CS 354
Machine Organization and Programming
Lecture 18

Michael Doescher
Summer 2020

Conditional Jumps
Iterative Structures
C --> Assembly
Stack Pointer
Load Effective Address

Conditional Jump

```
int a = 1, b=2;
if (a >= b) {
    stmts;
}
```

Conditional Jump

```
int a = 1, b=2;
if (a >= b) {
    stmts;
}
```

movl $1, %eax
movl $2, %ebx

Conditional Jump

```
int a = 1, b=2;
if (a >= b) {
    stmts;
}
```

```
movl $1, %eax
movl $2, %ebx
cmpl %ebx, %eax      (computes a-b.  What flags are set?)
                     (signed integers so SF, OF, and ZF)
```

Conditional Jump

```
int a = 1, b=2;
if (a >= b) {
    stmts;
}
L2:
```

```
movl $1, %eax
movl $2, %ebx
cmpl %ebx, %eax      (computes a-b.  What flags are set?)
jl L2                (signed integers so SF, OF, and ZF)
```

Conditional Jump

```
int a = 1, b=2;
if (a >= b) {
    stmts;
}
L2:
```

```
 1  001
-2  010
-1  111
```

movl $1, %eax
movl $2, %ebx
cmpl %ebx, %eax
jl L2

Conditional Jump

```
int a = 1, b=2;
if (a >= b) {
    stmts;
}
L2:
```

```
movl $1, %eax
movl $2, %ebx
cmpl %ebx, %eax
jl L2
```

```
 1  001
-2  010
-1  111

result is negative :: SF = 1
```

Conditional Jump

```
int a = 1, b=2;
if (a >= b) {
    stmts;
}
L2:
```

```
movl $1, %eax
movl $2, %ebx
cmpl %ebx, %eax
jl L2
```

```
 1  001
-2  010
-1  111

result is negative :: SF = 1
we did borrow (not important for signed math)
```

Conditional Jump

```
int a = 1, b=2;
if (a >= b) {
    stmts;
}
L2:
```

```
movl $1, %eax
movl $2, %ebx
cmpl %ebx, %eax
jl L2
```

```
 1  001
-2  010
-1  111

result is negative :: SF = 1
we did borrow (not important for signed math)
No overflow (a<0, b>0, result>0) or
            (a>0, b<0, result<0) :: OF = 0
```

Conditional Jump

```
int a = 1, b=2;
if (a >= b) {
    stmts;
}
L2:
```

```
movl $1, %eax
movl $2, %ebx
cmpl %ebx, %eax
jl L2
```

```
 1  001
-2  010
-1  111

result is negative :: SF = 1
we did borrow (not important for signed math)
No overflow (a<0, b>0, result>0) or
            (a>0, b<0, result<0) :: OF = 0

jl condition  SF ^ OF
               1 ^ 0 = 1
```

Conditional Jump

```
int a = -4, b=3;
if (a >= b) {
    stmts;
}
L2:
```

```
-4   100
-3   011
 1   001
```

```
movl $1, %eax
movl $2, %ebx
cmpl %ebx, %eax
jl L2
```

Conditional Jump

```
int a = -4, b=3;
if (a >= b) {
    stmts;
}
L2:
```

```
-4  100
-3  011
 1  001

result is positive :: SF = 0
```

```
movl $1, %eax
movl $2, %ebx
cmpl %ebx, %eax
jl L2
```

Conditional Jump

```
int a = -4, b=3;
if (a >= b) {
    stmts;
}
L2:
```

```
-4  100
-3  011
 1  001

result is positive :: SF = 0
Did overflow (a<0, b>0, result>0) :: OF = 1
```

```
movl $1, %eax
movl $2, %ebx
cmpl %ebx, %eax
jl L2
```

Conditional Jump

```
int a = -4, b=3;
if (a >= b) {
    stmts;
}
L2:
```

```
movl $1, %eax
movl $2, %ebx
cmpl %ebx, %eax
jl L2
```

```
-4  100
-3  011
 1  001

result is positive :: SF = 0
Did overflow (a<0, b>0, result>0) :: OF = 1

jl condition  SF ^ OF
               0 ^ 1 = 1
```

Conditional Jump

```
int a = -4, b=3;
if (a >= b) {
    stmts;
}
L2:
```

```
movl $1, %eax
movl $2, %ebx
cmpl %ebx, %eax
jl L2
```

```
-4  100
-3  011
 1  001

result is positive :: SF = 0
Did overflow (a<0, b>0, result>0) :: OF = 1

jl condition  SF ^ OF
               0 ^ 1 = 1


NUMBER LINE LOGIC
-4, -3, -2, -1, 0, 1, 2, 3
```

JUMP CONDITIONS

```
jmp
je
jne
jl          SF^OF
jle
jg
jge

jb
jbe
ja
jae
```

JUMP CONDITIONS

jmp	No Flag Requirements
je	
jne	
jl	SF^OF
jle	
jg	
jge	
jb	
jbe	
ja	
jae	

JUMP CONDITIONS

jmp	No Flag Requirements
je	ZF
jne	
jl	SF^OF
jle	
jg	
jge	
jb	
jbe	
ja	
jae	

JUMP CONDITIONS

jmp	No Flag Requirements
je	ZF
jne	~ZF
jl	SF^OF
jle	
jg	
jge	
jb	
jbe	
ja	
jae	

JUMP CONDITIONS

```
jmp       No Flag Requirements
je        ZF
jne       ~ZF
jl        SF^OF
jle       SF^OF | ZF
jg
jge

jb
jbe
ja
jae
```

JUMP CONDITIONS

jmp	No Flag Requirements
je	ZF
jne	~ZF
jl	SF^OF
jle	SF^OF | ZF
jg	~(SF^OF)
jge	
jb	
jbe	
ja	
jae	

JUMP CONDITIONS

```
jmp       No Flag Requirements
je        ZF
jne       ~ZF
jl        SF^OF
jle       SF^OF | ZF
jg        ~(SF^OF)
jge       ~(SF^OF) | ZF

jb
jbe
ja
jae
```

JUMP CONDITIONS

```
jmp       No Flag Requirements
je        ZF
jne       ~ZF
jl        SF^OF
jle       SF^OF | ZF
jg        ~(SF^OF)
jge       ~(SF^OF) | ZF

jb         CF
jbe
ja
jae
```

