Lec 7: Machine Code (2)

CS492E: Introduction to Software Security

Sang Kil Cha

Function: Adding an Abstraction Layer

```
int Red(int a1)
{
  int r = 0;
  return r + Blue(a1 - 42);
}
```

```
int Blue(int a1)
{
  int b = 1;
  return b + Purple(a1, b);
}
```

```
int Purple(int a1, int a2)
{
  int p = 2;
  return p + a1 – a2;
}
```

Function: Adding an Abstraction Layer

```
int Red(int a1)
{
  int r = 0;
  return r + Blue(a1 - 42);
}
```

```
int Blue(int a1)
{
  int b = 1;
  return b + Purple(a1, b);
}
```

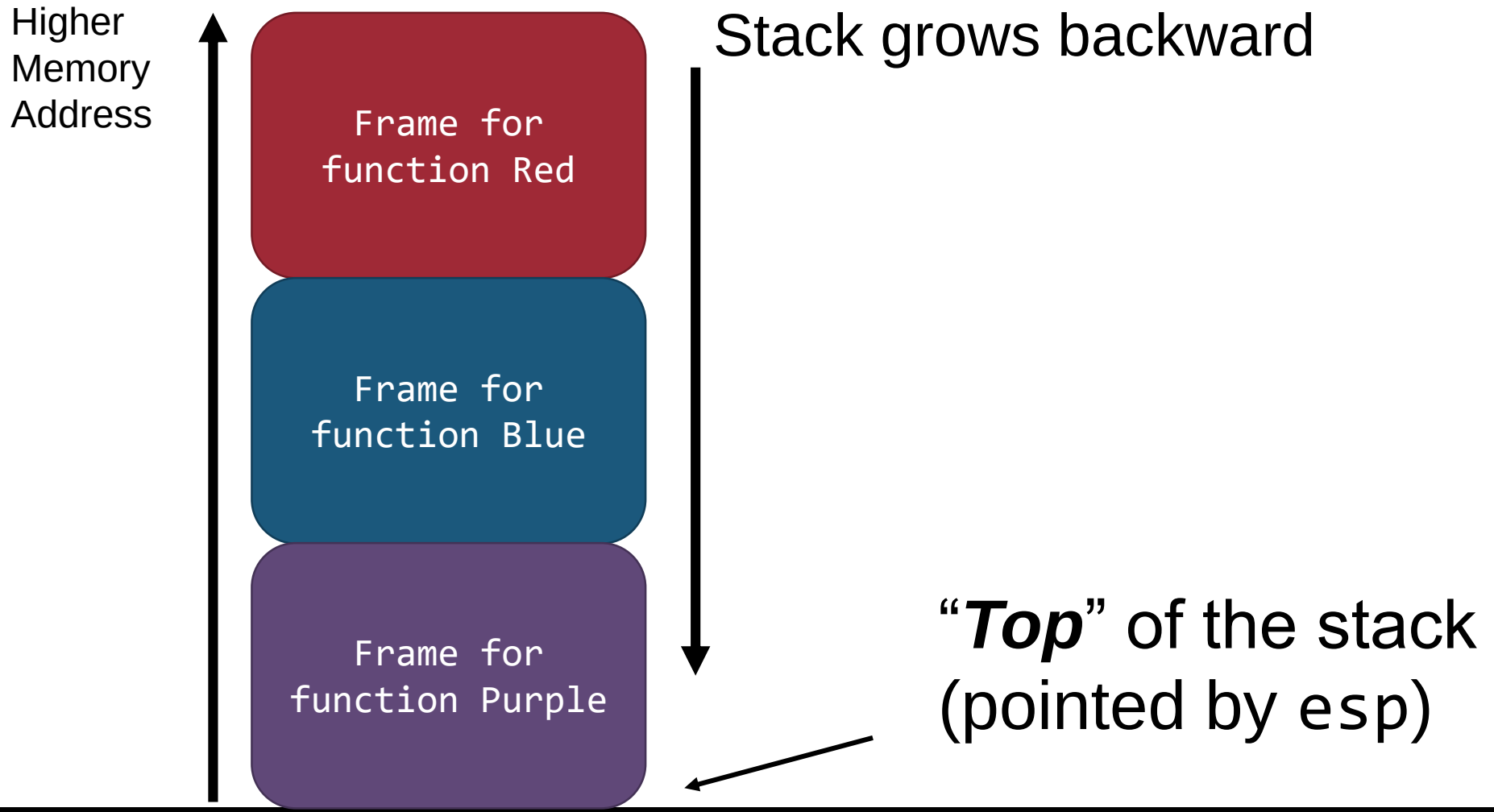
```
int Purple(int a1, int a2)
{
  int p = 2;
  return p + a1 – a2;
}
```

Q1. How to pass function parameters?

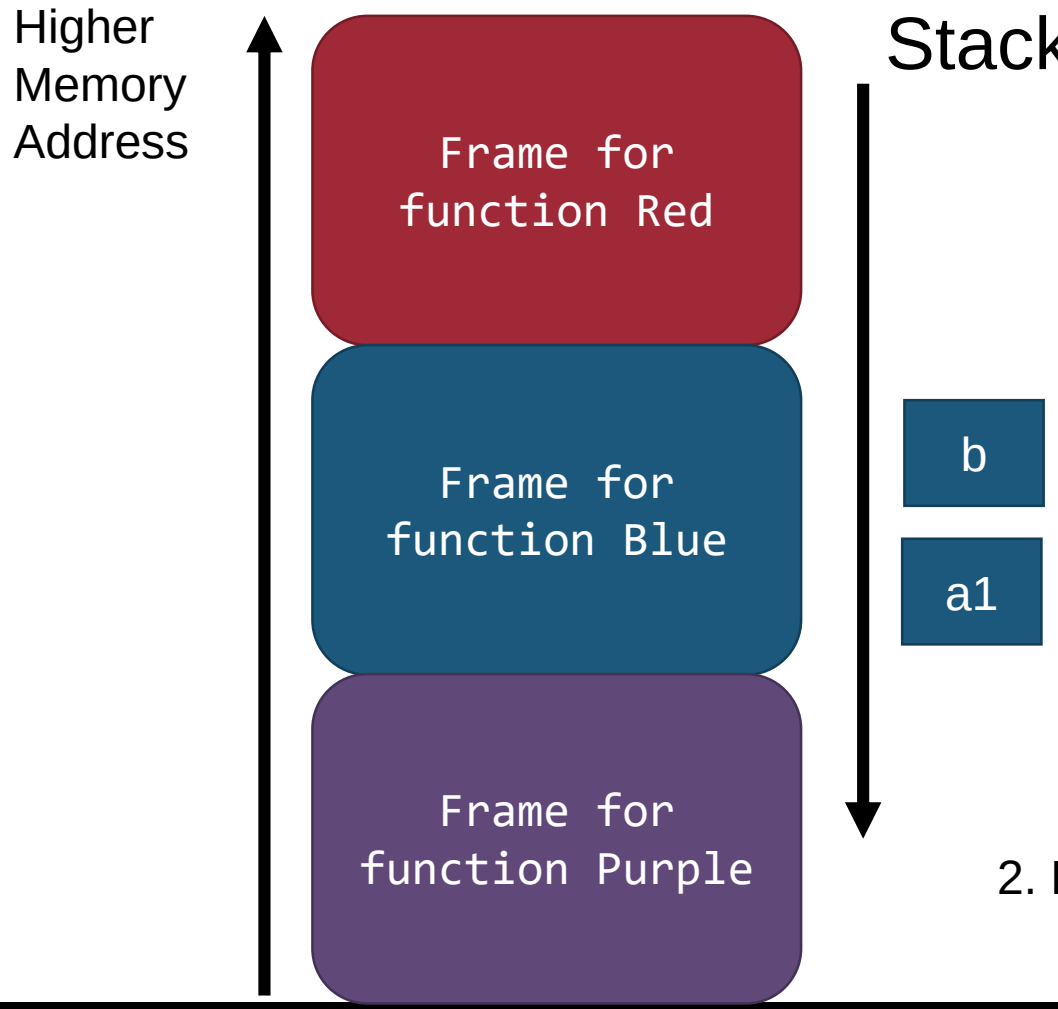
Q2. When a function returns, how to restore the register values of the caller function?

