<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVERVIEW</td>
<td>4</td>
</tr>
<tr>
<td>FUNCTIONAL BLOCK DIAGRAM</td>
<td>4</td>
</tr>
<tr>
<td>HARDWARE LAYOUT</td>
<td>5</td>
</tr>
<tr>
<td>KEYBOARD LAYOUT</td>
<td>7</td>
</tr>
<tr>
<td>HARDWARE FEATURES</td>
<td>9</td>
</tr>
<tr>
<td>MONITOR PROGRAM FEATURES</td>
<td>9</td>
</tr>
<tr>
<td>MEMORY AND I/O MAPS</td>
<td>10</td>
</tr>
<tr>
<td>PLD DECODER</td>
<td>11</td>
</tr>
<tr>
<td>GETTING STARTED</td>
<td>12</td>
</tr>
<tr>
<td>HOW TO ENTER PROGRAM USING HEX CODE</td>
<td>14</td>
</tr>
<tr>
<td>AUTO LOAD Subroutine DELAY</td>
<td>16</td>
</tr>
<tr>
<td>GPIO1 LED</td>
<td>17</td>
</tr>
<tr>
<td>RS232C PORT</td>
<td>18</td>
</tr>
<tr>
<td>DATA FRAME for UART COMMUNICATION</td>
<td>18</td>
</tr>
<tr>
<td>CONNECTING KIT TO TERMINAL</td>
<td>19</td>
</tr>
<tr>
<td>EXPANSION BUS HEADER</td>
<td>22</td>
</tr>
<tr>
<td>10ms TICK GENERATOR</td>
<td>23</td>
</tr>
<tr>
<td>CONNECTING LCD MODULE</td>
<td>24</td>
</tr>
<tr>
<td>LOGIC PROBE POWER SUPPLY</td>
<td>25</td>
</tr>
<tr>
<td>HARDWARE SCHEMATIC, BOM</td>
<td></td>
</tr>
<tr>
<td>MONITOR PROGRAM LISTINGS</td>
<td></td>
</tr>
</tbody>
</table>
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.
RS232C connector, DB9 male

DC jack, +9VDC

Selector for 10ms tick or INTR key
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

<table>
<thead>
<tr>
<th>COPY</th>
<th>R12</th>
<th>R13</th>
<th>R14</th>
<th>R15</th>
<th>PC</th>
<th>Q LED</th>
<th>EF2</th>
<th>RESET</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST</td>
<td>R8</td>
<td>R9</td>
<td>R10</td>
<td>R11</td>
<td>REG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUMP</td>
<td>R4</td>
<td>R5</td>
<td>R6</td>
<td>R7</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>D</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>ADDR</td>
<td></td>
<td>USER</td>
</tr>
</tbody>
</table>

#### 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
<table>
<thead>
<tr>
<th>INS</th>
<th>Insert byte, after current byte +512 locations will be shifted down</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEL</td>
<td>Delete current byte, 512 locations will be shifted up</td>
</tr>
<tr>
<td>USER</td>
<td>User key for monitor program customization</td>
</tr>
<tr>
<td>TEST</td>
<td>Test 10ms tick, SW1 when set to 10ms tick, the gpio1 LED will be counting at 10Hz rate</td>
</tr>
<tr>
<td>COPY</td>
<td>Copy block of memory, use with key + and GO, sequence will be START, END, DESTINATION, then GO</td>
</tr>
<tr>
<td>DUMP</td>
<td>Dump memory contents to 9600 terminal, will need RS232 cross cable and PC running terminal emulator</td>
</tr>
<tr>
<td>LOAD</td>
<td>Load Intel HEX file, 1ms character and line delay will be needed.</td>
</tr>
</tbody>
</table>
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.

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

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.

This 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.

```
8000 00.
```

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.

```
8000 F8.
```

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

```
8001 70.
```

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

The subroutine provides delay for testing USER code.

