

```

//-----
// C main line
//-----
-----
#include <m8c.h>
#include "UART_1.h"
#include "RX8_1.h"
#include "PWM8_1.h"
#include "port.h"
#include "COUNTER16_1.h"
#include "I2CM_1.h"

#define NAK_RESPONSE 0x00
BYTE rxBuf[8]; //holds data received form RTC
BYTE txCBuf[5]; //holds data sent to RTC
BYTE tenstemp; //temp variable
BYTE onestemp; //temp variable
BYTE bdata; //returns data from UART or RX8
BYTE ddata; //temp variable
BYTE oldtime[2]; //temp variable
BYTE status; //status for I2C
BYTE bRx8Status = 0x00; //status for RX8
BYTE bRxStatus = 0x00; //status for UART
BYTE check; //temp variable
BYTE entray; //temp variable
int swi; //switch for programming or
dispensing mode
int okay; //temp variable

typedef struct //struct for dispensing trays
{
    char name[14];
    BYTE time[2];
    BYTE start[2];
    BYTE stop[2];
    BYTE traynum;
    BYTE increment;
    char dosage;
}_tray;

_tray MDT[5]; //5 MDTs
//String declarations
const char entertime[20] = "ENTER THE TIME hh:mm";
const char amorpm[19] = "ENTER 1=AM or 2=PM\r";
const char okaytime[14] = "\rCORRECT? Y/N\r";
const char structure[6] = "hh:mm\r";
const char entername[16] = "ENTER MED. NAME\r";
const char entertray[18] = "ENTER TRAY NUMBER\r";
const char traytaken1[14] = "TRAY IS TAKEN\r";
const char traytaken2[17] = "1=OVERWRITE TRAY\r";
const char traytaken3[15] = "2=REENTER TRAY\r";
const char enterstart[21] = "ENTER THE START TIME";
const char enterstop[20] = "ENTER THE STOP TIME\r";
const char enterinc[20] = "HOURS BETWEEN MED's";
const char too[4] = " to ";
const char another[20] = "PROGRAM ANOTHER MED";
const char enterdosage[17] = "ENTER THE DOSAGE\r";
const char incanddos[16] = " INCRS - DOSAGE ";
const char intro[4] = "AMDS";

```

```

//Delay routine:
//Used mainly for delay in transferring data.
void Delay(void)
{
    int x=0;
    while(x!=400)
    {
        x = x+1;
    }
    return;
}

//Converts Hex values to BCD values
BYTE HEXTOBCD(BYTE num)
{
    BYTE tens;
    BYTE ones;
    BYTE temp;

    tens = 0x00;
    ones = 0x00;
    temp = num;
    while(temp <= num)
    {
        temp = temp - 10;
        tens = tens + 0x01;
    }
    tens = tens - 1;
    temp = temp + 10;
    tens = tens * 0x10;
    temp = tens|temp;
    return temp; //ones||tens;
}

//converts the ones digit of a BCD number
//to an ASCII value
BYTE onesASCITOB CD(BYTE num)
{
    switch(num)
    {
        case '0': return 0x00;
        case '1': return 0x01;
        case '2': return 0x02;
        case '3': return 0x03;
        case '4': return 0x04;
        case '5': return 0x05;
        case '6': return 0x06;
        case '7': return 0x07;
        case '8': return 0x08;
        case '9': return 0x09;
    }
}
//    return;
}

//converts the tens digit of a BCD number
//to an ASCII value
BYTE tensASCITOB CD(BYTE num)
{
    switch(num)
    {

```

```

        case '0': return 0x00;
        case '1': return 0x10;
        case '2': return 0x20;
        case '3': return 0x30;
        case '4': return 0x40;
        case '5': return 0x50;
        case '6': return 0x60;
        case '7': return 0x70;
        case '8': return 0x80;
        case '9': return 0x90;
    }
    // return;
}

//converts BCD value to ASCII
char BCDtoASCII(BYTE num)
{
    switch(num)
    {
        case 0x00: return '0';
        case 0x01: return '1';
        case 0x02: return '2';
        case 0x03: return '3';
        case 0x04: return '4';
        case 0x05: return '5';
        case 0x06: return '6';
        case 0x07: return '7';
        case 0x08: return '8';
        case 0x09: return '9';
        case 0x10: return '1';
        case 0x20: return '2';
        case 0x30: return '3';
        case 0x40: return '4';
        case 0x50: return '5';
        case 0x60: return '6';
        case 0x70: return '7';
        case 0x80: return '8';
        case 0x90: return '9';
    }
    // return;
}

//converts a BCD value to Integer
int BCDtoINT(BYTE num)
{
    switch(num)
    {
        case 0x00: return 0;
        case 0x01: return 1;
        case 0x02: return 2;
        case 0x03: return 3;
        case 0x04: return 4;
        case 0x05: return 5;
        case 0x06: return 6;
        case 0x07: return 7;
        case 0x08: return 8;
        case 0x09: return 9;
        case 0x10: return 10;
        case 0x20: return 20;
        case 0x30: return 30;
        case 0x40: return 40;
    }
}

