香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Tutorial 04: Stack and Queue Implementations

Yuhong LIANG

yhliang@cse.cuhk.edu.hk

Outline

- More MASM instructions
- Queue Implementation
- Stack implementation
  - Hints for programming exercise in Assignment 2.

More MASM instructions (1/4)

mov: data transfer instruction

- **mov EAX, 4** ; move value 4 to EAX
- **mov EBX, EAX** ; move the content of EAX into EBX
- **mov [EBX] , EAX** ; move the content of EAX to the memory pointed by the content of EBX
- ...

add/sub/imul: data arithmetic instructions

- **add EAX, 4** ; EAX = EAX + 4
- **add EBX, EAX** ; EBX = EBX + EAX
- **add EBX, LOC** ; EBX = EBX + content of memory pointed by address LOC
- ...

More MASM instructions (2/4)

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)

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

More MASM instructions (3/4)

jmp: control flow instruction

- **jmp code1** ; jump to the code1

cmp: comparison instruction

- **cmp eax, ebx** ; compare the content of eax, ebx → if eax – ebx = 0, Z = 1

je : control flow instruction

- usually used with cmp instruction
- **je code1** ; if Z = 1 (imply eax = ebx), then jump to the CODE1

More MASM instructions (4/4)

jmp: control flow instruction

- **jmp code1** ; jump to the code1

cmp: comparison ir

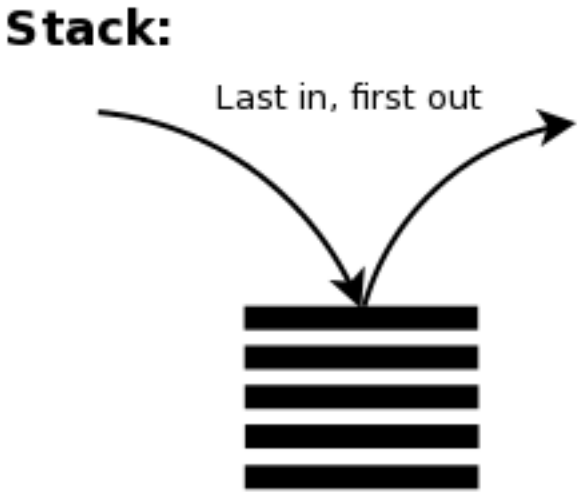
- **cmp eax, ebx** ; compa
  eax – ebx = 0, Z = 1

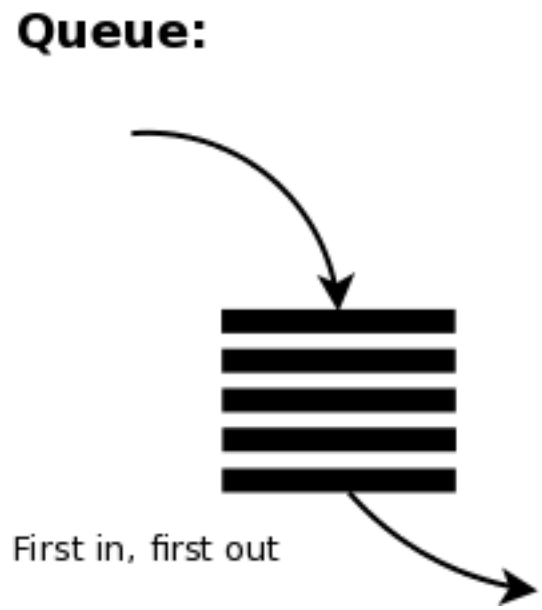
je : control flow inst

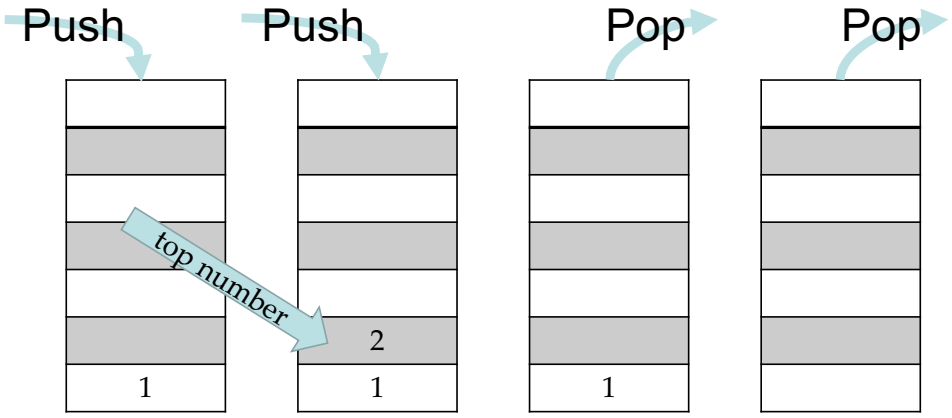
- usually used with cmp
- **je code1** ; if Z = 1 (imp
  CODE1

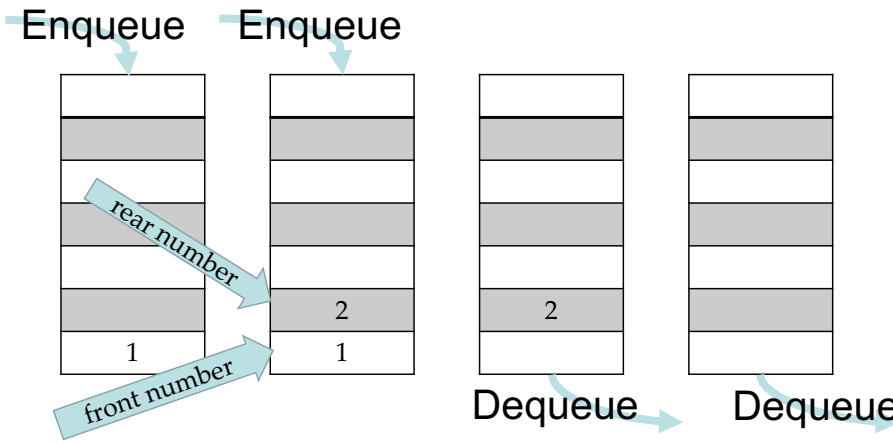
```
.code

input:
        cmp ECX, -3
        je showstack
        jmp pushnum

pushnum:
        mov EAX, 0
        jmp input

showstack:
        mov EAX, 1
        jmp input
```

