Sécurité des logiciels

Assembly language, part 3

Samuel Thibault <samuel.thibault@u-bordeaux.fr>
Pieces from Emmanuel Fleury <emmanuel.fleury@u-bordeaux.fr>

Indirect memory accesses

Memory accesses

```
int i = 42;

int j = 43;


main() {
  i = 1;

  j = i;


  j++;

}
```


```
i:
   .long 42
j:
   .long 43

main:
   movl $1, i

   movl i, j
   movl i, %eax
   movl %eax, j
   incl j

   ret
```

Indirect memory accesses

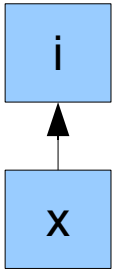
```
int i = 42;
```


```
i:
   .long 42


main() {
   i = 1;

   int *x = &i;

   *x = 0;

   (*x)++;

}
```

```
main:
   movl $1, i

   movl $i, %eax

   movl $0, (%eax)

   incl (%eax)

   ret
```

Indirect memory accesses

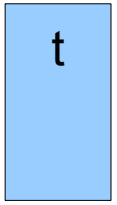
```
int i = 42;
int *x;


main() {
   i = 1;

   x = &i;

   *x = 0;



   (*x)++;
}
```




```
i:
   .long 42
x:
   .long 0
main:
   movl $1, i

   movl $i, x

   movl $0, (x)
   movl x, %eax
   movl $0, (%eax)

   incl (%eax)
   ret
```

Indirect memory accesses

```
int t[2] = {1,2};
```



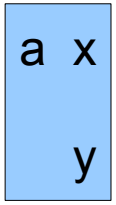

```
t:
   .long 1
   .long 2
```

```
main() {
   t[0] = 3;
   t[1] = 4;

   int *x = &t[1];

   *x = 0;

   (*x)++;
}
```

```
main:
   movl $3, t
   movl $4, t+4

   movl $t+4, %eax

   movl $0, (%eax)

   incl (%eax)
   ret
```

Indirect memory accesses

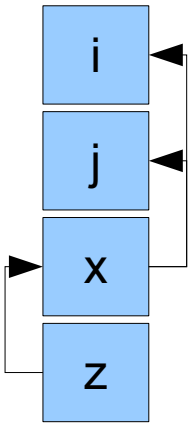
```
struct {
  int x;
  int y;
} a = {.x = 1, .y = 2};

main() {
  a.x = 3;
  a.y = 4;

  int *x = &a.y;

  *x = 0;

  (*x)++;
}
```




```
a:
  .long 1
  .long 2

main:
  movl $3, a
  movl $4, a+4

  movl $a+4, %eax

  movl $0, (%eax)

  incl (%eax)
  ret
```

Indirect memory accesses

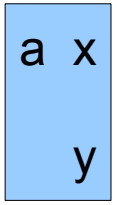
```
int i = 42, j = 43;
int *x;
int **z;


main() {
   x = &i;

   z = &x;
   [...]

   *z = &j;
}
```




```
i:.long 42
j:.long 43
x:.long 0
z:.long 0


main:
   movl $i, x

   movl $x, z
   [...]

   movl z, %eax
   movl $j, (%eax)
   ret
```

Indirect memory accesses

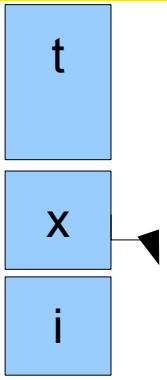
```
struct foo{
   int x;
   int y;
} a = {.x = 1, .y = 2};
struct foo *b = ...;

main() {


   b->x = 41;
   b->y = 42;

   b->y++;
}
```




```
a: .long 1
   .long 2


b: .long ...

main:
   movl b, %eax

   movl $41, 0(%eax)
   movl $42, 4(%eax)

   incl 4(%eax)
   ret
```

Indirect memory accesses

```
int t[10] = {0};

int *x = ...;
int i = ...;

main() {
  int *p;

  t[i] = 41;
  x[2] = 42;
  x[i] = 43;
  x[1]++;
  p = &t[i];
  p = &x[i];
}
```




```
t: .long 0
   ...
x: .long ...
i: .long ...

main:
   movl x, %eax
   movl i, %ebx
   movl $41, t(,%ebx,4)
   movl $42, 8(%eax)
   movl $42, (%eax,%ebx,4)
   incl 4(%eax)
   leal t(,%ebx,4), %ecx
   leal (%eax,%ebx,4), %ecx
   ret
```

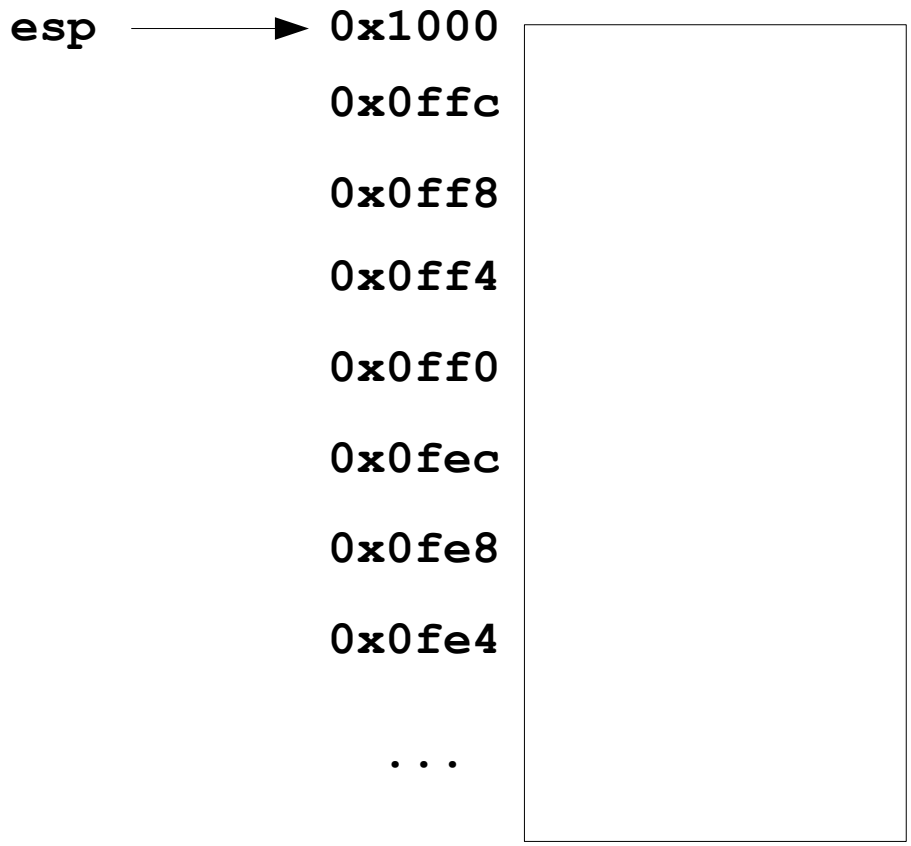
Stack

Stack

LIFO (Last In First Out)