```

```

        case 0x50: return 50;
        case 0x60: return 60;
        case 0x70: return 70;
        case 0x80: return 80;
        case 0x90: return 90;
    }
    // return;
}

//gets the time
void gettime()//BYTE *txCBuf)
{
    //BYTE status;                // I2C communication status
    BYTE i;                       // Temp counter variable
    BYTE c;
    //int x;
    //I2Cm_1_Start();            // Initialize I2C Master interface

    // In a endless loop, keep reading the time from the DS1307
    //do {

        // Write sub-address to DS1307
        // Perform a combined transfer by leaving off the Stop on the
Write
        // command and beginning the Read with a Repeat Start.

        I2Cm_1_bWriteCBytes(0x68,txCBuf,1,I2Cm_1_NoStop );

        status = I2Cm_1_fReadBytes(0x68,rxBuf,7,I2Cm_1_RepStart );

        if(status == 0) {
            // Flag an error condition
        }

        // This next section of code performs exactly the same sequence
        // as the above code, but with low level commands.

        I2Cm_1_fSendStart(0x68,I2Cm_1_WRITE);        // Do a write
        I2Cm_1_fWrite(0x00);                          // Set sub
address
                                                    // to zero
        I2Cm_1_fSendRepeatStart(0x68,I2Cm_1_READ);  // Do a read

        for(i = 0; i < 6; i++) {
            rxBuf[i] = I2Cm_1_bRead(I2Cm_1_ACKslave); // Read first 6
bytes,
                                                    // and ACK the
slave
        }

        rxBuf[7] = I2Cm_1_bRead(I2Cm_1_NAKslave); // Read data byte and
                                                    // NAK the slave
to signify
                                                    // end of read.

        I2Cm_1_SendStop();

        UART_1_SendData(16);                //Set the LCD in cursor mode
        Delay();
        UART_1_SendData(75);                //Set the LCD cursor position to 75
        Delay();
        //check = 0xAF;

```

```

        check = rxBuf[2]&0x20; //check AM or PM
        for(i = 3; i >= 1; i--)
        {
            if(i==3)
            {
                c = BCDtoASCI(rxBuf[i-1]&0x10); //Drop the
AM/PM bits and ones min bits
                UART_1_SendData(c); //Send tens
Hour digit
                Delay();
            }
            else
            {
                c = BCDtoASCI(rxBuf[i-1]&0xF0); //Drop tens
mins bits
                UART_1_SendData(c); //Send tens
mins bits
                Delay();
            }

            c = BCDtoASCI(rxBuf[i-1]&0x0F); //Sends ones
hour and mins bits
            UART_1_SendData(c);
            Delay();
            if(i > 1)
            {
                UART_1_SendData(':');
                Delay();
            }
        }

        if(check == 0x20) //Print am or pm
        {
            UART_1_SendData('P');
            Delay();
        }
        else
        {
            UART_1_SendData('A');
            Delay();
        }

        cur_home();
    }

//clear lcd screen
void clr_screen(void)
{
    UART_1_SendData(12);
}

//big character start
void bchar_start(void)
{
    UART_1_SendData(2);
}

//big character stop
void bchar_stop(void)

```

```

{
    UART_1_SendData(3);
}

//LCD Backlight on
void blight_on(void)
{
    UART_1_SendData(14);
}

//LCD Backlight off
void blight_off(void)
{
    UART_1_SendData(15);
}

//cursor to 1st position
void cur_home(void)
{
    UART_1_SendData(1);
}

//carriage return
void carr_ret(void)
{
    UART_1_SendData(13);
}

//Converts keyboard position to an
//ASCII value
BYTE KeyboardCon(BYTE keycode)
{
    switch(keycode)
    {
        case 0x00: return '1';
        case 0x01: return '2';
        case 0x02: return '3';
        case 0x03: return 'Z';
        case 0x04: return '4';
        case 0x05: return '5';
        case 0x06: return '6';
        case 0x07: return '7';
        case 0x09: return 'Q';
        case 0x0A: return 'W';
        case 0x0B: return 'E';
        case 0x0C: return 'R';
        case 0x0D: return 'T';
        case 0x0E: return 'Y';
        case 0x10: return 'X';
        case 0x11: return 'A';
        case 0x12: return 'S';
        case 0x13: return 'D';
        case 0x14: return 'F';
        case 0x15: return 'G';
        case 0x16: return 'H';
        case 0x17: return ' ';
        case 0x2C: return 'C';
        case 0x2D: return 'V';
        case 0x2E: return 'B';
        case 0x2F: return 'N';
        case 0x32: return 8; //backspace
    }
}

```

```

        case 0x34: return '8';
        case 0x35: return '9';
        case 0x36: return '0';
        case 0x3C: return 'U';
        case 0x3D: return 'I';
        case 0x3E: return 'O';
        case 0x3F: return 'P';
        case 0x41: return 13;//enter
        //case 0x49: return 0x13;//enter
        case 0x44: return 'J';
        case 0x45: return 'K';
        case 0x46: return 'L';
        case 0x49: return 14; //scroll up turn backlight on
        case 0x4C: return 'M';
        case 0x50: return '?'; //delete
        case 0x51: return '<'; //left arrow
        case 0x52: return 15; //scroll down turn backlight off
        case 0x53: return '>'; //right arrow
        case 0x33: return 12;
        case 0x3B: return 2; //Big char start
        case 0x42: return 3; //Big char stop
    }
}

//Gets Data from PC
BYTE PC(void)
{
    BYTE bData;
    //BYTE bRxStatus;
    //BYTE ddata;

    //while(1)
    //{
        /* wait for data to be received */

        //while( !( bRxStatus=bUART_1_ReadRxStatus() & UART_RX_COMPLETE )
);

        /* preset response to error */

        bData = NAK_RESPONSE;

        /* if no error condition then get byte received */

        if ( ! (bRxStatus & UART_RX_NO_ERROR) )
        {
            /* no error received */

            bData = bUART_1_ReadRxData();
        }
        return bData;
    }
}

//Gets Data from Keyboard
BYTE Keyboard(void)
{
    BYTE bData;
    //BYTE bRx8Status;
    BYTE ddata;

