8051 PROGRAMMING IN C

WHY PROGRAM 8051 IN C

- Compilers produce hex files that is downloaded to ROM of microcontroller
  - The size of hex file is the main concern
    - Microcontrollers have limited on-chip ROM
    - Code space for 8051 is limited to 64K bytes
- C programming is less time consuming, but has larger hex file size
- The reasons for writing programs in C
  - It is easier and less time consuming to write in C than Assembly
  - C is easier to modify and update
  - You can use code available in function libraries
  - C code is portable to other microcontroller with little of no modification

DATA TYPES

- A good understanding of C data types for 8051 can help programmers to create smaller hex files
  - Unsigned char
  - Signed char
  - Unsigned int
  - Signed int
  - Sbit (single bit)
  - Bit and sfr

DATA TYPES

- The character data type is the most natural choice
  - 8051 is an 8-bit microcontroller
- Unsigned char is an 8-bit data type in the range of 0 – 255 (00 – FFH)
  - One of the most widely used data types for the 8051
    - Counter value
    - ASCII characters
- C compilers use the signed char as the default if we do not put the keyword *unsigned*

DATA TYPES

Write an 8051 C program to send values 00 – FF to port P1.

Solution:

```
#include <reg51.h>
void main(void)
   {
     unsigned char z;
     for (z=0;z<=255;z++)
        P1=z;
   }
```

1. Pay careful attention to the size of the data
2. Try to use unsigned *char* instead of *int* if possible

Write an 8051 C program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C, and D to port P1.

Solution:

```
#include <reg51.h>
void main(void)
   {
     unsigned char mynum[]="012345ABCD";
     unsigned char z;
     for (z=0;z<=10;z++)
        P1=mynum[z];
   }
```

Write an 8051 C program to toggle all the bits of P1 continuously.

Solution:

```
//Toggle P1 forever
#include <reg51.h>
void main(void)
  {
    for (;;)
      {
        p1=0x55;
        p1=0xAA;
       }
  }
```

DATA TYPES

- The signed char is an 8-bit data type
  - Use the MSB D7 to represent – or +
  - Give us values from –128 to +127
- The unsigned int is a 16-bit data type
  - Takes a value in the range of 0 to 65535 (0000 – FFFFH)
  - Define 16-bit variables such as memory addresses
  - Set counter values of more than 256
  - Since registers and memory accesses are in 8-bit chunks, the misuse of int variables will result in a larger hex file
- Signed int is a 16-bit data type
  - Use the MSB D15 to represent – or +
  - We have 15 bits for the magnitude of the number from –32768 to +32767

DATA TYPES

Write an 8051 C program to toggle bit D0 of the port P1 (P1.0) 50,000 times.

Solution:

```
#include <reg51.h>
sbit MYBIT=P1^0;

void main(void)
  {
    unsigned int z;
    for (z=0;z<=50000;z++)
       {
         MYBIT=0;
         MYBIT=1;
       }
  }
```

sbit keyword allows access to the single bits of the SFR registers

TIME DELAY

- There are two way s to create a time delay in 8051 C
  - Using the 8051 timer (Chap. 9)
  - Using a simple for loop
  - be mindful of three factors that can affect the accuracy of the delay
    - The 8051 design
      - The number of machine cycle
      - The number of clock periods per machine cycle
    - The crystal frequency connected to the X1 – X2 input pins
    - Compiler choice
      - C compiler converts the C statements and functions to Assembly language instructions
      - Different compilers produce different code

TIME DELAY

Write an 8051 C program to toggle bits of P1 continuously forever with some delay.

Solution:

```
//Toggle P1 forever with some delay in between
//"on" and "off"
#include <reg51.h>
void main(void)
   {
     unsigned int x;
     for (;;)                    //repeat forever
        {
           p1=0x55;
           for (x=0;x<40000;x++); //delay size
                                  //unknown
           p1=0xAA;
           for (x=0;x<40000;x++);
         }
   }
```

We must use the oscilloscope to measure the exact duration

TIME DELAY

Write an 8051 C program to toggle bits of P1 continuously forever with some delay.

