```
1  // Monitor source code for
2  // Z180 Microprocessor Kit sdcc version
3  // Written by Wichit Sirichote (C) 2022
4
5  // Start up file was modified for Z180 kit hardware
6  // z180crt0.s
7
8  // using sdcc v3.3.0 #8604 May 11, 2013
9  // compile command
10 // sdcc -mz180 --code-loc 0x100 --data-loc 0xF000 --no-std-crt0 z180crt0.rel z180.c
11
12 // Hardware z180 PLCC, 32kB ROM 128kB RAM, sound chip SAA1099
13
14 // Memory layout
15
16 // 00000  ROM     first 64kB in COMMON 0 area
17 // 07FFF  32k ROM
18 //........................................
19 // 08000  32k RAM
20 //
21 // 0F000  system ram
22 //
23 // 0FFFF  SP
24 //
25 //-------------------------------------
26 // 10000  extended RAM in BANKED area
27 // 1FFFF
28 //-------------------------------------
29 // 20000  extended RAM in COMMON 1 area
30 // 27FFF
31 //-------------------------------------
32
33 //
34 // January 6, 2022  monitor program rev.1.0 tested with 32kB EPROM, 128kB RAM
35 //
36
37
38
39
40
41 #include <z180.h>
42
43 #pragma disable_warning 154
44
45 __sfr __at 0x40 gpio1;
46
47 __sfr __at 0x41 port0;  // key input
```

```
48  __sfr __at 0x42 port1; // digit
49  __sfr __at 0x43 port2; // segment
50
51
52  __sfr __at 0x60 sound_data; // sound chip data register
53  __sfr __at 0x61 sound_addr; // sound chip address register
54
55  #define speaker_on CNTLA0 |= 0x10;  // set RTS0
56  #define speaker_off CNTLA0 &= ~0x10;  // clear RTS0
57
58
59  #define BUSY 0x80
60
61  __sfr __at 0x80 LCD_command_write;
62  __sfr __at 0x81 LCD_data_write;
63  __sfr __at 0x82 LCD_command_read;
64  __sfr __at 0x83 LCD_data_read;
65
66  //prototype declaration
67
68  void test_serial();
69  void putch(unsigned char c);
70  void puts(char *s);
71  void beep();
72  void beep1();
73  void beep2();
74  void sound_off();
75  void sound_write(char a, char d);
76  void sound_test();
77  void delay_100ms(unsigned int j);
78  void noise_test();
79  void musical_note_test();
80
81
82
83
84  unsigned int i;
85  unsigned int j;
86
87  unsigned char n,d;
88  char k;
89  unsigned char u,q;
90  char o,key;
91  char x,z;
92  char hit, positive;
93  char flag;
94  char tick;
```

```
 95
 96  unsigned char bcc, save_bcc, bcc_error;
 97
 98  char save_acc;
 99  char type;
100  char test;
101
102  char count;
103
104
105  unsigned int temp,temp16;
106
107
108  int num, start,end, desti;
109
110  int t;
111
112  char state;
113
114  int PC, save_PC;
115  int USER_SP,SYS_SP;
116
117  int USER_AF,USER_BC,USER_DE,USER_HL;
118  int USER_AF_,USER_BC_,USER_DE_,USER_HL_;
119  int USER_IX, USER_IY;
120
121  int timeout;
122  char buffer[8];
123
124  char *dptr;
125  char *dptr2;
126
127
128  // table for converting 0-9 to 7-segment LED
129  const char convert[16]= {0xBD,0x30,0x9B,0xBA,0x36,0xAE,0xAF,0x38,0xBF,0xBE,0x3F,0xA7,0x8D,0xB3,0x8F,0x0F};
130
131  const char cold_msg[12]={0,0,0,0,0,0,0xa1,0x1f,0x0f,0x02,0x02,0x30};
132
133  const unsigned char note[12] = {5,33,60,85,109,132,153,173,192,210,227,243};
134
135  int timeout;
136
137  void LcdReady()
138  {
139   timeout=0;
140  while((LCD_command_read&BUSY) && (timeout<100))
141   timeout++; // wait until busy flag =0
```

```
}
void clr_screen(void)
{
LcdReady();
LCD_command_write=0x01;
}

void goto_xy(char x,char y)
{
LcdReady();
switch(y){
case 0 : LCD_command_write=0x80+x; break;
case 1 : LCD_command_write=0xC0+x; break;
case 2 : LCD_command_write=0x94+x; break;
case 3 : LCD_command_write=0xd4+x; break;
}
}
void InitLcd(void)
{
LcdReady();
LCD_command_write=0x38;
LcdReady();
LCD_command_write=0x0c;
clr_screen();
goto_xy(0,0);
}

void putch_lcd(char ch)
{
LcdReady();
LCD_data_write=ch;
}

void LCDWriteText(char *txt) {
  while(*txt)