Difference of Queue and Stack





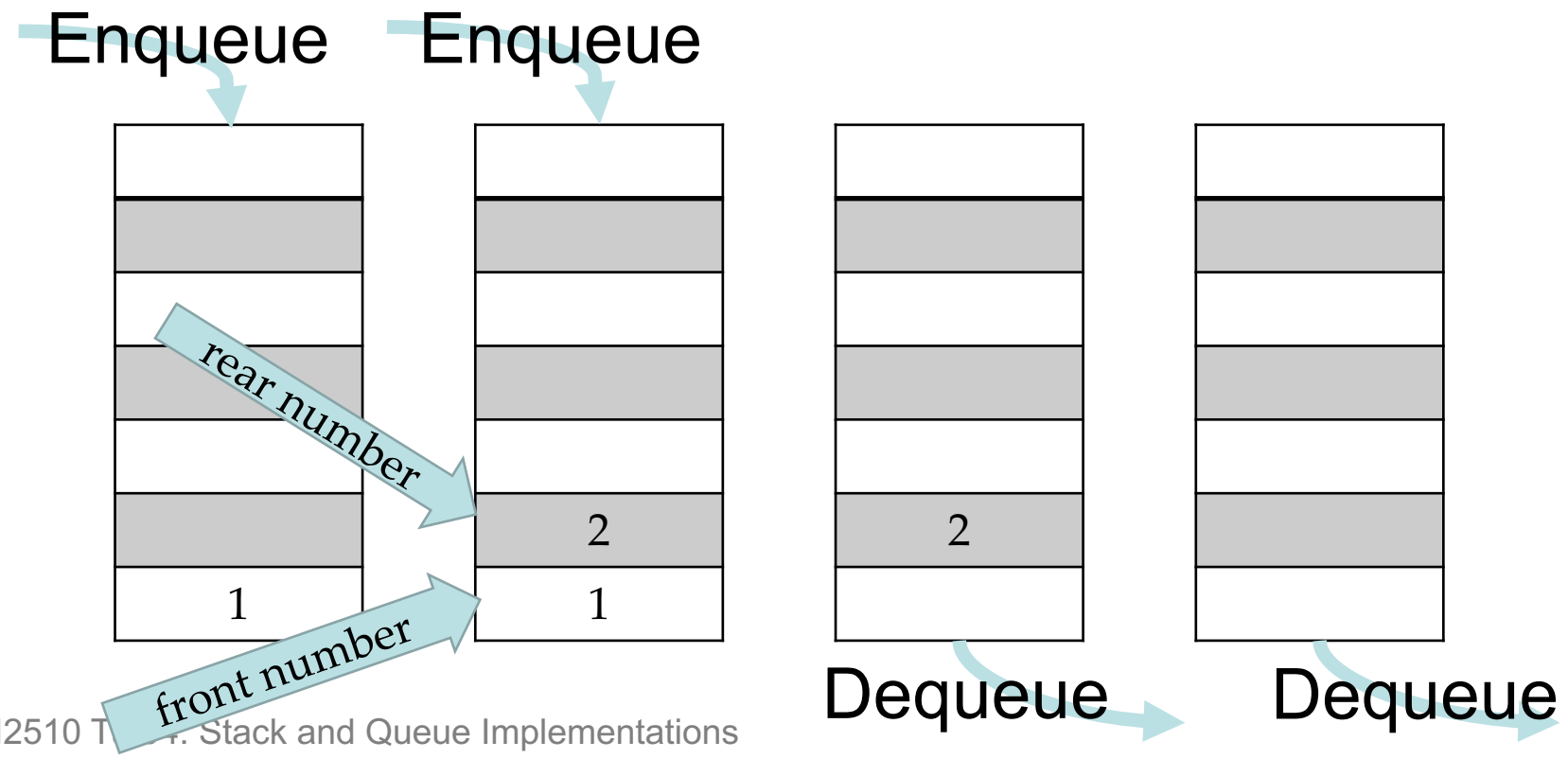




Basic knowledge of queue

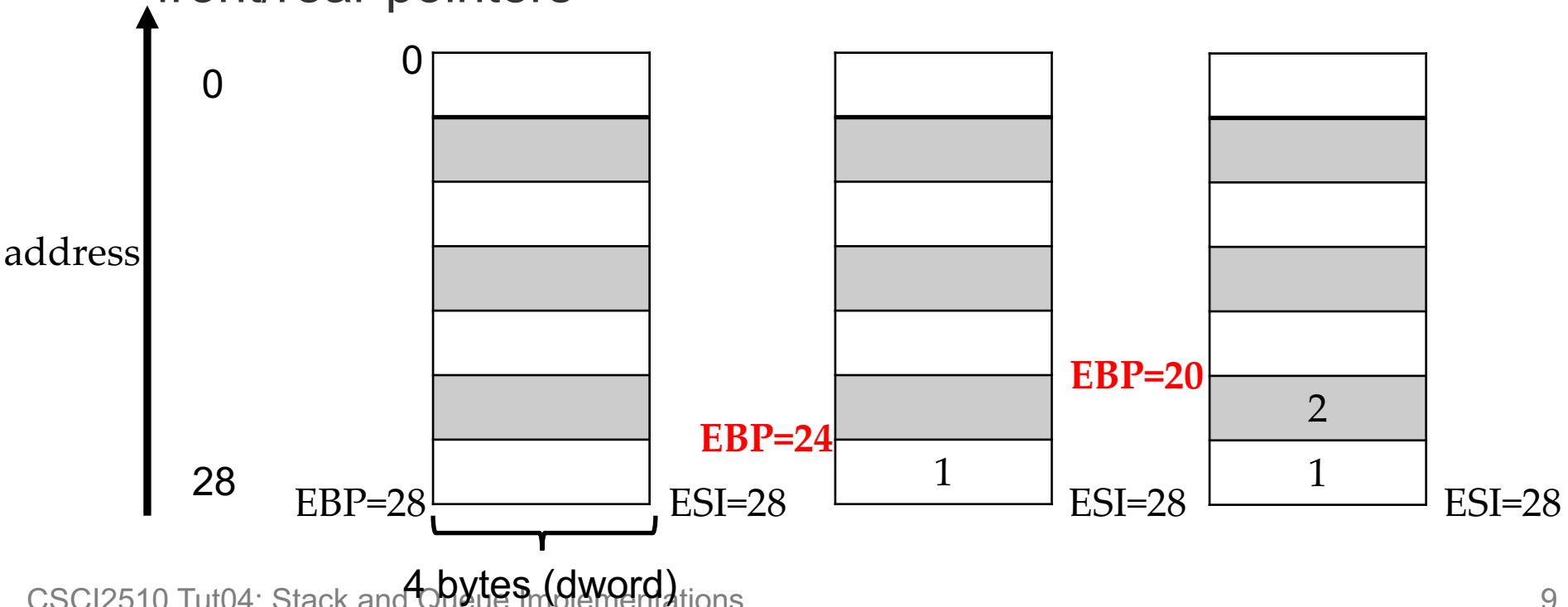
Queue: a list of data elements

- Enqueue : placing data into the **rear end** of queue
- Dequeue: remove data from the **front end** of queue
- A First-In-First-Out (FIFO) data structure