- push
- pop

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```

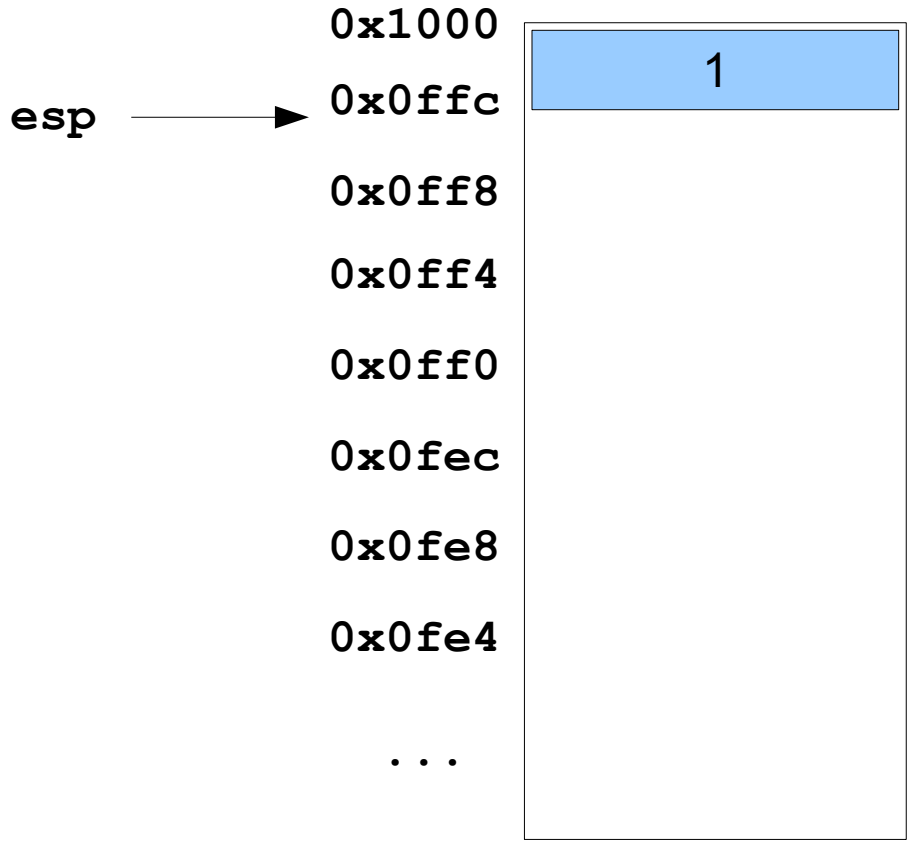


Stack

LIFO (Last In First Out)

- push
- pop

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```

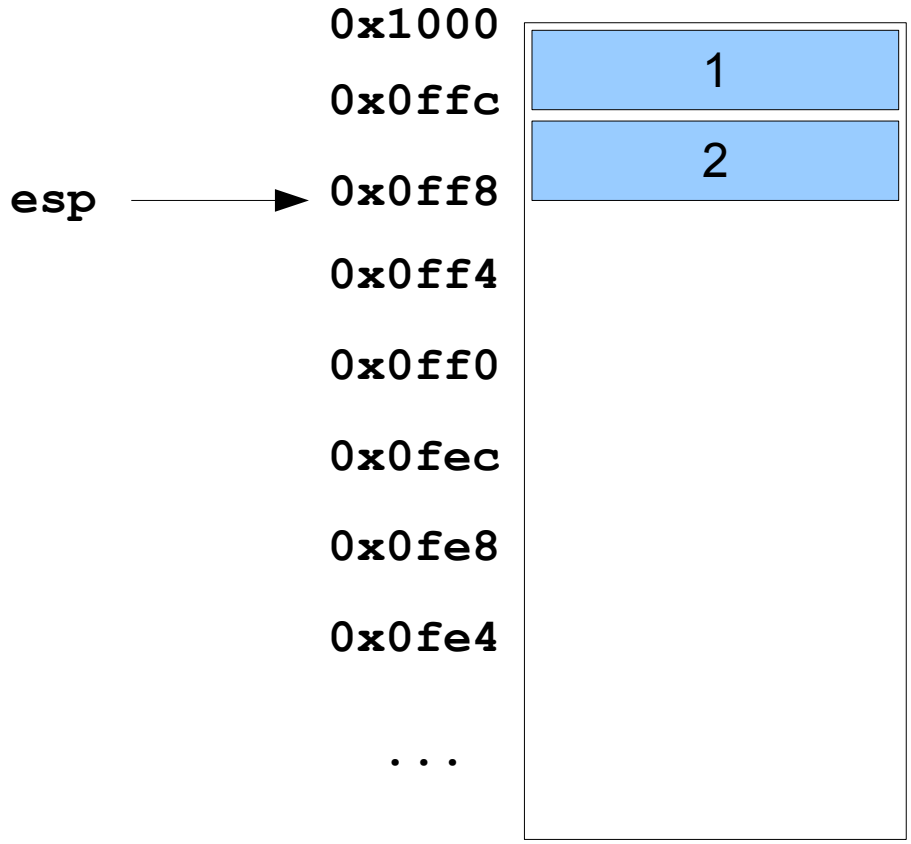


Stack

LIFO (Last In First Out)

- push
- pop

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```

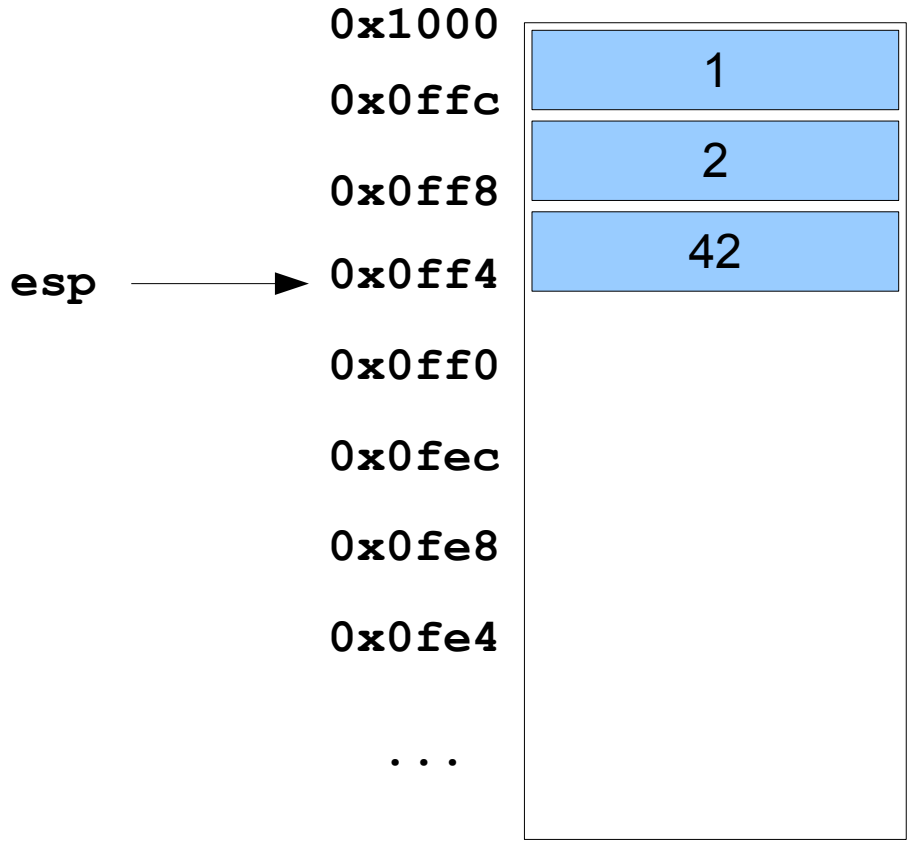


Stack

LIFO (Last In First Out)

