

MỤC LỤC

MỤC LỤC.....	1
Chương 4. Data Transfers, Addressing, and Arithmetic	4
Bài 1:	4
Bài 2	5
Bài 3*:.....	6
Bài 4:	7
Bài 5:	7
Chương 5. Procedures	10
Bài 1:	11
Bài 2:	12
Bài 3:	12
Bài 4:	13
Bài 5:	13
Bài 6:	14
Bài 7:	15
Bài 8:	16
Chương 6. Conditional Processing	17
Bài 1:	17
Bài 2:	18
Bài 3:	20

Bài 4:	21
Bài 5:	22
Bài 6:	24
Chương 7. Interger Arithmetic.....	26
Bài 1:	26
Bài 2:	28
Bài 3:	30
Bài 4:	31
Bài 5:	32
Bài 6:	33
Bài 7:	33
Bài 8:	34
Bài 9:	34
Bài 10:	34
Bài 11:	34
Bài 12:	35
Bài 13:	35
Chương 8: Advanced Procedures	35
Bài 1:	35
Bài 2:	35
Bài 3:	35
Bài 4:	36

Bài 5:	36
Bài 6:	36
Bài 7:	36
Bài 8:	36
Bài 9:	36
Bài 10:	36
Bài 11:	36
Bài 12:	36
Bài 13:	37

Chương 4. Data Transfers, Addressing, and Arithmetic

Bài 1:

.data

Sarray SWORD -1,-2,-3,-4

Viết chương trình copy giá trị của mảng vào thanh ghi EAX, EBX, ECX, EDX.

Gọi hàm DumpRegs

Giải thích các giá trị trong các thanh ghi

Trả lời:

+ Chương trình copy giá trị của mảng vào thanh ghi EAX, EBX, ECX, EDX:

```
.data
Sarray SWORD - 1, -2, -3, -4
.code
main PROC

    MOVZX EAX, [Sarray]           ; EAX = -1
    MOVZX EBX, [Sarray + 2]       ; EBX = -2
    MOVZX ECX, [Sarray + 4]       ; ECX = -3
    MOVZX EDX, [Sarray + 6]       ; EDX = -4

    call DumpRegs

exit
main endp
end main
```

+ Kết quả in ra màn hình:

EAX=FFFFFFFF EBX=FFFFFFFE ECX=FFFFFFFD EDX=FFFFFFFC

+ Giải thích:

Lệnh MOVZX EAX, [Sarray] sao chép 1 SWORD (2 byte) từ vị trí bắt đầu của mảng Sarray và mở rộng lên thành 4 byte vào thanh ghi EAX (4 byte). SWORD đầu tiên của mảng Sarray là -1. Nên EAX có giá trị là -1 = FFFFFFFFh

Lệnh `MOVSX EBX, [Sarray + 2]` sao chép 1 `SWORD` (2 byte) thứ 2 của mảng `Sarray` và mở rộng lên thành 4 byte vào thanh ghi `EBX` (4 byte). `SWORD` thứ 2 của mảng `Sarray` là -2. Nên `EBX` có giá trị là -2 = FFFFFFFEh

Tương tự `ECX` có giá trị là -3 = FFFFFFFDh

Tương tự `EDX` có giá trị là -4 = FFFFFFFCh

Bài 2

Viết chương trình sử dụng vòng lặp để tính toán bảy giá trị đầu tiên của dãy số Fibonacci,

Các giá trị trong thanh ghi `EAX` và hiển thị nó bằng câu lệnh gọi `DumpRegs` bên trong vòng lặp

Trả lời:

+ Chương trình:

```
.data
.code
main PROC

    MOV EAX, 1      ; Số fibonacci đầu tiên
    MOV EBX, 0      ; Lưu số fibonacci trước đó
    MOV ECX, 7      ; Số lần lặp

    LAP:           ; Tính số fibonacci tiếp theo
    call DumpRegs  ; Hiển thị số fibonacci hiện tại trong EAX
    MOV EDX, EAX   ; Lưu số fibonacci hiện tại vào EDX
    ADD EAX, EBX   ; Tính số fibonacci tiếp theo lưu vào EAX
    MOV EBX, EDX   ; Lưu số fibonacci hiện tại đó vào EBX
    LOOP LAP

exit
main endp
end main
```

+ Kết quả của `EAX` in ra màn hình sau 7 lần `call DumpRegs`:

Lần 1: `EAX` = 00000001

Lần 2: `EAX` = 00000001

Lần 3: EAX = 00000002

Lần 4: EAX = 00000003

Lần 5: EAX = 00000005

Lần 6: EAX = 00000008

Lần 7: EAX = 0000000D

Bài 3*:

Sử dụng một vòng lặp với địa chỉ gián tiếp hoặc được lập chỉ mục để đảo ngược các phần tử của một mảng số nguyên tại chỗ.

Không sao chép các phần tử sang bất kỳ mảng nào khác.

Trả lời:

```
.data
array BYTE 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
.code
main PROC
    leng = lengthof array                ; số phần tử của mảng
    MOV EBX, 0                          ; chỉ số đầu
    MOV EDX, leng - 1                  ; Chỉ số cuối
    MOV ECX, leng/2                    ; số lần đổi chỗ
    DOICHO:                             ; Đổi chỗ array[EBX] và array[EDX]
        MOV AL, array[EDX]
        XCHG array[EBX], AL
        XCHG array[EDX], AL
        INC EBX                        ; tăng chỉ số bé
        DEC EDX                        ; tăng chỉ số lớn
    LOOP DOICHO

    MOV ESI, OFFSET array
    MOV ECX, LENGTHOF array
    MOV EBX, TYPE array
    call DumpMem                        ; In mảng array ra màn hình

exit
main endp

end main
```

Kết quả in ra màn hình:

0A 09 08 07 06 05 04 03 02 01

Bài 4:

Viết chương trình thực hiện biểu thức số học sau:

$$\text{EAX} = -\text{val2} + 7 - \text{val3} + \text{val1}$$

Sử dụng các định nghĩa dữ liệu sau:

```
val1 SDWORD 8
val2 SDWORD 15
val3 SDWORD 20
```

Trong các chú thích bên cạnh mỗi lệnh, hãy viết giá trị thập lục phân của EAX.