JUMP CONDITIONS

```
jmp       No Flag Requirements
je        ZF
jne       ~ZF
jl        SF^OF
jle       SF^OF | ZF
jg        ~(SF^OF)
jge       ~(SF^OF) | ZF

jb         CF
jbe        CF | ZF
ja
jae
```

JUMP CONDITIONS

```
jmp       No Flag Requirements
je        ZF
jne       ~ZF
jl        SF^OF
jle       SF^OF | ZF
jg        ~(SF^OF)
jge       ~(SF^OF) | ZF

jb         CF
jbe        CF | ZF
ja         ~CF & ~ZF
jae
```

JUMP CONDITIONS

```
jmp       No Flag Requirements
je        ZF
jne       ~ZF
jl        SF^OF
jle       SF^OF | ZF
jg        ~(SF^OF)
jge       ~(SF^OF) | ZF

jb         CF
jbe        CF | ZF
ja         ~CF & ~ZF
jae        ~CF
```

Loop Transformations

```
for (init; cond; post-expr) {
    loop_body;
}

while (cond) {
    loop_ body;
}

do {
    loop_ body;
} while (cond);
```

Loop Transformations

```
for (int i = 0; i<5; i++) {
    sum += i;
}
```

```
int i = 0;
while (i<5) {
    sum += i;
    i++;
}
```

Loop Transformations

```
for (int i = 0; i<5; i++) {
    sum += i;
}
```

```
int i = 0;
while (i<5) {
    sum += i;
    i++;
}
```

```
int i = 0;
if (i<5)
  do {
      sum += i;
      i++;
  } while (i<5);
```

Goto Loop Transformations

```
for (int i = 0; i<5; i++) {
    sum += i;
}

int i = 0;
L2:
if (i < 5) {
      sum +=i
      i++;
      goto L2;
}
```

Goto Loop Transformations

```
int i = 0;
while (i<5) {
    sum += i;
    i++;
}
```

```
int i = 0;
L2:
if (i < 5) {
      sum +=i
      i++;
      goto L2;
}
```

Goto Loop Transformations

```
int i = 0;
if (i<5)
  do {
     sum += i;
     i++;
  } while (i<5);
```

```
int i = 0;
L2:
if (i < 5) {
     sum +=i
     i++;
     goto L2;
}
```

Loops -> Assembly

```
do {
    sum += i;
    i++;
} while (i<5);
```

```
L2:
addl %eax, %ebx
inc %eax
cmpl $5, %eax
jl L2
```

Loops -> Assembly

```
do {
    sum += i;
    i++;
} while (i<5);
```

```
L2:
addl %eax, %ebx
inc %eax
cmpl $5, %eax
jl L2
```

```
while (i<5) {
    sum+=i;
    i++;
}
```

```
jmp L4
L2:
addl %eax, %ebx
inc %eax
L4:
cmpl $5, %eax
jl L2
```

Condition Code Register : %eflags

- can't access this register directly 'hidden'

SET instructions - sets a single byte to 0 or 1

sete	setl	seta
setne	setle	setae
	setg	setb
	setge	setbe

Condition Code Register : %eflags

```
assume %eax contains $1, and %ebx contains $2

cmpl %ebx, %eax
```

Condition Code Register : %eflags

```
assume %eax contains $1, and %ebx contains $2

cmpl %ebx, %eax  (computes a-b and sets %eflags)
```

Condition Code Register : %eflags

```
assume %eax contains $1, and %ebx contains $2

cmpl %ebx, %eax  (computes a-b and sets %eflags)
1-2 = -1
```

Condition Code Register : %eflags

```
assume %eax contains $1, and %ebx contains $2

cmpl %ebx, %eax  (computes a-b and sets %eflags)
1-2 = -1
  SF = 1
  OF = 0
  SF ^ OF = 1
```

Condition Code Register : %eflags

```
assume %eax contains $1, and %ebx contains $2

cmpl %ebx, %eax  (computes a-b and sets %eflags)
1-2 = -1  :: SF = 1, OF = 0  -> SF^OF = 1

setl %al
%eax[........ ........][........ 00000001]
```

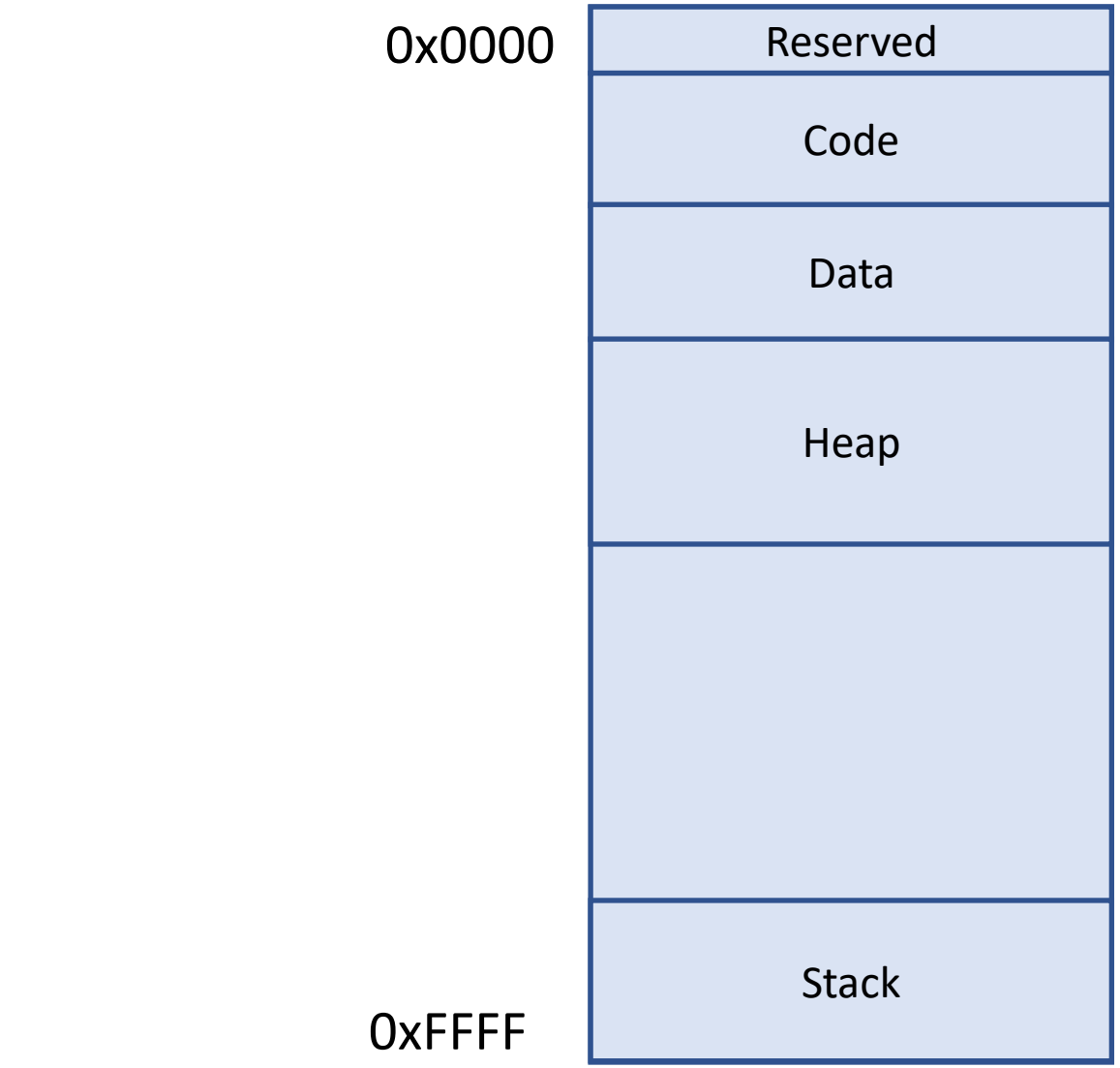
Condition Code Register : %eflags

```
assume %eax contains $1, and %ebx contains $2

cmpl %ebx, %eax  (computes a-b and sets %eflags)
1-2 = -1  :: SF = 1, OF = 0  -> SF^OF = 1

setl %al
%eax[........ ........][........ 00000001]

movzbl
   byte to long
   zero extension or sign extension (movsbl)
```