- push
- pop

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```

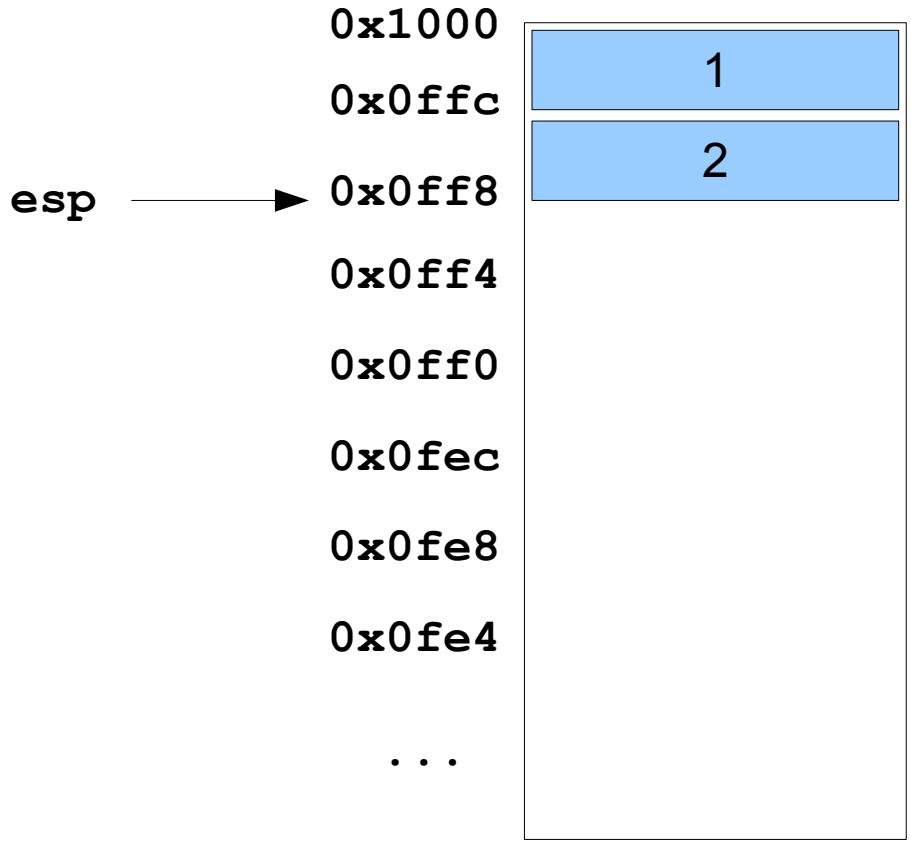


Stack

LIFO (Last In First Out)

- push
- pop

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```



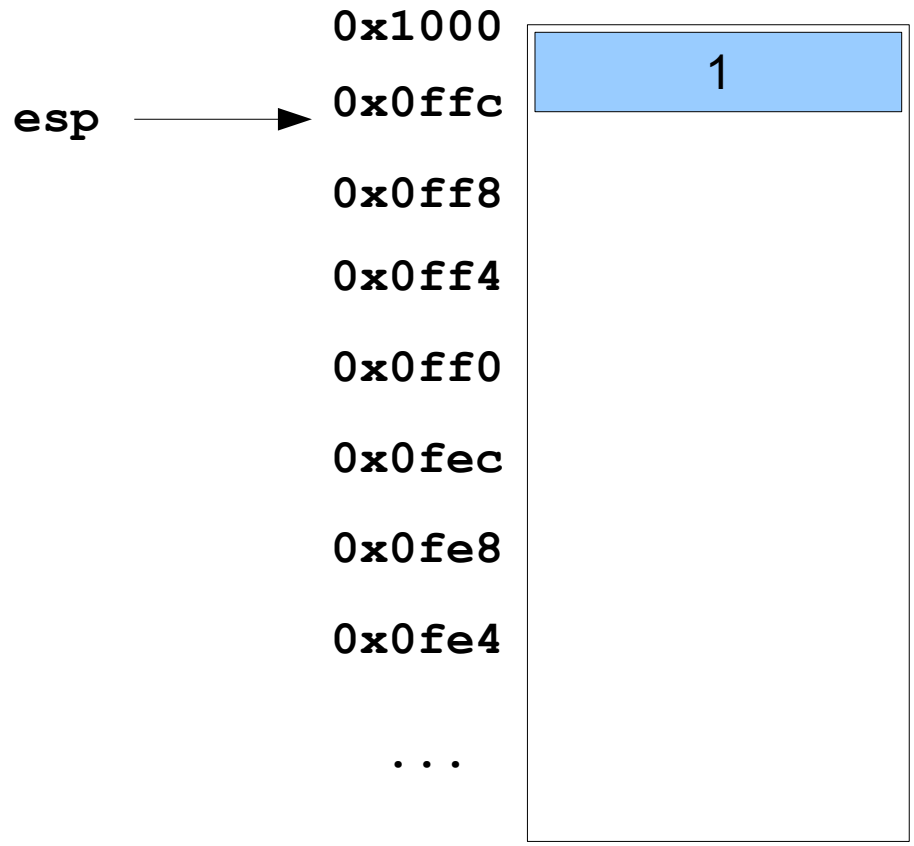

```
eax: 42
```

Stack

LIFO (Last In First Out)

- push
- pop

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```



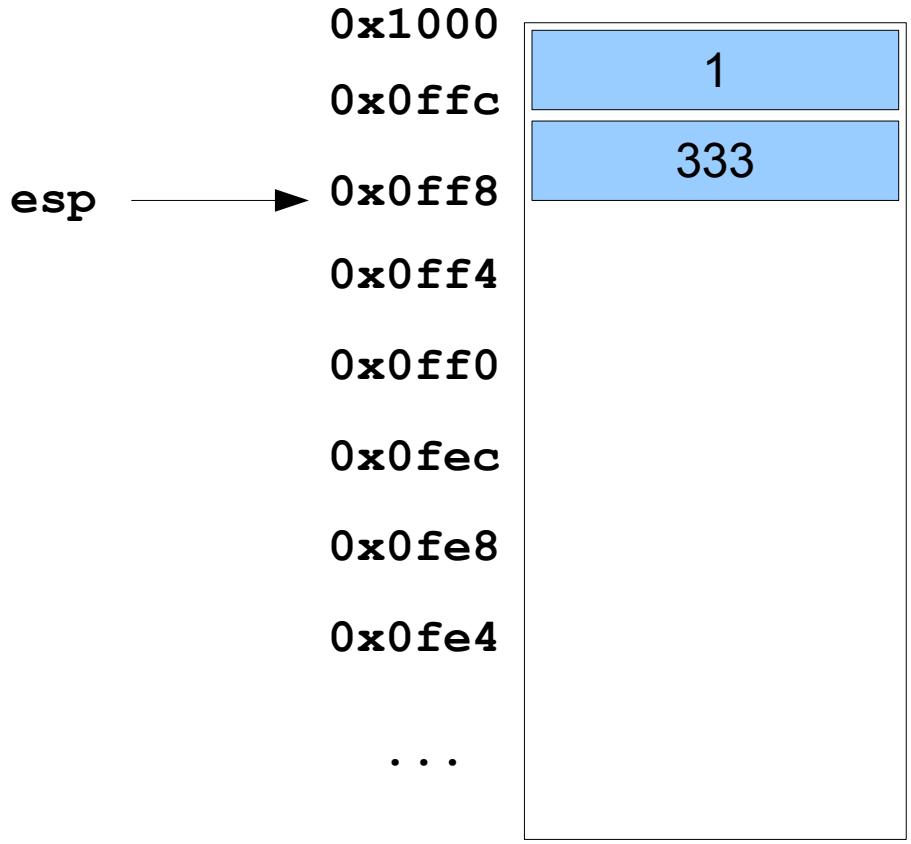
eax: 2

Stack

LIFO (Last In First Out)

