CSC 3210

Computer Organization and Programming

Chapter 4: Data Transfers, Addressing, and Arithmetic

Dr. Zulkar Nine

mnine@gsu.edu

Georgia State University

Spring 2021

Outline

- **Data Transfer Instructions**
- Addition and Subtraction
- Data-Related Operators and Directives
- Indirect Addressing
- JMP and LOOP Instructions

Data Transfer Instructions

- Operand Types
- Instruction Operand Notation
- Direct Memory Operands
- MOV Instruction
- Zero & Sign Extension
- XCHG Instruction
- Direct-Offset Instructions

Data Transfer Instructions: Operand Types

- **Immediate** – a constant integer (8, 16, or 32 bits)
  - value is **encoded** within the instruction
- **Register** – the name of a register
  - register name is **converted** to a number and **encoded** within the instruction
- **Memory** – reference to a location in memory
  - memory address is **encoded** within the instruction, or a register holds the address of a memory location

Listing File

```
00000000   .data
00000000 00000000  sum DWORD 0
00000000   .code
00000000   main proc
00000000   B8 00000008       mov eax, 8
00000005   83 C0 04          add  eax, 4
00000008   A3 00000000 R     mov sum, eax
```

Data Transfer Instructions: Instruction Operand Notation

Operand	Description
reg8	8-bit general-purpose register: AH, AL, BH, BL, CH, CL, DH, DL
reg16	16-bit general-purpose register: AX, BX, CX, DX, SI, DI, SP, BP
reg32	32-bit general-purpose register: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
reg	Any general-purpose register
sreg	16-bit segment register: CS, DS, SS, ES, FS, GS
imm	8-, 16-, or 32-bit immediate value
imm8	8-bit immediate byte value
imm16	16-bit immediate word value
imm32	32-bit immediate doubleword value
reg/mem8	8-bit operand, which can be an 8-bit general register or memory byte
reg/mem16	16-bit operand, which can be a 16-bit general register or memory word
reg/mem32	32-bit operand, which can be a 32-bit general register or memory doubleword
mem	An 8-, 16-, or 32-bit memory operand

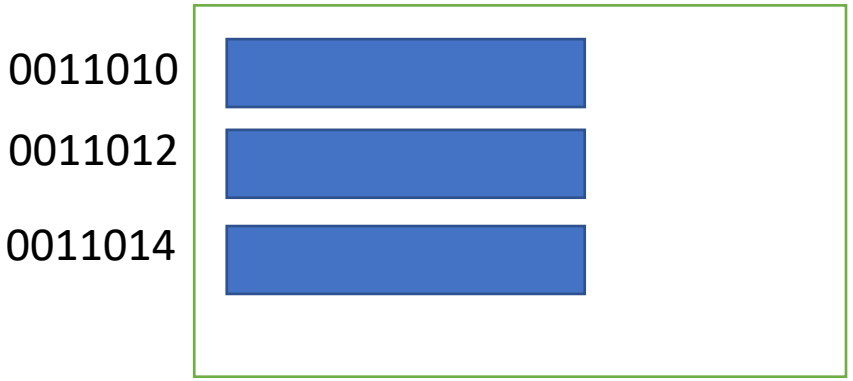
MOV **reg,reg**
MOV **mem,reg**
MOV **reg,mem**
MOV **mem,imm**
MOV **reg,imm**

```
MOVZX   reg32,reg/mem8
MOVZX   reg32,reg/mem16
MOVZX   reg16,reg/mem8
```

Data Transfer Instructions

- Register to Register

- Register to Memory , Vice versa



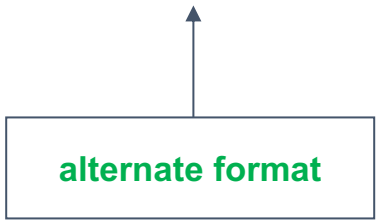
How to see variables in VS

Data Transfer Instructions: Direct Memory Operands

- A **direct memory operand** is a named reference to storage in memory

```
.data
var1 BYTE 10h
.code
mov al,var1     ; AL = 10h
```

```
mov al,[var1] ; AL = 10h
```



Data Transfer Instructions: Direct Memory Operands

- The **named reference (label)** is automatically dereferenced by the **assembler**

```
.data
var1 BYTE 10h
```

  - Suppose **var1** were located at offset 10400h.
  - The following instruction copies its **value** into the AL register:

```
mov al, var1
```

    - It would be **assembled** into the following machine instruction:

```
A0 00010400
```

Data Transfer Instructions: Direct Memory Operands

- The **named reference (label)** is automatically dereferenced by the **assembler**

A0 00010400

Listing File

- The **first byte** in the machine instruction is the **opcode**.
- The **remaining part** is the **32-bit hexadecimal address of var1**.

```
00000000   .data

00000000 00000000  sum DWORD 0

00000000   .code

00000000   main proc
00000000   B8 00000008       mov eax, 8
00000005   83 C0 04          add  eax, 4
00000008   A3 00000000 R     mov sum, eax
```

Data Transfer Instructions: MOV Instruction

- Move from **source** to **destination**.
- **Syntax:**

  MOV *destination*, *source*

- MOV instruction **formats**:

  MOV **reg,reg**
  MOV **mem,reg**
  MOV **reg,mem**
  MOV **mem,imm**
  MOV **reg,imm**

  - Both operands must be the same size.
  - Both operands cannot be memory operands.
  - The (IP, EIP, or RIP) cannot be a destination operand.

