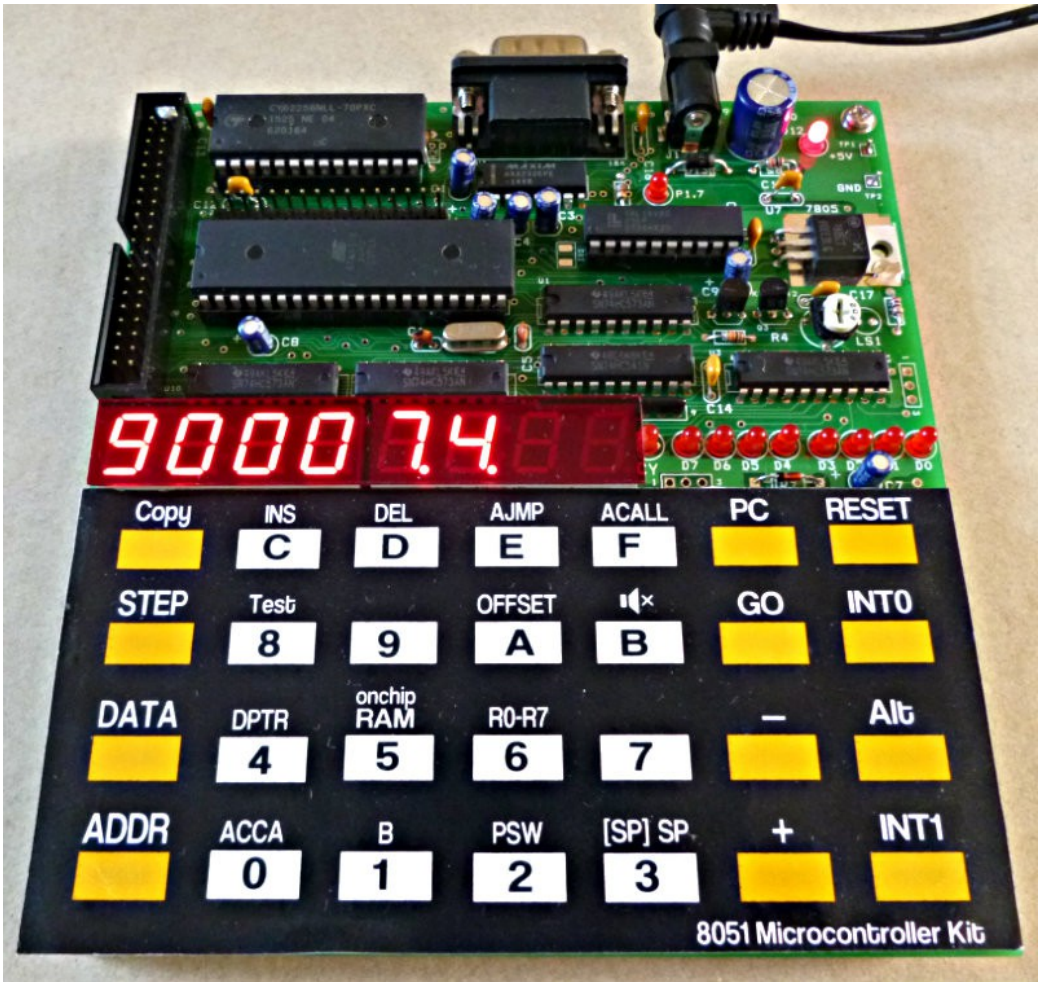
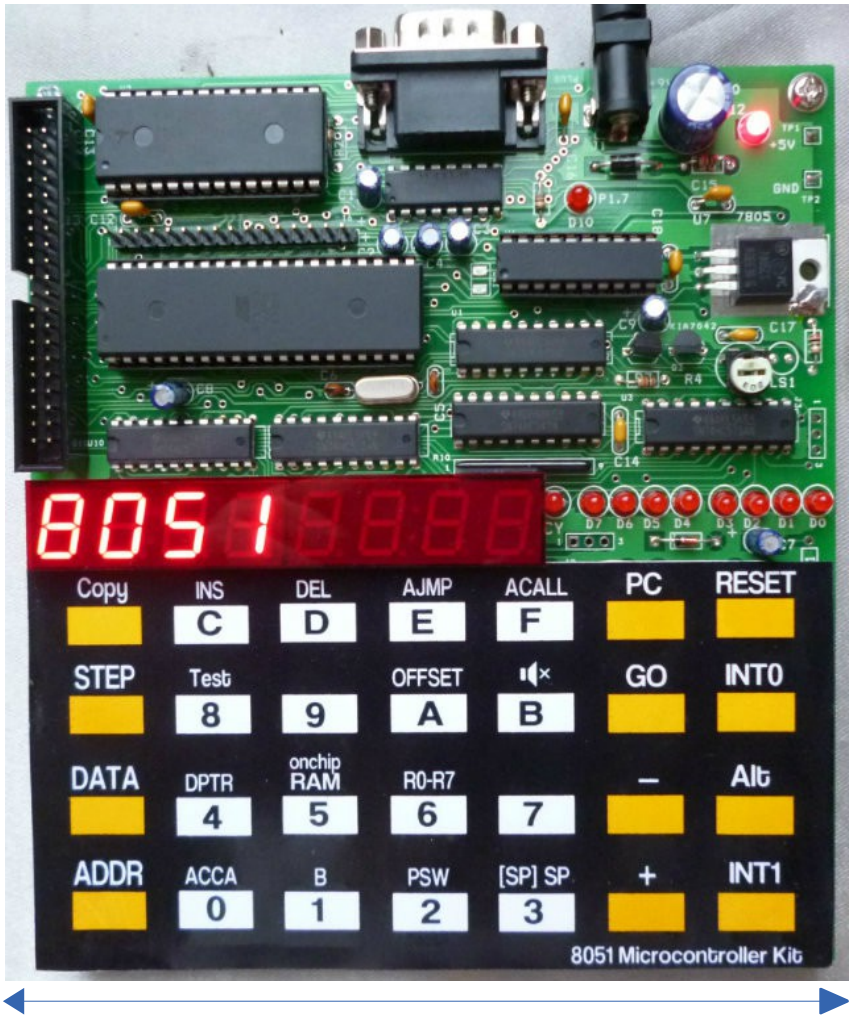
8051 Microcontroller Kit

User's Manual



Rev 2.0 December 10, 2015

Kit Dimension



5.70 (145)

5.30 (135)

inches(mm)

Contents

OVERVIEW

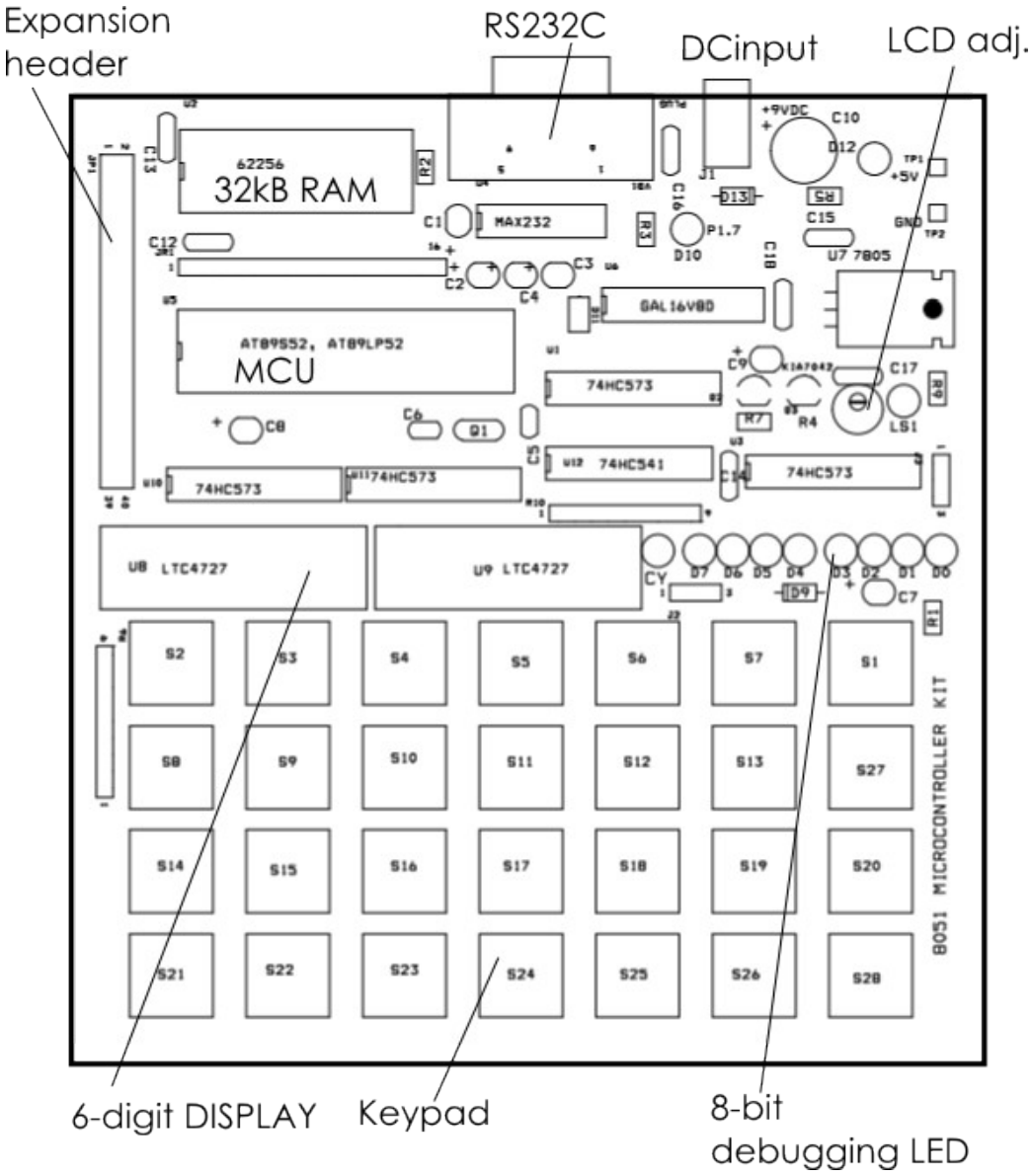
The 8051Microcontroller kit is the educational kit designed for self-learning how to program the 8051 microcontroller with machine code. The kit itself is the digital computer that uses 8051 as the CPU with 8-kB monitor program and a 32kB user RAM. We can learn the operation of the 8051 instruction by entering the machine code to memory and test it directly. The single step running controlled by system monitor allows single instruction execution. The result of code running can be tested with user registers and memory examination easily.

