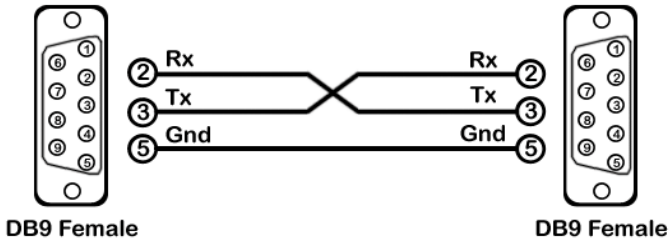
```
8000   F8 F0 B3 F8          LOAD R3, DELAY ; F001
8004   01 A3
8006   F8 70 B4 F8          LOAD R4, GPIO1 ; 7000
800A   00 A4

800C   15            LOOP:  INC R5     ; INCREMENT R5
800D   85                   GLO R5     ; GET LOW BYTE R5
800E   54                   STR R4     ; WRITE D TO GPIO1
800F   D3                   SEP R3     ; CALL DELAY
8010   30 0C                BR LOOP    ; JUMP TO LOOP
```

Enter the hex code and run it.

Can you change counting rate faster? How?

RS232C PORT

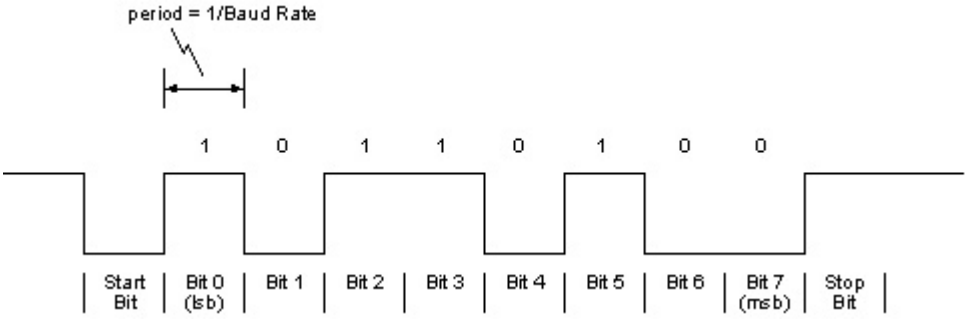
The RS232C port is for serial communication. We can use the RS232 cross cable or null MODEM cable to connect between the kit and terminal. The connector for both sides are DB9 female. We may build it or buying from computer stores.



For new PC or laptop computer without the RS232 port. It has only USB port, we may have the RS232C port by using the USB to RS232 converter.

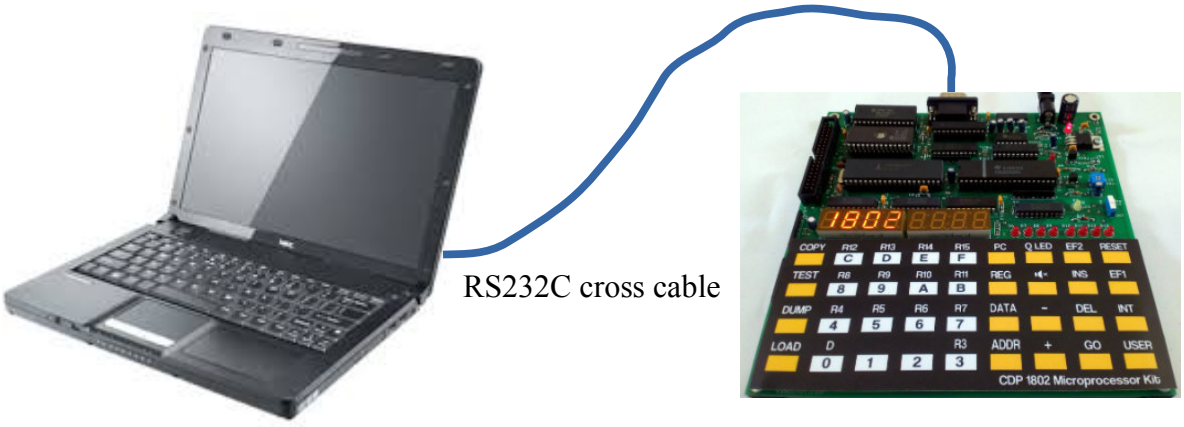
DATA FRAME for UART COMMUNICATION

Serial data that communicated between kit and terminal is asynchronous format. The CDP1802 kit has UART chip. The kit functions key has commands for HEX file downloading and memory dumping. Bit stream includes START bit, 8-data bit and one STOP bit. Bit period is 1/9600.



CONNECTING KIT TO TERMINAL

We can connect the CDP1802 kit to a terminal by RS232C cross cable. You may download free terminal program, teraterm from this URL, http://ttssh2.sourceforge.jp/index.html.en



VT100 Terminal

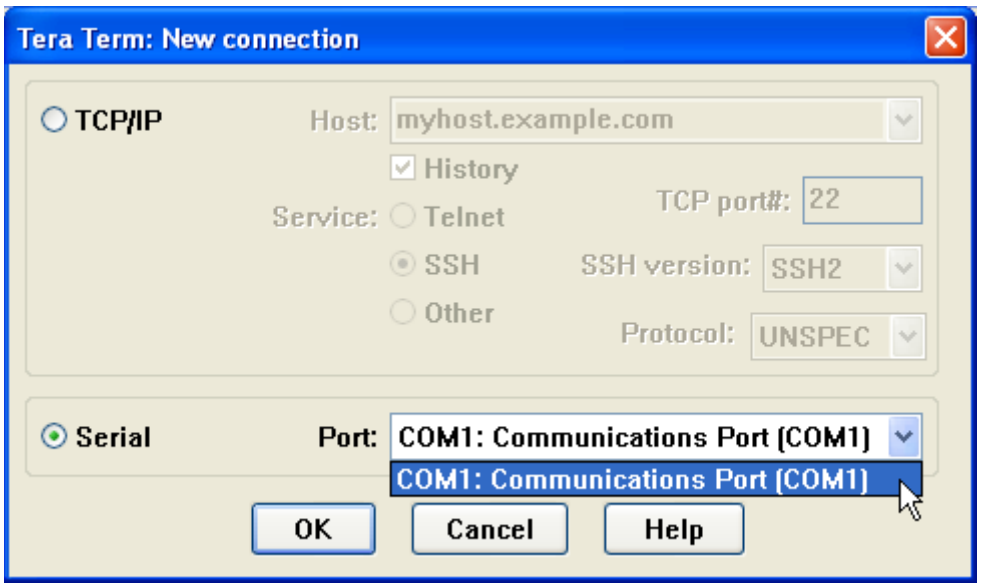
CDP1802 Kit

The example shows connecting laptop with COM1 port to the RS232C port of the CDP1802 kit. New laptop that has no COM port, we may use the USB-RS232 adapter for converting the USB port to RS232 port.