```

```

/* wait for data to be received */
while( !( bRx8Status=bRX8_1_ReadRxStatus() & RX8_RX_COMPLETE ) );

/* preset response to error */
bData = NAK_RESPONSE;

/* if no error condition then get byte received */
if ( ! (bRx8Status & RX8_RX_NO_ERROR) )
{
    /* no error received */

    bData = bRX8_1_ReadRxData();
    }
    ddata = bData;
    bData = KeyboardCon(ddata);
/* transmit the response data */
//UART_1_SendData(bData);
return bData;
}

//Increment the hours time
BYTE incrementtime(BYTE time,BYTE increment)
{
    BYTE temp;
    BYTE temp2;
    BYTE hexhours;

    temp = time&0x60;

    //PM
    if (temp == 0x60)
    {
        //clear out upper nibble & add increment
        temp2 = (time&0x1F) + increment;

        //convert to BCD if tens
        if (temp2 > 0x09)
        {
            temp2 = temp2 - 0x0A;
            temp2 = temp2|0x10;
        }

        //check 12AM
        if (temp2 == 0x12)
        {
            temp2 = temp2|0x40; //set to AM, & 12 hour mode
        }
        //check AM rollover
        else if(temp2 > 0x12)
        {
            temp2 = temp2 - 0x12; //subtract off 12 hours
            temp2 = temp2|0x40; //set to AM, & 12 hour mode
        }
        //check PM
        else

```

```

        {
            temp2 = time + increment;
        }
    }
    //AM
    else
    {

        //clear out upper nibble & add increment
        temp2 = (time&0x1F) + increment;

        //convert to BCD if tens
        if (temp2 > 0x09) //convert to BCD
        {
            temp2 = temp2 - 0x0A;
            temp2 = temp2|0x10;
        }

        //check 12PM
        if (temp2 == 0x12)
        {
            temp2 = temp2|0x60;
        }
        //check PM rollover
        else if (temp2 > 0x12)
        {
            temp2 = temp2 - 0x12;
            temp2 = hexhours|0x60;
        }
        //check AM
        else
        {
            //temp2 = time + increment;
            temp2 = temp2|0x40;
        }
    }
    return temp2;
}

//get data Byte from PC or Keyboard
BYTE cin(void)
{

    //BYTE bdata;
    BYTE temp;
    //BYTE ddata;

    int xx = 0;
    while(xx <= 0)
    {
        if(( bRX8_1_ReadRxStatus() & RX8_RX_COMPLETE )) //check
Keyboard
        {
            //bdata = Keyboard(bRX8_1_ReadRxStatus());
            // wait for data to be received
            //while( !( bRx8Status=bRX8_1_ReadRxStatus() & RX8
_RX_COMPLETE ) );
            // preset response to error
            bdata = NAK_RESPONSE;
            //if no error condition then get byte received
            if ( ! (bRx8Status & RX8_RX_NO_ERROR) )

```

```

        {
            //no error received
            bdata = bRX8_1_ReadRxData();
        }
        ddata = bdata;
        bdata = KeyboardCon(ddata);

        while( !( bRx8Status=bRX8_1_ReadRxStatus() & RX8
_RX_COMPLETE ) );
        temp = Keyboard();//bRX8_1_ReadRxStatus());

        while( !( bRx8Status=bRX8_1_ReadRxStatus() & RX8
_RX_COMPLETE ) );
        temp = Keyboard();//bRX8_1_ReadRxStatus());
        xx = 1;
    }
    else
    {
        if(( bUART_1_ReadRxStatus() & UART_RX_COMPLETE ))
//check PC ready
        {
            //bdata = PC(bUART_1_ReadRxStatus());
            bdata = NAK_RESPONSE;
            //if no error condition then get byte received
            if ( ! (bRxStatus & UART_RX_NO_ERROR) )
            {
                // no error received
                bdata = bUART_1_ReadRxData();
            }
            xx = 1;
        }
    }
}

return bdata;
}

//Print the const char intialized
//at the beginning of program
void ccout(char r[],int size)
{
    int l;
    for(l = 0; l <= size; l++)
    {
        if(r[l] == 13)
        {
            UART_1_SendData(r[l]);
            Delay();
            l = size;
        }
        else
        {
            UART_1_SendData(r[l]);
            Delay();
        }
    }
    return;
}

//display char string
void cout(const char r[],int size)