SAFTY INFORMATION

- The kit can be powered with AC adapter having DC output >7VDC. Common AC adapter with +9VDC is a good selection.
- Using higher voltage from the AC adapter will make 7805 becomes HOT!
- For switching AC adapter that accepts 100VAC to 240VAC, ensure the Earth grounding must be provided.
- The AC adapter that uses low frequency transformer is the best for young students.
- The kit has protection diode to prevent wrong polarity jack. The center pin must be positive.
- Plugging or removing the LCD module to the kit must be done when the board is powered off!

BOARD LAYOUT

The kit is built with double sides plate through hole PCB.

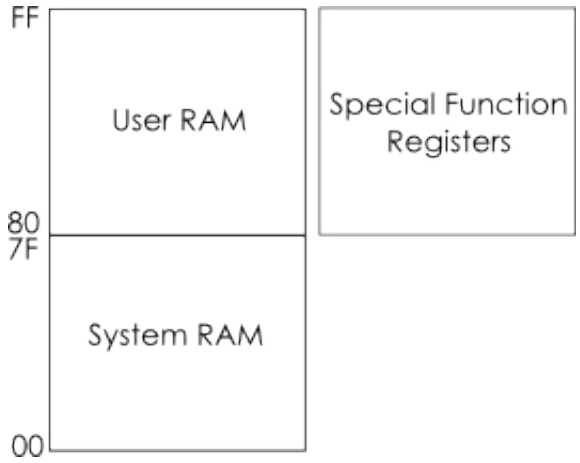


MEMORY LAYOUT

Onchip data memory: The microcontroller chip has 256 bytes onchip RAM. Location 00-7F are for system RAM and location 80-FF are user RAM. The special function register also has the same logical address 80-FF but only direct addressing will be used to access these registers. User RAM can be accessed only with indirect addressing.

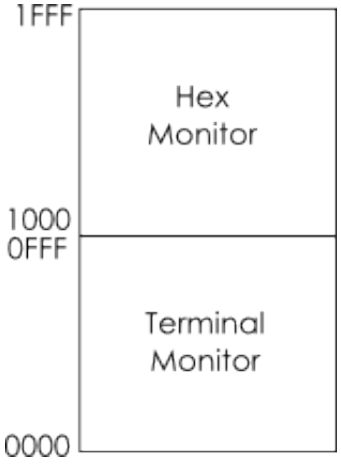
For the first 256 bytes, 00-7F we can use both direct or indirect addressing.

The onchip RAM is designed for fast access variables.

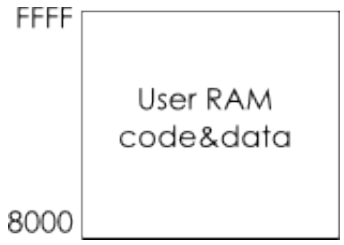


Onchip code memory: The onchip 8kB Flash memory is for monitor program. The first 4-kB is terminal monitor. It will need VT100 terminal with UART format, 9600 8n1 for connecting the kit.

The second 4-kB is hex monitor that uses display and keypad as the output/input device.

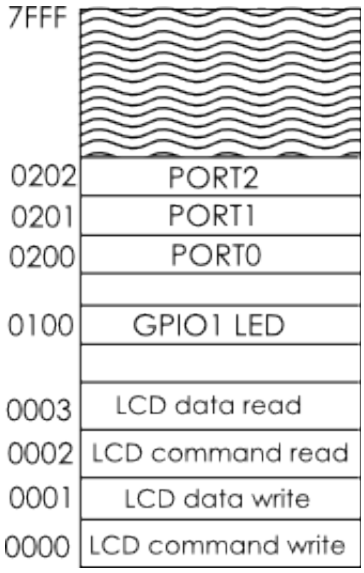


External memory: The kit has 32kB external memory using 62256 static RAM. It is located for both code and data memory space. Under monitor program running, the 62256 chip will be external data memory. So user can write the code to the 62256 chip. Under program running, it will be external code memory. The chip is enabled with RD && PSEN.



I/O LAYOUT

I/O layout: The I/O devices for 8051 kit uses the first 32kB external data memory space for decoding. Accessing to these devices can be done via MOVX instructions.



For example, GPIO1 LED is the 8-bit binary display. Suppose we want to display accumulator content, we can write,

```
74 1F       MOV A,#1FH
90 01 00    MOV DPTR,#0100H
F0          MOVX @DPTR,A
```

PORT0 to PORT2 are used to control display and keypad. We can learn how to scan display and keypad from the monitor source code listing.

From location 300 to 7FFF are available for I/O expansion.

DISPLAY and MONITOR COMMANDS

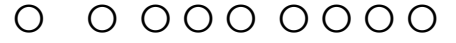
The kit provides two types of display. The first one is hex number display. It is 6-digit 7-segment LED.

The first four digits is for ADDRESS field and two digits for DATA field.

Shown memory location 9000 has data 74.

The hex display also used to display user registers, onchip RAM and settings for start, end, destination address.

The 2nd display is 8-bit debugging LED and one bit carry flag display.



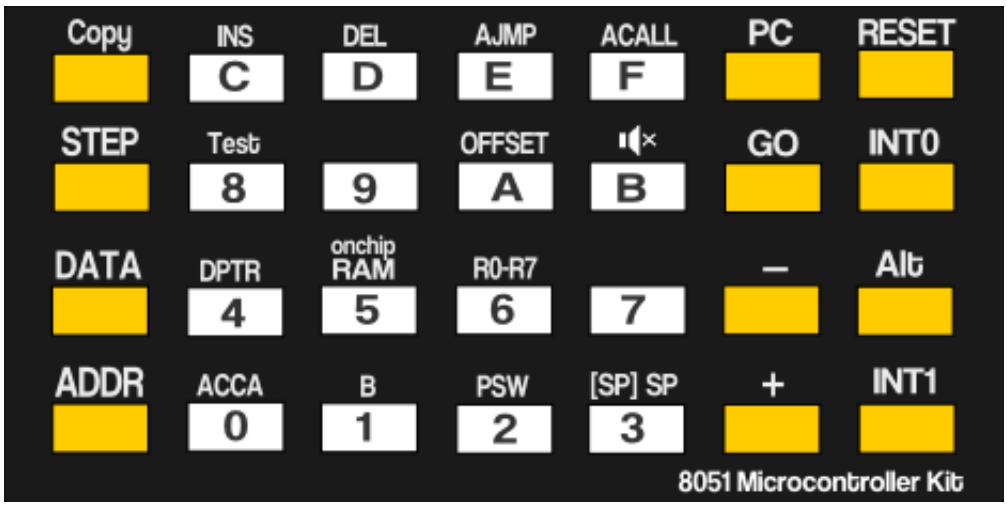
Under single-step operation, the 8-bit LED will show the Accumulator content. And the CY LED will show carry flag.

Under user program running, the 8-bit LED is named GPIO1 LED having location at the external memory at 100H. We will learn how to use it in the programming book.

The 8-bit LED represented by D0 to D7 symbol can be used to test the code and check the result with binary number display directly.

The hex display instead will help entering instruction in hex code easily.

The keypad layout is shown below. The kit has HEX keys, 0-F, Functions keys and CPU control keys.



CPU control keys:

RESET press RESET will force CPU to restart at the monitor program running. The text 8051 will show on the display.

INT0 Press INT0 will make INT0 pin to logic '0'. Release will make INT0 to logic '1'.

INT1 Press INT1 will make INT1 pin to logic '0'. Release will make INT1 to logic '1'.

Function keys:

PC set user Program Counter to 9000. Under single-step running, the PC key will set display address to current PC.

GO Jump from monitor program to user program at current display address. User registers will be loaded the jump to user code.

- Decrement display address by one.

+ Increment display address by one.

Copy Copy block of external data memory. Used with key + and GO for entering START, END and DESTINATION address.

STEP Execute current address with single instruction. The CPU registers will be saved to user registers.

DATA Set hex entry mode to data field.

ADDR Set hex entry mode to address field.

ALT Alternate function set, used with hex key.

ALT, 0 Display user Accumulator.

ALT, 1 Display user register B.

ALT, 2 Display user register PSW.

ALT, 3 Display user register SP and STACK's content [SP].

[SP] SP

Shown SP = 60 and STACK's content = 75.

ALT, 4 Display user register DPTR.

Shown DPTR = E000.

ALT, 5 Display onchip RAM 00-FF. Used with key + and – for changing location. Its contents can be modified with hex key directly.

CAUTION: modify some location that used by system monitor will cause system unstable. Press RESET will restart the CPU.

ALT, 6 Display user registers R0 to R7. Used with key + and – for changing location.

ALT, 8 Run hardware test program.

ALT, A Find OFFSET byte by entering the destination location and press key GO.

ALT, B BEEP ON/OFF.

ALT, C Insert one byte after current display address.

ALT, D Delete one byte at current display address.

ALT, E Find AJMP hex code (2 bytes) used with key GO.

ALT, F Find ACALL hex code (2 bytes) used with key GO.

Hex keys:

0-F Hex digit represents 4-bit binary number from 0000 to 1111.

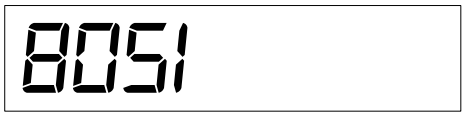
BASIC OPERATION

This section shows how to play the kit using hex monitor mode. No need any computers.

Power up the kit: the kit can be powered with any AC adapter having DC output approx. +9V >200mA. The center pin is POSITIVE(+). The example one is low frequency transformer. It is quite safe for students.

Enter the hex code:

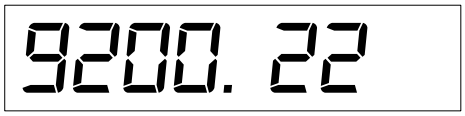
Power on reset display will be 8051 displaying.



Press key PC, the display will show location 9000 with its content.

Dot at data field indicates the HEX key entry will be entered in DATA field. We can press any hex key to modify it.