Ex:

```
.data
count BYTE 100
wVal  WORD 2
.code
    mov bl,count
    mov ax,wVal
    mov count,al

    mov al,wVal
    mov ax,count
    mov eax,count
```

Data Transfer Instructions: MOV Instruction

- Explain why each of the following MOV statements are invalid:

```
.data
bVal  BYTE   100
bVal2 BYTE   ?
wVal  WORD   2
dVal  DWORD  5

.code
        mov ds,45
        mov eax,wVal
        mov eip,dVal
        mov 25,bVal
        mov bVal2,bVal
```

- Both operands must be the same size.
- Both operands cannot be memory operands.
- The (IP, EIP, or RIP) cannot be a destination operand.

Data Transfer Instructions: MOV Instruction

- **Memory to Memory (problem):**
  - A single MOV instruction cannot be used to move data directly from one memory location to another.
  - Instead, you must move
    - the source operand's value to a register
    - before assigning its value to a memory operand:

Ex:

```
.data
var1 WORD ?
var2 WORD ?
.code
mov ax,var1
mov var2,ax
```

Data Transfer Instructions: MOV Instruction

- Overlapping Values
  - The same 32-bit register **can be modified** using differently sized data.
    - When oneWord is moved to AX, it **overwrites** the existing value of AL.
    - When oneDword is moved to EAX, it **overwrites** AX.
    - When 0 is moved to AX, it **overwrites** the lower half of EAX.

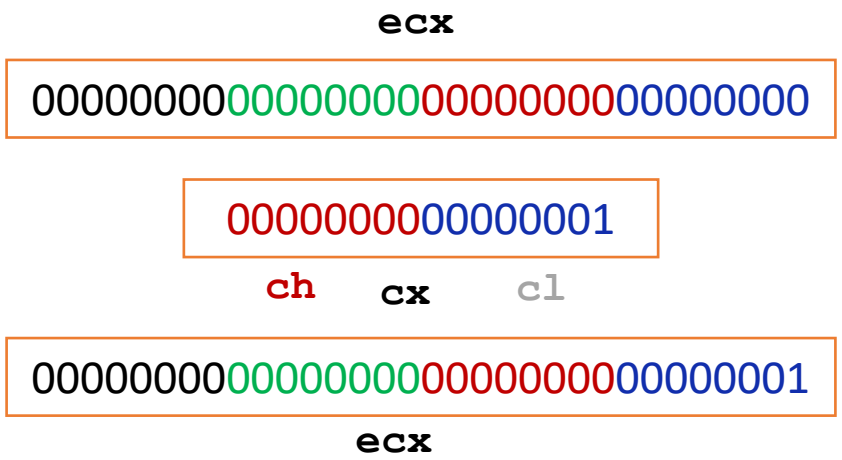
```
.data
oneWord WORD 1234h
oneDword DWORD 12345678h
```

```
.code
mov eax, 0
mov ax, oneWord          ; EAX = 00001234h
mov eax, oneDword        ; EAX = 12345678h
mov ax, 0                ; EAX = 12340000h
```

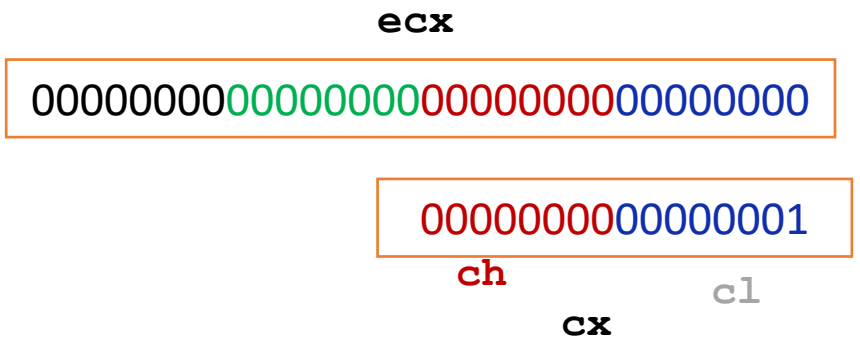
Data Transfer Instructions: Zero & Sign Extension