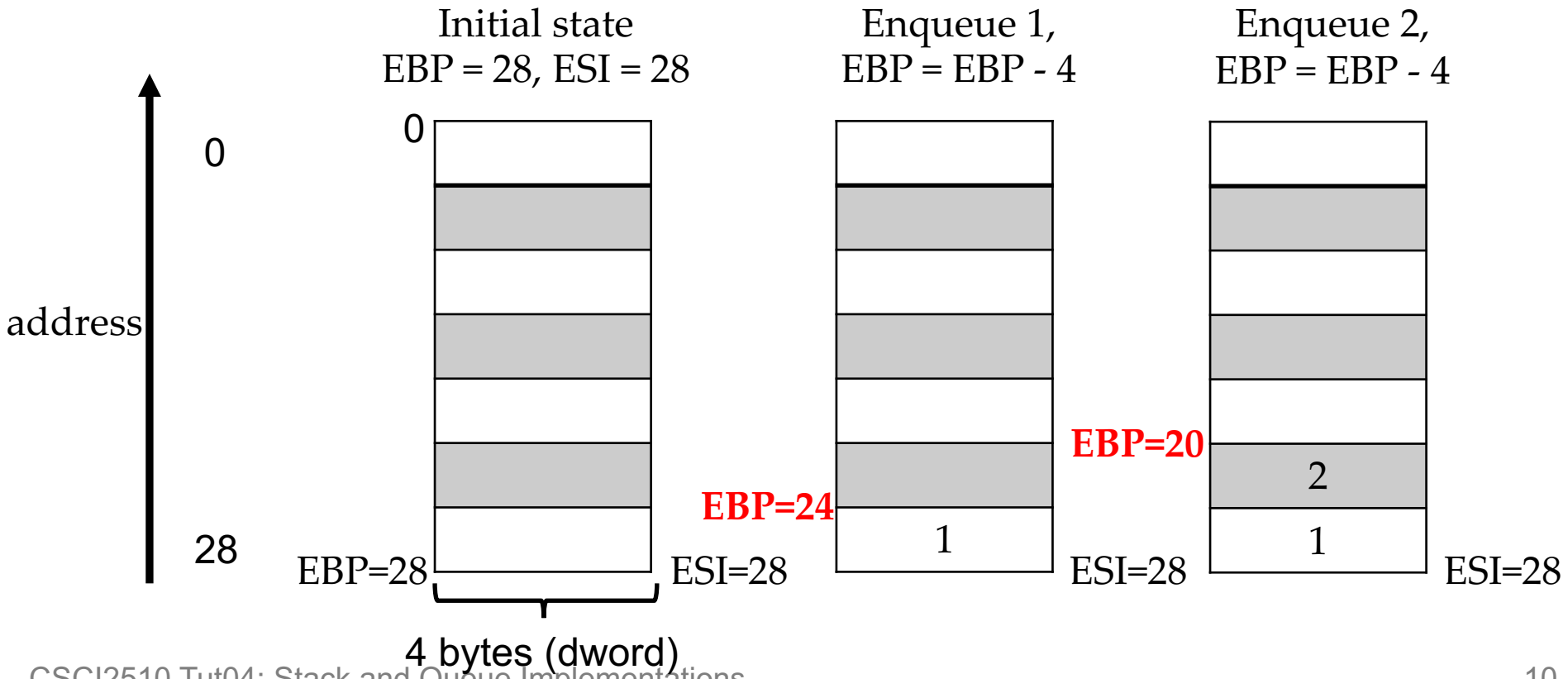
Queue Implementation

- EBP/ESI: initialized to largest memory address
  - EBP: rear pointer, point to the address of **the rear number**
  - ESI: front pointer, point to the address of **the last number (i.e., last front number) that has been dequeued**
  - there are **different** implementations for maintaining front/rear pointers



