CS33 Discussion 3

Procedure Calls

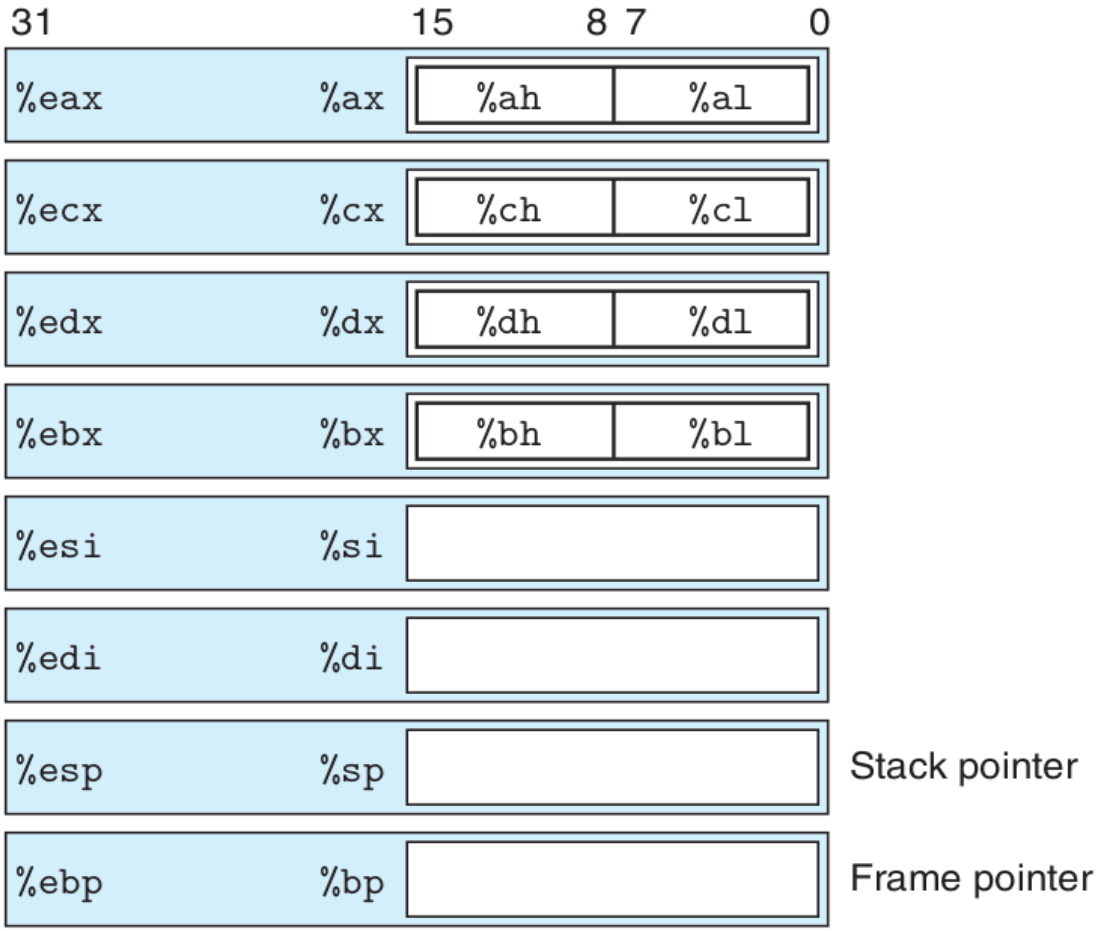
Brandon Wu

10.24.2014

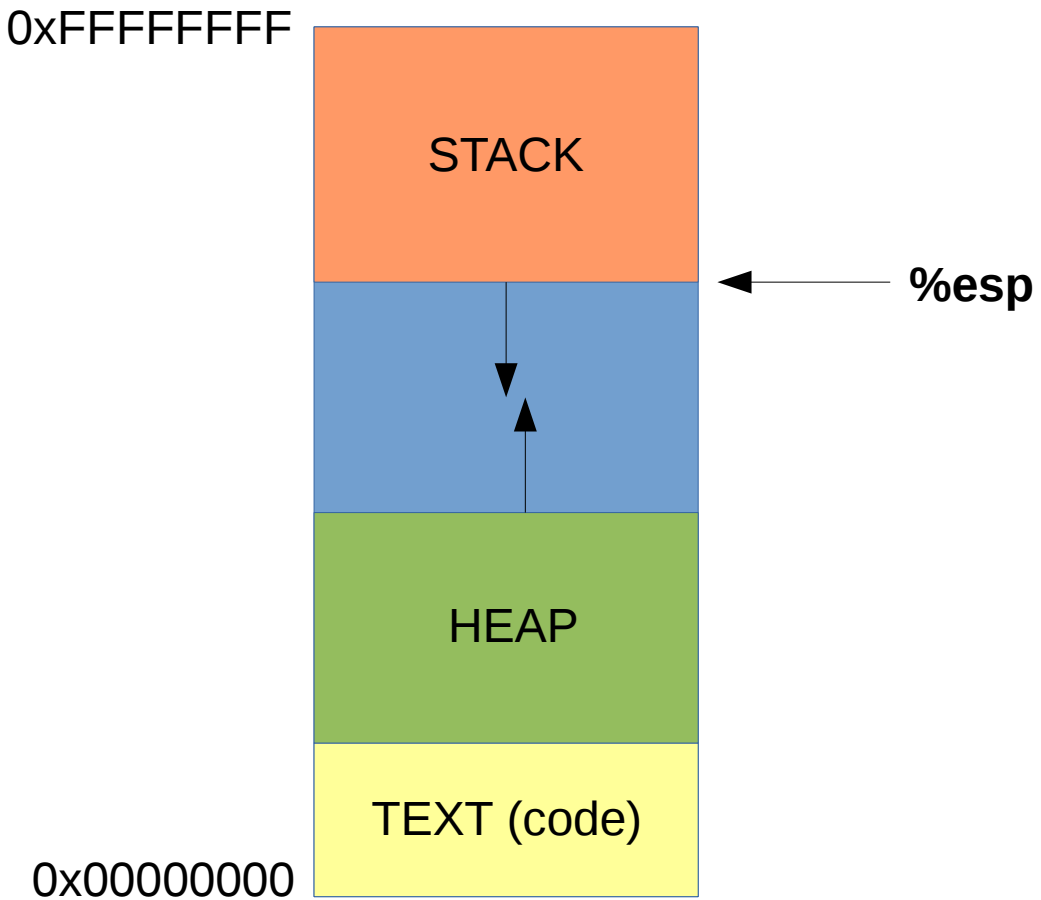
Last Time...

- IEEE Floating Point
- Intro to Machine Code
  - Registers
  - Addressing Modes
  - Arithmetic Operations

The registers again...