Solution:

```
//Toggle P1 forever with some delay in between
//"on" and "off"
#include <reg51.h>
void main(void)
  {
    unsigned int x;
    for (;;)                    //repeat forever
      {
        p1=0x55;
        for (x=0;x<40000;x++); //delay size
                               //unknown
        p1=0xAA;
        for (x=0;x<40000;x++);
       }
  }
```

We must use the oscilloscope to measure the exact duration

Write an 8051 C program to toggle bits of P1 ports continuously with a 250 ms.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
   {
     while (1)                    //repeat forever
        {
          p1=0x55;
          MSDelay(250);
          p1=0xAA;
          MSDelay(250);
        }
   }

void MSDelay(unsigned int itime)
   {
     unsigned int i,j;
     for (i=0;i<itime;i++)
        for (j=0;j<1275;j++);
   }
```

I/O PROGRAMMING

LEDs are connected to bits P1 and P2. Write an 8051 C program that shows the count from 0 to FFH (0000 0000 to 1111 1111 in binary) on the LEDs.

Solution:

```
#include <reg51.h>
#defind LED P2;

void main(void)
   {
      P1=00;                  //clear P1
      LED=0;                  //clear P2
      for (;;)                //repeat forever
         {
            P1++;             //increment P1
            LED++;            //increment P2
         }
   }
```

Ports P0 – P3 are byte-accessable and we use the P0 – P3 labels as defined in the 8051/52 header file.

I/O PROGRAMMING

Write an 8051 C program to get a byte of data form P1, wait 1/2 second, and then send it to P2.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);

void main(void)
   {
     unsigned char mybyte;
     P1=0xFF;                    //make P1 input port
     while (1)
        {
          mybyte=P1;             //get a byte from P1
          MSDelay(500);
          P2=mybyte;             //send it to P2
        }
   }
```

Bit-addressable I/O

Write an 8051 C program to toggle only bit P2.4 continuously without disturbing the rest of the bits of P2.

Solution:

```
//Toggling an individual bit
#include <reg51.h>
sbit mybit=P2^4;

void main(void)
  {
    while (1)
       {
         mybit=1;          //turn on P2.4
         mybit=0;          //turn off P2.4
       }
  }
```

Ports P0 – P3 are bit-addressable and we use *sbit* data type to access a single bit of P0 - P3

Use the Px^y format, where x is the port 0, 1, 2, or 3 and y is the bit 0 – 7 of that port

Bit-addressable I/O

Write an 8051 C program to monitor bit P1.5. If it is high, send 55H to P0; otherwise, send AAH to P2.

Solution:

```
#include <reg51.h>
sbit mybit=P1^5;

void main(void)
   {
     mybit=1;                    //make mybit an input
     while (1)
        {
          if (mybit==1)
              P0=0x55;
          else
              P2=0xAA;
        }
   }
```

Bit-addressable I/O

A door sensor is connected to the P1.1 pin, and a buzzer is connected to P1.7. Write an 8051 C program to monitor the door sensor, and when it opens, sound the buzzer. You can sound the buzzer by sending a square wave of a few hundred Hz.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);
sbit Dsensor=P1^1;
sbit Buzzer=P1^7;

void main(void)
   {
      Dsensor=1;                    //make P1.1 an input
      while (1)
          {
            while (Dsensor==1)//while it opens
               {
                  Buzzer=0;
                  MSDelay(200);
                  Buzzer=1;
                  MSDelay(200);
               }
          }
   }
```

Accessing SFR Addresses 80 - FFH

Write an 8051 C program to turn bit P1.5 on and off 50,000 times.

Solution:

```
sbit MYBIT=0x95;

void main(void)
   {
     unsigned int z;
     for (z=0;z<50000;z++)
         {
           MYBIT=1;
           MYBIT=0;
         }
   }
```

We can access a single bit of any SFR if we specify the bit address

Notice that there is no `#include <reg51.h>`. This allows us to access any byte of the SFR RAM space 80 – FFH. This is widely used for the new generation of 8051 microcontrollers.

LOGIC OPERATIONS

- Logical operators
  - AND (&&), OR (||), and NOT (!)
- Bit-wise operators
  - AND (&), OR (|), EX-OR (^), Inverter (~), Shift Right (>>), and Shift Left (<<)
    - These operators are widely used in software engineering for embedded systems and control

Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	B	A&B	A|B	A^B	~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

LOGIC OPERATIONS

Run the following program on your simulator and examine the results.

Solution:

```
#include <reg51.h>

void main(void)
  {
    P0=0x35 & 0x0F;          //ANDing
    P1=0x04 | 0x68;          //ORing
    P2=0x54 ^ 0x78;          //XORing
    P0=~0x55;                //inversing
    P1=0x9A >> 3;            //shifting right 3
    P2=0x77 >> 4;            //shifting right 4
    P0=0x6 << 4;             //shifting left 4
  }
```

LOGIC OPERATIONS

Write an 8051 C program to toggle all the bits of P0 and P2 continuously with a 250 ms delay. Using the inverting and Ex-OR operators, respectively.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);

void main(void)
   {
      P0=0x55;
      P2=0x55;
      while (1)
          {
             P0=~P0;
             P2=P2^0xFF;
             MSDelay(250);
          }
   }
```

Bit-wise Operators in C

Write an 8051 C program to get bit P1.0 and send it to P2.7 after inverting it.

Solution:

```
#include <reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit membit;

void main(void)
  {
    while (1)
       {
         membit=inbit;    //get a bit from P1.0
         outbit=~membit; //invert it and send
                          //it to P2.7
       }
  }
```

Write an 8051 C program to read the P1.0 and P1.1 bits and issue an ASCII character to P0 according to the following table.

P1.1	P1.0	
0	0	send '0' to P0
0	1	send '1' to P0
1	0	send '2' to P0
1	1	send '3' to P0

Solution:

```
#include <reg51.h>

void main(void)
   {
     unsignbed char z;
     z=P1;
     z=z&0x3;

...
```

```
...
switch (z)
        {
          case(0):
             {
                P0='0';
                break;
             }
          case(1):
             {
                P0='1';
                break;
             }
          case(2):
             {
                P0='2';
                break;
             }
          case(3):
             {
                P0='3';
                break;
             }
        }
   }
```

Write a C program to send out the value 44H serially one bit at a time via P1.0. The LSB should go out first.

Solution:

```
#include <reg51.h>
sbit P1b0=P1^0;
sbit regALSB=ACC^0;

void main(void)
  {
    unsigned char conbyte=0x44;
    unsigned char x;
    ACC=conbyte;
    for (x=0;x<8;x++)
       {
         P1b0=regALSB;
         ACC=ACC>>1;
       }
  }
```

Write a C program to send out the value 44H serially one bit at a time via P1.0. The MSB should go out first.

Solution:

```
#include <reg51.h>
sbit P1b0=P1^0;
sbit regAMSB=ACC^7;

void main(void)
  {
    unsigned char conbyte=0x44;
    unsigned char x;
    ACC=conbyte;
    for (x=0;x<8;x++)
       {
         P1b0=regAMSB;
         ACC=ACC<<1;
       }
  }
```

Example 9-21

Write an 8051 C program to toggle only bit P1.5 continuously every 50 ms. Use Timer 0, mode 1 (16-bit) to create the delay. Test the program on the (a) AT89C51 and (b) DS89C420.