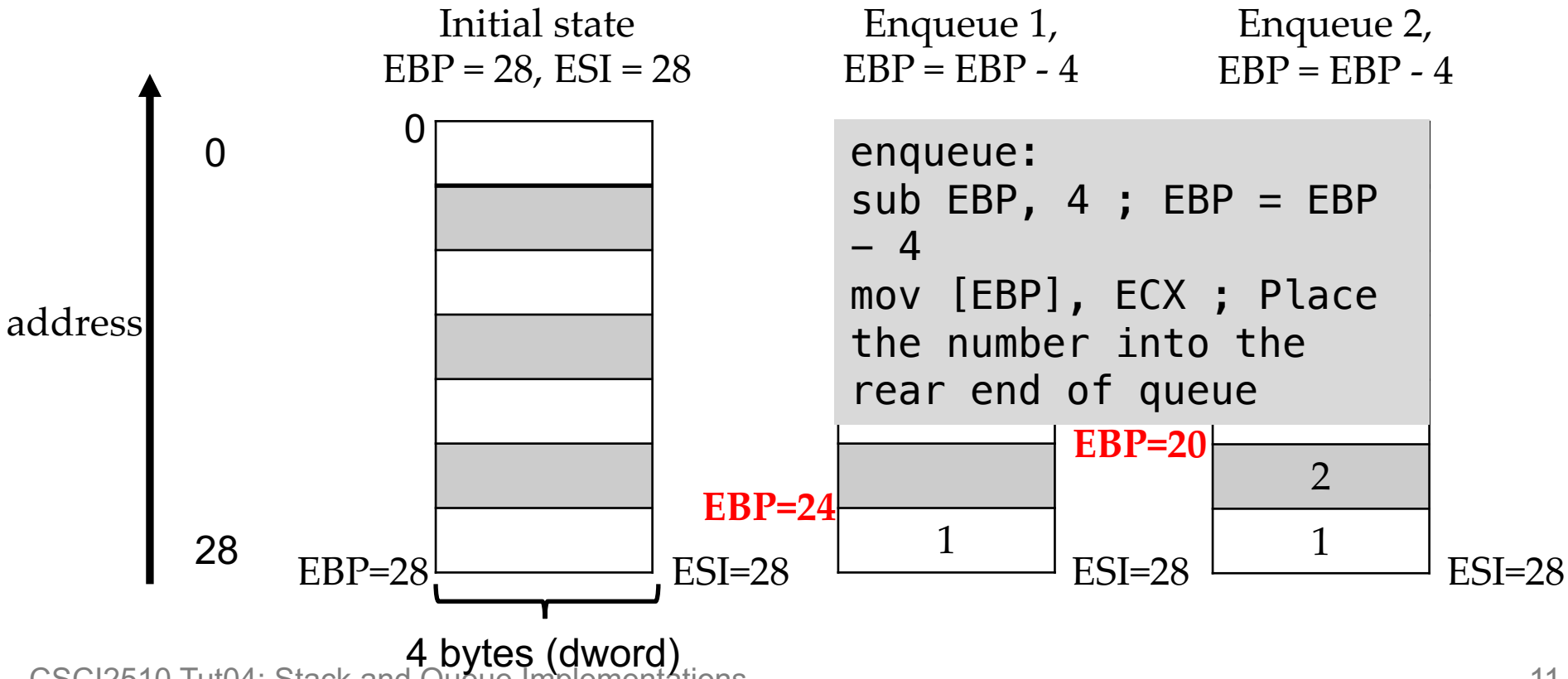
Queue Implementation: Enqueue (1/2)

- Enqueue pseudo code:
  1) EBP = EBP – 4
  2) Place the number into the rear end of queue



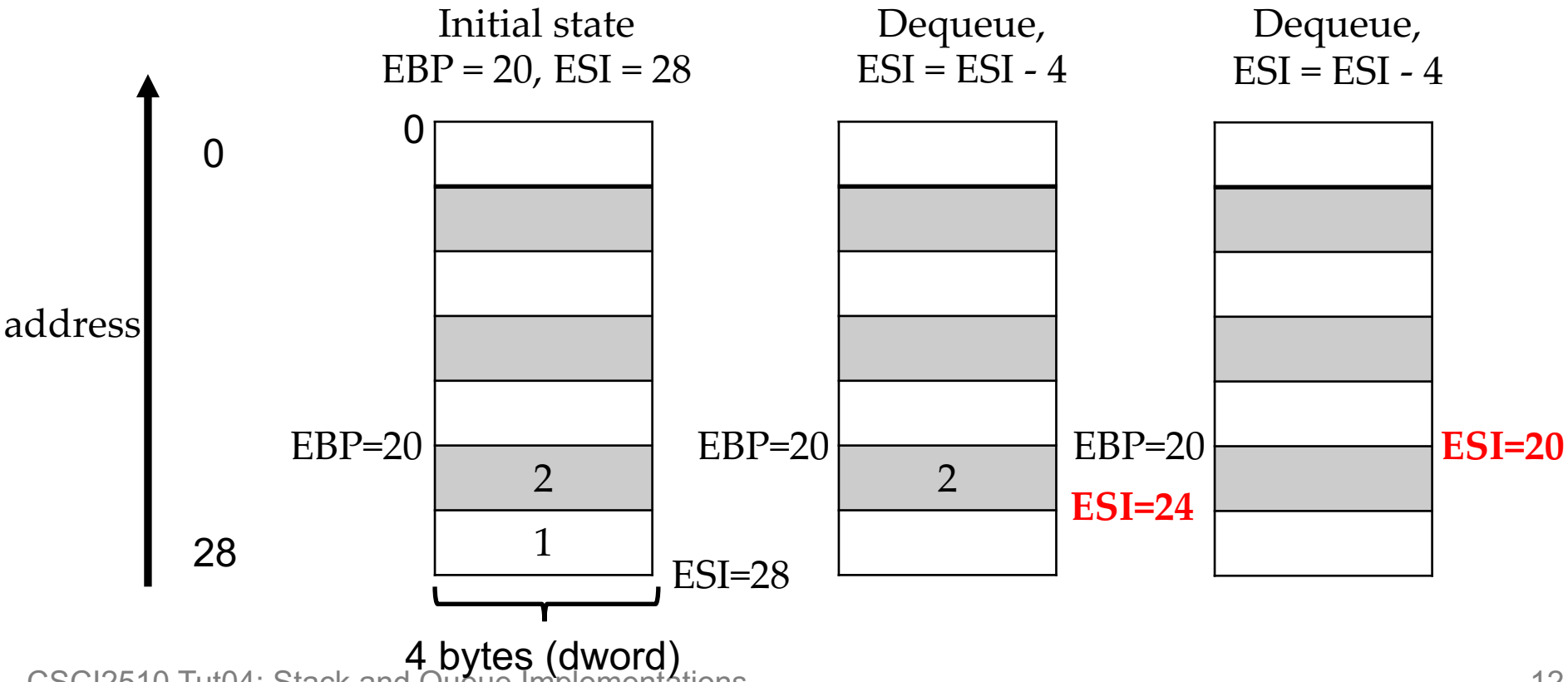
Queue Implementation: Enqueue (2/2)

- Enqueue pseudo code:
  1) EBP = EBP – 4
  2) Place the number into the rear end of queue