Stack Recap

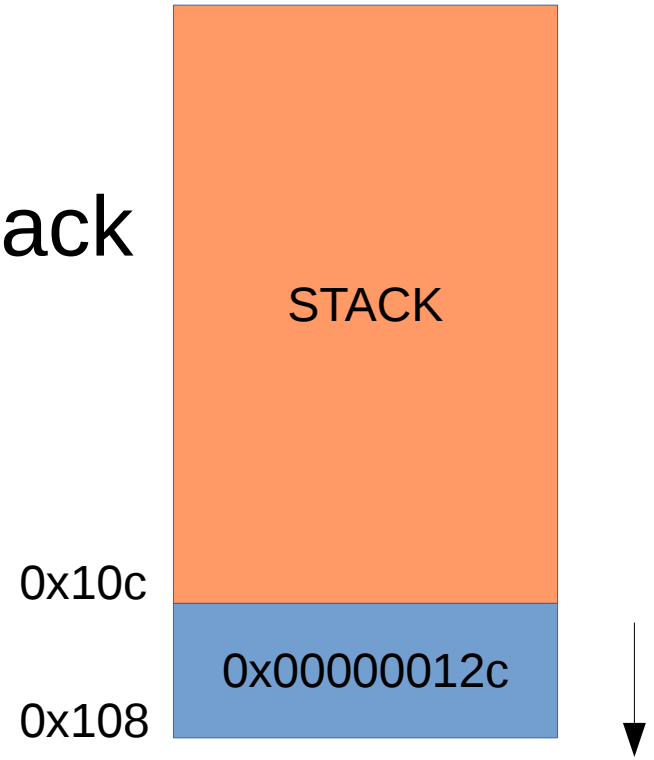


Push

pushl <src>

- Push the value to top of stack
- Decrement %esp

pushl %ebp



Before:

%ebp = 0x12c
%esp = 0x10c

After:

%ebp = 0x12c
%esp = 0x108

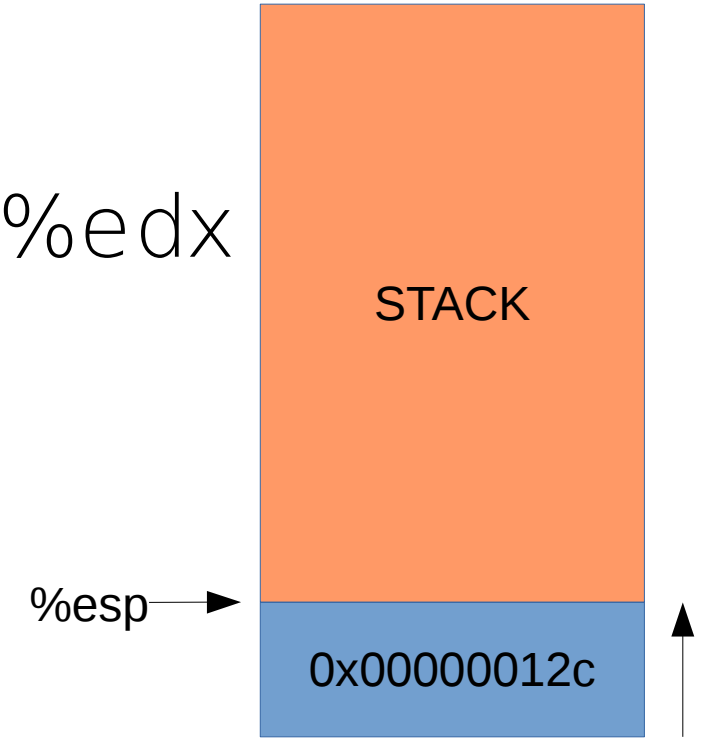
Pop

popl %edx

- Store top of stack into %edx
- Increment stack ptr

Before:
%edx = 0x2AB
%esp = 0x108

After:
%edx = 0x12c
%esp = 0x10c



Procedure Calls

Example 1

```
int doSomething(int a, int b) {
        return a+b;
}
```

- Compile: gcc -c example1.c -g
- Dissasemble obj file:
  - objdump -d example1.o > example1.s
- Build executable:
  - gcc -o example1 example1.o

Example 1: Dissasembly

```
0000004c <doSomething>:
  4c:   55              push   %ebp
  4d:   89 e5           mov    %esp,%ebp
  4f:   8b 45 0c        mov    0xc(%ebp),%eax
  52:   8b 55 08        mov    0x8(%ebp),%edx
  55:   01 d0           add    %edx,%eax
  57:   5d              pop    %ebp
  58:   c3              ret
```

