香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Tutorial 03: MASM Addressing Modes

Yuhong Liang

yhliang@cse.cuhk.edu.hk

Program Structure Review

```
.386
.model flat, stdcall
option casemap:none
include windows.inc
include kernel32.inc
include user32.inc
```

Assembler Directives

```
.data
MsgCaption db "CSCI2510 Tutorial", 0
MsgBoxText db "Hello, World!", 0
```

Data Segment

```
.code
start:
    invoke MessageBox, NULL,addr MsgBoxText, addr MsgCaption, MB_OK
    invoke ExitProcess,NULL
end start
```

Code Segement

Program Structure Review

- Assembler Directives
  - Telling the assembler what to do:
    - Option, configuration, syntax etc…
- Data Segment
  - Declare and apply some memory space in primary memory (e.g. RAM)
  - Assign value to corresponding data object
- Code Segment
  - State the following segment is the program assembly code
  - Call function with arguments in data segment

Outline

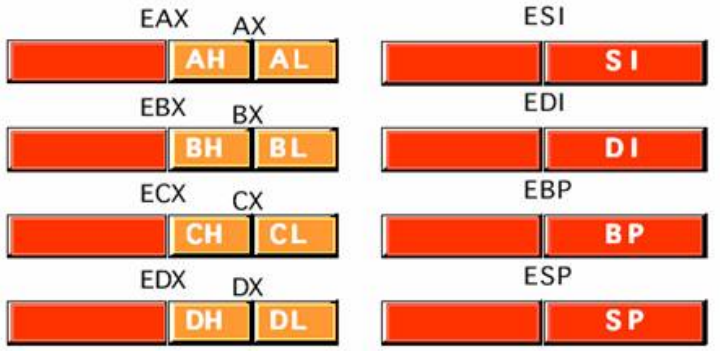
- Registers and Memory Declaration in MASM
- `mov`: Data Movement Instruction
- Addressing Modes in MASM
- Basic Debugging Operations in MASM
  - Print
  - IDE built in

Registers in MASM (1/2)

- General Purpose 32-bit Registers
  - EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP
- Subsections may be used.
  - The least significant 16-bit of EAX can be treated as a 16-bit register called AX.
  - The least significant byte of AX can be used as a single 8-bit register called AL, while the most significant byte of AX can be used as a single 8-bit register called AH.

Registers in MASM (2/2)

- These registers can be used by most of instructions:
  - mov EAX, 100: EAX stores 100
  - add EAX, 1: EAX = EAX + 1
  - …