```

```

{
    int l;
    for(l = 0; l <= size; l++)
    {
        UART_1_SendData(r[l]);
        Delay();
    }
    return;
}

//Sets the clock
void setClock(void)
{
    I2Cm_1_Start();           // Initialize I2C Master
interface
    // Set the time
    status = I2Cm_1_bWriteBytes(0x68,txCBuf,9,I2Cm_1
_CompleteXfer);
    //I2Cm_1_Stop();
    return;
}

//HEART OF OPERATION
void main()
{
    char Y; //Yes/No character
    char a; //am or pm character
    BYTE x[8];
    BYTE TempTime[2]; //old time
    //Flags
    int button_flag = 0; //0=not pressed, 1=pressed
    int lid_flag = 0; // 0=closed, 1=open
    int dispense_flag = 1; //0=no dispensing, 1=dispense
    int i, j, p; //loop variables
    int z = 1;

    // Intialization of API's
    M8C_EnableGInt;
    PWM8_1_Start();
    Counter16_1_DisableInt();
    Counter16_1_Start();
    UART_1_Start(UART_PARITY_NONE);
    UART_1_DisableInt();
    RX8_1_Start(RX8_PARITY_NONE);
    RX8_1_DisableInt();

    outPort0_7(0);           //intiallize lid 1
    outPort0_5(0);           //intiallize lid 2
    outPort0_3(0);           //intiallize lid 3
    outPort0_1(0);           //intiallize lid 4
    outPort2_7(0);           //intiallize lid 5

    lid_flag = 1;
    i = 0;

    outPort1_1(0);
    outPort1_6(0);

    Delay();
    UART_1_SendData(12); //clear screen
    Delay();
}

```

```

bchar_start();           //set big character mode
Delay();
cout(intro,4);           //Display AMDS
Delay();

//zero all MDT variables
for(j = 0; j <= 4; j++)
{
    MDT[j].traynum = 0x00;
    MDT[j].start[1] = 0x00;
    MDT[j].start[0] = 0x00;
    MDT[j].stop[1] = 0x00;
    MDT[j].stop[0] = 0x00;
    MDT[j].time[1] = 0x00;
    MDT[j].time[0] = 0x00;
    Delay();
}

for(j = 0; j <= 1000; j++)
{
    Delay();
}

clr_screen();
Delay();

//SET TIME FUNCITONALITY
outPort1_1(0);

while(z >=1)
{
    cout(entertime,19);
    txCBuf[0] = 0x00;           //address port of RTC
    txCBuf[1] = 0x00;           //seconds are zero
    txCBuf[4] = 0x93;           //start clocking
    tenstemp = 0x41;
    tenstemp = cin();           //input tens digit of hours
    UART_1_SendData(tenstemp); //display digit
    Delay();
    tenstemp = tensASCITOBcd(tenstemp); //convert the digit to
BCD
    onestemp = cin();           //input ones digit of hours
    UART_1_SendData(onestemp); //display digit
    Delay();
    onestemp = onesASCITOBcd(onestemp); //convert the
digit to BCD
    txCBuf[3] = tenstemp|onestemp; //load tens BCD and ones
BCD to hours of buffer
    Delay();
    UART_1_SendData(':'); //display colon
    tenstemp = cin();           //input tens digit of mins
    UART_1_SendData(tenstemp); //display digit
    Delay();
    tenstemp = tensASCITOBcd(tenstemp); //convert the digit to
BCD
    onestemp = cin();           //input ones digit of mins
    UART_1_SendData(onestemp); //display digit
    Delay();
    onestemp = onesASCITOBcd(onestemp); //convert the

```

```

digit to BCD
txCBuf[2] = tenstemp|onestemp;          //load tens BCD and ones
BCD to mins buffer
Delay();
clr_screen();
Delay();

//Display the HH:MM and ask to Enter 1 for am or 2 for pm
UART_1_SendData(BCDtoASCI(txCBuf[3]&0xF0));
Delay();
UART_1_SendData(BCDtoASCI(txCBuf[3]&0x0F));
Delay();
UART_1_SendData(':');
Delay();
UART_1_SendData(BCDtoASCI(txCBuf[2]&0xF0));
Delay();
UART_1_SendData(BCDtoASCI(txCBuf[2]&0x0F));
Delay();
UART_1_SendData(13);
Delay();
cout(amorpm,18);
a = cin(); //inputs 1 or 2 for am or pm
if(a == '1')
{
    txCBuf[3] = txCBuf[3]|0x40;
}
else
{
    txCBuf[3] = txCBuf[3]|0x60;
}

clr_screen();
Delay();
UART_1_SendData(BCDtoASCI(txCBuf[3]&0x10));
Delay();
UART_1_SendData(BCDtoASCI(txCBuf[3]&0x0F));
Delay();
UART_1_SendData(':');
Delay();
UART_1_SendData(BCDtoASCI(txCBuf[2]&0xF0));
Delay();
UART_1_SendData(BCDtoASCI(txCBuf[2]&0x0F));
Delay();
if(a == '1')
{
    UART_1_SendData('A');
    Delay();
}
else
{
    UART_1_SendData('P');
    Delay();
}
cout(okaytime,13);
Y = cin();

if(Y == 'Y')
{
    clr_screen();
    Delay();
}