Instruction

Instruction Address Offset

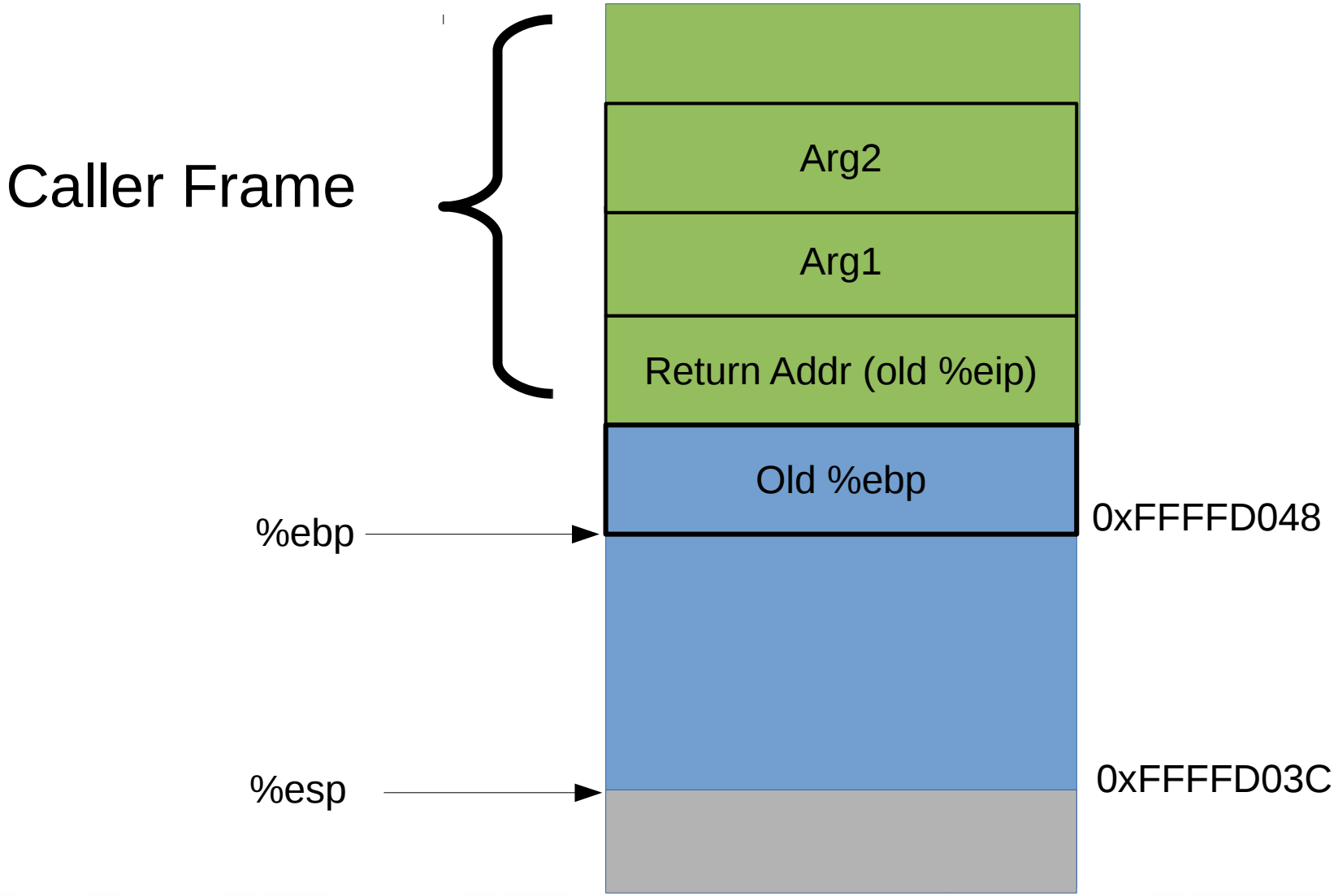
“Human Readable” Format

Questions...

- How do we pass parameters?
- What executes after the return?
- Where does the return value go?