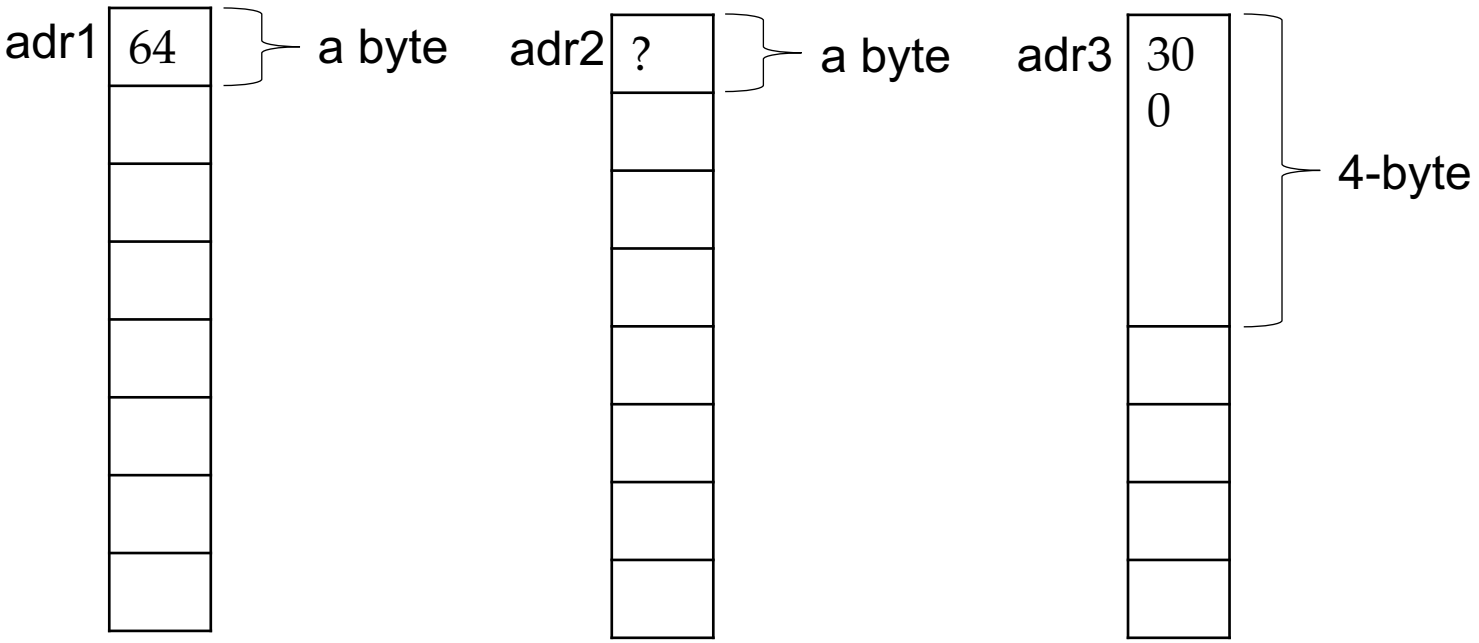


- Some special and advanced usages of these registers will be introduced later:
  - EAX, EDX: div instructions …
  - EBP, ESP: push, pop …
    - https://stackoverflow.com/questions/12503850/why-esp-register-is-discouraged-to-use-while-using-push-or-pop-instructions
  - ESI, EDI: string instructions …
    - https://www.tutorialspoint.com/assembly_programming/assembly_movs_instruction.htm

Memory Allocation in MASM (1/3)

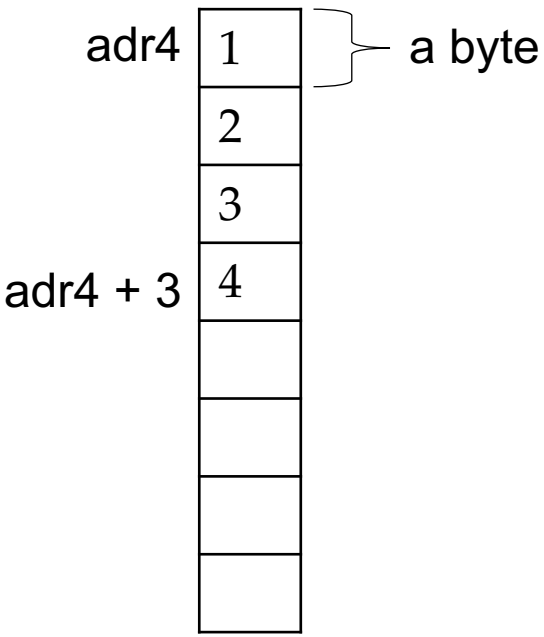
.data segment

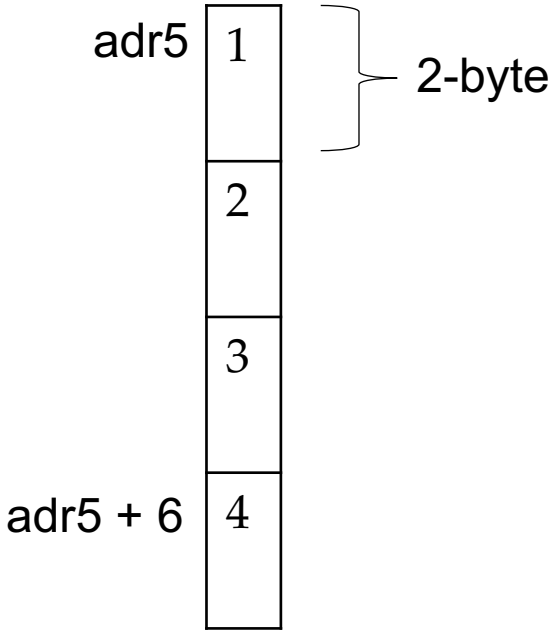
- adr1 DB 64
  - Declare a byte, referred to as location adr1, containing the value 64.
- adr2 DB ?
  - Declare an uninitialized byte, referred to as location adr2.
- adr3 DD 300
  - Declare a 4-byte value, referred to as location adr3, initialized to 300.



