1. Largest.

org 000h

mov 42h,#06h

mov 43h,#02h

mov 44h,#03h

mov 45h,#04h

mov 46h,#05h

mov b,42h

mov a,b

mov r3,#00

mov r0,#43h

subt:subb a,@r0

mov 48h,a

jnc next

mov a,@r0

inc r0

inc r3

cjne r3,#04,subt

mov r4,a

sjmp stop

next:mov a,b

inc r0

inc r3

cjne r3,#04,subt

mov r4,a

stop:nop

end

1. prime.

org 000h

mov a,#0

mov r0,a

mov r1,a

dec r0

loop1:mov b,r0

mov a,r1

div ab

mov r2,b

cjne r2,#00,loop2

clr c //clear carry if not prime

sjmp stop

loop2:dec r0

cjne r0,#01,loop1

setb c//set carry flag if prime

stop:nop

end

1. ascending.

org 000h

mov 32h,#01

mov 33h,#02

mov 34h,#03

mov 35h,#04

mov 36h,#05

loop4:mov r2,#04 //counter

mov r3,#04 //counter

mov r0,#32h

loop3:mov b,@r0

inc r0

mov a,@r0

cjne a,b,loop1 //(a<b c=1; a>b c=0

// if u put jnc instead of jc in the preceding step

//u will end up with ascending order.

loop1:jc loop2

mov @r0,b

dec r0

mov @r0,a

inc r0

djnz r3,loop3

loop2:djnz r3,loop3

djnz r2,loop4

sjmp stop

stop:nop

end

1. ncr,

org 000h

mov b,#05 //value of n=5

mov a,b

subb a,#01

mov r0,a

abcd:mov a,r0

mul ab

mov b,a

djnz r0,abcd

mov r2,a//r2 has n-factorial stored.

// instead of puttting 5 in B initially put it in A , it reduces the process.

mov a,#05 //value of n=5

mov r0,#02 //value of r=2

subb a,r0

mov b,a

subb a,#01

xch a,b

mov r1,b

efg:mov b,r1

mul ab

djnz r1,efg

mov r3,a //r3 had (n-r)factorial stored.

mov a,#02 //value of r=2

mov b,a

subb a,#01

xch a,b

mov r1,b

hig:mov b,r1

mul ab

djnz r1,hig

mov r4,a //r4 has r-factorial stored.

mov a,r3

mov b,r4

mul ab//result/denominator is in A and B has 00

mov b,a//the result/denominator is in b

mov a,r2

div ab //result is in a

mov r5,a //final result is in r5

end

1. power,

org 000h

mov a,#03

mov r3,#00

abcd:mov b,#03

mult:mul ab

mov 32h,a

inc r3

cjne r3,#02,abcd

sjmp stop

stop:nop

end

1. smallest

org 000h

mov 42h,#08h

mov 43h,#02h

mov 44h,#03h

mov 45h,#04h

mov 46h,#05h

mov r3,#00

mov r0,#43h

mov b,42h

abcd:mov a,b

subt:subb a,@r0

mov 48h,a

jnc next

//mov a,@r0

inc r0

inc r3

cjne r3,#04,abcd

mov r4,b

sjmp stop

next:mov b,@r0

inc r0

inc r3

cjne r3,#04,

mov r4,b

stop:nop

end

1. factorial

mov a,#04

mov r1,a

subb a,#01

mov r0,a

mov a,r1

loop1:mov b,r0

mul ab//result is in A and B has 00

djnz r0,loop1

stop:nop

end

1. find 1s

org 000h

mov a,#21h

mov r3,#08

back:rrc a

jc next

inc r5

djnz r3,back

sjmp stop

next:inc r4

djnz r3,back

stop:nop

end

1. binary to gray

/\*org 000h

mov a,#05

mov r0,a

rrc a

mov r1,a

mov a,r0

xrl a,r1

mov r2,a

end //single binary number to gray

\*/

1. array of binary to gray

org 000h

mov 31h,#01

mov 32h,#02

mov 33h,#03

mov 34h,#04

mov 35h,#05

//mov r2,#05//counter

mov r1,#41h

mov r0,31h

loop1:clr c

mov a,r0

//mov r0,a

rrc a

mov r2,a

mov a,r0

xrl a,r2

mov @r1,a

inc r1

inc r0

cjne r0,#06,loop1

end

1. sum of even/odd terms In an array.