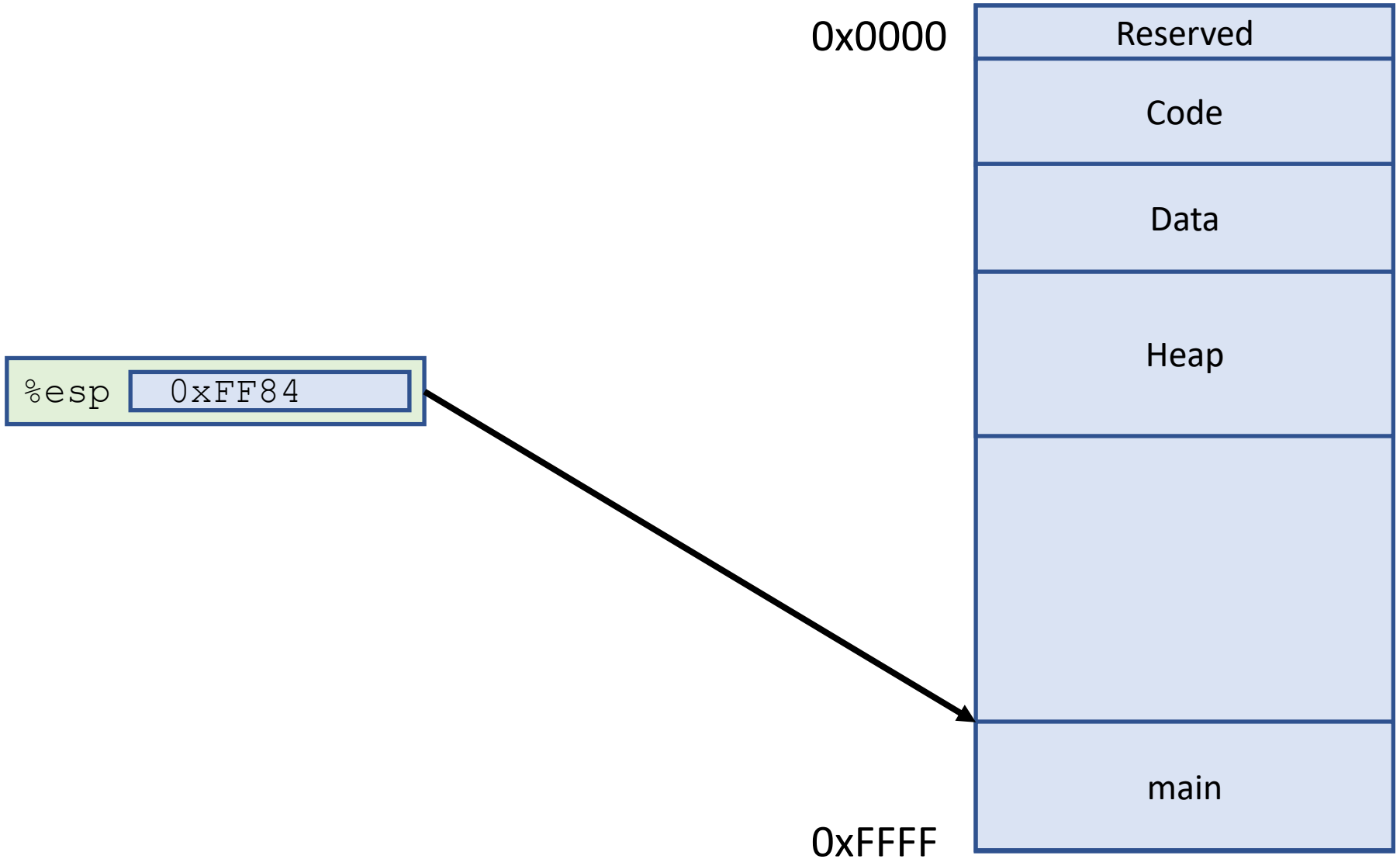
Condition Code Register : %eflags

```
assume %eax contains $1, and %ebx contains $2

cmpl %ebx, %eax  (computes a-b and sets %eflags)
1-2 = -1  :: SF = 1, OF = 0  -> SF^OF = 1

setl %al
%eax[........ ........][........ 00000001]

movzbl (byte to long) zero extension or sign extension

movzbl %al, %ebx
%ebx[00000000 00000000][00000000 00000001]
```