To change the ADDRESS field, press key ADDR and enter the hex number.



Single Stepping:

Key STEP when pressed will make the microcontroller executes the single instruction then returns to monitor, saves the CPU registers to user registers. We can check the operation of 8051 instruction easily.

Let use try the test code below.

ADDR	HEX	instruction
9000	04	INC A
9001	01 00	AJMP 9000

This small program will increment the accumulator by one with INC A instruction. Then repeat it by jumping to 9000 with AJMP 9000.

Enter the hex code 04, 01 and 00 to memory at 9000, 90001 and 9002.

Press PC to set start PC to 9000. Press STEP. Keep press it. What is happening?

The debug 8-bit LED will show the accumulator content. We will see the binary number increment by one every-time the code 04 was executed.

Can we make the running LED?

Here is the test code for running LED.

```
ADDR  Hex     label   instruction
9000    74 01         MOV A,#01
9002    23      loop  RL A
9003    01 02         AJMP loop
```

Now the hex code are 74, 01, 23, 01 and 02.

Press RESET, then PC, Enter the hex code to the memory. When finished, press PC the STEP. Keep press it, did you see the running LED?

If we change 23 to 33, what is happening?

Run user code:

Key GO will make the CPU to jump from monitor program to user code at the current display. Fast key is PC to set current display to 9000. The example code below will show how to use key GO.

```
0001   0000              P1.7    .EQU 97H
0002   0000
0003   9000                      .ORG 9000H
0004   9000
0005   9000 B2 97       MAIN    CPL P1.7
0006   9002 11 06               ACALL DELAY
0007   9004 01 00               AJMP MAIN
0008   9006
0009   9006 7F 32       DELAY    MOV R7,#50
0010   9008 7E 00       DELAY1   MOV R6,#0
0011   900A DE FE               DJNZ R6,$
0012   900C DF FA               DJNZ R7,DELAY1
0013   900E 22                  RET
0014   900F
0015   900F                      .END

tasm: Number of errors = 0
```

The listing above shows the output from TASM assembler as the listing file. We will learn how to use the assembler for longer program on later.

For now let us try enter the hex code to memory from location 9000 to 900E. The hex code will be B2, 97, 11, 06, 01, 00, 7F, 32, 7E, 00, DE, FE, DF, FA, 22.

When finish, press key PC, the display will show 9000. Press key GO to run.

Did you see the blinking at P1.7 LED?

Can you change blinking rate? How?

Finding the OFFSET byte:

For SJMP, JUMP with flag or CJNE instructions, the 2nd byte or the 3rd byte will be OFFSET byte. It is the distant between destination and current program counter.

OFFSET = DESTINATION – CURRENT PROGRAM COUNTER

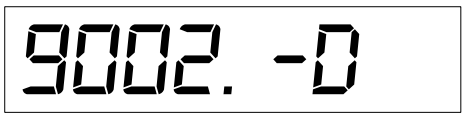
The kit provides key for finding such OFFSET byte easily.

Suppose we enter below code.

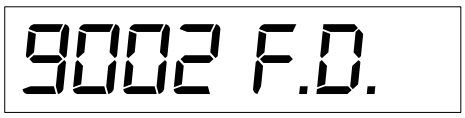
```
9000   04      INC A
9001   80 XX   SJMP 9000
```

We enter 04 at 9000, 80 at 9001 and 9002 is the OFFSET byte.

Shown 9002 has C3. To find OFFSET byte, press ALT, OFFSET.



Enter the destination address, which is 9000. Then press key GO.



The OFFSET byte at 9002 will be FD or -3.

Similarly for CJNE, but the OFFSET byte will be the 3rd byte

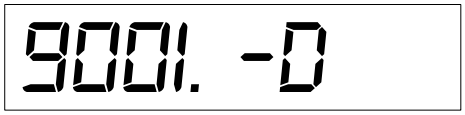
Finding HEX CODE for AJMP or ACALL instructions:

The AJMP and ACALL instructions will merge the destination address with only 11 bits to the instruction hex code. For LJMP and LCALL it will use 16-bit destination. The kit provides key for finding such hex code for AJMP and ACALL easily.

```
9000   04      INC A
9001   XX YY   AJMP 9000
```

We enter 04 at 9000 then at 9001 we will find the hex code for AJMP 9000.

Press ALT, AJMP



Enter 9000 then press key GO.

Press key +, we will get 9002 = 00.

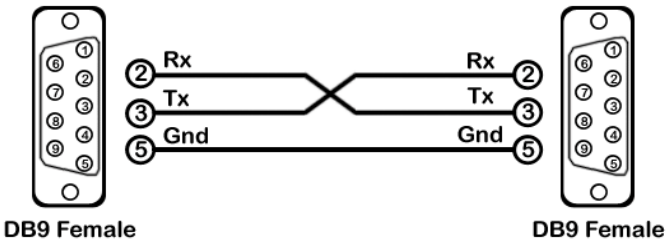
Thus the AJMP 9000 will have the two bytes hex code, 01, 00.

Similarly for ACALL instruction. Steps will be the same.

Connecting VT100 terminal

The kit has RS232 port with male connector. We can use this port to connect another computer for data communication.

The cable is cross cable with DB9 female connector for both ends. We can make it or buy from computer shop.

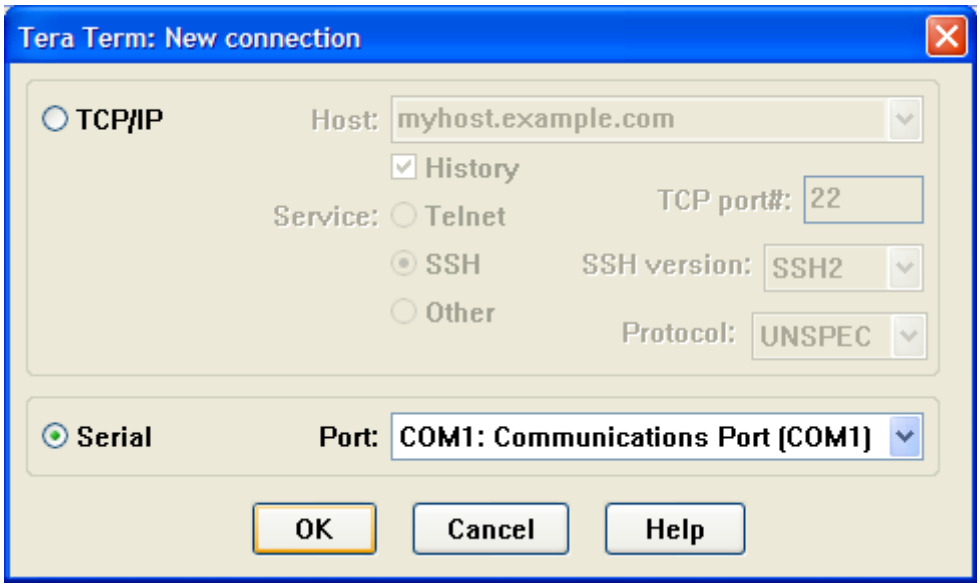


We can use the laptop with RS232 port and have the VT100 terminal emulator program to connect to the kit.

If your computer does not have the COM port, you can use the USB to RS232 port converter.

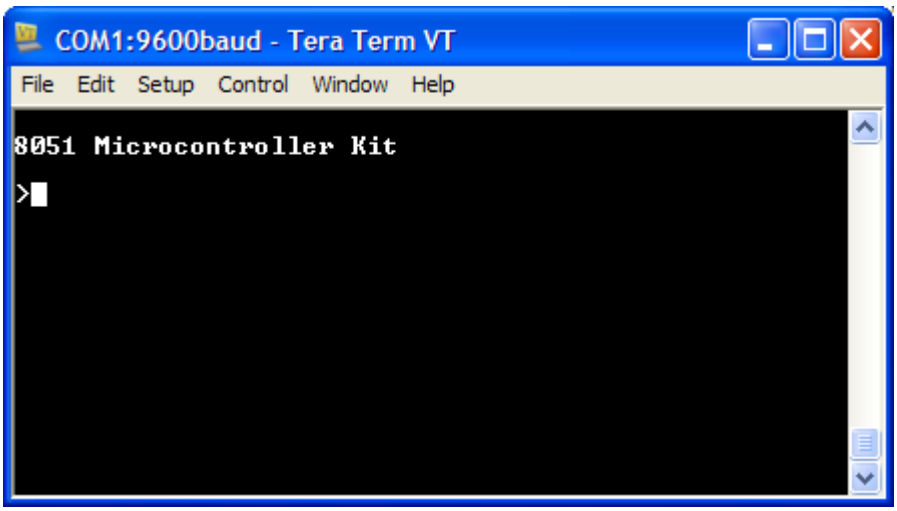
You may download free terminal program, teraterm from this URL,
http://ttssh2.sourceforge.jp/index.html.en

Click and run Tera Term.

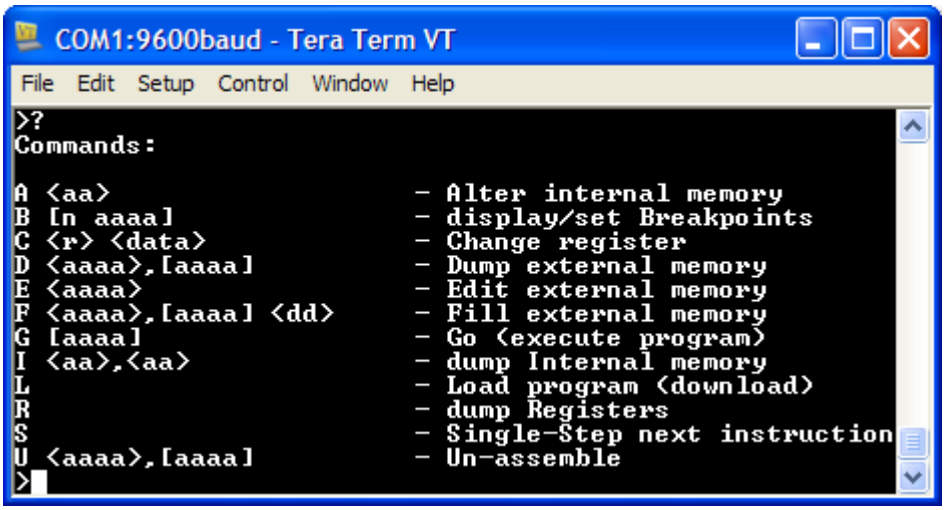


Select Serial, shown the available serial port is COM1.
The default settings are 9600 bps, 8 data bit, no parity and one stop bit.