Memory Allocation in MASM (2/3)

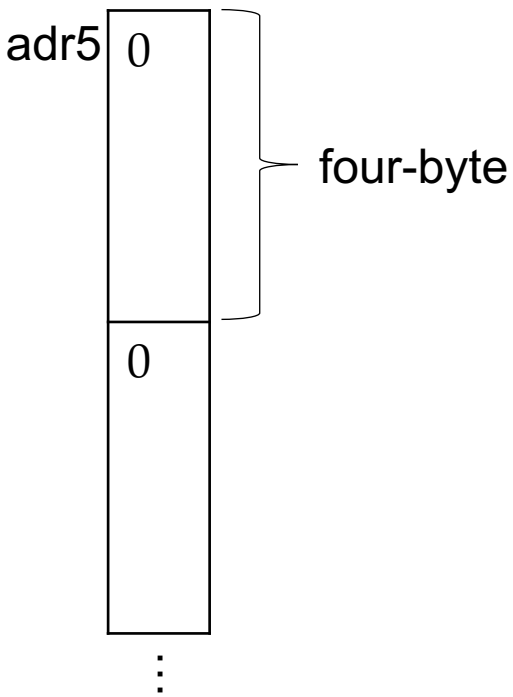
- adr4 DB 1, 2, 3, 4
  - Declare four one-byte values, initialized to 1, 2, 3 and 4. The value of location adr4 + 3 will be 4.
- adr5 DW 1, 2, 3, 4
  - Declare four 2-byte values, initialized to 1, 2, 3 and 4. The value of location adr4 + 6 will be 4.

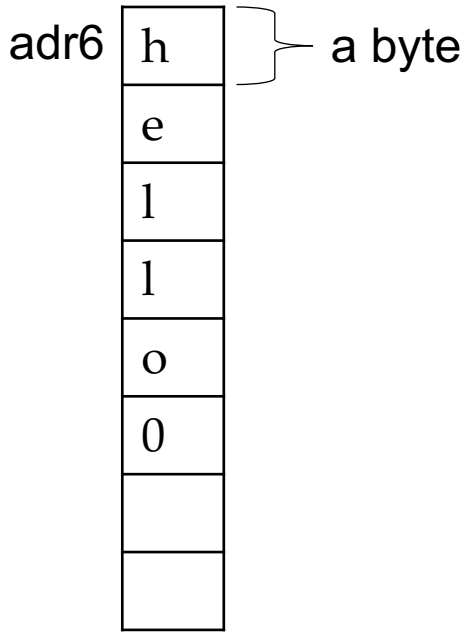




Memory Allocation in MASM (3/3)

- adr6 DD 100 DUP(0) = adr6 DD 0, 0, 0, 0, 0 …
  - Declare 100 4-byte value starting at location adr6, all initialized to 0
- adr7 DB 'hello',0
  - Declare 6 bytes starting at the address adr7, initialized to the ASCII character values for hello and the null (0) byte.





mov: Data Movement Instruction

mov

- Syntax:

mov <reg>,<con>

mov <reg>,<reg>

mov <mem>,<con>

mov <reg>,<mem>

mov <mem>,<reg>

- Semantics:

The mov instruction moves the data represented by second label (i.e.register contents, memory contents, or a constant value) into the location represented by first label (i.e. a register or memory).

- <reg>：usually EAX,EBX,ECX,EDX…
- <con>: -1,0,1,2,3,4…
- <mem>: LOC, number[register], [combination of number and register]…

mov destination source

More instruction:

https://software.intel.com/content/www/us/en/develop/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4.html

Addressing Modes in MASM (1/8)

- Addressing Modes: the ways for specifying the contents or locations of instruction operands.