- **Sign extension problem:**
  - MOV cannot directly copy data from a **smaller** operand to a **larger** one
  - **Workarounds**:
    - Suppose **count** (**unsigned**, **16 bits**) must be moved to **ECX** (32 bits).
    - **Trick**: Set **ECX** to **zero** and move **count** to **CX**:

```
.data
count WORD 1
.code
mov ecx,0
mov cx,count
```



  - What happens if we try the same approach with a signed integer equal to -16?


ecx
00000000000000000000000000000000
0000000000000001
ch
cl
cx

Data Transfer Instructions: Zero & Sign Extension

- **Sign extension problem:**
  - What happens if we try the same approach with a signed integer equal to -16?

```
.data
signedVal SWORD -16              ; FFF0h (-16)
.code
mov ecx,0
mov cx,signedVal                 ; ECX = 0000FFF0h (+65,520)
```

-16 Decimal → Hex

10000

8-bit:

00010000

2's complement: 1 1 1

1 1 1 0 1 1 1 1

+1

1 1 1 1 0 0 0 0

8-bit Hex = F 0

-16 in 16-bit

0000 0000 0001 0000

1 1 1

2's complement:

1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1

+1

1 1 1 1 1 1 1 1 1 1 1 0 0 0 0

8-bit 16

pat it with 1's

F F F 0

AX BX CX DX

SI DI SP BP

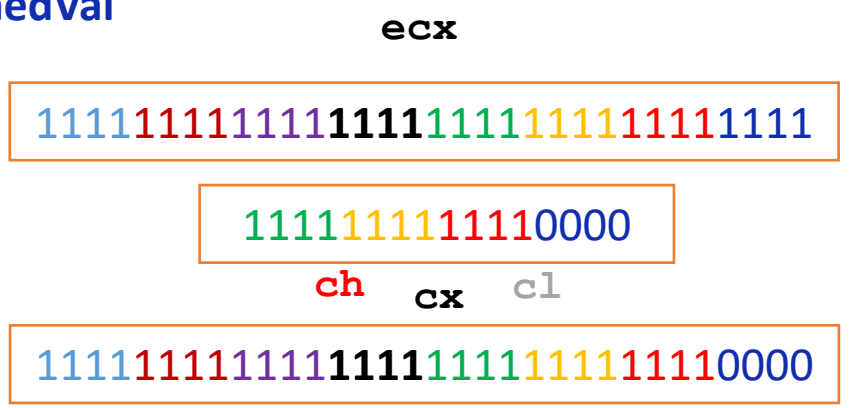
Data Transfer Instructions: Zero & Sign Extension

- **Sign extension problem:**

```
.data
signedVal SWORD -16             ; FFF0h (-16)
.code
mov ecx,0
mov cx,signedVal                ; ECX = 0000FFF0h (+65,520)
```

  - If we had filled **ECX** first with FFFFFFFFh and then copied **signedVal** to **CX**:

```
mov  ecx,FFFFFFFFh
mov  cx,signedVal          ; ECX = FFFFFFF0h (-16)
```



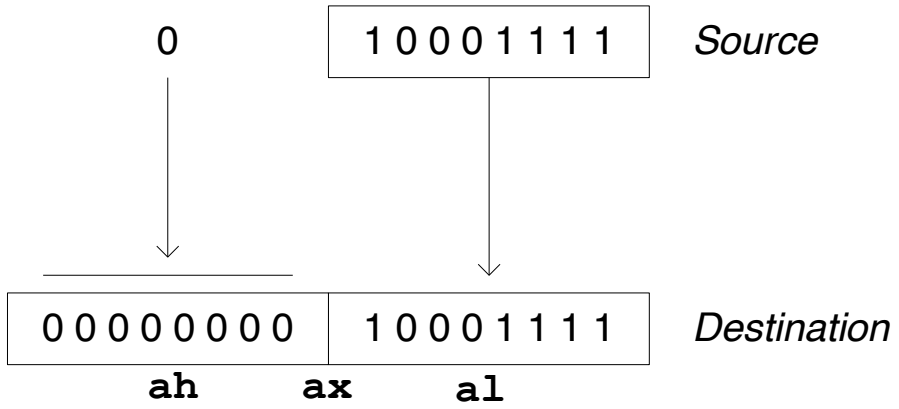
- **MOVZX** and **MOVSX** instructions to deal with both unsigned and signed integers.

Data Transfer Instructions: Zero Extension (MOVZX)

- The MOVZX instruction
- When you copy a smaller value into a larger destination,
  - the MOVZX instruction fills (extends) **the upper half** of the destination with **zeros**.

```
mov bl,10001111b
movzx ax,bl      ; zero-extension
```

```
MOVZX   reg32,reg/mem8
MOVZX   reg32,reg/mem16
MOVZX   reg16,reg/mem8
```



The destination must be a register.