- push
- pop

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```



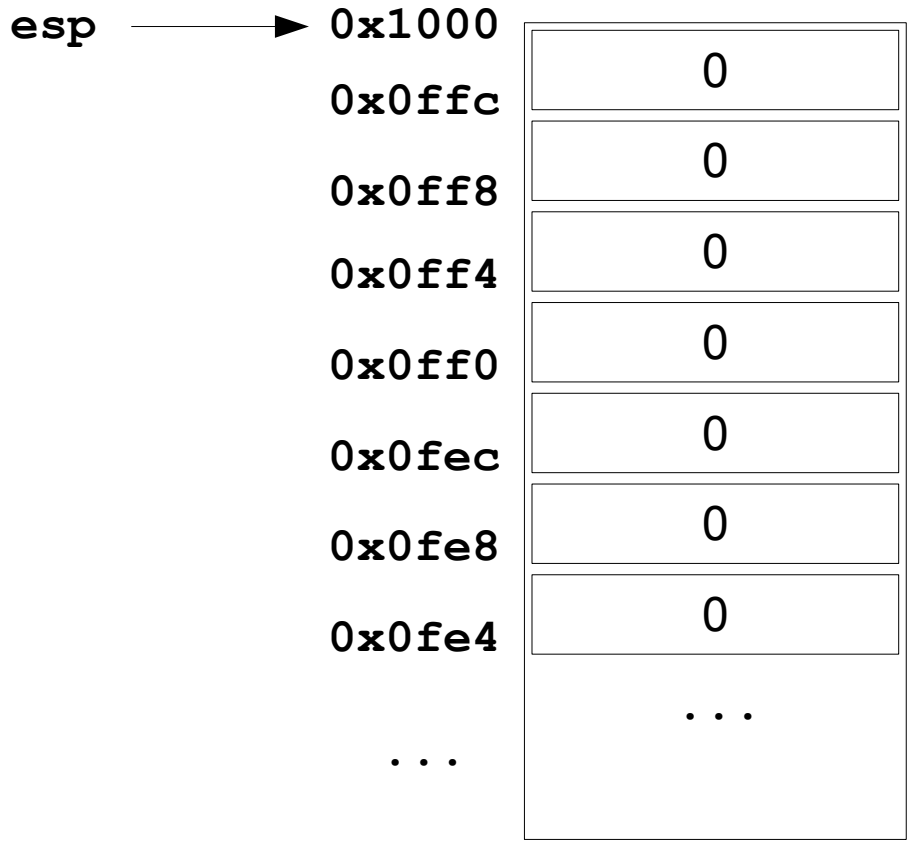

```
eax: 2
```

Stack

For real

- initially 0 memory
- no cleanup

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```

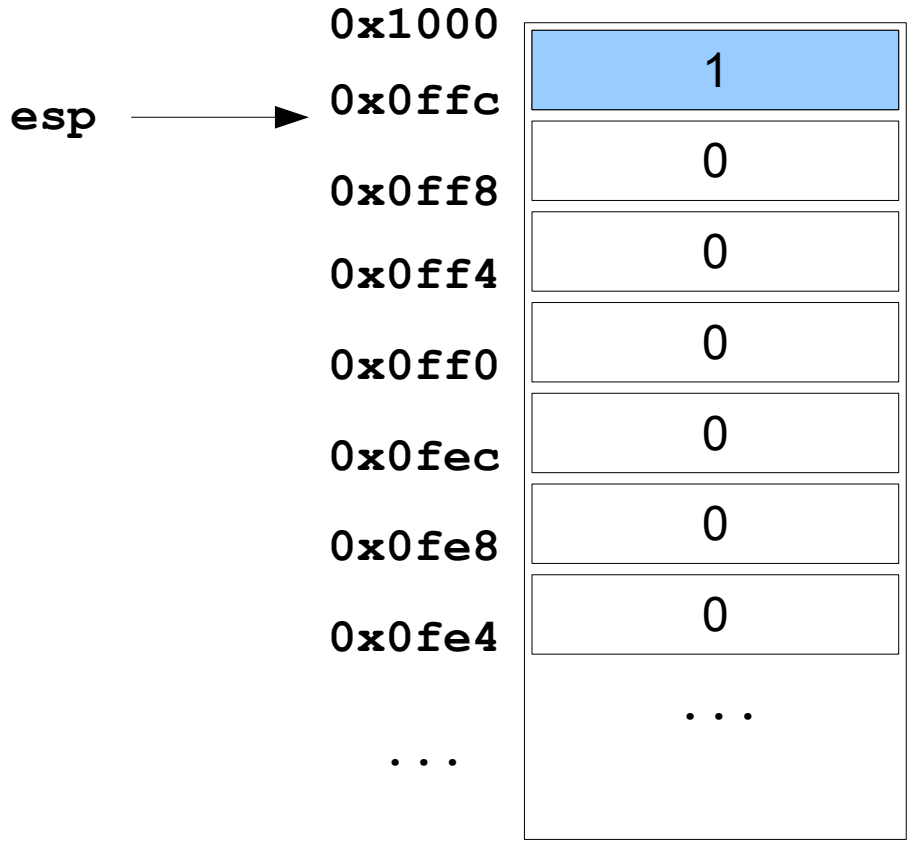


Stack

For real

- initially 0 memory
- no cleanup

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```

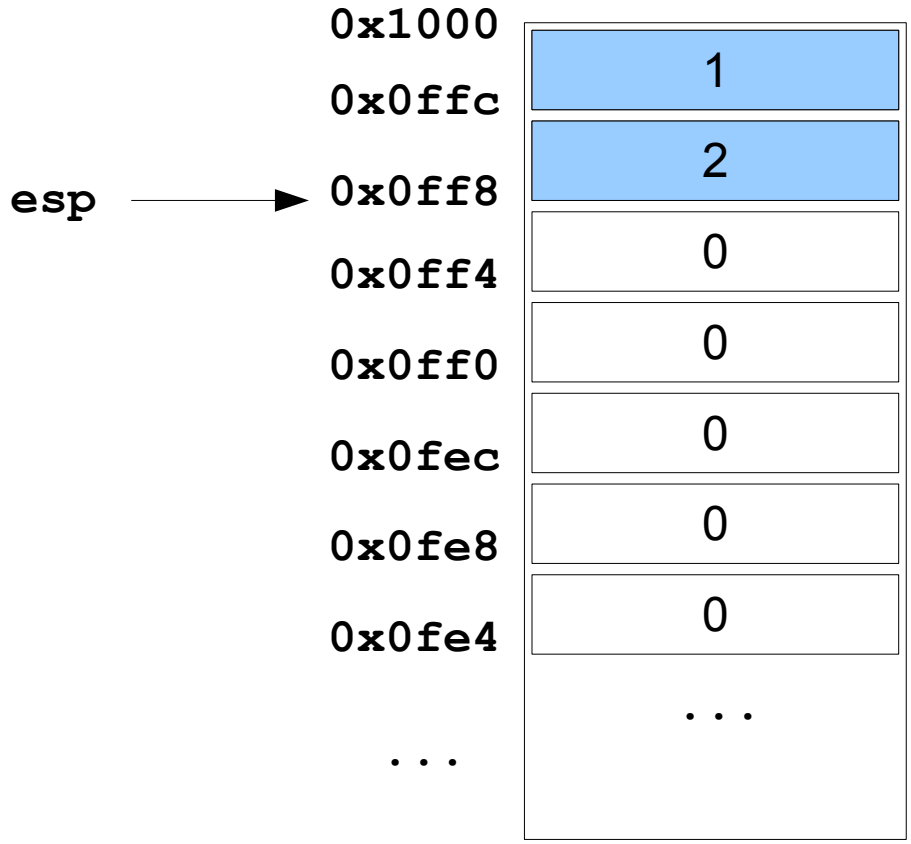


Stack

For real

- initially 0 memory
- no cleanup

