Assembly language programming
By xorpd

Basic Assembly

The Stack

Objectives

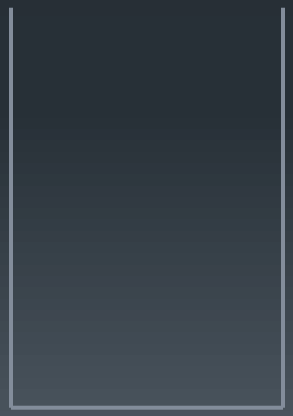
- We learn about the **stack** data structure.
- We study the x86 **stack** implementation and instructions.
- We see simple examples of using the **stack**.

Stack

- Abstract idea for storing data.
- Two operations are allowed: PUSH and POP.
- The last element pushed is the first element to be popped.
  - LIFO - Last In First Out.

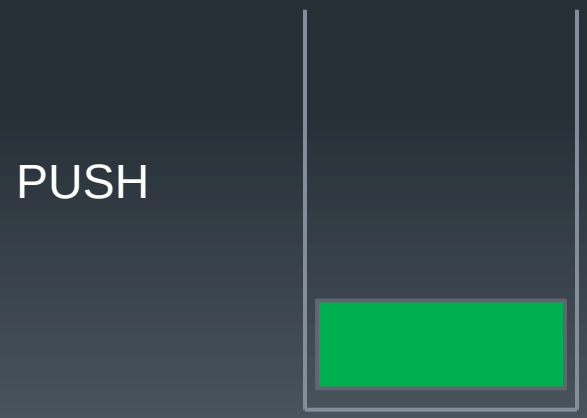
Stack

- Abstract idea for storing data.
- Two operations are allowed: PUSH and POP.
- The last element pushed is the first element to be popped.
  - LIFO - Last In First Out.



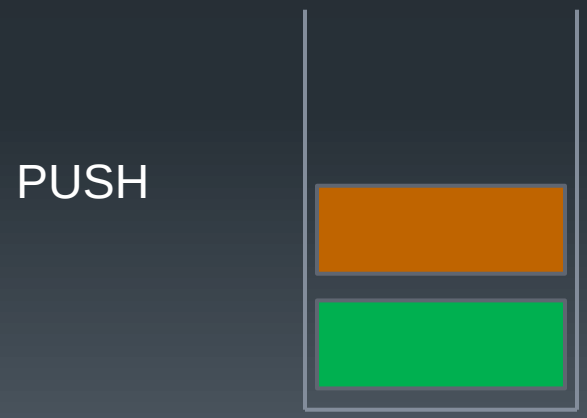
Stack

- Abstract idea for storing data.
- Two operations are allowed: PUSH and POP.
- The last element pushed is the first element to be popped.
  - LIFO - Last In First Out.



Stack

- Abstract idea for storing data.
- Two operations are allowed: PUSH and POP.
- The last element pushed is the first element to be popped.
  - LIFO - Last In First Out.



Stack

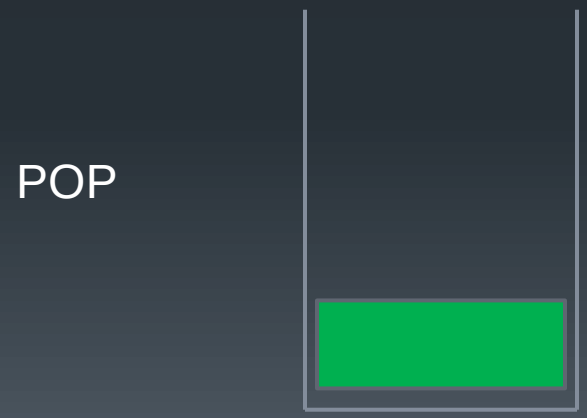
- Abstract idea for storing data.
- Two operations are allowed: PUSH and POP.
- The last element pushed is the first element to be popped.
  - LIFO - Last In First Out.

Stack

- Abstract idea for storing data.
- Two operations are allowed: PUSH and POP.
- The last element pushed is the first element to be popped.
  - LIFO - Last In First Out.

