香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Tutorial 05: Hints for Stack Implementation

Yuhong LIANG

yhliang@cse.cuhk.edu.hk

Outline

- Review of idiv Instruction
- Stack Basics
- Tracking stack.asm
- Hints for Stack Implementation (Assignment 2)

idiv instruction (1/3)

idiv: data arithmetic instruction

- The idiv instruction divides **the contents of the 64 bit integer EDX:EAX** by the specified operand value.
- **Quotient result->EAX**
- **Remainder->EDX**
- Syntax: idiv EBX (EBX is divisor)

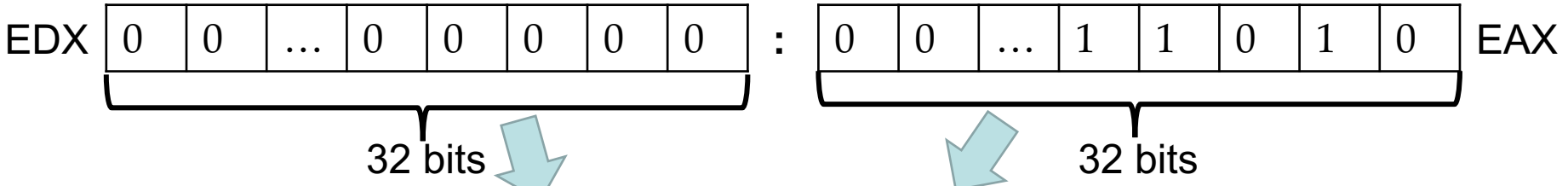
":" means concatenation. The dividend is the concatenation result of EDX and EAX. This is fixed.

mov EAX, 26
mov EBX, 5
mov EDX, 0
idiv EBX
EAX = 5 ; EDX= 1
EDX:EAX = 26 EBX = 5 26 / 5 = 5 …… 1

mov EAX, 0
mov EBX, 4
mov EDX, 1
idiv EBX
EAX = 1073741824 ; EDX= 0
EDX:EAX = 4294967296 EBX = 4 4294967296 / 4 = 1073741824 …… 0

idiv instruction (2/3)

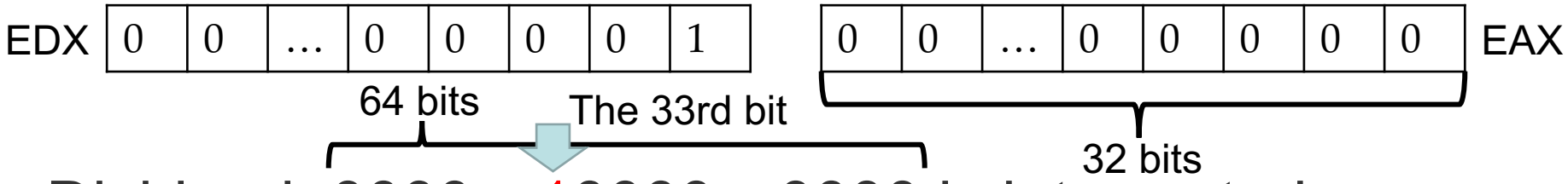
mov EAX, 26
mov EBX, 5
mov EDX, 0
idiv EBX
EAX = 5 ; EDX= 1
EDX:EAX = 26 EBX = 5 26 / 5 = 5 …… 1



- Dividend: 0000000….000000011010 is interpreted as 26 using 2's-complement. (64 bits)
- Divisor: EBX = 5
- Quotient result put in EAX, Remainder put in EDX,

idiv instruction (3/3)

mov EAX, 0
mov ECX, 4
mov EDX, 1
idiv ECX
EAX = 1073741824 ; EDX= 0
EDX:EAX = 4294967296 ECX = 4 4294967296 / 4 = 1073741824 …… 0

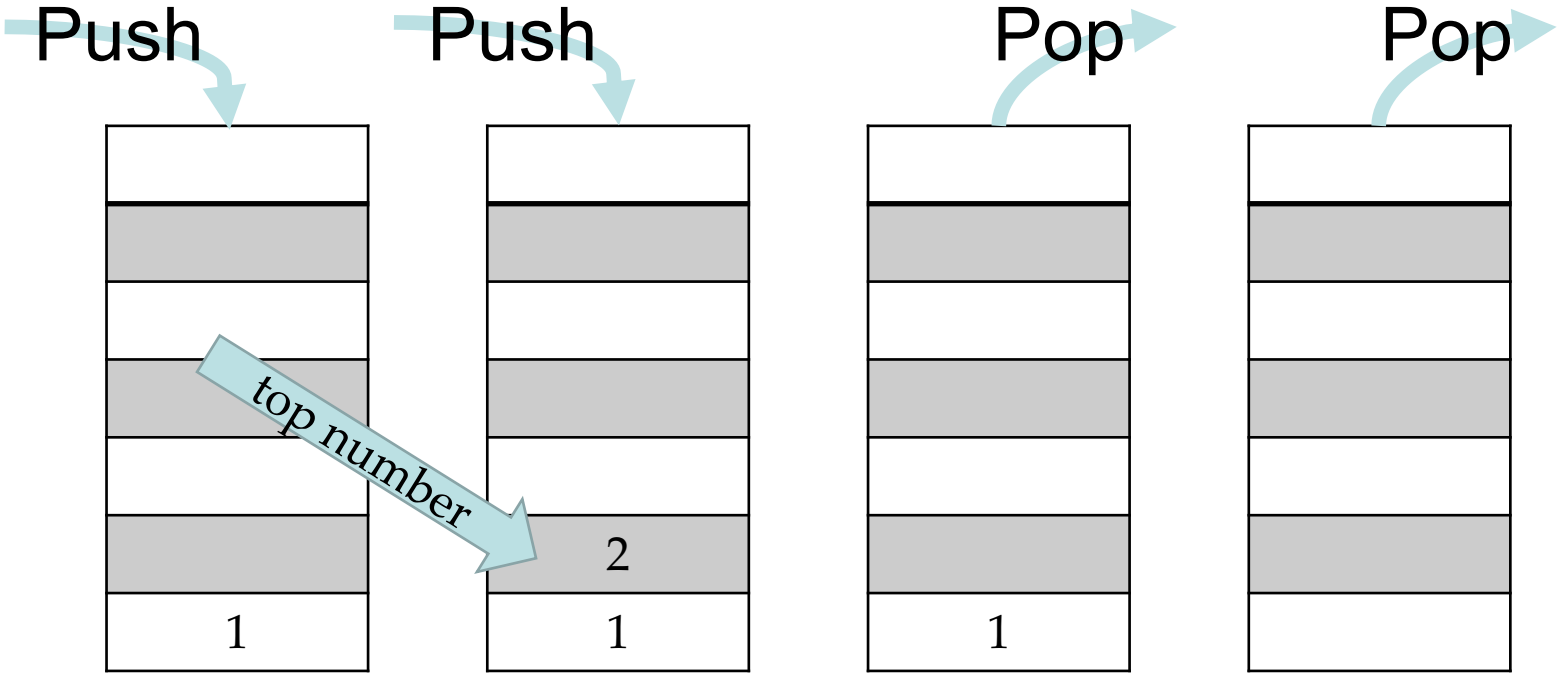


- Dividend: 0000…10000…0000 is interpreted as 4294967296 using 2's-complement.
- Divisor: ECX = 4
- Quotient result put in EAX, Remainder put in EDX,