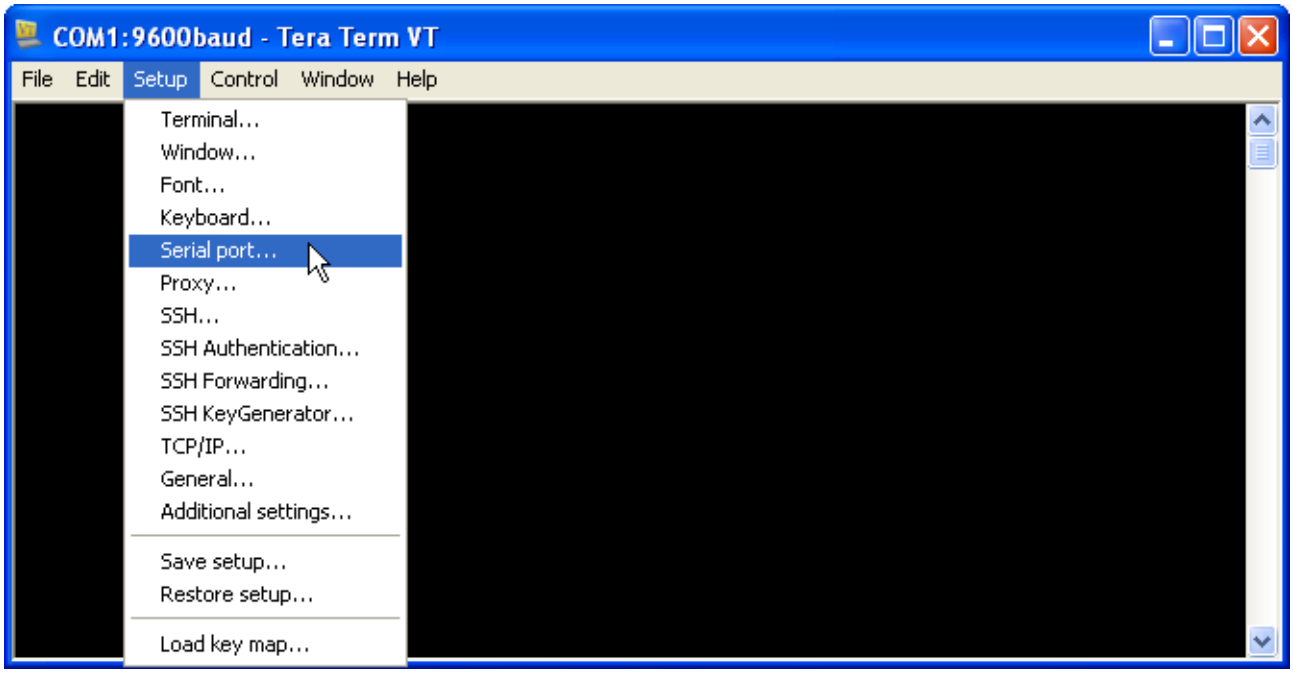
To download Intel hex file that generated from the assembler or c compiler, set serial port speed to 9600 bit/s, 8-data bit, no parity, no flow control, one stop bit.

One ms delay for character and line.

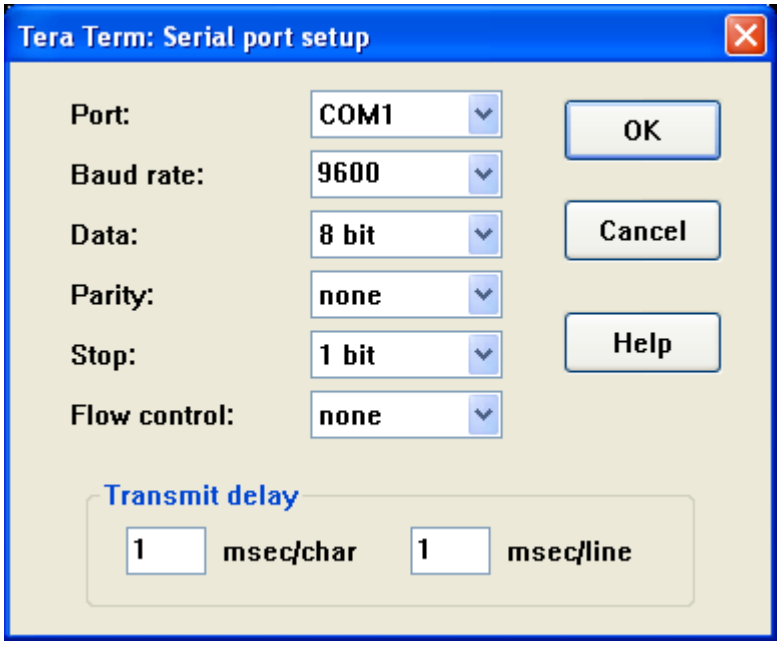
Step 1 Run teraterm, then click at Serial connection.



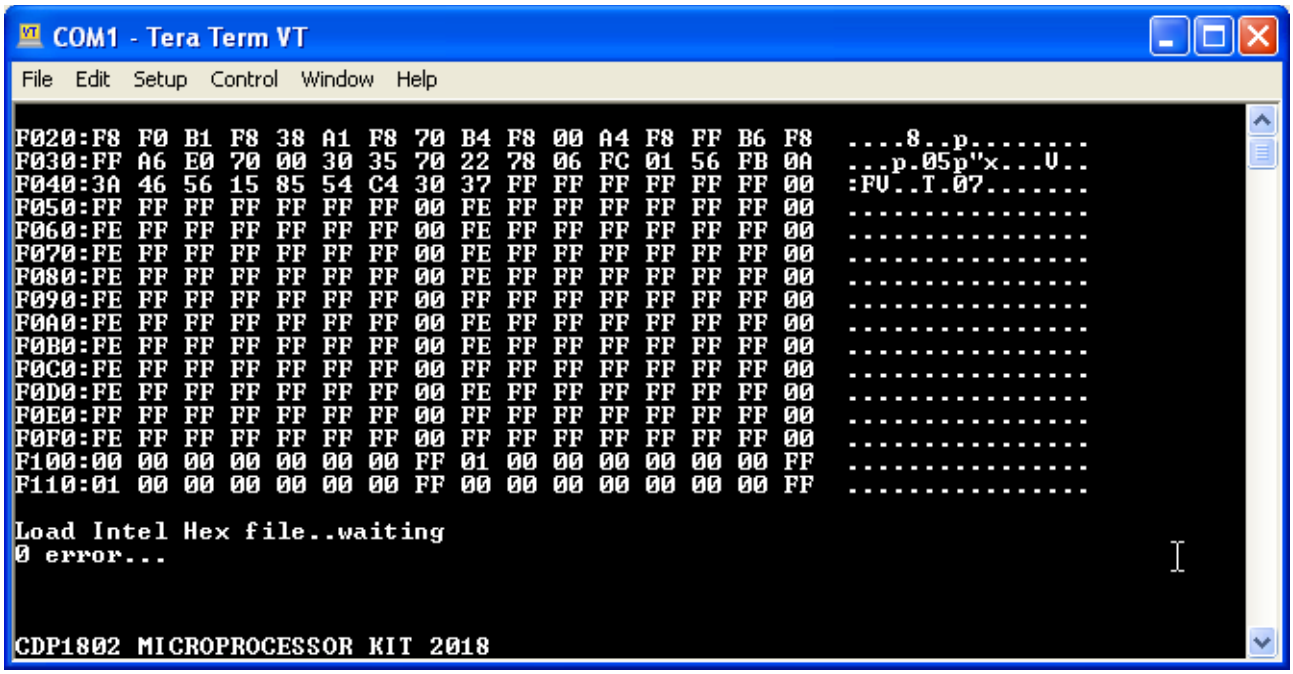
Step 2 Click setup>Serial port.



Step 3 Set serial port speed to 9600 and format as shown below.



Step 4 Press DUMP, the kit will print memory contents.