Solution:

```
#include <reg51.h>
void T0M1Delay(void);
sbit mybit=P1^5;
void main(void){
  while (1) {
    mybit=~mybit;
    T0M1Delay();
  }
}
void T0M1Delay(void){
  TMOD=0x01;
  TL0=0xFD;
  TH0=0x4B;
  TR0=1;
  while (TF0==0);
  TR0=0;
  TF0=0;
}
```

FFFFH – 4BFDH = B402H

= 46082 + 1 = 46083

46083 × 1.085 μs = 50 ms

Example 9-25

A switch is connected to pin P1.2. Write an 8051 C program to monitor SW and create the following frequencies on pin P1.7:
SW=0: 500Hz
SW=1: 750Hz, use Timer 0, mode 1 for both of them.

Solution:

```
#include <reg51.h>
sbit sw=P1^2;
sbit mybit=P1^7;
void T0M1Delay(unsigned char);
void main(void){
  SW=1;
  while (1) {
    mybit=~mybit;
    if (SW==0)
       T0M1Delay(0);
    else
       T0M1Delay(1);
  }
}

.....
```

Example 9-25

```
.....

void T0M1Delay(unsigned char c){
   TMOD=0x01;
   if (c==0) {
      TL0=0x67;
      TH0=0xFC;
   }
   else {
      TL0=0x9A;
      TH0=0xFD;
   }
   TR0=1;
   while (TF0==0);
   TR0=0;
   TF0=0;
}
```

FC67H = 64615

65536 – 64615 = 921

921 × 1.085 µs = 999.285 µs

1 / (999.285 µs × 2) = 500 Hz

Example 9-24

Write an 8051 C program to create a frequency of 2500 Hz on pin P2.7. Use Timer 1, mode 2 to create delay.

Solution:

```
#include <reg51.h>
void T1M2Delay(void);
sbit mybit=P2^7;
void main(void){
   unsigned char x;
   while (1) {
      mybit=~mybit;
      T1M2Delay();
   }
}
void T1M2Delay(void){
   TMOD=0x20;
   TH1=-184;
   TR1=1;
   while (TF1==0);
   TR1=0;
   TF1=0;
}
```

1/2500 Hz = 400 µs
400 µs /2 = 200 µs
200 µs / 1.085 µs = 184

SERIAL PORT PROGRAMMING IN C

Example 10-15

Write a C program for 8051 to transfer the letter "A" serially at 4800 baud continuously. Use 8-bit data and 1 stop bit.

Solution:

```
#include <reg51.h>
void main(void){
  TMOD=0x20;            //use Timer 1, mode 2
  TH1=0xFA;             //4800 baud rate
  SCON=0x50;
  TR1=1;
  while (1) {
    SBUF='A';           //place value in buffer
    while (TI==0);
    TI=0;
  }
}
```

Example 10-16

Write an 8051 C program to transfer the message "YES" serially at 9600 baud, 8-bit data, 1 stop bit. Do this continuously.

Solution:

```
#include <reg51.h>
void SerTx(unsigned char);
void main(void){
   TMOD=0x20;              //use Timer 1, mode 2
   TH1=0xFD;               //9600 baud rate
   SCON=0x50;
   TR1=1;                  //start timer
   while (1) {
      SerTx('Y');
      SerTx('E');
      SerTx('S');
   }
}
void SerTx(unsigned char x){
   SBUF=x;                 //place value in buffer
   while (TI==0);          //wait until transmitted
   TI=0;
}
```

Example 10-17

Program the 8051 in C to receive bytes of data serially and put them in P1. Set the baud rate at 4800, 8-bit data, and 1 stop bit.

Solution:

```
#include <reg51.h>
void main(void){
  unsigned char mybyte;
  TMOD=0x20;           //use Timer 1, mode 2
  TH1=0xFA;            //4800 baud rate
  SCON=0x50;
  TR1=1;               //start timer
  while (1) {          //repeat forever
    while (RI==0);     //wait to receive
    mybyte=SBUF;       //save value
    P1=mybyte;         //write value to port
    RI=0;
  }
}
```

Example 11-14

Write a C program that continuously gets a single bit of data from P1.7 and sends it to P1.0, while simultaneously creating a square wave of 200 µs period on pin P2.5. Use Timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.