    putch_lcd(*txt++);
}

void print_hex(char n)
{
     unsigned char temp;
   temp= n;
   temp>>=4;
   temp += 0x30;
   if (temp>=0x3A) putch_lcd (temp+7);
    else putch_lcd (temp);
   temp = n&0xf;
```

```
   temp += 0x30;
      if (temp>=0x3A) putch_lcd (temp+7);
     else putch_lcd (temp);
}

void printInt(unsigned int k)
{
    int temp;
 putch_lcd((k/10000)+0x30);
    temp = k%10000;
 putch_lcd((temp/1000)+0x30);
    k = temp;
 temp = k%1000;
 putch_lcd((temp/100)+0x30);
 k = temp;
 temp = k%100;
 putch_lcd((temp/10)+0x30);
 k = temp;
 temp = k%10;
 putch_lcd(temp+0x30);

}

//entry is HL for Y and X

void gotoxy()
{
   __asm

       PUSH HL
    CALL _goto_xy
    POP HL

   __endasm;

}


// entry is HL pointer
void printtext()
{
 __asm

       PUSH HL
    CALL _LCDWriteText
    POP HL

```

```
236     __endasm;
237
238  }
239
240  // entry is A
241  void printchar()
242  {
243    __asm
244
245      PUSH HL
246      CALL _putch_lcd
247      POP HL
248
249      __endasm;
250  }
251
252  // entry is A
253  void printhex()
254  {
255        __asm
256      LD L,A
257      PUSH HL
258      CALL _print_hex
259      POP HL
260           __endasm;
261
262  }
263
264  // entry with HL
265  void printInteger()
266  {
267      __asm
268
269        PUSH HL
270        CALL _printInt
271      POP HL
272
273     __endasm;
274
275  }
276
277
278  void delay()
279  {
280   int i;
281
282   for(i=0; i<10000; i++)
```

```
   ;
}


void dot_address()
{
 buffer[0]=buffer[0]&~0x40;
 buffer[1]=buffer[1]&~0x40;

 buffer[2]=0;  // left blank

  buffer[3]=buffer[3]|0x40;
   buffer[4]=buffer[4]|0x40;
    buffer[5]=buffer[5]|0x40;
    buffer[6]=buffer[6]|0x40;


    buffer[7]= 0xbd;  // for upper four bits
}


void dot_data()
{

 buffer[0]=buffer[0]|0x40;
 buffer[1]=buffer[1]|0x40;

 buffer[2]=0;  // left blank

 buffer[3]=buffer[3]&~0x40;
   buffer[4]=buffer[4]&~0x40;
    buffer[5]=buffer[5]&~0x40;
    buffer[6]=buffer[6]&~0x40;

    buffer[7]= 0xbd;  // for upper four bits

}


void hex4(int h)
{
    temp16 = h;
 buffer[3]= convert[temp16&0xf];
 temp16>>=4;
 buffer[4]= convert[temp16&0xf];
 temp16>>=4;
 buffer[5]=convert[temp16&0xf];
```

```
 temp16>>=4;
 buffer[6]=convert[temp16&0xf];
}

void hex2(char h)
{
    temp = h;
 buffer[0]= convert[temp&0xf];
 temp>>=4;
 buffer[1]= convert[temp&0xf];
}

void address_display()
{

 temp = PC;
 hex4(temp);
}

void data_display()
{
    dptr = PC;

 d = *dptr;

 buffer[0]= convert[d&0xf];
 d >>=4;
 buffer[1]=convert[d&0xf];
 dot_data();

}

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 print_error()
{

  buffer[5]= 0x8f;
  buffer[4]= 3;
  buffer[3]=3;
  buffer[2]=0;
  buffer[1]=0;
  buffer[0]=0;
  state=0;
}


void key_plus()
{

     if(state==1 || state==2)
     {
       PC++;
       read_memory();
    key_data();
   }
   if(state==4)
 {
    start=num;
       hit =0;
    positive=1;


 }

 if(state==5)
 {
```

```
424
425       state=6;
426     start = num;
427     hit=0;
428     buffer[0]=0x8f;  /* end cursor */
429       return;
430
431    }
432
433
434       if(state==6)
435    {
436
437       state=7;
438     end = num;
439     hit=0;
440     buffer[0]=0xb3;  /* destination cursor */
441
442     if(end <= start) print_error();
443
444    }
445
446
447    if(state==8)
448    {
449     state=9;
450     start=num;
451     hit=0;
452     buffer[0]=0xb3;  // destination
453
454       }
455
456  }
457
458  void key_minus()
459  {
460     if(state==1 | state ==2)
461     {
462      PC--;
463      read_memory();
464    key_data();
465     }
466
467     if(state==4)
468    {
469       start=num;
470          hit =0;
```

```
        positive=0;
  }
 }

 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();
  }