```

```

        setClock();
        z = 0;
    }
    else
    {
        z=1;
        Delay();
        clr_screen();
        Delay();
    }
}

//END OF SET TIME.
TempTime[0] = 0x00;
TempTime[1] = 0x00;

swi = 0;
oldtime[0] = 0x00;
oldtime[1] = 0x00;
swi = 0;

while(1)
{
    inPort1_3(swi);
    //swi = 0;
    if(swi == 0x00)
    {
        outPort1_2(0);
        outPort1_0(1);
        gettime();
        i = 0;
        for(j = 1; j <= 4; j++)
        {
            if((oldtime[0] != rxBuf[1]))
            {
                if (((MDT[j-1].time[1]==rxBuf[2])&&(MDT[j-
1].time[0]==rxBuf[1]))||
                ((MDT[j-1].start[1]==rxBuf[2])&&(MDT
[j-1].start[0]==rxBuf[1]))||
                ((MDT[j-1].stop[1]==rxBuf[2])&&(MDT
[j-1].stop[0]==rxBuf[1])))
                {
                    i = j;
                    j = 4;
                    oldtime[0] = rxBuf[1];
                    oldtime[1] = rxBuf[2];
                }
            }
        }
        dispense_flag = i;

        while (dispense_flag > 0)
        {
            i = i - 1;
            //gettime();
            //scroll through all MDTs

            //check MDT dispense times

```

```

//save time
if((TempTime[1] == 0x00)&&(TempTime[0] == 0x00))
{
    TempTime[1] = rxBuf[2]; //hours
    TempTime[0] = rxBuf[1]; //minutes
}
//display information
clr_screen(); //clear the screen
Delay();
blight_off(); //turn backlight off
Delay();
bchar_start(); //switch to big char
Delay();
//cout(MDT[i].dosage, 1); //display
medication dosage

UART_1_SendData(MDT[i].dosage);
Delay();
bchar_stop(); //switch to small char
Delay();
name
ccout(MDT[i].name, 14); //display medication

//set dispense button status
inPort1_6(button_flag);
//check dispense button
while (button_flag == 0)
{
    outPort1_1(1); //turn on LED and buzzer
alarms

    //flash LCD
    for (j = 0; j < 10000; j++)
    {
        blight_on(); //turn backlight
on

        inPort1_6(button_flag);
        if(button_flag == 1)
        {
            j = 10000;
        }
    }
    blight_off(); //turn backlight off
    for (j = 0; j < 10000; j++)
    {
        blight_off(); //turn backlight
on

        if(button_flag == 1)
        {
            j = 10000;
        }
        else
        {
            inPort1_6(button_flag);
        }
    }
}
outPort1_1(0);
outPort1_6(0);
//open according lid and set closure status
switch (i)
{

```

```

        case 0: //outPort0_6(0);
                outPort0_7(1);

//open lid 1

                inPort0_6(lid_flag);
                for(p = 0; p<= 100; p++)
                {
                        Delay();
                }
                outPort0_7(0);
                break;
        case 1: //outPort0_4(0);
                outPort0_5(1);

//open lid 2

                inPort0_4(lid_flag);
                for(p = 0; p<= 100; p++)
                {
                        Delay();
                }
                outPort0_5(0);
                break;
        case 2: //outPort0_2(0);
                outPort0_3(1);

//open lid 3

                inPort0_2(lid_flag);
                for(p = 0; p<= 100; p++)
                {
                        Delay();
                }
                outPort0_3(0);
                break;
        case 3: //outPort0_0(0);
                outPort0_1(1);

//open lid 4

                inPort0_0(lid_flag);
                for(p = 0; p<= 100; p++)
                {
                        Delay();
                }
                outPort0_1(0);
                break;
        case 4: //outPort2_6(0);
                outPort2_7(1);

//open lid 5

                inPort2_6(lid_flag);
                for(p = 0; p<= 100; p++)
                {
                        Delay();
                }
                outPort2_7(0);
                break;
    }
    lid_flag = 1;
    while (lid_flag >= 1) //check for lid closure
    {
        //set closure status
        switch (i)
        {
            case 0: inPort0_6(lid_flag);
                    break;
            case 1: inPort0_4(lid_flag);
                    break;

```

```

        case 2: inPort0_2(lid_flag);
                break;
        case 3: inPort0_0(lid_flag);
                break;
        case 4: inPort2_6(lid_flag);
                break;
    }
}

outPort0_6(0);
outPort0_4(0);
outPort0_2(0);
outPort0_0(0);
outPort0_2(0);
//increment to next dispense time
if(MDT[i].time[1] == MDT[i].stop[1])
{
    MDT[i].time[1] = MDT[i].start[1];
}
else
{
    MDT[i].time[1] = incrementtime(MDT[i].time
[1], MDT[i].increment);
}
//check other MDT times
dispense_flag = 0;
gettime();
clr_screen();
Delay();
j = i;
i = 0;
for (p = j+2; p <= 5; p++)
{
    if ((MDT[p-1].time[0] <= rxBuf[1])&&(MDT
[p-1].time[0] >= TempTime[0])&&
(MDT[p-1].time[1] <= rxBuf[2])&&(MDT
[p-1].time[1] >= TempTime[1])||
(MDT[p-1].start[0] > rxBuf[1])&&(MDT
[p-1].start[0] < TempTime[0])&&
(MDT[p-1].start[1] > rxBuf[2])&&(MDT
[p-1].start[1] < TempTime[1])||
(MDT[p-1].stop[0] > rxBuf[1])&&(MDT
[p-1].stop[0] < TempTime[0])&&
(MDT[p-1].stop[1] > rxBuf[2])&&(MDT
[p-1].stop[1] < TempTime[1]))
    {
        i = p;
        p = 5; //set dispense flag
    }
}
dispense_flag = i;
}
TempTime[0] = 0x00;
TempTime[1] = 0x00;
}
else
{
    //PROGRAMMING MODE
    dispense_flag = 0;
    outPort1_2(1);
}