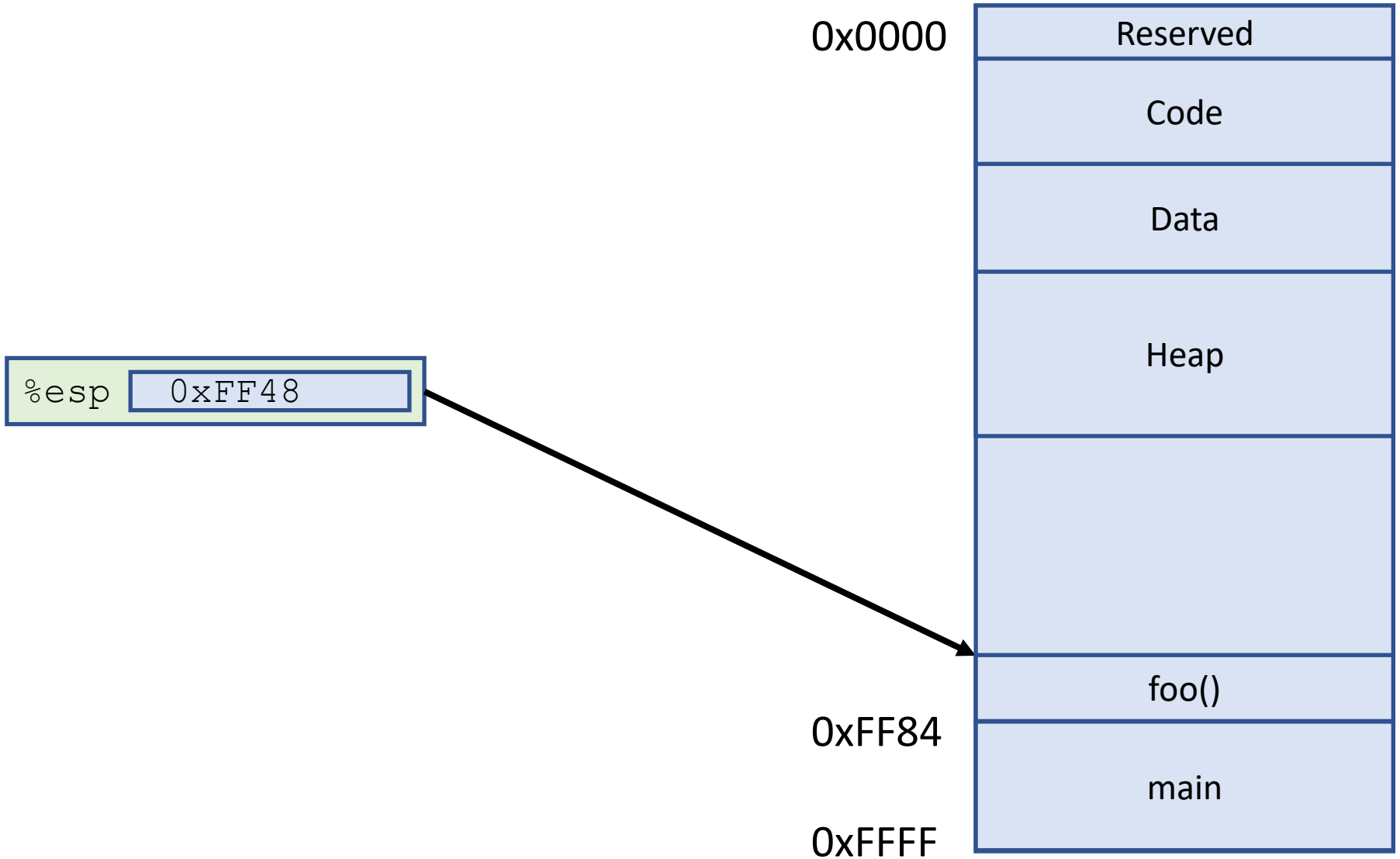
Memory Address Space



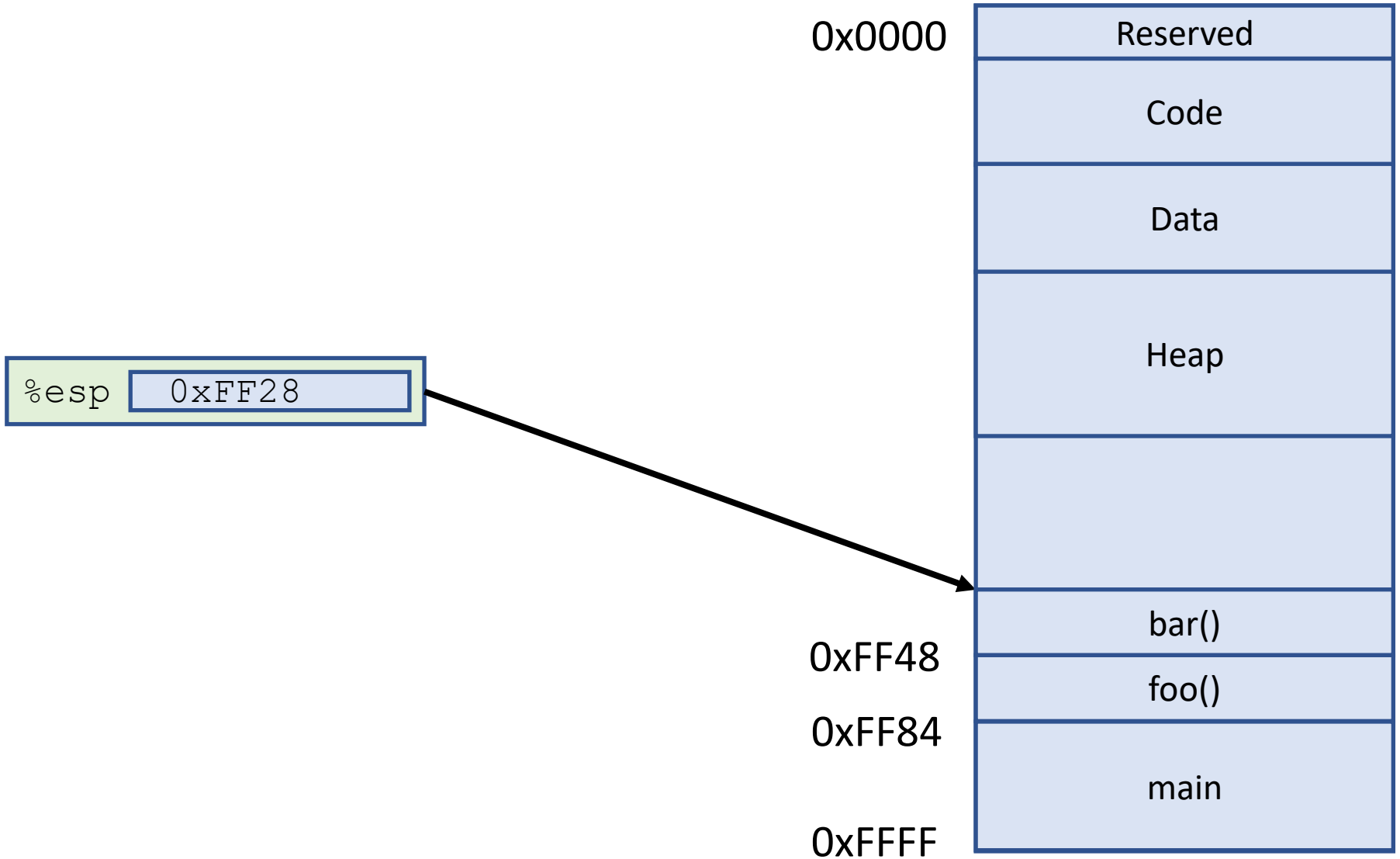
Memory Address Space



Memory Address Space