Address Mode	Assembler Syntax	MASM Syntax	Addressing Function
1) Immediate	$\#Value$	$Value$	$Operand = Value$
2) Register	Ri	Ri	$EA = Ri$
3) Absolute	LOC	LOC	$EA = LOC$
4) Register indirect	(Ri)	$[Ri]$	$EA = [Ri]$
5) Index	$X(Ri)$	$X[Ri]$	$EA = [Ri] + X$
6) Base with index	(Ri, Rj)	$[Ri][Rj]$ or $[Ri + Rj]$	$EA = [Ri] + [Rj]$

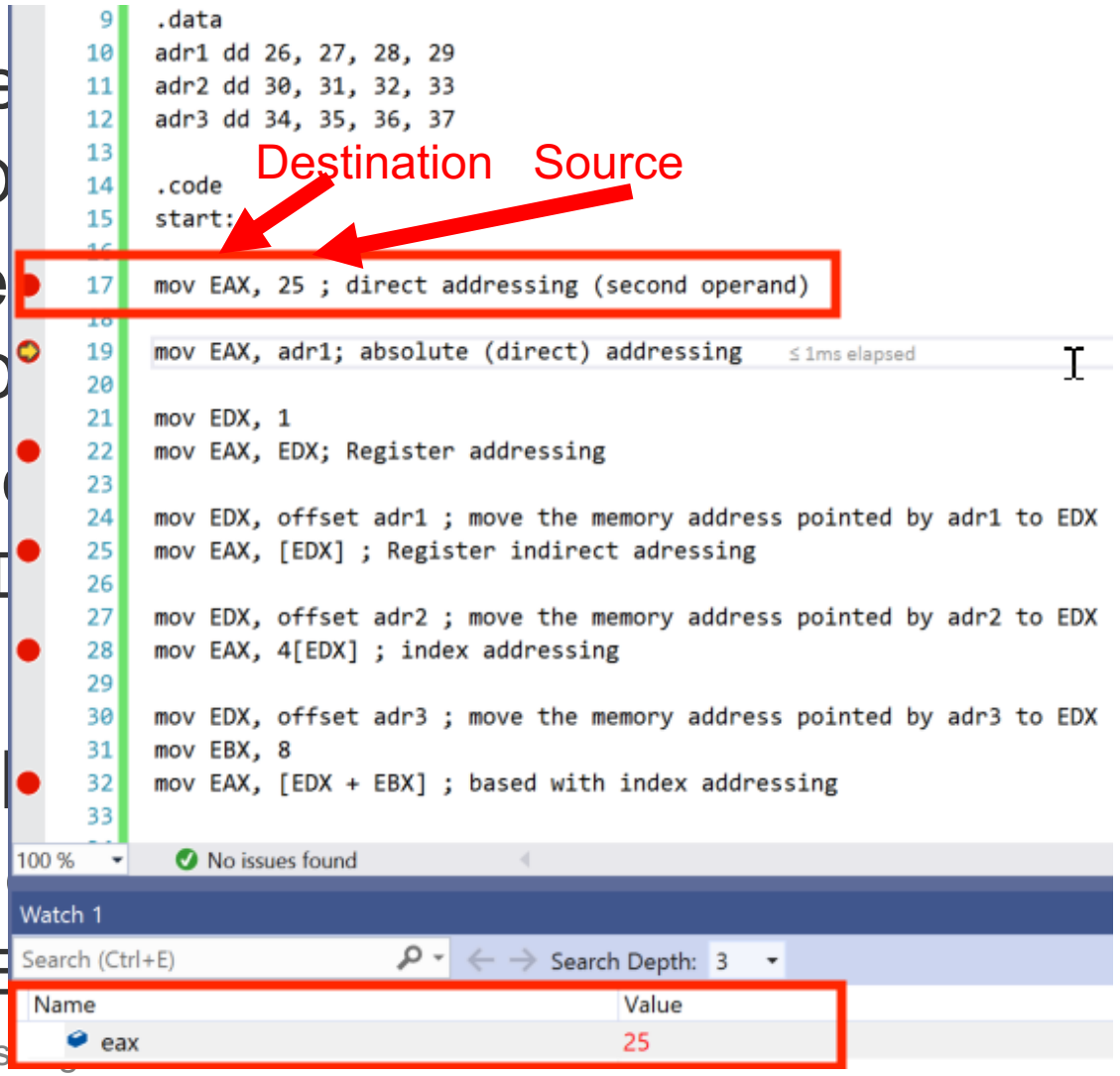
$Value$: a signed number
EA: the effective address of a register or a memory location
X: an index value

Addressing Modes in MASM (2/8)

- Immediate addressing
  - MOV EAX, 25
- Absolute addressing (Direct addressing)
  - MOV EAX, adr1;
- Register addressing
  - MOV EAX, EDX
- Register indirect addressing
  - MOV EAX, [EDX]
- Index addressing
  - MOV EAX, 4[EDX]
- Based with index
  - MOV EAX, [EBX + ECX]