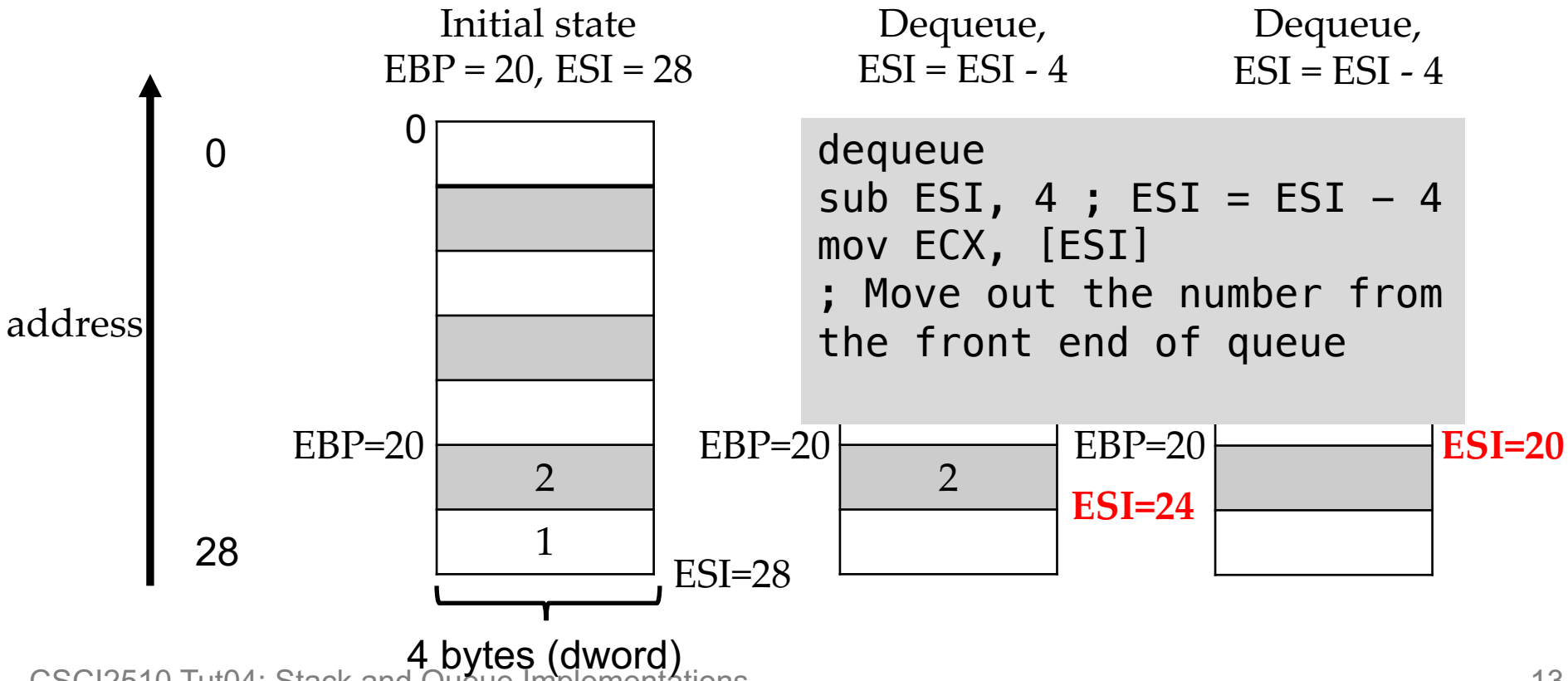
Queue Implementation: Dequeue (1/2)

- Dequeue pseudo code:
  1) ESI = ESI – 4
  2) Move out the number from the front end of queue



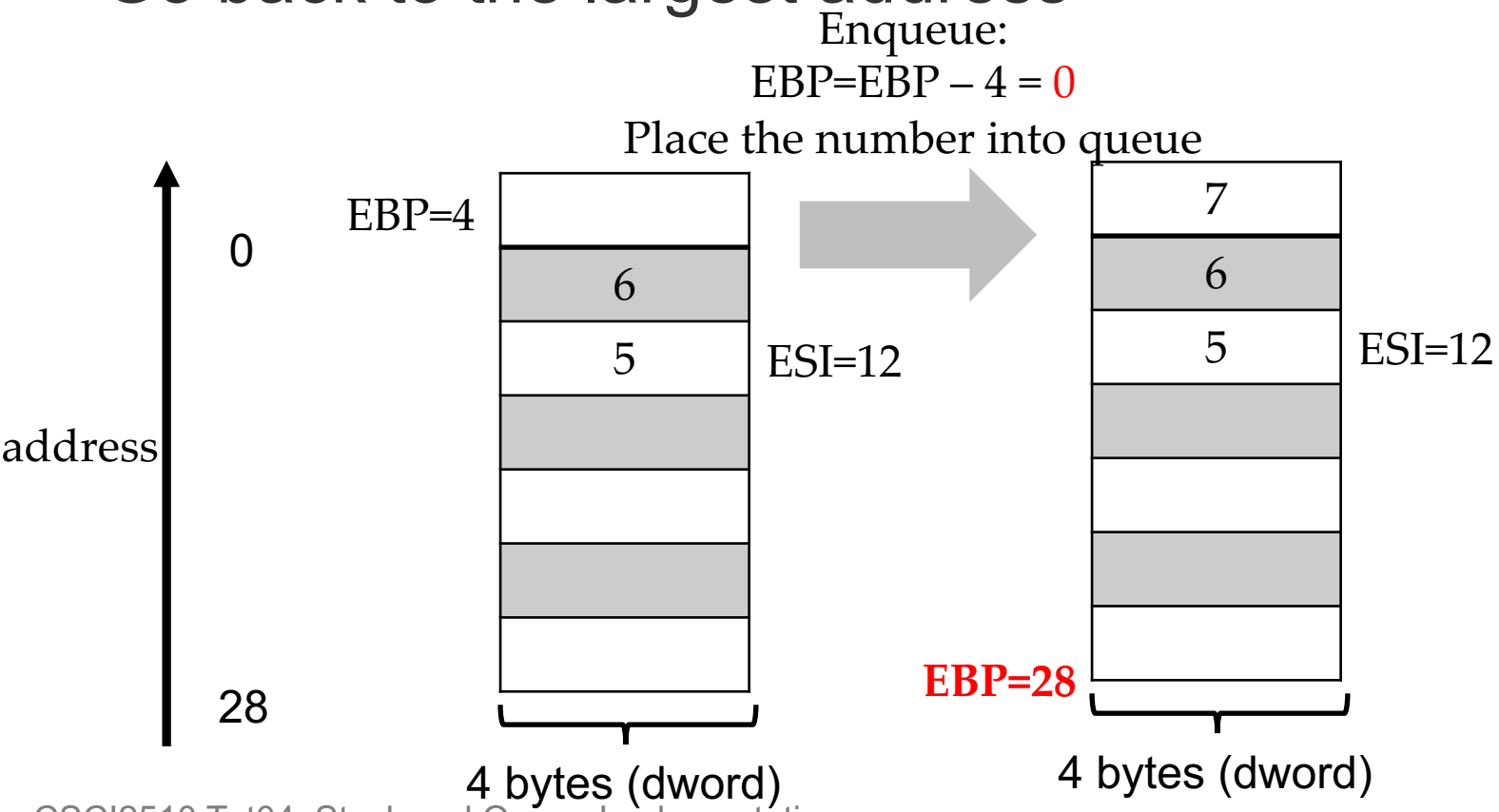
Queue Implementation: Dequeue (2/2)