Let us see another example:

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.
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.
CONNECTION 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

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
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[] = "CDP1802 MICROPROCESSOR KIT 2018";
char text2[] = "CDP1802 KIT-----";
char text3[] = "32k RAM 9600UART";
char text4[] = "Load Intel Hex file..waiting"
char text5[] = "Bye check sum error!"
char text6[] = "0 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, 0x00, 0xA7, 0x27, 0x87, 0x3A, 0x7, 0x26, 0x86, 0x3A, 0x4, 0x30,
char interrupt_test[] = { 0xF8, 0xF0, 0xB1, 0x8F, 0x38, 0xA1, 0xF8, 0x70, 0xB4, 0xF8, 0x00, 0xA4, 0xF8, 0x0F8, 0x0B6, 0x8F, 0xFF, 0xA6, 0xE0, 0x70, 0x00, 0x30, 0x35, 0x70, 0x22, 0x78, 0x06, 0xFC, 0x1, 0x56, 0xFB, 0x0A, 0x3A, 0x46, 0x56, 0x15, 0x85, 0x54, 0xC4, 0x30, 0x37 };

// interrupt service routine @ F020

char read_record1();
void key_dump();
void pstr(char *s);
char n, c, t, hit;

int i, j, k;
int temp;
unsigned int warm;

char flag;

char bcc, save_bcc, bcc_error, record_type;

char o, q, key;
char key_pressed;
char buffer[6];

int temp, temp16, timeout;
int num, start, end, desti;

unsigned int PC, save_PC;

char state, x, hit;

char *dptr;
char *dptr2;

char user_d, user_df, user_xp;

int user_r3, user_r4, user_r5, user_r6, user_r7, user_r8, user_r9, user_r10;
int user_r11, user_r12, user_r13, user_r14, user_r15;

void delay_num1()
{
    temp = 0;
    temp = 0;
}

void delay_ms(unsigned int w)
{
    unsigned int n;
    for (n = 0; n < w; n++)
        ;
}

void blink_q()
{
    asm(" seq 
" );
    delay_ms(100);
    asm(" req 
" );
    delay_ms(2000);
    asm(" seq 
" );
    delay_ms(100);
    asm(" req 
" );
    delay_ms(2000);
}

char scan()
{
    k = 1;
    key = 0xff;
q = 0;
for (i=0; i<6; i++)
{
    *digit = ~k;
    *segment = buffer[i];
    if (buffer[i] != 0x30 && buffer[i] != 0x38 && buffer[i] != 0x70) delay_ms(3);
    else delay_num1();
    *segment = 0;
    // delay_ms(1);
    o = *port0;
    for (n=0; n<6; n++)
    {
        if ((o&1)==0)
            {key=q;
             }
        else q++;
        o = o >> 1;
    }
    k = k << 1;
    }
    key_pressed=key;
    // *gpio1=key;
    return key_pressed;
}
void dot_address()
{
    buffer[0]=buffer[0]&~0x40;
    buffer[1]=buffer[1]&~0x40;
}
void dot_data()
{
    buffer[0]=buffer[0]|0x40;
    buffer[1]=buffer[1]|0x40;
    buffer[2]=buffer[2]&~0x40;
    buffer[3]=buffer[3]&~0x40;
    buffer[4]=buffer[4]&~0x40;
    buffer[5]=buffer[5]&~0x40;
}
void hex4(int h)
{
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()
{
  dpdr = (char *)PC;
  n = *dpdr;
  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++;
  }
```c
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;
    
```
hit=1;

PC <<= 4;
PC |= key;
read_memory();
dot_address();
}

/* insert byte and shift 512 bytes down */

void insert()
{
    dptr = (char *)PC;
    for (j = 512; j > 0; j--)
    {
        *(dptr + j) = *(dptr + j - 1);
    }
    *(dptr + 1) = 0; /* insert next byte */
    PC++;
    read_memory();
    state = 2;
}