- **What happens when I make a function call?**

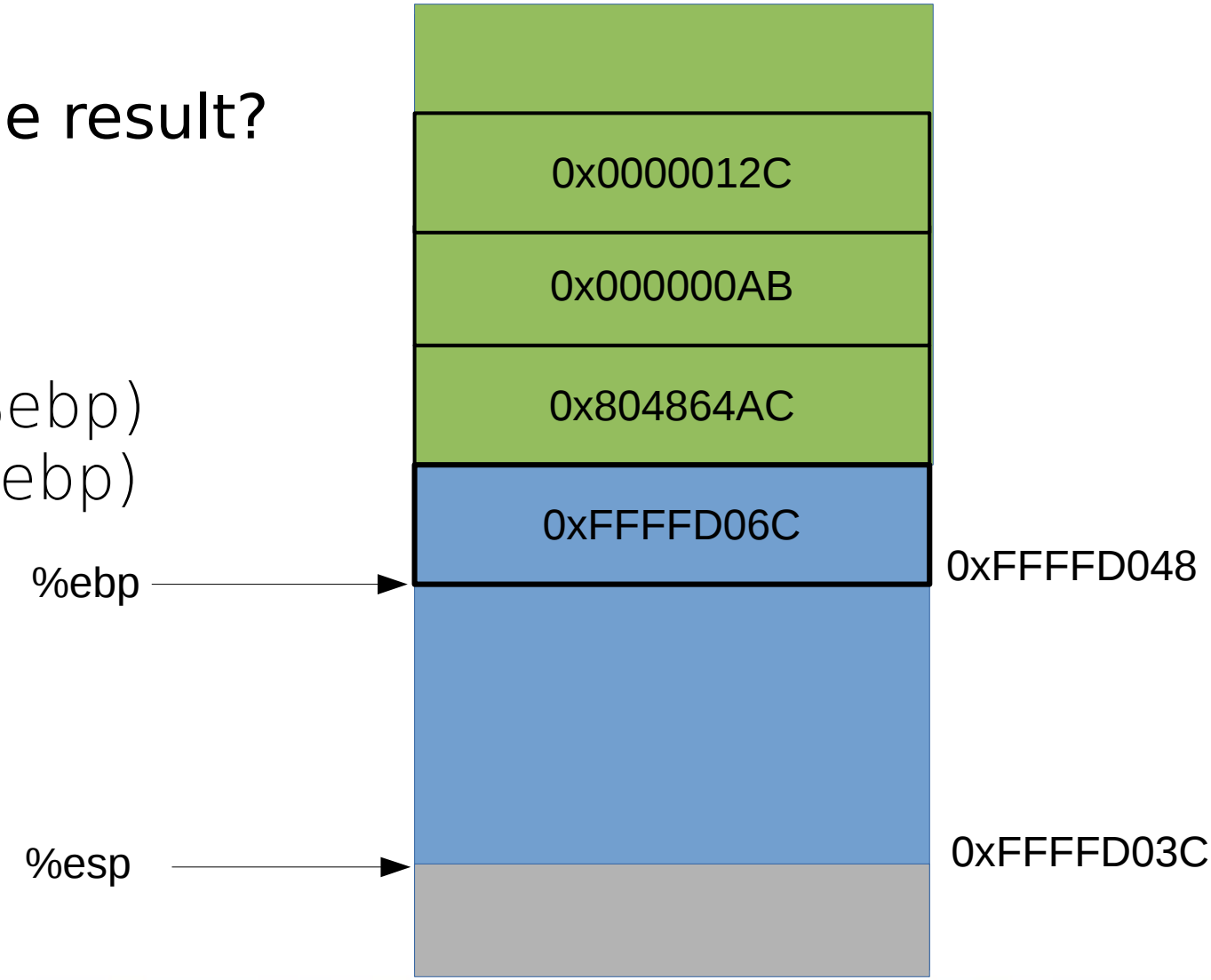
Stack Frames



Example 2

What is the result?

