```
//////////////16 bit addition
// perform addition of two 16 bit numbers 1234h and 9522h
// store the values in r0 ~ r3
// the result in r7 and r6
// store the result in 51h and 50h  (optional operation)
// expected output= a756h

org 00h

mov ro, #34h
mov r1, #12h
mov r2, #22h
mov r3, #095h  // for data > 7fh need to append zero before.
clr c // clear carry

mov a,ro // LSB addition
add a,r2
mov r6,a  // store lsb in r6
mov 50h, a

mov a,r1 // MSB addion
addc a,r3   // add with carry, if generated in the previous operation
mov r7,a  // msb in r7
mov 51h, a

end  // end of program

////////////////// 16 bit subtraction

// perform subtraction of two 16 bit numbers 2295h and 1234h
// store the values in r0 ~ r3
// the result in r7 and r6
// store the result in 51h and 50h (optional operation)
// expected output= 1061h


org 00h

mov ro, #095h
mov r1, #22h
mov r2, #34h
mov r3, #12h
clrc // clear carry

mov a,ro   // LSB subtraction
subb a,r2
mov r5,a
mov 50h,a

mov r1,a  // MSB subtraction
subb a,r3
mov r6,a
mov 51h,a
```

```
end  //end of program


// 8 bit multiplication
mov a, #25h
mov b, #65h
mul ab   ;  result 0e99h   b=09h, a= 99h



////////////// 16 bit multiplication
// perform multiplication of two 16 bit numbers FFFFh and EDFDh
// result stored in 23,22,21,20 h registers
//Expected result EF9C1063
// proceedure
// srep 1 : LSB1*LSB2
//Step2: LSB1*MSB1
//step3: MSB1* LSB2
//step 4: MSB1*MSB2


org 00h

mov r1, #0EDh ; MSB1
mov r2, #0FFh ; MSB 2
mov r3, #0FDh ; LSB 1
mov r4, #0FFh ;LSB2


mov a,r3  // LSB2*LSB1
mov b,r4
mul ab
mov 20h,a //result-LSB  stored in 20h
mov 21h,b

mov a,r3 //MSB2*LSB1
mov b, r2
mul ab
mov 22h,b
addc a,21h  //previous result MSB
mov 21h,a

mov a,r1 // MSB1*LSB2
mov b,r4
mul ab
addc a,21h
mov 21h,a  // LSB2 stored

mov a,b  // previous result MSB
addc a,22h
mov 22h,a
mov a,r1
mov b,r2
```

```
mul ab
addc a,22h
mov 22h,a  //MSB 1 stored
mov a,b
addc a,#00h
mov 23h,a // MSB 2 stored

end
```

```
// ALP for ASCII to packed BCD
// uppacked BCD  upper nibble =0, 0<Lower nibble<10
// packed BCD  0<  upper nibble<10,  and 0<Lower nibble<10
// noraml ASCII range 0~127, Extended ASCII range 0~266
// two send 32 two key strokes(bytes) 3 = 33h and 2 = 32h required. with
packed BCD one byte 32h is required.
//upper nibble 3 and lower nibble 2. So data is packed.
// This program converts two keys '3' and '2' to packed BCD

org 00h

  mov a,#33h   ;  key '3'
  anl a,#0fh ; mask upper nibble
  swap a ; nibbles are swaped  a=30h

  mov b,a; b=30h

  mov a,#32h   ;  key '2'
  anl a,#0fh ; mask upper nibble   a=02h

  orl a,b ; or operation of a & b     A=A OR B      = 32h packed decimal
  mov r1,a

 end
```

```
////////////////

// ALP for  packed BCD to ASCII
// two bcd numbers 29h
// store the data in r1 and r2
// expected out put is two ascii numbers r1=32h and  r2= 39h

org 00h
  mov a, #29h
  mov r0,a
  anl a,#0fh
  orl a, #30h ;  a=09
  mov r1,a ;     r1=39h  first character

  mov a,r0
```

```
  swap a ;       upper and lower nibbles get swapped
  anl a,#0fh ;   a=2
  orl a, #30h
  mov r2,a;       r2=32h  second character

  end
//////////////////////

// ALP for  unpacked BCD to ASCII
// two unpackedbcd numbers 2h , 9h
// store the output data in r1 and r2
// expected out put is two ascii numbers r1=32h and  r2= 39h

org 00h

  mov a,#2h
  swap a
  orl a,#9h;      a= 29h

  mov r0,a
  anl a,#0fh ;    a=09
  orl a, #30h
  mov r1,a ;      r1=39h  first character

  mov a,r0
  swap a ;      upper and lower nibbles get swapped
  anl a,#0fh
  orl a, #30h
  mov r2,a;      r2=32h  first character

  end
//////////////////////




// Program 1
// ALP for Reversal of a given string without null character

org 00h

        mov dptr,#mydata
        mov r1,#0eh  // number of bytes to read
      mov r0 ,#4eh // where to store
back: clr a
      movc a, @a+dptr
      mov @r0,a
      dec r0  // store in reverse order
      inc dptr
      djnz r1,back

here: nop // no operation give one machine clock delay
      sjmp here  // infinite loop
```

```
org 300h