Q3. Where do we store local variables?

Stack


Higher
Memory
Address
Frame for
function Red
Frame for
function Blue
Frame for
function Purple
Stack grows backward
"Top" of the stack
(pointed by esp)

Calling Convention (cdecl)

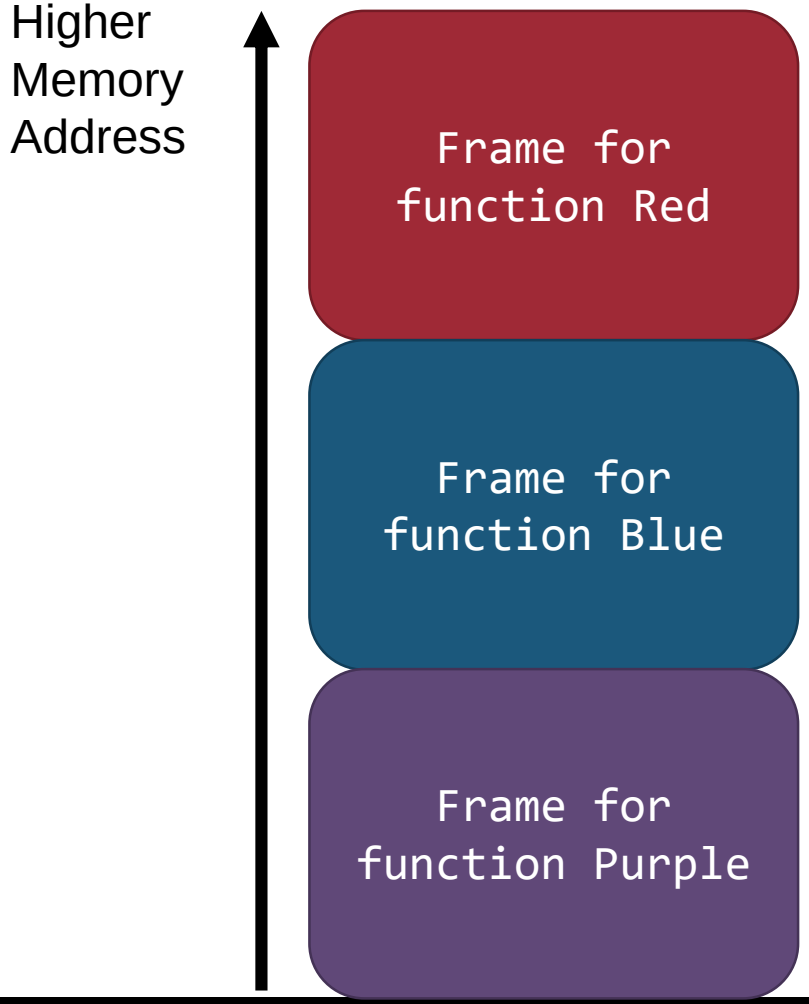



```
Purple(a1, b);
```

1. Push arguments in reverse order

2. Return value is stored in eax

Stack Frame



- Local variables for Blue
- Link to function Red
- Temporary space
- Function-call-related space
- Frame will be cleared when Blue returns

```c
int Red(int a1)
{
  int r = 0;
  return r + Blue(a1 - 42);
}
```

```c
int Blue(int a1)
{
  int b = 1;
  return b + Purple(a1, b);
}
```

```c
int Purple(int a1, int a2)
{
  int p = 2;
  return p + a1 – a2;
}
```