Chèn lệnh gọi DumpRegs vào cuối chương trình.

Trả lời:

```
.data
    val1 SDWORD 8
    val2 SDWORD 15
    val3 SDWORD 20
.code
main PROC
    MOV EAX, val2      ; EAX = val2 = 15
    NEG EAX            ; EAX = -val2 = -15
    ADD EAX, 7         ; EAX = -val2 + 7 = -8
    SUB EAX, val3      ; EAX = -val2 + 7 - val3 = -28
    ADD EAX, val1      ; EAX = -val2 + 7 - val3 + val1 = -20
    call DumpRegs      ; In các thanh ghi ra màn hình
exit
main endp

end main
```

Kết quả in ra màn hình:

EAX=FFFFFFEC

(EAX = -20)

Bài 5:

Viết chương trình sử dụng lệnh LOOP với địa chỉ gián tiếp sao chép một chuỗi từ nguồn đến đích, đảo ngược thứ tự ký tự trong quá trình. Sử dụng các biến sau::

```
source BYTE "This is the source string",0
```

target BYTE SIZEOF source DUP('#')

Trả lời:

+ Chương trình:


```

.data
    source BYTE "This is the source string", 0
    target BYTE SIZEOF source DUP('#')

.code
main PROC
    MOV ESI, OFFSET source      ; ESI = địa chỉ offset của chuỗi source
    MOV EDI, OFFSET target      ; EDI = địa chỉ offset của chuỗi target
    MOV EDX, LENGTHOF source    ; EDX = độ dài chuỗi source
    SUB EDX, 2                  ; EDX = độ dài chuỗi source - 2: sao chép từ ký tự
chuỗi của source
    MOV EBX, 0                  ; EBX = 0: sao chép vào ký tự đầu của target
    MOV ECX, LENGTHOF source    ; ECX = độ dài chuỗi target
    DEC ECX                     ; ECX = độ dài chuỗi target - 1: số ký tự được sao chép
SaoChepDaoNguoc:
    MOV AL, [ESI + EDX]         ; sao chép từ chuỗi source
    MOV [EDI + EBX], AL         ; vào chuỗi target
    DEC EDX                     ; giảm chỉ số ở chuỗi source
    INC EBX                      ; tăng chỉ số ở chuỗi target
    LOOP SaoChepDaoNguoc        ; tiếp tục sao chép cho đến hết

    MOV AL, 0                   ; thêm 0
    MOV [EDI + EBX], AL         ; vào cuối target

    MOV EDX, OFFSET target
    call WriteString            ; in chuỗi target ra màn hình
exit
main endp

end main

```

+ Kết quả chuỗi target in ra màn hình:

gnirts ecruos eht si sihT

Chương 5. Procedures

Procedure	Description
CloseFile	Closes a disk file that was previously opened.
Clrscr	Clears the console window and locates the cursor at the upper left corner.
CreateOutputFile	Creates a new disk file for writing in output mode.
Crlf	Writes an end-of-line sequence to the console window.
Delay	Pauses the program execution for a specified <i>n</i> -millisecond interval.
DumpMem	Writes a block of memory to the console window in hexadecimal.
DumpRegs	Displays the EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EFLAGS, and EIP registers in hexadecimal. Also displays the most common CPU status flags.
GetCommandTail	Copies the program's command-line arguments (called the <i>command tail</i>) into an array of bytes.
GetDateTime	Gets the current date and time from the system.
GetMaxXY	Gets the number of columns and rows in the console window's buffer.
GetMseconds	Returns the number of milliseconds elapsed since midnight.
GetTextColor	Returns the active foreground and background text colors in the console window.
Gotoxy	Locates the cursor at a specific row and column in the console window.
IsDigit	Sets the Zero flag if the AL register contains the ASCII code for a decimal digit (0–9).
MsgBox	Displays a popup message box.
MsgBoxAsk	Display a yes/no question in a popup message box.
OpenInputFile	Opens an existing disk file for input.
ParseDecimal32	Converts an unsigned decimal integer string to 32-bit binary.
ParseInteger32	Converts a signed decimal integer string to 32-bit binary.
Random32	Generates a 32-bit pseudorandom integer in the range 0 to FFFFFFFFh.
Randomize	Seeds the random number generator with a unique value.
RandomRange	Generates a pseudorandom integer within a specified range.
ReadChar	Waits for a single character to be typed at the keyboard and returns the character.
ReadDec	Reads an unsigned 32-bit decimal integer from the keyboard, terminated by the Enter key.
ReadFromFile	Reads an input disk file into a buffer.