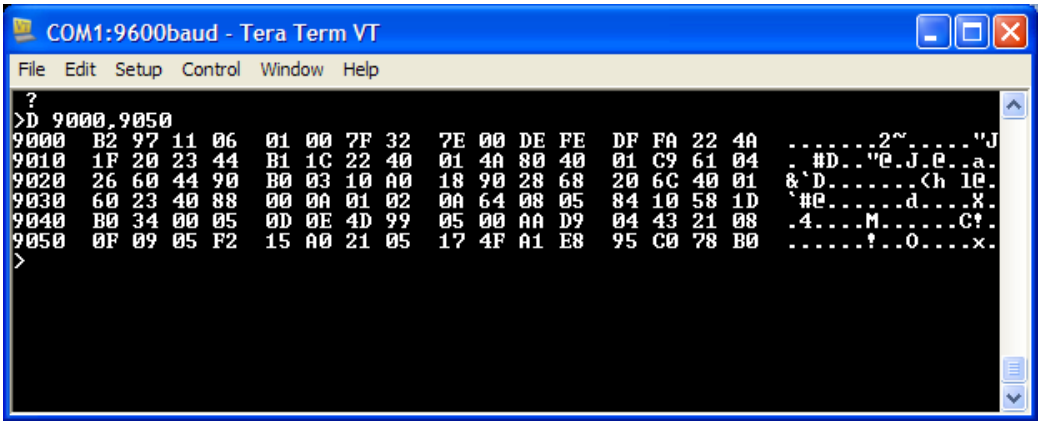
Press ENTER or SPACEBAR.



The kit will enter serial connected automatically. The prompt shows the user enter commands. Press ? For help.



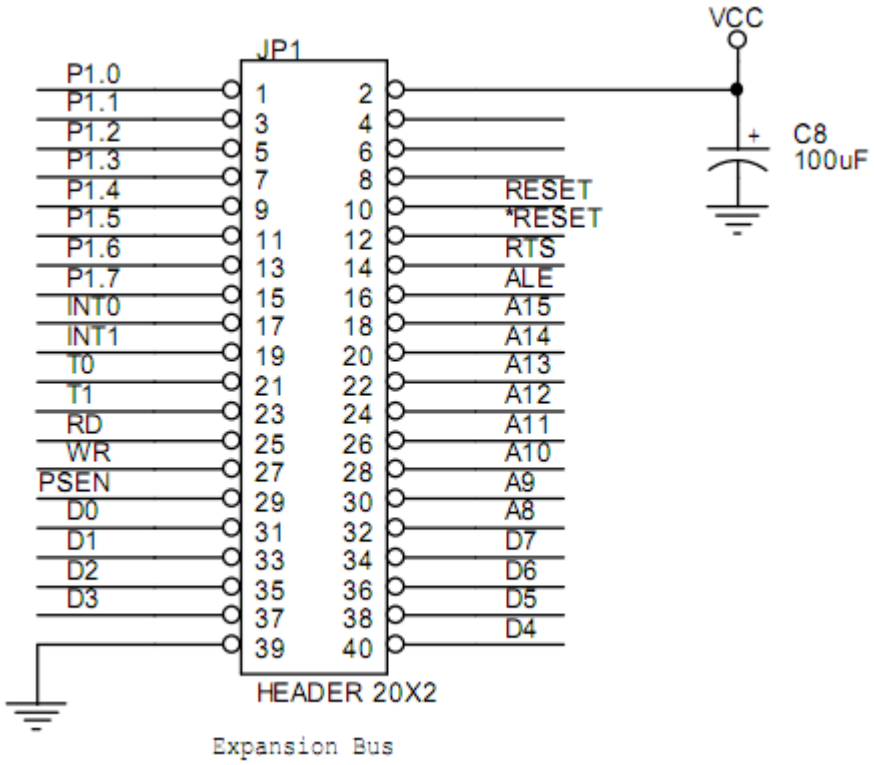
We can dump the external memory with command >D 9000,9020



Command L will use for Intel hex file downloading.

Expansion Header

The expansion header is 40-pin socket. Students can make the interfacing board using the 8051 port P1 or memory mapped I/O from 300H to 7FFFH. The interrupt pins and timer input are also available for experimentation.



Connecting LCD

The kit has 16-pins LCD bus interface header. We can play with the HD44780 based text LCD module.

Caution:

1. plugging or removing the LCD module must be done when the kit was powered off.

2. The small POT, R4 is contrast adjustment. Use a small tip screw driver to adjust until the black line appeared.

Testing is very easy with key, ALT, TEST. The text "8051 Microcontroller Kit" will be displayed. Adjust R4 for nice view.

Any text LCD having HD44780 controller can be used. Refer to its data sheet for the display location addressing.