Compile

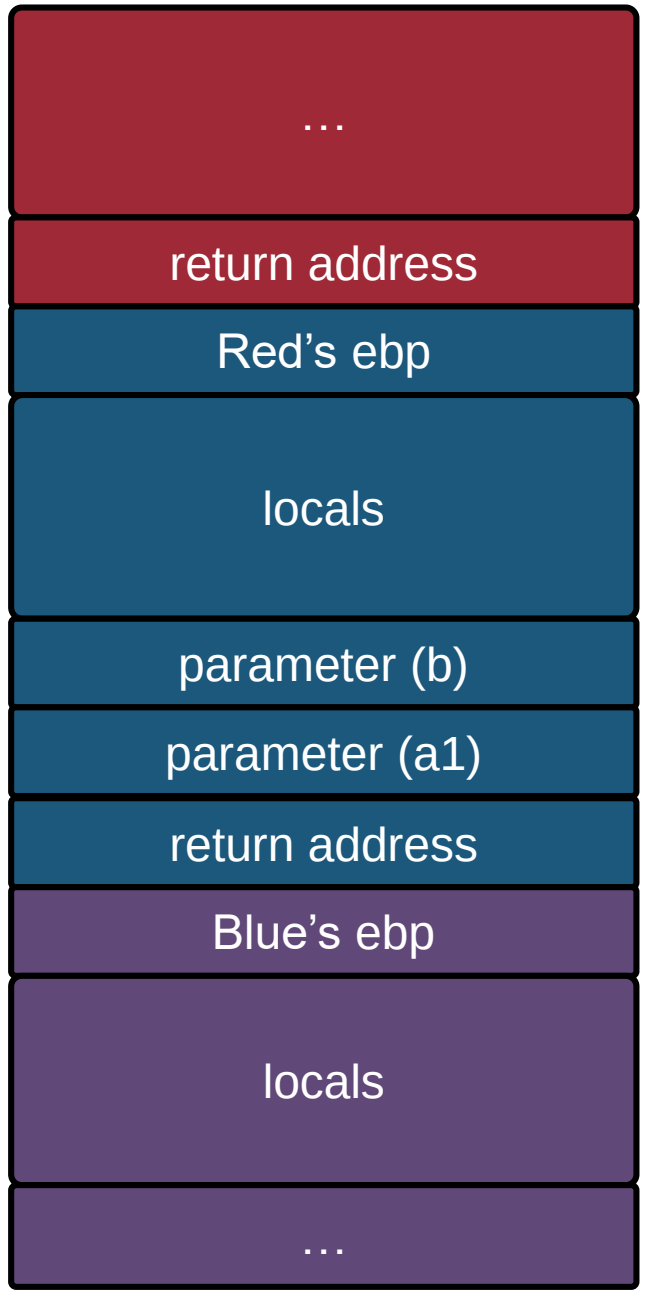
```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWORD PTR [esp+0x4],eax
  36:   mov    eax,DWORD PTR [ebp+0x8]
  39:   mov    DWORD PTR [esp],eax
  3c:   call   Purple
  41:   mov    edx,DWORD PTR [ebp-0xc]
  44:   add    eax,edx
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```

```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWORD PTR [esp+0x4],eax
  36:   mov    eax,DWORD PTR [ebp+0x8]
  39:   mov    DWORD PTR [esp],eax
  3c:   call   Purple
  41:   mov    edx,DWORD PTR [ebp-0xc]
  44:   add    eax,edx
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```

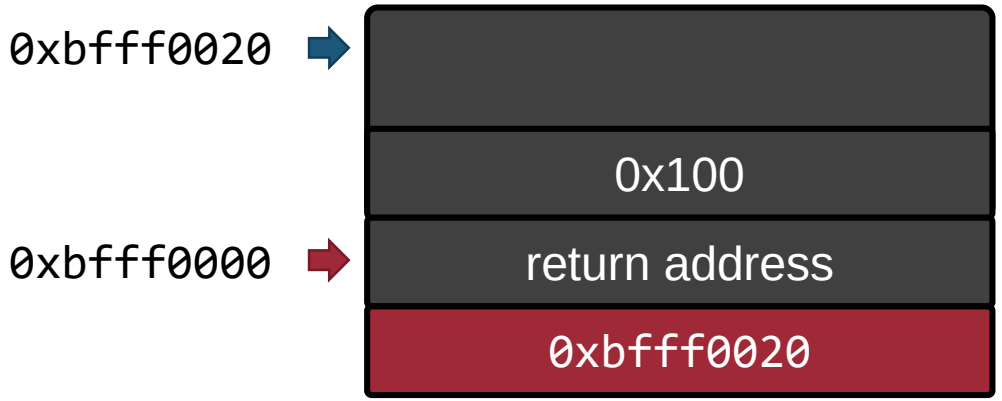
Address

Execution

Stack



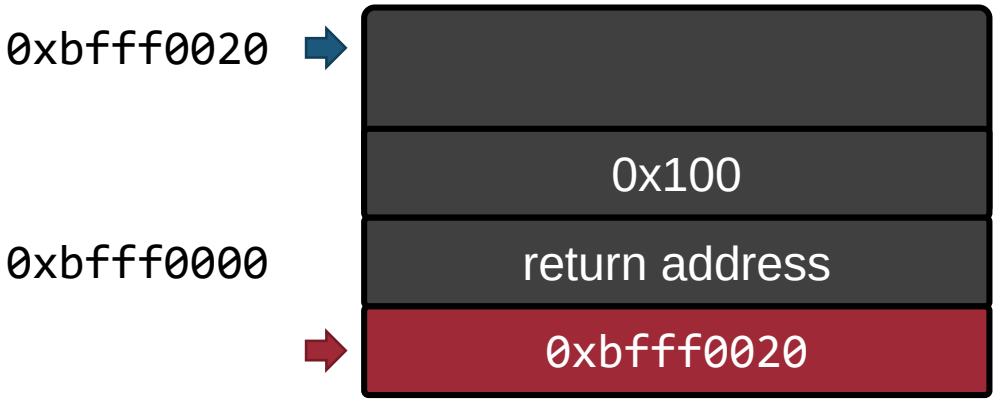
```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWORD PTR [esp+0x4],eax
  36:   mov    eax,DWORD PTR [ebp+0x8]
  39:   mov    DWORD PTR [esp],eax
  3c:   call   Purple
  41:   mov    edx,DWORD PTR [ebp-0xc]
  44:   add    eax,edx
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```



Execution Context

```
esp = 0xbfff0000
ebp = 0xbfff0020
eip = 0x0
```

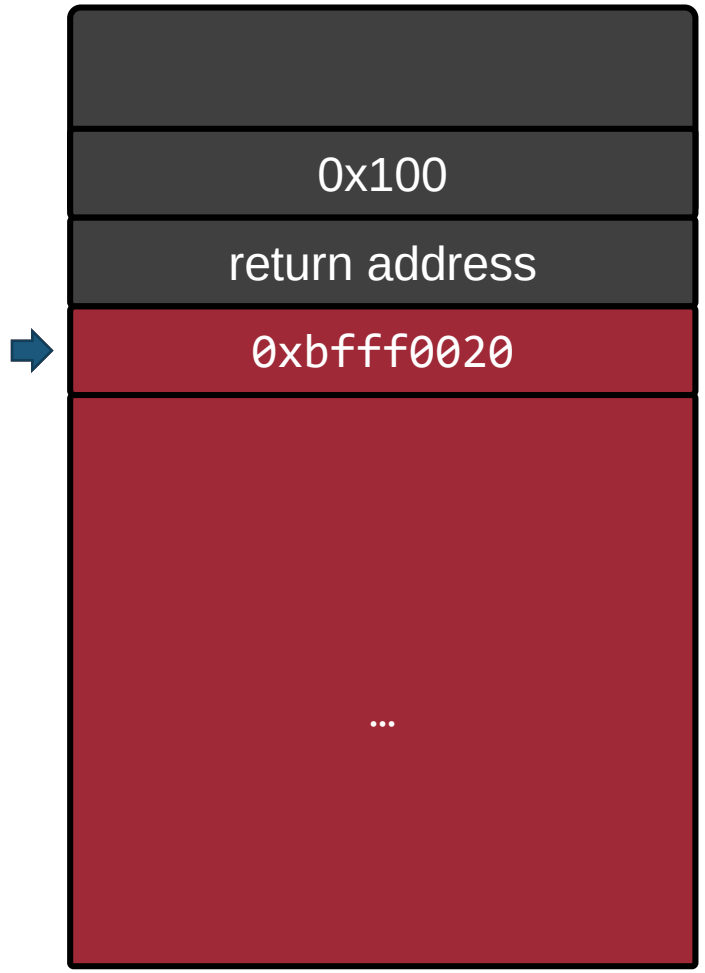
```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWORD PTR [esp+0x4],eax
  36:   mov    eax,DWORD PTR [ebp+0x8]
  39:   mov    DWORD PTR [esp],eax
  3c:   call   Purple
  41:   mov    edx,DWORD PTR [ebp-0xc]
  44:   add    eax,edx
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```