Stack

- Abstract idea for storing data.
- Two operations are allowed: PUSH and POP.
- The last element pushed is the first element to be popped.
  - LIFO - Last In First Out.



Stack

- Abstract idea for storing data.
- Two operations are allowed: PUSH and POP.
- The last element pushed is the first element to be popped.
  - LIFO - Last In First Out.

Stack (Cont.)

- Real Life stacks:

Andy Rennie
https://farm2.staticflickr.com/1169/4608617962_92d50edbfb.jpg

Michael Mandibeg
http://www.flickr.com/photos/theredproject/3293550847/

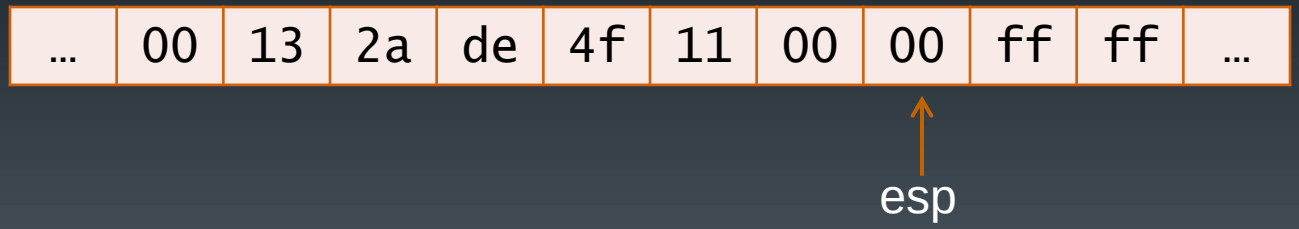
http://www.flickr.com/photos/aloha75/8395557674/

Stew Dean
http://www.flickr.com/photos/stewdean/8699198747/

ESP

- ESP is a 32 bits register. (Extended Stack Pointer)

- At the moment your code begins to run, esp already contains an address of a location in memory called “the stack”.
  - ESP and the stack are set up automatically by the operation system.



- There are some special instructions that deal with ESP and the stack.

PUSH

- PUSH arg
  - Push onto the stack.

- Two forms:
  - arg is of size 16 bit:
    - $esp \leftarrow esp - 2$
    - $word\ [esp] \leftarrow arg$
  - arg is of size 32 bit:
    - $esp \leftarrow esp - 4$
    - $dword\ [esp] \leftarrow arg$

PUSH

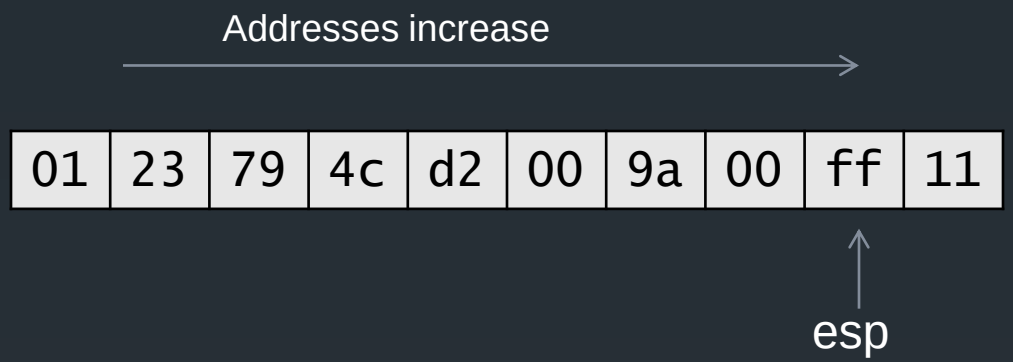
- PUSH arg
  - Push onto the stack.

- Two forms:
  - arg is of size 16 bit:
    - $esp \leftarrow esp - 2$
    - $word\ [esp] \leftarrow arg$
  - arg is of size **32** bit:
    - $esp \leftarrow esp - 4$
    - $dword\ [esp] \leftarrow arg$

PUSH

- PUSH arg
  - Push onto the stack.