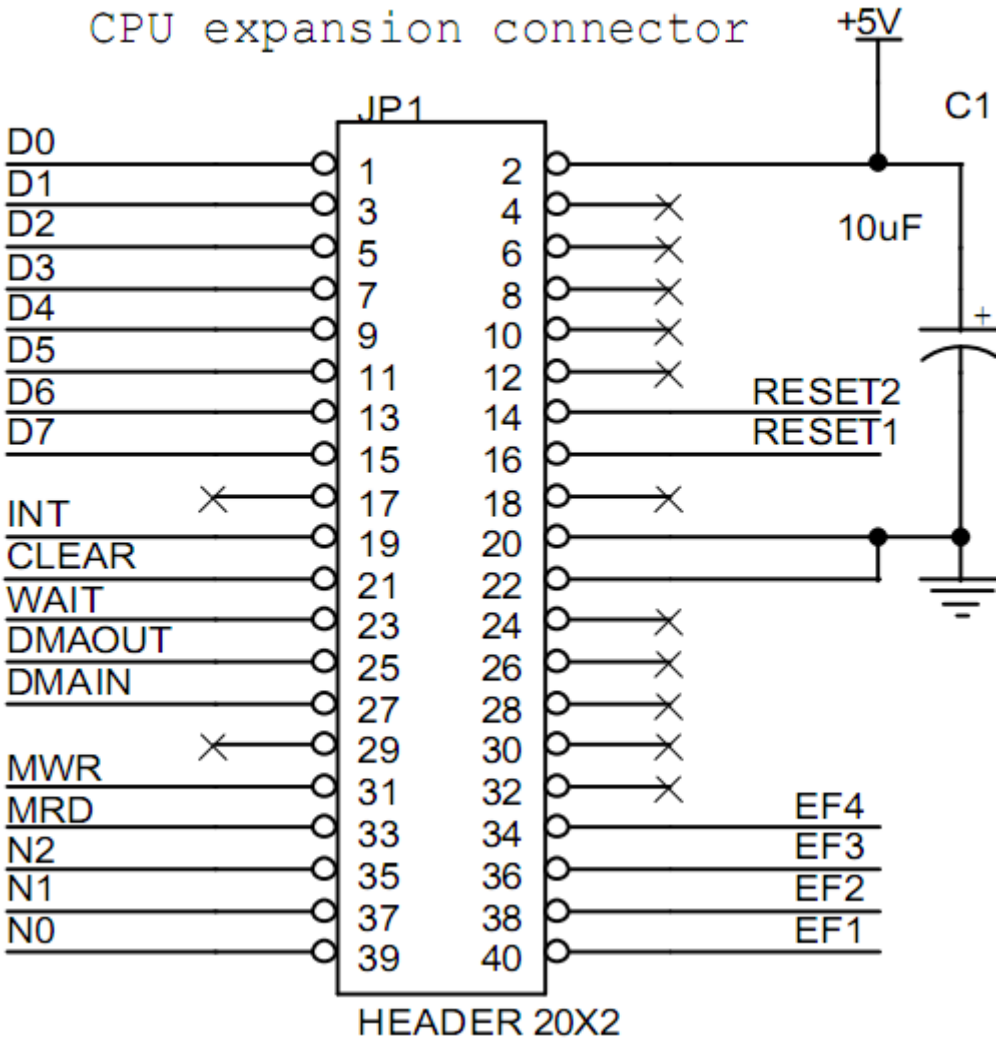
Key LOAD will wait for Intel hex file.

Click File>Send File..> then select the HEX file to be sent, click OPEN.

The gpio1 LED will show byte being received. When completed, the kit display will be turned on.

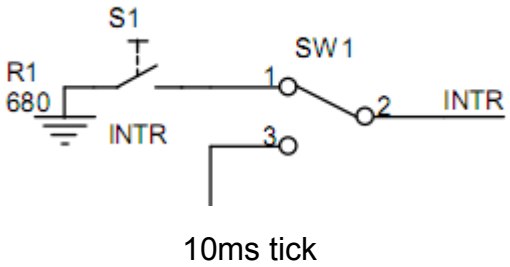
EXPANSION BUS HEADER

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. CDP1802 provides N0,N1 and N2 for I/O interface.



10ms TICK GENERATOR

SW1 is a selector for interrupt source between key INTR or 10ms tick produced by 89C2051 microcontroller. Tick generator is software controlled using timer0 interrupt in the 89C2051 chip. The active low tick signal is sent to P3.7. For tick running indicator, P1.7 drives D2 LED.



Tick is a 10ms periodic signal for triggering the CDP1802 INTRPT pin. When select SW1 to Tick, the 8080 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.

NOTE: Key TEST, will use 10ms tick for making binary counting at 10Hz rate.

CONNECTING LCD MODULE

JR1 is 16-pin header for connecting the LCD module. Any text LCD with HD44780 compatible controller can be used. R12 is a current limit resistor for back-light. R13 is trimmer POT for contrast adjustment. The LCD module is interfaced to the CDP1802 bus directly. The command and data registers are located in memory space having address from 7200H to 7203H.

Be advised that plugging or removing the LCD module must be done when the kit is powered off.

Text LCD module accepts ASCII codes for displaying the message on screen.
Without settings the LCD by software, no characters will be displayed. The first line will be black line by adjusting the R10 for contrast adjustment.

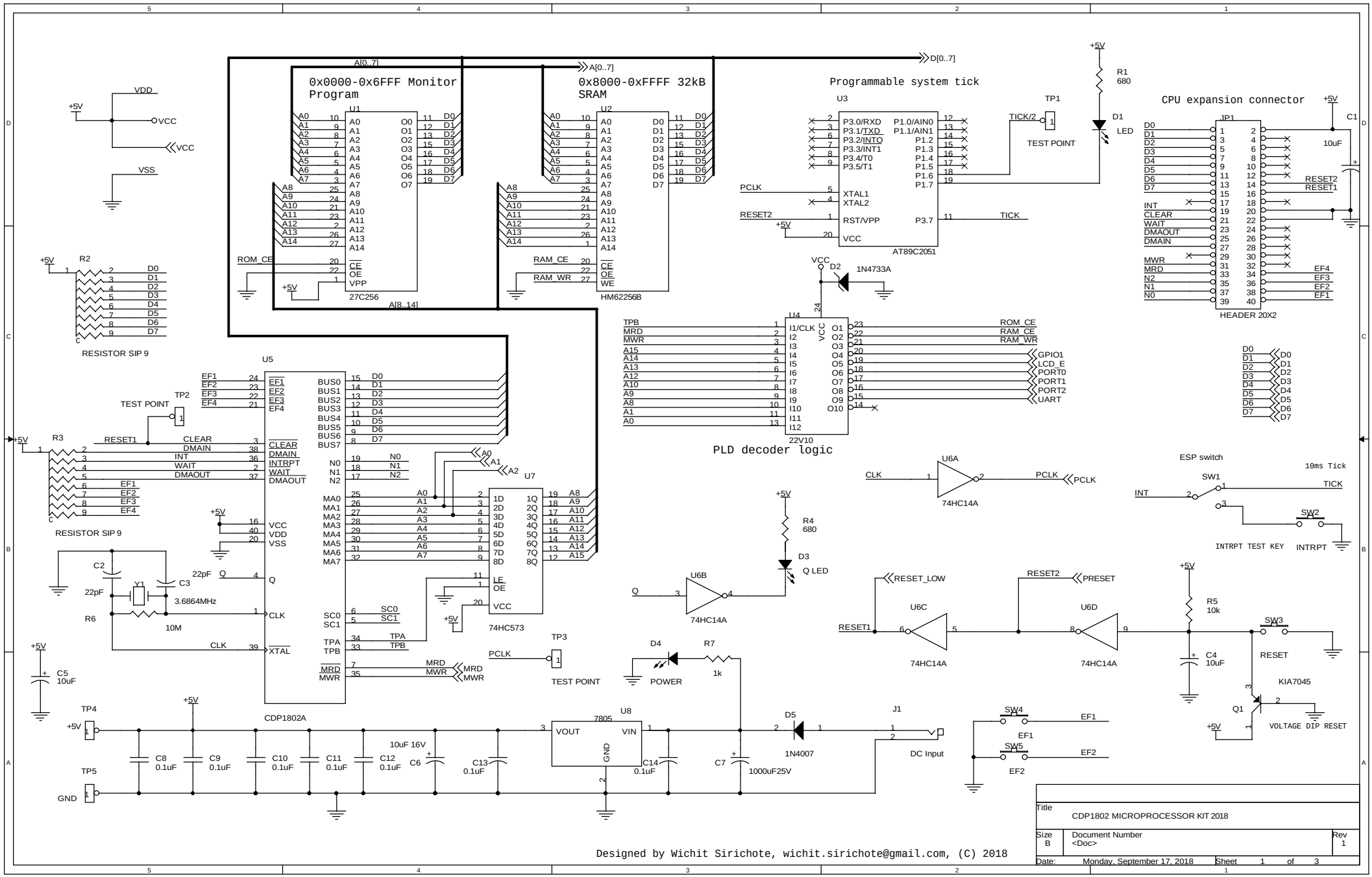
If the LCD module is connected, the system monitor will write CDP1802 to LCD.