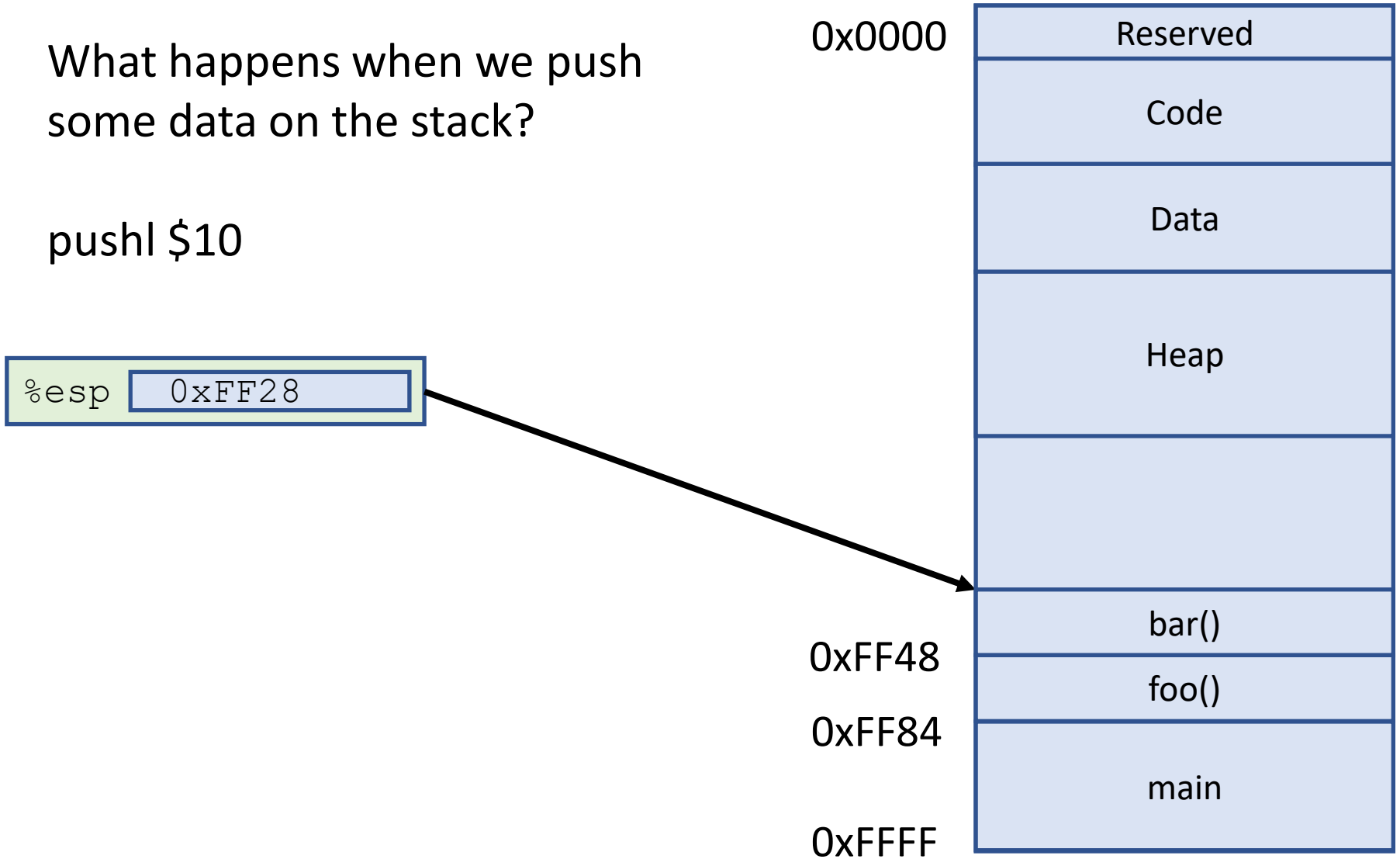
Memory Address Space



Memory Address Space

What happens when we push some data on the stack?