- Two forms:
  - arg is of size 16 bit:
    - $esp \leftarrow esp - 2$
    - $word\ [esp] \leftarrow arg$
  - arg is of size **32** bit:
    - $esp \leftarrow esp - 4$
    - $dword\ [esp] \leftarrow arg$

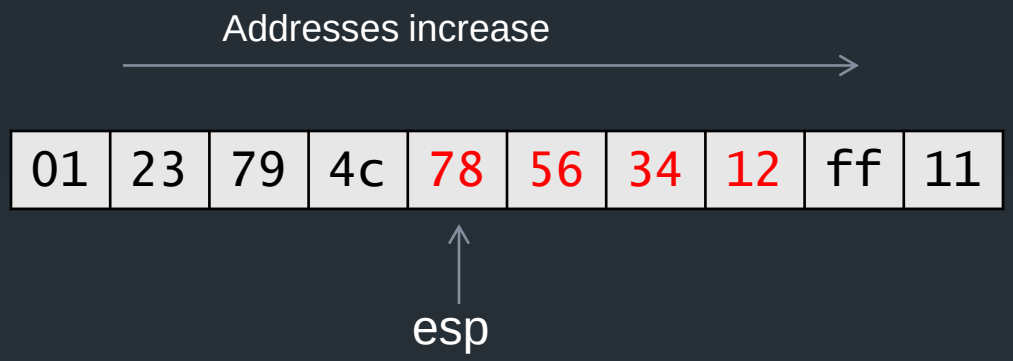



```
push    12345678h
mov     eax,3h
push    eax
```

PUSH

- PUSH arg
  - Push onto the stack.

- Two forms:
  - arg is of size 16 bit:
    - $esp \leftarrow esp - 2$
    - $word\ [esp] \leftarrow arg$
  - arg is of size **32** bit:
    - $esp \leftarrow esp - 4$
    - $dword\ [esp] \leftarrow arg$

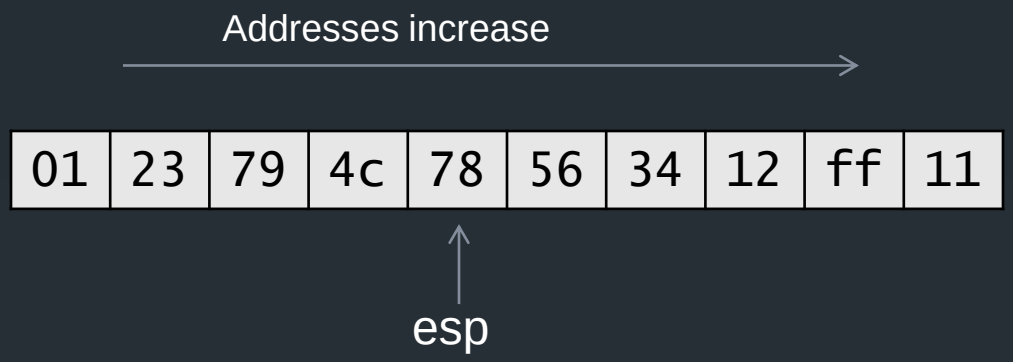



```
push    12345678h
mov     eax,3h
push    eax
```

PUSH

- PUSH arg
  - Push onto the stack.

- Two forms:
  - arg is of size 16 bit:
    - $esp \leftarrow esp - 2$
    - $word\ [esp] \leftarrow arg$
  - arg is of size **32** bit:
    - $esp \leftarrow esp - 4$
    - $dword\ [esp] \leftarrow arg$