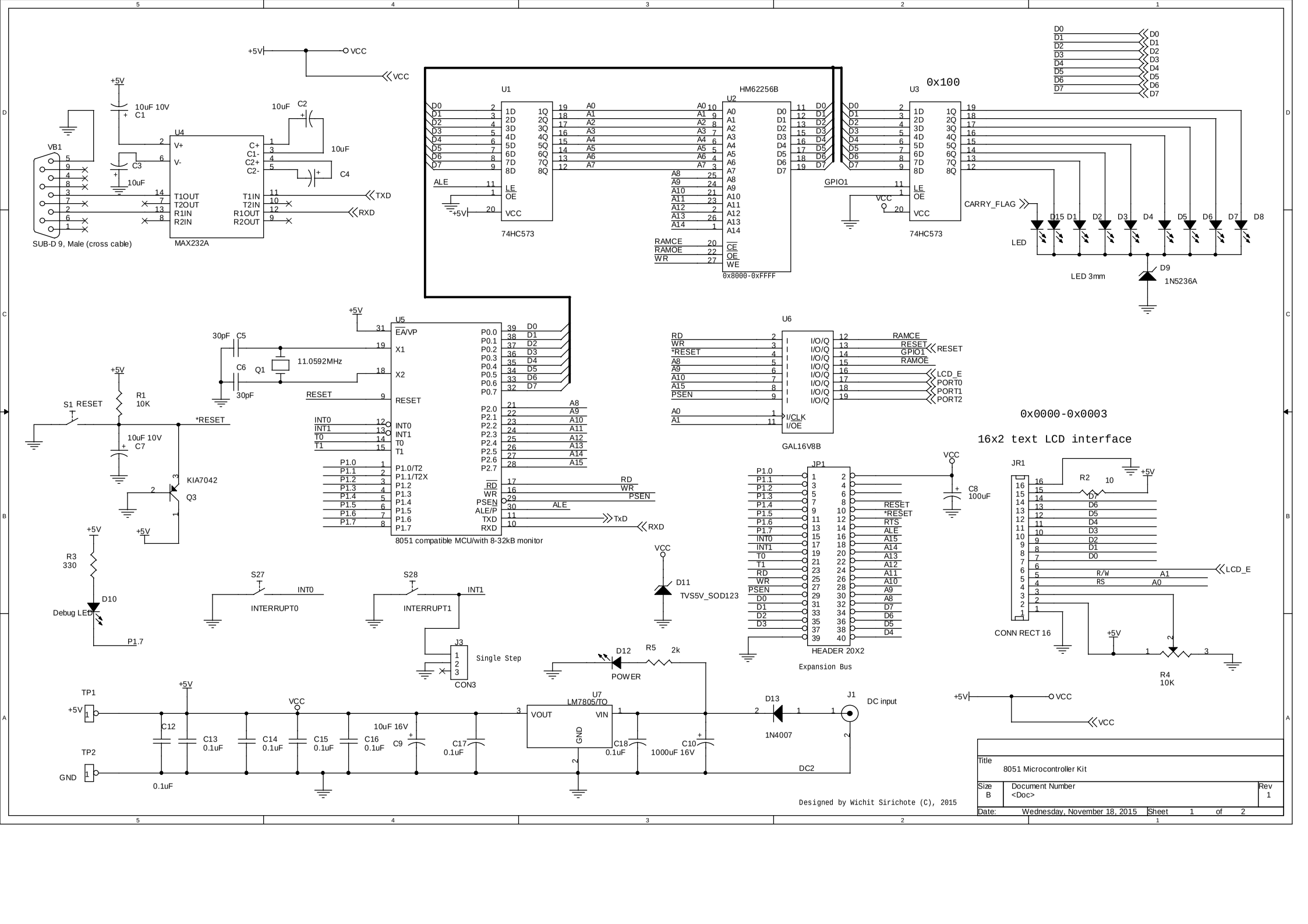
LOGIC PROBE POWER SUPPLY

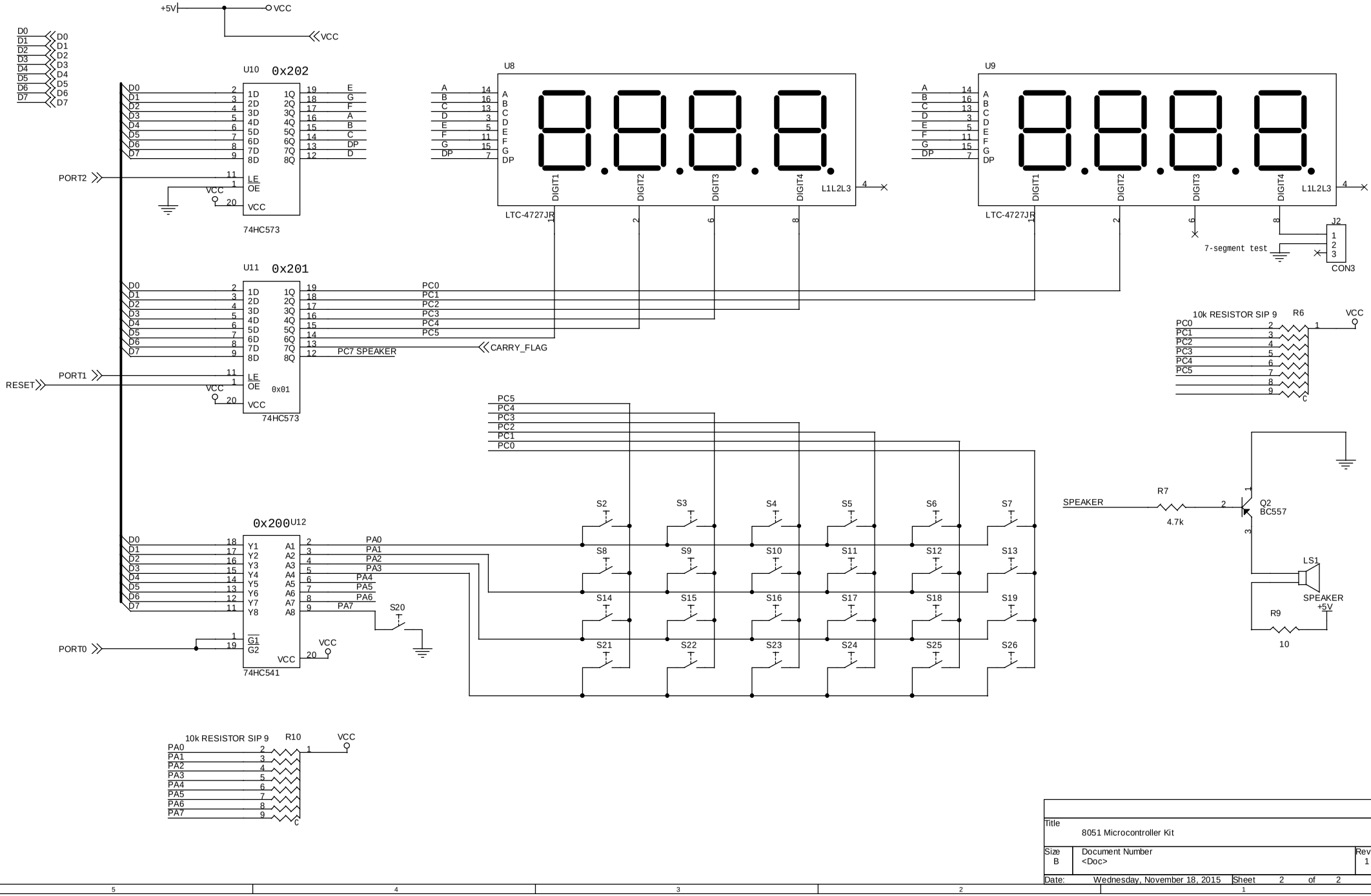
The kit provides test points TP1(+5V) and TP2(GND) for using the logic probe. Students may learn digital logic signals with logic probe easily. Red clip is for +5V, TP1 and Black clip for GND, TP2.

HARDWARE SPECIFICATIONS

Processor	AT89S52 microcontroller, 11.0592MHz 8kB monitor program.
Memory	256 bytes onchip memory, 32kB external code and data memory, 62256
I/O port	Onchip P1 and P3.
Timer/counter	16-bit timer 0,1,2.
Watchdog	Onchip Watchdog reset
RESET	Brownout reset chip, KIA4072.
Decoder	PLD memory and I/O decoder chip, GAL16V8
Display	Red color, 6-digit high brightness 7-segment LED, LTC4727 8-bit debugging LED, Carry flag display and P1.7 LED.
Keypad	16-hex keys, 9 function keys and 3 CPU control keys.
Serial port	RS232C 9600 8n1, male connector.
LCD	16-pin CPU bus interface for text LCD.
Expansion	40-pin header.
Electrical	Power consumption 80mA @+5VDC.
Mechanical	Dimensions: 5.7 x 5.3 inches

SCHEMATIC, PARTS LIST


SUB-D 9, Male (cross cable)
MAX232A
74HC573
HM62256B
0x8000-0xFFFF
0x100
LED
LED 3mm
1N5236A
8051 compatible MCU/with 8-32kB monitor
11.0592MHz
KIA7042
GAL16V8B
0x0000-0x0003
16x2 text LCD interface
HEADER 20X2
Expansion Bus
CONN RECT 16
INTERRUPT0
INTERRUPT1
Single Step
POWER
TVS5V_SOD123
LM7805/TO
1N4007
DC input
Designed by Wichit Sirichote (C), 2015
Title
8051 Microcontroller Kit
Size
B
Document Number
<Doc>
Rev
1
Date:
Wednesday, November 18, 2015
Sheet
1
of
2



Title	8051 Microcontroller Kit	
Size B	Document Number <Doc>	Rev 1
Date: Wednesday, November 18, 2015	Sheet 2 of 2	

PARTS LIST

Semiconductors

U2 HM62256B, 32kB SRAM
U4 MAX232A
U5 AT89S52 microcontroller
U6 GAL16V8D
U7 LM7805/TO
U9,U8 LTC-4727JR, 7-segment
U12 74HC541
U1,U3,U10,U11 74HC573
Q2 BC557 or BC327
Q3 KIA7042
D1,D2,D4,D5,D6,D7,D8,D15 LED
D3 LED 3mm
D9 1N5236A
D10 Debug 3mm LED
D11 TVS5V
D12 POWER LED
D13 1N4007

LS1 SPEAKER
Q1 11.0592MHz Xtal

S1 RESET switch
S2,S3,S4,S5,S6,S7,S8,S9, tact SW
S10,S11,S12,S13,S14,S15,
S16,S17,S18,S19,S20,S21,
S22,S23,S24,S25,S26
S27 INTERRUPT0
S28 INTERRUPT1

All switches are 12mm TACT switch

TP1 +5V
TP2 GND
VB1 SUB-D 9, Male (cross cable)

PCB double side plate through hole
LED cover Clear RED color acrylic plastic
Keyboard sticker printable SVG file

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

R1,R4 10K
R9,R2 10 or 5
R3 330, or 680
R5 2k or 1k
R10,R6 10k RESISTOR SIP 9
R7 4.7k

Capacitors

C1,C7 10uF 10V
C2,C3,C4 10uF
C5,C6 30pF disceramic
C8 100uF or 10uF 16V
C9 10uF 16V
C10 470uF 25V
C12,C13,C14,C15,C16 0.1uF
C17,C18 0.1uF multilayer
C12 10uF 10V electrolytic
C19,C18 100nF disc ceramic

Additional parts

JP1 HEADER 20X2
JR1 CONN RECT 16
J1 DC input JACK
J2,J3 CON3 (do not populate)