Execution Context

```
esp = 0xbffefffc
ebp = 0xbfff0020
eip = 0x1
```

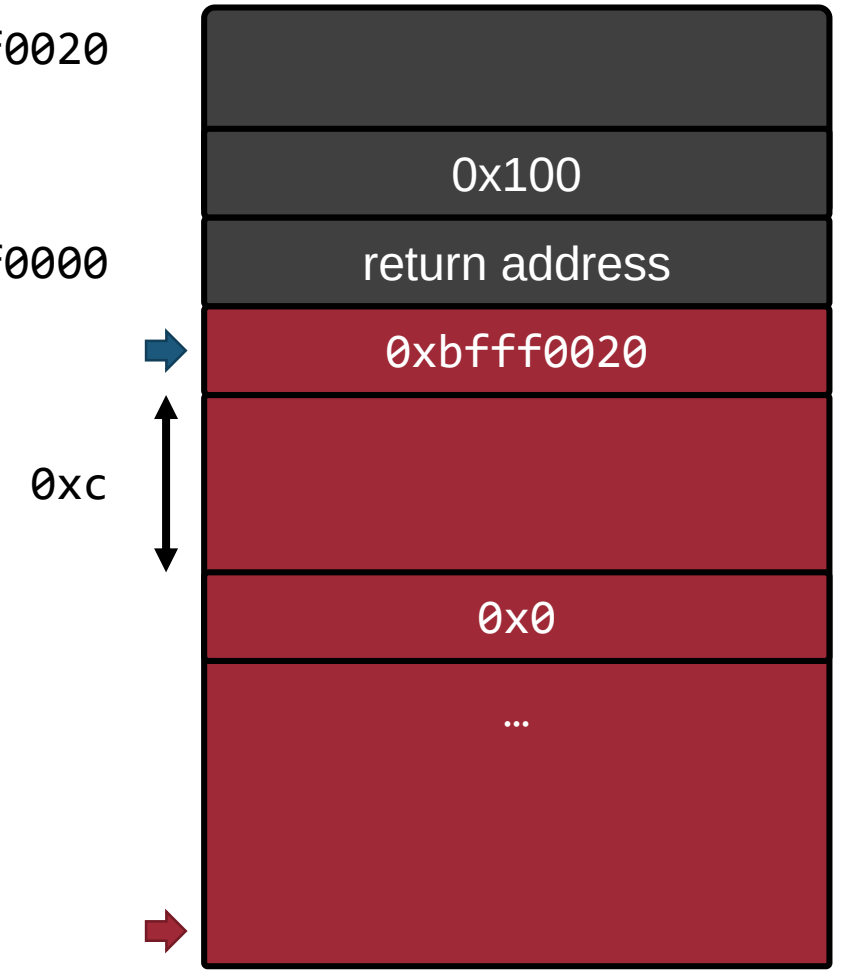
```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWORD PTR [esp+0x4],eax
  36:   mov    eax,DWORD PTR [ebp+0x8]
  39:   mov    DWORD PTR [esp],eax
  3c:   call   Purple
  41:   mov    edx,DWORD PTR [ebp-0xc]
  44:   add    eax,edx
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```



Execution Context

```
esp = 0xbffefffc
ebp = 0xbffefffc
eip = 0x3
```

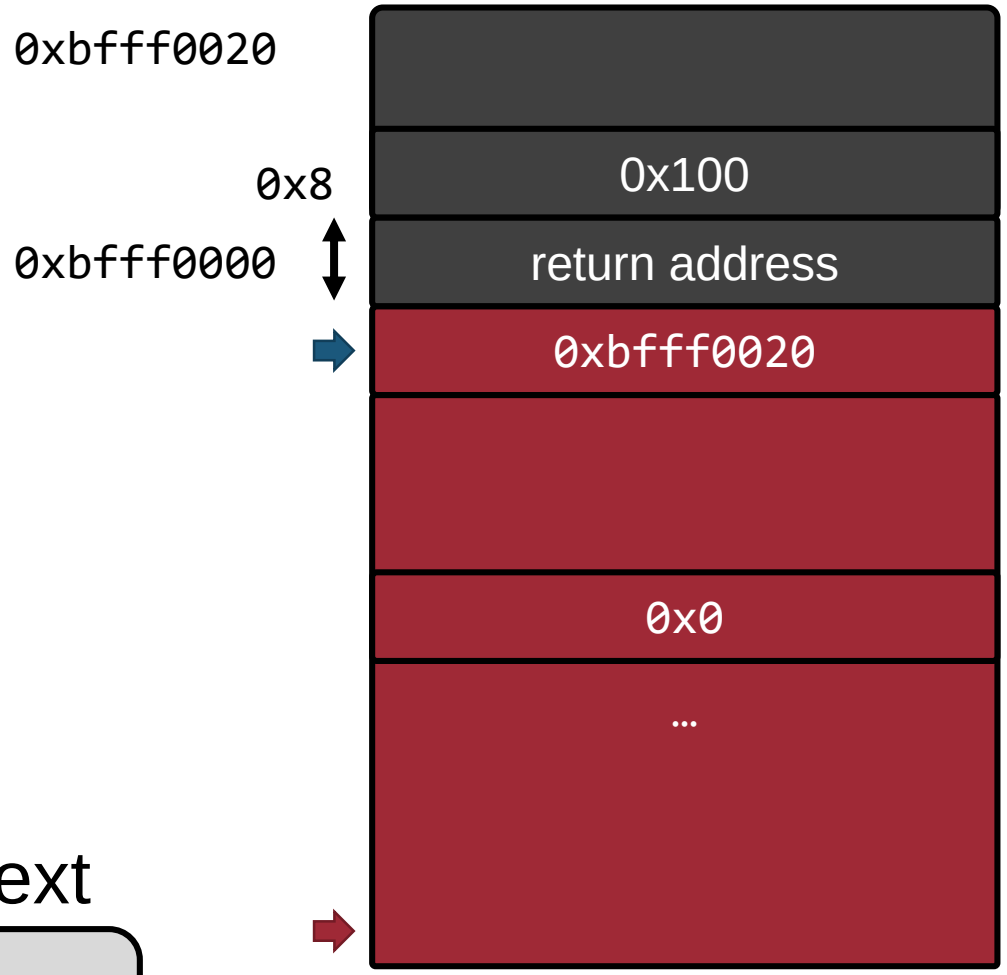
```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWORD PTR [esp+0x4],eax
  36:   mov    eax,DWORD PTR [ebp+0x8]
  39:   mov    DWORD PTR [esp],eax
  3c:   call   Purple
  41:   mov    edx,DWORD PTR [ebp-0xc]
  44:   add    eax,edx
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```



