Lab: 13

Department of Computer Science

Iqra University Islamabad

Computer Organization and Assembly Language

Maqsood Ahmed

ID: 38186

6.2.1 Lab Work: Demonstrating the Compare Instruction

The following program demonstrates the compare instruction and the affected flags.

```
TITLE Demonstrating the Compare Instruction  (cmp.asm)

.686
.MODEL flat, stdcall
.STACK
INCLUDE Irvine32.inc
 .data
var1    SDWORD  -3056
 .code

main PROC

mov eax, 0f7893478h

mov ebx,  1234F678h

cmp al,  bl

cmp ax,  bx

cmp eax, ebx

cmp eax, var1

exit

main ENDP

END main
```

To manually execute the compare instructions by performing the subtraction, let's break down each comparison step by step. We will analyze the overflow flag (OF), carry flag (CF), sign flag (SF), and zero flag (ZF).

Variables and Registers Initialization:
eax = 0f7893478h`
ebx = 1234F678h`
var1 = -3056` (in hexadecimal, `var1 = FFFFF470h` because -3056 in 32-bit signed integer representation is `0xFFFFF470`)

1. Compare `al` and `bl`
al = 78h` (last byte of `eax`)
bl = 78h` (last byte of `ebx`)

Perform subtraction: `78h - 78h = 0h`

Flags:
OF = 0
CF = 0
SF = 0
ZF = 1

2. Compare `ax` and `bx`
ax = 3478h` (last 2 bytes of `eax`)
bx = F678h` (last 2 bytes of `ebx`)

Perform subtraction: `3478h - F678h`

Convert `F678h` to decimal: `-3944` (because `F678h` is negative in 16-bit signed representation)
Perform the operation: `13432 - (-3944) = 13432 + 3944 = 17376`

Convert `17376` to hexadecimal: `43E0h`

Flags:
OF = 0
CF = 1
SF = 0
ZF = 0

3. Compare `eax` and `ebx`
eax = 0f7893478h`
ebx = 1234F678h`

Perform subtraction: `0F7893478h - 1234F678h = -0344FB200h`

Convert `0F7893478h` and `1234F678h` to decimal:
0F7893478h = 663282872

1234F678h = 305428856

Perform the operation: `663282872 - 305428856 = 357854016`

Convert `357854016` to hexadecimal: `15534F40h`

Flags:
OF = 0
CF = 1
SF = 0
ZF = 0

4. Compare `eax` and `var1`
eax = 0f7893478h`
var1 = FFFFF470h`

Perform subtraction: `0F7893478h - FFFFF470h = 0F788F008h`

Convert `var1` to decimal: `-3056`

Perform the operation: `663282872 - (-3056) = 663282872 + 3056 = 663285928`

Convert `663285928` to hexadecimal: `0F7894340h`

Flags:
OF = 0
CF = 1
SF = 0
ZF = 0

Summary of Flags:
cmp al, bl: `OF = 0`, `CF = 0`, `SF = 0`, `ZF = 1`**
cmp ax, bx: `OF = 0`, `CF = 1`, `SF = 0`, `ZF = 0`**
cmp eax, ebx: `OF = 0`, `CF = 1`, `SF = 0`, `ZF = 0`**
cmp eax, var1: `OF = 0`, `CF = 1`, `SF = 0`, `ZF = 0`**

These are the values for the flags after performing the comparisons as described.

The provided assembly code finds the maximum of three integers (`var1`, `var2`, and `var3`) and then outputs the maximum signed integer and the maximum unsigned integer in both decimal and hexadecimal formats. Here's the breakdown of the assembly code:

```
.686
.MODEL flat, stdcall
.STACK
INCLUDE Irvine32.inc

.data
var1    DWORD  -30  ; Equal to FFFFFFE2 (hex)
var2    DWORD  12
var3    DWORD  7
max1    BYTE    "Maximum Signed Integer = ",0
max2    BYTE    "Maximum Unsigned Integer = ",0

.code
main PROC
   ; Finding Signed Maximum
   mov eax, var1
   cmp eax, var2
   jge L1
   mov eax, var2
L1:
   cmp eax, var3
   jge L2
   mov eax, var3
L2:
   lea  edx, max1
   call WriteString
   call WriteInt
   call Crlf

   ; Finding Unsigned Maximum
   mov eax, var1
   cmp eax, var2
   jae L3
```

```
    mov eax, var2
L3:
    cmp eax, var3
    jae L4
    mov eax, var3
L4:
    lea  edx, max2
    call WriteString
    call WriteHex
    call Crlf

    exit
main ENDP
END main
```