mydata: db "ECE department"  //define byte
     end


///////////////
// Program 2
// ALP for Reversal of a given string with null character
//
org 00h

        mov dptr,#mydata
           mov r0 ,#4eh // where to store
back: clr a
      movc a, @a+dptr
      jz here  // if a=0 , end of data
      mov @r0,a
      dec r0  // store in reverse order
      inc dptr
      sjmp back

here: sjmp here  // infinite loop


org 300h

mydata: db "ECE department",0 //define byte  End of data is  "0"

       end

///////////////
// Program 3
// ALP for finding a character in a given string
//

org 00h

        mov dptr,#mydata
           mov r0 ,#4eh // where to store
back: clr a
      movc a, @a+dptr
      mov @r0,a
      acall count  // absolute call within 2KB
      jz here
      dec r0  // store in reverse order
      inc dptr
        sjmp back

here: sjmp here  // infinite loop


org 300h
```

```
mydata: db "ECE department",0 //define byte  End of data is  "0"


org 400h

count: cjne  a,#"E", cnt // compare a with "E"
       inc r1
cnt:   ret

end

///////////////
// Program 4
// ALP for sorting of numbers in ascending order
//

org 00h

         mov r4,#05h

again:   mov r3, #04h
         mov r0, #30h
         clr c
ascnd:   mov a, @r0
         mov r1,a
         inc r0
         mov a,@r0
         subb a,r1   // compare two consecutive numbers
         jnc skip
         mov a,@r0
         dec r0
         mov @r0,a
         mov a,r1
         inc r0
         mov @r0,a

skip:    djnz r3, ascnd
         djnz r4, again

         end

///////////////
```

```
// Program 1
// ALP for checksum calculation
// inetl hex formar and motorola hex format use check sum on 16 byte data
```

```
// procedure: add all the bytes, neglect the carry if any. Perform the
Two's complement of the reult
// expeted output for the data : "Test Data Chksum"  --> 3B

org 00h

        mov dptr,#mydata
        mov r1,#10h  // 16 bytes or 32 bytes for checksum

back: clr a
      movc a, @a+dptr
      add a,b     // 8 bit addition , carry ?
        mov b,a
      inc dptr
      djnz r1,back
      mov a,b
      cpl a  // complement on accumulator is not allowed.
      add a,#1
      mov r0,a

org 100h

mydata: db "Test Data Chksum"  //define byte  data in hex:  54 65 73 74
20 44 61 74 61 20 43 68 6b 73 75 6d
     end


//////////////
```

```
// ASM for stepper motor operation
// connect first four pins of p0 to stepper motor windings


org 00
      mov p0,#00h // port 0 as output

start:mov a,#66h
      mov p0,a  // drive p0

// clockwise operation 2540 steps

clkw:  mov r0,#10
clkw1: mov r1,#254
clkw2: rr a          // rotate right
       acall delay
       mov p0,a
       djnz r1,clkw2
       djnz r0,clkw1

// anti clockwise operation2540 steps
```

```
aclkw:  mov r0,#10
aclkw1: mov r1,#254
aclkw2: rl a         // rotata left
        acall delay
        mov p0,a
        djnz r1,aclkw2
      djnz r0,aclkw1
      sjmp start

// delay subroutine

delay: mov r2,#100
here1: mov r3,#100
here2:djnz r3,here2
      djnz r2,here1
      ret

     end
```

```
// ASM to drive common anode seven segment display
// gfecdab seven outputs
// Common anode alpha numeric lookup table
// Numbers: 0-0c0h, 1-0f9h, 2-0a4h, 3- 0b0h, 4-099h, 5-092h, 6-082h, 7-
0f8h, 8-080h, 9-090h
// alphabets: A-04h, b-03h, C-46h, d-21h, E-06h, F-0Dh, H-09h, I-47h, J-
71h, L-47h, n-2bh
//          : o-23h, P-0ch, r-2fh, S-12, U-42h, y-11, Z-28h
//////////

org 00h

       mov p0,00h

start: mov dptr,#mydata
         mov r3,#21  // number of elements in array
         mov p0,#00  // make all the leds glow

loop1:     clr a
      movc a,@a+dptr
      acall delay
      inc dptr
      mov p0,a
      djnz r3,loop1

      sjmp start // forever loop


// delay sub routine
```

```
delay:  mov r2,#04
here: mov r0,#250
here1:      mov r1,#250
here2:      nop
      nop
      djnz r1,here2
      djnz r0,here1
      djnz r2,here
      ret

org 100h
mydata: db 0c0h,0f9h,0a4h,0b0h, 099h, 092h, 082h, 0f8h, 080h, 090h,  04h,
03h, 46h, 21h, 06h, 0Dh, 09h, 47h, 71h, 47h, 2bh

   end
```