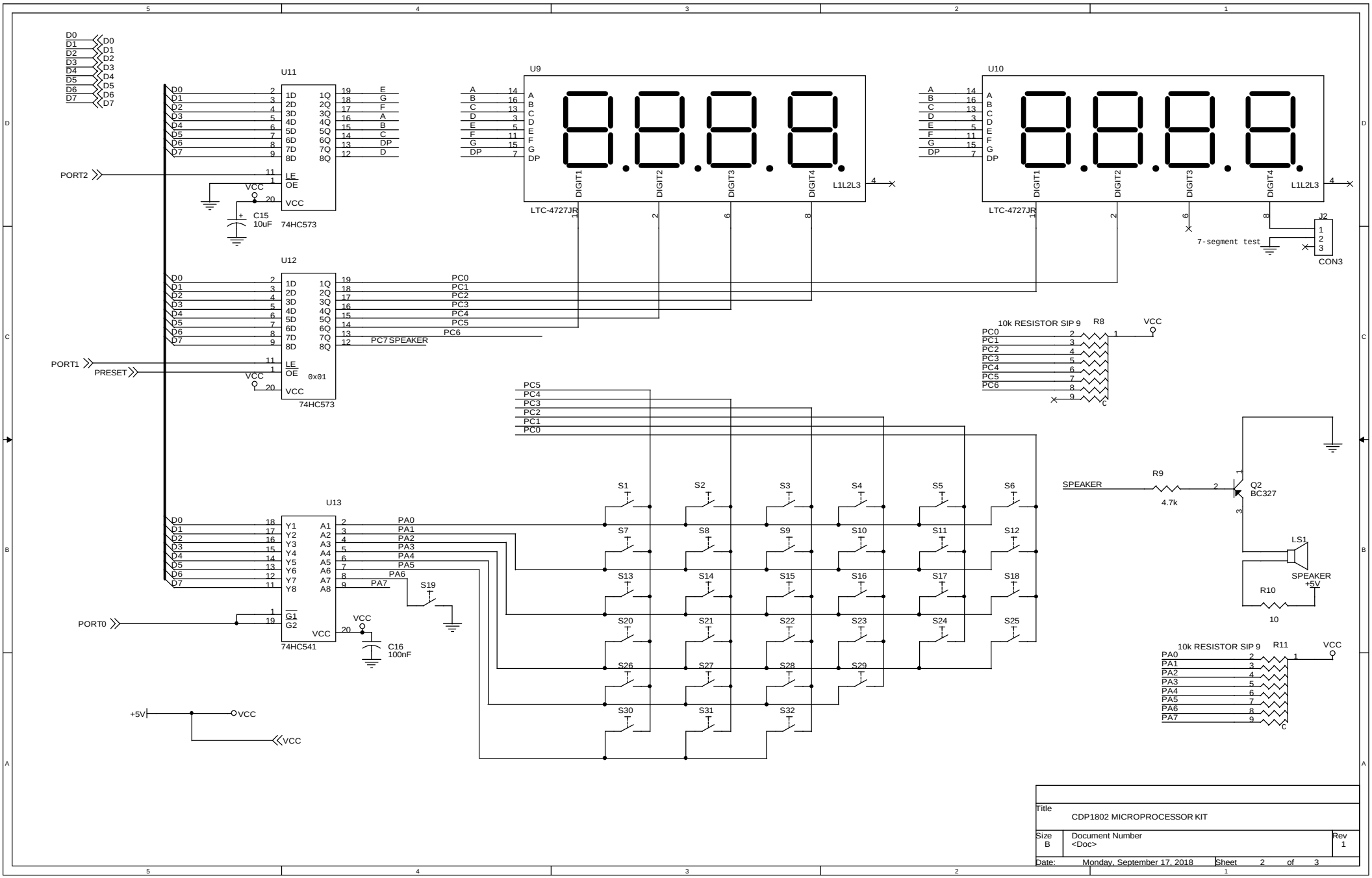
Any text LCD with HD44780 compatible controller can be used.

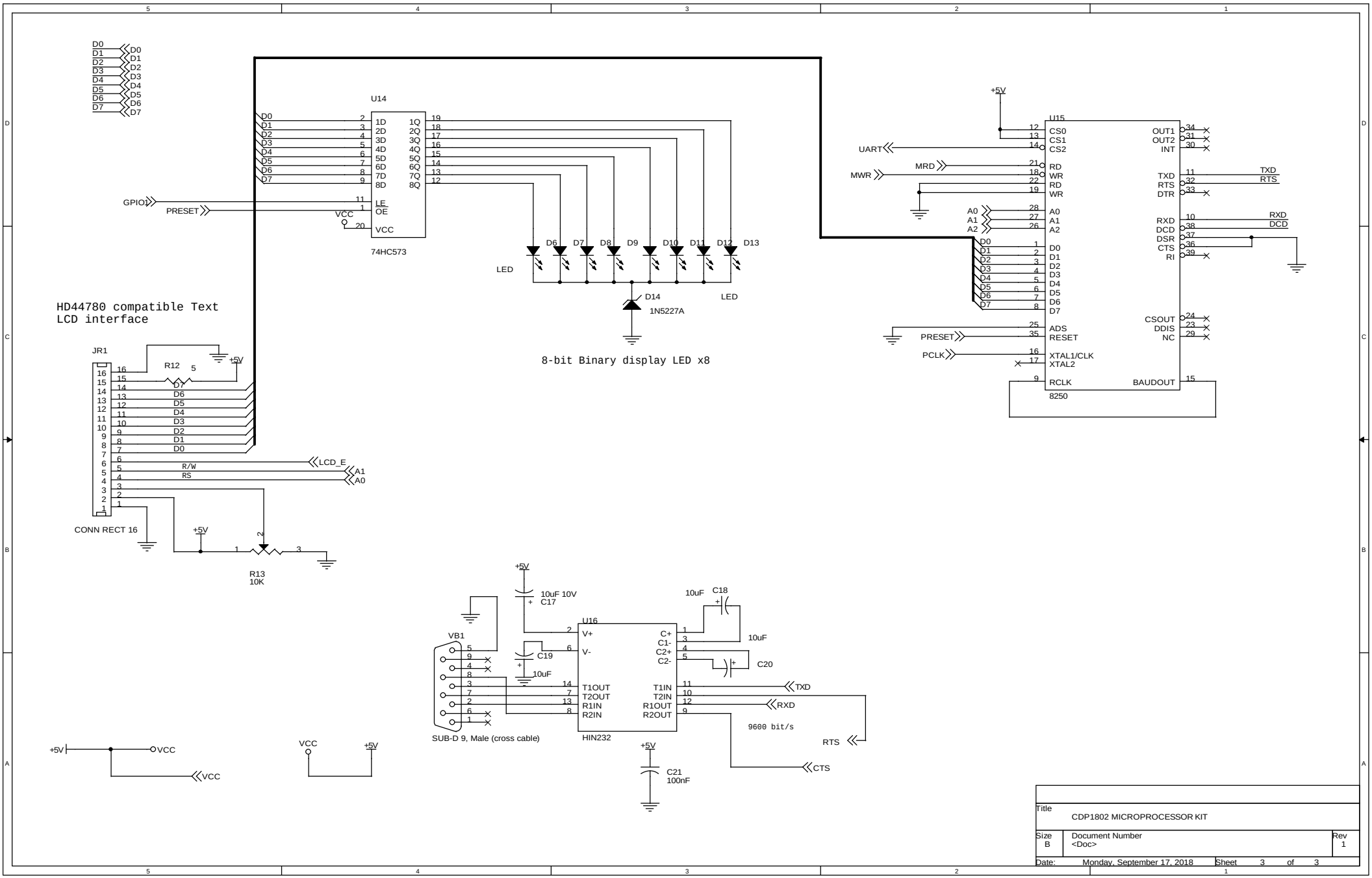
LOGIC PROBE POWER SUPPLY

The kit provides test points TP4(+5V) and TP5(GND) for using the logic probe. Students may learn digital logic signals with logic probe easily. Tick signal is indicated by D1 LED blinking. Red clip is for +5V and Black clip for GND.