Monitor Source Code Listing

```
// Monitor source code for 8051 Microcontroller kit v1.0
// (C) 2015 Wichit Sirichote

// project files: STARTUP.A51
//                main.c
//                C51S.LIB is needed for compiling with source code +asm code
//                         C51 option must tick Generate Assembler SRC and Assemble SRC

// Microcontroller chip: 89S52, Xtal frequency 11.0592MHz
// Monitor code is stored in 8kB onchip Flash memory
// External memory 32kB are decoded as code and data memory 0x8000-0xffff
// First 128 bytes internal RAM were used for system variables and stack
// Second 128 bytes are free for user program, but need indirect addressing only. Direct
// is for special function registers

// serial port settings is 9600 8n1
// default mode is hexkey monitor
// Automaticaly connected to 9600 terminal by key SP or ENTER hit
// commented statements were used for program developemnt


#include <REGX52.H>
#include <stdlib.h>

// two bits shared with digit control at port1
#define speaker 0x80
#define carry_flag 0x40
#define warm_code 0xff

xdata char gpio1 _at_ 0x100;
xdata char port0 _at_ 0x200;   // keypad
xdata char port1 _at_ 0x201;   // digit
xdata char port2 _at_ 0x202;   // segment

#define BUSY 0x80
xdata char LCD_command_write _at_ 0x0000;
xdata char LCD_data_write _at_ 0x0001;
xdata char LCD_command_read _at_ 0x0002;
xdata char LCD_data_read _at_ 0x0003;

//xdata char connect;
int PC;
int display_PC;
char display_RAM; // current intenal ram address

int SAVE_PC;

char code convert[16]= {0xBD,0x30,0x9B,0xBA,0x36,0xAE,0xAF,0x38,0xBF,0xBE,0x3F,0xA7,0x8D,
char code keycode[25]={0x10,0x11,0x12,0x13,0x0f,0x0b,7,3,0x0e,0x0a,6,2,0x0d,9,5,1,0x0c,8,


char buffer[6]; // display buffer

unsigned char n;
int j;
unsigned char k;
char i,u,q,o,key;
unsigned int temp;

// dptr is used for accessing external data memory
char xdata *dptr;
char code *dptr2; // for code memory access

char idata *r0; // for internal memory access

char xdata state,hit;
unsigned int xdata start,end,destination;

// user registers
char user_ACC, user_B,user_PSW, user_DPH,user_DPL;
char user_r0,user_r1,user_r2,user_r3,user_r4,user_r5,user_r6,user_r7;
char user_sp,system_sp;

char step,auto_key,debug_cy;
unsigned int temp2;
char user_register;
```

```
77  char beep_flag;
78  char tick,s;
79
80
81  //function prototype
82
83  char scan();
84
85
86  LcdReady()
87  {
88  while(LCD_command_read&BUSY)
89  continue; // wait until busy flag =0
90  }
91  void clr_screen(void)
92  {
93  LcdReady();
94  LCD_command_write=0x01;
95  }
96  goto_xy(char x,char y)
97  {
98  LcdReady();
99  switch(y){
100 case 0 : LCD_command_write=0x80+x; break;
101 case 1 : LCD_command_write=0xC0+x; break;
102 case 2 : LCD_command_write=0x94+x; break;
103 case 3 : LCD_command_write=0xd4+x; break;
104 }
105 }
106 InitLcd(void)
107 {
108 LcdReady();
109 LCD_command_write=0x38;
110 LcdReady();
111 LCD_command_write=0x0c;
112 clr_screen();
113 goto_xy(0,0);
114 }
115 char *Puts(char *str)
116 {
117 unsigned char i;
118 for (i=0; str[i] != '\0'; i++){
119 LcdReady();
120 LCD_data_write=str[i];
121 }
122 return str;
123 }
124 void putch_lcd(char ch)
125 {
126 LcdReady();
127 LCD_data_write=ch;
128 }
129
130 void LCDWriteText(char *txt) {
131   while(*txt)
132     putch_lcd(*txt++);
133 }
134
135 void LCDWriteConstText(const char *txt) {
136   while(*txt)
137     putch_lcd(*txt++);
138 }
139
140
141
142 invalid()
143 {
144  port2=0;
145  port1 = 0x80;
146  while(port0 !=-1)
147  continue;
148 }
149
150 print_error()
151 {
152   buffer[5]= 0x8f;
```

```
   buffer[4]= 3;
   buffer[3]=3;
   buffer[2]=0;
   buffer[1]=0;
   buffer[0]=0;
   state=0;
}


address_display()
{

 temp = display_PC;
 buffer[2]= convert[temp&0xf];
 temp>>=4;
 buffer[3]= convert[temp&0xf];
 temp>>=4;
 buffer[4]=convert[temp&0xf];
 temp>>=4;
 buffer[5]=convert[temp&0xf];

}

data_display()
{
    dptr =display_PC;
 dptr2=display_PC;
 if(display_PC&0x8000)n = *dptr;
 else n=*dptr2;  // access also code memory
 //gpio1 =n;
 buffer[0]= convert[n&0xf];
 n>>=4;
 buffer[1]=convert[n&0xf];
}

read_external_mem()
{
    address_display();
 data_display();
}

dot_address()
{
 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;
}


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;
}

key_address()
{
 read_external_mem();

    dot_address();
 hit=0;

 state=1;
 }

key_data()
{
 read_external_mem();
```

```
229       dot_data();
230       hit=0;
231    state=2;
232   }
233
234   key_plus()
235   {
236       if(state==10)
237    {
238    start=display_PC;
239    buffer[0]= 0x8f;
240    state=11;
241    hit=0;
242    }
243    else{
244    if(state==11)
245    {
246     end=display_PC;
247     buffer[0]=0xb3;
248     state=12;
249     hit=0;
250    }
251
252    }
253
254
255    if(state==8)
256    {
257
258     if(++user_register>7) user_register=0;
259     dump_register();
260
261
262    }
263
264
265    if(state==7)
266    {
267
268     display_RAM++;
269     dump_iRAM();
270     hit=0;
271
272
273    }
274    if(state==1 || state==2 || state==4 || state==5 || state==6)
275    {
276
277       display_PC++;
278    key_data();
279     }
280   }
281
282   key_minus()
283   {
284       if(state==8)
285    {
286
287     if(--user_register<0) user_register=7;
288     dump_register();
289
290
291    }
292
293
294    if(state==7)
295    {
296
297     display_RAM--;
298     dump_iRAM();
299
300     hit=0;
301
302
303    }
304    if(state==1 || state==2 || state==4 || state==5 || state==6)
```

```
 {
    display_PC--;
 key_data();
 }
}

key_PC()
{
     display_PC=SAVE_PC;
  key_data();


}


address_hex()
{
    if(hit==0) display_PC=0;
 {
    hit=1;

    display_PC<<=4;
    display_PC |= key;
    read_external_mem();
    dot_address();
 }
}

word_enter()
{

  if(hit==0) display_PC=0;
  {
     hit=1;
   display_PC<<=4;
   display_PC |= key;
   address_display();
   dot_address();
  }
}

data_hex()
{
   dptr = display_PC;
 n = *dptr;
 if(hit==0) n=0;
 {
  hit =1;
  n<<=4;
  n|=key;
  *dptr = n;

     if(n != *dptr) invalid();

  read_external_mem();

     dot_data();



 }



}

internal_data_hex()
{
   r0 = display_RAM;
 n = *r0;
 if(hit==0) n=0;
 {
  hit =1;
  n<<=4;
  n|=key;
  *r0 = n;
```

```
381
382       if(n != *r0) invalid();
383
384     dump_iRAM();
385
386        dot_data();
387
388
389
390    }
391
392
393
394   }
395
396
397
398   turn_off_display()
399   {
400       gpio1 = port2=0;
401    port1 = 0x80;
402   }
403
404   find_ajmp()
405   {
406       destination = display_PC;
407
408   // HL=0xf100;
409   // *HL=start;
410   // *(HL+1)=destination;
411
412           temp=destination;
413           o=temp&0xff;
414           n=temp>>=8;
415           n<<=5;
416           n|=1; // mearge the first opcode with 01
417
418     dptr = start;
419     display_PC=start;
420
421           *dptr=n;
422           *(dptr+1)=o;
423
424      if( abs(destination-start) >=2048) print_error();
425     else
426     {
427
428      read_external_mem();
429      dot_data();
430
431     }
432   }
433
434   find_offset()
435   {
436
437     destination= display_PC;
438      j = destination - start;
439      n = j&0xff;  // get only 8-bit offset
440       dptr = start-1;
441    *(dptr)=n;
442
443    display_PC= start-1;
444    read_external_mem();
445    dot_data();
446
447   }
448
449   find_acall()
450   {
451       destination = display_PC;
452
453           temp=destination;
454           o=temp&0xff;
455           n=temp>>=8;
456           n<<=5;
```

```
457           n|=0x11; // mearge the first opcode with 0x11
458
459      dptr = start;
460      display_PC=start;
461
462           *dptr=n;
463           *(dptr+1)=o;
464     if( abs(destination-start) >=2048) print_error();
465     else
466     {
467      read_external_mem();
468      dot_data();
469     }
470
471
472   }
473
474   copy_data()
475   {
476      destination=display_PC;
477      temp=end-start;
478      dptr2= start;
479      dptr = destination;
480
481      if(end <= start) print_error();
482      else
483      {
484
485      for(i=0; i<temp; i++)
486      {
487     *(dptr+i)= *(dptr2+i);
488      }
489
490      display_PC= destination;
491      read_external_mem();
492      dot_data();
493      state=2;
494      }
495
496   }
497
498   GO()
499   {
500     {
501         turn_off_display();
502
503      PC = display_PC;
504
505       #pragma asm
506
507      MOV system_sp,SP
508      MOV SP,user_sp
509
510         PUSH PC?+1  // push low byte
511      PUSH PC?   // then high byte
512
513         PUSH user_r0
514      PUSH user_r1
515         PUSH user_r2
516         PUSH user_r3
517         PUSH user_r4
518         PUSH user_r5
519         PUSH user_r6
520         PUSH user_r7
521
522      PUSH user_B
523      PUSH user_PSW
524      PUSH user_DPL
525      PUSH user_DPH
526      PUSH user_ACC
527
528    // reload user registers to CPU registers
529      POP ACC
530         POP DPH
531            POP DPL
532
```

```
533     POP PSW
534     ANL PSW,#11100111b //ensure to use bank 0 registers
535     POP B
536
537           POP 7
538           POP 6
539           POP 5
540           POP 4
541           POP 3
542           POP 2
543           POP 1
544           POP 0
545
546           RET    // jump to user code
547
548      #pragma endasm
549
550     }
551
552
553
554   }
555
556
557   key_GO()
558   {
559       switch(state)
560    {
561     case 12: copy_data();break;
562     case 4: find_ajmp();break;
563     case 5: find_offset(); break;
564     case 6: find_acall(); break;
565     case 1: GO(); break;
566     case 2: GO(); break;
567
568
569
570    }
571
572    }
573
574   delay(int i)
575   {
576
577      for(j=0; j<i; j++)
578      continue;
579   }
580
581   void beep()
582   {
583
584      if(auto_key==0)
585      {
586      for(k=0; k<40; k++)
587      {
588       port1 = ~speaker&0x3f;
589    delay(57);      // calibrated 525Hz (523Hz is needed)