Procedure	Description
ReadInt	Reads a 32-bit signed decimal integer from the keyboard, terminated by the Enter key.
ReadKey	Reads a character from the keyboard's input buffer without waiting for input.
ReadString	Reads a string from the keyboard, terminated by the Enter key.
SetTextColor	Sets the foreground and background colors of all subsequent text output to the console.
Str_compare	Compares two strings.
Str_copy	Copies a source string to a destination string.
Str_length	Returns the length of a string in EAX.
Str_trim	Removes unwanted characters from a string.
Str_ucase	Converts a string to uppercase letters.
WaitMsg	Displays a message and waits for a key to be pressed.
WriteBin	Writes an unsigned 32-bit integer to the console window in ASCII binary format.
WriteBinB	Writes a binary integer to the console window in byte, word, or doubleword format.
WriteChar	Writes a single character to the console window.
WriteDec	Writes an unsigned 32-bit integer to the console window in decimal format.
WriteHex	Writes a 32-bit integer to the console window in hexadecimal format.
WriteHexB	Writes a byte, word, or doubleword integer to the console window in hexadecimal format.
WriteInt	Writes a signed 32-bit integer to the console window in decimal format.
WriteStackFrame	Writes the current procedure's stack frame to the console.
WriteStackFrameName	Writes the current procedure's name and stack frame to the console.
WriteString	Writes a null-terminated string to the console window.
WriteToFile	Writes a buffer to an output file.
WriteWindowsMsg	Displays a string containing the most recent error generated by MS-Windows.

Bài 1:

Xóa màn hình, delay chương trình trong 500 mili giây và kết xuất các thanh ghi và cờ.

Trả lời:

+ Chương trình:

```
.data
.code
main PROC
    call Clrscr          ; Xóa màn hình
    MOV EAX, 500         ; EAX = 500
    call Delay           ; Delay chương trình 500ms
    call DumpRegs        ; Kết xuất các thanh ghi
exit
main endp
end main
```

Bài 2:

Hiển thị một chuỗi kết thúc bằng null và di chuyển con trỏ đến đầu dòng màn hình tiếp theo.

Trả lời:

+ Chương trình:

```
.data
    msg BYTE "Xin chao cac ban",0
.code
main PROC
    MOV EDX, OFFSET msg        ; EDX = đại chỉ offset msg
    call WriteString           ; Hiển thị chuỗi msg
    call Crlf                  ; Di chuyển con trỏ đến đầu dòng màn hình tiếp theo
exit
main endp
end main
```

Bài 3:

Tạo và hiển thị mười số nguyên có dấu giả ngẫu nhiên trong phạm vi 0 - 99.

Chuyển từng số nguyên tới WriteInt trong EAX và hiển thị trên một dòng riêng biệt.

Trả lời:

+ Chương trình:

```
.data
.code
main PROC
    MOV ECX, 20                ; Lặp 10 lần
TaoSoNgauNhan:
    MOV EAX, 100               ; EAX = 100
    call RandomRange           ; Tạo số ngẫu nhiên từ 0 đến 99, lưu trong EAX
    call WriteInt              ; In số nguyên có dấu EAX lên màn hình
    call crlf                  ; Xuống dòng
    LOOP TaoSoNgauNhan         ; Lặp tạo số ngẫu nhiên 10 lần
exit
main endp
end main
```

Bài 4:

Viết chương trình xóa màn hình, nhắc người dùng nhập hai số nguyên, cộng các số nguyên và hiển thị tổng của chúng.

Trả lời:

+ Chương trình:

```
.data
    msg1 BYTE 'Nhap so thu nhat: ', 0
    msg2 BYTE 'Nhap so thu hai: ', 0
    msg3 BYTE 'Tong 2 so vua nhap la: ', 0
.code
main PROC

    MOV EDX, OFFSET msg1
    call WriteString
    call ReadInt                ; Doc so thu nhat lưu vào EAX
    MOV EBX, EAX                ; Lưu số thứ nhất vào EBX

    MOV EDX, OFFSET msg2
    call WriteString
    call ReadInt                ; Đọc số thứ 2 lưu vào EAX

    ADD EAX, EBX                ; Cộng số thứ nhất với số thứ 2 lưu vào EAX

    MOV EDX, OFFSET msg3
    call WriteString
    call WriteInt               ; Hiển thị tổng 2 số ra màn hình

exit
main endp
end main
```

Bài 5:

Thủ tục RandomRange từ thư viện Irvine32 tạo ra một số nguyên giả ngẫu nhiên từ 0 đến N-1.

Viết một thủ tục BetterRandomRange cải tiến tạo ra một số nguyên ngẫu nhiên từ M đến N-1 với M, N nhập từ bàn phím.

```
mov ebx,-300 ; lower bound
mov eax,100 ; upper bound
```

call BetterRandomRange

Trả lời:

```
.data
.code
main PROC
    MOV EBX, -300
    MOV EAX, 100
    call BetterRandomRange
    call WriteInt
exit
main endp

BetterRandomRange PROC
    call Randomize
    SUB EAX, EBX
    call RandomRange
    ADD EAX, EBX
ret
BetterRandomRange ENDP

end main
```

Bài 6:

Viết chương trình tạo và hiển thị 20 chuỗi ngẫu nhiên, mỗi chuỗi gồm 10 chữ cái viết hoa {A..Z}.

Trả lời:

+ Chương trình:

```
.data
    tmp BYTE 20 DUP(?), 0
.code
main PROC
    call Randomize
    MOV ECX, 20
TaoChuoi:
    MOV EDX, OFFSET tmp
    call RandomString
    call WriteString
    call crlf
    LOOP TaoChuoi
exit
main endp
;----- Hàm tạo chữ cái ngẫu nhiên -----
RandomChar PROC
    MOV EAX, 26
```

```

call RandomRange
ADD EAX, 'A'
ret
RandomChar ENDP
;----- Hàm tạo chuỗi có 10 ký tự ngẫu nhiên -----
RandomString PROC USES ECX ESI EAX
    MOV ECX, 10
    MOV ESI, 0
TaoKyTu:
    call RandomChar
    MOV [EDX + ESI], EAX
    INC ESI
    LOOP TaoKyTu
ret
RandomString ENDP

end main

```

Bài 7:

Nhập vào một mảng các số nguyên Dword. Tính tổng các phần tử của mảng

Trả lời:

```

.data
    array DWORD 30 DUP(?)
    msg1 BYTE 'So phan tu cua mang: ',0
    msg2 BYTE 'Phan tu thu ', 0
    msg3 BYTE ' = ',0
    msg4 BYTE 'Tong so phan tu cua mang la: ',0
.code
main PROC
;Nhap so phan tu cua mang vao ECX
    MOV EDX, OFFSET msg1
    call WriteString
    call ReadDec
    MOV ECX, EAX
; Nhap cac phan tu cua mang
    MOV EBX, 1
    PUSH ECX
    MOV ESI, OFFSET array
NhapPhanTu:
    MOV EDX, OFFSET msg2
    call WriteString
    MOV EAX, EBX
    call WriteDec
    MOV EDX, OFFSET msg3
    call WriteString
    call ReadInt
    MOV [ESI], EAX
    INC EBX
    ADD ESI, TYPE array
    LOOP NhapPhanTu
; Tinh tong cac phan tu cua mang
    MOV EBX, 0
    MOV EAX, 0

```

```
        POP ECX
TinhTong:
        ADD EAX, array[EBX]
        ADD EBX, 4
        LOOP TinhTong
; In ket qua
        MOV EDX, OFFSET msg4
        call WriteString
        call WriteInt
exit
main endp
end main
```

Bài 8:

Nhập vào một chuỗi ký tự, đảo ngược chuỗi không dùng thêm mảng phụ

Chương 6. Conditional Processing

Directive	Description
.BREAK	Generates code to terminate a .WHILE or .REPEAT block
.CONTINUE	Generates code to jump to the top of a .WHILE or .REPEAT block
.ELSE	Begins block of statements to execute when the IF condition is false
.ELSEIF <i>condition</i>	Generates code that tests <i>condition</i> and executes statements that follow, until an .ENDIF directive or another .ELSEIF directive is found
.ENDIF	Terminates a block of statements following an IF , .ELSE, or .ELSEIF directive
.ENDW	Terminates a block of statements following a .WHILE directive
IF <i>condition</i>	Generates code that executes the block of statements if <i>condition</i> is true.
.REPEAT	Generates code that repeats execution of the block of statements until <i>condition</i> becomes true
.UNTIL <i>condition</i>	Generates code that repeats the block of statements between .REPEAT and .UNTIL until <i>condition</i> becomes true
.UNTILCXZ	Generates code that repeats the block of statements between .REPEAT and .UNTILCXZ until CX equals zero
.WHILE <i>condition</i>	Generates code that executes the block of statements between .WHILE and .ENDW as long as <i>condition</i> is true

Bài 1:

Triển khai mã C ++ sau bằng hợp ngữ, sử dụng .IF có cấu trúc khối và Chỉ thị .WHILE. Giả sử rằng tất cả các biến là số nguyên có dấu 32 bit:

```
int array [] = {10,60,20,33,72,89,45,65,72,18};
```

```
int sample= 50;
```

```
int ArraySize = sizeof array / sizeof sample;
```

```
int index = 0;
```

```
int sum = 0;
```

```
while (index <ArraySize)
```

```
{
```

```
if (array [index] <= sample)
```

```

{

sum += array [index];

}

index ++;

}

```

Trả lời:

+ Chương trình:

```

.data
    sum SDWORD 0
    sample SDWORD 50
    array1 SDWORD 10, 60, 20, 33, 72, 89, 45, 65, 72, 18
    ArraySize = ($ - array1)/TYPE sample
.code
main PROC
    MOV EAX, 0
    MOV ESI, 0
    MOV EDX, sample
    .WHILE ESI < ArraySize
        .IF array1[ESI*4] <= EDX
            ADD EAX, array1[ESI*4]
        .ENDIF
        INC ESI
    .ENDW
    MOV sum, EAX
    call WriteInt
exit
main endp
end main

```

Bài 2:

Sử dụng bảng sau làm hướng dẫn, hãy viết một chương trình yêu cầu người dùng nhập điểm kiểm tra số nguyên từ 0 đến 100. Chương trình sẽ hiển thị loại chữ cái thích hợp:

Score Range	Letter Grade
90 to 100	A
80 to 89	B
70 to 79	C
60 to 69	D
0 to 59	F

Trả lời:

```
.data
msg1 BYTE 'Input Score (0 - 100) : ',0
msg2 BYTE '-> Letter Grade: ',0
msg3 BYTE 'XXX --- Invalid Score --- XXX',0
msg4 BYTE '----- Continue (y/n): ',0

.code
main PROC
    MOV AL, 'y'
    .WHILE AL == 'y'
        MOV EDX, OFFSET msg1
        call WriteString
        call ReadDec
        .IF EAX > 100
            MOV EDX, OFFSET msg3
            call WriteString
            call Crlf
        .ELSE
            MOV EDX, OFFSET msg2
            call WriteString
            .IF EAX >= 90
                MOV EAX, 'A'
                call WriteChar
            .ELSEIF EAX >= 80
                MOV EAX, 'B'
                call WriteChar
            .ELSEIF EAX >= 70
                MOV EAX, 'C'
                call WriteChar
            .ELSEIF EAX >= 60
                MOV EAX, 'D'
                call WriteChar
            .ELSE
                MOV EAX, 'F'
                call WriteChar
            .ENDIF
            call Crlf
        .ENDIF
        MOV EDX, OFFSET msg4
        call WriteString
        call ReadChar
    .ENDWHILE
main ENDPROC
```

```

        call Crlf
    .ENDW

exit
main endp
end main

```

Bài 3:

Sử dụng chương trình giải từ bài tập trước làm điểm bắt đầu, thêm các tính năng sau:

- Chạy lặp lại để có thể nhập nhiều điểm thi.
- Tích lũy bộ đếm số điểm kiểm tra.
- Thực hiện kiểm tra phạm vi trên đầu vào của người dùng: Hiển thị thông báo lỗi nếu điểm kiểm tra nhỏ hơn 0 hoặc lớn hơn 10

Trả lời:

```

.data
strInt BYTE 10 DUP(?)
_int32 SDWORD 0
msg1 BYTE 'Input Score (0 - 100) : ',0
msg2 BYTE '-> Letter Grade: ',0
msg3 BYTE 'XXX --- Invalid Score --- XXX',0
msg4 BYTE '----- Continue (y/n): ',0

.code
main PROC
    MOV AL, 'y'
    .WHILE AL == 'y'
        MOV EDX, OFFSET msg1
        call WriteString
        MOV EDX, OFFSET strInt
        MOV ECX, SIZEOF strInt
        call ReadString
        MOV ECX, LENGTHOF strInt
        call ParseInteger32
        MOV _int32, EAX
        JC Invalid
        .IF _int32 <= 100 && _int32 >= 0
            JMP Valid
        .ENDIF
        Invalid:
            MOV EDX, OFFSET msg3
            call WriteString
            call Crlf
            JMP Lap
        Valid:
            MOV EDX, OFFSET msg2

```

```

        call WriteString
        .IF _int32 >= 90
            MOV EAX, 'A'
        .ELSEIF _int32 >= 80
            MOV EAX, 'B'
        .ELSEIF _int32 >= 70
            MOV EAX, 'C'
        .ELSEIF _int32 >= 60
            MOV EAX, 'D'
        .ELSE
            MOV EAX, 'F'
        .ENDIF
        call WriteChar
        call Crlf
    Lap:
        MOV EDX, OFFSET msg4
        call WriteString
        call ReadChar
        call Crlf
.ENDW

exit
main endp
end main

```

Bài 4:

Sử dụng ví dụ về Đăng ký đại học từ Phần 6.7.3 làm điểm bắt đầu, hãy làm như sau:

- Mã hóa logic bằng cách sử dụng CMP và các lệnh nhảy có điều kiện (thay vì các lệnh .IF và .ELSEIF).
- Thực hiện kiểm tra phạm vi về giá trị tín dụng; nó không được nhỏ hơn 1 hoặc lớn hơn 30. Nếu mục nhập không hợp lệ được phát hiện, hãy hiển thị thông báo lỗi thích hợp.
- Nhắc người dùng về các giá trị điểm trung bình và tín chỉ.
- Hiển thị một thông báo hiển thị kết quả đánh giá, chẳng hạn như “Học sinh có thể đăng ký” hoặc “Học sinh không thể đăng ký”.

Bài 5:

Tạo một chương trình có chức năng như một máy tính boolean đơn giản cho các số nguyên 32 bit. Nó sẽ hiển thị một menu yêu cầu người dùng thực hiện lựa chọn từ danh sách sau:

1. x AND y
2. x OR y
3. NOT x
4. x XOR y
5. Exit program

Khi người dùng thực hiện một lựa chọn, hãy gọi một thủ tục hiển thị tên của thao tác sắp được thực hiện.

Trả lời:

```
INCLUDE Irvine32.inc

.data
    msg1 BYTE '1. x AND y',0Dh, 0Ah,\
            '2. x OR y', 0Dh, 0Ah,\
            '3. NOT x', 0Dh, 0Ah,\
            '4. x XOR y', 0Dh, 0Ah,\
            '5. Exit program', 0Dh, 0Ah,\
            'Chooise: ',0
    msg2 BYTE 'Nhap x: ',0
    msg3 BYTE 'Nhap y: ',0
    msg4 BYTE 'Ket qua: ',0
    msg5 BYTE 'Khong hojp le!',0
.code
main PROC
    .REPEAT
        MOV EDX, OFFSET msg1
        call WriteString
        call ReadDec
        MOV ECX, EAX
        .IF ECX == 1
            call ProAND
        .ELSEIF ECX == 2
            call ProOR
        .ELSEIF ECX == 3
            call ProNOT
```

```

        .ELSEIF ECX == 4
            call ProXOR
        .ELSEIF EDX != 5
            MOV EDX, OFFSET msg5
            call WriteString
            call Crlf
            .CONTINUE
        .ENDIF
    .UNTIL ECX == 5

exit
main endp

;----- PROC x AND y -----
ProAND PROC USES EDX
    MOV EDX, OFFSET msg2
    call WriteString
    call ReadDEC
    MOV EBX, EAX
    MOV EDX, OFFSET msg3
    call WriteString
    call ReadDec
    AND EAX, EBX
    MOV EDX, OFFSET msg4
    call WriteString
    call WriteDec
    call Crlf
ret
ProAND ENDP

; ----- - PROC x OR y----- -
ProOR PROC
    MOV EDX, OFFSET msg2
    call WriteString
    call ReadDEC
    MOV EBX, EAX
    MOV EDX, OFFSET msg3
    call WriteString
    call ReadDec
    OR EAX, EBX
    MOV EDX, OFFSET msg4
    call WriteString
    call WriteDec
    call Crlf
ret
ProOR ENDP

; ----- - PROC NOT x----- -
ProNOT PROC
    MOV EDX, OFFSET msg2
    call WriteString
    call ReadDEC
    NOT EAX
    MOV EDX, OFFSET msg4
    call WriteString
    call WriteDec
    call Crlf
ret

```

```

ProNOT ENDP

; ----- - PROC x XOR y----- -
PROXOR PROC
    MOV EDX, OFFSET msg2
    call WriteString
    call ReadDEC
    MOV EBX, EAX
    MOV EDX, OFFSET msg3
    call WriteString
    call ReadDec
    XOR EAX, EBX
    MOV EDX, OFFSET msg4
    call WriteString
    call WriteDec
    call Crlf
ret
ProXOR ENDP

end main