HARDWARE SCHEMATIC, PARTS LIST


0x0000-0x6FFF Monitor Program
0x8000-0xFFFF 32kB SRAM
Programmable system tick
CPU expansion connector
PLD decoder logic
ESP switch
10ms Tick
CDP1802 MICROPROCESSOR KIT 2018
Designed by Wichit Sirichote, wichit.sirichote@gmail.com, (C) 2018
Date: Monday, September 17, 2018
Sheet 1 of 3


U11
74HC573
U12
74HC573
U13
74HC541
U9
LTC-4727JR
U10
LTC-4727JR
L1L2L3
DIGIT1
DIGIT2
DIGIT3
DIGIT4
PORT2
PORT1
PRESET
PORT0
VCC
C15
10uF
C16
100nF
J2
CON3
7-segment test
10k RESISTOR SIP 9
R8
R9
4.7k
R10
10
R11
Q2
BC327
LS1
SPEAKER
+5V
PC7 SPEAKER
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
S15
S16
S17
S18
S19
S20
S21
S22
S23
S24
S25
S26
S27
S28
S29
S30
S31
S32
Title
CDP1802 MICROPROCESSOR KIT
Size
B
Document Number
<Doc>
Rev
1
Date:
Monday, September 17, 2018
Sheet
2
of
3



Title	CDP1802 MICROPROCESSOR KIT	
Size B	Document Number <Doc>	Rev 1
Date: Monday, September 17, 2018	Sheet 3 of 3	

PARTS LIST

Semiconductors

U1 27C256, 32kB EPROM
U2 HM62256B, 32kB SRAM
U3 AT89C2051, preprogrammed 10ms tick generator
U4 22V10, preprogrammed PLD
U5 CDP1802A, Intersil CMOS microprocessor
U6 74HC14A
U7,U11,U12,U14 74HC573
U8 7805, +5V voltage regulator
U10,U9 LTC-4727JR, high brightness 7 segment display
U13 74HC541
U15 INS8250, UART
U16 HIN232, RS232 converter
Q1 KIA7042, voltage detector
Q2 BC557, PNP transistor

D1,D6,D7,D8,D9,D10,D11, 3mm LED (red)
D12,D13
D2 1N4733A
D3 Q LED (yellow)
D4 POWER LED
D5 1N4001
D14 1N5227A

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

R4,R1 680
R3,R2 RESISTOR SIP 9
R13,R5 10k
R6 10M
R7 1k
R11,R8 10k RESISTOR SIP 9
R9 4.7k
R10 20
R12 5

Capacitors

C1,C4,C5,C15,C18,C19,C20 10uF
C3,C2 22pF
C6 10uF 16V
C7 1000uF25V
C8,C9,C10,C11,C12 0.1uF
C13,C14 0.1uF
C21,C16 100nF
C17 10uF 10V

Additional parts

JP1 HEADER 20X2
JR1 CONN RECT 16
J1 DC Input
J2 CON3
LS1 SPEAKER
SW2 INTRPT
SW3 RESET
SW4 EF1
SW5 EF2
S1,S2,S3,S4,S5,S6,S7,S8, 12mm tact switch
S9,S10,S11,S12,S13,S14,
S15,S16,S17,S18,S19,S20,
S21,S22,S23,S24,S25,S26,
S27,S28,S29,S30,S31,S32
SW1 ESP switch, 10ms selector switch
VB1 SUB-D 9, Male (cross cable)
Y1 3.6864MHz Xtal
PCB double side plate through hole
display filter sheet,
Keyboard sticker printable SVG file

MONITOR PROGRAM LISTINGS

```
/*
    Monitor source code for CDP1802 Microprocessor Kit 2018
    (C) 2018 by Wichit Sirichote
 Xtal frequency 3.6864MHz, 32kB RAM, 28kB ROM, 4 kB memory mapped I/O
 ROM 0-6FFF,
 I/O 7000-7FFF
 RAM 8000

*/

// prototype declaration

void key_dump();
void pstr(char *s);
void read_record1();


volatile unsigned char *gpio1 = (unsigned char*) 0x7000;
volatile unsigned char *segment = (unsigned char*) 0x7102;
volatile unsigned char *digit = (unsigned char*) 0x7101;
volatile unsigned char *port0 = (unsigned char*) 0x7100;
volatile unsigned char *port1 = (unsigned char*) 0x7101;
volatile unsigned char *port2 = (unsigned char*) 0x7102;

volatile unsigned char *uart_buffer = (unsigned char*) 0x7300;
volatile unsigned char *uart_line_status = (unsigned char*) 0x7305;
volatile unsigned char *uart_fifo = (unsigned char*) 0x7302;
volatile unsigned char *uart_lcr = (unsigned char*) 0x7303;
volatile unsigned char *uart_divisor_lsb = (unsigned char*) 0x7300;
volatile unsigned char *uart_divisor_msb = (unsigned char*) 0x7301;
volatile unsigned char *uart_scr = (unsigned char*) 0x7307;

volatile unsigned char *LCD_cwr = (unsigned char*) 0x7200;
volatile unsigned char *LCD_dwr = (unsigned char*) 0x7201;
volatile unsigned char *LCD_crd = (unsigned char*) 0x7202;
volatile unsigned char *LCD_drd = (unsigned char*) 0x7203;

volatile unsigned char *tick = (unsigned char*) 0xffff;

#define BUSY 0x80



char convert[16]= {0xBD,0x30,0x9B,0xBA,0x36,0xAE,0xAF,0x38,0xBF,0xBE,0x3F,0xA7,0x8D,0xB3,0x8F

char text1[]={"\r\n\n\n\nCDP1802 MICROPROCESSOR KIT 2018"};
char text2[]={"CDP1802 KIT-----"};
char text3[]={"32k RAM 9600UART"};
char text4[]={"\r\nLoad Intel Hex file..waiting"};
char text5[]={"\r\nBye check sum error!"};
char text6[]={"\r\n0 error..."};









//---------------------------- utility subroutines ----------------------------------------
// user delay subroutine will be loaded into RAM at location F000
// to use it, load register with F001, the set it to be program counter
char user_delay[]={0xD0,0xF8,0x64,0xA6,0xF8,00,0xA7,0x27,0x87,0x3A,07,0x26,0x86,0x3A,04,0x30,

// interrupt service routine @ F020
char interrupt_test[]={0xF8,0xF0,0xB1,0xF8,0x38,0xA1,0xF8,0x70,0xB4,0xF8,00,0xA4,0xF8,
0xFF,0xB6,0xF8,0xFF,0xA6,0xE0,0x70,0x00,0x30,0x35,0x70,0x22,0x78,0x06,0xFC,01,0x56,0xFB,
 0x0A,0x3A,0x46,0x56,0x15,0x85,0x54,0xC4,0x30,0x37};