/* delete current byte and shift 512 bytes up */

void cut_byte()
{
    dptr = (char *)PC;
    for (j = 0; j < 512; j++)
    {
        *(dptr + j) = *(dptr + j + 1);
    }
    read_memory();
    state = 2;
}

void key_go()
{
    if (state == 1 || state == 2)
    {
        asm(" ldAD r5, _PC \n"
             "   ldn r5 \n"
             "   phi r0 \n"
             "   inc r5 \n"
             "   ldn r5 \n"
             "   plo r0 \n"
             "   sep r0\n"
             ");
    }
    if (state == 7)
    {
        desti = num;
        temp = end - start;
        dptr = (char *)start;
        dptr2 = (char *) desti;
        for (i = 0; i < temp; i++)
        {
            *(dptr2 + i) = *(dptr + i);
        }
        PC = desti;
        read_memory();
        dot_data();
        state = 2;
    }
427
428
429
430  }
431
432  }
433
434  void key_test()
435  {
436  
437  PC = 0xf020; // test 10ms timer SW1 selects 10ms
438  state = 1;
439  key_go();
440  
441  }
442
443  
444  void key_load()
445  {
446  
447  pstr(text4);
448  read_record1();
449  
450  if(bcc_error) pstr(text5);
451  else pstr(text6);
452  
453  PC = 0x8000;
454  key_data();
455  
456  
457  
458  }
459
460  
461  void key_reg()
462  {
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];
498    buffer[4]=0;
499    buffer[5]=0;
500    buffer[1]=0xb3; // flag DF
501    buffer[0]=0x0f; //
502 }
503
504 void reg_xp()
505 {
506     n = user_xp; // xp
507     buffer[2]= convert[n&0xf];
508     n = n>>4;
509     buffer[3]= convert[n&0xf];
510     buffer[4]=0;
511     buffer[5]=0;
512     buffer[1]=0x13; // XP
513     buffer[0]=0x1F; //
514 }
515
516 void reg_r3()
517 {
518     hex4(user_r3);
519     buffer[1]=0x03;
520     buffer[0]=0xba;
521 }
522
523 void reg_r4()
524 {
525     hex4(user_r4); //
526     buffer[1]=0x03; //
527     buffer[0]=0x36;
528 }
529
530 void reg_r5()
531 {
532     hex4(user_r5); //
533     buffer[1]=0x03; //
534     buffer[0]=0xae;
535 }
536
537 void reg_r6()
538 {
539     hex4(user_r6); //
540     buffer[1]=0x03; //
541     buffer[0]=0xaf;
542 }
543
544 void reg_r7()
545 {
546     hex4(user_r7); //
547     buffer[1]=0x03; //
548     buffer[0]=0x38;
549 }
550
551 void reg_r8()
552 {
553     hex4(user_r8); //
554 }
buffer[1]=0x03; //
buffer[0]=0xbf;
}
void reg_r9()
{
    hex4(user_r9); //
    buffer[1]=0x03; //
    buffer[0]=0xbe;
}
void reg_r10()
{
    hex4(user_r10); //
    buffer[1]=0x03; //
    buffer[0]=0x3f;
}
void reg_r11()
{
    hex4(user_r11); //
    buffer[1]=0x03; //
    buffer[0]=0xa7;
}
void reg_r12()
{
    hex4(user_r12); //
    buffer[1]=0x03; //
    buffer[0]=0x8d;
}
void reg_r13()
{
    hex4(user_r13); //
    buffer[1]=0x03; //
    buffer[0]=0xb3;
}
void reg_r14()
{
    hex4(user_r14); //
    buffer[1]=0x03; //
    buffer[0]=0x8f;
}
void reg_r15()
{
    hex4(user_r15); //
    buffer[1]=0x03; //
    buffer[0]=0x0f;
}

void reg_display()
{
    switch(key)
    {   
        case 0: reg_d(); break;
// case 1: reg_df(); break;  // future work
// case 2: reg_xp(); break;
case 3: reg_r3(); break;
case 4: reg_r4(); break;
case 5: reg_r5(); break;
case 6: reg_r6(); break;
case 7: reg_r7(); break;
case 8: reg_r8(); break;
case 9: reg_r9(); break;
case 10: reg_r10(); break;
case 11: reg_r11(); break;
case 12: reg_r12(); break;
case 13: reg_r13(); break;
case 14: reg_r14(); break;
case 15: reg_r15(); break;
}
}
}
}
}
}

void enter_num()
{
if (hit==0) num=0;
{
  hit=1;
  num<<=4;
  num |= key;
  hex4(num);
}
}

void clear_buffer()
{
for (i=0; i<6; i++)
  *(buffer+i)=0;
}

void key_copy()
{
  state=5;
  hit=0;
  clear_buffer();
  buffer[2]=0xbd;
  buffer[0]=0xae;
  buffer[1]=0;
}