 }

 void data_hex()
 {

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

   *dptr = x;

   read_memory();

      dot_data();
  }
 }


 void key_reg()
 {
    buffer[7]= 0;
```

```
518     buffer[6]= 0;
519
520
521     buffer[5]= 0x03;
522     buffer[4]= 0x8F;
523     buffer[3]= 0xad;
524     buffer[2]=0;
525     buffer[1]=0;
526     buffer[0]=0;
527
528     state = 3;    /* register display state = 3 with hex key */
529
530  }
531
532
533  void reg_AF()
534  {
535
536      temp16 = USER_AF;
537
538   hex4(temp16);
539
540
541   buffer[1] = 0x3f;
542   buffer[0] = 0x0f;
543  }
544
545  void reg_BC()
546  {
547
548     temp16 = USER_BC;
549
550   hex4(temp16);
551
552
553   buffer[1] = 0xa7;
554   buffer[0] = 0x8d;
555  }
556
557  void reg_DE()
558  {
559
560     temp16 = USER_DE;
561
562   hex4(temp16);
563
564
```

```
565    buffer[1] = 0xb3;
566    buffer[0] = 0x8f;
567   }
568   void reg_HL()
569   {
570
571      temp16 = USER_HL;
572
573    hex4(temp16);
574
575
576    buffer[1] = 0x37;
577    buffer[0] = 0x85;
578   }
579
580   void reg_AF_()
581   {
582
583       temp16 = USER_AF_;
584
585    hex4(temp16);
586
587
588    buffer[1] = 0x3f;
589    buffer[0] = 0x0f|0x40;
590   }
591
592   void reg_BC_()
593   {
594
595      temp16 = USER_BC_;
596
597    hex4(temp16);
598
599
600    buffer[1] = 0xa7;
601    buffer[0] = 0x8d|0x40;
602   }
603
604   void reg_DE_()
605   {
606
607      temp16 = USER_DE_;
608
609    hex4(temp16);
610
611
```

```
612    buffer[1] = 0xb3;
613    buffer[0] = 0x8f|0x40;
614   }
615   void reg_HL_()
616   {
617
618      temp16 = USER_HL_;
619
620    hex4(temp16);
621
622
623    buffer[1] = 0x37;
624    buffer[0] = 0x85|0x40;
625   }
626
627   void reg_IX()
628   {
629
630      temp16 = USER_IX;
631
632    hex4(temp16);
633
634
635    buffer[1] = 0x30;
636    buffer[0] = 0x07;
637   }
638
639   void reg_IY()
640   {
641
642      temp16 = USER_IY;
643
644    hex4(temp16);
645
646
647    buffer[1] = 0x30;
648    buffer[0] = 0xb6;
649   }
650   void reg_SP()
651   {
652
653      temp16 = USER_SP;
654
655    hex4(temp16);
656
657
658    buffer[1] = 0xae;
```

```
659    buffer[0] = 0x1f;
660  }
661
662  void flag_low()
663  {
664
665    flag = (char)USER_AF;
666    temp16 = 0;
667
668    if(flag&1) temp16 |=1;
669    if(flag&2) temp16 |=0x10;
670    if(flag&4) temp16 |=0x100;
671
672    hex4(temp16);
673    buffer[0] = 0x85;
674    buffer[1]= 0x0f;
675  }
676
677  void flag_high()
678  {
679
680    flag = (char)USER_AF;
681    temp16 = 0;
682
683    if(flag&0x10) temp16 |=1;
684    if(flag&0x40) temp16 |=0x100;
685    if(flag&0x80) temp16 |=0x1000;
686
687    hex4(temp16);
688    buffer[0] = 0x37;
689    buffer[1]= 0x0f;
690  }
691
692
693  void flag_low_()
694  {
695
696    flag = (char)USER_AF_;
697    temp16 = 0;
698
699    if(flag&1) temp16 |=1;
700    if(flag&2) temp16 |=0x10;
701    if(flag&4) temp16 |=0x100;
702
703    hex4(temp16);
704    buffer[0] = 0x85|0x40;
705    buffer[1]= 0x0f;
```

```
706  }
707
708  void flag_high_()
709  {
710
711    flag = (char)USER_AF_;
712    temp16 = 0;
713
714    if(flag&0x10) temp16 |=1;
715    if(flag&0x40) temp16 |=0x100;
716    if(flag&0x80) temp16 |=0x1000;
717
718    hex4(temp16);
719    buffer[0] = 0x37|0x40;
720    buffer[1]= 0x0f;
721  }
722
723
724
725  void reg_display()
726  {
727
728    switch(key)
729   {
730     case 0: reg_AF(); break;
731     case 1: reg_BC(); break;
732     case 2: reg_DE(); break;
733     case 3: reg_HL(); break;
734     case 4: reg_AF_(); break;
735     case 5: reg_BC_(); break;
736     case 6: reg_DE_(); break;
737     case 7: reg_HL_(); break;
738     case 8: reg_IX(); break;
739     case 9: reg_IY(); break;