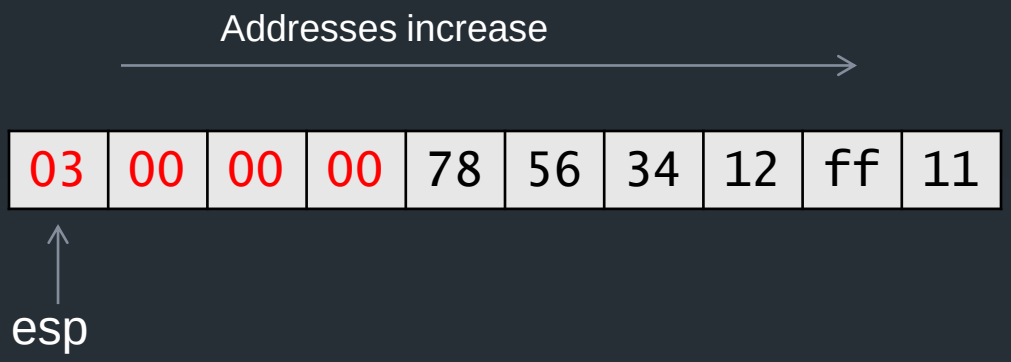



```
push    12345678h
mov     eax,3h
push    eax
```

PUSH

- PUSH arg
  - Push onto the stack.

- Two forms:
  - arg is of size 16 bit:
    - $esp \leftarrow esp - 2$
    - $word\ [esp] \leftarrow arg$
  - arg is of size **32** bit:
    - $esp \leftarrow esp - 4$
    - $dword\ [esp] \leftarrow arg$




```
push    12345678h
mov     eax,3h
push    eax
```

POP

- POP arg
  - Pop a value from the stack.

- Two forms:
  - arg is of size 16 bit:
    - $arg \leftarrow word\ [esp]$
    - $esp \leftarrow esp + 2$
  - arg is of size 32 bit:
    - $arg \leftarrow dword\ [esp]$
    - $esp \leftarrow esp + 4$

POP

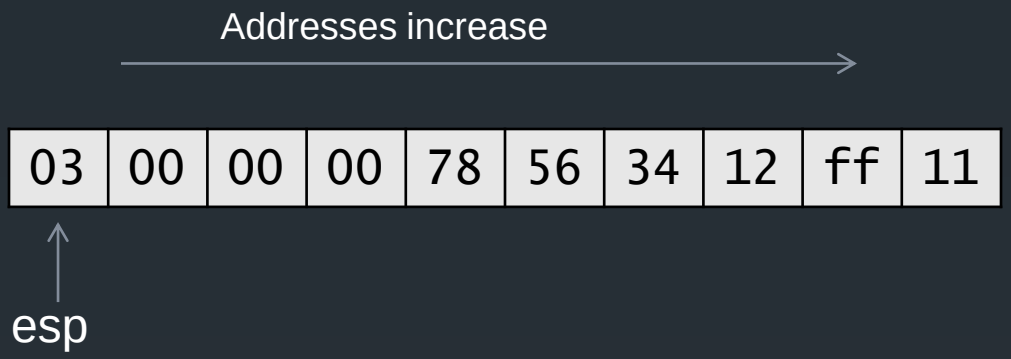
- POP arg
  - Pop a value from the stack.

- Two forms:
  - arg is of size 16 bit:
    - $arg \leftarrow word\ [esp]$
    - $esp \leftarrow esp + 2$
  - arg is of size **32** bit:
    - $arg \leftarrow dword\ [esp]$
    - $esp \leftarrow esp + 4$

POP

- POP arg
  - Pop a value from the stack.

- Two forms:
  - arg is of size 16 bit:
    - $arg \leftarrow word\ [esp]$
    - $esp \leftarrow esp + 2$



  - arg is of size **32** bit:
    - $arg \leftarrow dword\ [esp]$
    - $esp \leftarrow esp + 4$