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```

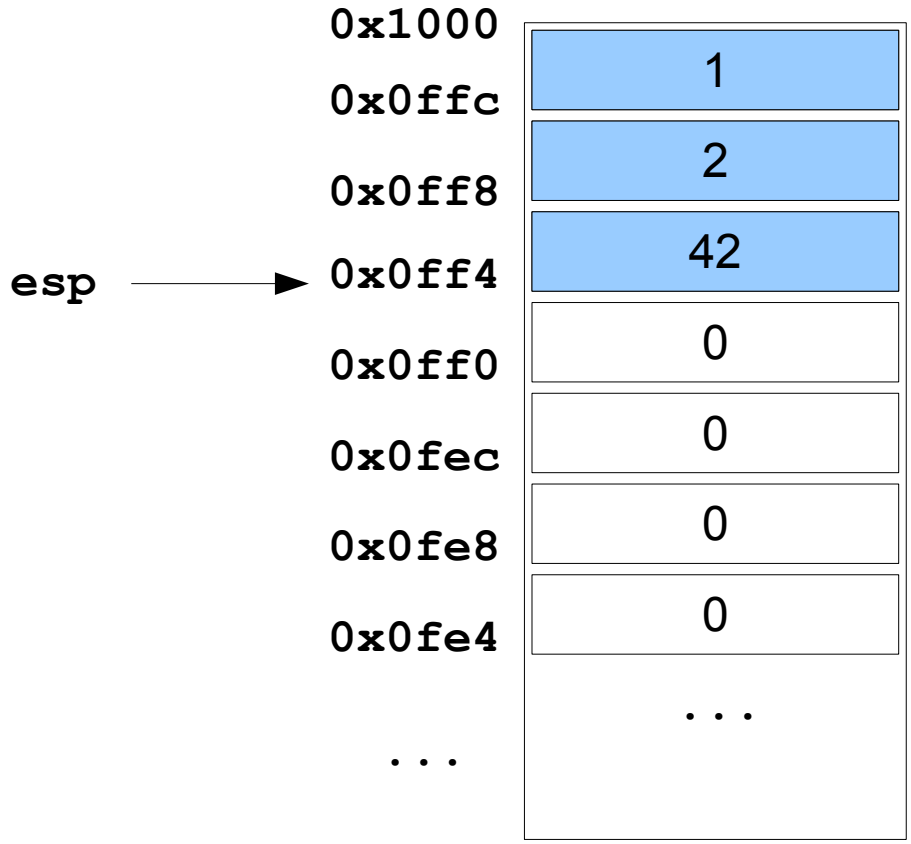


Stack

For real

- initially 0 memory
- no cleanup

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```

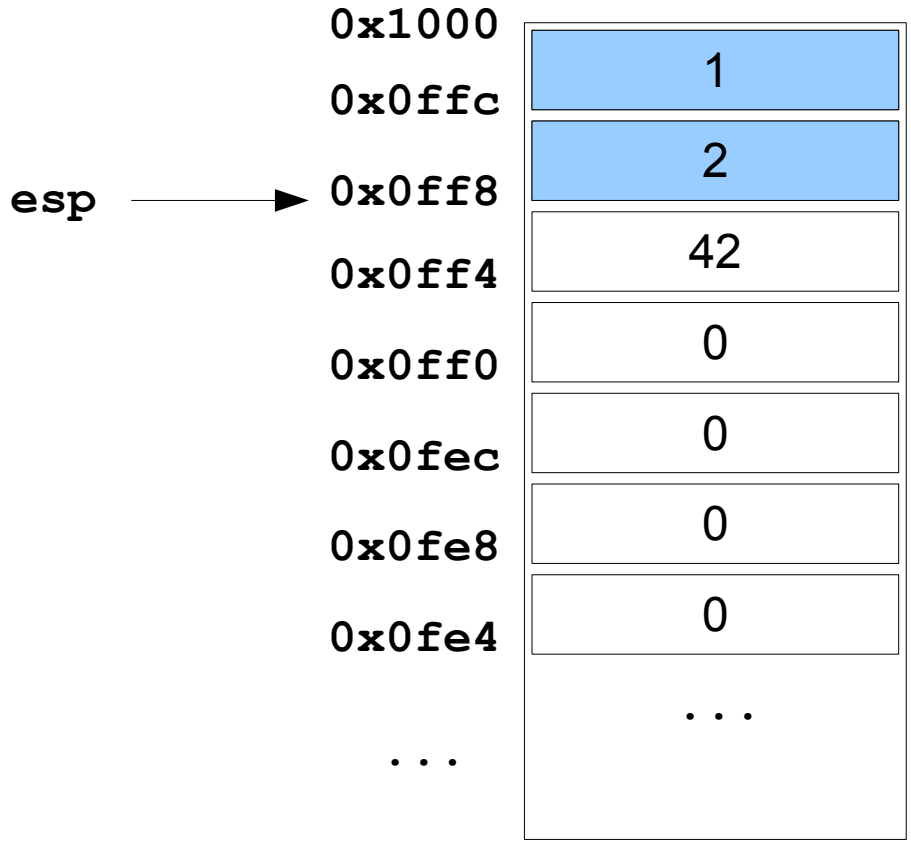


Stack

For real

- initially 0 memory
- no cleanup

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```



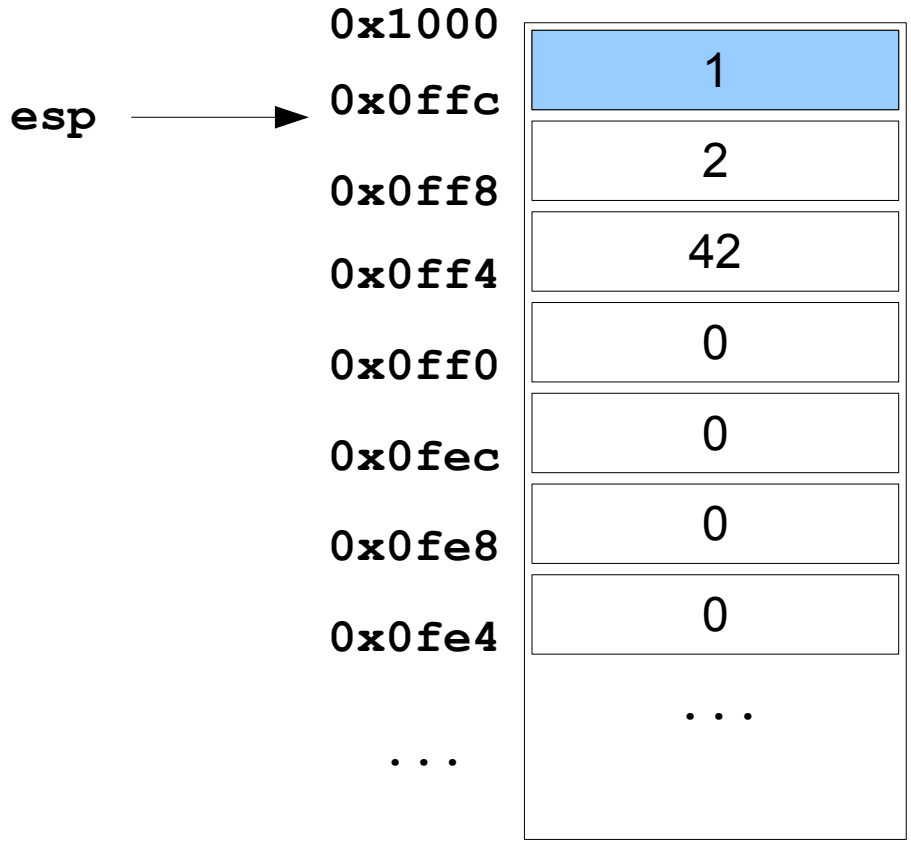

```
eax: 42
```

Stack

For real

- initially 0 memory
- no cleanup

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```