Basic knowledge of stack

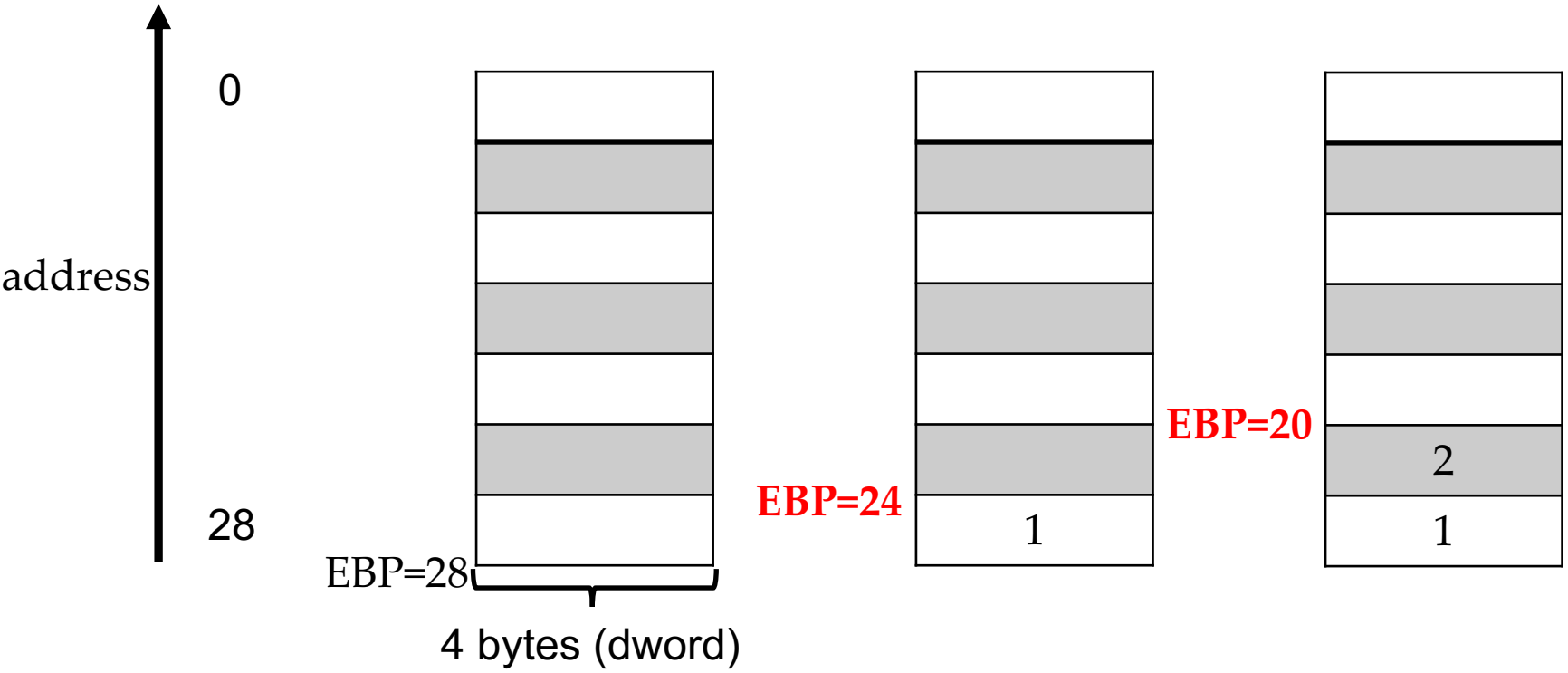
Stack: a list of data elements

- Pushnum : placing data **at the top** end of a stack
- Popnum: removing **top data** from top end of a stack
- A Last-In-First-Out (LIFO) data structure



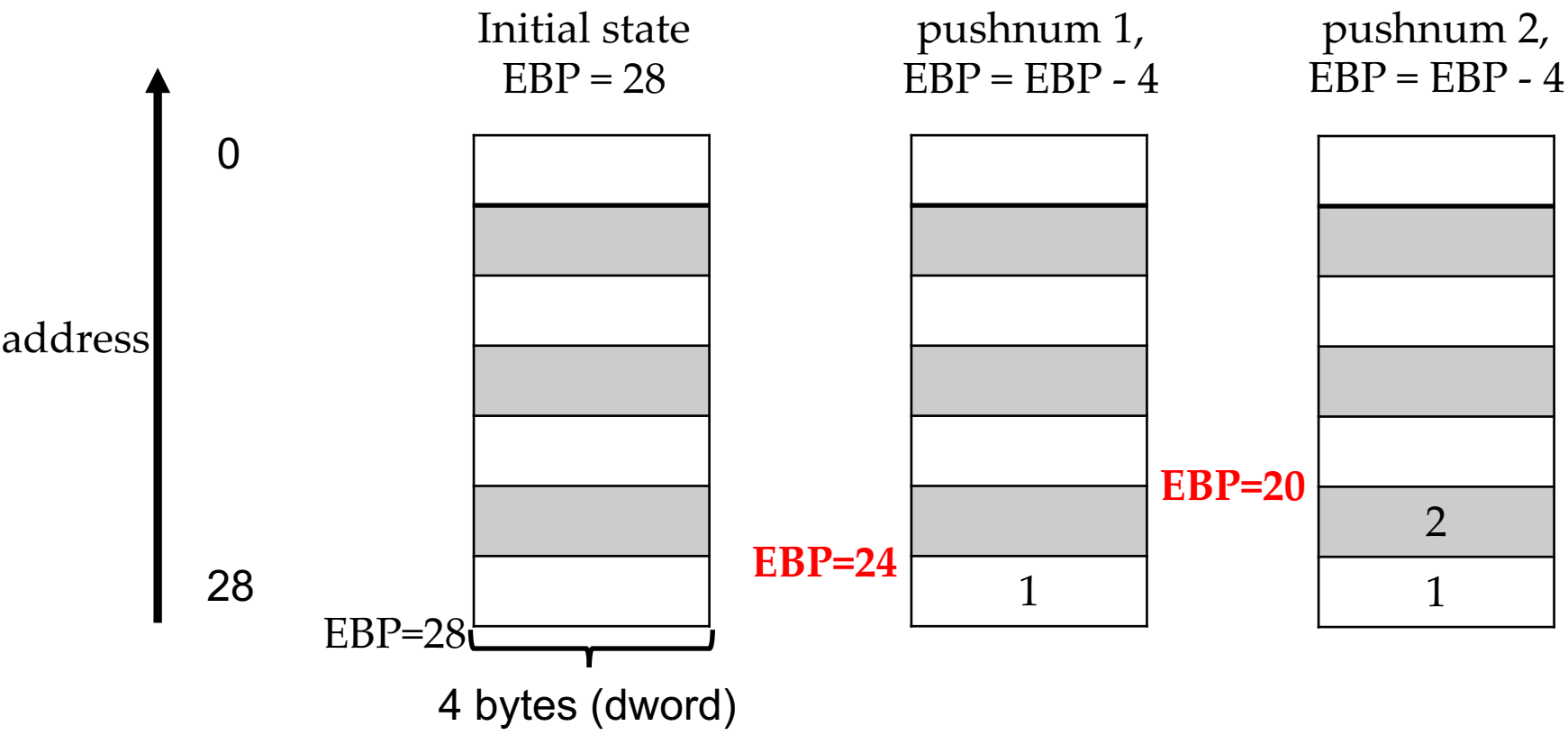
Stack Implementation

- EBP: initialize to largest address
  - EBP: top pointer, point to the address of the top number
- there are **different** implementations for maintaining top pointers