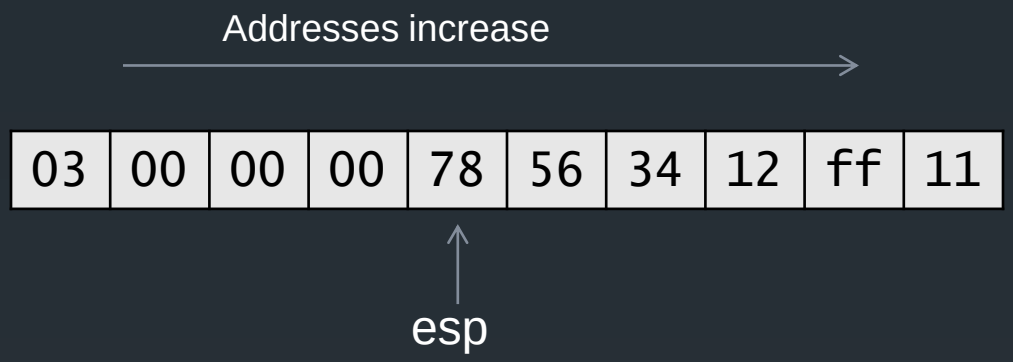
```
pop     eax
pop     ecx
```

eax	ecx
????????	????????

POP

- POP arg
  - Pop a value from the stack.

- Two forms:
  - arg is of size 16 bit:
    - $arg \leftarrow word\ [esp]$
    - $esp \leftarrow esp + 2$
  - arg is of size **32** bit:
    - $arg \leftarrow dword\ [esp]$
    - $esp \leftarrow esp + 4$



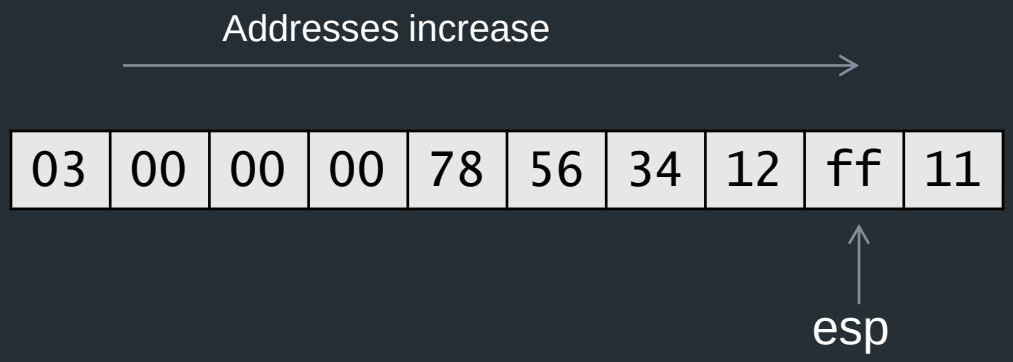

```
pop     eax
pop     ecx
```

eax	ecx
00000003	????????

POP

- POP arg
  - Pop a value from the stack.

- Two forms:
  - arg is of size 16 bit:
    - $arg \leftarrow word\ [esp]$
    - $esp \leftarrow esp + 2$
  - arg is of size **32** bit:
    - $arg \leftarrow dword\ [esp]$
    - $esp \leftarrow esp + 4$




```
pop     eax
pop     ecx
```

eax	ecx
00000003	12345678

Example – Exchanging values

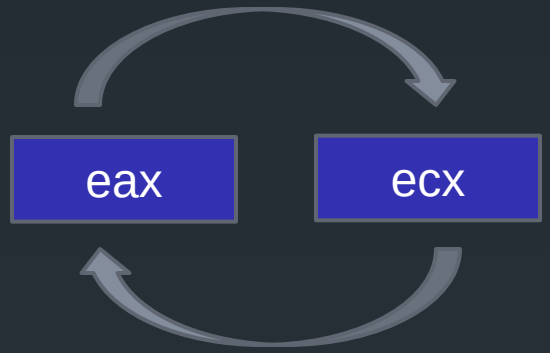
- Exchanging two values:

```
push    eax
push    ecx
pop     eax
pop     ecx
```

Example – Exchanging values

- Exchanging two values:

```
push    eax
push    ecx
pop     eax
pop     ecx
```