590    port1 = speaker|0x3f;
591    delay(57);
592
593       }
594
595      }
596   }
597
598
599   // insert 512 down
600
601   insert_byte()
602   {
603    dptr=display_PC;
604       for(j=512; j>0; j--)
605    {
606     *(dptr+j)=*(dptr+j-1);
607    }
608    *(dptr+1)=0;  // insert next byte
```

```
 display_PC++;
    read_external_mem();
 state=2;
}

// lift 512 bytes up
delete_byte()
{
  dptr=display_PC;
    for(j=0; j<512; j++)
 {
  *(dptr+j)=*(dptr+j+1);
 }
    read_external_mem();
 state=2;
}

display_byte(char x)
{

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

display_acc()
{

   display_byte(user_ACC);
   buffer[2]=0;
   buffer[3]=0x3f;;
   buffer[4]=0;
   buffer[5]=0;


}

display_B()
{

   display_byte(user_B);
   buffer[2]=0;
   buffer[3]=0xa7;;
   buffer[4]=0;
   buffer[5]=0;
}

display_SP()
{

   display_byte(user_sp);
   buffer[2]=0;


   r0=user_sp;
   n = *r0;
   buffer[3]= convert[n&0xf];
   n>>=4;
   buffer[4]=convert[n&0xf];

   buffer[5]=0;

}

display_psw()
{
   display_byte(user_PSW);
   buffer[2]=0;
   buffer[3]=0xa9;
   buffer[4]=0xae;
   buffer[5]=0x1f;

}

display_word(unsigned int x)
{
```

```
685    temp = x;
686    buffer[2]= convert[temp&0xf];
687    temp>>=4;
688    buffer[3]= convert[temp&0xf];
689    temp>>=4;
690    buffer[4]=convert[temp&0xf];
691    temp>>=4;
692    buffer[5]=convert[temp&0xf];
693    }
694
695    display_dptr()
696    {
697    temp2=user_DPH;
698    temp2 <<=8;
699    temp2|=user_DPL;
700       display_word(temp2);
701    buffer[1]= 0xb3;
702    buffer[0]= 0x1f;
703   }
704
705   set_ajmp()
706   {
707       state=4; // state=4 is for AJMP opcode computing
708    address_display();
709    dot_address();
710    start=display_PC;
711    buffer[1]=2;
712    buffer[0]=0xb3;
713    hit=0;
714   }
715
716   set_offset()
717   {
718      state=5;   // state=5 for computing offset byte
719      address_display();
720      dot_address();
721      start= display_PC+1;
722      buffer[1]=2;
723      buffer[0]=0xb3; // put destination?
724      hit =0;
725
726   }
727
728   set_acall()
729   {
730    state=6;  // state=6 is for AJMP opcode computing
731    address_display();
732    dot_address();
733    start=display_PC;
734    buffer[1]=2;
735    buffer[0]=0xb3;
736    hit=0;
737
738
739
740   }
741
742   dump_iRAM()
743   {
744      r0=display_RAM;
745      n= *r0;
746      buffer[0]= convert[n&0xf]|0x40;
747      n>>=4;
748      buffer[1]=convert[n&0xf]|0x40;
749
750      buffer[2]=0;
751
752      n=display_RAM;
753
754       buffer[3]= convert[n&0xf];
755      n>>=4;
756      buffer[4]=convert[n&0xf];
757      buffer[5]=0;
758
759
760   }
```

```
761
762  dump_register()
763  {
764     buffer[5]=0;
765     buffer[4]= 3;  // display r
766     buffer[3]=convert[user_register];
767     buffer[2]=0;
768     r0=&user_r0;
769     n=*(r0+user_register);
770
771     buffer[0]= convert[n&0xf];
772     n>>=4;
773     buffer[1]=convert[n&0xf];
774  }
775
776  internal_ram()
777  {
778     state=7; // for internal RAM display
779
780     dump_iRAM();
781
782
783  }
784
785  displayr0_r7()
786  {
787    state=8;  // for r0-r7 display
788    dump_register();
789  }
790
791  beep_control()
792  {
793
794     beep_flag^=1;
795
796
797  }
798
799  hardware_test()
800  {
801    o=0x80;
802   for(i=0; i<8; i++)
803   {
804    P1^=0x80;
805    beep();
806    delay(1000);
807    gpio1= o>>=1;
808   }
809   buffer[0]=0xff;
810   buffer[1]=0xff;
811   buffer[2]=0xff;
812   buffer[3]=0xff;
813   buffer[4]=0xff;
814   buffer[5]=0xff;
815
816   for(j=0; j<200; j++) scan();
817
818     k=0;   // k=1 LCD present, k=0 no LCD
819   for(j=0; j<200; j++)
820   {
821    if((LCD_command_read&BUSY)==0) k=1;
822    else k=0;
823   }
824
825   if(k==1)
826   {
827    InitLcd();
828    Puts("8051 Microcontroller");
829    goto_xy(0,1);
830    Puts("Kit v1.0");
831   }
832
833    // test 10ms tick using timer2
834
835     RCAP2H = 0xdc;
836     RCAP2L = 0x00;  // reload value is 0xDC00 for 10ms tick generator
```

```
837
838     // T2CON=0;
839      TR2=1; // run timer2
840
841      for(;;)
842      {
843      while(TF2==0) scan();
844      {
845    TF2=0;
846    if(++tick>100)
847    {
848    buffer[2]=buffer[3]=buffer[4]=buffer[5]=0;
849
850    tick=0;
851    display_byte(s);
852    gpio1=s++;
853    P1_7 ^=1;
854    }
855
856      }
857
858      }
859
860  }
861
862
863  // ALT key used with hex key 0-f with state = 3
864
865  alt_func()
866  {
867    switch(key){
868
869      case 0x0c: insert_byte();break;
870      case 0x0d: delete_byte(); break;
871      case 0: display_acc(); break;
872      case 1: display_B(); break;
873      case 2: display_psw(); break;
874      case 3: display_SP(); break;
875      case 4: display_dptr(); break;
876      case 5: internal_ram(); break;
877      case 6: displayr0_r7(); break;
878      case 8: hardware_test(); break;
879      case 0xe: set_ajmp(); break;
880      case 0xa: set_offset(); break;
881      case 0xb: beep_control(); break;
882      case 0xf: set_acall(); break;
883
884
885
886    }
887
888  }
889
890  show_alt()
891  {
892     buffer[5]= 0x3f;
893     buffer[4]= 0x85;
894     buffer[3]= 0x87;
895     buffer[2]=0;
896     buffer[1]=0;
897     buffer[0]=0;
898     state=3;
899  }
900
901
902
903  // test at 8000b
904  void timer0int (void) interrupt 1 using 1
905  {
906
907        step =1;
908     #pragma asm
909
910         CLR ET0
911       CLR TF0
912
```

```
913      // CLR P1.7
914
915            // save CPU registers to user registers
916
917        PUSH ACC
918        PUSH DPH
919           PUSH DPL
920
921     PUSH PSW
922     PUSH B
923
924           PUSH 7
925           PUSH 6
926           PUSH 5
927           PUSH 4
928           PUSH 3
929           PUSH 2
930           PUSH 1
931           PUSH 0
932
933     POP user_r0
934        POP user_r1
935           POP user_r2
936           POP user_r3
937           POP user_r4
938           POP user_r5
939           POP user_r6
940           POP user_r7
941
942        POP user_B
943        POP user_PSW
944        POP user_DPL
945        POP user_DPH
946        POP user_ACC
947
948      // use debug led and cy led to show the contents of accumulator and carry flag
949     MOV DPTR,#100H
950     MOVX @DPTR,A
951
952     JB  CY,skip_carry0
953     clr a
954     mov debug_cy,a
955     sjmp skip_carry1
956
957   skip_carry0:
958
959           mov a,#1
960     mov debug_cy,a
961
962   skip_carry1:
963
964
965     POP SAVE_PC
966        POP SAVE_PC+1
967
968     mov display_PC,SAVE_PC
969     mov display_PC+1,SAVE_PC+1
970
971     lcall key_data  // update display
972
973       MOV user_SP,SP  // restore system stack
974       MOV SP,system_SP
975
976
977
978       RETI
979
980      #pragma endasm
981   }
982
983   key_step()
984   {
985
986           step=0;
987
988     #pragma asm
```

```
989
990       CLR TR0
991       MOV A,TMOD
992       ANL A,#11110000b
993       ORL A,#1         // set timer0 16-bit mode
994       MOV TMOD,A
995       MOV TH0,#0FFh
996       MOV TL0,#0FEh
997       CLR TF0
998
999       MOV system_sp,SP
1000      MOV SP,user_sp
1001
1002         MOV PC?,SAVE_PC
1003      MOV PC?+1,SAVE_PC+1
1004
1005      PUSH PC?+1  // push low byte
1006      PUSH PC?    // then high byte
1007
1008    // push user regsiters to STACK
1009       PUSH user_r0
1010     PUSH user_r1
1011        PUSH user_r2
1012        PUSH user_r3
1013        PUSH user_r4
1014        PUSH user_r5
1015        PUSH user_r6
1016        PUSH user_r7
1017
1018     PUSH user_B
1019     PUSH user_PSW
1020     PUSH user_DPL
1021     PUSH user_DPH
1022     PUSH user_ACC
1023
1024    // reload user registers to CPU registers
1025     POP ACC
1026        POP DPH
1027           POP DPL
1028
1029     POP PSW
1030     ANL PSW,#11100111b //ensure to use bank 0 registers
1031     POP B
1032
1033           POP 7
1034           POP 6
1035           POP 5
1036           POP 4
1037           POP 3
1038           POP 2
1039           POP 1
1040           POP 0
1041
1042      ORL IE,#10000010b // enable timer0 interrupt
1043      SETB TR0
1044
1045      RET
1046
1047
1048    #pragma endasm
1049  }
1050
1051
1052
1053  key_copy()
1054  {
1055
1056    state=10;  // copy function
1057    buffer[0]=0xae;
1058
1059     address_display();
1060     dot_address();
1061    // start= display_PC;
1062     buffer[1]=2;
1063     buffer[0]=0xae;
1064
```

```
1065     // buffer[0]=0xb3; // put destination?
1066      hit =0;
1067   }
1068
1069
1070   key_exe()
1071   {
1072
1073       if(beep_flag==0) beep();  // beep when pressed
1074
1075      if(key>15)
1076      {
1077
1078      switch(key) // for function key
1079      {
1080    case 0x17: key_address(); break;
1081    case 0x16: key_data(); break;
1082    case 0x13: key_plus(); break;
1083    case 0x12: key_minus();break;
1084    case 0x10: key_PC(); break;
1085    case 0x11: key_GO(); break;
1086    case 0x18: show_alt(); break;
1087    case 0x14: key_copy(); break;
1088    case 0x15: key_step(); break;
1089
1090    }
1091
1092    }
1093    else
1094    {
1095    switch(state) // for hex key enter
1096     {
1097      case 1: address_hex(); break;
1098      case 2: data_hex(); break;
1099      case 3: alt_func(); break;
1100      case 4: word_enter(); break;
1101      case 5: word_enter(); break;
1102      case 6: word_enter(); break;
1103      case 7: internal_data_hex(); break;
1104      case 10: word_enter(); break;
1105      case 11: word_enter(); break;
1106      case 12: word_enter(); break;
1107
1108
1109     }
1110
1111        }
1112
1113   }
1114
1115   delay_us()
1116   {
1117
1118    #pragma asm
1119
1120       NOP  // restart with E1 written for serial connected
1121          NOP
1122       NOP
1123       NOP
1124       NOP
1125
1126       #pragma endasm
1127
1128
1129
1130   }
1131
1132   char scan()
1133   {
1134       k=1;
1135    u=0;