org 000h

mov 31h,#01

mov 32h,#02

mov 33h,#03

mov 34h,#04

mov 35h,#05

mov r2,#05

mov r3,#00

mov r0,31h

loop2:mov a,r0

rrc a

jc loop1 ;if you put JNC instead of JC you'll get sum of even numbers.

inc r0

djnz r2,loop2

sjmp stop

loop1:mov a,r0

add a,r3

mov r3,a

inc r0

djnz r2,loop2

stop:nop

end

1. sum of +ve/-ve numbers.

org 000h

mov 31h,#01

mov 32h,#02

mov 33h,#03

mov 34h,#04

mov 35h,#05

mov r2,#05

mov r3,#00

mov r0,31h

loop2:mov a,r0

rlc a

jc loop1 ;if u put JNC instead of JC u'll get sum of positive numbers.

inc r0

djnz r2,loop2

sjmp stop

loop1:mov a,r0

add a,r3

mov r3,a

inc r0

djnz r2,loop2

stop:nop

end

1. lcm ang gcd

org 000h

mov 30h,#03

mov 31h,#03

mov r2,#00

mov a,30h

mov b,31h

cjne a,b,loop1 ;compare and check if A and B are equal.

mov 32h,a //lcm stoed in 33h

sjmp hcf

loop1:mov a,30h ;smaller no

mov b,31h ;bigger no

subb a,b

jnc loop4 ;if no carry(a>b)

;if carry (b>a)

loop3:mov r0,31h ;bigger no

mov r1,30h ;smaller no

mov a,r0 ; A has smaller no

inc r2 //counter is increased

mov b,r2 ;counter is shifted to B

mul ab ;multiply smaller number \* counter

;result is stored in A.

mov r4,a

mov b,r0 ; bigger no is stored in B

div ab ;result / bigger no

mov r3,b ;remainder is stored in R3 from B

cjne r3,#00,loop1

mov 32h,r4 //lcm stored in 33h

sjmp hcf

;if no carry (a>b)

loop4:inc r2

mov r0,30h//bigger

mov r1,31h//smaller

mov a,r1

mov b,r2

mul ab

mov r4,a

mov b,r0

div ab

mov r3,b

cjne r3,#00,loop1

mov 32h,r4 //lcm stored in 33h

sjmp hcf

//hcf

hcf:mov a,31h

mov b,30h

mul ab

mov b,32h

div ab

mov 33h,a //hcf stored in 33h

sjmp stop

stop:nop

end

1. conditional lcm and hcf

org 000h

mov r5,#'l'

mov r6,#'g'

mov r7,#'g'

mov 30h,#03

mov 31h,#06

mov r2,#00

mov a,30h

mov b,31h

cjne a,b,loop1 ;compare and check if A and B are equal.

mov 32h,a //lcm stoed in 33h

mov a,r7

cjne a,#'g',stop

sjmp hcf

loop1:mov a,30h ;smaller no

mov b,31h ;bigger no

subb a,b

jnc loop4 ;if no carry(a>b)

;if carry (b>a)

loop3:mov r0,31h ;bigger no

mov r1,30h ;smaller no

mov a,r1 ; A has smaller no

inc r2 //counter is increased

mov b,r2 ;counter is shifted to B

mul ab ;multiply smaller number \* counter

;result is stored in A.

mov r4,a

mov b,r0 ; bigger no is stored in B

div ab ;result / bigger no

mov r3,b ;remainder is stored in R3 from B

cjne r3,#00,loop1

mov 32h,r4 //lcm stored in 33h

mov a,r7

cjne a,#'g',stop

sjmp hcf

;if no carry (a>b)

loop4:inc r2

mov r0,30h//bigger

mov r1,31h//smaller

mov a,r1

mov b,r2

mul ab

mov r4,a

mov b,r0

div ab

mov r3,b

cjne r3,#00,loop1

mov 32h,r4 //lcm stored in 33h

mov a,r7

cjne a,#'g',stop

sjmp hcf

//hcf

hcf:mov a,31h

mov b,30h

mul ab

mov b,32h

div ab

mov 33h,a //hcf stored in 33h

sjmp stop

stop:nop

end

1. string equality

org 000h

mov r3,#00 //counter

mov r2,#00

ljmp main1

org 200h

mydata1:db "abcd",0

org 300h

mydata2:db "abcd",0

main1:mov dptr,#200h

mov a,#01 //calculation of dptr , cause while running loops,dptr ahs to increase

mov b,r3 //counter, according to number of loops completed.