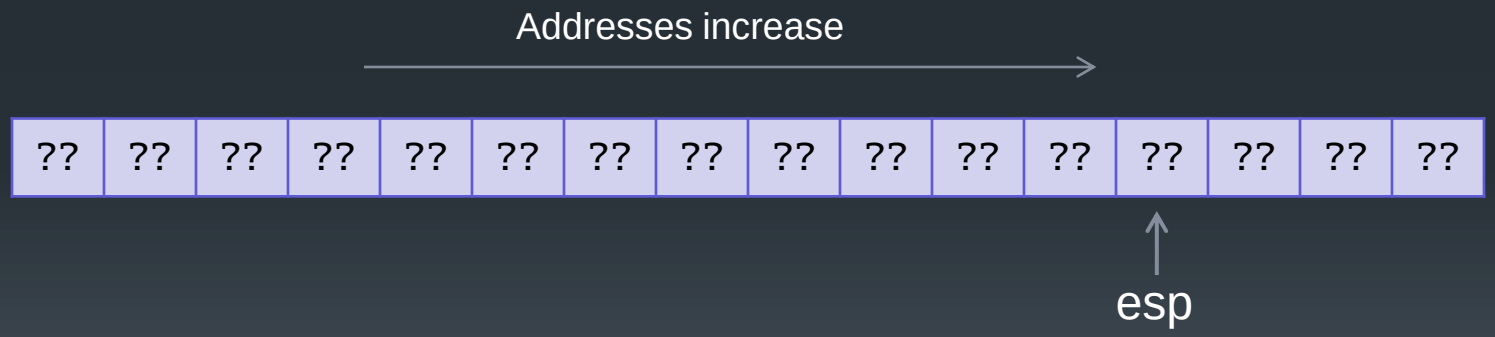


Example – Exchanging values

- Exchanging two values:

```
push    eax
push    ecx
pop     eax
pop     ecx
```

eax	ecx	esp
12345678	aabbccdd	001ff704



Example – Exchanging values

- Exchanging two values:

```
push    eax
push    ecx
pop     eax
pop     ecx
```

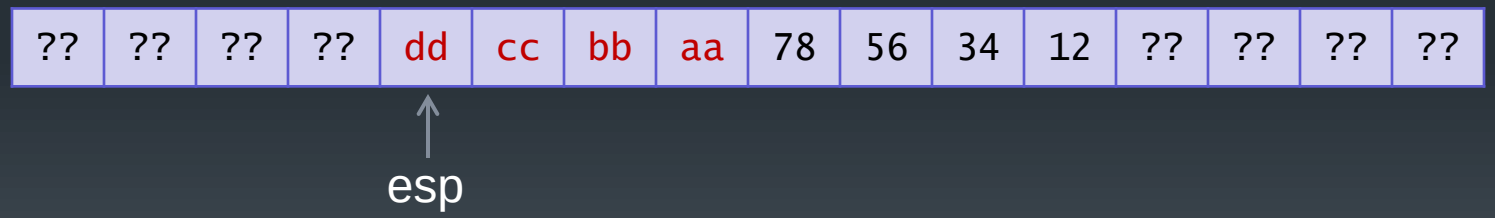
eax	ecx	esp
12345678	aabbccdd	001ff700

Example – Exchanging values

- Exchanging two values:

```
push    eax
push    ecx
pop     eax
pop     ecx
```

eax	ecx	esp
12345678	aabbccdd	001ff6fc

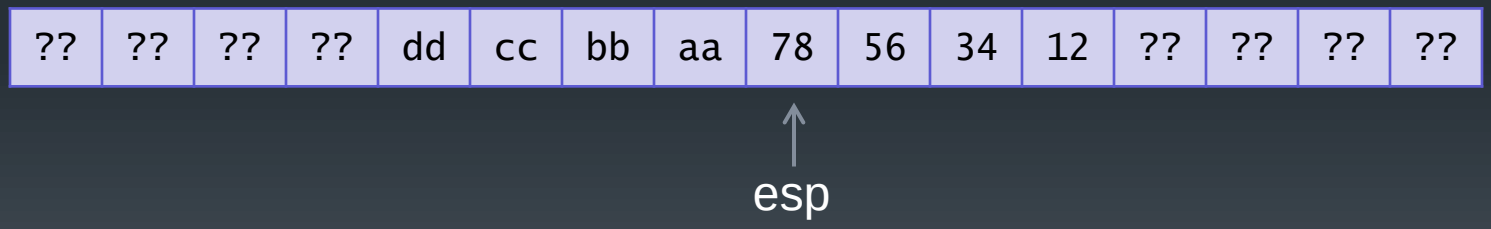


Example – Exchanging values

- Exchanging two values:

```
push    eax
push    ecx
pop     eax
pop     ecx
```