Now, let's break down the console output:

1. Maximum Signed Integer: This part compares the signed integers and prints the maximum among them.
2. Maximum Unsigned Integer: This part compares the integers as if they were unsigned and prints the maximum among them.

For the given inputs:

- Maximum Signed Integer: 12
- Maximum Unsigned Integer: 7

OUTPUT:

```
Maximum Signed Integer = 12
Maximum Unsigned Integer = 00000007
```

6.5.3 Lab Work: Translating Nested Control Structures

Translate the following high-level control structure into assembly-language code:

`while (a <= b)`
`{ a++;`
`if (b == c)`
`a = a + b`
`else {`
`b = b - a`
`c--;`
`}`
`}`
	`Mov eax, a`
	`Mov ebx, b`
	`Mov ecx, c`
	`Cmp eax, ebx`
	`Jbe ws`
	`Jmp wExit`
	
	`Ws:`
	`Inc eax`
	`Cmp ebx, ecx`
	`Je L1`
	`Sub ebx, eax`
	`Dec ecx`
	
	`L1:`
	`Add eax, ebx`
	
	`wExit:`
	`exit`

6.7 Lab Work: Linear Search of an Integer Array

```
TITLE Linear Search (search.asm)

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data
; First element is at index 0
intArray      SDWORD   18,20,35,-12,66,4,-7,100,15
item          SDWORD   -12
FoundStr      BYTE    " is found at index ", 0
NotFoundStr   BYTE    " is not found", 0
```

```
.code
main PROC
   mov    ecx, LENGTHOF intArray   ; loop counter
   mov    eax, item                ; item to search
   mov    esi, -1                  ; index to intArray
L1:
   inc    esi                      ; increment index before search
   cmp    intArray[4*esi], eax     ; compare array element with item
   loopnz L1                       ; loop as long as item not found

   jne    notFound                 ; item not found

found:
   call   WriteInt                 ; write item
   mov    edx, OFFSET FoundStr
   call   WriteString              ; " is found at index "
   mov    eax, esi
   call   WriteDec                 ; Write index
   jmp    quit

notFound:
   call   WriteInt                 ; write item
   mov    edx, OFFSET NotFoundStr
   call   WriteString              ; " is not found"

quit:
   call   Crlf
   exit
main ENDP
END main
```

OUTPUT:

-12 is found at index 3

6.7.1 Lab Work: Assemble, Link, and Run search.exe

Check your answer in the above program and make the necessary corrections.

Modify the *item* value in the above program from **-12** to **100**. Write below the console output. Reassemble, link, and run the modified program and check your answer.

Console Output (item = 100)

100 is found at index 7

Repeat the above process but with *item* equal to **-10**. Write the Console Output in the box shown below.