Data Transfer Instructions: Zero Extension (MOVZX)

The following examples use registers for all operands, showing all the size variations:

```
mov    bx,0A69Bh
movzx  eax,bx                  ; EAX = 0000A69Bh
movzx  edx,bl                  ; EDX = 0000009Bh
movzx  cx,bl                   ; CX  = 009Bh
```

The following examples use memory operands for the source and produce the same results:

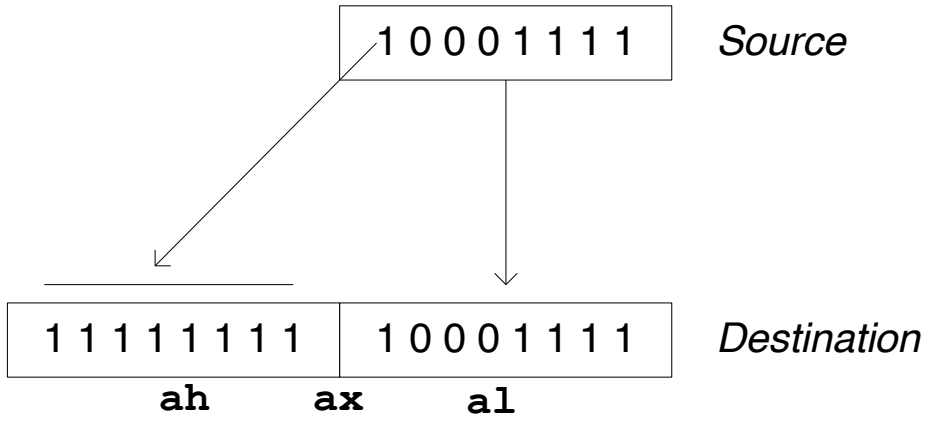
```
.data
byte1  BYTE 9Bh
word1  WORD 0A69Bh
.code
movzx  eax,word1               ; EAX = 0000A69Bh
movzx  edx,byte1               ; EDX = 0000009Bh
movzx  cx,byte1                ; CX  = 009Bh
```

Data Transfer Instructions: **Sign Extension (MOVSX)**

- The MOVSX instruction
- It fills the upper half of the destination with a copy of the source operand's sign bit.

```
MOVSX  reg32,reg/mem8
MOVSX  reg32,reg/mem16
MOVSX  reg16,reg/mem8
```

```
mov bl,10001111b
movsx ax,bl      ; sign extension
```



The destination must be a register.

Data Transfer Instructions: **Sign Extension (MOVSX)**

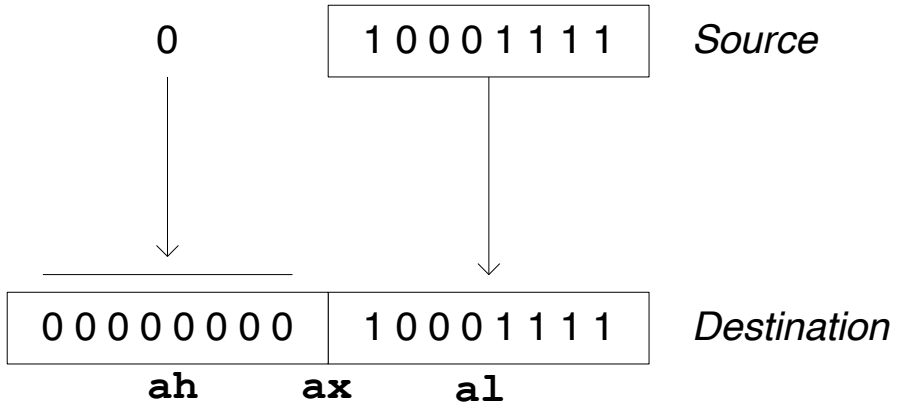
- In the following example,
  - the hexadecimal value moved to BX is A69B,
  - so the leading "A" digit tells us that the highest bit is set.

```
mov     bx, A69Bh
movsx eax,bx            ; EAX = FFFFA69Bh
movsx edx,bl            ; EDX = FFFFFF9Bh
movsx cx,bl             ; CX = FF9Bh
```