Solution:

We will use timer 0 mode 2 (auto-reload). One half of the period is 100 µs. 100/1.085 µs = 92, and TH0 = 256 - 92 = 164 or A4H

```
#include <reg51.h>
sbit SW   =P1^7;
sbit IND  =P1^0;
sbit WAVE =P2^5;
void timer0(void) interrupt 1 {
  WAVE=~WAVE;   //toggle pin
}
void main() {
  SW=1;           //make switch input
  TMOD=0x02;
  TH0=0xA4;       //TH0=-92
  IE=0x82;        //enable interrupt for timer 0
  while (1) {
    IND=SW;       //send switch to LED
  }
}
```

Example 11-16

Write a C program using interrupts to do the following:
(a) Receive data serially and send it to P0
(b) Read port P1, transmit data serially, and give a copy to P2
(c) Make timer 0 generate a square wave of 5 kHz frequency on P0.1
Assume that XTAL = 11.0592 MHz. Set the baud rate at 4800.

Solution:

```
#include <reg51.h>
sbit WAVE =P0^1;

void timer0() interrupt 1 {
  WAVE=~WAVE;   //toggle pin
}

void serial0() interrupt 4 {
  if (TI==1) {
    TI=0;        //clear interrupt
  }
  else {
    P0=SBUF;     //put value on pins
    RI=0;        //clear interrupt
  }
}
.....
```

```
.....

void main() {
  unsigned char x;
  P1=0xFF;       //make P1 an input
  TMOD=0x22;
  TH1=0xF6;      //4800 baud rate
  SCON=0x50;
  TH0=0xA4;      //5 kHz has T=200us
  IE=0x92;       //enable interrupts
  TR1=1;         //start timer 1
  TR0=1;         //start timer 0
  while (1) {
    x=P1;        //read value from pins
    SBUF=x;      //put value in buffer
    P2=x;        //write value to pins
  }
}
```

8031/51 INTERFACING TO EXTERNAL MEMORY

- The number of bits that a semiconductor memory chip can store is called chip *capacity*
  - It can be in units of Kbits (kilobits), Mbits (megabits), and so on
- This must be distinguished from the storage capacity of computer systems
  - While the memory capacity of a memory IC chip is always given bits, the memory capacity of a computer system is given in bytes
    - 16M memory chip – 16 megabits
    - A computer comes with 16M memory – 16 megabytes

- Memory chips are organized into a number of locations within the IC
  - Each location can hold 1 bit, 4 bits, 8 bits, or even 16 bits, depending on how it is designed internally
    - The number of locations within a memory IC depends on the address pins
    - The number of bits that each location can hold is always equal to the number of data pins
- To summarize
  - A memory chip contain 2^x location, where x is the number of address pins
  - Each location contains y bits, where y is the number of data pins on the chip
  - The entire chip will contain $2^x \times y$ bits

- One of the most important characteristics of a memory chip is the speed at which its data can be accessed
  - To access the data, the address is presented to the address pins, the READ pin is activated, and after a certain amount of time has elapsed, the data shows up at the data pins
  - The shorter this elapsed time, the better, and consequently, the more expensive the memory chip
  - The speed of the memory chip is commonly referred to as its *access time*

Example

A given memory chip has 12 address pins and 4 data pins. Find: (a) The organization, and (b) the capacity.

Solution:

(a) This memory chip has 4096 locations ($2^{12} = 4096$), and each location can hold 4 bits of data. This gives an organization of 4096×4, often represented as $4K \times 4$.

(b) The capacity is equal to 16K bits since there is a total of 4K locations and each location can hold 4 bits of data.

Example

A 512K memory chip has 8 pins for data. Find:

(a) The organization, and (b) the number of address pins for this memory chip.