Execution Context

```
esp = 0xbffeffd4
ebp = 0xbffefffc
eip = 0x6
```

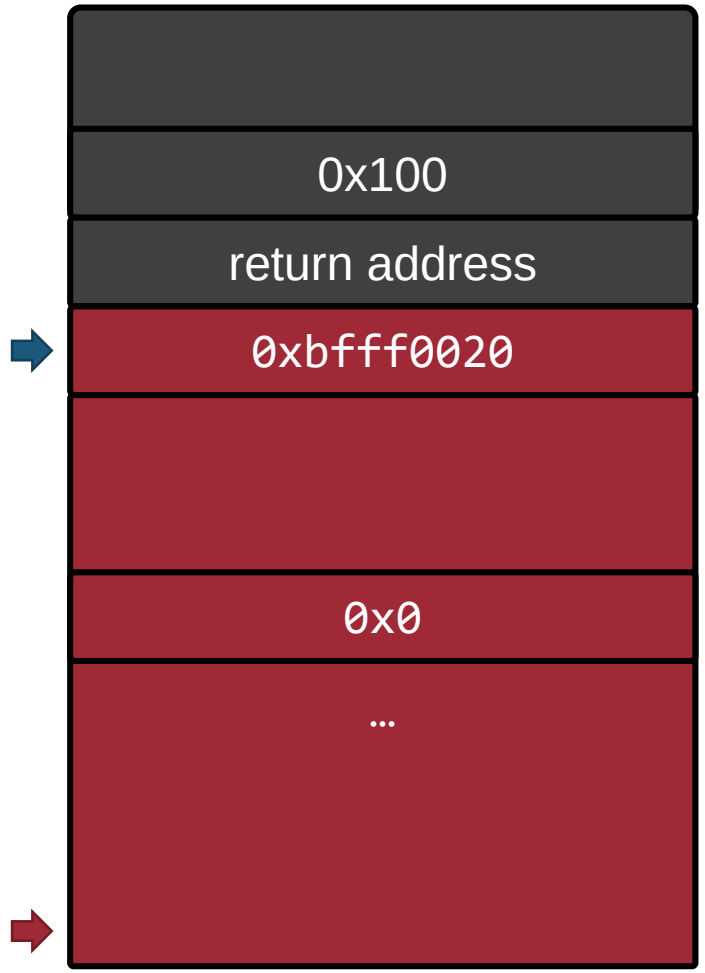
```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWORD PTR [esp+0x4],eax
  36:   mov    eax,DWORD PTR [ebp+0x8]
  39:   mov    DWORD PTR [esp],eax
  3c:   call   Purple
  41:   mov    edx,DWORD PTR [ebp-0xc]
  44:   add    eax,edx
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```



Execution Context

```
esp = 0xbffeffd4
ebp = 0xbffefffc
eip = 0xd
eax = 0x100
```

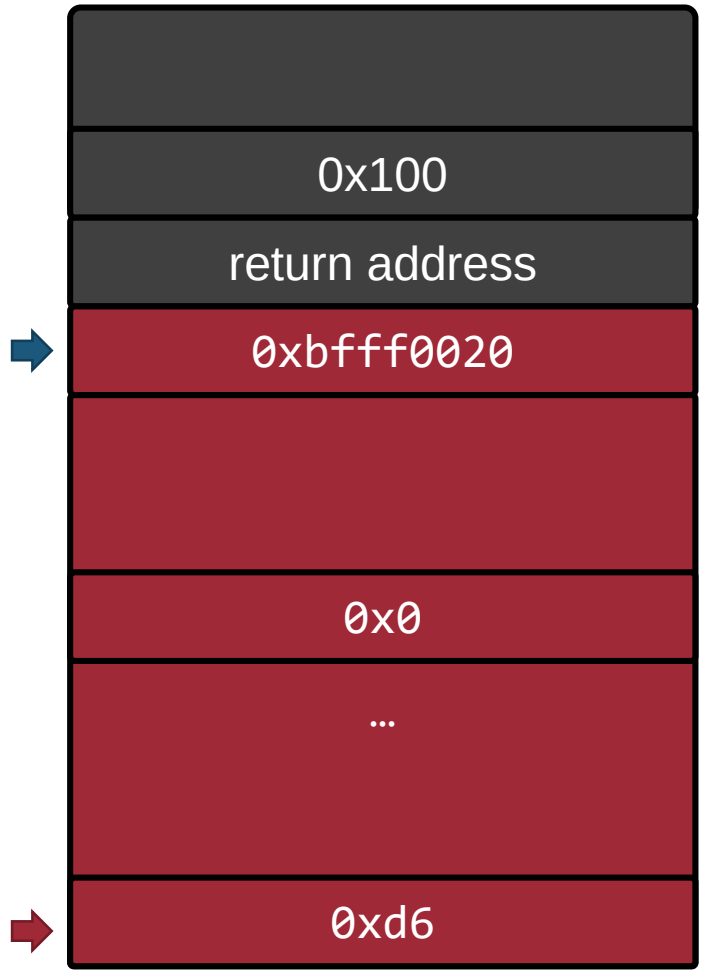
```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWORD PTR [esp+0x4],eax
  36:   mov    eax,DWORD PTR [ebp+0x8]
  39:   mov    DWORD PTR [esp],eax
  3c:   call   Purple
  41:   mov    edx,DWORD PTR [ebp-0xc]
  44:   add    eax,edx
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```