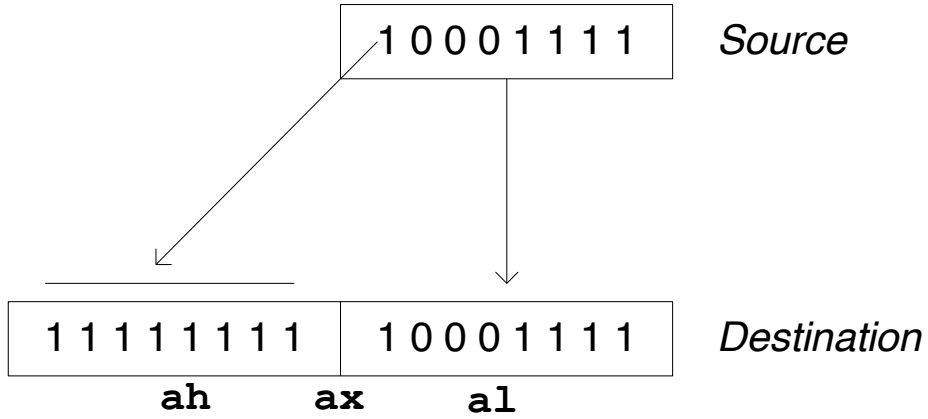
Data Transfer Instructions: Zero & Sign Extension

- MOVZX vs. MOVSX

MOVZX



MOVSX



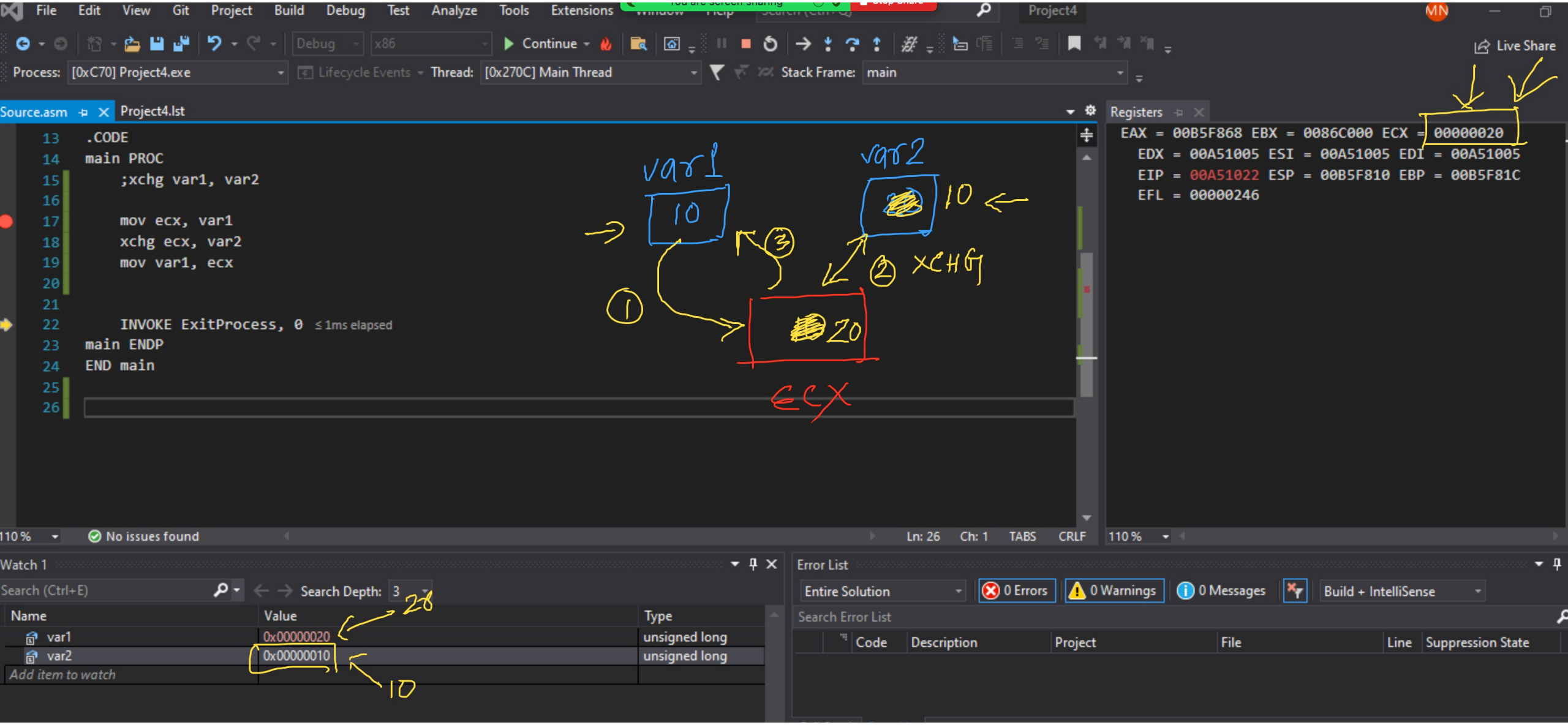
Data Transfer Instructions: XCHG Instruction