eax	ecx	esp
aabbccdd	aabbccdd	001ff700

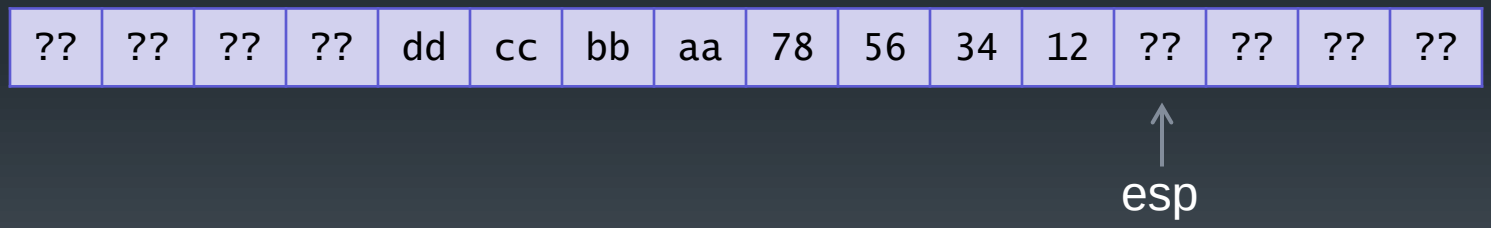


Example – Exchanging values

- Exchanging two values:

```
push    eax
push    ecx
pop     eax
pop     ecx
```

eax	ecx	esp
aabbccdd	12345678	001ff704



Example – Saving and restoring

- Example: We want to keep the ecx register unchanged.

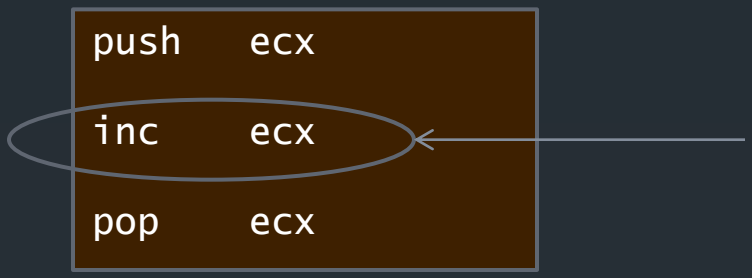
```
push    ecx

inc     ecx

pop     ecx
```

Example – Saving and restoring

- Example: We want to keep the ecx register unchanged.

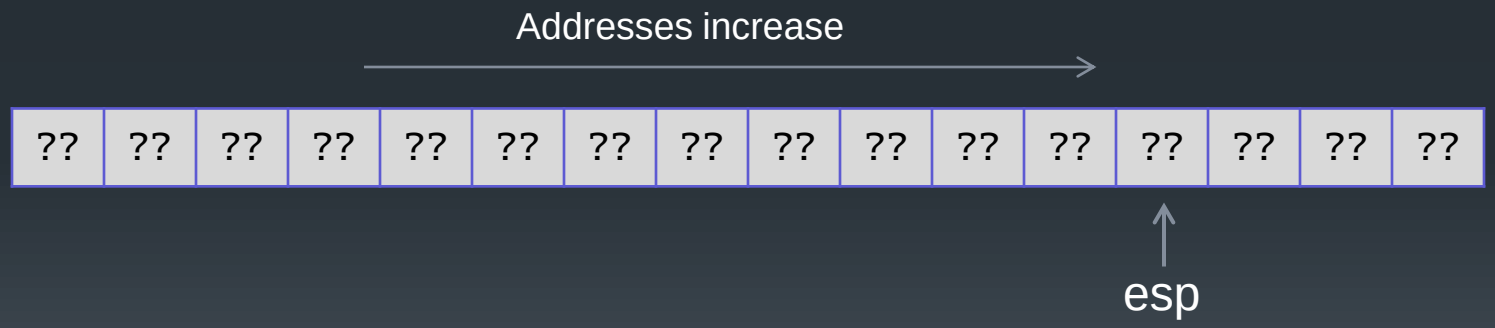


Example – Saving and restoring

- Example: We want to keep the ecx register unchanged.

```
push    ecx
inc     ecx
pop     ecx
```