```

```

outPort1_0(0);
gettime();
TempTime[1] = rxBuf[2]; //hours
TempTime[0] = rxBuf[1]; //minutes

while(dispense_flag <= 0)
{
    okay = 1;
    while(okay != 0)
    {
        clr_screen();
        Delay();
        cout(entertray,17);
        Delay();
        entray = cin();
        //entray = '1';
        clr_screen();
        Delay();
        switch(entray)
        {
            case '1': i = 0;
                if(MDT[0].traynum !=
0x00)
                {
                    UART_1_SendData
                    Delay();
                    clr_screen();
                    Delay();
                    cout
                    cout
                    cout
                    a = cin();
                    //a = '1';
                    if(a == '1')
                    {
                        MDT
                        okay = 0;
                    }
                }
            else
            {
                UART_1_SendData
                Delay();
                MDT[0].traynum =
                okay = 0;
            }
            break;
            case '2': i = 1;
                if(MDT[1].traynum !=
0x00)
                {
                    UART_1_SendData

```

```

(' ');

(traytaken1,13);
(traytaken2,16);
(traytaken3,14);

[1].traynum = onesASCITOB CD(entray);

(entray);

onesASCITOB CD(entray);

0x00)

(' ');

(traytaken1,14);
(traytaken2,17);
(traytaken3,15);

[2].traynum = onesASCITOB CD(entray);

(entray);

onesASCITOB CD(entray);

Delay();
clr_screen();
Delay();
cout

cout

cout

a = cin();
if(a == '1')
{
    MDT

    okay = 0;
}
}
else
{
    UART_1_SendData

    Delay();
    MDT[1].traynum =

    okay = 0;
}
break;

case '3': i = 2;
if(MDT[2].traynum !=

{
    UART_1_SendData

    Delay();
    clr_screen();
    Delay();
    cout

    cout

    cout

    a = cin();
    if(a == '1')
    {
        MDT

        okay = 0;
    }
}
else
{
    UART_1_SendData

    Delay();
    MDT[2].traynum =

    okay = 0;
}
}

```



```

(entray);
onesASCITOBBCD(entray);

UART_1_SendData
Delay();
MDT[4].traynum =
okay = 0;
}
break;
}
}
clr_screen();
Delay();
cout(entername,15);
for(p = 0; p <= 20; p++)
{
    MDT[i].name[p] = cin();
    UART_1_SendData(MDT[i].name[p]);
    if(MDT[i].name[p] == 13)
    {
        p = 20;
    }
}

clr_screen();
Delay();
cout(enterdosage,16);
MDT[i].dosage = cin();
clr_screen();
Delay();

cout(enterstart,19);
cout(structure,5);
tenstemp = cin();
UART_1_SendData(tenstemp);
Delay();
tenstemp = tensASCITOBBCD(tenstemp);
onestemp = cin();
UART_1_SendData(onestemp);
Delay();
onestemp = onesASCITOBBCD(onestemp);
MDT[i].start[1] = tenstemp|onestemp;
Delay();
UART_1_SendData(':');
tenstemp = cin();
UART_1_SendData(tenstemp);
Delay();
tenstemp = tensASCITOBBCD(tenstemp);
onestemp = cin();
UART_1_SendData(onestemp);
Delay();
onestemp = onesASCITOBBCD(onestemp);
MDT[i].start[0] = tenstemp|onestemp;
Delay();
//clr_screen();
UART_1_SendData(13);
Delay();

cout(amorpm,18);
a = cin();
if(a == '1')
{

```

```

        MDT[i].start[1] = MDT[i].start[1]|0x40;
    }
    else
    {
        MDT[i].start[1] = MDT[i].start[1]|0x60;
    }

    clr_screen();
    Delay();
    cout(enterstop,19);
    cout(structure,5);
    tenstemp = cin();
    UART_1_SendData(tenstemp);
    Delay();
    tenstemp = tensASCITOB CD(tenstemp);
    onestemp = cin();
    UART_1_SendData(onestemp);
    Delay();
    onestemp = onesASCITOB CD(onestemp);
    MDT[i].stop[1] = tenstemp|onestemp;
    Delay();
    UART_1_SendData(':');
    tenstemp = cin();
    UART_1_SendData(tenstemp);
    Delay();
    tenstemp = tensASCITOB CD(tenstemp);
    onestemp = cin();
    UART_1_SendData(onestemp);
    Delay();
    onestemp = onesASCITOB CD(onestemp);
    MDT[i].stop[0] = tenstemp|onestemp;
    Delay();
    UART_1_SendData(13);
    Delay();

    cout(amorpm,18);
    a = cin();
    if(a == '1')
    {
        MDT[i].stop[1] = MDT[i].stop[1]|0x40;
    }
    else
    {
        MDT[i].stop[1] = MDT[i].stop[1]|0x60;
    }

    clr_screen();
    Delay();
    cout(enterinc,20);
    MDT[i].increment = cin();
    UART_1_SendData(MDT[i].increment);
    MDT[i].increment = onesASCITOB CD(MDT
[i].increment);

    clr_screen();
    Delay();
    ccout(MDT[i].name,20);
    UART_1_SendData(BCDtoASCI(MDT[i].start[1]&
0x10));

    Delay();
    UART_1_SendData(BCDtoASCI(MDT[i].start[1]&