740     case 0x0a: reg_SP(); break;
741     case 0x0d: flag_low(); break;
742     case 0x0c: flag_high(); break;
743     case 0x0e: flag_high_(); break;
744     case 0x0f: flag_low_(); break;
745      }
746  }
747
748  /* insert byte and shift 512 bytes down */
749
750  void insert()
751  {
752   if(state==1 || state==2)
```

```
753   {
754   dptr=PC;
755      for(j=512; j>0; j--)
756   {
757    *(dptr+j)=*(dptr+j-1);
758   }
759   *(dptr+1)=0;  /* insert next byte */
760   PC++;
761      read_memory();
762   state=2;
763   }
764  }
765
766
767  /* delete current byte and shift 512 bytes up */
768
769  void cut_byte()
770  {
771     if(state==1 || state==2)
772    {
773
774     dptr=PC;
775      for(j=0; j<512; j++)
776    {
777     *(dptr+j)=*(dptr+j+1);
778    }
779      read_memory();
780    state=2;
781    }
782  }
783
784
785  // similar to key go, but turn on break signal, single instruction was fetched
786  // then generated nmi interrupt
787
788
789  void key_step()
790  {
791     __asm
792
793     ld (_SYS_SP),sp   ; save system stack
794     ld sp,(_USER_SP)  ; reload with user stack
795
796    ; load CPU registers with user registers
797
798
799     ld hl,(_USER_IY)
```

```
800           push  hl
801           ld hl,(_USER_IX)
802       push hl
803       ld hl,(_USER_DE)
804           push hl
805       ld hl,(_USER_BC)
806           push hl
807           ld hl,(_USER_AF)
808           push hl
809
810       pop af
811          pop bc
812       pop de
813       pop ix
814       pop iy
815
816        ld hl,(_PC)
817        push hl
818
819        ld (_save_acc),a
820        ld a,#0xbf
821        out (_port1),a
822        ld a,(_save_acc)  ; 1st M1
823        ld hl,(_USER_HL)  ; 2nd M1
824        nop               ; 3rd M1
825        ret               ; 4th M1
826
827        __endasm;
828
829   }
830
831
832   void test_led()
833   {
834
835    while(1)
836    {
837     gpio1 = x++;
838     delay();
839    }
840   }
841
842   void test_speaker()
843   {
844
845     ;
846
```

```
847  }
848
849  // insert high byte vector f0 to I register
850  // low byte will be FF (pull-up data bus)
851  // store service address to f0ff
852  // enable interrupt mode 2
853
854  void test_10ms()
855  {
856     z=0;
857   __asm
858
859     ld a,#0xf0
860           ld i,a
861     ld hl,#_service_10ms
862     ld (0xf0ff),hl
863        im 2
864           ei
865   here:  jr here
866
867      __endasm;
868
869  }
870
871  void service_10ms()
872  {
873      if(++tick>10)
874   {
875     tick=0;
876
877   gpio1 = z++;
878
879   }
880   __asm
881     ei
882     __endasm;
883
884
885  }
886
887
888  void key_go(){
889
890   if(state==1 || state==2)
891   {
892
893     speaker_on
```

```
894
895       __asm
896
897      ld (_SYS_SP),sp   ; save system stack
898      ld sp,(_USER_SP)  ; reload with user stack
899
900     ; load CPU registers with user registers
901
902
903      ld hl,(_USER_IY)
904           push  hl
905           ld hl,(_USER_IX)
906      push hl
907      ld hl,(_USER_DE)
908           push hl
909      ld hl,(_USER_BC)
910           push hl
911           ld hl,(_USER_AF)
912           push hl
913
914      pop af
915         pop bc
916      pop de
917      pop ix
918      pop iy
919
920      ld hl,(_PC)
921      push hl
922
923      ld hl,(_USER_HL)
924
925      __endasm;
926    }
927
928
929      if(state==4)
930    {
931
932       desti = num;
933
934      if(positive==0) start= start-desti;
935         else start = start+desti;
936
937       hex4(start);
938       hit=0;
939
940    }
```

```
941
942   if(state==7)
943   {
944    desti = num;
945    temp = end-start;
946    dptr = start;
947    dptr2 = desti;
948
949    for(i=0; i<temp; i++)