```

```
 72
 73
 74
 75    char n,c,t,hit;
 76
 77
 78    int i,j,k;
 79    int temp;
 80    unsigned int warm;
 81
 82    char flag;
 83
 84
 85    char bcc, save_bcc, bcc_error, record_type;
 86
 87
 88    char o,q,key;
 89    char key_pressed;
 90    char buffer[6];
 91
 92    int temp,temp16,timeout;
 93    int num, start, end, desti;
 94
 95
 96    unsigned int PC, save_PC;
 97
 98    char state,x,hit;
 99
100    char *dptr;
101    char *dptr2;
102
103    char user_d, user_df, user_xp;
104    int user_r3,user_r4,user_r5,user_r6,user_r7,user_r8,user_r9,user_r10;
105    int user_r11,user_r12,user_r13,user_r14,user_r15;
106
107
108
109
110    void delay_num1()
111    {
112      temp=0;
113      temp=0;
114    }
115
116    void delay_ms(unsigned int w)
117    {
118     unsigned int n;
119       for( n = 0; n < w; n++)
120        ;
121
122    }
123
124    void blink_q()
125    {
126     asm(" seq \n");
127      delay_ms(100);
128     asm(" req \n");
129      delay_ms(2000);
130
131     asm(" seq \n");
132      delay_ms(100);
133     asm(" req \n");
134      delay_ms(2000);
135    }
136
137
138    char scan(){
139
140        k = 1;
141
142     key = 0xff;
```

```
143    q = 0;
144
145       for(i=0; i<6; i++)
146       {
147         *digit = ~k;
148
149     *segment = buffer[i];
150
151     if(buffer[i] != 0x30 && buffer[i] != 0x38 && buffer[i] != 0x70) delay_ms(3);
152     else delay_num1();
153
154     *segment = 0;
155
156    //  delay_ms(1);
157
158      o= *port0;
159
160     for(n=0; n<6; n++)
161     {
162       if((o&1)==0)
163      {key=q;
164
165
166      }
167
168        else q++;
169        o = o >> 1;
170     }
171
172      k = k << 1;
173     }
174
175         key_pressed=key;
176
177     //  *gpio1=key;
178
179
180     return key_pressed;
181
182   }
183
184
185   void dot_address()
186   {
187    buffer[0]=buffer[0]&~0x40;
188    buffer[1]=buffer[1]&~0x40;
189
190
191     buffer[2]=buffer[2]|0x40;
192      buffer[3]=buffer[3]|0x40;
193       buffer[4]=buffer[4]|0x40;
194       buffer[5]=buffer[5]|0x40;
195
196   }
197
198
199   void dot_data()
200   {
201
202     buffer[0]=buffer[0]|0x40;
203    buffer[1]=buffer[1]|0x40;
204
205    buffer[2]=buffer[2]&~0x40;
206      buffer[3]=buffer[3]&~0x40;
207       buffer[4]=buffer[4]&~0x40;
208       buffer[5]=buffer[5]&~0x40;
209
210   }
211
212   void hex4(int h)
213   {
```

```
    temp16 = h;
 buffer[2]= convert[temp16&0xf];
 temp16>>=4;
 buffer[3]= convert[temp16&0xf];
 temp16>>=4;
 buffer[4]=convert[temp16&0xf];
 temp16>>=4;
 buffer[5]=convert[temp16&0xf];
}


void address_display()
{

 temp16 = PC;

 hex4(temp16);

}

void data_display()
{
    dptr = (char *)PC;

 n = *dptr;

 buffer[0]= convert[n&0xf];
 n = n>>4;
 buffer[1]=convert[n&0xf];
}

void read_memory()
{
    address_display();
 data_display();
}


void key_address()
{


 state = 1;

   read_memory();


    dot_address();
 hit=0;




 }

void key_data()
{

 read_memory();
    dot_data();
    hit=0;
 state=2;

}

void key_plus()
{

     if(state==1 || state==2)
     {
       PC++;
```

```
         read_memory();
      key_data();
     }

    if(state==5)
{
     state=6;
 start = num;
 hit=0;
 buffer[0]=0x8f;  /* end cursor */
 return;

}

if(state==6)
{
      state=7;
 end = num;
 hit=0;
 buffer[0]=0xb3;  /* destination cursor */

// if(end <= start) print_error();




}



}

void key_minus()
{
   if(state==1 | state ==2)
   {
    PC--;
    read_memory();
 key_data();
   }
}

void data_hex()
{

   dptr = (char *)PC;
 x = *dptr;
 if(hit==0) x=0;
 {
  hit =1;
  x = x << 4;
  x = x|key;

  *dptr = x;

  read_memory();

      dot_data();
 }
}

void key_PC()
{
     PC=save_PC;
  key_data();
}

void hex_address()
{
    if(hit==0) PC=0;
 {
```

```
356       hit=1;
357
358       PC<<=4;
359       PC |= key;
360       read_memory();
361       dot_address();
362    }
363   }
364
365   /* insert byte and shift 512 bytes down */
366
367   void insert()
368   {
369    dptr=(char *)PC;
370       for(j=512; j>0; j--)
371    {
372     *(dptr+j)=*(dptr+j-1);
373    }
374    *(dptr+1)=0;  /* insert next byte */
375    PC++;
376       read_memory();
377    state=2;
378   }
379
380
381   /* delete current byte and shift 512 bytes up */
382
383   void cut_byte()
384   {
385     dptr=(char *)PC;
386       for(j=0; j<512; j++)
387    {
388     *(dptr+j)=*(dptr+j+1);
389    }
390       read_memory();
391    state=2;
392   }
393
394   void key_go(){
395
396    if(state==1 || state==2)
397    {
398
399     asm(" ldAD r5,_PC \n"
400             "    ldn r5 \n"
401                 "    phi  r0 \n"
402                 "    inc r5  \n"
403                 "    ldn r5  \n"
404      "    plo r0  \n"
405      "    sep r0\n"
406             );
407
408    }
409
410    if(state==7)
411    {
412           desti = num;
413    temp = end-start;
414    dptr = (char *)start;
415    dptr2 = (char *) desti;
416
417    for(i=0; i<temp; i++)
418    {
419     *(dptr2+i)=*(dptr+i);
420    }
421    PC = desti;
422           read_memory();
423           dot_data();
424    state=2;
425
426
```

```
427
428
429
430     }
431
432    }
433
434    void key_test()
435    {
436
437     PC = 0xf020;  // test 10ms timer SW1 selects 10ms
438
439     state = 1;
440
441     key_go();
442
443
444    }
445
446    void key_load()
447    {
448
449      pstr(text4);
450      read_record1();
451
452      if(bcc_error) pstr(text5);
453        else pstr(text6);
454
455     PC = 0x8000;
456     key_data();
457
458
459
460    }
461
462    void key_reg()
463    {
464       buffer[5]= 0x03;
465       buffer[4]= 0x8F;
466       buffer[3]= 0xad;
467       buffer[2]=0;
468       buffer[1]=0;
469       buffer[0]=0;
470
471       state = 3;    /* register display state = 3 with hex key */
472
473    }
474
475
476
477    void reg_d()
478    {
479
480        n = user_d;
481
482     buffer[2]= convert[n&0xf];
483     n = n>>4;
484     buffer[3]=convert[n&0xf];
485     buffer[4]=0;
486     buffer[5]=0;
487     buffer[1]=0;
488     buffer[0]=0xb3;   // reg d
489    }
490
491    void reg_df()
492    {
493
494        n = user_df;  // DF =  0 or 1
495
496     buffer[2]= 0;
497     buffer[3]=convert[n&0xf];
```

```
 buffer[4]=0;
 buffer[5]=0;
 buffer[1]=0xb3;   // flag DF
 buffer[0]=0x0f;   //
}

void reg_xp()
{

    n = user_xp;  // xp

 buffer[2]= convert[n&0xf];
 n = n>>4;

 buffer[3]=convert[n&0xf];
 buffer[4]=0;
 buffer[5]=0;
 buffer[1]=0x13;   // XP
 buffer[0]=0x1F;   //
}

void reg_r3()
{

    hex4(user_r3);

 buffer[1]=0x03;
 buffer[0]=0xba;
}

void reg_r4()
{

    hex4(user_r4);  //

 buffer[1]=0x03;   //
 buffer[0]=0x36;
}

void reg_r5()
{

    hex4(user_r5);  //

 buffer[1]=0x03;   //
 buffer[0]=0xae;
}

void reg_r6()
{

    hex4(user_r6);  //

 buffer[1]=0x03;   //
 buffer[0]=0xaf;
}

void reg_r7()
{

    hex4(user_r7);  //

 buffer[1]=0x03;   //
 buffer[0]=0x38;
}

void reg_r8()
{

    hex4(user_r8);  //