Execution Context

```
esp = 0xbffeffd4
ebp = 0xbffefffc
eip = 0x10
eax = 0x100
```

```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWORD PTR [esp+0x4],eax
  36:   mov    eax,DWORD PTR [ebp+0x8]
  39:   mov    DWORD PTR [esp],eax
  3c:   call   Purple
  41:   mov    edx,DWORD PTR [ebp-0xc]
  44:   add    eax,edx
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```

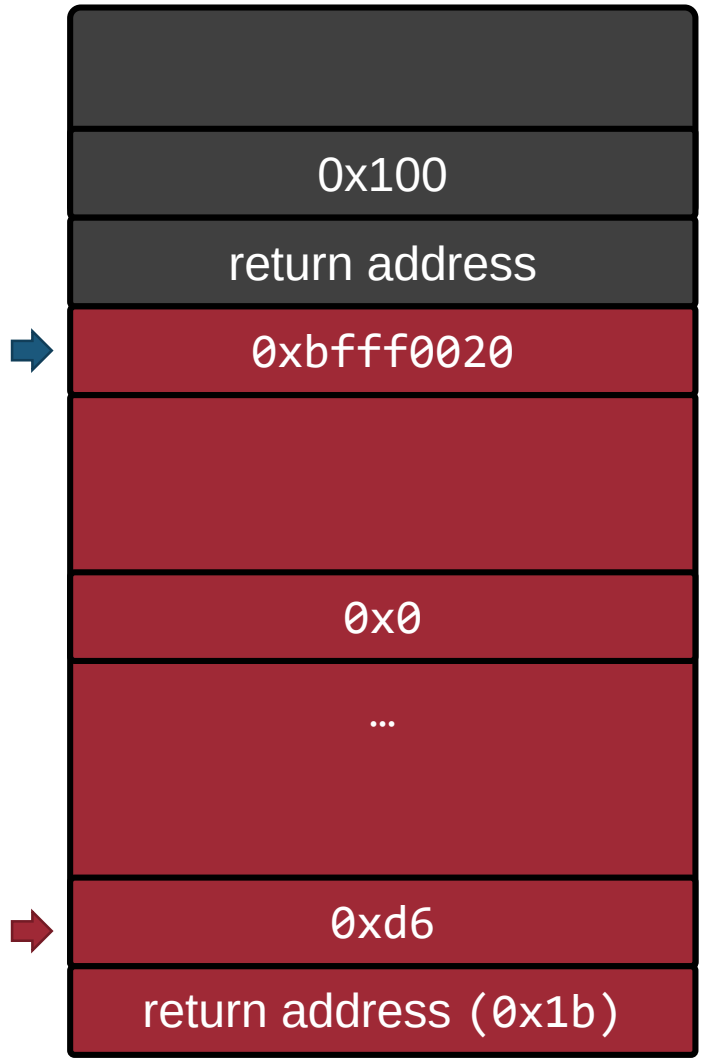


Execution Context

```
esp = 0xbffeffd4
ebp = 0xbffefffc
eip = 0x13
eax = 0xd6
```

```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWORD PTR [esp+0x4],eax
  36:   mov    eax,DWORD PTR [ebp+0x8]
  39:   mov    DWORD PTR [esp],eax
  3c:   call   Purple
  41:   mov    edx,DWORD PTR [ebp-0xc]
  44:   add    eax,edx
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```

