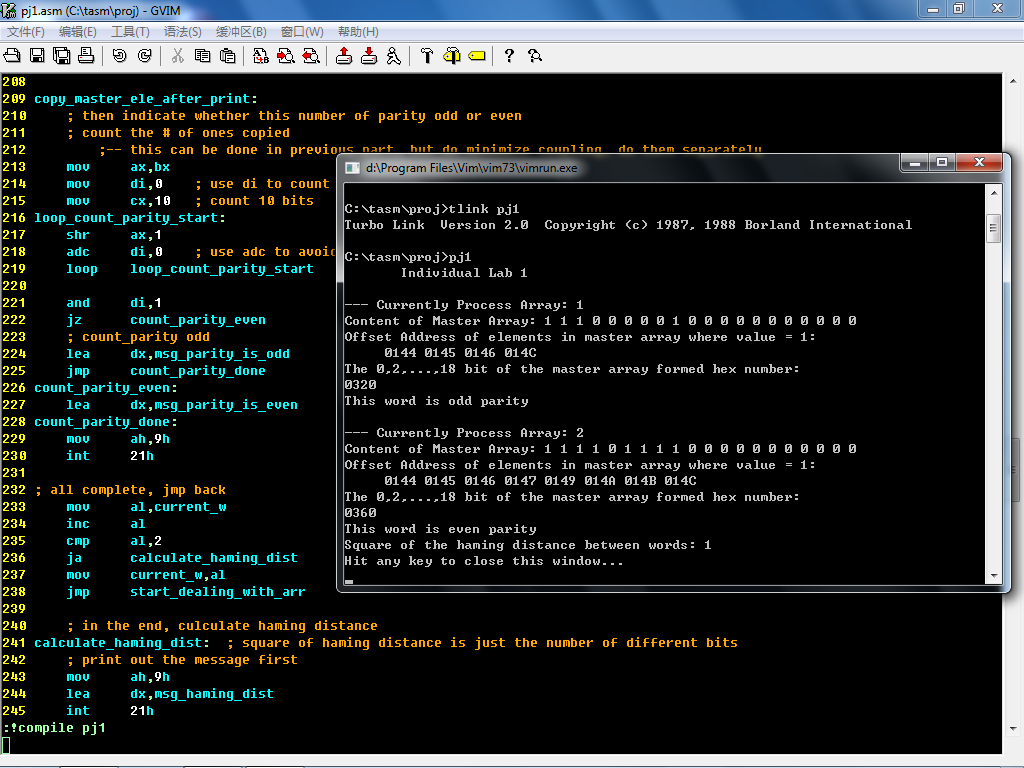
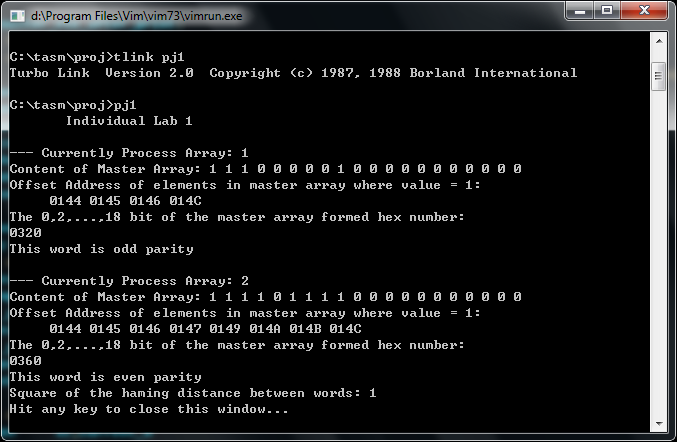
Name: Song Yangyu

Matric Number: A0077863N

Working Environment: VIM + commend line



Solely the screen (output):



## How to test the code:

For easier testing, I defined the array1 and array2 as follows:

arr1 db 1,3,4,6,8,10,15

arr2 db 1,2,3,5,8,9,10,11,12,13,14,15,16,17,18,2

Compile and run the code, then I can get the output as follow:

## Output:

Individual Lab 1 -- Song Yangyu -- A0077863N

--- Currently Process Array: 1

Content of Master Array: 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0

Offset Address of elements in master array where value = 1:

0161 0163 0164 0166 0168

The 0,2,...,18 bit of the master array formed hex number:

0300

This word is even parity

--- Currently Process Array: 2

Content of Master Array: 1 1 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0

Offset Address of elements in master array where value = 1:

0161 0162 0163 0165 0168 0169

The 0,2,...,18 bit of the master array formed hex number:

03A0

This word is even parity

Square of the hamming distance between words: 5

## Basic Logic of the code:

1. Meaning of the variables:
   1. I used “msg\_\*” to store the message showing to the user. Following this naming standard is a good practice of software engineering.
   2. I used “M\_SIZE”, “A1\_SIZE”, and “A2\_SIZE” to define the size of the array; later if one want to change the size of array, one can easily do so by changing the defined variable here.
   3. “word1”,”word2” are the 2 word formed by the two words
   4. current\_w: Because we do all the tasks to array1 and array2 separately, it’s the beat if we can reuse the repeated code. The variable current\_w is for storing which array we’re currently handling: either array1, or array 2. Later in the code when I need to differentiate, I can do so by refer to this variable.
   5. param\_list: I used procedure to print number. This param\_list is supposed to be called by all procedure who want to print number
2. The procedure proc\_print\_num:

I use this procedure to print out number in hex form. It takes three parameters:

* Number to print
* Number of digits to display: it would display the left most (MSB) that number of digits
* Separator: the letter to print after printing the number. This is useful when one wants to print out a list of numbers

Note that one can also use stack to pass the parameters. By doing so, we can actually allow recursion. But in this piece of code, for clarity I used parameter table instead (Note that the con of using a parameter table is that. one can only call one level of function; if one call another function inside the procedure, the parameter would be overwritten)

## Code:

NOTE: I suggest to refer to the code.asm for the details of the code.

stk segment stack

db 128 dup(?)

tos label word

stk ends

; data segment

data segment

msg\_welcome db " Individual Lab 1",10,"$"

msg\_current\_processing\_arr db 10,"--- Currently Process Array: ","$"

msg\_prompt\_marr\_content db "Content of Master Array: ","$"

msg\_show\_offset\_addr db "Offset Address of elements in master array where value = 1:",10," ","$"

msg\_show\_copied\_content db "The 0,2,...,18 bit of the master array formed hex number: ","$"

msg\_debug\_pass db " DEBUG: pass here.",10,"$"

msg\_parity\_is\_odd db "This word is odd parity",10,"$"

msg\_parity\_is\_even db "This word is even parity",10,"$"

msg\_haming\_dist db "Square of the hamming distance between words: ","$"

M\_SIZE equ 20

A1\_SIZE equ 7

A2\_SIZE equ 16

PARAM\_L\_SiZE equ 10

master\_arr db M\_SIZE dup(0)

arr1 db 1,3,4,6,8,10,15

arr2 db 1,2,3,5,8,9,10,11,12,13,14,15,16,17,18,20

word1 dw ?

word2 dw ?

current\_w db 1

param\_list dw PARAM\_L\_SiZE dup(?) ; support up to 10 word of parameters

data ends

; code segment

code segment

assume cs:code, ss:stk, ds:data

start:

; jump the read input part -- for testing simply define

; update the ds

mov ax,data

mov ds,ax

; update the ss

mov ax,stk

mov ss,ax

; and update the stack pointer

mov sp,offset tos

; print out msg

mov ah,9h

lea dx,msg\_welcome

int 21h

start\_dealing\_with\_arr:

; init the master array -- set it all 0

mov si,0

loop\_start\_dealing\_with\_arr\_init:

mov master\_arr + si,0

inc si

cmp si,M\_SIZE

jb loop\_start\_dealing\_with\_arr\_init

; then the welcome message for array specific

mov ah,9h

lea dx,msg\_current\_processing\_arr

int 21h

; finish the welcome str

mov ah,2 ; put array num

mov dl,current\_w

add dl,'0'

int 21h

mov ah,2 ; then print a new line

mov dl,10

int 21h

; fill in master array using appropriate

; this part to load appropriate address into bx, and size into al

; bx - offset address of the corresponding array

; al - size of array

cmp current\_w,1

je loop\_start\_copying\_arr1

cmp current\_w,2

je loop\_start\_copying\_arr2

loop\_start\_copying\_arr1:

mov bx,offset arr1

mov ax,A1\_SIZE

jmp loop\_start\_copying\_end\_jmp

loop\_start\_copying\_arr2:

mov bx,offset arr2

mov ax,A2\_SIZE

jmp loop\_start\_copying\_end\_jmp ; can ignore this jump, but for systematic, put it

loop\_start\_copying\_end\_jmp:

mov si,0

loop\_start\_copying:

mov cl,byte ptr [bx] + si ; get the element in array

cmp cl,10

jae after\_assigning

mov ch,0

mov di,cx

dec di ; need to decrease element cuz the first index is 1

mov master\_arr + di,1 ; copy number 1 to master[di]

after\_assigning:

inc si

cmp si,ax ; al stores the size of array

jb loop\_start\_copying ; if there's no loop

; display the content of the master array

mov ah,9h

lea dx,msg\_prompt\_marr\_content

int 21h

mov si,0

loop\_show\_marr\_content:

cmp byte ptr [offset master\_arr] + si,1

jb cmp\_assign\_ascii0

mov dl,'1' ; didn't jump, so content is 1

jmp cmp\_assign\_complete

cmp\_assign\_ascii0:

mov dl,'0'

cmp\_assign\_complete:

mov ah,02 ; then print out the res

int 21h

inc si

mov dl,' ' ; use a space to separate

mov ah,02

int 21h

cmp si,M\_SIZE

jb loop\_show\_marr\_content

; print a new line char to complete this part

mov dl,10

mov ah,02

int 21h

; display the offset address of the maste array that are 1s

; show message first

mov ah,9h

lea dx,msg\_show\_offset\_addr

int 21h

mov cx,0

loop\_disp\_marr\_offset\_content\_1:

mov bx,offset master\_arr

add bx,cx

cmp [bx],byte ptr 1

jb loop\_disp\_marr\_offset\_done\_print

; then call the print method to print addrss

mov param\_list,bx

mov param\_list[2],4 ; print 4 digits

mov byte ptr param\_list[4],' ' ; separate by space char

call proc\_print\_num

loop\_disp\_marr\_offset\_done\_print:

inc cx

cmp cx,M\_SIZE

jb loop\_disp\_marr\_offset\_content\_1

; in the end, print a new line

mov ah,2h

mov dl,10

int 21h

; form a word from the master array

; print out message first

mov ah,9h

lea dx,msg\_show\_copied\_content

int 21h

;-- copy the content to bl

mov cx,10

mov si,0

mov bx,0

loop\_copy\_master\_ele\_start:

shl bx,1

cmp byte ptr [offset master\_arr] + si,1 ; need to copy form MSB to LSB, cuz using shift

jb copy\_master\_ele\_on\_0

or bx,1 ; set the bl bit to be 1

copy\_master\_ele\_on\_0: ; no need to copy actually, cuz it's 0 by default

add si,2

loop loop\_copy\_master\_ele\_start ; jump on no sign -- when cx has sign, means < 0

; then print out a new line afterwards

mov ah,2

mov dl,10

int 21h

; then print out the num

mov param\_list,bx

mov param\_list[2],4 ; display 4 digits, cuz it's a 10 bit number

mov byte ptr param\_list[4],10 ; print a new line

call proc\_print\_num

; save the word

cmp current\_w,1

jmp save\_word\_1\_jmp

cmp current\_w,2

jmp save\_word\_2\_jmp

save\_word\_1\_jmp:

mov word1,bx

jmp end\_save\_word\_jmp

save\_word\_2\_jmp:

mov word2,bx

jmp end\_save\_word\_jmp

end\_save\_word\_jmp:

copy\_master\_ele\_after\_print:

; then indicate whether this number of parity odd or even

; count the # of ones copied

;-- this can be done in previous part, but do minimize coupling, do them separately

mov ax,bx

mov di,0 ; use di to count

mov cx,10 ; count 10 bits

loop\_count\_parity\_start:

shr ax,1

adc di,0 ; use adc to avoid jump -- therefore increase efficiency

loop loop\_count\_parity\_start

and di,1

jz count\_parity\_even

; count\_parity odd

lea dx,msg\_parity\_is\_odd

jmp count\_parity\_done

count\_parity\_even:

lea dx,msg\_parity\_is\_even

count\_parity\_done:

mov ah,9h

int 21h

; all complete, jmp back

mov al,current\_w

inc al

cmp al,2

ja calculate\_haming\_dist

mov current\_w,al

jmp start\_dealing\_with\_arr

; in the end, culculate haming distance

calculate\_haming\_dist: ; square of haming distance is just the number of different bits

; print out the message first

mov ah,9h

lea dx,msg\_haming\_dist

int 21h

mov ax,word1

xor ax,word2

mov dl,1 ; use dl to count

mov cx,10

count\_bits\_haming\_dist:

shr ax,1

adc dl,0

loop count\_bits\_haming\_dist

; 1 bit digit, can print out directly -- cuz certainly less than 10 bits of difference

add dl,'0'

mov ah,2h

int 21h

; then print out a new line char

mov dl,10

mov ah,2h

int 21h

end\_proc:

mov ah,4ch

mov al,0h ; the return value

int 21h

; use this procedure to print a number

; use the param\_list to pass parameter:

; the following elements are stored in the stack:

; number to print -- param\_list

; # of digits to display -- param\_list[2]

; separator -- param\_list[4]

; caller is in charge of saving and restoring the bp

proc\_print\_num proc

push bp

mov bp,sp

push ax ; save register

push bx

push cx

push dx

push si

push di

; read the papameter list from the parameter table

mov bx,param\_list

mov si,param\_list[2]

print\_loop:

mov dl,bh

; value to print, dl cuz calling int 21,2

mov cl,4

shr dl,cl ; leave the 4 higher bit only

cmp dl,10

jb add\_to\_char

add dl,'A'-'0' - 10

add\_to\_char:

add dl,'0'

mov ah,02

int 21h ; print one character

; update the bx

mov cl,4 ; 4 bit to shift each time

shl bx,cl

dec si

jnz print\_loop

; then print out the separator -- the char to print after each number

mov dl,byte ptr param\_list[4]

and dl,dl

jz proc\_print\_separator\_done

mov ah,02

int 21h

proc\_print\_separator\_done:

; clean up

pop di

pop si

pop dx

pop cx

pop bx

pop ax

pop bp

; then shift by 2\*3 parameters passed by the caller

ret

proc\_print\_num endp

; the end prec

code ends

end start