Solution:

(a) A memory chip with 8 data pins means that each location within the chip can hold 8 bits of data. To find the number of locations within this memory chip, divide the capacity by the number of data pins. $512K/8 = 64K$; therefore, the organization for this memory chip is $64K \times 8$

(b) The chip has 16 address lines since $2^{16} = 64K$

ROM (Read-only Memory)

- ROM is a type of memory that does not lose its contents when the power is turned off
  - ROM is also called *nonvolatile* memory
- There are different types of read-only memory
  - PROM
  - EPROM
  - EEPROM
  - Flash EPROM
  - Mask ROM

PROM (Programmable ROM)

- PROM refers to the kind of ROM that the user can burn information into
  - PROM is a user-programmable memory
  - For every bit of the PROM, there exists a fuse
- If the information burned into PROM is wrong, that PROM must be discarded since its internal fuses are blown permanently
  - PROM is also referred to as OTP (one-time programmable)
  - Programming ROM, also called *burning* ROM, requires special equipment called a ROM burner or ROM programmer

- EPROM was invented to allow making changes in the contents of PROM after it is burned
  - In EPROM, one can program the memory chip and erase it thousands of times
- A widely used EPROM is called UV-EPROM
  - UV stands for ultra-violet
  - The only problem with UV-EPROM is that erasing its contents can take up to 20 minutes
  - All UV-EPROM chips have a window that is used to shine ultraviolet (UV) radiation to erase its contents

- There is an EPROM programmer (burner), and there is also separate EPROM erasure equipment
- The major disadvantage of UV-EPROM, is that it cannot be programmed while in the system board
- Notice the pattern of the IC numbers
  - Ex. 27128-25 refers to UV-EPROM that has a capacity of 128K bits and access time of 250 nanoseconds
  - 27xx always refers to UV-EPROM chips

- EEPROM has several advantage over EPROM
  - Its method of erasure is electrical and therefore instant, as opposed to the 20-minute erasure time required for UV-EPROM
  - One can select which byte to be erased, in contrast to UV-EPROM, in which the entire contents of ROM are erased
  - One can program and erase its contents while it is still in the system board
    - EEPROM does not require an external erasure and programming device
    - The designer incorporate into the system board the circuitry to program the EEPROM

- Flash EPROM has become a popular user-programmable memory chip since the early 1990s
  - The process of erasure of the entire contents takes less than a second, or might say in a flash
    - The erasure method is electrical
    - It is commonly called flash memory
  - The major difference between EEPROM and flash memory is
    - Flash memory's contents are erased, then the entire device is erased
      - There are some flash memories are recently made so that the erasure can be done block by block
    - One can erase a desired section or byte on EEPROM

Mask ROM

- Mask ROM refers to a kind of ROM in which the contents are programmed by the IC manufacturer, not user-programmable
  - The terminology mask is used in IC fabrication
  - Since the process is costly, mask ROM is used when the needed volume is high and it is absolutely certain that the contents will not change
  - The main advantage of mask ROM is its cost, since it is significantly cheaper than other kinds of ROM, but if an error in the data/code is found, the entire batch must be thrown away

RAM (Random Access Memory)

- RAM memory is called *volatile* memory since cutting off the power to the IC will result in the loss of data
  - Sometimes RAM is also referred to as RAWM (read and write memory), in contrast to ROM, which cannot be written to
- There are three types of RAM
  - Static RAM (SRAM)
  - NV-RAM (nonvolatile RAM)
  - Dynamic RAM (DRAM)

- Storage cells in static RAM memory are made of flip-flops and therefore do not require refreshing in order to keep their data
- The problem with the use of flip-flops for storage cells is that each cell require at least 6 transistors to build, and the cell holds only 1 bit of data
  - In recent years, the cells have been made of 4 transistors, which still is too many
  - The use of 4-transistor cells plus the use of CMOS technology has given birth to a high-capacity SRAM, but its capacity is far below DRAM

NV-RAM (Nonvolatile RAM)