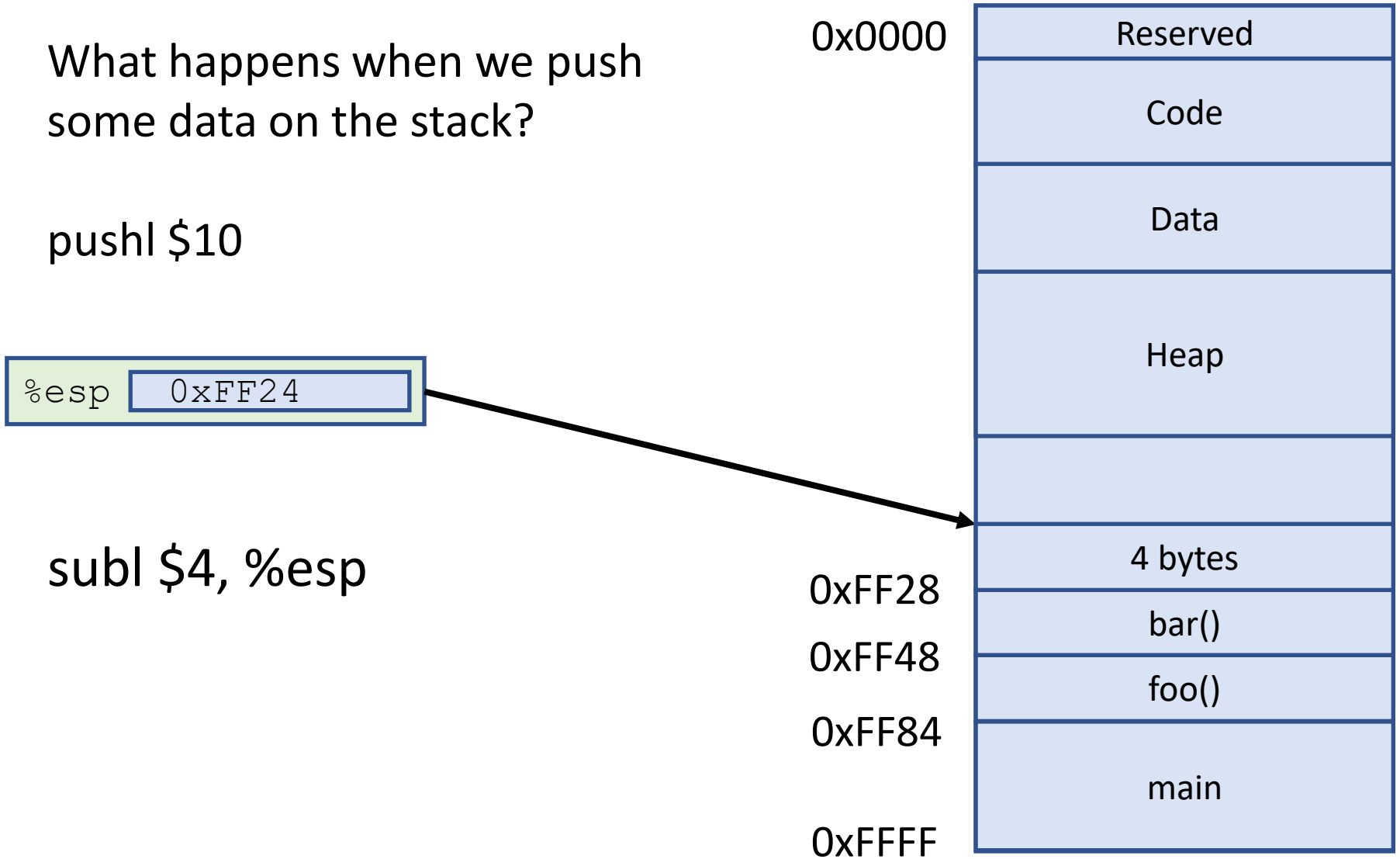
pushl $10



Memory Address Space

What happens when we push some data on the stack?

pushl $10

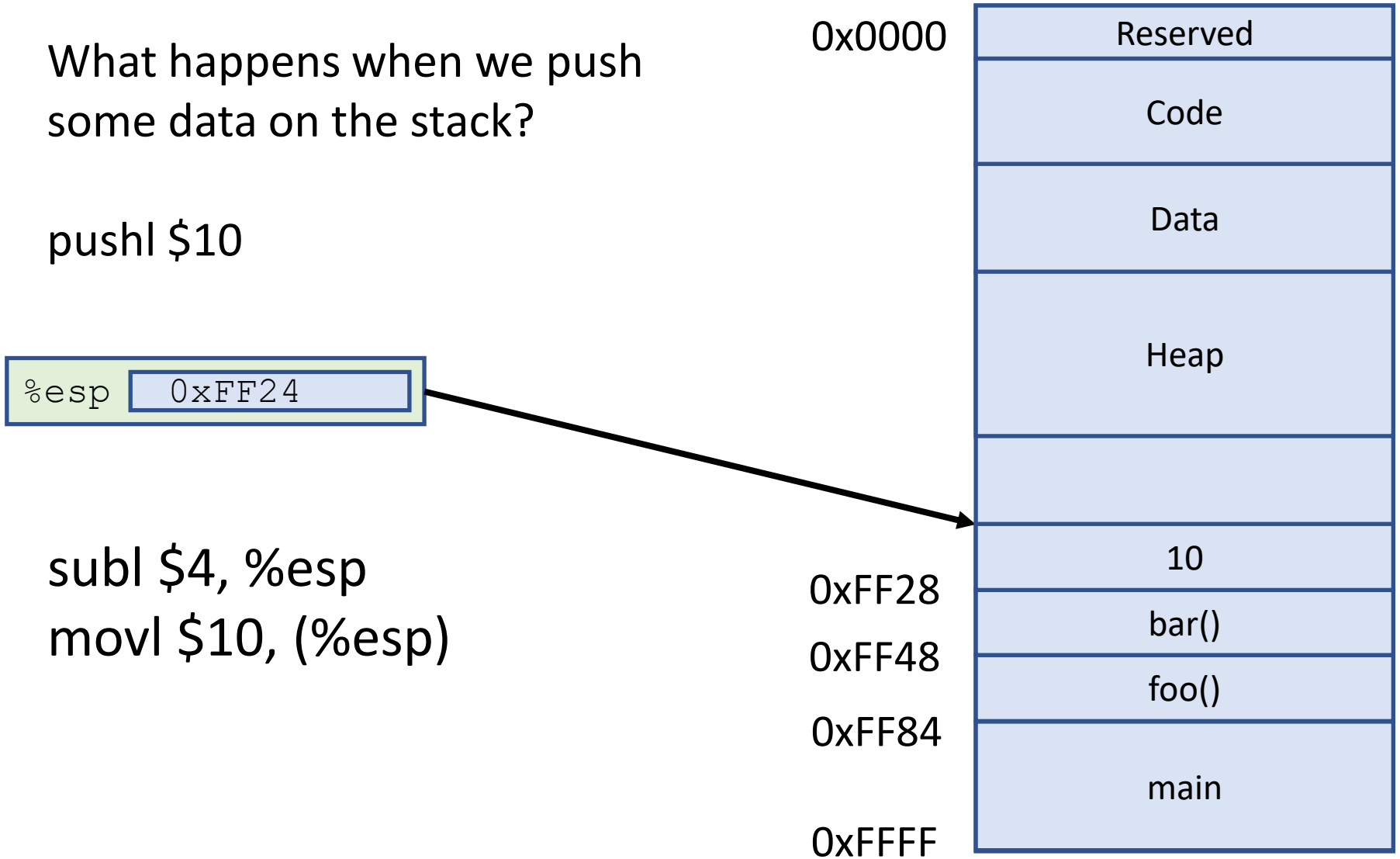


subl $4, %esp

Memory Address Space

What happens when we push some data on the stack?