950    {
951     *(dptr2+i)=*(dptr+i);
952    }
953    PC = desti;
954          read_memory();
955          dot_data();
956    state=2;
957
958
959
960   }
961
962   if(state==9)
963   {
964    desti = num;
965    temp =desti-(start+2);
966
967    dptr = start;
968    *(dptr+1)= (char)temp;
969
970    PC = start+1;
971    read_memory();
972    dot_data();
973    state=2;
974   }
975
976   if(state==10)
977   {
978      switch(test)
979    {
980          case 0: test_led(); break;
981    case 1: test_10ms(); break;
982    //case 2: test_serial(); break;
983
984
985    }
986
987   }
```

```
 988
 989  }
 990
 991  void dot_4address()
 992  {
 993
 994    buffer[3]|= 0x40;
 995      buffer[4]|= 0x40;
 996       buffer[5] |= 0x40;
 997           buffer[6] |= 0x40;
 998  }
 999
1000
1001  void enter_num()
1002  {
1003     if(hit==0) num=0;
1004   {
1005      hit=1;
1006
1007      num<<=4;
1008      num |= key;
1009      hex4(num);
1010      dot_4address();
1011
1012
1013
1014   }
1015  }
1016
1017
1018
1019  void clear_buffer()
1020  {
1021   for(i=0; i<6; i++)
1022    *(buffer+i)=0;
1023  }
1024
1025
1026
1027
1028  void key_copy()
1029  {
1030
1031   if(state==1 || state==2)
1032   {
1033    state=5;
1034    hit=0;
```

```
1035
1036    num = PC;
1037
1038    buffer[0]=0xae;
1039    buffer[1]=0;
1040           dot_4address();
1041
1042     }
1043  }
1044
1045  void key_rel()
1046  {
1047
1048   if(state==1 || state==2)
1049   {
1050    state=8;
1051    hit=0;
1052
1053    num=PC; // in case no enter
1054
1055
1056    buffer[0]=0xae;
1057
1058    buffer[1]=0;
1059
1060     dot_4address();
1061  }
1062
1063  }
1064
1065
1066  char getchar()
1067  {
1068   while ((STAT0&0x80)==0)
1069       ;
1070      return RDR0;
1071  }
1072
1073  // return if no character
1074
1075  char getchars()
1076  {
1077
1078   if ((STAT0&0x80)==0) return 0;
1079   else return RDR0;
1080  }
1081
```

```
1082  void test_uart()
1083  {
1084   while(1)
1085     putch(getchar());
1086  }
1087
1088  unsigned char nibble2hex(unsigned char c)
1089  {
1090    if(c<0x40) return (c-0x30);
1091    else return (c-0x37);
1092  }
1093
1094  unsigned char gethex()
1095  {
1096    unsigned char a,b;
1097
1098    a = getchar();
1099    b = getchar();
1100
1101    a = nibble2hex(a)<<4;
1102    b = nibble2hex(b);
1103    a = a|b;
1104    bcc = bcc+a;  /* compute check sum */
1105
1106    return a;
1107  }
1108
1109  int get16bitaddress()
1110  {
1111    unsigned int load_address;
1112
1113    load_address =0;
1114
1115    load_address |= gethex();
1116    load_address <<=8;
1117    load_address |= gethex();
1118
1119    return load_address;
1120  }
1121
1122
1123
1124  void get_record()
1125  {
1126       type =0;
1127
1128       bcc_error=0;
```

```
1129
1130    while(type==0)
1131   {
1132    bcc =0;
1133
1134    while(getchar()!= ':')
1135     ;
1136      count=gethex();
1137
1138
1139    dptr=get16bitaddress();
1140
1141    type = gethex();
1142
1143
1144    for(j=0; j<count; j++)
1145   {
1146    *(dptr+j)=gethex();
1147   }
1148
1149       save_bcc = ~bcc+1;
1150
1151    d = gethex();  // get byte check sum
1152
1153    gpio1 = d;
1154
1155
1156    if(save_bcc != d) bcc_error=1;
1157
1158     }
1159
1160      gpio1 = 0;  // turm indicator off
1161
1162    if (bcc_error==0) puts(" Complete...");
1163   else puts(" Checksum error!...");
1164
1165
1166  }
1167
1168  void read_hex_file()
1169  {
1170
1171    get_record();
1172
1173  }
1174
1175
```

```
1176
1177  void putch(unsigned char c)
1178  {
1179
1180     while ((STAT0&2)==0)
1181       ;
1182     TDR0 = c;
1183
1184  }
1185
1186
1187  void puts(char *s)
1188  {
1189          while( *s )
1190    {