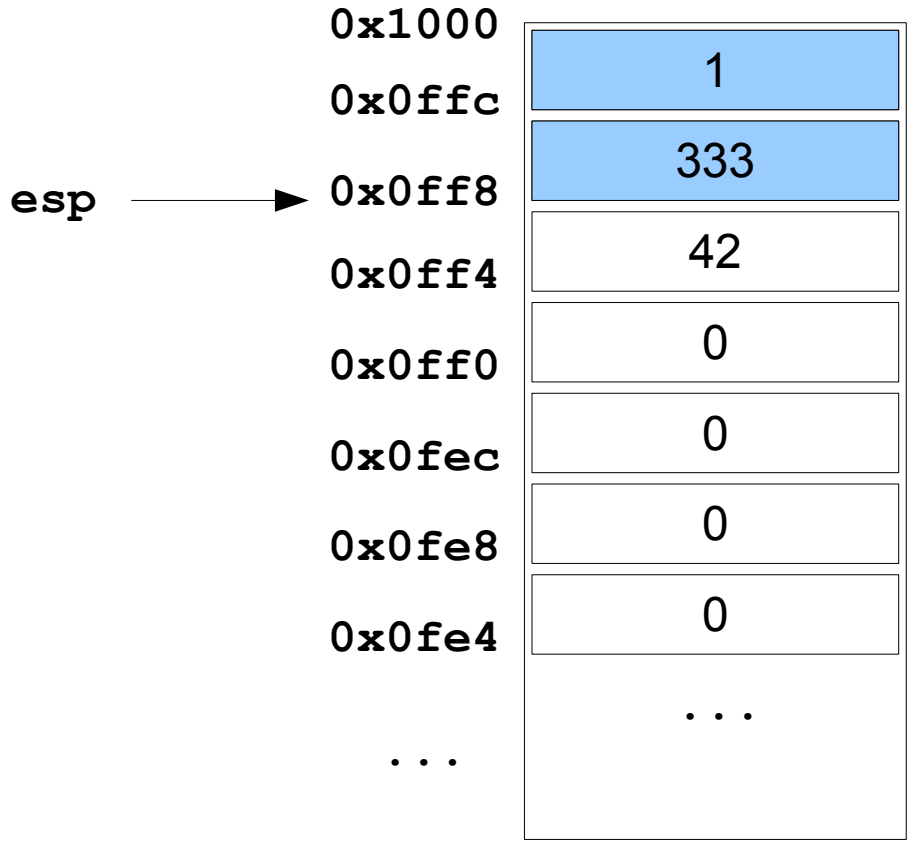

```
eax: 2
```

Stack

For real

- initially 0 memory
- no cleanup

```
pushl $1
pushl $2
pushl $42
popl %eax
popl %eax
pushl $333
```

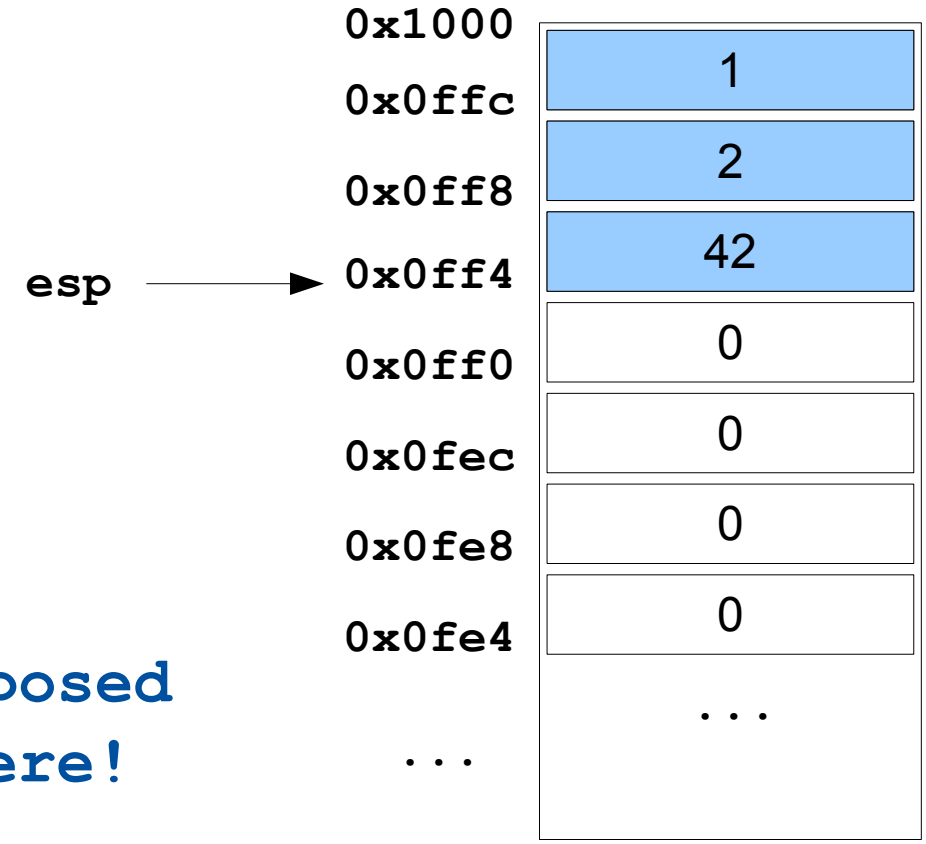



```
eax: 2
```

Indirect indexing

From there

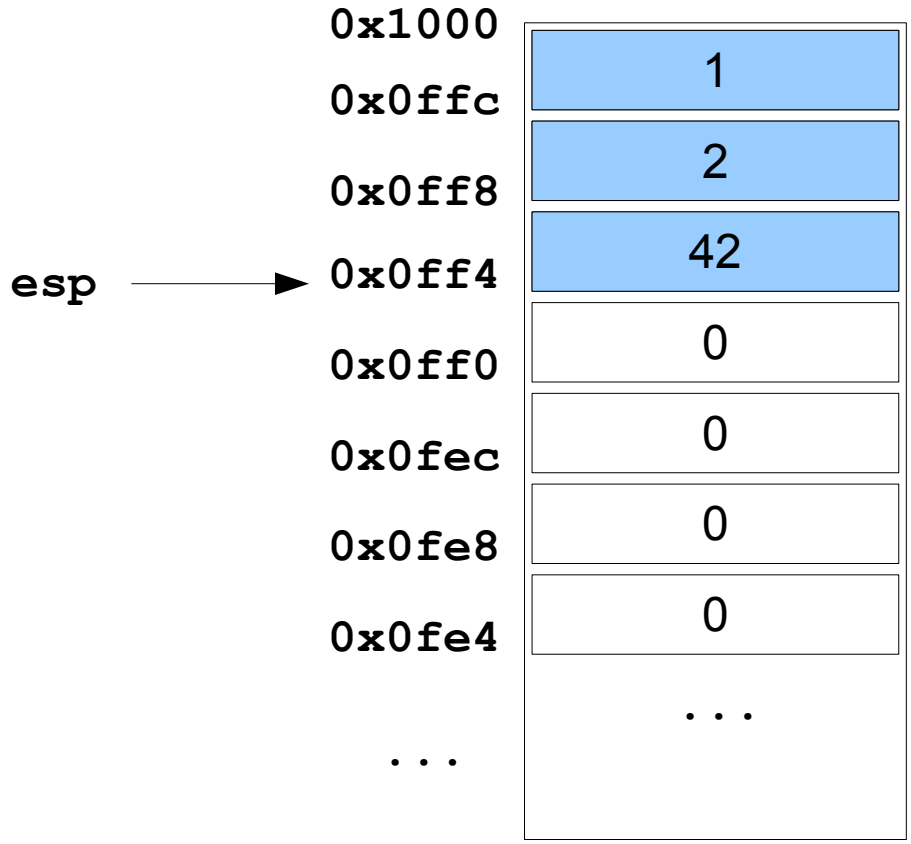
```
movl %esp, %eax
-> 0xff4
movl (%esp),%eax
-> 42
movl 4(%esp),%eax
-> 2
movl -4(%esp),%eax
-> 0          # but not supposed
              # to access here!
```