pushl $10

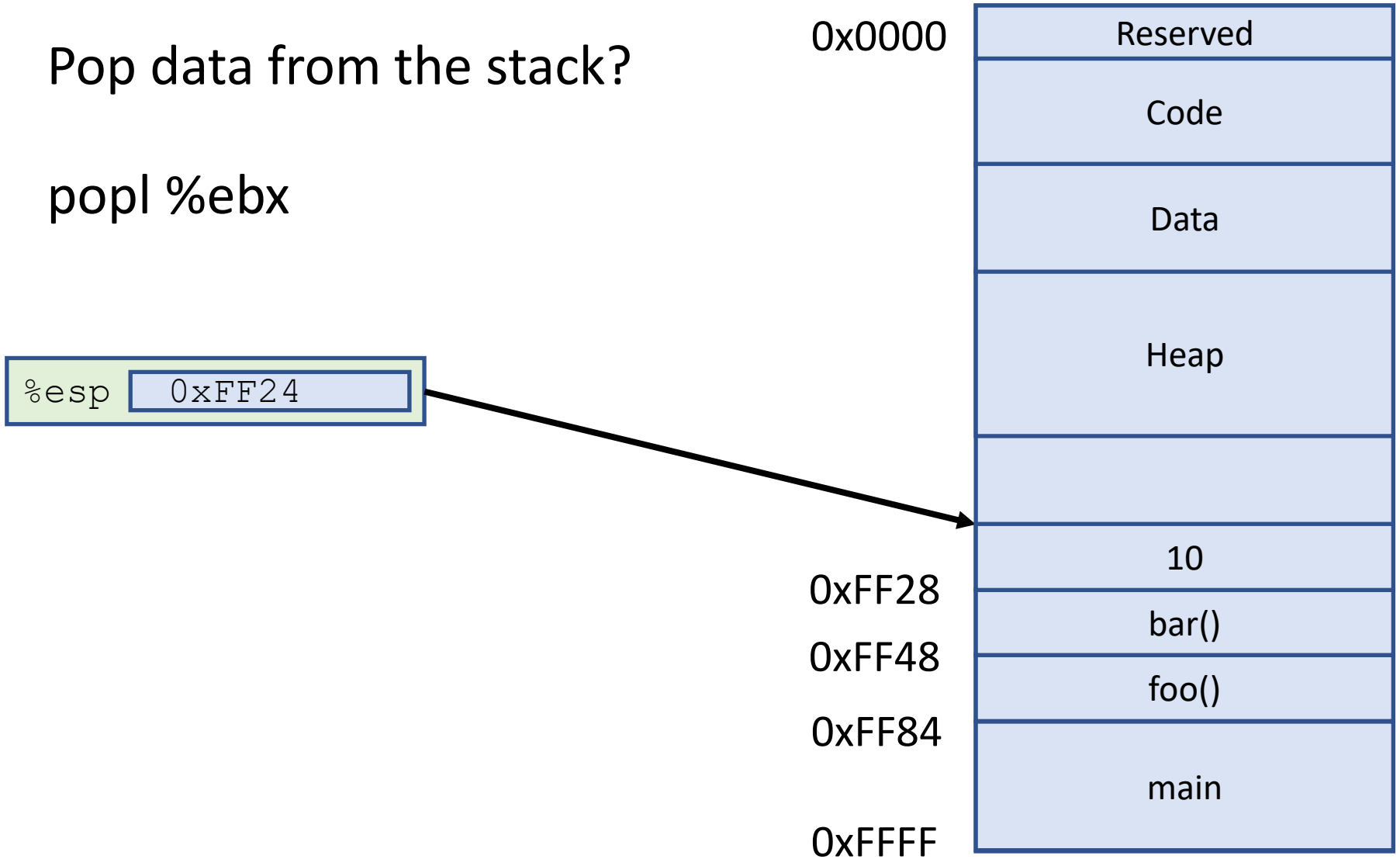


subl $4, %esp
movl $10, (%esp)

Memory Address Space

Pop data from the stack?