- **XCHG exchanges** the values of two operands.
  - **At least one operand must be a register**.
  - No **immediate** operands are permitted.

```
XCHG  reg,reg
XCHG  reg,mem
XCHG  mem,reg
```

```
xchg ah,al          ; exchange 8-bit regs
xchg ax,bx          ; exchange 16-bit regs
xchg eax,ebx        ; exchange 32-bit regs
xchg var1,bx ; exchange 16-bit mem op with BX
```

```
xchg var1,var2      ; error: two memory operands
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions
Project4
Debug
x86
Continue
Live Share
Process: [0xC70] Project4.exe
Lifecycle Events
Thread: [0x270C] Main Thread
Stack Frame: main
Source.asm
Project4.lst
13 .CODE
14 main PROC
15 ;xchg var1, var2
16
17 mov ecx, var1
18 xchg ecx, var2
19 mov var1, ecx
20
21
22 INVOKE ExitProcess, 0 ≤ 1ms elapsed
23 main ENDP
24 END main
25
26
var1
10
var2
10
20
XCHG
1
2
3
ecx
Registers
EAX = 00B5F868 EBX = 0086C000 ECX = 00000020
EDX = 00A51005 ESI = 00A51005 EDI = 00A51005
EIP = 00A51022 ESP = 00B5F810 EBP = 00B5F81C
EFL = 00000246
110 %
No issues found
Ln: 26 Ch: 1 TABS CRLF
110 %
Watch 1
Search (Ctrl+E)
Search Depth: 3
Name
Value
Type
var1
0x00000020
unsigned long
var2
0x00000010
unsigned long
Add item to watch
20
10
Error List
Entire Solution
0 Errors
0 Warnings
0 Messages
Build + IntelliSense
Search Error List
Code
Description
Project
File
Line
Suppression State

Data Transfer Instructions: **XCHG Instruction**

- **XCHG** **exchanges** the values of two operands.
  - **At least one operand must be a register**.
  - No **immediate** operands are permitted.

```
XCHG  reg,reg
XCHG  reg,mem
XCHG  mem,reg
```

```
.data
var1 WORD 1000h
var2 WORD 2000h
.code
mov ax,val1
xchg ax,val2
mov val1,ax
```

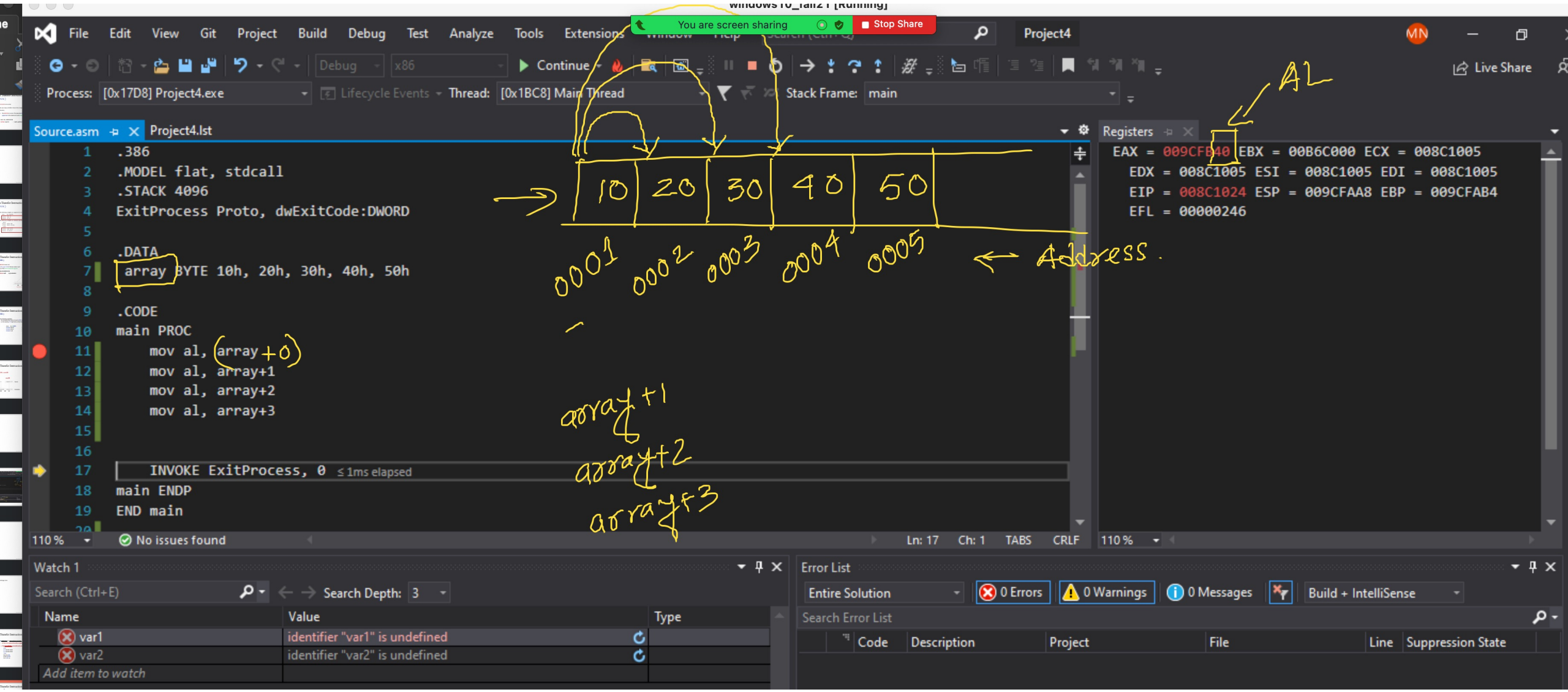
Data Transfer Instructions: **Direct-Offset Operands**

- A constant offset is added to a data label to produce an **effective address** (EA).
- The address is dereferenced to get the **value inside its memory location**.