- Dequeue pseudo code:
  1) ESI = ESI – 4
  2) Move out the number from the front end of queue



Queue Implementation: Circular (1/2)

- EBP/ESI reach the minimum address
  - There are still space in the larger addresses
  - The front number is in the space of larger addresses
- Go back to the largest address

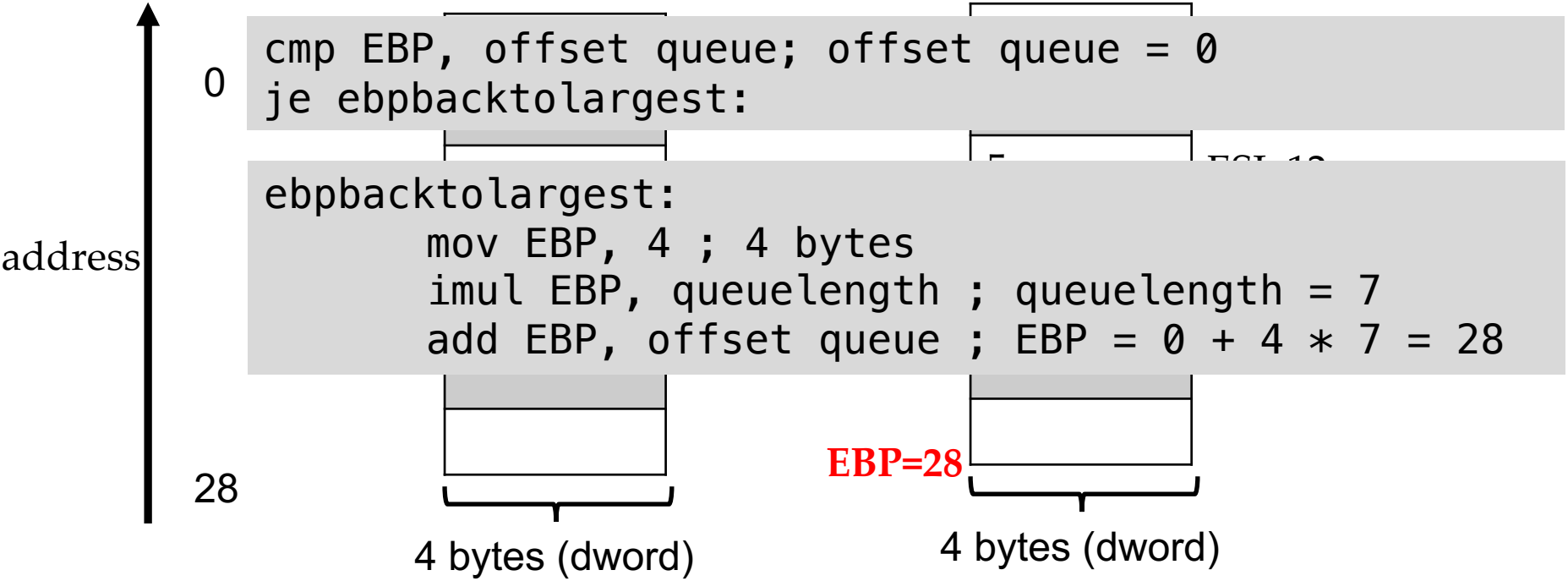


Queue Implementation: Circular (2/2)

- EBP/ESI reach the minimum address
- Go back to the largest address pseudo code:
  1) Compared with minimum address
  2) EBP = largest address

```
cmp EBP, offset queue; offset queue = 0
je ebpbacktolargest:
```

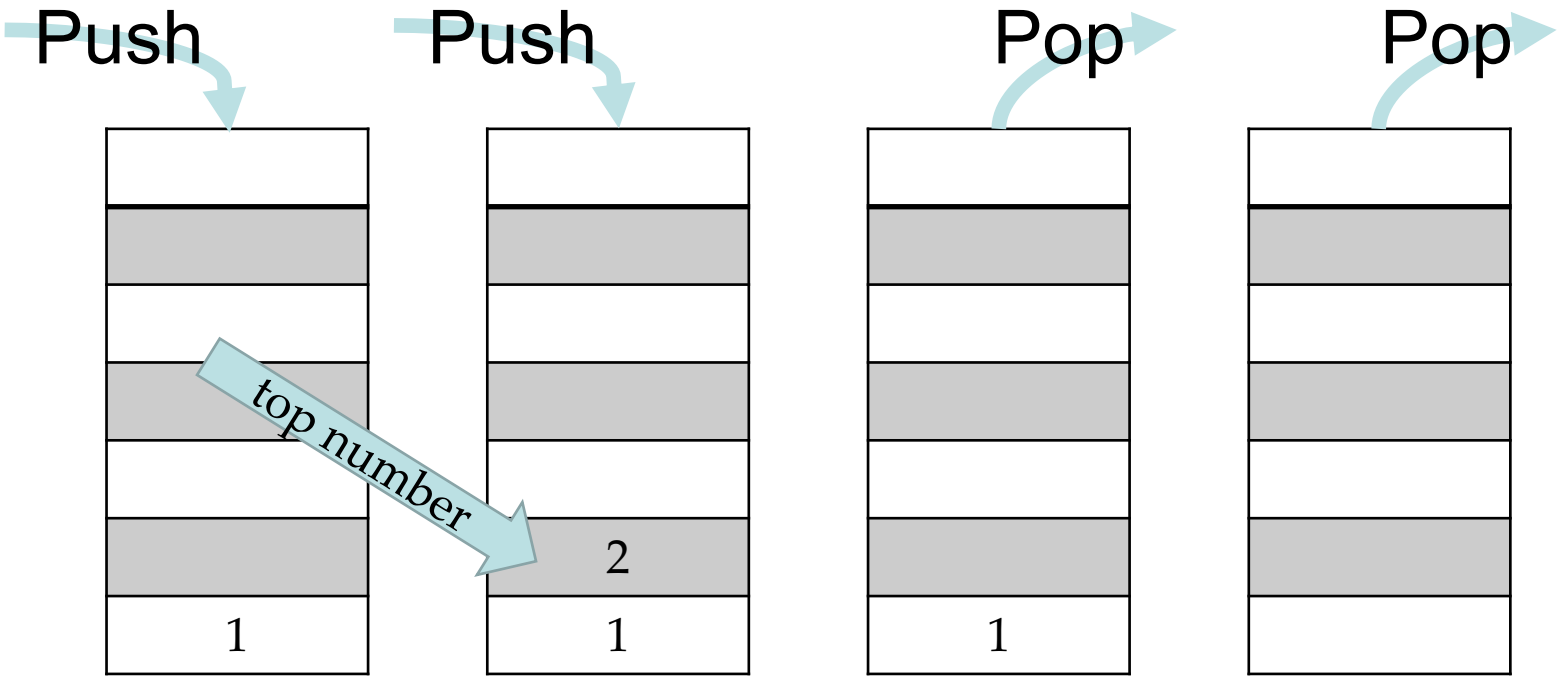
```
ebpbacktolargest:
        mov EBP, 4 ; 4 bytes
        imul EBP, queuelength ; queuelength = 7
        add EBP, offset queue ; EBP = 0 + 4 * 7 = 28
```