```

Bài 6:

Sửa đổi chương trình mã hóa trong Phần 6.3.4 theo cách sau: Cho phép người dùng nhập khóa mã hóa gồm nhiều ký tự. Sử dụng khóa này để mã hóa và giải mã bản rõ bằng cách XOR từng ký tự của khóa với một byte tương ứng trong thông báo. Lặp lại phép nhiều lần nếu cần cho đến khi tất cả các byte văn bản thuần túy được dịch. Ví dụ, giả sử khóa bằng “ABXmv # 7”. Đây là cách khóa sẽ mã với các byte văn bản thuần túy:

Plain text	T	h	i	s		i	s		a		P	l	a	i	n	t	e	x	t		m	e	s	s	a	g	e	(etc.)
Key	A	B	X	m	v	#	7	A	B	X	m	v	#	7	A	B	X	m	v	#	7	A	8	X	m	v	#	7

(The key repeats until it equals the length of the plain text...)

Trả lời:

```

INCLUDE Irvine32.inc

.data
    key BYTE 8 DUP(?)
    plainText BYTE 50 DUP(?)
    ketQua BYTE 50 DUP(?)
    msg1 BYTE 'Key (8 bit): ', 0
    msg2 BYTE 'Plain text: ', 0
    msg3 BYTE 'Ket qua: ', 0

.code

```



```

main PROC
    call NhapKey

    MOV EDX, OFFSET msg2
    call WriteString
    MOV EDX, OFFSET plainText
    MOV ECX, 50
    call ReadString
    MOV ECX, EAX

    call MaHoa

    MOV EDX, OFFSET msg3
    call WriteString
    MOV EDX, OFFSET ketQua
    call WriteString
    call Crlf

exit
main endp
; -----PROC Nhập Key----- -
NhapKey PROC USES ECX EAX
    MOV EDX, OFFSET msg1
    call WriteString
    MOV EDX, OFFSET key
    MOV ECX, 9
    call ReadString
    .IF EAX != 8
        call NhapKey
    .ENDIF
ret
NhapKey ENDP

; ----- PROC Mã hóa Plain Text bằng Key đã nhập -----
MaHoa PROC USES EBX EDX EAX
    MOV EAX, 0
    MOV EBX, 0
    .WHILE EAX < ECX
        MOV DL, plainText[EAX]
        XOR DL, key[EBX]
        MOV ketQua[EAX], DL
        INC EAX
        INC EBX
        .IF EBX == 8
            MOV EBX, 0
        .ENDIF
    .ENDW
ret
MaHoa ENDP

end main

```

Chương 7. Integer Arithmetic

Table 6-4 Jumps Based on Unsigned Comparisons.

Mnemonic	Description
JA	Jump if above (if $leftOp > rightOp$)
JNBE	Jump if not below or equal (same as JA)
JAЕ	Jump if above or equal (if $leftOp \geq rightOp$)
JNB	Jump if not below (same as JAЕ)
JB	Jump if below (if $leftOp < rightOp$)
JNAЕ	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if $leftOp \leq rightOp$)
JNA	Jump if not above (same as JBE)

Table 6-5 Jumps Based on Signed Comparisons.

Mnemonic	Description
JG	Jump if greater (if $leftOp > rightOp$)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if $leftOp \geq rightOp$)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if $leftOp < rightOp$)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if $leftOp \leq rightOp$)
JNG	Jump if not greater (same as JLE)

Bài 1:

Ước chung lớn nhất (GCD) của hai số nguyên là số nguyên lớn nhất sẽ chia đều cả hai số nguyên. Thuật toán GCD liên quan đến phép chia số nguyên trong một vòng lặp, được mô tả bằng mã C++ sau:

```
int GCD (int x, int y)
```

```
{
```

```
    x = abs (x); // giá trị tuyệt đối
```

```
    y = abs (y);
```

```
    do {
```

```
        int n = x% y;
```

```

x = y;

y = n;

} while (y > 0);

trả về x;

}

```

Thực hiện hàm này bằng hợp ngữ và viết một chương trình thử nghiệm gọi hàm nhiều lần, chuyển cho nó các giá trị khác nhau. Hiện thị tất cả kết quả trên màn hình

Trả lời:

```

INCLUDE Irvine32.inc

.data
val1 SDWORD 1 DUP(0)
val2 SDWORD 1 DUP(0)
msg1 BYTE 0Dh, 0Ah, 'So thu nhat: ',0
msg2 BYTE 'So thu hai: ',0
msg3 BYTE 'Uoc chung lon nhat cua 2 so: ',0
msg4 BYTE 0Dh, 0Ah, 0Dh, 0Ah, '----- Tiep tuc (y/n) : ',0

.code
main PROC
    MOV AL, 'y'
    .REPEAT
        MOV EDX, OFFSET msg1
        call WriteString
        call ReadInt
        MOV val1, EAX

        MOV EDX, OFFSET msg2
        call WriteString
        call ReadInt
        MOV val2, EAX

        call UCLN

        MOV EDX, OFFSET msg3
        call WriteString
        call WriteInt

        MOV EDX, OFFSET msg4
        call WriteString
        call ReadChar
    .UNTIL AL != 'y'
main ENDP

```

```

exit
main endp

UCLN PROC USES EBX EDX
    MOV EAX, val1
    .IF val1 < 0
        NEG EAX
    .ENDIF

    MOV EBX, val2
    .IF val2 < 0
        NEG EBX
    .ENDIF

    .REPEAT
        MOV EDX, 0
        DIV EBX
        MOV EAX, EBX
        MOV EBX, EDX
    .UNTIL EBX <= 0

ret
UCLN ENDP

end main