>Returns internal code hex keys and function keys *
char key_code(char n)
{
  char d;
  if(n == 0x10) return 0;
  if(n == 0x21) return 1;
  if(n == 0x1b) return 2;
  if(n == 0x15) return 3;
  if(n == 0x20) return 5;
  if(n == 0x1a) return 6;
if (n == 0x14) return 7;
if (n == 0x1c) return 8;
if (n == 0x1f) return 9;
if (n == 0x19) return 0xa;
if (n == 0x13) return 0xb;
if (n == 0x22) return 0xc;
if (n == 0x1e) return 0xd;
if (n == 0x18) return 0xe;
if (n == 0x12) return 0xf;
if (n == 0xc) return 0x10; // pc
if (n == 0xd) return 0x11; // reg
if (n == 0xe) return 0x12; // data
if (n == 0xf) return 0x13; // address
if (n == 6) return 0x14;
if (n == 7) return 0x15; // MUTE
if (n == 8) return 0x16; // -
if (n == 9) return 0x17; // +
if (n == 0) return 0x18; // load
if (n == 1) return 0x19; // insert
if (n == 2) return 0x1a; // delete
if (n == 3) return 0x1b; // go
if (n == 0x13) return 0x1c;
if (n == 0x1d) return 0x1d; // test
if (n == 0x17) return 0x1e; // dump
if (n == 0x00) return 0x1f;
if (n == 0x23) return 0x20; // copy
return 0xff;

void delay_beep()
{
    for (j=0; j<2; j++)
        continue;
}

void beep()
{
    *port2=0;
    for(x=0; x<60; x++)
    {
        *port1 = (char) ~0x80;
        delay_beep();
        *port1 = 0xff;
        delay_beep();
    }
}

void key_mute()
{
    flag ^= 1;
}

void key_qled()
{
    while(1)
    {
        asm(" seq \n");
        delay_ms(500);
        asm(" req \n");
    }
delay_ms(30000);
}

void key_exe()
{
    if(flag==0) beep();
    if( key>15)
    {
        if(key==0x13) key_address();
        if(key==0x12) key_data();
        if(key==0x17) key_plus();
        if(key==0x16) key_minus();
        if(key==0x10) key_PC();
        if(key==0x1b) key_go();
        if(key==0x11) key_reg();
        if(key==0x19) insert();
        if(key==0x1a) cut_byte();
        if(key==0x1e) key_dump();
        if(key==0x1d) key_test();
        if(key==0x18) key_load();
        if(key==0x20) key_copy();
        if(key==0x15) key_mute();
        if(key==0x14) key_qled();
    }
    else
    {
        switch(state)
        {
            case 1: hex_address(); break;
            case 2: data_hex(); break;
            case 3: reg_dispaly(); break;
            case 5: enter_num(); break;
            case 6: enter_num(); break;
            case 7: enter_num(); break;
        }
    }
}