Stack Implementation: Pushnum

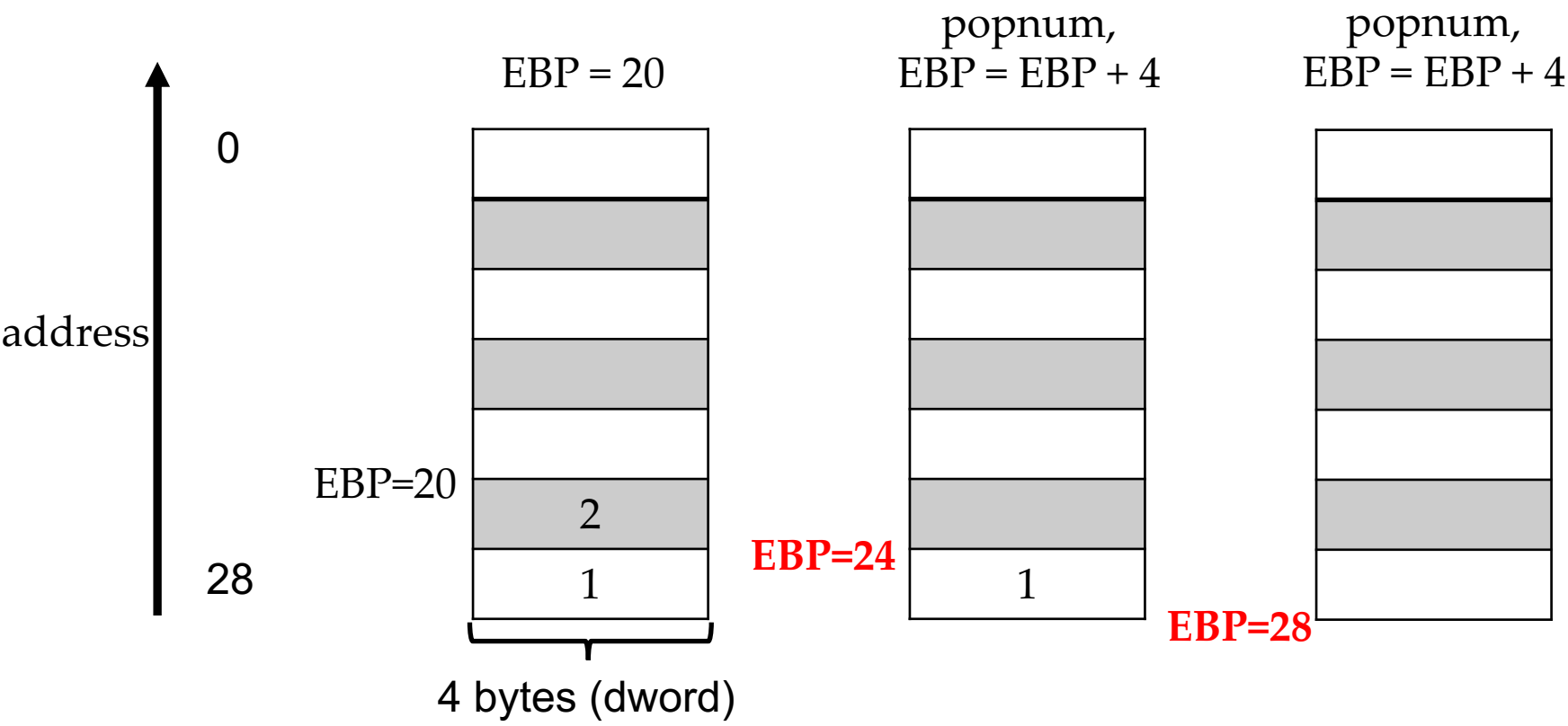
- Pushnum pseudo code:
  1) EBP = EBP – 4
  2) place the number into the top end of stack



Stack Implementation: Popnum

- Popnum pseudo code:
  1) EBP = EBP + 4

```
If we want to get the top
number, we need to get the top
number before the addition:
  1) Get the top number
  2) EBP = EBP + 4
```

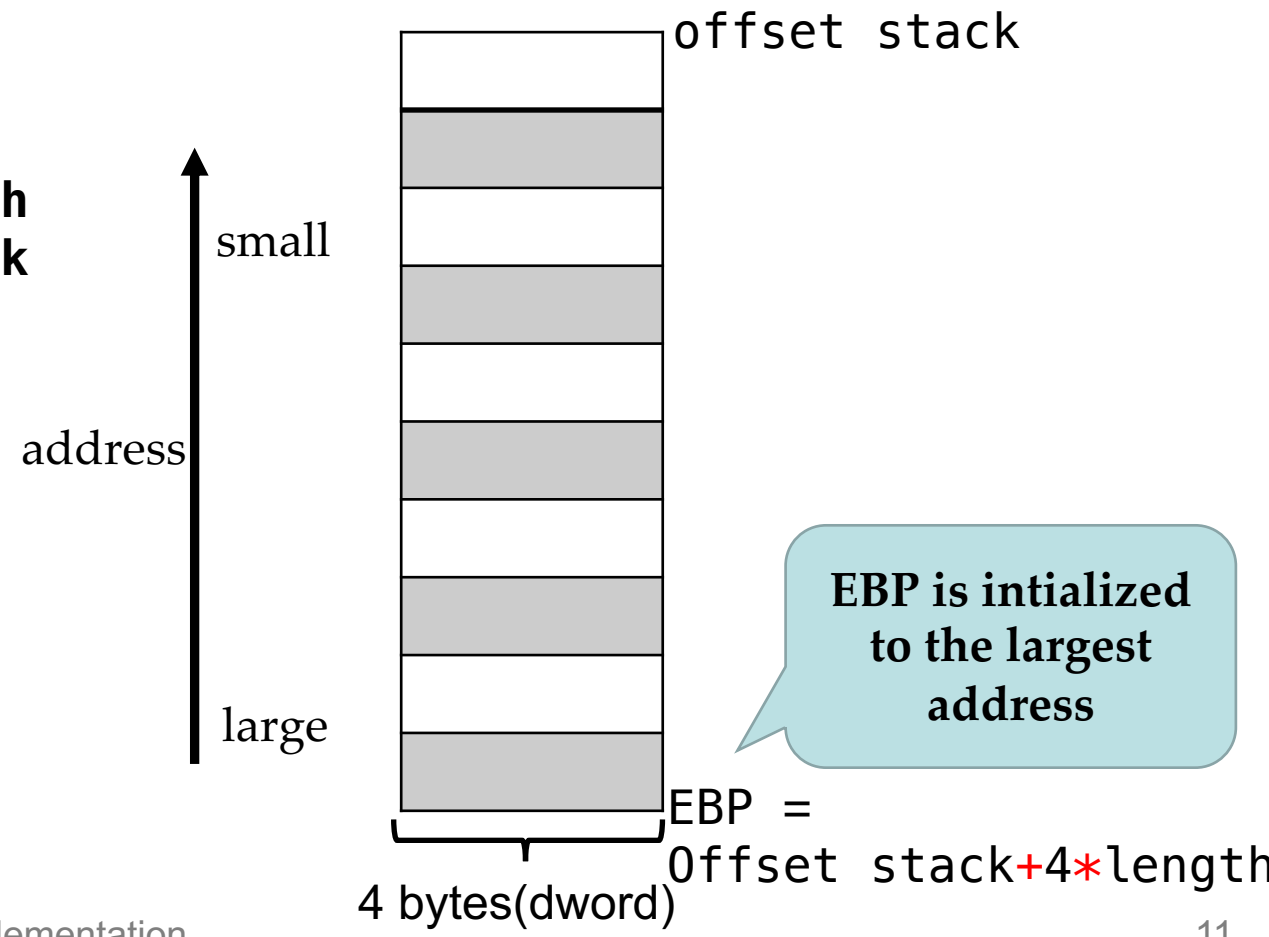


Assignment 2 Programming Exercise

- In addition to the processor stack, it may be convenient to maintain our own stack in programs. In this programming exercise, we are going to implement a stack using MASM IA-32 assembly language. In our implementation, the stack is allocated a fixed amount of memory space to store **at most ten** positive numbers of 32-bits (**dword**), and the stack grows toward **lower-numbered** address locations. In addition, the stack can be manipulated via the following functions:
  - pushnum: Input a **positive number** to push it onto the top of stack;
  - popnum: Input **0** to pop and print out the number from the top of the stack;
  - gettop: Input **-1** to print out the number on the top of the stack without popping it;
  - getsize: Input **-2** to print out the size of numbers that have been pushed into the stack;
  - showstack: Input **-3** to print out the contents of the stack.
- *Note: It is not allowed to define additional variables in ".data".*