```

```

0x0F));
    Delay();
    UART_1_SendData(':');
    Delay();
    UART_1_SendData(BCDtoASCII(MDT[i].start[0]&
0xF0));
    Delay();
    UART_1_SendData(BCDtoASCII(MDT[i].start[0]&
0x0F));
    Delay();
    check = MDT[i].start[1]&0x20;
    if(check == 0x20)
    {
        UART_1_SendData('P');
        Delay();
    }
    else
    {
        UART_1_SendData('A');
        Delay();
    }
    cout(too,3);
    UART_1_SendData(BCDtoASCII(MDT[i].stop[1]&0x10));
    Delay();
    UART_1_SendData(BCDtoASCII(MDT[i].stop[1]&0x0F));
    Delay();
    UART_1_SendData(':');
    Delay();
    UART_1_SendData(BCDtoASCII(MDT[i].stop[0]&0xF0));
    Delay();
    UART_1_SendData(BCDtoASCII(MDT[i].stop[0]&0x0F));
    Delay();
    check = MDT[i].stop[1]&0x20;
    if(check == 0x20)
    {
        UART_1_SendData('P');
        Delay();
    }
    else
    {
        UART_1_SendData('A');
        Delay();
    }
    UART_1_SendData(13);
    Delay();
    UART_1_SendData(BCDtoASCII(MDT[i].increment));
    Delay();
    cout(incanddos,15);
    Delay();
    UART_1_SendData(MDT[i].dosage);
    Delay();

    cout(okaytime,13);
    Y = cin();
    if(Y == 'Y')
    {
        MDT[i].time[0] = MDT[i].start[0];
        MDT[i].time[1] = MDT[i].start[1];
        clr_screen();
        Delay();
        cout(another,19);

```

```

        cout(okaytime,13);
        a = cin();
        if(a == 'N')
        {
            dispense_flag = 1;
            inPort1_3(swi);
            while(swi != 0)
            {
                inPort1_3(swi);
            }
            //swi = 1;
            outPort1_3(0);
            clr_screen();
        }
        else
        {
            dispense_flag = 0;
        }
    }
else
{
    for(p = 0; p <= 20; p++)
    {
        if(MDT[i].name[p] == 13)
        {
            MDT[i].name[p] = 0x00;
            p = 20;
        }
        else
        {
            MDT[i].name[p] = 0x00;
        }
    }
    MDT[i].traynum = 0x00;
    MDT[i].dosage = '';
    MDT[i].time[0] = 0x00;
    MDT[i].time[1] = 0x00;
    MDT[i].start[0] = 0x00;
    MDT[i].start[1] = 0x00;
    MDT[i].stop[0] = 0x00;
    MDT[i].stop[1] = 0x00;
    MDT[i].increment = 0x00;
}
}
}
}
}

/* HEADER FILE STARTS HERE */

/*PORT.H : This header file contains all of the definitions needed to
output or input
to a port. If a binary or hex number should be
outputted to the whole port:
call outPort*(somevalue). If one pin should be
asserted call outPort*_(1 or 0)
*/
#define outPort0(h) (PRT2DR = h)
#define outPort0_0(b) (PRT0DR = (b==0) ? (PRT0DR&0xFE) : (PRT0DR|0x01))
#define outPort0_1(b) (PRT0DR = (b==0) ? (PRT0DR&0xFD) : (PRT0DR|0x02))