void scan1()
{
    while(scan() != 0xff)
    {
        delay_ms(10);
        while(scan() == 0xff)
        {
            delay_ms(5);
            key = scan();
            key = key_code(key);
            // *gpio1=key; // debug
            if((key>=0) && (key <0x30)) key_exe();
        }
    }
}
```c
853 void init_8250()
854 {
855   *uart_lcr = 0x83;
856   *uart_divisor_lsb = 24;  // computed for 3.6864MHz
857   *uart_divisor_msb = 0;
858   *uart_fifo = 7;
859   *uart_lcr = 3;
860 }
861
862 void cout(char n)
863 {
864   while ((*uart_line_status & 0x20) == 0)
865   { *uart_buffer = n;
866   }
867
868 char cin()
869 {
870   while ((*uart_line_status & 1) == 0)
871   { c = *uart_buffer;
872   return c;
873 }
874
875 void pstr(char *s)
876 {
877   while (*s)
878   {
879     cout(*s);
880     s++;
881   }
882 }
883
884 void test_uart()
885 {
886   for (c = 0x20; c < 0x80; c++)
887     cout(c);
888 }
889
890 void newline()
891 {
892   cout(0x0a);
893   cout(0x0d);
894 }
895
896 void send_hex(char n)
897 {
898   k = n>>4;
899   k = k&0xf;
900   if (k>9) cout(k+0x37); else cout(k+0x30);
901   k = n&0xf;
902   if (k>9) cout(k+0x37); else cout(k+0x30);
903 }
904
905 void send_word_hex(int n)
906 {
907   temp16 = n>>8;
908   k = temp16&0xff;
909   send_hex(k);
910   k = n&0xff;
911   send_hex(k);
912 }
913
914 void key_dump()
915 {
916```
int j, p;
dptr = (char *)PC;

for (j = 0; j < 16; j++)
{
    newline();
    send_word_hex(PC);
    cout(':');
    for (p = 0; p < 16; p++)
    {
        send_hex(*(dptr + p));
        cout('0x20');
    }

    cout('0x20');
}

for (p = 0; p < 16; p++)
{
    q = *(dptr + p);
    if (q >= 0x20 && q < 0x80) cout(q);
    else cout('.');
}
dptr += 16;
PC += 16;
}

// PC = dptr;
key_address();
char nibble2hex(char c)
{
    char n;
    if (c < 0x40) return (c - 0x30);
    else return (c - 0x37);
}

char gethex()
{
    char a, b;
    a = cin();
    b = cin();
    a = nibble2hex(a) << 4;
    b = nibble2hex(b);
    a = a | b;
    bcc = bcc + a; /* compute check sum */
    return (a);
}

int get16bitaddress()
{
    unsigned int load_address;
    load_address = 0;
    load_address |= gethex();
    load_address <<= 8;
    load_address |= gethex();
    return load_address;
}
void read_record1()
{
    char x;
    char byte_count;
    int address16bit;
    bcc_error=0;
    do{
        bcc =0;
        while(cin()!=':');
        byte_count = gethex(); /* only data record */
        dptr = (char *) get16bitaddress();
        record_type= gethex();
        // now read byte
        for(x=0; x<byte_count; x++)
        {
            *(dptr+x) = gethex();
        }
        // bcc = gethex();
        // bcc = ~bcc; /* one's complement */
        // bcc= bcc; // two's complement
        *gpio1=bcc; /* loading indicator */
        // save_bcc= bcc;
        // if(save_bcc != gethex()) bcc_error=1;
    }while(record_type==0);
}

//---------------------------------------------------------------------
//---------------------------- LCD drivers ------------------------------
/* LCD driver */
void LcdReady()
{
    timeout=0;
    while(((*LCD_crd& 0x80 )== 1 ) && (timeout< 500 ))
        ++timeout;
}