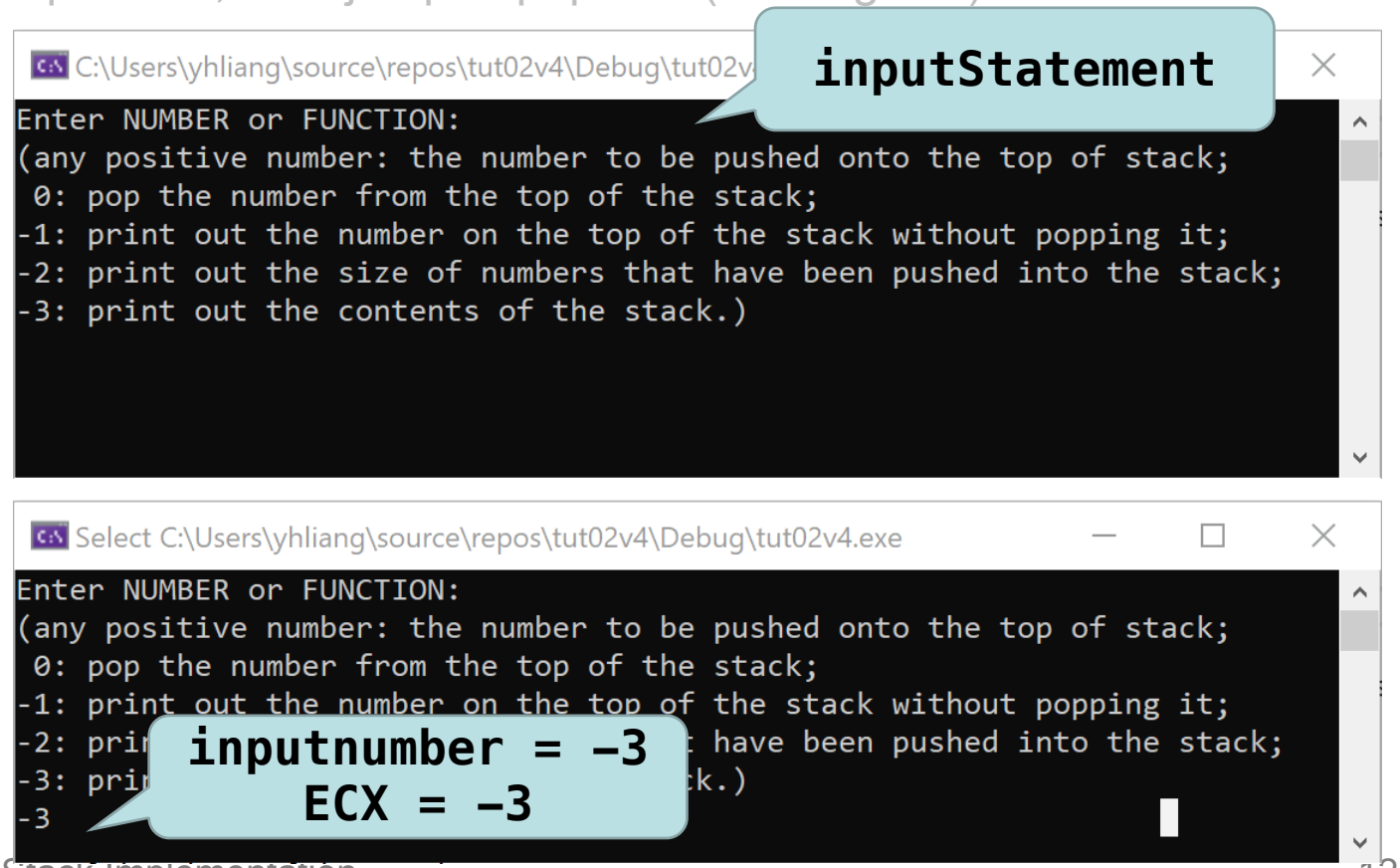
Tracing stack.asm: start

```
.data
stack dword 10 dup(1)
stacklength dword 10 ; stack length
…

.code
start:
mov EBP, 4
imul EBP, stacklength
add EBP, offset stack

jmp input
```



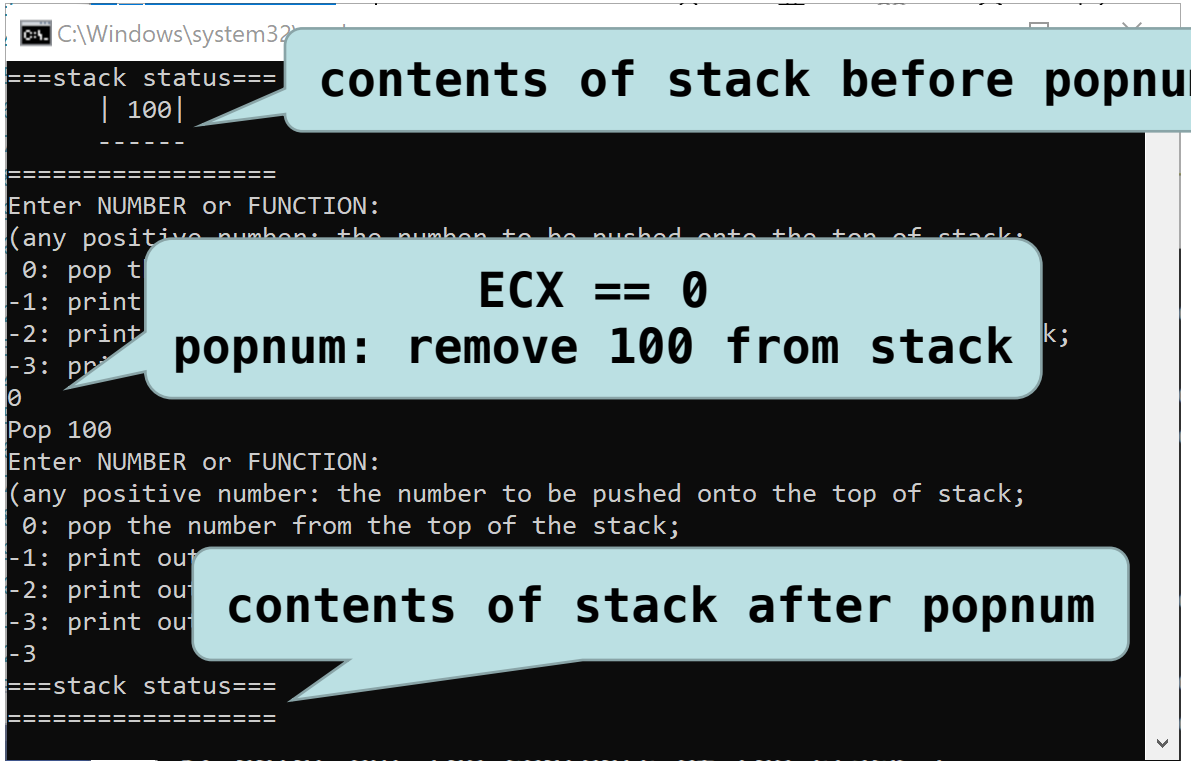
Tracing stack.asm: input (1/4)

```
input:
invoke crt_printf, addr inputStatement
invoke crt_scanf, addr numberFormat, addr inputnumber
mov ECX, inputnumber
; compare content of ECX with 0 (missing line)
; if content of ECX is equal to 0, then jump to popnum (missing line)
cmp ECX, -3
je showstack
jmp pushnum
```



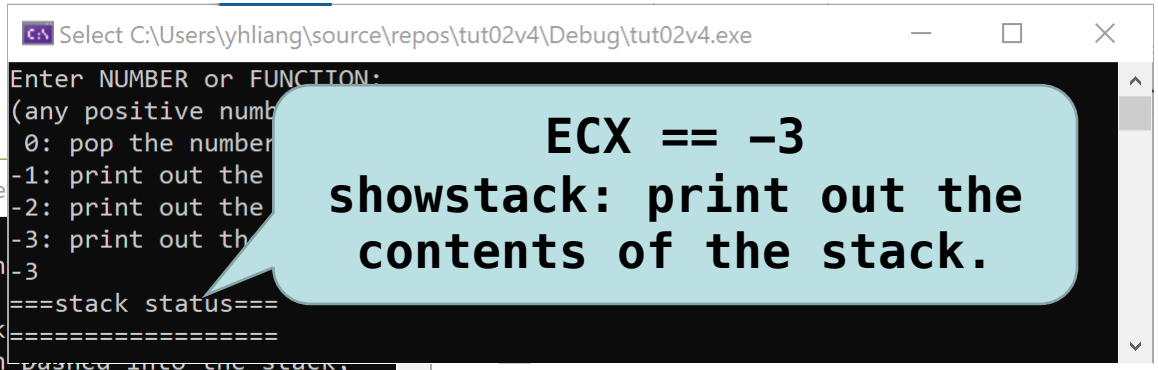
Tracing stack.asm: input (2/4)

