**Logical group of instructions related programs**

**;ARM ALP to demonstrate different logic operations**

area reset,data,readonly

export \_\_Vectors

\_\_Vectors

dcd 0

dcd Reset\_Handler

area mycode,code,readonly

entry

export Reset\_Handler

Reset\_Handler

**;setting particular bits without disturbing other bits**

mov r0,#0x0000

mov r1,#0x1001

orr r0,r1

**;clearing/masking particular bits without disturbing other bits**

mov r2,#0x1111

mov r3,#0x1101

and r2,r3

**;reverse of clearing/masking particular bits without disturbing other bits**

mov r4,#0x1111

mov r5,#0x1101

bic r4,r5

**;toggling of particular bits without disturbing other bits**

mov r6,#0x1001

mov r7,#0x1101

eor r6,r7

stop b stop

end

**;ARM ALP to find number of zeros and ones**

area reset,data,readonly

export \_\_Vectors

\_\_Vectors

dcd 0

dcd Reset\_Handler

area mycode,code,readonly

entry

export Reset\_Handler

Reset\_Handler

mov r0,#0 ; no. of zeros

mov r1,#0 ; no. of ones

mov r2,#2 ; no. of words

ldr r3,=value

nword mov r4,#32 ; no. of bits to be checked

ldr r5,[r3],#4

cont movs r5,r5,ror#1

bcs ones

add r0,r0,#1

b nxt

ones add r1,r1,#1

nxt subs r4,r4,#1

bne cont

subs r2,r2,#1

cmp r2,#0

bne nword

stop b stop

value dcd 0x55555555,0x55555555

end

**;ARM ALP to find number of odd and even numbers**

area reset,data,readonly

export \_\_Vectors

\_\_Vectors

dcd 0

dcd Reset\_Handler

area mycode,code,readonly

entry

export Reset\_Handler

Reset\_Handler

mov r0,#0 ; no. of odd nos.

mov r1,#0 ; no. of even nos

mov r2,#7 ; no. of words

ldr r3,=value

cont ldr r4,[r3],#4

ands r4,r4,#1<<0 ; check LSB bit, 0 = even; 1=odd

bhi odd

add r1,r1,#1

b nxt

odd add r0,r0,#1

nxt subs r2,r2,#1

cmp r2,#0

bne cont

stop b stop

value dcd 0x12345678

dcd 0x80489861

dcd 0x11111110

dcd 0x33333333

dcd 0xe6055462

dcd 0xaaaaaaa6

dcd 0x99999994

end

**;ARM ALP to find number of negative and positive numbers**

area reset,data,readonly

export \_\_Vectors

\_\_Vectors

dcd 0

dcd Reset\_Handler

area mycode,code,readonly

entry

export Reset\_Handler

Reset\_Handler

mov r0,#0 ; no. of negative nos.

mov r1,#0 ; no. of positive nos

mov r2,#7 ; no. of words

ldr r3,=value

cont ldr r4,[r3],#4

ands r4,r4,#1<<31; check MSB bit,if 1=negative;0=positive

bhi negative

add r1,r1,#1

b nxt

negative add r0,r0,#1

nxt subs r2,r2,#1

cmp r2,#0

bne cont

stop b stop

value dcd 0x12345678

dcd 0x80489867

dcd 0x11111111

dcd 0x33333333

dcd 0xe605546c

dcd 0xaaaaaaaa

dcd 0x99999999

end

**;ARM ALP to separate odd and even numbers**

area reset,data,readonly

export \_\_Vectors

\_\_Vectors

dcd 0

dcd Reset\_Handler

area mycode,code,readonly

entry

export Reset\_Handler

Reset\_Handler

mov r0,#10 ; no. of datas

ldr r1,=nums

ldr r2,=even

ldr r3,=odds

nxt1 ldrb r4,[r1],#1

ands r5,r4,#1<<0; check LSB bit, 1=odd;0=even

bhi odd

strb r4,[r2],#1

b nxt

odd strb r4,[r3],#1

nxt subs r0,r0,#1

cmp r0,#0

bne nxt1

stop b stop

nums dcb 0,1,2,3,4,5,6,7,8,9

area mydata1,data,readwrite

even dcb 0

area mydata2,data,readwrite

odds dcb 0

end

**;ARM ALP to separate positive and negative numbers**

area reset,data,readonly

export \_\_Vectors

\_\_Vectors

dcd 0

dcd Reset\_Handler

area mycode,code,readonly

entry

export Reset\_Handler

Reset\_Handler

mov r0,#5 ; no. of words

ldr r1,=nums

ldr r2,=pos

ldr r3,=negs

nxt1 ldr r4,[r1],#4

ands r5,r4,#1<<31;check MSB bit,if 0=positive;1=negative

bhi neg

rev r6,r4

str r6,[r2],#4

b nxt

neg rev r6,r4

str r6,[r3],#4

nxt subs r0,r0,#1

cmp r0,#0

bne nxt1

stop b stop

nums dcd 0x90000000

dcd 0x10000000

dcd 0x80000000

dcd 0xabcdef90

dcd 0x20000000

area mydata1,data,readwrite

pos dcd 0

area mydata2,data,readwrite

negs dcd 0

end

**;ARM ALP to check the number is 2 out of 5 code or not**

area reset,data,readonly

export \_\_Vectors

\_\_Vectors

dcd 0

dcd Reset\_Handler

area mycode,code,readonly

entry

export Reset\_Handler

Reset\_Handler

; check 1st condition i.e. first 3 MSB bits Zeros or not

ldr r0,num

ands r0,#0xe0; 0001 0010 & 1110 0000

cmp r0,#0

bne invalid

; check 2nd condition i.e. first 5 LSB bits contains 2 no. of ones

mov r1,#5;first 5 LSB bits to be checked

mov r2,#0; no. of ones

ldr r3,num

repeat movs r3,r3,ror#1

bcc nxt

add r2,r2,#1

nxt subs r1,r1,#1

cmp r1,#0

bne repeat

cmp r2,#2

bne invalid

mov r12,#0xffff ; 2 out of 5 code

b stop

invalid mov r12,#0x0000 ; not 2 out of 5 code

stop b stop

num dcb 0x12 ; no. to be checked

end

**Logic : 1st condition -> First 3 MSB bits must be zero**

**2nd condition-> First 5 LSB bits must contain 2 numbers of ones**

**Ex: 0x12 -> 0001 0010 ----2 out 0f 5 code**

**0x13 -> 0001 0011-----not 2 out of 5 code**