Console Output (item = –10)
-10 is not found

6.8.1 Lab Work: Implementing a Switch Statement

Here's the code with the provided scenario of user input sequence 1, 3, 2, 0 and the corresponding console output:

```
TITLE Demonstrating Indirect Jump (IndirectJump.asm)

; This program shows the implementation of a switch statement
; A jump table and indirect jump are used

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data
value      SDWORD  0
valuestr    BYTE    "Value = ",0
```

```
prompt     BYTE    "Enter Selection [Quit=0,Inc=1,Dec=2,Add5=3,Sub5=4]:
",0

.code
main PROC
   ; Start at break to bypass the jump table
   jmp break

; Jump table is an array of labels (instruction addresses)
table      DWORD   case0, case1, case2, case3, case4

; Implementing a Switch Statement
case0:
   exit
case1:
   inc value
   jmp break
case2:
   dec value
   jmp break
case3:
   add value, 5
   jmp break
case4:
   sub value, 5
   jmp break

break:
   ; Display value
   mov  edx, OFFSET valuestr
   call WriteString
   mov  eax, value
   call WriteInt
   call Crlf

   ; Prompt for the user to enter his selection
   mov  edx, OFFSET prompt
   call WriteString

   ; Read input character and check its value
```

```
readch:
   mov  eax,0        ; clear eax before reading
   call ReadChar
   cmp  al, '0'
   jb   out_of_range   ; character < '0'
   cmp  al, '4'
   ja   out_of_range   ; character > '4'
   call WriteChar      ; echo character
   call Crlf
   sub  al, 30h        ; convert char into number
   jmp  table[4*eax]   ; Indirect jump using table

; Out of range: ignore input and read again
out_of_range:
   jmp  readch

main ENDP
END main
```

And the corresponding console output for the user input sequence 1, 3, 2, 0 would be:

OUTPUT:

```
Value = 1
Enter Selection [Quit=0,Inc=1,Dec=2,Add5=3,Sub5=4]:
Value = 6
Enter Selection [Quit=0,Inc=1,Dec=2,Add5=3,Sub5=4]:
Value = 5
Enter Selection [Quit=0,Inc=1,Dec=2,Add5=3,Sub5=4]:
```

Review Questions:

1. Which conditional jump instructions are based on unsigned comparisons?
- Instructions based on unsigned comparisons include:
- JA` (Jump if Above)
- JAE` (Jump if Above or Equal)
- JB` (Jump if Below)
- JBE` (Jump if Below or Equal)

2. Which conditional jump instruction is based on the contents of the ECX register?
- The conditional jump instruction based on the contents of the ECX register is `LOOP`.

3. (Yes/No) Are the JA and JNBE instructions equivalent?
- Yes, `JA` and `JNBE` instructions are equivalent. They both jump if above (unsigned comparison).

4. (Yes/No) Will the following code jump to the Target label?
```assembly
mov ax, -42
cmp ax, 26
ja Target
```
- Yes, the code will jump to the `Target` label because -42 is considered greater than 26 when treated as an unsigned integer.

5. Write instructions that jump to label L1 when the unsigned integer in DX is less than or equal to the unsigned integer in CX.
```assembly
cmp cx, dx ; Compare unsigned integers in CX and DX
jbe L1 ; Jump to L1 if CX <= DX

6. (Yes/No) The LOOPE instruction jumps to a label if and only if the zero flag is clear.
- No, the `LOOPE` instruction (also known as `LOOPZ`) jumps to a label if the zero flag is set (indicating equality), not clear.

7. Implement the following statements in assembly language, for signed integers:
; if (ebx > ecx) X = 1;
cmp ebx, ecx
jg X_greater_than_1
; else
mov X, 0
jmp end_if_1
X_greater_than_1:
mov X, 1
end_if_1:

```
  ; if (edx <= ecx && ecx <= ebx) X = 1; else X = -1;
  cmp edx, ecx
  jle edx_less_than_or_equal_to_ecx
  mov X, -1
  jmp end_if_2
edx_less_than_or_equal_to_ecx:
  cmp ecx, ebx
  jle ecx_less_than_or_equal_to_ebx
  mov X, -1
  jmp end_if_2
ecx_less_than_or_equal_to_ebx:
  mov X, 1
end_if_2:

  ; while (ebx > ecx || ebx < edx) X++;
start_loop:
  cmp ebx, ecx
  jg increase_X
  cmp ebx, edx
  jl increase_X
  jmp end_while
increase_X:
  inc X
  jmp start_loop
end_while:
```

PROGRAMMING EXERCISES:

1. Program to Compute Minimum and Maximum Values of Signed 32-bit Integers:

```
TITLE Minimum and Maximum of Signed 32-bit Integers

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data
prompt BYTE "Enter an integer (or enter any invalid input to terminate): ", 0
minVal SDWORD ?
maxVal SDWORD ?