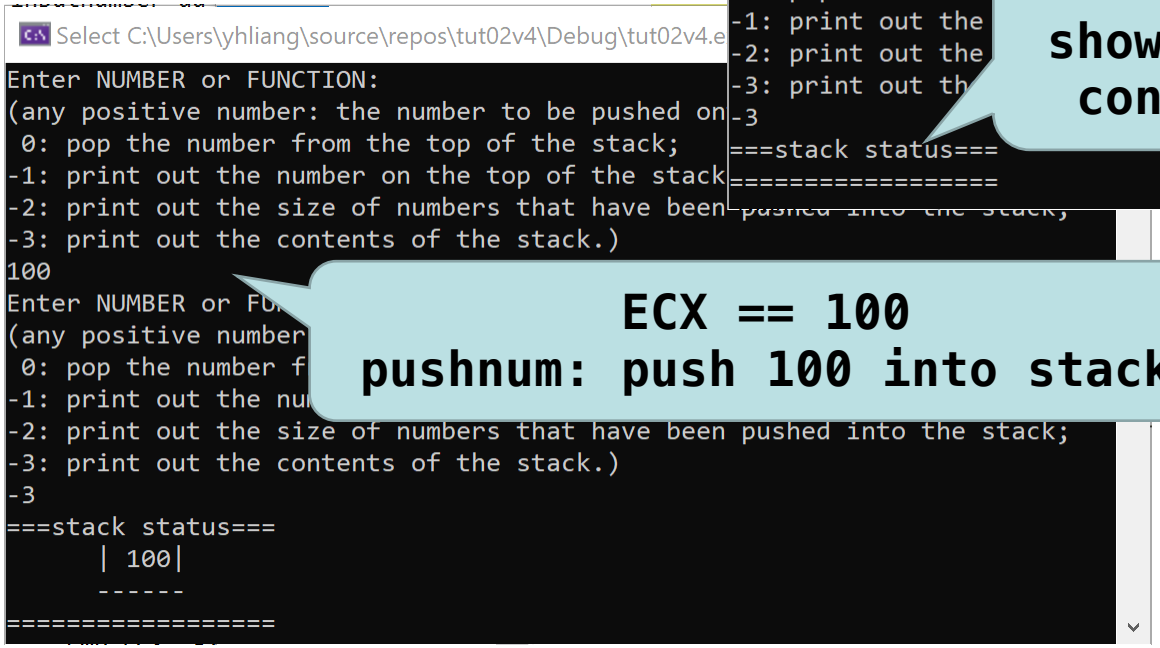
```
input:
invoke crt_printf, addr inputStatement
invoke crt_scanf, addr numberFormat, addr inputnumber
mov ECX, inputnumber
; compare content of ECX with 0 (missing line)
; if content of ECX is equal to 0, then jump to popnum (missing line)
cmp ECX, -3
je showstack
jmp pushnum
```



Tracing stack.asm: input (3/4)

```
input:
invoke crt_printf, addr inputStatement
invoke crt_scanf, addr numberFormat, addr inputnumber
mov ECX, inputnumber
; compare content of ECX with 0 (missing line)
; if content of ECX is equal to 0, then jump to popnum (missing line)
cmp ECX, -3
je showstack
jmp pushnum
```





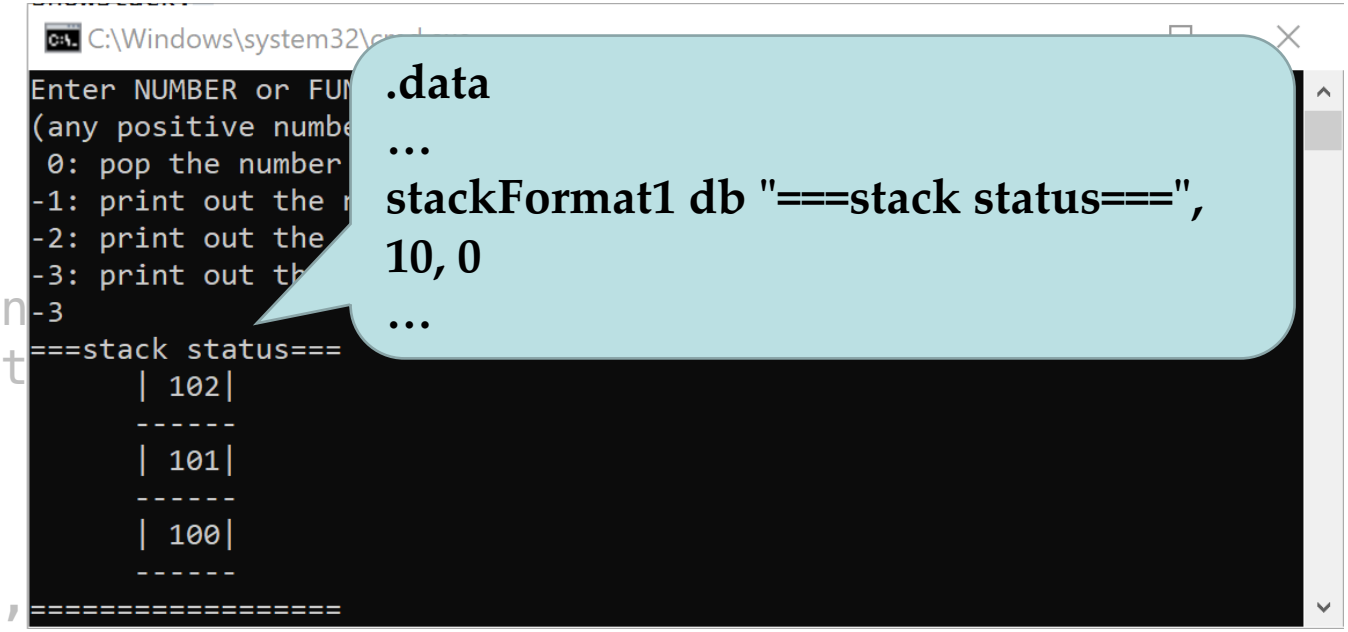
Tracing stack.asm: input (4/4)

- Three functions mentioned in input: pushnum, popnum, showstack
- Let's track showstack first
- Popnum and pushnum will be track in page 18

```
input:
invoke crt_printf, addr inputStatement
invoke crt_scanf, addr numberFormat, addr inputnumber
mov ECX, inputnumber
; compare content of ECX with 0 (missing line)
; if content of ECX is equal to 0, then jump to popnum (page 18) (missing line)
cmp ECX, -3
je showstack
jmp pushnum (page 18)
```