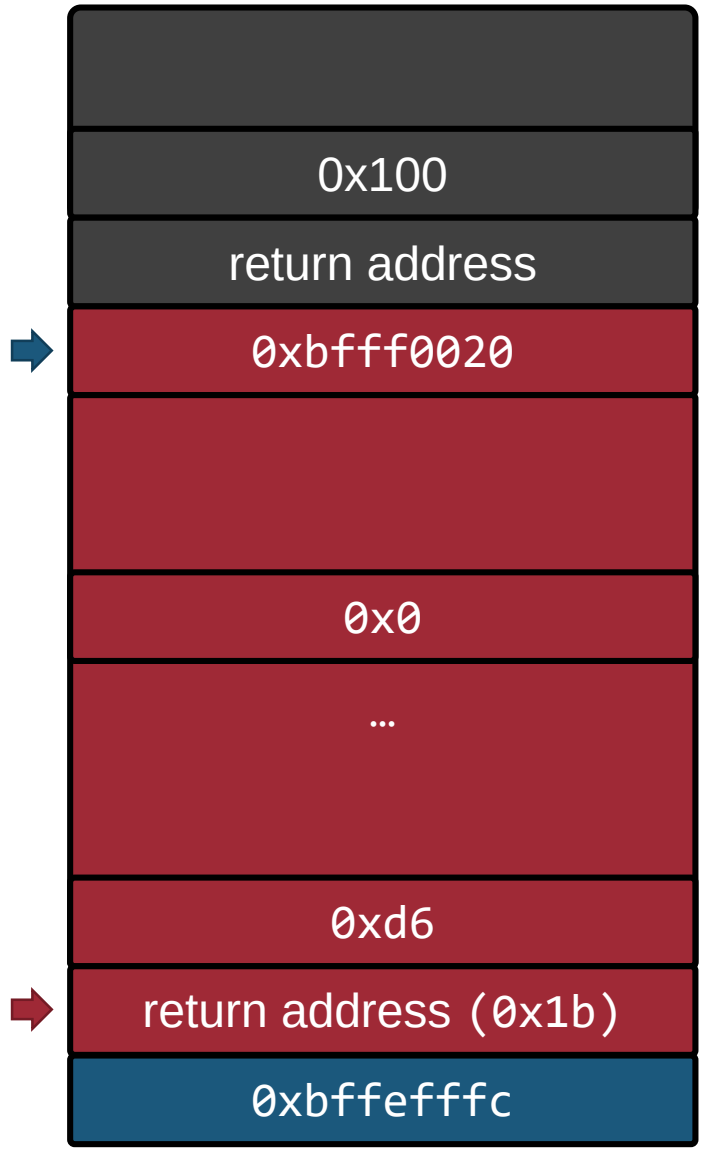
Return to here



Execution Context

```
esp = 0xbffeffd4
ebp = 0xbffefffc
eip = 0x16
eax = 0xd6
```

```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWORD PTR [esp+0x4],eax
  36:   mov    eax,DWORD PTR [ebp+0x8]
  39:   mov    DWORD PTR [esp],eax
  3c:   call   Purple
  41:   mov    edx,DWORD PTR [ebp-0xc]
  44:   add    eax,edx
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```



Execution Context

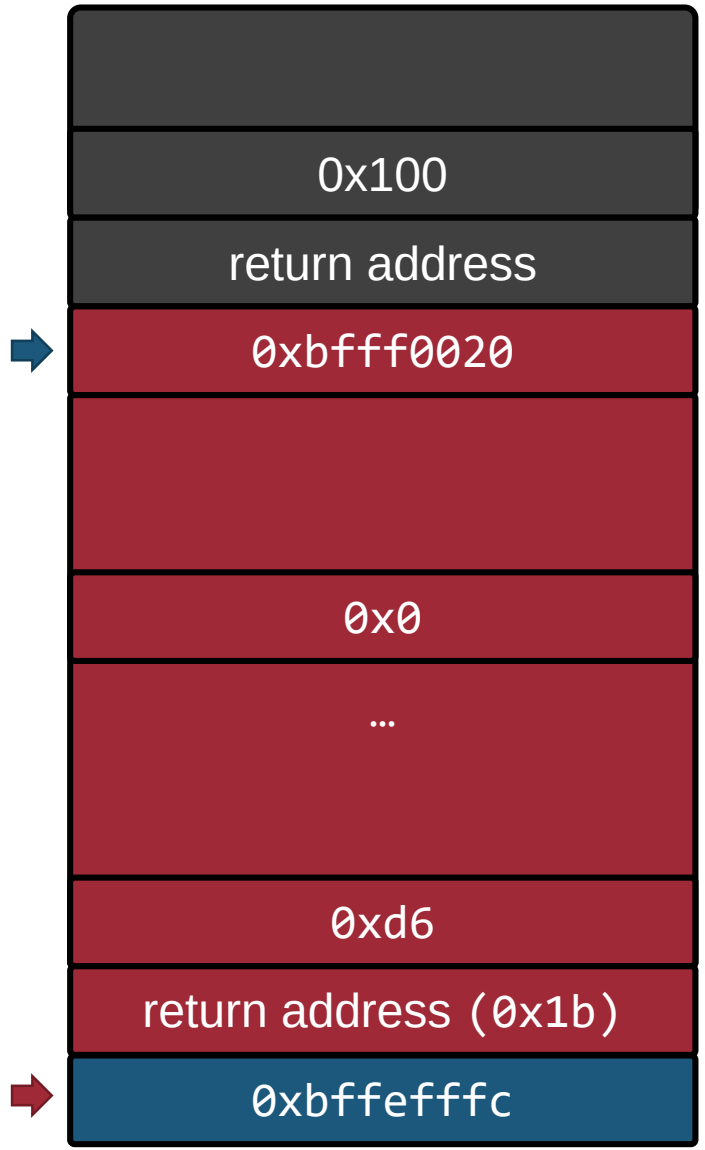
```
esp = 0xbffeffd8
ebp = 0xbffefffc
eip = 0x22
eax = 0xd6
```

```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp+0x8]
  32:   mov    DWO
  36:   mov    eax
  39:   mov    DWO
  3c:   call   Pur
  41:   mov    edx,DWORD
  44:   add    eax,ed
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```

Let's fast forward to here

Execution Context

```
esp = 0xbffeffd4
ebp = 0xbffefffc
eip = 0x23
eax = 0xd6
```

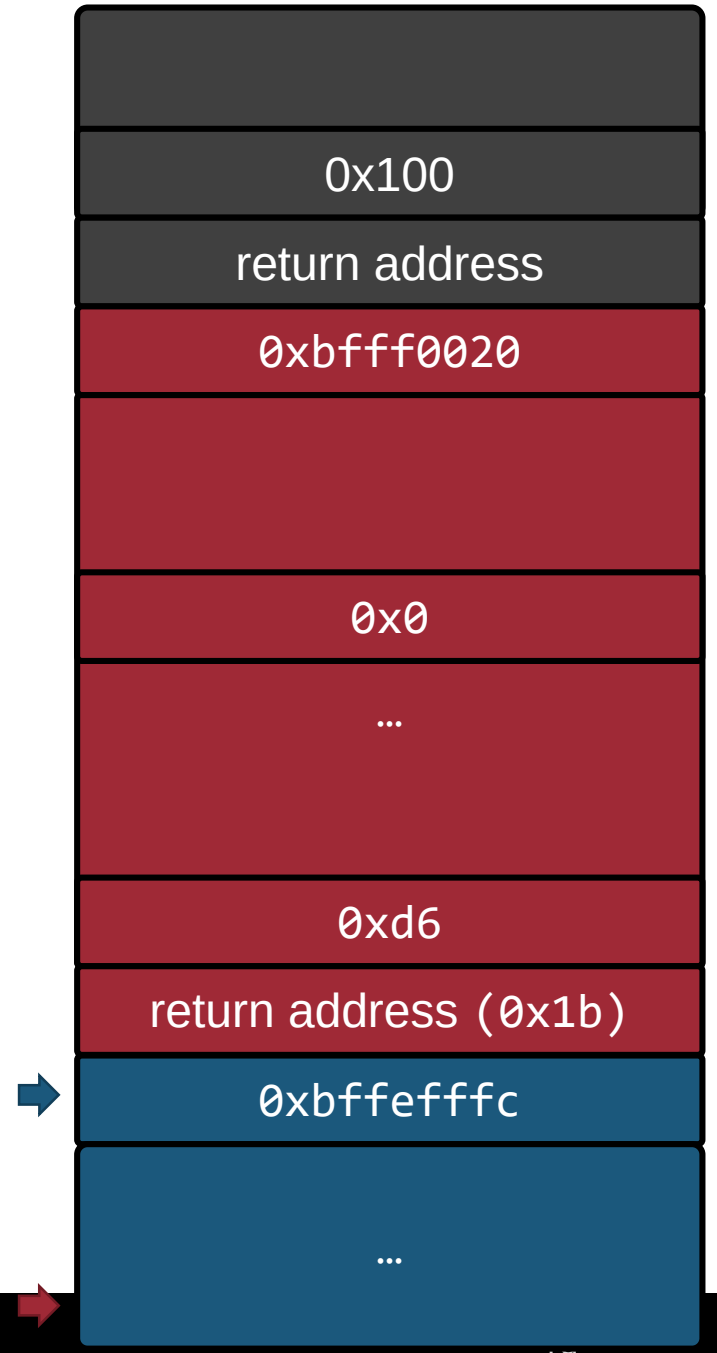


```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWO
  36:   mov    eax
  39:   mov    DWO
  3c:   call   Pur
  41:   mov    edx,DWORD
  44:   add    eax,ed
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```

mov esp, ebp
pop ebp

Execution Context

esp = 0xbffeffac
ebp = 0xbffeffd4
eip = 0x46