```

```
569     buffer[1]=0x03;   //
570     buffer[0]=0xbf;
571    }
572    void reg_r9()
573    {
574
575        hex4(user_r9);  //
576
577     buffer[1]=0x03;   //
578     buffer[0]=0xbe;
579    }
580    void reg_r10()
581    {
582
583        hex4(user_r10);  //
584
585     buffer[1]=0x03;   //
586     buffer[0]=0x3f;
587    }
588    void reg_r11()
589    {
590
591        hex4(user_r11);  //
592
593     buffer[1]=0x03;   //
594     buffer[0]=0xa7;
595    }
596    void reg_r12()
597    {
598
599        hex4(user_r12);  //
600
601     buffer[1]=0x03;   //
602     buffer[0]=0x8d;
603    }
604    void reg_r13()
605    {
606
607        hex4(user_r13);  //
608
609     buffer[1]=0x03;   //
610     buffer[0]=0xb3;
611    }
612
613    void reg_r14()
614    {
615
616        hex4(user_r14);  //
617
618     buffer[1]=0x03;   //
619     buffer[0]=0x8f;
620    }
621
622    void reg_r15()
623    {
624
625        hex4(user_r15);  //
626
627     buffer[1]=0x03;   //
628     buffer[0]=0x0f;
629    }
630
631
632
633
634    void reg_display()
635    {
636
637      switch(key)
638     {
639        case 0: reg_d(); break;
```

```
640       //case 1: reg_df(); break;  // future work
641       //case 2: reg_xp(); break;
642       case 3: reg_r3(); break;
643       case 4: reg_r4(); break;
644       case 5: reg_r5(); break;
645       case 6: reg_r6(); break;
646       case 7: reg_r7(); break;
647       case 8: reg_r8(); break;
648       case 9: reg_r9(); break;
649       case 10: reg_r10(); break;
650       case 11: reg_r11(); break;
651       case 12: reg_r12(); break;
652       case 13: reg_r13(); break;
653       case 14: reg_r14(); break;
654       case 15: reg_r15(); break;
655
656        }
657    }
658
659
660    void enter_num()
661    {
662       if(hit==0) num=0;
663     {
664        hit=1;
665
666        num<<=4;
667        num |= key;
668        hex4(num);
669
670     }
671    }
672
673    void clear_buffer()
674    {
675     for(i=0; i<6; i++)
676      *(buffer+i)=0;
677    }
678
679
680    void key_copy()
681    {
682
683
684     state=5;
685      hit=0;
686      clear_buffer();
687             buffer[2]= 0xbd;
688
689      buffer[0]=0xae;
690      buffer[1]=0;
691
692
693    }
694
695
696
697
698
699    /* return internal code hex keys and function keys  */
700
701    char key_code(char n)
702    {
703     char d;
704     if(n == 0x10) return 0;
705     if(n == 0x21) return 1;
706     if(n == 0x1b) return 2;
707     if(n == 0x15) return 3;
708     if(n == 0x16) return 4;
709     if(n == 0x20) return 5;
710     if(n == 0x1a) return 6;
```

```
711     if(n == 0x14) return 7;
712     if(n == 0x1c) return 8;
713     if(n == 0x1f) return 9;
714     if(n == 0x19) return 0xa;
715     if(n == 0x13) return 0xb;
716     if(n == 0x22) return 0xc;
717     if(n == 0x1e) return 0xd;
718     if(n == 0x18) return 0xe;
719     if(n == 0x12) return 0xf;
720
721     if(n == 0xc) return 0x10;  // pc
722     if(n == 0xd) return 0x11;  // reg
723     if(n == 0xe) return 0x12;  // data
724     if(n == 0xf) return 0x13;  // address
725
726     if(n == 6) return 0x14;
727     if(n == 7) return 0x15;  // MUTE
728     if(n == 8) return 0x16;  // -
729     if(n == 9) return 0x17;  // +
730
731     if(n == 0) return 0x18;  // load
732     if(n == 1) return 0x19;  // insert
733     if(n == 2) return 0x1a;  // delete
734     if(n == 3) return 0x1b;  // go
735
736        if(n == 0x13) return 0x1c;
737     if(n == 0x1d) return 0x1d; // test
738     if(n == 0x17) return 0x1e; // dump
739     if(n == 0x00) return 0x1f;
740     if(n == 0x23) return 0x20; // copy
741
742     return 0xff;
743
744    }
745
746
747    void delay_beep()
748    {
749       for(j=0; j<2; j++)
750       continue;
751    }
752
753
754    void beep()
755    {
756
757       *port2=0;
758
759       for(x=0; x<60; x++)
760       {
761       *port1 = (char) ~0x80;
762        delay_beep();
763       *port1 = 0xff;
764        delay_beep();
765       }
766
767    }
768
769    void key_mute()
770    {
771        flag ^=1;
772    }
773
774    void key_qled()
775    {
776
777     while(1)
778     {
779     asm(" seq \n");
780     delay_ms(500);
781     asm(" req \n");
```

```
782
783
784
785         delay_ms(30000);
786       }
787      }
788
789
790      void key_exe()
791      {
792
793        if(flag==0) beep();
794
795       if( key>15)
796       {
797        if(key==0x13) key_address();
798        if(key==0x12) key_data();
799        if(key==0x17) key_plus();
800        if (key==0x16) key_minus();
801        if (key==0x10) key_PC();
802        if (key==0x1b) key_go();
803        if (key==0x11) key_reg();
804        if (key==0x19) insert();
805        if (key==0x1a) cut_byte();
806        if (key==0x1e) key_dump();
807        if (key==0x1d) key_test();
808        if (key==0x18) key_load();
809        if (key==0x20) key_copy();
810        if (key==0x15) key_mute();
811        if (key==0x14) key_qled();
812
813
814         }
815
816       else
817       {
818        switch(state)
819        {
820
821         case 1: hex_address(); break;
822         case 2: data_hex(); break;
823         case 3: reg_display(); break;
824         case 5: enter_num(); break;
825         case 6: enter_num(); break;
826         case 7: enter_num(); break;
827               }
828
829       }
830
831      }
832
833      void scan1(){
834
835       while(scan() != 0xff)
836        ;
837       delay_ms(10);
838
839       while(scan() == 0xff)
840        ;
841       delay_ms(5);
842
843       key = scan();
844
845       key = key_code(key);
846
847      // *gpio1=key;   // debug
848
849       if((key>=0) && (key <0x30))  key_exe();
850      }
851
852      //------------------------------ UART --------------------------------
```

```
853
854    void init_8250()
855    {
856        *uart_lcr = 0x83;
857     *uart_divisor_lsb = 24;  // computed for 3.6864MHz
858     *uart_divisor_msb = 0;
859      *uart_fifo =7;
860         *uart_lcr = 3;
861    }
862
863    void cout(char n)
864    {
865     while((*uart_line_status & 0x20)==0)
866      ;
867     *uart_buffer=n;
868    }
869
870    char cin()
871    {
872     while((*uart_line_status & 1)==0)
873      ;
874     c =*uart_buffer;
875
876       return c;
877    }
878
879    void pstr(char *s)
880    {
881      while( *s )
882       {
883                   cout(*s);
884                   s++;
885             }
886    }
887
888    void test_uart()
889    {
890
891       for(c=0x20; c<0x80; c++)
892        cout(c);
893    }
894
895    void newline()
896    {
897     cout(0x0a);
898     cout(0x0d);
899    }
900
901    void send_hex(char n)
902    {
903       k = n>>4;
904       k = k&0xf;
905
906       if (k>9) cout(k+0x37); else cout(k+0x30);
907       k= n&0xf;
908       if (k>9) cout(k+0x37); else cout(k+0x30);
909    }
910
911    void send_word_hex(int n)
912    {
913       temp16 = n>>8;
914       k = temp16&0xff;
915       send_hex(k);
916       k = n&0xff;
917       send_hex(k);
918    }
919
920
921
922    void key_dump()
923    {
```

```
924       int j,p;
925
926       dptr = (char *)PC;
927
928       for(j=0; j<16; j++)
929       {
930         newline();
931         send_word_hex(PC);
932         cout(':');
933            for(p=0; p<16; p++)
934            {
935
936             send_hex(*(dptr+p));
937             cout(0x20);
938             }
939
940            cout(0x20);
941
942            for (p=0; p<16; p++)
943            {
944              q=*(dptr+p);
945             if(q >= 0x20 && q < 0x80) cout(q);
946              else cout('.');