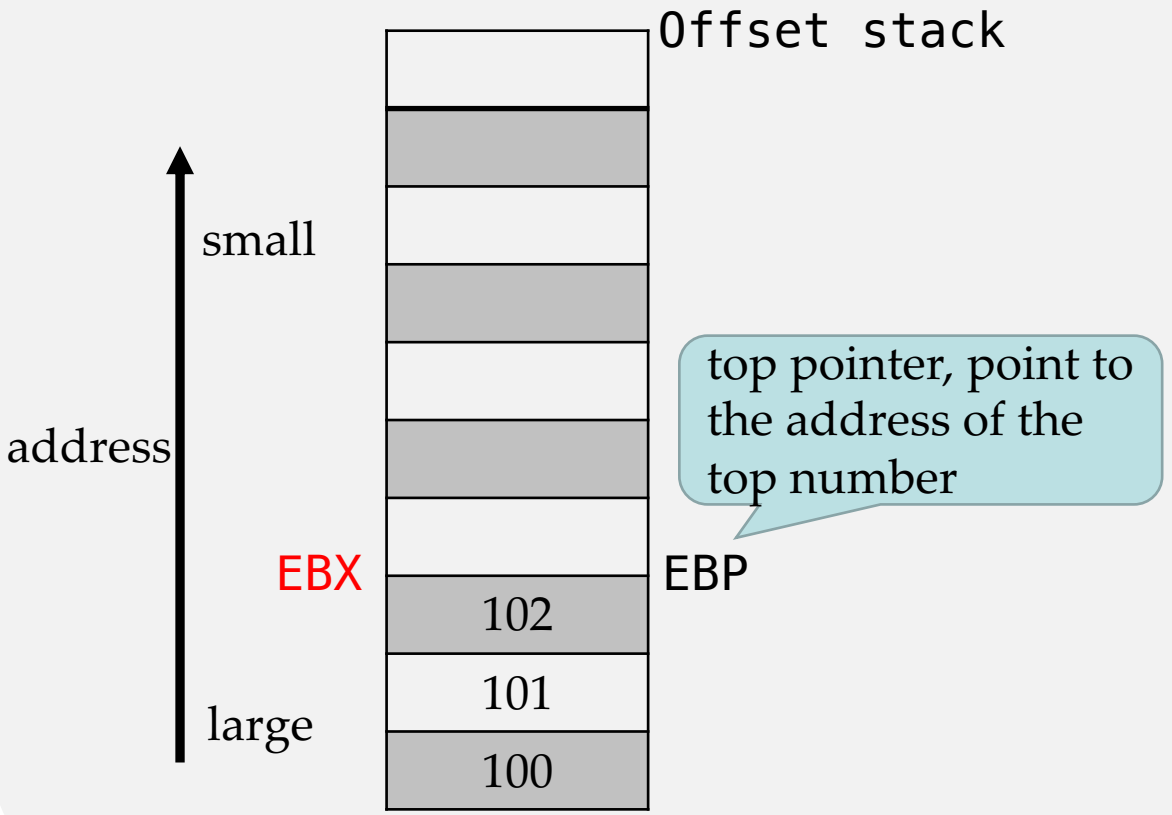
Tracing stack.asm: showstack (1/5)

```
showstack:
invoke crt_printf, addr stackFormat1
mov EBX, EBP
jmp showstackdata

showstackdata:
mov EAX, 4
imul EAX, stacklen
add EAX, offset st
cmp EBX, EAX
je showstackend
mov ECX, [EBX]
invoke crt_printf,
invoke crt_printf, addr stackFormat3
add EBX, 4
jmp showstackdata

showstackend:
invoke crt_printf, addr stackFormat4
jmp input
```



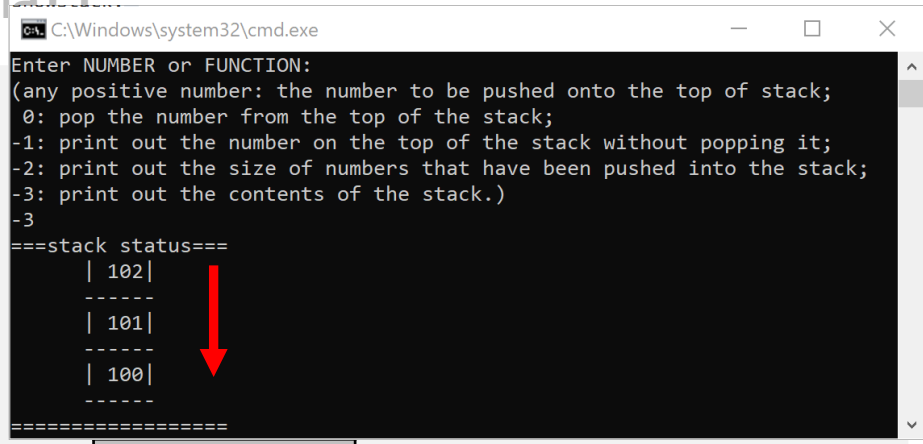
Tracing stack.asm: showstack (2/5)

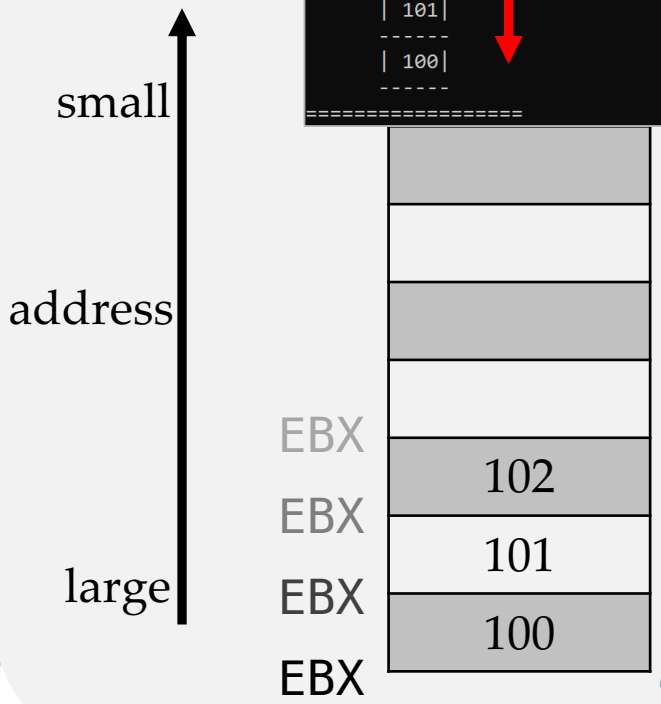
```
showstack:
invoke crt_printf, addr stackFormat1
mov EBX, EBP
jmp showstackdata

showstackdata:
mov EAX, 4
imul EAX, stacklength
add EAX, offset stack
cmp EBX, EAX
je showstackend
mov ECX, [EBX]
invoke crt_printf, add
invoke crt_printf, add
add EBX, 4
jmp showstackdata

showstackend:
invoke crt_printf, addr
jmp input
```



Tracing stack.asm: showstack (3/5)

```
showstack:
invoke crt_printf, addr stackFormat1
mov EBX, EBP
jmp showstackdata

showstackdata:
mov EAX, 4
imul EAX, stacklength
add EAX, offset stack
cmp EBX, EAX
je showstackend
mov ECX, [EBX]
invoke crt_printf, add
invoke crt_printf, add
add EBX, 4
jmp showstackdata

showstackend:
invoke crt_printf, addr
jmp input
```