```
.data
arrayB BYTE 10h,20h,30h,40h
.code
mov al,arrayB                    ; AL = 10h
mov al,arrayB+1                  ; AL = 20h

  mov al,[arrayB+1]        ; alternative notation
```

Why doesn't arrayB+1 produce 11h?


File Edit View Git Project Build Debug Test Analyze Tools Extensions
You are screen sharing
Stop Share
Project4
Live Share
Process: [0x17D8] Project4.exe
Thread: [0x1BC8] Main Thread
Stack Frame: main
Source.asm
Project4.lst
.386
.MODEL flat, stdcall
.STACK 4096
ExitProcess Proto, dwExitCode:DWORD
.DATA
array BYTE 10h, 20h, 30h, 40h, 50h
.CODE
main PROC
mov al, array+0
mov al, array+1
mov al, array+2
mov al, array+3
INVOKE ExitProcess, 0
main ENDP
END main
Registers
EAX = 009CFB40 EBX = 00B6C000 ECX = 008C1005
EDX = 008C1005 ESI = 008C1005 EDI = 008C1005
EIP = 008C1024 ESP = 009CFAA8 EBP = 009CFAB4
EFL = 00000246
AL
10 20 30 40 50
0001 0002 0003 0004 0005
Address.
array+1
array+2
array+3
No issues found
Watch 1
Search (Ctrl+E)
Search Depth: 3
Name
Value
Type
var1
identifier "var1" is undefined
var2
identifier "var2" is undefined
Add item to watch
Error List
Entire Solution
0 Errors
0 Warnings
0 Messages
Build + IntelliSense
Search Error List
Code
Description
Project
File
Line
Suppression State

Data Transfer Instructions: **Direct-Offset Operands**

```
.data
arrayW  WORD 1000h,2000h,3000h
arrayD  DWORD 1,2,3,4
.code
mov ax, [arrayW+2]              ; AX = 2000h
mov ax, [arrayW+4]              ; AX = 3000h
mov eax,[arrayD+4]              ; EAX = 00000002h
```

Will the following statements assemble?

```
mov ax,[arrayW-2]          ; ??
mov eax,[arrayD+16]        ; ??
```

Data Transfer Instructions: **Direct-Offset Operands**