1191                  putch(*s);
1192                  s++;
1193          }
1194  }
1195
1196
1197  void newline()
1198  {
1199   putch(0x0a);
1200   putch(0x0d);
1201  }
1202
1203  void send_hex(char n)
1204  {
1205     k = n>>4;
1206     k = k&0xf;
1207
1208     if (k>9) putch(k+0x37); else putch(k+0x30);
1209     k= n&0xf;
1210     if (k>9) putch(k+0x37); else putch(k+0x30);
1211  }
1212
1213  void send_word_hex(int n)
1214  {
1215     temp16 = n>>8;
1216     k = temp16&0xff;
1217     send_hex(k);
1218     k = n&0xff;
1219     send_hex(k);
1220  }
1221
1222
```

```
1223
1224  void key_dump()
1225  {
1226     int j,p;
1227
1228     newline();
1229
1230     dptr = PC;
1231
1232
1233     for(j=0; j<16; j++)
1234     {
1235       newline();
1236
1237       send_word_hex((int)dptr);
1238
1239    putch(':');
1240    putch(0x20);
1241
1242          for(p=0; p<16; p++)
1243          {
1244
1245           send_hex(*(dptr+p));
1246           putch(0x20);
1247           }
1248
1249          putch(0x20);
1250
1251          for (p=0; p<16; p++)
1252          {
1253            q=*(dptr+p);
1254
1255           if((q >= 0x20) && (q < 0x80)) putch(q);
1256            else putch('.');
1257
1258          }
1259
1260
1261           dptr+=16;
1262     }
1263     //newline();
1264     PC = (int)dptr;
1265     key_address();
1266  }
1267
1268
1269  void key_load()
```

```
{
 puts("\r\n\nDownload Intel hex file...");
 read_hex_file();
 key_data();
}



void key_test()
{

  if(++test>9) test=0;

    buffer[2]=0x87;
 buffer[3]=0xae;
 buffer[4]=0x8f;
 buffer[5]=0x87;

 hex2(test);
    state = 10;

}

void key_halt()
{

      speaker_on
 //sound_test();
 //delay_100ms(30);

 musical_note_test();

 speaker_off
 delay_100ms(30);
 speaker_on

 noise_test();


      speaker_off


  __asm

      HALT
  __endasm;

```

```
}


void key_exe()
{



 if( key>15)
  {


 switch(key)
   {
  case 0x17: key_address(); break;
  case 0x16: key_data(); break;
  case 0x21: key_plus(); break;
  case 0x20: key_minus(); break;
  case 0x14: key_PC(); break;
  case 0x24: key_go(); break;

 // case 0x23: key_step(); break;

  case 0x15: key_reg(); break;
  case 0x18: insert(); break;
  case 0x19: cut_byte(); break;
 // case 0x18: flag = flag^1; break;
  case 0x22: test_10ms(); break;
  case 0x11: key_rel(); break;
  case 0x10: key_copy(); break;
  case 0x12: key_dump(); break;
  case 0x13: key_load(); break;

  case 0x23: key_halt(); break;

   }
   }

 else
  {

   switch(state)
    {
   case 1: hex_address(); break;
   case 2: data_hex(); break;
   case 3: reg_display(); break;
   case 5: enter_num(); break;
```

```
1364     case 6: enter_num(); break;
1365     case 7: enter_num(); break;
1366     case 8: enter_num(); break;
1367     case 9: enter_num(); break;
1368      }
1369
1370    }
1371
1372  }
1373
1374
1375
1376  char key_code(char n)
1377  {
1378    switch(n)
1379    {
1380      case 0x22: key= 0; break;
1381         case 0x1b: key= 1; break;
1382      case 0x15: key= 2; break;
1383      case 0x0f: key= 3; break;
1384      case 0x1c: key= 4; break;
1385      case 0x1a: key= 5; break;
1386      case 0x14: key= 6; break;
1387      case 0x0e: key= 7; break;
1388      case 0x16: key= 8; break;
1389      case 0x19: key= 9; break;
1390      case 0x13: key= 0x0a; break;
1391      case 0x0d: key= 0x0b; break;
1392      case 0x10: key= 0x0c; break;
1393      case 0x1e: key= 0x0d; break;
1394      case 0x18: key= 0x0e; break;
1395      case 0x12: key= 0x0f; break;
1396
1397      case 0x17: key= 0x10; break;
1398         case 0x1d: key= 0x11; break;
1399      case 0x23: key= 0x12; break;
1400      case 0x21: key= 0x13; break;
1401      case 0x0c: key= 0x14; break;
1402      case 0x07: key= 0x15; break;