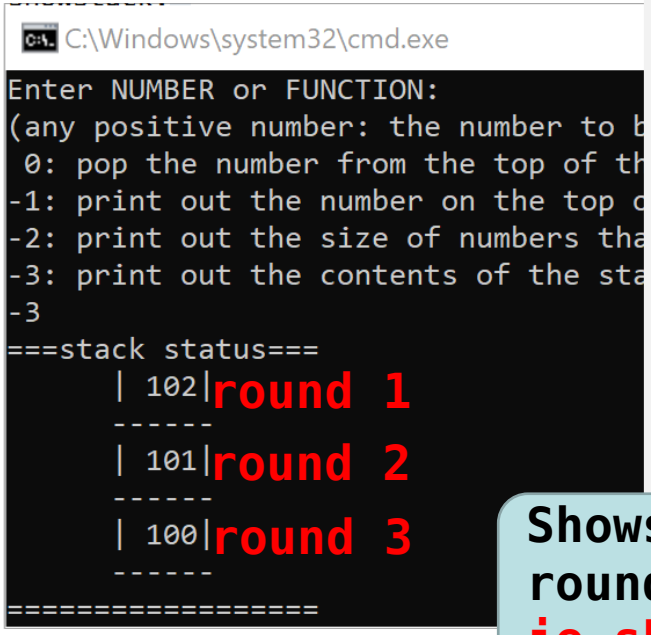


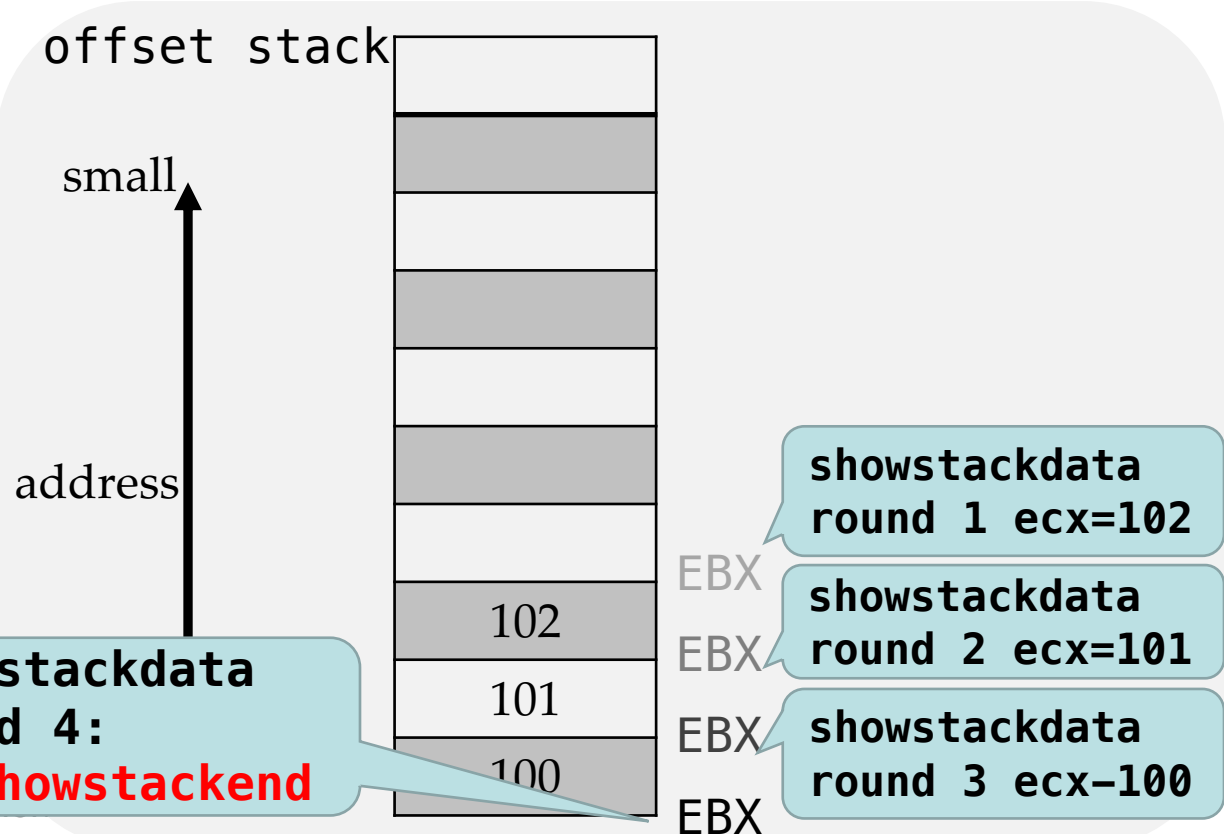
When EBX reach largest address, No content need to be printed.

EBX == Offset stack +4*length

Tracing stack.asm: showstack (4/5)

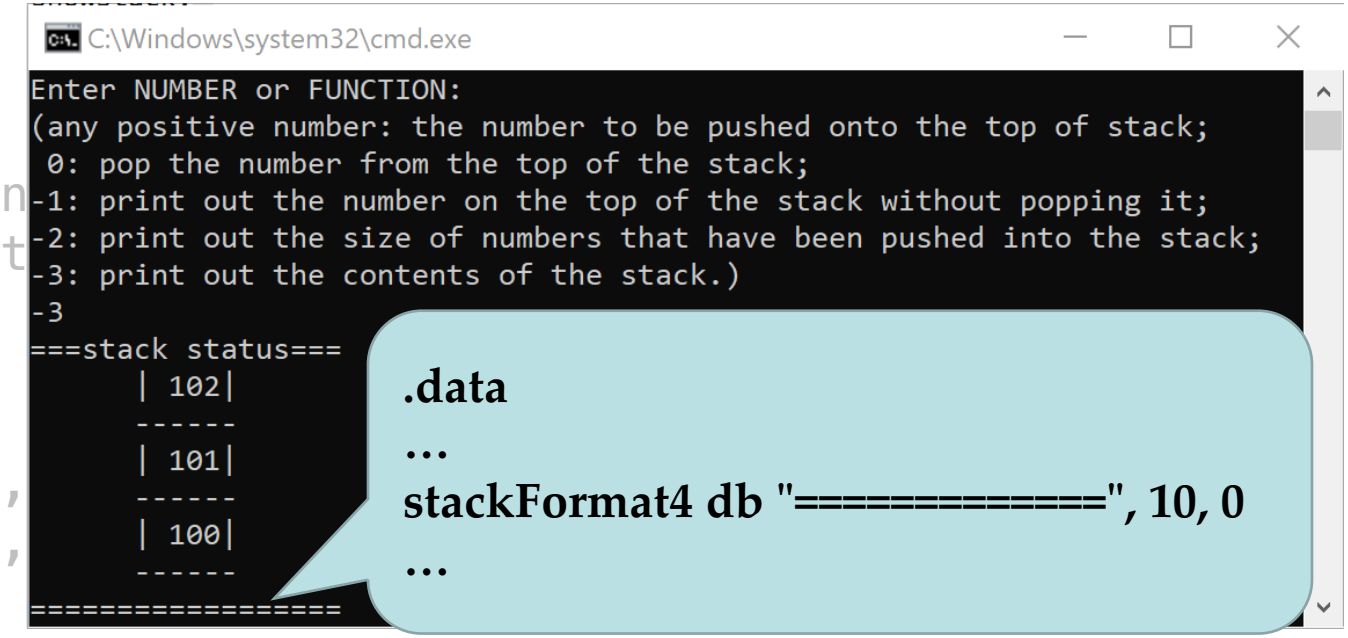
```
showstackdata:
...
cmp EBX, EAX
je showstackend
mov ECX, [EBX]
invoke crt_printf, addr stackFormat2, ECX
invoke crt_printf, addr stackFormat3
add EBX, 4
jmp showstackdata
```





Tracing stack.asm: showstack (5/5)

```
showstack:
invoke crt_printf, addr stackFormat1
mov EBX, EBP
jmp showstackdata

showstackdata:
mov EAX, 4
imul EAX, stacklen
add EAX, offset st
cmp EBX, EAX
je showstackend
mov ECX, [EBX]
invoke crt_printf,
invoke crt_printf,
add EBX, 4
jmp showstackdata

showstackend:
invoke crt_printf, addr stackFormat4
jmp input
```



Tracing stack.asm: remaining parts

```
pusherror:
invoke crt_printf, addr pushErrorStatement
jmp exitprogram

poperror:
invoke crt_printf, addr popErrorStatement
jmp exitprogram

isempty:
invoke crt_printf, addr outputFormatForPrintIsEmpty
jmp input

exitprogram:
invoke ExitProcess, NULL
```

Print out alert message when pushing a number into the full stack. (exercise 2)

Print out alert message when poping a number from the empty stack. (exercise 2)

Print out the alert message for function gettop (exercise 3)

Exit the program

Assignment 2 Programming Exercise

- In addition to the processor stack, it may be convenient to maintain our own stack in programs. In this programming exercise, we are going to implement a stack using MASM IA-32 assembly language. In our implementation, the stack is allocated a fixed amount of memory space to store **at most ten** positive numbers of 32-bits (**dword**), and the stack grows toward **lower-numbered** address locations. In addition, the stack can be manipulated via the following functions:
  - pushnum: Input a **positive number** to push it onto the top of stack;
  - popnum: Input **0** to pop and print out the number from the top of the stack;
  - gettop: Input **-1** to print out the number on the top of the stack without popping it;
  - getsize: Input **-2** to print out the size of numbers that have been pushed into the stack;
  - showstack: Input **-3** to print out the contents of the stack.

  Note: It is not allowed to define additional variables in ".data".