Addressing Modes in MASM (3/8)

- Immediate addressing
  - MOV EAX, 25
- Absolute addre
  - MOV EAX, LO
- Register addre
  - MOV EAX, ED
- Register indire
  - MOV EAX, [ED
- Index addressi
  - MOV EAX, 32[
- Based with ind
  - MOV EAX, [EB



Addressing Modes in MASM (4/8)

- Immediate addressing
  - MOV EAX, 25
- Absolute addressing (Direct addressing)
  - MOV EAX, adr1;
- Register addressir
  - MOV EAX, EDX
- Register indirect a
  - MOV EAX, [EDX]
- Index addressing
  - MOV EAX, 4[EDX]
- Based with index
  - MOV EAX, [EBX +

```
9   .data
10  adr1 dd 26, 27, 28, 29
11  adr2 dd 30, 31, 32, 33
12  adr3 dd 34, 35, 36, 37
13
14  .code
15  start:
16
17  mov EAX, 25 ; direct addressing (second operand)
18
19  mov EAX, adr1; absolute (direct) addressing
20
21  mov EDX, 1
22  mov EAX, EDX; Register addressing ≤ 2ms elapsed
23
24  mov EDX, offset adr1 ; move the memory address pointed by adr1 to EDX
25  mov EAX, [EDX] ; Register indirect adressing
26
27  mov EDX, offset adr2 ; move the memory address pointed by adr2 to EDX
28  mov EAX, 4[EDX] ; index addressing
29
30  mov EDX, offset adr3 ; move the memory address pointed by adr3 to EDX
31  mov EBX, 8
32  mov EAX, [EDX + EBX] ; based with index addressing
33
```

100 % No issues found

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value
eax	26

Addressing Modes in MASM (5/8)

- Immediate addressing
  - MOV EAX, 25
- Absolute addressing
  - MOV EAX, adr1;
- Register addressing
  - MOV EAX, EDX
- Register indirect add
  - MOV EAX, [EDX]
- Index addressing
  - MOV EAX, 4[EDX]
- Based with index
  - MOV EAX, [EBX + E

```
9   .data
10  adr1 dd 26, 27, 28, 29
11  adr2 dd 30, 31, 32, 33
12  adr3 dd 34, 35, 36, 37
13
14  .code
15  start:
16
17  mov EAX, 25 ; direct addressing (second operand)
18
19  mov EAX, adr1; absolute (direct) addressing
20
21  mov EDX, 1
22  mov EAX, EDX; Register addressing
23
24  mov EDX, offset adr1 ; move the memory address pointed by adr1 to EDX
25  mov EAX, [EDX] ; Register indirect adressing  ≤ 1ms elapsed
26
27  mov EDX, offset adr2 ; move the memory address pointed by adr2 to EDX
28  mov EAX, 4[EDX] ; index addressing
29
30  mov EDX, offset adr3 ; move the memory address pointed by adr3 to EDX
31  mov EBX, 8
32  mov EAX, [EDX + EBX] ; based with index addressing
33
```

100 % No issues found

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value
eax	1

Addressing Modes in MASM (6/8)

- Immediate addres
  - MOV EAX, 25
- Absolute addressi
  - MOV EAX, adr1;
- Register addressi
  - MOV EAX, EDX
- Register indirect addressing
  - MOV EAX, [EDX]
- Index addressing
  - MOV EAX, 4[EDX
- Based with index
  - MOV EAX, [EBX +

```
9   .data
10  adr1 dd 26, 27, 28, 29
11  adr2 dd 30, 31, 32, 33
12  adr3 dd 34, 35, 36, 37
13
14  .code
15  start:
16
17  mov EAX, 25 ; direct addressing (second operand)
18
19  mov EAX, adr1; absolute (direct) addressing
20
21  mov EDX, 1
22  mov EAX, EDX; Register addressing
24  mov EDX, offset adr1 ; move the memory address pointed by adr1 to
25  mov EAX, [EDX] ; Register indirect adressing
26
27  mov EDX, offset adr2 ; move the memory address pointed by adr2 to
28  mov EAX, 4[EDX] ; index addressing  ≤ 1ms elapsed
29
30  mov EDX, offset adr3 ; move the memory address pointed by adr3 to
31  mov EBX, 8
32  mov EAX, [EDX + EBX] ; based with index addressing
33
```

offset:
"memory address" not "content" in the memory

100 % No issues found

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value
eax	26

Addressing Modes in MASM (7/8)