1403      case 0x08: key= 0x16; break;
1404      case 0x09: key= 0x17; break;
1405      case 0x06: key= 0x18; break;
1406      case 0x01: key= 0x19; break;
1407      case 0x02: key= 0x20; break;
1408      case 0x03: key= 0x21; break;
1409      case 0x20: key= 0x22; break;
1410      case 0x1f: key= 0x23; break;
```

```
1411     case 0x00: key= 0x24; break;
1412
1413
1414
1415    }
1416     return key;
1417
1418   }
1419
1420
1421   void delay_num1()
1422   {
1423     temp=0;
1424     temp=0;
1425   }
1426
1427
1428   void delay_ms(int j)
1429   {
1430    int i;
1431    for(i=0; i<j; i++)
1432       ;
1433
1434   }
1435
1436
1437   char scan()
1438   {
1439    k = 1;
1440    u = 0;
1441    key = -1;
1442    q = 0;
1443
1444
1445      for(i=0; i<8; i++)
1446      {
1447        port1= ~k;
1448        port2=buffer[i];
1449     x = buffer[i];
1450
1451    if((x != 0x30) && (x != 0x38) && (x != 0x70) && (x != 2)) delay_ms(15);
1452     else delay_ms(5);
1453
1454     port2=0;
1455
1456      delay_ms(5);
1457
```

```
    o= port0;

   for(n=0; n<6; n++)
   {
     if((o&1)==0)
    {key=q;
        }

     else q++;
     o >>= 1;
   }

    k <<= 1;
  }

     return key;

}



void scan1()
{
    while( scan() != -1)
      continue;    // if key pressed, keep read it
   delay_ms(10);

   while(scan() == -1)
      continue;   // if no key pressed, keep read it
   delay_ms(10);

    key = scan();  // get key

   // gpio1 = key;  // check scan code


 key = key_code(key);

 //gpio1 = key;  // return internal key code

    // beep();   // beep when press

     key_exe();


}
```

```
1505
1506  //save CPU registers to user registers
1507
1508  void service_nmi()
1509  {
1510   __asm
1511
1512       ld (_save_acc),a
1513
1514    ld a,(_save_acc)
1515
1516
1517          ld (_temp16),hl   ; save hl
1518    pop hl
1519          ld (_save_PC),hl  ; save current PC
1520    ld (_USER_SP),sp  ; save stack pointer
1521    ld hl,(_temp16)   ; restore hl
1522
1523       push af
1524          push bc
1525          push de
1526    push hl
1527          push ix
1528          push iy
1529
1530          pop  hl
1531          ld (_USER_IY),hl
1532          pop  hl
1533          ld (_USER_IX),hl
1534    pop hl
1535    ld (_USER_HL),hl
1536          pop hl
1537          ld (_USER_DE),hl
1538          pop hl
1539    ld (_USER_BC),hl
1540          pop hl
1541          ld (_USER_AF),hl
1542
1543        ; now save prime registers
1544
1545    ex  af,af'
1546       exx
1547
1548    push af
1549          push bc
1550          push de
1551    push hl
```

```
1552
1553    pop hl
1554    ld (_USER_HL_),hl
1555         pop hl
1556         ld (_USER_DE_),hl
1557         pop hl
1558    ld (_USER_BC_),hl
1559         pop hl
1560         ld (_USER_AF_),hl
1561
1562
1563    ld sp,(_SYS_SP)   ; restore system stack
1564
1565    __endasm ;
1566
1567
1568    speaker_off
1569
1570    key_PC();  // update display
1571
1572  }
1573
1574
1575
1576  // init UART channel0, 9600 8n1 XTAL 12.288MHz
1577
1578  void uart_init()
1579  {
1580   CNTLA0 = 0x64;  // 8 data bit no parity one stop bit
1581   CNTLB0 = 0x82; // divide ratio 4, prescale = 10, sampling 16, get 9600
1582
1583  }
1584
1585  void sound_write(char a, char d)
1586  {
1587     sound_addr = a;
1588     sound_data = d;
1589  }
1590
1591  void sound_off()
1592  {
1593      sound_write(0x14,0); // disable frequency channel 0
1594  }
1595
1596  void sound_on()
1597  {
1598
```

```
1599      sound_write(0x14,1); // enable frequency channel 0
1600
1601  }
1602
1603
1604
1605
1606  void beep()
1607  {
1608    sound_write(0x14,1); // enable frequency channel 0
1609   // sound_write(0x00,0x22); // set amplitude 20%