Indirect indexing

I.e.

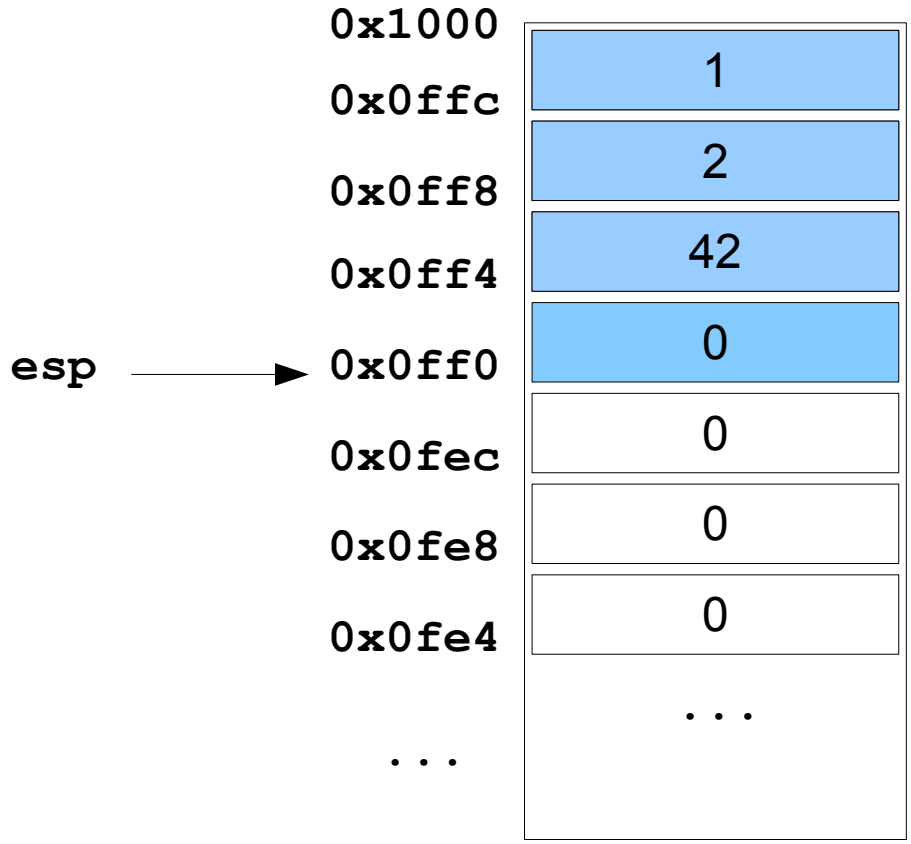
```
pushl $333

<=>

subl $4,%esp
movl $333,(%esp)
```



Indirect indexing

I.e.

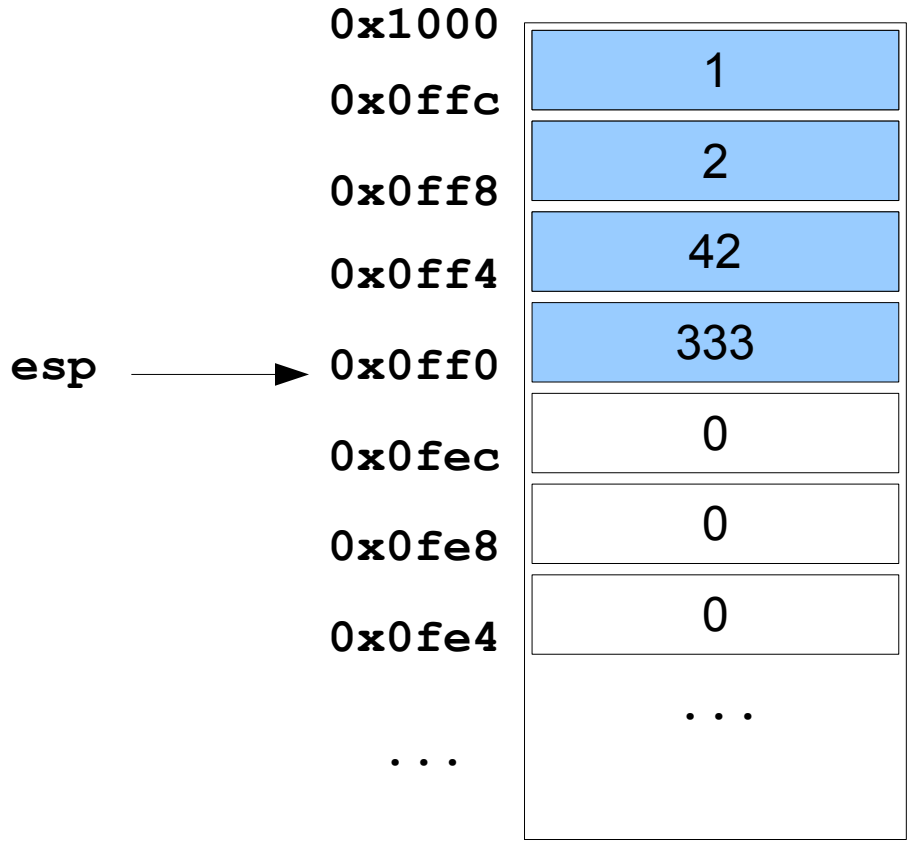
```
pushl $333

<=>

subl $4,%esp
movl $333,(%esp)
```



Indirect indexing

I.e.