- Immediate addressing
  - MOV EAX, 25
- Absolute addressin
  - MOV EAX, adr1;
- Register addressin
  - MOV EAX, EDX
- Register indirect ac
  - MOV EAX, [EDX]
- Index addressing
  - MOV EAX, 4[EDX]
- Based with index
  - MOV EAX, [EBX +

```
9   .data
10  adr1 dd 26, 27, 28, 29
11  adr2 dd 30, 31, 32, 33
12  adr3 dd 34, 35, 36, 37
13
14  .code
15  start:
16
17  mov EAX, 25 ; direct addressing (second operand)
18
19  mov EAX, adr1; absolute (direct) addressing
20
21  mov EDX, 1
22  mov EAX, EDX; Register addressing
23
24  mov EDX, offset adr1 ; move the memory address pointed by adr1 to EDX
25  mov EAX, [EDX] ; Register indirect adressing
26
27  mov EDX, offset adr2 ; move the memory address pointed by adr2 to EDX
28  mov EAX, 4[EDX] ; index addressing
29
30  mov EDX, offset adr3 ; move the memory address pointed by adr3 to EDX
31  mov EBX, 8
32  mov EAX, [EDX + EBX] ; based with index addressing ≤ 1ms elapsed
33
```

100 % No issues found

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value
eax	31

Addressing Modes in MASM (8/8)

- Immediate addres
  - MOV EAX, 25
- Absolute addressi
  - MOV EAX, adr1;
- Register addressir
  - MOV EAX, EDX
- Register indirect a
  - MOV EAX, [EDX]
- Index addressing
  - MOV EAX, 4[EDX]
- Based with index
  - MOV EAX, [EBX + ECX]

```
9   .data
10  adr1 dd 26, 27, 28, 29
11  adr2 dd 30, 31, 32, 33
12  adr3 dd 34, 35, 36, 37
13
14  .code
15  start:
16
17  mov EAX, 25 ; direct addressing (second operand)
18
19  mov EAX, adr1; absolute (direct) addressing
20
21  mov EDX, 1
22  mov EAX, EDX; Register addressing
23
24  mov EDX, offset adr1 ; move the memory address pointed by adr1 to EDX
25  mov EAX, [EDX] ; Register indirect adressing
26
27  mov EDX, offset adr2 ; move the memory address pointed by adr2 to EDX
28  mov EAX, 4[EDX] ; index addressing
29
30  mov EDX, offset adr3 ; move the memory address pointed by adr3 to EDX
31  mov EBX, 8
32  mov EAX, [EDX + EBX] ; based with index addressing
33
```