Hints for Exercise 1

- **Exercise 1 (30 pts)**
  - Complete the provided MASM IA-32 assembly program named **stack.asm** to implement a stack. There are **six** "missing lines" in total.

```
input:
invoke crt_printf, addr inputStatement
invoke crt_scanf, addr numberFormat, addr inputnumber
mov ECX, inputnumber
; compare content of ECX with 0 (missing line)
; if content of ECX is equal to 0, then jump to popnum (missing line)
cmp ECX, -3
je showstack
jmp pushnum

pushnum:
; decrease the top pointer by 4 (missing line)
; push the inputnumber into stack in memory (missing line)
jmp input

popnum:
; get the top data of in the stack in memory, and load it to ECX (missing line)
invoke crt_printf, addr outputFormatForPop, ECX
; increase the top buffer by 4 (missing line)
jmp input
```

```
Pushnum:
1)  EBP = EBP – 4
2)  place the number into
    the top end of stack
```

```
If we want to get the top number,
we need to get the top number before the addition
```

```
Popnum: 1)Do the addition
```

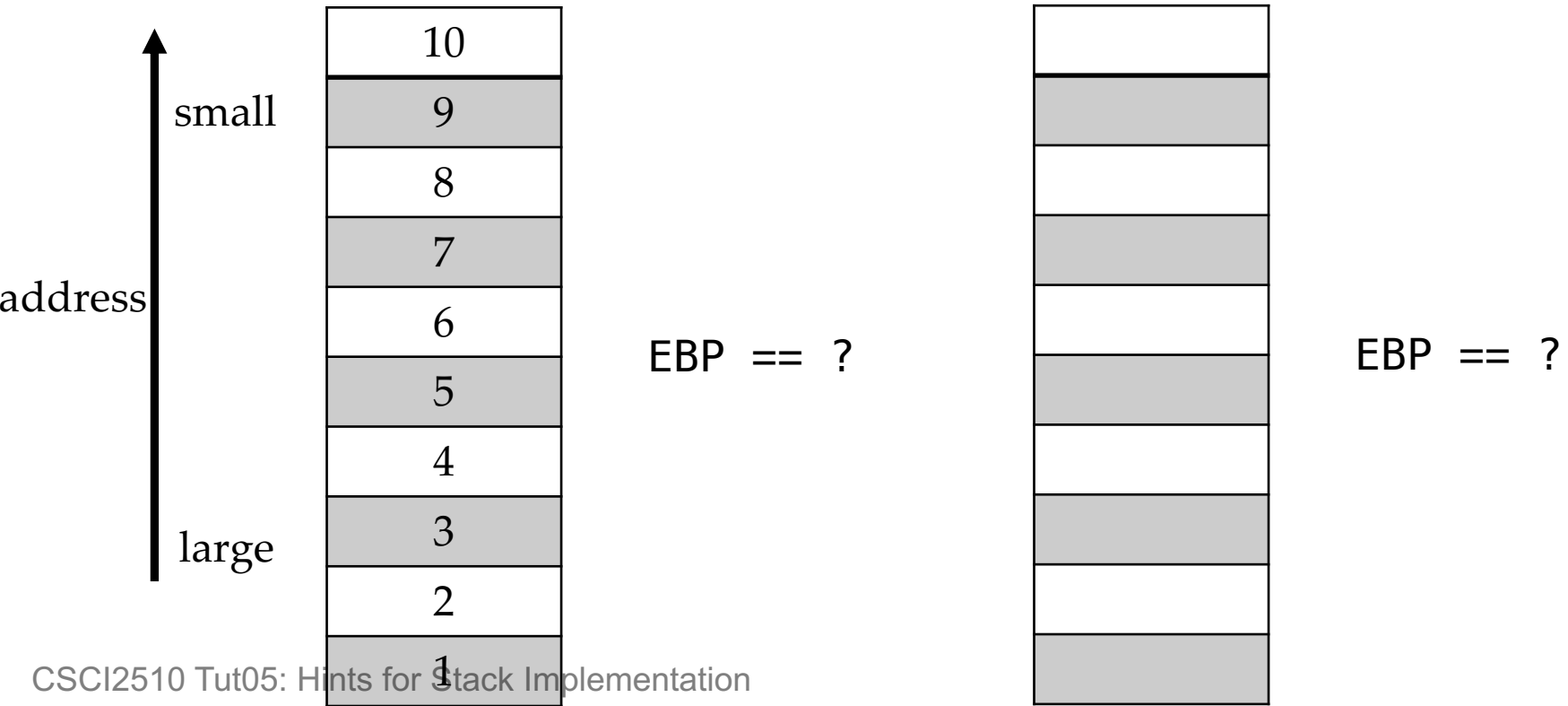
Hints for Exercise 2 (1/2)

- **Exercise 2 (10 pts)**
  - Our stack is only allocated a fixed amount of space in the memory. Therefore, it is important to avoid pushing an item onto the stack when the stack has reached its maximum size. Also, it is important to avoid attempting to pop an item off an empty stack. Revise the program **stack.asm** to handle the following two possible errors by showing alert messages as follows:
    - **Possible error 1:** Push a number into the stack when the stack is full.
    - **Possible error 2:** Pop a number from the stack when the stack is empty.

Hints for Exercise 2 (2/2)

- **Possible error 1:** Push a number into the stack when the stack is full
- **Possible error 2:** Pop a number from the stack when the stack is empty.
  - When stack is full/empty, what is the content of EBP?

`offset stack`



Hints for Exercise 3 (1/2)

- **Exercise 3 (10 pts)**
  - Implement the following two new functions gettop and getsize in the program **stack.asm**:
  - gettop: Print out the number on the top of the stack without popping it.
    - Similar to popnum, but not remove the data
    - When stack is empty, what is the content of EBP?
  - getsize: Print out the size of numbers that **have been pushed** into the stack.

Hints for Exercise 3 (2/2)

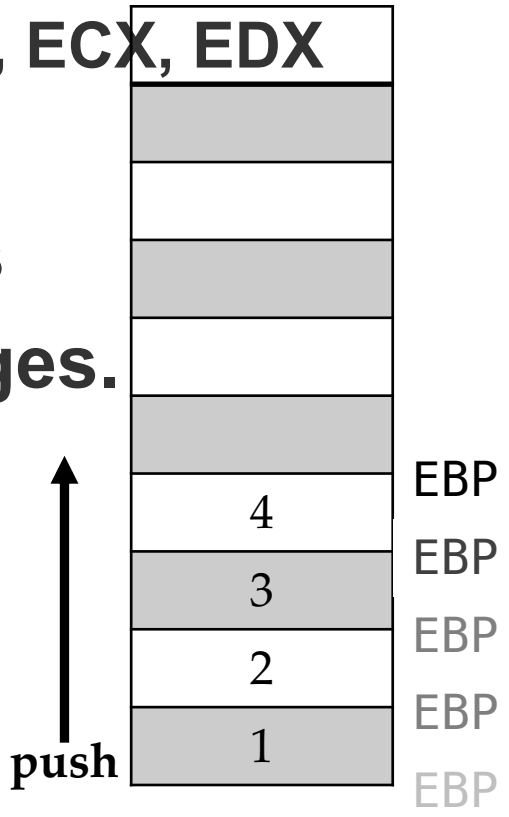
- **Exercise 3 (10 pts)**
  - Implement the following two new functions gettop and getsize in the program **stack.asm**:
  - gettop: Print out the number on the top of the stack without popping it.
    - Similar to popnum, but not remove the data
    - When stack is empty, what is the content of EBP?
  - getsize: Print out the size of numbers that **have been pushed** into the stack.
    - What is the relationship between the content of EBP and the size of stack
    - Use what arithmetic instructions to calculate the relationship.

Other Hints for Stack Implementation

- Use debug methods (Tut02) to see the content of registers.
  - crt_printf changes the value of EAX, ECX, EDX

- Draw the memory status pictures

to see how top pointer (EBP) changes.



summary

- Review of idiv Instruction
- Review of Stack Basics
- Tracking stack.asm
- Hints for Stack Implementation (Assignment 2)