mul ab //number by which dptr has to increase is stored in a.

mov r4,a

sjmp loop6

loop6:jz loop2 //when value of A is 0,the value pointed by increased dptr is the value used for calculation.hence jumps to actual comparision.

dec a

sjmp loop7

loop7:inc dptr //dptr is increased according to number of loops.

sjmp loop6

loop2:clr a //a has zero.

movc a,@a+dptr

jz stop

mov r0,a

sjmp main2

main2:mov dptr,#300h

mov a,#01 //calculation of dptr , cause while running loops,dptr ahs to increase.

mov b,r3 //counter, according to number of loops completed.

mul ab //number by which dptr has to increase is stored in a.

mov r4,a

sjmp loop3

loop3:jz loop5 //when value of A is 0,the value pointed by increased dptr is the value used for calculation.hence jumps to actual comparision.

dec a

sjmp loop4

loop4:inc dptr //dptr is increased according to number of loops.

sjmp loop3

loop5:clr a

movc a,@a+dptr

jz stop

mov 25h,a

mov a,r0

cjne a,25h,loop1

inc r3 //updation of no of loops.(helps to increase dptr accordingly).

sjmp main1

loop1:mov 30h,#01// put 01 in 30h , if strings are not equal.

sjmp stop

stop:nop

end

1. stack and data memory

org 000h

ljmp main

org 200h

main:mov r2,#10h

mov sp,#30h

mov r7,#05h

abcd:mov r3,02

push 03

inc r2

djnz r7,abcd

end

1. odd sum average

ORG 000H

mov 75h,#08 // counter

mov r0,#40h //memory of 1st element

mov 30h,#09 //data-1st

abcd:mov @r0,30h

inc r0

inc 30h

djnz 75h,abcd

mov r0,40h

mov r5,#08

mov r3,#00

mov r4,#00

loop1:mov a,r0

rrc a

//if carty is set, the number is odd

jc odd

inc r0

djnz r5,loop1

sjmp avg

odd:mov a,r0

add a,r3

mov r3,a //move answer to r3

inc r0

inc r4

djnz r5,loop1

sjmp avg

avg:mov a,r3

mov b,r4

div ab

mov r6,a//final answer in r6

end

1. search key and update index in a stack

org 00h

mov 75h,#05h;array size

mov r0,#30h;starting address

mov 30h,#2h

mov 31h,#2h

mov 32h,#76h

mov 33h,#2h

mov 34h,#2h

mov r4,75h;index holder

mov a,75h

addc a,r0

dec a

mov r0,a

compare:cjne @r0,#76h,next;here the immediate no is the key

sjmp ok

ok:push 4

inc r5

next:dec r0

djnz r4,compare

//cjne @r0,#2h,go

//sjmp found

sjmp go

found:push 4

inc r5

go:nop

end

easier…

* org 00h

mov r0,#30h;starting address

//data loading to memory locations

mov 30h,#21h

mov 31h,#22h

mov 32h,#76h

mov 33h,#55h

mov 34h,#21h

//end of data loading

//searching loop

mov r2,#1;index counter

compare:cjne @r0,#78h,next;here the immediate no is the key

inc r3;r3 is the counter that counts no. of occurences

push 2

next:inc r2

inc r0

cjne r2,#06,compare

end

* org 00h

mov r0,#30h;starting address

mov 30h,#2h

mov 31h,#2h

mov 32h,#76h

mov 33h,#2h

mov 34h,#2h

mov r2,#0

compare:cjne @r0,#2h,next;here the immediate no is the key

inc r0

inc r2

sjmp ok

ok:push 2

cjne r2,#5,sj

sjmp go

sj:sjmp compare

next:inc r0

inc r2

cjne r2,#5,compare

go:nop

end

1. add char in bw word of string

/\*NOTE: the string is stored in ROM so u cant manipulate it directly,

u need to first put each element into internal memory and then

while space' ' comes add \* instead of space \*/

org 0000h

mov r1,#30h//from 30h string is stored after moving.

//mov r2,#00h

LJMP main

org 200h

string: db 'kle tech hubballi',0

main:

mov dptr,#200h

mov B,#' '

back:clr a

movc a,@a+dptr

cjne a,#' ',abcd

mov @r1,#'\*'

inc r1

sjmp do

abcd:mov @r1,a

inc r1

do:inc dptr

cjne a,#00h,back

last:NOP

END