Basic knowledge of stack

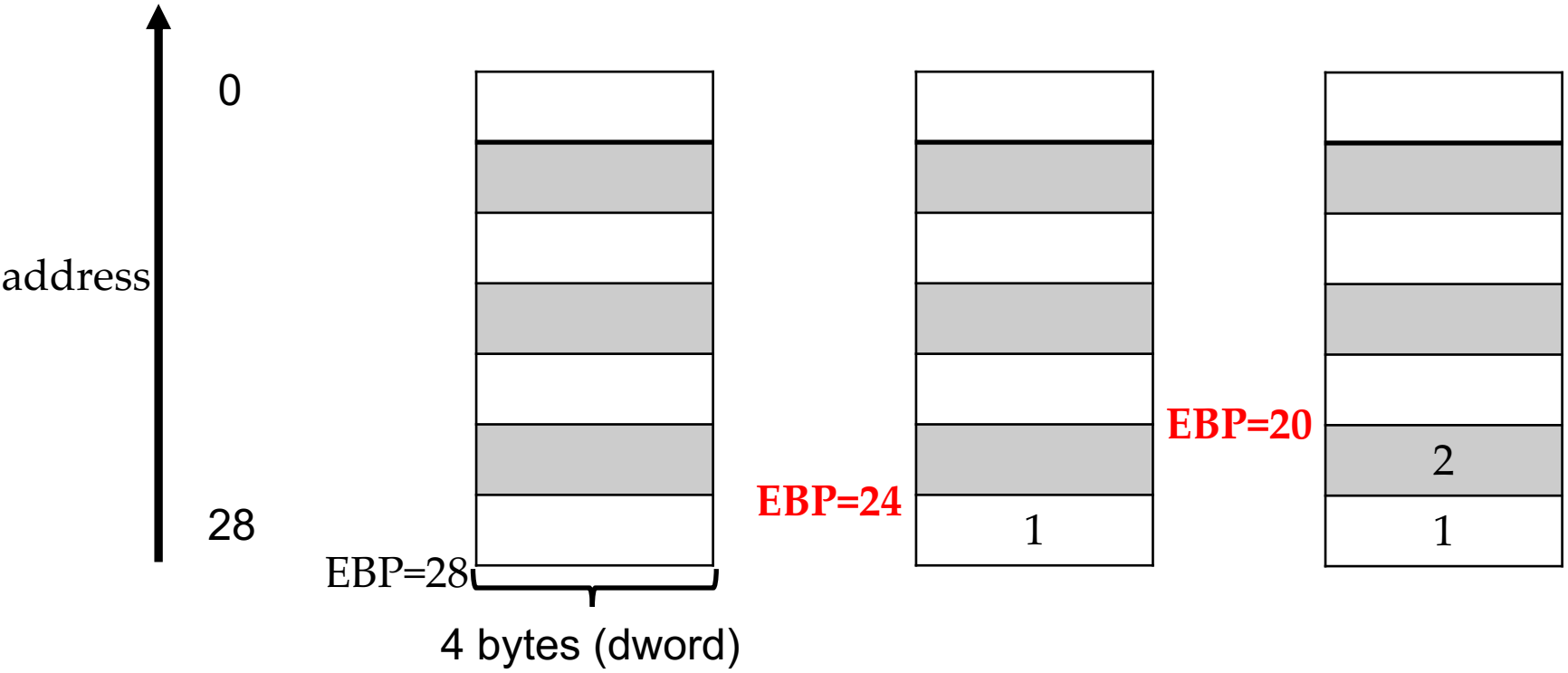
Stack: a list of data elements

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



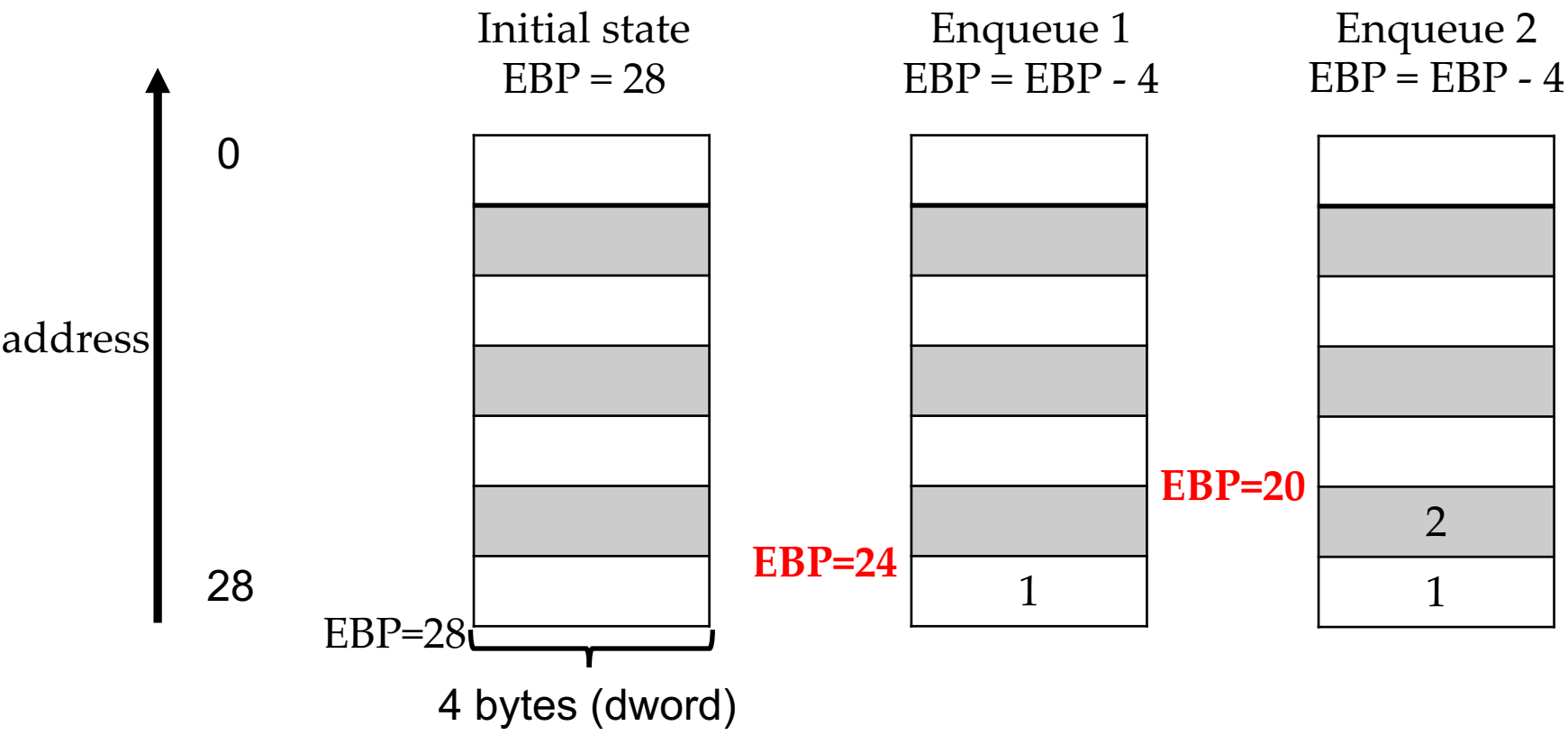
Stack Implementation

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



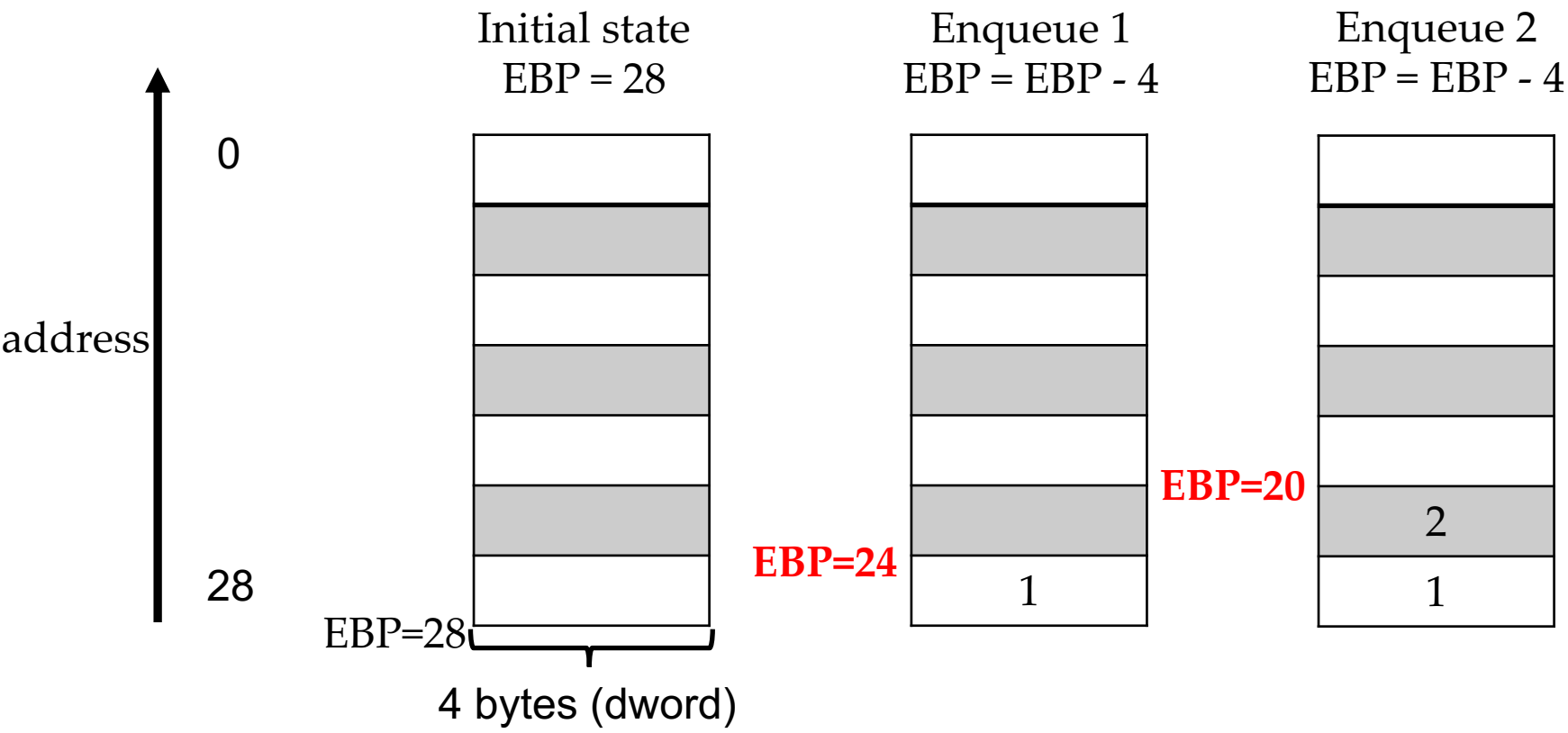
Stack Implementation: Pushnum

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



Stack Implementation: Popnum

- Popnum pseudo code:
  1) remove number from the top end of stack
  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.
- Let's read the stack.asm.

Summary

- More MASM instructions
- Queue Implementation
- Stack implementation
  - Hints for programming exercise in Assignment 2.