.code
```

```
main PROC
   mov ebx, 1  ; Set initial flag to continue loop
inputLoop:
   mov edx, OFFSET prompt
   call WriteString
   call ReadInt
   mov eax, eax  ; Preserve the entered integer in EAX
   cmp eax, 0    ; Check if entered input is valid
   je exitLoop   ; If input is zero, exit loop
   cmp ebx, 0    ; Check if previous input was invalid
   je exitLoop   ; If previous input was invalid, exit loop
   mov ebx, 0    ; Set flag to continue loop

   ; Update minVal and maxVal
   cmp eax, minVal
   jl updateMin
   cmp eax, maxVal
   jg updateMax
   jmp inputLoop

updateMin:
   mov minVal, eax
   jmp inputLoop

updateMax:
   mov maxVal, eax
   jmp inputLoop

exitLoop:
   ; Display minimum and maximum values
   mov edx, OFFSET prompt
   call WriteString
   mov eax, minVal
   call WriteInt
   mov edx, OFFSET prompt
   call WriteString
   mov eax, maxVal
   call WriteInt

   call Crlf
   exit
main ENDP
END main
```

2. Program to Compute Letter Grade from Test Score:

```
TITLE Compute Letter Grade from Test Score

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data
prompt BYTE "Enter the test score (0-100): ", 0
invalidMsg BYTE "Error: Invalid test score!", 0
gradeMsg BYTE "Letter grade: ", 0

.code
main PROC
   mov edx, OFFSET prompt
   call WriteString
   call ReadInt
   cmp eax, 0
   jl invalidScore
   cmp eax, 100
   jg invalidScore

   ; Compute letter grade
   cmp eax, 90
   jge gradeA
   cmp eax, 85
   jge gradeBplus
   cmp eax, 80
   jge gradeB
   cmp eax, 75
   jge gradeCplus
   cmp eax, 70
   jge gradeC
   cmp eax, 65
   jge gradeDplus
   cmp eax, 60
   jge gradeD
   jmp gradeF

gradeA:
   mov edx, OFFSET gradeMsg
   call WriteString
```

```
    mov edx, OFFSET BYTE "A+", 0
    call WriteString
    jmp endProgram

gradeBplus:
    mov edx, OFFSET gradeMsg
    call WriteString
    mov edx, OFFSET BYTE "B+", 0
    call WriteString
    jmp endProgram

gradeB:
    mov edx, OFFSET gradeMsg
    call WriteString
    mov edx, OFFSET BYTE "B", 0
    call WriteString
    jmp endProgram

gradeCplus:
    mov edx, OFFSET gradeMsg
    call WriteString
    mov edx, OFFSET BYTE "C+", 0
    call WriteString
    jmp endProgram

gradeC:
    mov edx, OFFSET gradeMsg
    call WriteString
    mov edx, OFFSET BYTE "C", 0
    call WriteString
    jmp endProgram

gradeDplus:
    mov edx, OFFSET gradeMsg
    call WriteString
    mov edx, OFFSET BYTE "D+", 0
    call WriteString
    jmp endProgram

gradeD:
    mov edx, OFFSET gradeMsg
    call WriteString
    mov edx, OFFSET BYTE "D", 0
    call WriteString
    jmp endProgram
```

```
gradeF:
   mov edx, OFFSET gradeMsg
   call WriteString
   mov edx, OFFSET BYTE "F", 0
   call WriteString

endProgram:
   call Crlf
   exit

invalidScore:
   mov edx, OFFSET invalidMsg
   call WriteString
   call Crlf
   exit

main ENDP
END main
```

3. Convert Hexadecimal Character to Decimal:

```
TITLE Convert Hexadecimal Character to Decimal

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data
prompt BYTE "Enter a hexadecimal character (0-9, A-F): ", 0
errorMsg BYTE "Error: Invalid input!", 0
resultMsg BYTE "Decimal value: ", 0