```
.data
arrayW  WORD 1000h,2000h,3000h
arrayD  DWORD 1,2,3,4
.code
mov ax, [arrayD+0]              ; AX = 2000h
mov ax, [arrayD+4]              ; AX = 3000h
mov eax,[arrayD+8]              ; EAX = 00000002h
```

Will the following statements assemble?

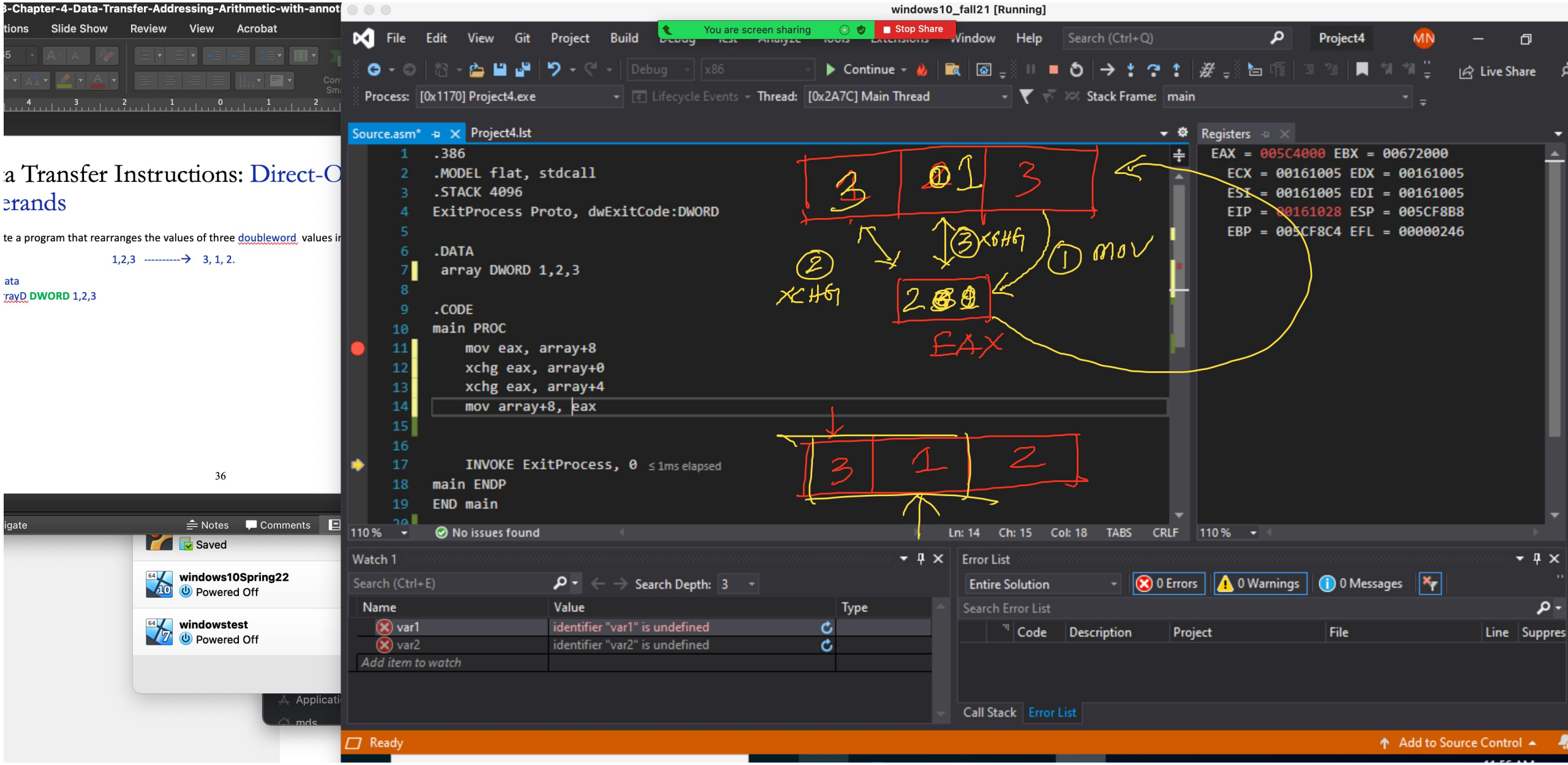
```
mov ax,[arrayW-2]          ; ??
mov eax,[arrayD+16]        ; ??
```

Data Transfer Instructions: Direct-Offset Operands

- Write a program that rearranges the values of three doubleword values in the following array as:

1,2,3 ----------→ 3, 1, 2.

```
.data
arrayD DWORD 1,2,3
```


windows10_fall21 [Running]
You are screen sharing
Stop Share
Source.asm*
Project4.lst
.386
.MODEL flat, stdcall
.STACK 4096
ExitProcess Proto, dwExitCode:DWORD
.DATA
array DWORD 1,2,3
.CODE
main PROC
mov eax, array+8
xchg eax, array+0
xchg eax, array+4
mov array+8, eax
INVOKE ExitProcess, 0
main ENDP
END main
Registers
EAX = 005C4000 EBX = 00672000
ECX = 00161005 EDX = 00161005
ESI = 00161005 EDI = 00161005
EIP = 00161028 ESP = 005CF8B8
EBP = 005CF8C4 EFL = 00000246
Watch 1
Error List
Entire Solution
0 Errors
0 Warnings
0 Messages
Name
Value
Type
var1
identifier "var1" is undefined
var2
identifier "var2" is undefined
Add item to watch
Code
Description
Project
File
Line
Call Stack
Error List
Ready
Add to Source Control
Transfer Instructions: Direct-O
erands
1,2,3 ----------→ 3, 1, 2.
DWORD 1,2,3
36
Notes
Comments
Saved
windows10Spring22
Powered Off
windowstest
Powered Off

Data Transfer Instructions: Direct-Offset Operands

- Write a program that rearranges the values of three doubleword values in the following array as:

1,2,3 ----------→ 3, 1, 2.

```
.data
arrayD DWORD 1,2,3
```

  - **Step1: copy** the **FIRST** value into **EAX** and **exchange** it with the value in **the SECOND position**.

```
mov eax,  arrayD
xchg eax, [arrayD+4]
```

```
XCHG  reg,reg
XCHG  reg,mem
XCHG  mem,reg
```

- **Step 2:** Exchange **EAX** with the **THIRD** array value and copy the value in **EAX** to the **FIRST** array position.

```
xchg eax,[arrayD+8]
mov  arrayD,eax
```