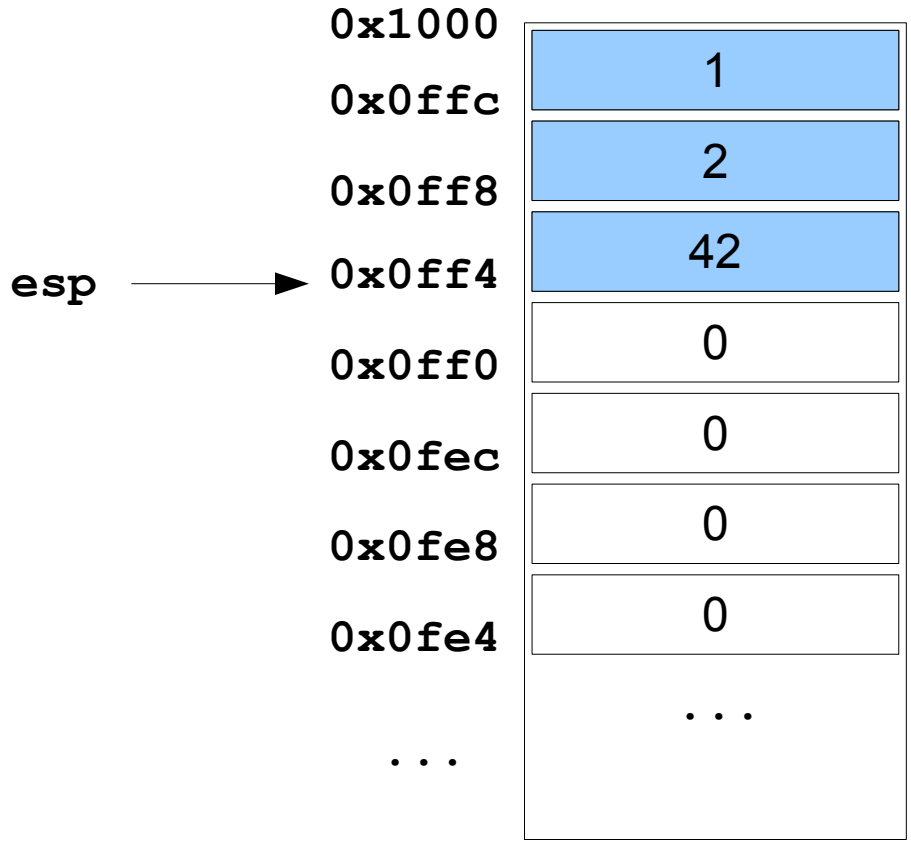
```
pushl $333

<=>

subl $4,%esp
movl $333,(%esp)
```



Indirect indexing

I.e.

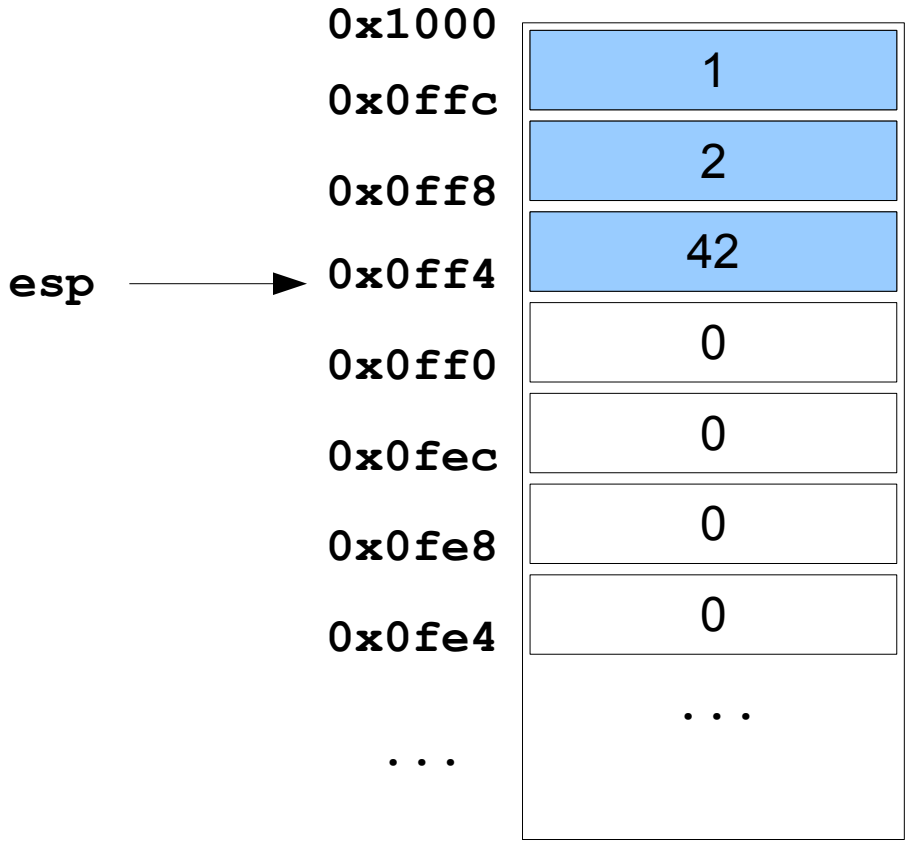
```
popl %eax

<=>

movl (%esp),%eax
addl $4,%esp
```



Indirect indexing

I.e.

```
popl %eax

<=>

movl (%esp),%eax
addl $4,%esp
```




```
eax: 42
```

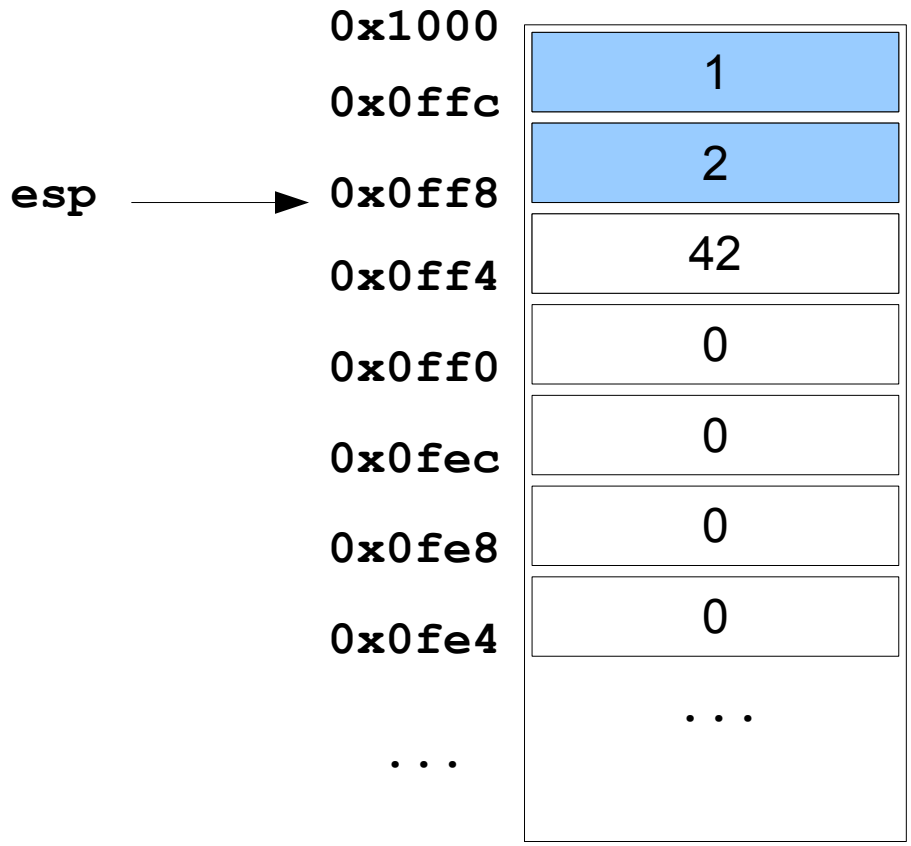
Indirect indexing

I.e.

```
popl %eax
```

```
<=>
```

```
movl (%esp),%eax
addl $4,%esp
```




```
eax: 42
```

Indirect indexing

And with ebp

```
movl %ebp, %eax
-> 0xff8
movl (%ebp),%eax
-> 2
movl 4(%ebp),%eax
-> 1
movl -4(%ebp),%eax
-> 42
```