```

Bài 2:

Viết một thủ tục có tên IsPrime đặt cờ Zero nếu số nguyên 32 bit được truyền vào thanh ghi EAX là số nguyên tố. Tối ưu hóa vòng lặp của chương trình để chạy hiệu quả nhất có thể. Viết chương trình thử nghiệm nhắc người dùng nhập một số nguyên, gọi IsPrime và hiển thị thông báo cho biết giá trị có phải là số nguyên tố hay không. Tiếp tục nhắc người dùng nhập số nguyên và gọi IsPrime cho đến khi người dùng nhập -1

Trả lời:

```

INCLUDE Irvine32.inc

.data
    msg1 BYTE 'Nhap so nguyen: ',0
    msg2 BYTE 'Khong phai so nguyen to!', 0Dh, 0Ah, 0Dh, 0Ah, 0
    msg3 BYTE 'La so nguyen to!',0Dh,0Ah, 0Dh, 0Ah,0

.code
main PROC

TiepTuc:
    MOV EDX, OFFSET msg1
    call WriteString
    call ReadInt

```

```

        .IF EAX == -1
            JMP KT
        .ENDIF

        call IsPrime

        JZ SoNguyenTo
        JMP KhongNguyenTo

SoNguyenTo:
    MOV EDX, OFFSET msg3
    call WriteString
    JMP TiepTuc

KhongNguyenTo:
    MOV EDX, OFFSET msg2
    call WriteString
    JMP TiepTuc

KT:

exit
main endp

IsPrime PROC USES EAX
    CMP EAX, 2
    JL KSNT

    CMP EAX, 4
    JL SNT

    MOV EBX, EAX
    MOV ECX, 2
    .WHILE ECX < EBX
        MOV EAX, EBX
        MOV EDX, 0
        DIV ECX
        .IF EDX == 0
            JMP KSNT
        .ENDIF
        INC ECX
    .ENDW

    SNT:
        CMP EAX, EAX
        JMP KetThuc

    KSNT:
        MOV EAX, 1
        CMP EAX, 0

    KetThuc:
ret
IsPrime ENDP

end main

```

Bài 3:

Thời gian của thư mục tệp MS-DOS sử dụng các bit từ 0 đến 4 cho giây, bit 5 đến 10 cho phút và bit 11 đến 15 cho giờ (đồng hồ 24 giờ).

Ví dụ: giá trị nhị phân sau cho biết thời gian là 02:16:14, ở định dạng hh: mm: ss:
00010 010000 00111

Viết thủ tục có tên ShowFileTime nhận giá trị thời gian tệp nhị phân trong thanh ghi AX và hiển thị thời gian bằng hh : mm: định dạng ss.

Trả lời:

```
INCLUDE Irvine32.inc

.data
    thoigian WORD 0001001000000111b
    msg BYTE 'Thoi gian quy doi: ',0
    msg2 BYTE ' : ',0

.code
main PROC
    MOV AX, thoigian
    call ShowFileTime
exit
main endp

ShowFileTime PROC

    MOV EDX, OFFSET msg
    call WriteString

    MOV BX, AX

    MOV CL, BH
    SHR CL, 3
    CMP CL, 10
    JGE L1
    MOV EAX, 0
    call WriteDec

L1:
    MOVZX EAX, CL
    call WriteDec
    MOV EDX, offset msg2
    call WriteString

    MOV CX, BX
    SHR CX, 5
    AND CL, 00111111b
    CMP CL, 10
    JGE L2
    MOV EAX, 0
```

```

        call WriteDec
L2 :    MOVZX EAX, CL
        call WriteDec
        MOV EDX, offset msg2
        call WriteString

        MOV CL, BL
        AND CL, 00011111b
        CMP CL, 10
        JGE L3
        MOV EAX, 0
        call WriteDec
L3 :    MOVZX EAX, CL
        call WriteDec

ret
ShowFileTime ENDP

end main

```

Bài 4:

Viết một chương trình thực hiện mã hóa đơn giản bằng cách xoay mỗi byte bản rõ một số vị trí khác nhau theo các hướng khác nhau. Ví dụ: trong mảng sau đại diện cho khóa mã hóa, giá trị âm cho biết xoay sang trái và giá trị dương cho biết xoay sang phải. Số nguyên ở mỗi vị trí cho biết độ lớn của chuyển động quay:

key BYTE -2, 4, 1, 0, -3, 5, 2, -4, -4, 6

Trả lời:

```

INCLUDE Irvine32.inc

.data
key BYTE -2, 4, 1, 0, -3, 5, 2, -4, -4, 6
banRo BYTE 50 DUP(?)
ketQua BYTE 50 DUP(?)
msg1 BYTE 'Nhập ban rõ: ',0
msg2 BYTE 'Ket qua: ',0

.code
main PROC
    MOV EDX, OFFSET msg1
    call writeString
    MOV EDX, OFFSET banRo
    MOV ECX, 50
    call ReadString

    call MaHoa

```

```

        MOV EDX, OFFSET msg2
        call WriteString
        MOV EDX, OFFSET ketQua
        call WriteString
exit
main endp

MaHoa PROC USES ECX EDX ESI EDI
        MOV ESI, 0           ; Chỉ số mảng bản rõ
        MOV EDI, 0           ; Chỉ số mảng key
        MOV EDX, lengthof key ; Chiều dài mảng key
        .WHILE ESI < EAX
            MOV BL, banRo[ESI]
            MOV ketQua[ESI], BL
            MOV CL, key[EDI]
            CMP CL, 0
            JL XoayTrai
            ROR ketQua[ESI], CL
            JMP L1
        XoayTrai:
            NEG CL
            ROL ketQua[ESI], CL
        L1:
            INC ESI
            INC EDI
            .IF EDI == EDX
                MOV EDI, 0
            .ENDIF
        .ENDW
        MOV ketQua[ESI], 0
ret
MaHoa ENDP

end main