947
948            }
949
950
951             dptr+=16;
952      PC+=16;
953       }
954       newline();
955         // PC = dptr;
956       key_address();
957    }
958
959
960    char nibble2hex(char c)
961    {
962     char n;
963      if(c<0x40) return (c-0x30);
964      else return (c-0x37);
965    }
966
967    char gethex()
968    {
969      char a,b;
970
971      a = cin();
972      b = cin();
973
974      a = nibble2hex(a)<<4;
975      b = nibble2hex(b);
976      a = a|b;
977      bcc = bcc+a;  /* compute check sum */
978
979      return (a);
980    }
981
982    int get16bitaddress()
983    {
984      unsigned int load_address;
985
986      load_address =0;
987
988      load_address |= gethex();
989      load_address <<=8;
990      load_address |= gethex();
991
992      return load_address;
993    }
994
```

```
995
996   // read Intel hex record
997   // :20 0000 00 71F8FFB2F8F0A2F800B3F814A37BD37AD3300DD0F864A6F800A727873A1A2686EB
998
999   void read_record1()
1000  {
1001
1002     char x;
1003     char byte_count;
1004
1005     int address16bit;
1006
1007     bcc_error=0;
1008
1009    do{
1010
1011
1012     bcc =0;
1013
1014     while(cin()!= ':')
1015      ;
1016
1017
1018     byte_count = gethex();  /* only data record */
1019
1020     dptr = (char *) get16bitaddress();
1021
1022   record_type= gethex();
1023
1024   // now read byte
1025
1026    for(x=0; x<byte_count; x++)
1027     {
1028      *(dptr+x) = gethex();
1029     }
1030
1031    // bcc = gethex();
1032   //  bcc = ~bcc;   /* one's complement */
1033  //   bcc+= bcc;  // two's complement
1034
1035     *gpio1=bcc;    /* loading indicator */
1036
1037
1038  // save_bcc= bcc;
1039
1040   // if(save_bcc != gethex()) bcc_error=1;
1041
1042    }
1043
1044    while(record_type==0);
1045
1046
1047  }
1048
1049
1050  //---------------------------------------------------------------------
1051
1052
1053
1054  //---------------------------- LCD drivers ----------------------------
1055
1056
1057  /*   LCD driver   */
1058
1059  void LcdReady()
1060  {
1061     timeout=0;
1062
1063     while(((*LCD_crd&0x80)==1) && (timeout<500))
1064      ++timeout;
1065  }
```

```
1066
1067    void clr_screen()
1068    {
1069        LcdReady();
1070      *LCD_cwr=0x01;
1071    }
1072
1073
1074    void goto_xy(int x, int y)
1075    {
1076        LcdReady();
1077
1078        switch(y)
1079     {
1080     case 0: *LCD_cwr=0x80+x; break;
1081        case 1: *LCD_cwr=0xC0+x; break;
1082        case 2: *LCD_cwr=0x94+x; break;
1083        case 3: *LCD_cwr=0xd4+x; break;
1084        }
1085    }
1086
1087
1088    void InitLcd()
1089    {
1090        LcdReady();
1091        *LCD_cwr=0x38;
1092        LcdReady();
1093        *LCD_cwr=0x0c;
1094        clr_screen();
1095        goto_xy(0,0);
1096     delay_ms(100);
1097    }
1098
1099
1100    void PutLCD(char *str)
1101    {
1102        char i;
1103        for (i=0; str[i] != '\0'; i++)
1104     {
1105        LcdReady();
1106        *LCD_dwr=str[i];
1107        }
1108
1109
1110    }
1111
1112    void putch_lcd(char ch)
1113    {
1114        LcdReady();
1115        *LCD_dwr=ch;
1116    }
1117    //------------------------- end of LCD drivers ---------------------------
1118
1119    void load_user_subroutines()
1120    {
1121     PC = 0xf000;
1122     dptr = (char *) PC;
1123
1124     for (i=0; i<17 ;i++ )
1125     {
1126          *(dptr+i) = user_delay[i];
1127          }
1128
1129        PC = 0xf020;
1130
1131     dptr = (char *) PC;
1132
1133     for (i=0; i<sizeof(interrupt_test) ;i++ )
1134     {
1135          *(dptr+i) = interrupt_test[i];
1136          }
```

```
1137
1138
1139
1140    }
1141
1142
1143
1144
1145    void service_break()
1146    {
1147
1148       asm(
1149        " plo r2     ; save D first \n"
1150           " ldAD r1, _USER_D \n"
1151           " glo r2 \n"
1152           " str r1 \n"
1153
1154           " ldAD r1, _USER_R3 \n"
1155           " ghi r3 \n"
1156           " str r1 \n"
1157           " inc r1 \n"
1158           " glo r3 \n"
1159           " str r1 \n"
1160
1161           " ldAD r1, _USER_R4\n"
1162           " ghi r4 \n"
1163           " str r1 \n"
1164           " inc r1 \n"
1165           " glo r4 \n"
1166           " str r1 \n"
1167
1168           " ldAD r1, _USER_R5\n"
1169           " ghi r5 \n"
1170           " str r1 \n"
1171           " inc r1 \n"
1172           " glo r5 \n"
1173           " str r1 \n"
1174
1175           " ldAD r1, _USER_R6\n"
1176           " ghi r6 \n"
1177           " str r1 \n"
1178           " inc r1 \n"
1179           " glo r6 \n"
1180           " str r1 \n"
1181
1182           " ldAD r1, _USER_R7\n"
1183           " ghi r7 \n"
1184           " str r1 \n"
1185           " inc r1 \n"
1186           " glo r7 \n"
1187           " str r1 \n"
1188
1189       " ldAD r1, _USER_R8\n"
1190           " ghi r8 \n"
1191           " str r1 \n"
1192           " inc r1 \n"
1193           " glo r8 \n"
1194           " str r1 \n"
1195
1196       " ldAD r1, _USER_R9\n"
1197           " ghi r9 \n"
1198           " str r1 \n"
1199           " inc r1 \n"
1200           " glo r9 \n"
1201           " str r1 \n"
1202
1203       " ldAD r1, _USER_R10\n"
1204           " ghi r10 \n"
1205           " str r1 \n"
1206           " inc r1 \n"
1207           " glo r10 \n"
```

```
1208            " str r1 \n"
1209
1210        " ldAD r1, _USER_R11\n"
1211            " ghi r11 \n"
1212            " str r1 \n"
1213            " inc r1 \n"
1214            " glo r11 \n"
1215            " str r1 \n"
1216
1217        " ldAD r1, _USER_R12\n"
1218            " ghi r12 \n"
1219            " str r1 \n"
1220            " inc r1 \n"
1221            " glo r12 \n"
1222            " str r1 \n"
1223
1224        " ldAD r1, _USER_R13\n"
1225            " ghi r13 \n"
1226            " str r1 \n"
1227            " inc r1 \n"
1228            " glo r13 \n"
1229            " str r1 \n"
1230
1231        " ldAD r1, _USER_R14\n"
1232            " ghi r14 \n"
1233            " str r1 \n"
1234            " inc r1 \n"
1235            " glo r14 \n"
1236            " str r1 \n"
1237
1238        " ldAD r1, _USER_R15\n"
1239            " ghi r15 \n"
1240            " str r1 \n"
1241            " inc r1 \n"
1242            " glo r15 \n"
1243            " str r1 \n"
1244
1245            " lbr 0    ; return to system monitor\n"
1246
1247        );
1248
1249    }
1250
1251    void main()
1252    {
1253
1254     *segment = 0;
1255     *digit = 0xff;
1256     *gpio1 = 0;
1257
1258     *tick=0;
1259
1260     state =0;
1261
1262        init_8250();
1263        InitLcd();
1264
1265     if(warm != 0xaaaa)
1266     {
1267
1268         flag =0;
1269      warm = 0xaaaa;
1270     }
1271
1272
1273
1274     blink_q();
1275
1276      PC = 0x8000;
1277         save_PC = 0x8000;
1278
```

```
1279    //  asm("ldAD r1,9000h \n");    // load interrupt vector at 9000h
1280
1281
1282         load_user_subroutines();
1283
1284     dptr= (char *) PC;
1285
1286        // test_uart();
1287
1288    PutLCD(text2);
1289    goto_xy(0,1);
1290       PutLCD(text3);
1291
1292     pstr(text1);
1293
1294
1295
1296    buffer[0]=0;
1297    buffer[1]=0;
1298    buffer[2]=convert[2];
1299    buffer[3]=convert[0];
1300    buffer[4]=convert[8];
1301    buffer[5]=convert[1];
1302
1303    while(1){
1304
1305             scan1();
1306    }
1307    }
1308
1309    void setRAM()
1310    {
1311       asm("ROM_END1:  EQU  $\n");
1312       asm("    org 0fe00h\n");  // space for monitor variables
1313    }
1314
1315
1316
1317
1318
```

NOTE