ecx	esp
00030200	001ff704



Example – Saving and restoring

- Example: We want to keep the ecx register unchanged.

```
push    ecx
inc     ecx
pop     ecx
```

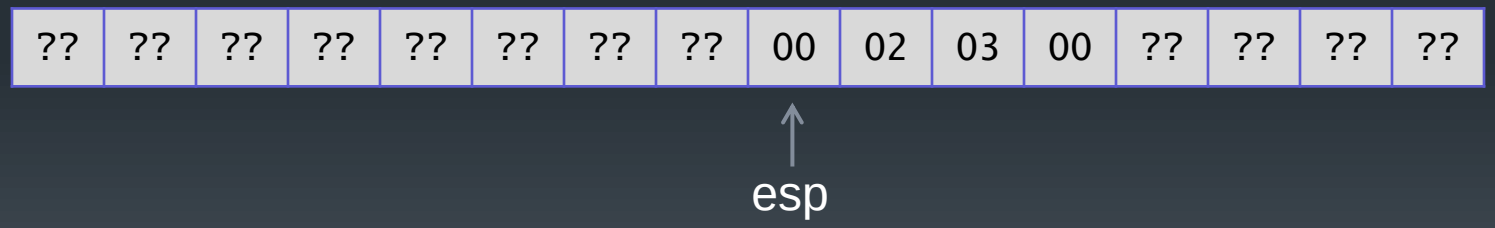
ecx	esp
00030200	001ff700

Example – Saving and restoring

- Example: We want to keep the ecx register unchanged.

```
push    ecx
inc     ecx
pop     ecx
```

ecx	esp
00030201	001ff700

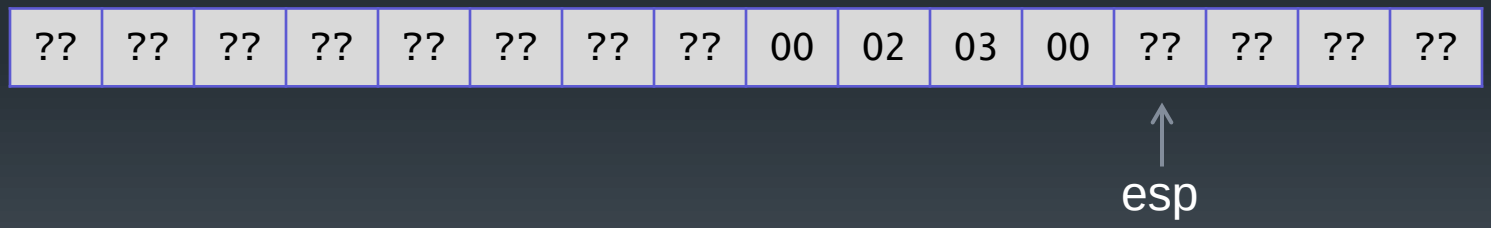


Example – Saving and restoring

- Example: We want to keep the ecx register unchanged.

```
push    ecx
inc     ecx
pop     ecx
```

ecx	esp
00030200	001ff704



Example – Saving and restoring (cont.)

- Keeping a few registers and then restoring them:

```
push    ecx
push    eax
push    ebx

... ; some code

pop     ebx
pop     eax
pop     ecx
```

- Note the push and pop order.

Summary

- A stack is an abstract idea for storing data.
  - Only PUSH and POP.
  - Last In First Out.
- x86 stack:
  - ESP points to the “top” of the stack.
  - PUSH decreases esp and writes to the stack.
  - POP reads from the stack and increases esp.

- Examples:
  - Exchanging values.
  - Saving and restoring.