```

Bài 5:

Viết một thủ tục tính tích từ 1-n với N nhập vào từ bàn phím

Trar loiwf:

```

INCLUDE Irvine32.inc

.data
    msg1 BYTE 'Nhap n = ', 0
    msg2 BYTE 'Tich tu 1 den n la: ', 0
.code
main PROC
    MOV EDX, OFFSET msg1
    call WriteString

    call ReadDec

    MOV EDX, offset msg2
    call WriteString

```



```

        .IF EAX == 0
            call WriteDec
        .ELSE
            call Tich
            call WriteDec
        .ENDIF

exit
main endp
;----- Tinh tich tu 1 den n voi n = EAX luu vao EAX
Tich PROC USES EBX ECX
    MOV ECX, EAX
    MOV EAX, 1
    MOV EBX, 1
    .WHILE EBX <= ECX
        MUL EBX
        INC EBX
    .ENDW
ret
Tich ENDP

end main

```

Bài 6:_____

~~T Viét một thủ tục tính tổng của dãy số từ 1/1 đến 1/n~~

Bài 7:

Viết chương trình tìm BCNN của 2 số a, b

Trả lời:

```

INCLUDE Irvine32.inc

.data
msg1 BYTE 'Nhap so thu nhat: ',0
msg2 BYTE 'Nhap so thu hai: ',0
msg3 BYTE 'BCNN cua 2 so la: ',0
.code
main PROC
    MOV EDX, OFFSET msg1
    call WriteString
    call ReadDec
    MOV EBX, EAX

    MOV EDX, OFFSET msg2
    call WriteString
    call ReadDec
    MOV EDX, EAX

    call BCNN

    MOV EDX, OFFSET msg3

```

```

        call WriteString
        call WriteDec

exit
main endp

; Tích 2 số eax và ebx lưu vào eax
BCNN PROC USES EBX ECX EDX
        .IF EAX < EBX
            XCHG EBX, EAX
        .ENDIF
        MOV ECX, 0
        .REPEAT
            INC ECX
            PUSH EAX
            MUL ECX
            DIV EBX
            POP EAX
        .UNTIL EDX == 0
        MUL ECX
ret
BCNN ENDP

end main

```

Bài 8:

Nhập vào một số a. Viết một thủ tục kiểm tra xem số a của phải là số hoàn chỉnh không

Trả lời:

Bài 9:

Viết chương trình nhập dãy các số nguyên dương từ bàn phím, cho tới khi nhập số âm thì kết thúc nhập, in giá trị lớn nhất

Bài 10:

Lập trình hợp ngữ theo yêu cầu sau:

- Đếm số ký tự X có trong một String

Bài 11:

Lập trình hợp ngữ Triển khai theo thủ tục theo yêu cầu sau:

- Thay thế ký tự X bằng ký tự Y có trong một String

Bài 12:

Lập trình hợp ngữ theo yêu cầu sau:

- Thể hiện lại bộ cấu trúc if ... then ... else của ngôn ngữ C
- Thể hiện lại cấu trúc while của ngôn ngữ C
- Thể hiện lại cấu trúc for của ngôn ngữ C

Trả lời:

Bài 13:

Câu hỏi: Lập trình hợp ngữ theo yêu cầu sau:

- Tính tổng của dãy số từ 1 đến n
- Triển khai theo thủ tục

Chương 8: Advanced Procedures

Bài 1:

Viết một chương trình thử nghiệm sử dụng INVOKE để gọi WriteColorChar và hiển thị một hình vuông màu (10 hàng x 20 cột) với các cặp màu xanh lam xen kẽ và thanh dọc màu trắng. Gọi một thủ tục riêng khi in từng hàng của hình vuông.

Bài 2:

Tạo một mảng các số nguyên được sắp xếp ngẫu nhiên. Viết Hàm hoán đổi Swap, hãy viết một vòng lặp trao đổi từng cặp số nguyên liên tiếp trong mảng.

Bài 3:

Tính n giai thừa dùng hàm đệ quy

Bài 4:

Nhập vào một mảng gồm n phần tử. Viết hàm sắp xếp mảng

Bài 5:

Viết chương trình nhập N từ bàn phím. Kiểm tra xem N có là số nguyên tố hay không

Bài 6:

Nhập vào N . Viết hàm in ra các số nguyên tố từ 1 đến N

Bài 7:

Tìm n số nguyên tố đầu tiên

Bài 8:

Viết chương trình nhập dãy các số nguyên dương từ bàn phím, cho tới khi nhập số âm thì kết thúc nhập, tìm giá trị lớn nhất

Bài 9:

Viết chương trình chọn ngẫu nhiên một kí tự c nào đó, cho phép lặp nhập một kí tự từ bàn phím, nếu trùng với c thì thông báo chọn đúng và kết thúc

Bài 10:

Viết chương in bảng cửu chương ra màn hình

Bài 11:

Viết chương trình nhập vào một chuỗi ký tự bất kỳ và xoá k ký tự của chuỗi ký tự bắt đầu từ vị trí thứ n .

Bài 12:

Nhập chuỗi họ tên (không quá 40 kí tự), chuẩn hoá chuỗi đó (kí tự đầu từ viết hoa, các kí tự khác viết thường, các từ cách nhau 1 dấu cách)

Bài 13:

Viết hàm nhập mảng, in mảng, hàm sắp xếp mảng.