1136    key=-1; // if no key pressed key=-1
1137    q=0; // key code
1138
1139      for(i=0; i<6; i++)
1140      {
```

```
1141    if(debug_cy==0) port1 = (~k)&0xbf;// write digit and turn carry off
1142      else port1 = (~k)|0x40;  // turn carry bit
1143
1144     port2 = buffer[i]; // write segment
1145     delay_us();
1146     //delay(10);
1147     port2 = 0;
1148       o= port0; // read keypad
1149      for(n=0; n<4; n++)
1150     {
1151       if((o&1)==0) key=q;
1152       else q++;
1153       o>>=1;
1154     }
1155
1156      k<<=1;
1157
1158    }
1159    o = port0;
1160    if((o&0x80)==0) key=0x18; // check key S20
1161
1162    return key;
1163  }
1164
1165
1166
1167  chk_serial()
1168  {
1169
1170     if(RI)
1171     {
1172    RI=0;
1173    if(SBUF == ' ' || SBUF == 0x0d) // hit space bar or CR to connect
1174    {
1175      gpio1=0;
1176      port2=0;
1177      port1=0xbf;
1178        dptr = 0xffff; // store 0xE1 at location FFFF for serial mode
1179      *dptr = 0xe1;
1180
1181      // enable WDT to reset CPU
1182        //  WDTRST = 0x1e;
1183   //WDTRST = 0xe1;
1184
1185   #pragma asm
1186
1187      LJMP 0;   // restart with E1 written for serial connected
1188
1189     #pragma endasm
1190
1191
1192    }
1193
1194     }
1195
1196
1197  }
1198
1199  loop_scan()
1200  {
1201     for(temp=0; temp<150; temp++)
1202     scan();
1203  }
1204
1205
1206  void scan1()
1207  {
1208     temp2=0;
1209
1210     while((scan()!= -1) && (auto_key==0))
1211
1212     {chk_serial();
1213
1214     if((++temp2>1000) && (auto_key==0)) auto_key=1;
1215
1216     }
```

```
1217     delay(100);
1218
1219
1220     if(auto_key) loop_scan();
1221
1222    // delay(100);
1223
1224     while(scan()== -1)
1225    {   chk_serial();
1226      auto_key=0;
1227    }
1228     delay(100);
1229
1230      key = scan();
1231    // check key in range
1232     if(key>=0 && key <25)
1233     { key= keycode[key];
1234       key_exe();
1235     }
1236
1237  }
1238
1239
1240  main()
1241  {
1242    k=1;
1243    n=0;
1244    buffer[5]= convert[8];
1245    buffer[4]= convert[0];
1246    buffer[3]=convert[5];
1247    buffer[2]=convert[1];
1248    buffer[1]=0;
1249    buffer[0]=0;
1250
1251    gpio1=step=0;
1252      dptr = 0xffff; // store 0xE1 at location FFFF for serial mode
1253      *dptr = 0; // clear serial mode
1254
1255      dptr = 0x800b;  // insert LJMP 100B at user RAM 800B
1256      *dptr = 0x02;
1257      *(dptr+1)= 0x10;
1258      *(dptr+2)= 0x0b;
1259
1260    PC = display_PC = SAVE_PC= 0x9000; //current PC is 0x8000
1261
1262   // display_RAM=0x80;
1263   // user_register=0;
1264
1265    state=0;
1266    user_sp = 0x60; // set user stack at 60h
1267    auto_key=0;
1268    user_PSW=0;
1269    debug_cy=0;
1270
1271       r0=warm_code;
1272    n= *r0;
1273    if(n != 0xaa)
1274    {     // cold reset preset values
1275     *r0=0xaa; // save it for warm boot
1276     display_RAM=0x80; // display RAM address not change for warm reset
1277     user_register=0; // user pointer as well
1278     beep_flag=0; // reset beep flag to enable
1279    }
1280
1281       while(1)
1282    {
1283    scan1();
1284     }
1285  }
```

```
$NOMOD51
;------------------------------------------------------------------------------
;  This file is part of the C51 Compiler package
;  Copyright (c) 1988-2005 Keil Elektronik GmbH and Keil Software, Inc.
;  Version 8.01
;
;  *** <<< Use Configuration Wizard in Context Menu >>> ***
;------------------------------------------------------------------------------
;  STARTUP.A51:  This code is executed after processor reset.
;
;  To translate this file use A51 with the following invocation:
;
;     A51 STARTUP.A51
;
;  To link the modified STARTUP.OBJ file to your application use the following
;  Lx51 invocation:
;
;     Lx51 your object file list, STARTUP.OBJ  controls
;
;------------------------------------------------------------------------------
;
;  User-defined <h> Power-On Initialization of Memory
;
;  With the following EQU statements the initialization of memory
;  at processor reset can be defined:
;
; <o> IDATALEN: IDATA memory size <0x0-0x100>
;     <i> Note: The absolute start-address of IDATA memory is always 0
;     <i>       The IDATA space overlaps physically the DATA and BIT areas.
IDATALEN        EQU     80H
;
; <o> XDATASTART: XDATA memory start address <0x0-0xFFFF>
;     <i> The absolute start address of XDATA memory
XDATASTART      EQU     0
;
; <o> XDATALEN: XDATA memory size <0x0-0xFFFF>
;     <i> The length of XDATA memory in bytes.
XDATALEN        EQU     0
;
; <o> PDATASTART: PDATA memory start address <0x0-0xFFFF>
;     <i> The absolute start address of PDATA memory
PDATASTART      EQU     0H
;
; <o> PDATALEN: PDATA memory size <0x0-0xFF>
;     <i> The length of PDATA memory in bytes.
PDATALEN        EQU     0H
;
;</h>
;------------------------------------------------------------------------------
;
;<h> Reentrant Stack Initialization
;
;  The following EQU statements define the stack pointer for reentrant
;  functions and initialized it:
;
; <h> Stack Space for reentrant functions in the SMALL model.
;  <q> IBPSTACK: Enable SMALL model reentrant stack
;     <i> Stack space for reentrant functions in the SMALL model.
IBPSTACK        EQU     0       ; set to 1 if small reentrant is used.
;  <o> IBPSTACKTOP: End address of SMALL model stack <0x0-0xFF>
;     <i> Set the top of the stack to the highest location.
IBPSTACKTOP     EQU     0xFF +1     ; default 0FFH+1
; </h>
;
; <h> Stack Space for reentrant functions in the LARGE model.
;  <q> XBPSTACK: Enable LARGE model reentrant stack
;     <i> Stack space for reentrant functions in the LARGE model.
XBPSTACK        EQU     0       ; set to 1 if large reentrant is used.
;  <o> XBPSTACKTOP: End address of LARGE model stack <0x0-0xFFFF>
;     <i> Set the top of the stack to the highest location.
XBPSTACKTOP     EQU     0xFFFF +1   ; default 0FFFFH+1
; </h>
;
; <h> Stack Space for reentrant functions in the COMPACT model.
;  <q> PBPSTACK: Enable COMPACT model reentrant stack
;     <i> Stack space for reentrant functions in the COMPACT model.
```

```
77  PBPSTACK        EQU     0       ; set to 1 if compact reentrant is used.
78  ;
79  ;   <o> PBPSTACKTOP: End address of COMPACT model stack <0x0-0xFFFF>
80  ;     <i> Set the top of the stack to the highest location.
81  PBPSTACKTOP     EQU     0xFF +1     ; default 0FFH+1
82  ; </h>
83  ;</h>
84  ;------------------------------------------------------------------------------
85  ;
86  ;  Memory Page for Using the Compact Model with 64 KByte xdata RAM
87  ;  <e>Compact Model Page Definition
88  ;
89  ;  <i>Define the XDATA page used for PDATA variables.
90  ;  <i>PPAGE must conform with the PPAGE set in the linker invocation.
91  ;
92  ; Enable pdata memory page initalization
93  PPAGEENABLE     EQU     0       ; set to 1 if pdata object are used.
94  ;
95  ; <o> PPAGE number <0x0-0xFF>
96  ; <i> uppermost 256-byte address of the page used for PDATA variables.
97  PPAGE           EQU     0
98  ;
99  ; <o> SFR address which supplies uppermost address byte <0x0-0xFF>
100 ; <i> most 8051 variants use P2 as uppermost address byte
101 PPAGE_SFR       DATA    0A0H
102 ;
103 ; </e>
104 ;------------------------------------------------------------------------------
105
106 ; Standard SFR Symbols
107 ACC     DATA    0E0H
108 B       DATA    0F0H
109 SP      DATA    81H
110 DPL     DATA    82H
111 DPH     DATA    83H
112
113                 NAME    ?C_STARTUP
114
115
116 ?C_C51STARTUP   SEGMENT   CODE
117 ?STACK          SEGMENT   IDATA
118
119                 RSEG    ?STACK
120                 DS      1
121
122                 EXTRN CODE (?C_START)
123                 PUBLIC  ?C_STARTUP
124
125                 CSEG    AT 1000h     ;8000h
126 ?C_STARTUP:     LJMP    STARTUP1
127
128                 RSEG    ?C_C51STARTUP
129
130 STARTUP1:
131
132 ; do not clear iRAM
133
134 IF IDATALEN <> 0
135                 MOV     R0,#IDATALEN - 1
136                 CLR     A
137 IDATALOOP:      NOP                         ; MOV     @R0,A
138                 DJNZ    R0,IDATALOOP
139 ENDIF
140
141 IF XDATALEN <> 0
142                 MOV     DPTR,#XDATASTART
143                 MOV     R7,#LOW (XDATALEN)
144   IF (LOW (XDATALEN)) <> 0
145                 MOV     R6,#(HIGH (XDATALEN)) +1
146   ELSE
147                 MOV     R6,#HIGH (XDATALEN)
148   ENDIF
149                 CLR     A
150 XDATALOOP:      MOVX    @DPTR,A
151                 INC     DPTR
152                 DJNZ    R7,XDATALOOP
```

```
153                 DJNZ    R6,XDATALOOP
154   ENDIF
155
156   IF PPAGEENABLE <> 0
157                 MOV     PPAGE_SFR,#PPAGE
158   ENDIF
159
160   IF PDATALEN <> 0
161                 MOV     R0,#LOW (PDATASTART)
162                 MOV     R7,#LOW (PDATALEN)
163                 CLR     A
164   PDATALOOP:    MOVX    @R0,A
165                 INC     R0
166                 DJNZ    R7,PDATALOOP
167   ENDIF
168
169   IF IBPSTACK <> 0
170   EXTRN DATA (?C_IBP)
171
172                 MOV     ?C_IBP,#LOW IBPSTACKTOP
173   ENDIF
174
175   IF XBPSTACK <> 0
176   EXTRN DATA (?C_XBP)
177
178                 MOV     ?C_XBP,#HIGH XBPSTACKTOP
179                 MOV     ?C_XBP+1,#LOW XBPSTACKTOP
180   ENDIF
181
182   IF PBPSTACK <> 0
183   EXTRN DATA (?C_PBP)
184                 MOV     ?C_PBP,#LOW PBPSTACKTOP
185   ENDIF
186
187                 MOV     SP,#?STACK-1
188
189   ; This code is required if you use L51_BANK.A51 with Banking Mode 4
190   ;<h> Code Banking
191   ; <q> Select Bank 0 for L51_BANK.A51 Mode 4
192   #if 0
193   ;     <i> Initialize bank mechanism to code bank 0 when using L51_BANK.A51 with Banking Mc
194   EXTRN CODE (?B_SWITCH0)
195                 CALL    ?B_SWITCH0      ; init bank mechanism to code bank 0
196   #endif
197   ;</h>
198                 LJMP    ?C_START
199
200                 END
201
```