1610      speaker_on       // turn on 100% volume control)
1611   delay_ms(1500);
1612    sound_write(0x14,0); // disable frequency channel 0
1613      speaker_off      // turn off volume control)
1614
1615  }
1616
1617  void beep1()
1618  {
1619   sound_write(0x10,6); // octave channel 0
1620    sound_write(0x14,1); // enable frequency channel 0
1621   delay_ms(2000);
1622    sound_write(0x14,0); // disable frequency channel 0
1623
1624  }
1625
1626  void beep2()
1627  {
1628   sound_write(0x10,5); // octave channel 0
1629    sound_write(0x14,1); // enable frequency channel 0
1630   delay_ms(2000);
1631    sound_write(0x14,0); // disable frequency channel 0
1632
1633  }
1634
1635
1636  void sound_init()
1637  {
1638     sound_write(0x10,5); // octave channel 0
1639     sound_write(0x00,0x88); //  amplitude channel 0 Right max/2 LEFT max/2
1640        sound_write(0x08,0xff); // max frequency channel 0
1641     sound_write(0x15,0); // noise turn off
1642        sound_write(0x18,0); // envelope 0 control
1643     sound_write(0x14,0); // disable frequency channel 0
1644     sound_write(0x1C,1); // enable sound ALL channels
1645
```

```
1646  }
1647
1648  unsigned char octave,freq;
1649
1650
1651  void sound_test()
1652  {
1653     sound_write(0x14,1); // enable frequency channel 0
1654
1655
1656   for (octave=1; octave<7; octave++)
1657   {
1658       sound_write(0x10,octave);  // vary octave 0-7
1659    gpio1 = octave;
1660
1661     for (freq=0; freq<0xff; freq++)
1662     {
1663             sound_write(0x08,freq); // vary freqeuncy 0 to max
1664          delay_ms(300);
1665     }
1666
1667      }
1668       sound_write(0x14,0); // disable frequency channel 0
1669
1670
1671      gpio1 =0;
1672
1673  }
1674
1675  // init timer0 for 10Hz triggering
1676  void timer0_init()
1677  {
1678       RLDR0L = 0x00;
1679       RLDR0H = 0x78;
1680    TCR = 1; // run timer0
1681  }
1682
1683  void delay_100ms(unsigned int j) // x100ms
1684  {
1685       timer0_init();
1686
1687    for(i=0; i<j; i++)
1688   {
1689     while((TCR&0x40)==0)
1690       continue;
1691     n = TCR;      // clear timer flag 0
1692     n = TMDR0L;
```

```
 }
}

void noise_test()
{

  sound_write(0x15,1);

     sound_write(0x16,3);

  delay_100ms(50);  // 5s noise sound

  sound_write(0x15,0);

}


void musical_note_test()
{
    sound_write(0x14,1); // enable frequency channel 0

  for(octave=3; octave<6; octave++)
 {
 sound_write(0x10,octave);  // octave 0-7
    gpio1 = octave;

 for (k=0; k<12 ; k++ )
 {
  sound_write(0x08,note[k]);
  delay_100ms(1);
 }

 }
     gpio1 = 0;
   sound_write(0x14,0); // disable frequency channel 0

}

main()
{

  port1 = 0xff; //
  port2 = 0;

  //CNTLA0 = 0x00;   // turn on RTS (0% volume control)
     speaker_off
```

```
1740     sound_init();  // init sound chip
1741
1742    CBAR = 0x21;  // bank starts at 10000 common 1 starts at 20000
1743    CBR = 0x00;   // common base for common 1 area
1744    BBR = 0x00;   // bank base
1745
1746
1747    gpio1 = 0;  // turn off led
1748    buffer[0]=0x00;
1749    buffer[1]=0xb5;
1750    buffer[2]=0x1f;
1751    buffer[3]=0x8d; //0xa1;
1752    buffer[4]=0x00;
1753    buffer[5]=0xbd;
1754    buffer[6]=0xbf;
1755    buffer[7]=0x30;
1756
1757     PC = 0x8000;
1758        save_PC = 0x8000;
1759
1760     USER_SP = 0xFF00;
1761
1762     state=0;    // reset message
1763     test = 0xff;
1764
1765     InitLcd();
1766     LCDWriteText("Z180 Kit+SAA1099");
1767     goto_xy(0,1);
1768        LCDWriteText(" 32kROM 128kRAM ");
1769
1770     uart_init();
1771
1772
1773     puts("\r\n\nZ180 MICROPROCESSOR KIT 2022");
1774
1775    while(1)
1776    {
1777    scan1();
1778
1779    }
1780  }
1781
1782
1783
1784
```