void clr_screen()
{
    LcdReady();
    *LCD_cwr=0x01;
}

void goto_xy(int x, int y)
{
    LcdReady();
    switch(y)
    {
    case 0: *LCD_cwr=0x80+x; break;
    case 1: *LCD_cwr=0xC0+x; break;
    case 2: *LCD_cwr=0x94+x; break;
    case 3: *LCD_cwr=0xd4+x; break;
    }
}

void InitLcd()
{
    LcdReady();
    *LCD_cwr=0x38;
    LcdReady();
    *LCD_cwr=0x0C;
    clr_screen();
    goto_xy(0,0);
    delay_ms(100);
}

void PutLCD(char *str)
{
    char i;
    for (i=0; str[i] != '\0'; i++)
    {
        LcdReady();
        *LCD_dwr=str[i];
    }
}

void putch_lcd(char ch)
{
    LcdReady();
    *LCD_dwr=ch;
}

void load_user_subroutines()
{
    PC = 0xf000;
    dptr = (char *) PC;
    for (i=0; i<17 ;i++ )
    {
        *(dptr+i) = user_delay[i];
    }
    PC = 0xf020;
    dptr = (char *) PC;
    for (i=0; i<sizeof(interrupt_test) ;i++ )
    {
        *(dptr+i) = interrupt_test[i];
    }

    //------------------------- end of LCD drivers --------------------------

void service_break()
{
  asm(
    " plo r2 ; save D first \n"
    " ldAD r1, _USER_D \n"
    " glo r2 \n"
    " str r1 \n"
    " ldAD r1, _USER_R3 \n"
    " ghi r3 \n"
    " str r1 \n"
    " inc r1 \n"
    " glo r3 \n"
    " str r1 \n"
    " ldAD r1, _USER_R4\n"
    " ghi r4 \n"
    " str r1 \n"
    " inc r1 \n"
    " glo r4 \n"
    " str r1 \n"
    " ldAD r1, _USER_R5\n"
    " ghi r5 \n"
    " str r1 \n"
    " inc r1 \n"
    " glo r5 \n"
    " str r1 \n"
    " ldAD r1, _USER_R6\n"
    " ghi r6 \n"
    " str r1 \n"
    " inc r1 \n"
    " glo r6 \n"
    " str r1 \n"
    " ldAD r1, _USER_R7\n"
    " ghi r7 \n"
    " str r1 \n"
    " inc r1 \n"
    " glo r7 \n"
    " str r1 \n"
    " ldAD r1, _USER_R8\n"
    " ghi r8 \n"
    " str r1 \n"
    " inc r1 \n"
    " glo r8 \n"
    " str r1 \n"
    " ldAD r1, _USER_R9\n"
    " ghi r9 \n"
    " str r1 \n"
    " inc r1 \n"
    " glo r9 \n"
    " str r1 \n"
    " ldAD r1, _USER_R10\n"
    " ghi r10 \n"
    " str r1 \n"
    " inc r1 \n"
    " glo r10 \n"
  );
" str r1 \n"
" ldAD r1, __USER_R11"r11
" ghi r11 \n"r11
" str r1 \n"
" inc r1 \n"
" glo r11 \n"
" str r1 \n"
" ldAD r1, __USER_R12"r12
" ghi r12 \n"r12
" str r1 \n"
" inc r1 \n"
" glo r12 \n"
" str r1 \n"
" ldAD r1, __USER_R13"r13
" ghi r13 \n"r13
" str r1 \n"
" inc r1 \n"
" glo r13 \n"
" str r1 \n"
" ldAD r1, __USER_R14"r14
" ghi r14 \n"r14
" str r1 \n"
" inc r1 \n"
" glo r14 \n"
" str r1 \n"
" ldAD r1, __USER_R15"r15
" ghi r15 \n"r15
" str r1 \n"
" inc r1 \n"
" glo r15 \n"
" str r1 \n"
" lbr 0 ; return to system monitor\n"
}

void main()
{
*segment = 0;
*digit = 0xff;
*gpio1 = 0;
*tick= 0;
*state = 0;
init_8250();
InitLcd();
if(warm != 0xaaaa)
{
    flag =0;
    warm = 0xaaaa;
}
blink_q();

PC = 0x8000;
save_PC = 0x8000;
// asm("ldAD r1,9000h \n"); // load interrupt vector at 9000h
load_user_subroutines();
dptr= (char *) PC;
// test_uart();
PutLCD(text2);
goto_xy(0,1);
PutLCD(text3);
pstr(text1);

buffer[0]=0;
buffer[1]=0;
buffer[2]=convert[2];
buffer[3]=convert[0];
buffer[4]=convert[8];
buffer[5]=convert[1];
while(1){
    scan1();
}

void setRAM()
{
    asm("ROM_END1: EQU $\n");
    asm("org 0fe00h\n"); // space for monitor variables
}
NOTE