- NV-RAM combines the best of RAM and ROM
  - The read and write ability of RAM, plus the nonvolatility of ROM
- NV-RAM chip internally is made of the following components
  - It uses extremely power-efficient SRAM cells built out of CMOS
  - It uses an internal lithium battery as a backup energy source
  - It uses an intelligent control circuitry
    - The main job of this control circuitry is to monitor the V_{cc} pin constantly to detect loss of the external power supply

Checksum Byte ROM

- To ensure the integrity of the ROM contents, every system must perform the checksum calculation
  - The process of checksum will detect any corruption of the contents of ROM
  - The checksum process uses what is called a checksum byte
    - The checksum byte is an extra byte that is tagged to the end of series of bytes of data

Checksum Byte ROM

- To calculate the checksum byte of a series of bytes of data
  - Add the bytes together and drop the carries
  - Take the 2's complement of the total sum, and that is the checksum byte, which becomes the last byte of the series
- To perform the checksum operation, add all the bytes, including the checksum byte
  - The result must be zero
  - If it is not zero, one or more bytes of data have been changed

Assume that we have 4 bytes of hexadecimal data: 25H, 62H, 3FH, and 52H.(a) Find the checksum byte, (b) perform the checksum operation to ensure data integrity, and (c) if the second byte 62H has been changed to 22H, show how checksum detects the error.

Solution:

(a) Find the checksum byte.

```
    25H
+   62H
+   3FH
+   52H
   118H
```

The checksum is calculated by first adding the bytes. The sum is 118H, and dropping the carry, we get 18H. The checksum byte is the 2's complement of 18H, which is E8H

(b) Perform the checksum operation to ensure data integrity.

```
    25H
+   62H
+   3FH
+   52H
+   E8H
   200H (dropping the carries)
```

Adding the series of bytes including the checksum byte must result in zero. This indicates that all the bytes are unchanged and no byte is corrupted.

(c) If the second byte 62H has been changed to 22H, show how checksum detects the error.

```
    25H
+   22H
+   3FH
+   52H
+   E8H
   1C0H (dropping the carry, we get C0H)
```

Adding the series of bytes including the checksum byte shows that the result is not zero, which indicates that one or more bytes have been corrupted.

DRAM (Dynamic RAM)

- Dynamic RAM uses a capacitor to store each bit
  - It cuts down the number of transistors needed to build the cell
  - It requires constant refreshing due to leakage
- The advantages and disadvantages of DRAM memory
  - The major advantages are high density (capacity), cheaper cost per bit, and lower power consumption per bit
  - The disadvantages is that
    - it must be refreshed periodically, due to the fact that the capacitor cell loses its charge;
    - While it is being refreshed, the data cannot be accessed

Packing Issue in DRAM

- In DRAM there is a problem of packing a large number of cells into a single chip with the normal number of pins assigned to addresses
  - Using conventional method of data access, large number of pins defeats the purpose of high density and small packaging
    - For example, a 64K-bit chip (64K×1) must have 16 address lines and 1 data line, requiring 16 pins to send in the address
  - The method used is to split the address in half and send in each half of the address through the same pins, thereby requiring fewer address pins

Packing Issue in DRAM

- Internally, the DRAM structure is divided into a square of rows and columns
- The first half of the address is called the row and the second half is called column
  - The first half of the address is sent in through the address pins, and by activating RAS (row address strobe), the internal latches inside DRAM grab the first half of the address
  - After that, the second half of the address is sent in through the same pins, and by activating CAS (column address strobe), the internal latches inside DRAM latch the second half of the address

- In the discussion of ROM, we noted that all of them have 8 pins for data
  - This is not the case for DRAM memory chips, which can have any of the x1, x4, x8, x16 organizations

Discuss the number of pins set aside for address in each of the following memory chips. (a) 16K×4 DRAM (b) 16K×4 SRAM

Solution :

Since $2^{14} = 16K$:

(a) For DRAM we have 7 pins (A0-A6) for the address pins and 2 pins for RAS and CAS

(b) For SRAM we have 14 pins for address and no pins for RAS and CAS since they are associated only with DRAM. In both cases we have 4 pins for the data bus.