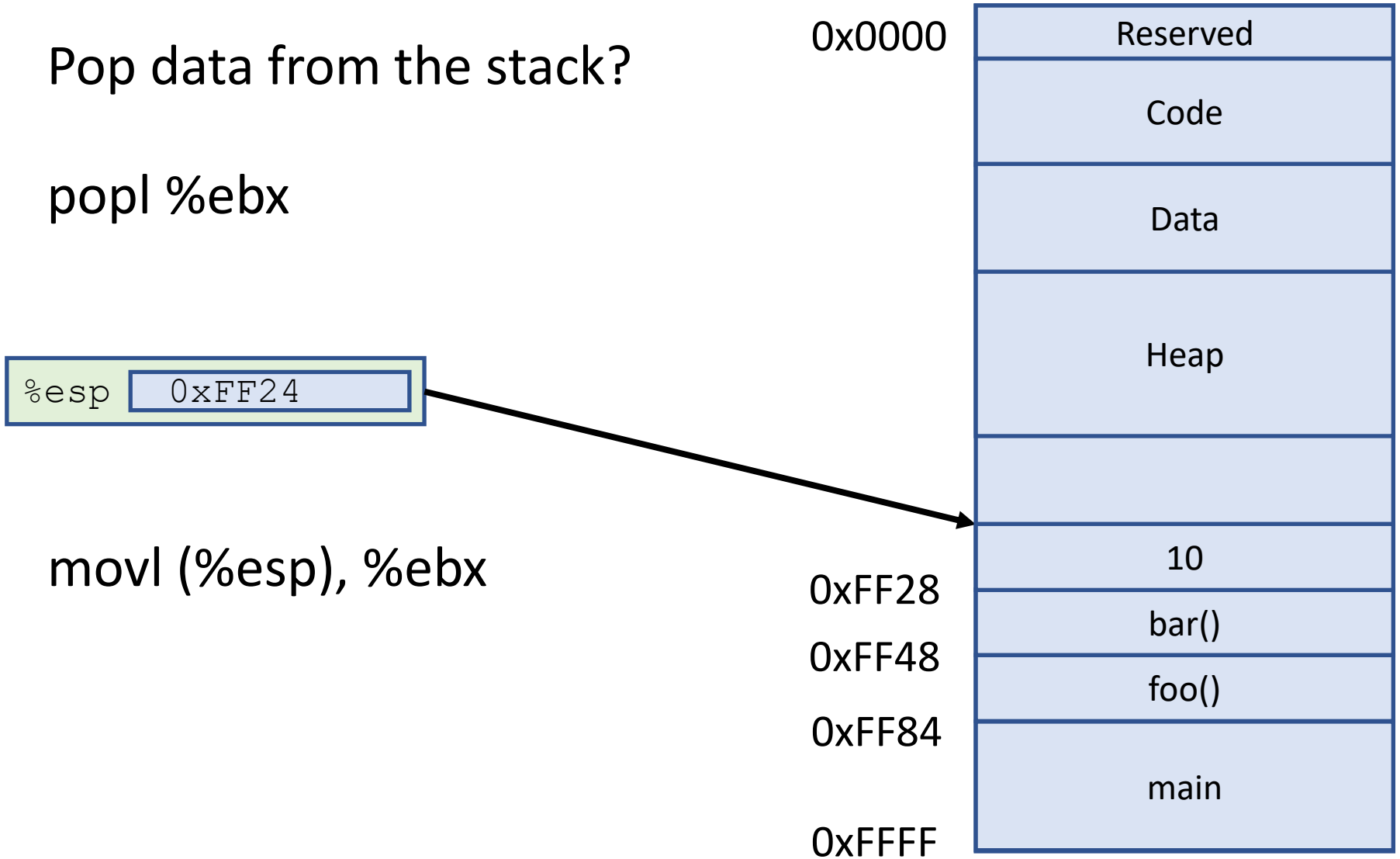
popl %ebx



Memory Address Space

Pop data from the stack?

popl %ebx

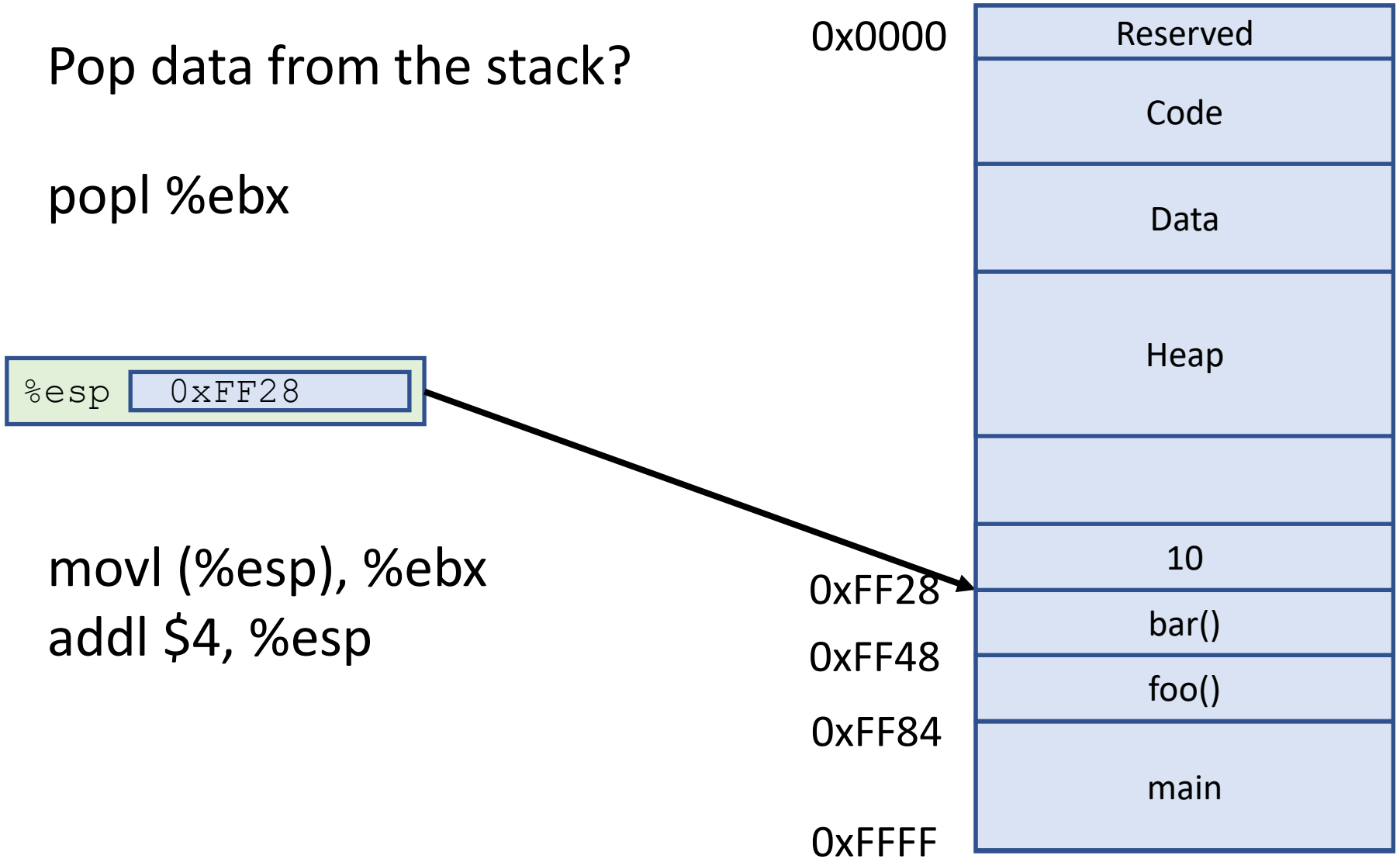


movl (%esp), %ebx

Memory Address Space

Pop data from the stack?

popl %ebx



movl (%esp), %ebx
addl $4, %esp

CS 354
Machine Organization and Programming
Lecture 18

Michael Doescher
Summer 2020

Conditional Jumps
Iterative Structures
C --> Assembly
Stack Pointer
Load Effective Address