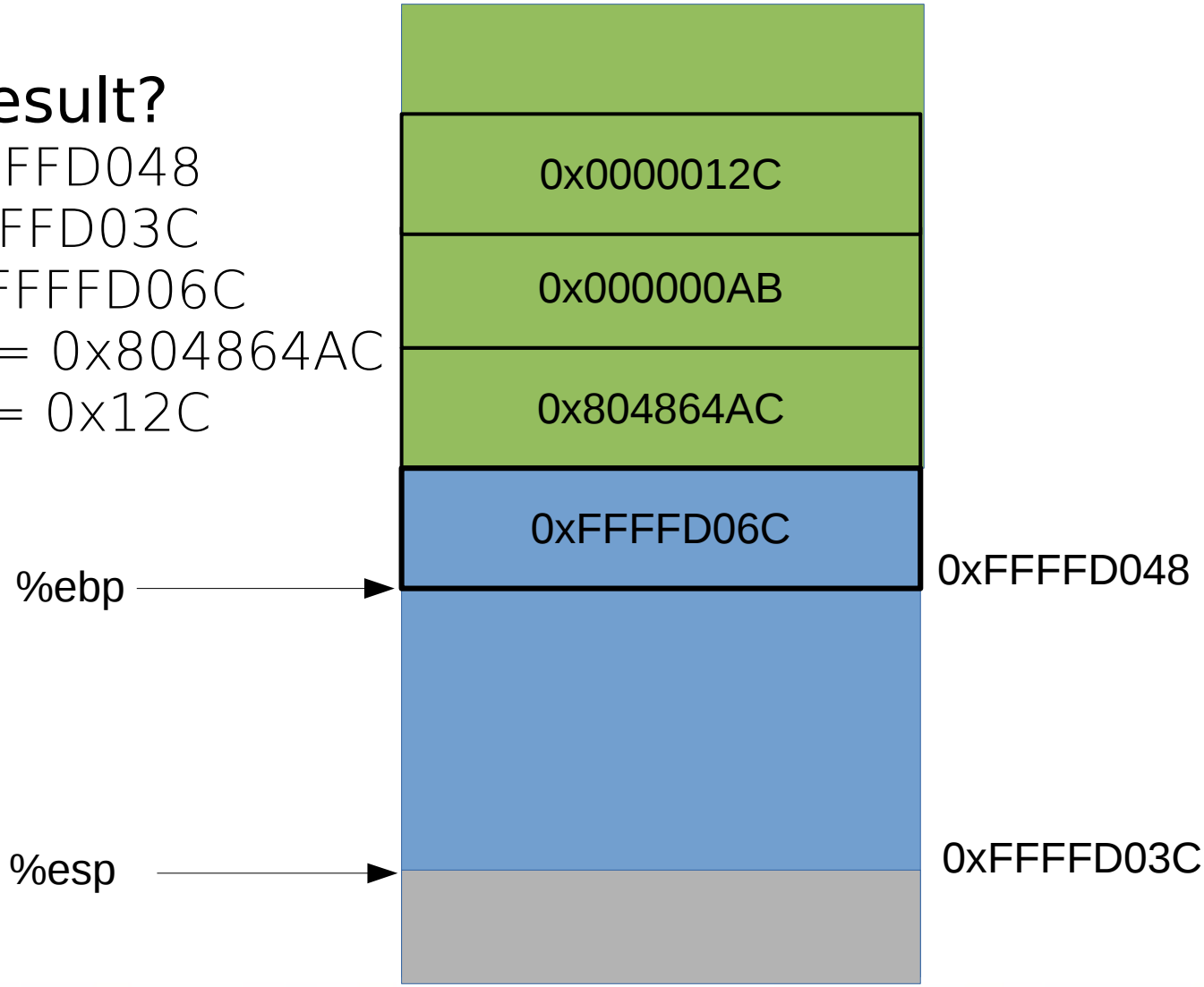
1. %ebp
2. %esp
3. (%ebp)
4. $0x4(%ebp)
5. $0xc(%ebp)



Example 2

What is the result?

1. %ebp = 0xFFFFD048
2. %esp = 0xFFFFD03C
3. (%ebp) = 0xFFFFD06C
4. $0x4(%ebp) = 0x804864AC
5. $0xc(%ebp) = 0x12C



Example 1 Revisted

```
0000004c <doSomething>:
  4c:   55              push   %ebp
  4d:   89 e5           mov    %esp,%ebp
  4f:   8b 45 0c        mov    0xc(%ebp),%eax
  52:   8b 55 08        mov    0x8(%ebp),%edx
  55:   01 d0           add    %edx,%eax
  57:   5d              pop    %ebp
  58:   c3              ret
```