.code
main PROC
   mov edx, OFFSET prompt
   call WriteString
   call ReadChar
   cmp al, '0'
   jb invalidInput
   cmp al, '9'
   jbe numericChar
   cmp al, 'A'
```

```
    jb invalidInput
    cmp al, 'F'
    ja invalidInput
    sub al, 'A' - 10  ; Convert from ASCII to decimal
    jmp displayResult

numericChar:
    sub al, '0'       ; Convert from ASCII to decimal

displayResult:
    mov edx, OFFSET resultMsg
    call WriteString
    movzx eax, al     ; Zero-extend AL to EAX
    call WriteDec
    call Crlf
    exit

invalidInput:
    mov edx, OFFSET errorMsg
    call WriteString
    call Crlf
    exit

main ENDP
END main
```

4. Convert Lowercase Letters to Uppercase in a String:

```
TITLE Convert Lowercase Letters to Uppercase

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data
prompt BYTE "Enter a string (up to 50 characters): ", 0
resultMsg BYTE "Modified string: ", 0

.code
main PROC
    mov edx, OFFSET prompt
    call WriteString
    call ReadString
```

```
    mov esi, OFFSET prompt  ; Point to the input string
    mov edi, OFFSET resultMsg
convertLoop:
    mov al, [esi]
    cmp al, 0
    je endConvertLoop
    cmp al, 'a'
    jl notLowercase
    cmp al, 'z'
    jg notLowercase
    sub al, 32  ; Convert lowercase to uppercase
notLowercase:
    mov [edi], al
    inc esi
    inc edi
    jmp convertLoop
endConvertLoop:
    mov edx, OFFSET resultMsg
    call WriteString
    mov eax, OFFSET prompt
    call WriteString
    call Crlf
    exit

main ENDP
END main
```

5. Replace Characters in a String:

```
TITLE Replace Characters in a String

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data
prompt BYTE "Enter a string (up to 50 characters): ", 0
charPrompt BYTE "Enter a character to replace: ", 0
resultMsg BYTE "Modified string: ", 0

.code
main PROC
    mov edx, OFFSET prompt
    call WriteString
```

```
    call ReadString
    mov esi, OFFSET prompt  ; Point to the input string
    mov edi, OFFSET resultMsg
    mov ecx, 0            ; Initialize counter for matched characters
replaceLoop:
    mov al, [esi]
    cmp al, 0
    je endReplaceLoop
    mov edx, OFFSET charPrompt
    call WriteString
    call ReadChar
    cmp al, [esi]
    jne notMatched
    mov [edi], ' '        ; Replace matched character with a blank
    inc ecx              ; Increment counter for matched characters
    jmp nextChar
notMatched:
    mov [edi], al
nextChar:
    inc esi
    inc edi
    jmp replaceLoop
endReplaceLoop:
    mov edx, OFFSET resultMsg
    call WriteString
    mov eax, OFFSET prompt
    call WriteString
    call Crlf
    exit

main ENDP
END main
```

6. Compute Smallest Integer n Such That 1 + 2 + ... + n > Sum:

```
TITLE Compute Smallest Integer n Such That 1 + 2 + ... + n > Sum

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data
prompt BYTE "Enter a sum (> 1): ", 0
errorMsg BYTE "Error: Invalid input!", 0
```

```
resultMsg BYTE "Smallest integer n: ", 0

.code
main PROC
   mov edx, OFFSET prompt
   call WriteString
   call ReadInt
   cmp eax, 2
   jle invalidInput

   mov ebx, 1  ; Initialize n
   mov ecx, eax  ; Sum to compare

sumLoop:
   add ecx, ebx  ; Increment sum by n
   inc ebx       ; Increment n
   cmp ecx, eax
   jle sumLoop   ; Continue until sum exceeds input

   mov edx, OFFSET resultMsg
   call WriteString
   mov eax, ebx
   call WriteInt
   call Crlf
   exit

invalidInput:
   mov edx, OFFSET errorMsg
   call WriteString
   call Crlf
   exit

main ENDP
END main
```

THE END