100 % No issues found

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value
eax	36

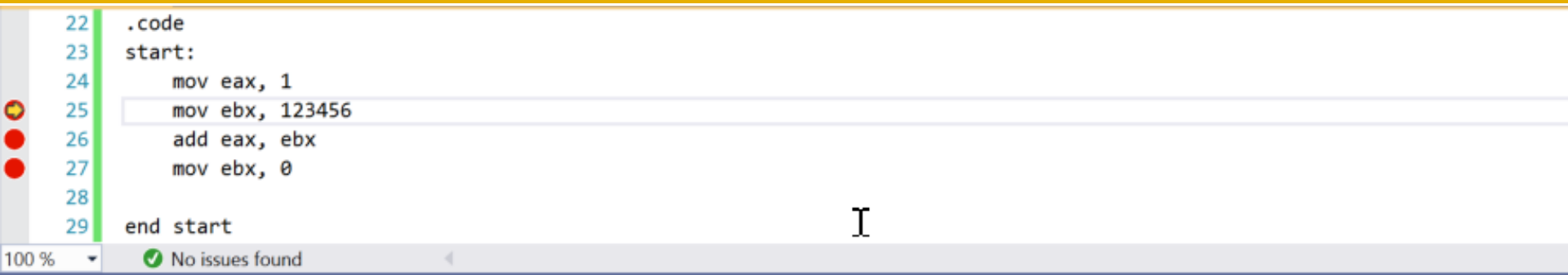
Basic Debugging Operations

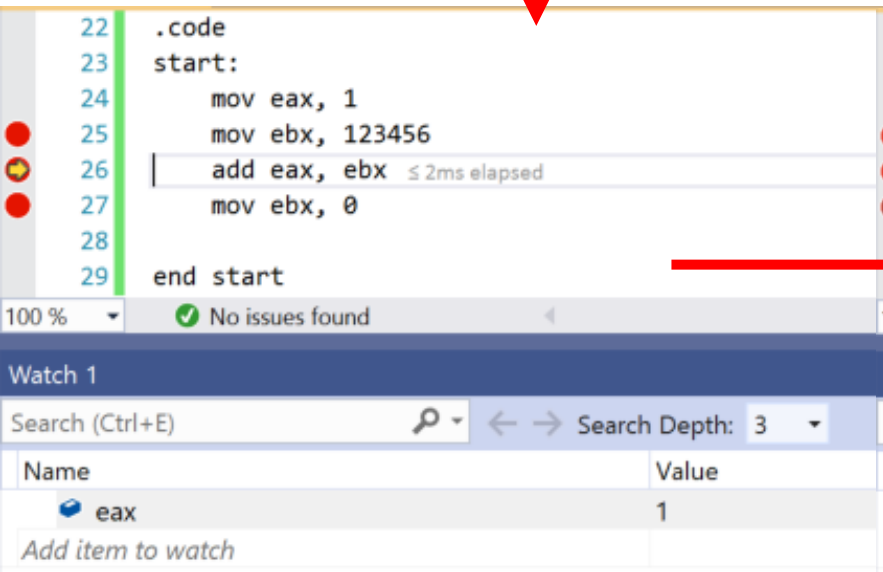
- **Method 1: Print the log**
  - Function in MASM
  - Three Print Functions
    - crt_printf
    - StdOut
    - MessageBox
- **Method 2: Built-in debugging** (local windows debugger)

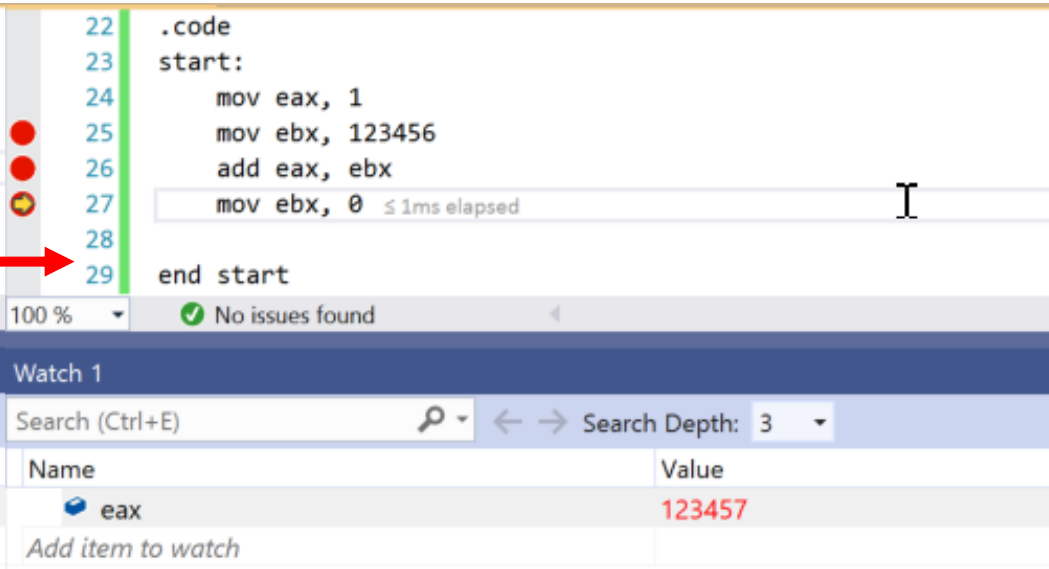
Built-in Debugging: Monitor Registers

- Steps:
  - Add breakpoint(s) at some instruction(s)
  - Start Debugging (F5)
  - Menu Debug --> Window --> Registers to watch the value of each register
  - Put the name of the register into the watch table

Built-in Debugging: Monitor Registers







Summary

- Registers and Memory Declaration in MASM
- `mov`: Data Movement Instruction
- Addressing Modes in MASM
- Basic Debugging Operations in MASM
  - Print
  - IDE built in