```

```

#define outPort0_2(b) (PRT0DR = (b==0) ? (PRT0DR&0xFB) : (PRT0DR | 0x04))
#define outPort0_3(b) (PRT0DR = (b==0) ? (PRT0DR&0xF7) : (PRT0DR | 0x08))
#define outPort0_4(b) (PRT0DR = (b==0) ? (PRT0DR&0xEF) : (PRT0DR | 0x10))
#define outPort0_5(b) (PRT0DR = (b==0) ? (PRT0DR&0xDF) : (PRT0DR | 0x20))
#define outPort0_6(b) (PRT0DR = (b==0) ? (PRT0DR&0xBF) : (PRT0DR | 0x40))
#define outPort0_7(b) (PRT0DR = (b==0) ? (PRT0DR&0x7F) : (PRT0DR | 0x80))
#define outPort1(h) (PRT2DR = h)
#define outPort1_0(b) (PRT1DR = (b==0) ? (PRT1DR&0xFE) : (PRT1DR | 0x01))
#define outPort1_1(b) (PRT1DR = (b==0) ? (PRT1DR&0xFD) : (PRT1DR | 0x02))
#define outPort1_2(b) (PRT1DR = (b==0) ? (PRT1DR&0xFB) : (PRT1DR | 0x04))
#define outPort1_3(b) (PRT1DR = (b==0) ? (PRT1DR&0xF7) : (PRT1DR | 0x08))
#define outPort1_4(b) (PRT1DR = (b==0) ? (PRT1DR&0xEF) : (PRT1DR | 0x10))
#define outPort1_5(b) (PRT1DR = (b==0) ? (PRT1DR&0xDF) : (PRT1DR | 0x20))
#define outPort1_6(b) (PRT1DR = (b==0) ? (PRT1DR&0xBF) : (PRT1DR | 0x40))
#define outPort1_7(b) (PRT1DR = (b==0) ? (PRT1DR&0x7F) : (PRT1DR | 0x80))
#define outPort2(h) (PRT2DR = h)
#define outPort2_0(b) (PRT2DR = (b==0) ? (PRT2DR&0xFE) : (PRT2DR | 0x01))
#define outPort2_1(b) (PRT2DR = (b==0) ? (PRT2DR&0xFD) : (PRT2DR | 0x02))
#define outPort2_2(b) (PRT2DR = (b==0) ? (PRT2DR&0xFB) : (PRT2DR | 0x04))
#define outPort2_3(b) (PRT2DR = (b==0) ? (PRT2DR&0xF7) : (PRT2DR | 0x08))
#define outPort2_4(b) (PRT2DR = (b==0) ? (PRT2DR&0xEF) : (PRT2DR | 0x10))
#define outPort2_5(b) (PRT2DR = (b==0) ? (PRT2DR&0xDF) : (PRT2DR | 0x20))
#define outPort2_6(b) (PRT2DR = (b==0) ? (PRT2DR&0xBF) : (PRT2DR | 0x40))
#define outPort2_7(b) (PRT2DR = (b==0) ? (PRT2DR&0x7F) : (PRT2DR | 0x80))
#define outPort3(h) (PRT2DR = h)
#define outPort3_0(b) (PRT3DR = (b==0) ? (PRT3DR&0xFE) : (PRT3DR | 0x01))
#define outPort3_1(b) (PRT3DR = (b==0) ? (PRT3DR&0xFD) : (PRT3DR | 0x02))
#define outPort3_2(b) (PRT3DR = (b==0) ? (PRT3DR&0xFB) : (PRT3DR | 0x04))
#define outPort3_3(b) (PRT3DR = (b==0) ? (PRT3DR&0xF7) : (PRT3DR | 0x08))
#define outPort3_4(b) (PRT3DR = (b==0) ? (PRT3DR&0xEF) : (PRT3DR | 0x10))
#define outPort3_5(b) (PRT3DR = (b==0) ? (PRT3DR&0xDF) : (PRT3DR | 0x20))
#define outPort3_6(b) (PRT3DR = (b==0) ? (PRT3DR&0xBF) : (PRT3DR | 0x40))
#define outPort3_7(b) (PRT3DR = (b==0) ? (PRT3DR&0x7F) : (PRT3DR | 0x80))

#define inPort0(b) (b = PRT0DR)
#define inPort0_0(b) (b = PRT0DR&0x01)
#define inPort0_1(b) (b = PRT0DR&0x02)
#define inPort0_2(b) (b = PRT0DR&0x04)
#define inPort0_3(b) (b = PRT0DR&0x08)
#define inPort0_4(b) (b = PRT0DR&0x10)
#define inPort0_5(b) (b = PRT0DR&0x20)
#define inPort0_6(b) (b = PRT0DR&0x40)
#define inPort0_7(b) (b = PRT0DR&0x80)
#define inPort1(b) (b = PRT1DR)
#define inPort1_0(b) (b = PRT1DR&0x01)
#define inPort1_1(b) (b = PRT1DR&0x02)
#define inPort1_2(b) (b = PRT1DR&0x04)
#define inPort1_3(b) (b = PRT1DR&0x08)
#define inPort1_4(b) (b = PRT1DR&0x10)
#define inPort1_5(b) (b = PRT1DR&0x20)
#define inPort1_6(b) (b = PRT1DR&0x40)
#define inPort1_7(b) (b = PRT1DR&0x80)
#define inPort2(b) (b = PRT2DR)
#define inPort2_0(b) (b = PRT2DR&0x01)
#define inPort2_1(b) (b = PRT2DR&0x02)
#define inPort2_2(b) (b = PRT2DR&0x04)
#define inPort2_3(b) (b = PRT2DR&0x08)
#define inPort2_4(b) (b = PRT2DR&0x10)
#define inPort2_5(b) (b = PRT2DR&0x20)
#define inPort2_6(b) (b = PRT2DR&0x40)
#define inPort2_7(b) (b = PRT2DR&0x80)

```

```
#define inPort3(b) (b = PRT3DR)
#define inPort3_0(b) (b = PRT3DR&0x01)
#define inPort3_1(b) (b = PRT3DR&0x02)
#define inPort3_2(b) (b = PRT3DR&0x04)
#define inPort3_3(b) (b = PRT3DR&0x08)
#define inPort3_4(b) (b = PRT3DR&0x10)
#define inPort3_5(b) (b = PRT3DR&0x20)
#define inPort3_6(b) (b = PRT3DR&0x40)
#define inPort3_7(b) (b = PRT3DR&0x80)
```