Save base ptr of Caller

%eip = 0x4c

Example 1 Revisted

```
0000004c <doSomething>:
  4c:   55              push   %ebp
  4d:   89 e5           mov    %esp,%ebp
  4f:   8b 45 0c        mov    0xc(%ebp),%eax
  52:   8b 55 08        mov    0x8(%ebp),%edx
  55:   01 d0           add    %edx,%eax
  57:   5d              pop    %ebp
  58:   c3              ret
```

Update BP to current frame

%eip = 0x4d

Example 1 Revisted

```
0000004c <doSomething>:
  4c:   55              push   %ebp
  4d:   89 e5           mov    %esp,%ebp
  4f:   8b 45 0c        mov    0xc(%ebp),%eax
  52:   8b 55 08        mov    0x8(%ebp),%edx
  55:   01 d0           add    %edx,%eax
  57:   5d              pop    %ebp
  58:   c3              ret
```

Grab arguments

%eip = 0x4f

Example 1 Revisted

```
0000004c <doSomething>:
  4c:   55                  push   %ebp
  4d:   89 e5               mov    %esp,%ebp
  4f:   8b 45 0c            mov    0xc(%ebp),%eax
  52:   8b 55 08            mov    0x8(%ebp),%edx
  55:   01 d0               add    %edx,%eax
  57:   5d                  pop    %ebp
  58:   c3                  ret
```

Compute

%eip = 0x55

Example 1 Revisted

```
0000004c <doSomething>:
  4c:   55              push   %ebp
  4d:   89 e5           mov    %esp,%ebp
  4f:   8b 45 0c        mov    0xc(%ebp),%eax
  52:   8b 55 08        mov    0x8(%ebp),%edx
  55:   01 d0           add    %edx,%eax
  57:   5d              pop    %ebp
  58:   c3              ret
```

Restore caller frame

%eip = 0x5d

Example 1 Revisted

```
0000004c <doSomething>:
  4c:   55                  push   %ebp
  4d:   89 e5               mov    %esp,%ebp
  4f:   8b 45 0c            mov    0xc(%ebp),%eax
  52:   8b 55 08            mov    0x8(%ebp),%edx
  55:   01 d0               add    %edx,%eax
  57:   5d                  pop    %ebp
  58:   c3                  ret
```

Restore control to caller

%eip <= old %eip

Some Remarks

- Why does this code not allocate stack space?
- How do I (the caller) get the return value?
- Need to do some work on **ret**
  - Set %eip to %eip of caller's next instruction
  - e.g %eip = old %eip + 0x5;

Intro To GDB: The GNU Debugger

- Run executable “example1” in gdb:

  ```
  gdb example1
  ```

GDB Basics

info breakpoints

info registers

- Print out information on breakpoints/registers

b main

- Insert breakpoint at label “main”

run

- Start execution until next breakpoint

GDB Continued

set disassemble-next-line on

- Shows disassembly of nxt instruction

disas main

- Shows disassembly of “main”

stepi/nexti

- “step into”/ “step over”

p $eax

- Print value of register %eax

GDB Continued 2

x $eax

- Print value at (%eax)

x $ebp+0x8

- Print value at 0x8($ebp)

x/10b $ecx

- Print 10 bytes starting at memory location %ecx
- Can also specify h (half words), w (words), g (giant words), s (string)

Extra Practice: What does this code do?

```
00000040 <mystery>:
  40:    55                push   %ebp
  41:    89 e5             mov    %esp,%ebp
  43:    83 ec 10          sub    $0x10,%esp
  46:    8b 45 08          mov    0x8(%ebp),%eax
  49:    8b 10             mov    (%eax),%edx
  4b:    8b 45 0c          mov    0xc(%ebp),%eax
  4e:    89 c1             mov    %eax,%ecx
  50:    d3 e2             shl    %cl,%edx
  52:    89 d0             mov    %edx,%eax
  54:    89 45 fc          mov    %eax,-0x4(%ebp)
  57:    8b 45 fc          mov    -0x4(%ebp),%eax
  5a:    c3                ret
```