```
<Red>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x28
   6:   mov    DWORD PTR [ebp-0xc],0x0
   d:   mov    eax,DWORD PTR [ebp+0x8]
  10:   sub    eax,0x2a
  13:   mov    DWORD PTR [esp],eax
  16:   call   Blue
  1b:   mov    edx,DWORD PTR [ebp-0xc]
  1e:   add    eax,edx
  20:   leave
  21:   ret

<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWOR
  36:   mov    eax
  39:   mov    DWO
  3c:   call   Pur
  41:   mov    edx
  44:   add    eax,edx
  46:   leave
  47:   ret

<Purple>:
  48:   push   ebp
  49:   mov    ebp,esp
  4b:   sub    esp,0x10
  4e:   mov    DWORD PTR [ebp-0x4],0x2
  55:   mov    eax,DWORD PTR [ebp+0x8]
  58:   mov    edx,DWORD PTR [ebp-0x4]
  5b:   add    eax,edx
  5d:   sub    eax,DWORD PTR [ebp+0xc]
```

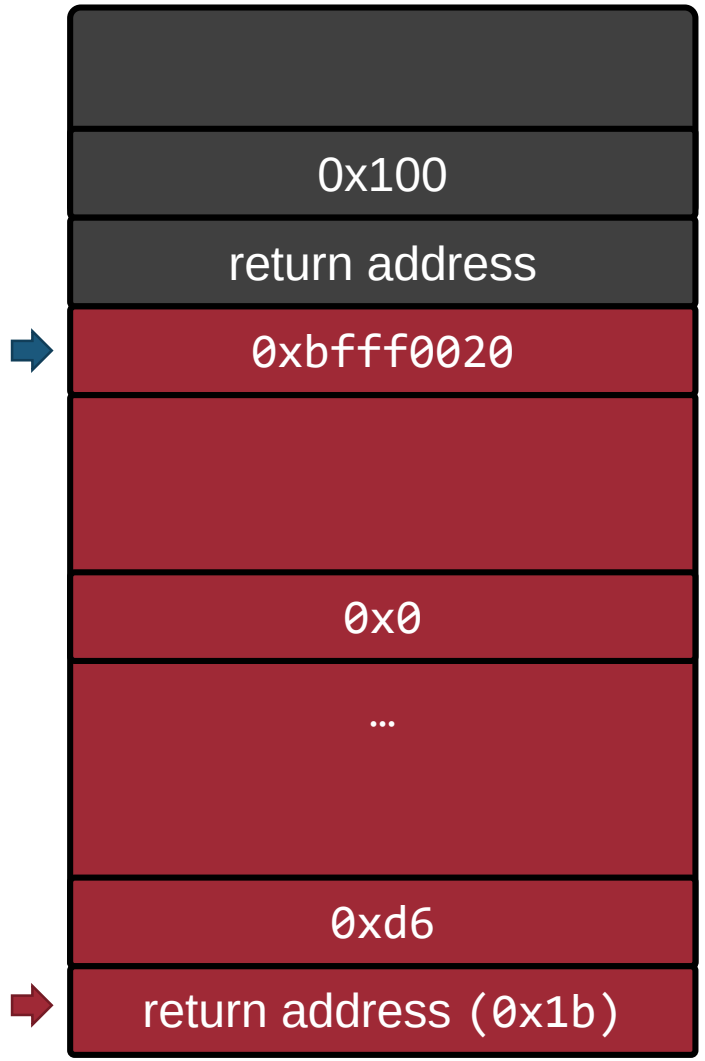
pop eip

Execution Context

esp = 0xbffeffd8

ebp = 0xbfff0004

eip = 0x47



Calling Convention

```
<Blue>:
  22:   push   ebp
  23:   mov    ebp,esp
  25:   sub    esp,0x28
  28:   mov    DWORD PTR [ebp-0xc],0x1
  2f:   mov    eax,DWORD PTR [ebp-0xc]
  32:   mov    DWORD PTR [esp+0x4],eax
  36:   mov    eax,DWORD PTR [ebp+0x8]
  39:   mov    DWORD PTR [esp],eax
  3c:   call   Purple
  41:   mov    edx,DWORD PTR [ebp-0xc]
  44:   add    eax,edx
  46:   leave
  47:   ret
```

```
int Blue(int a1)
{
  int b = 1;
  return b + Purple(a1, b);
}
```

b

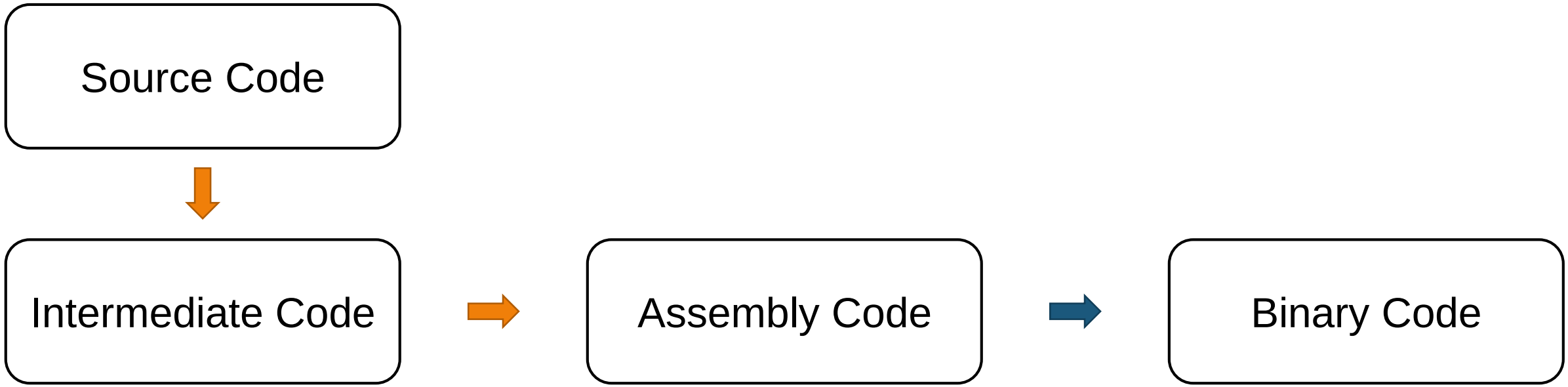
a1

Passing parameter values in a reverse order

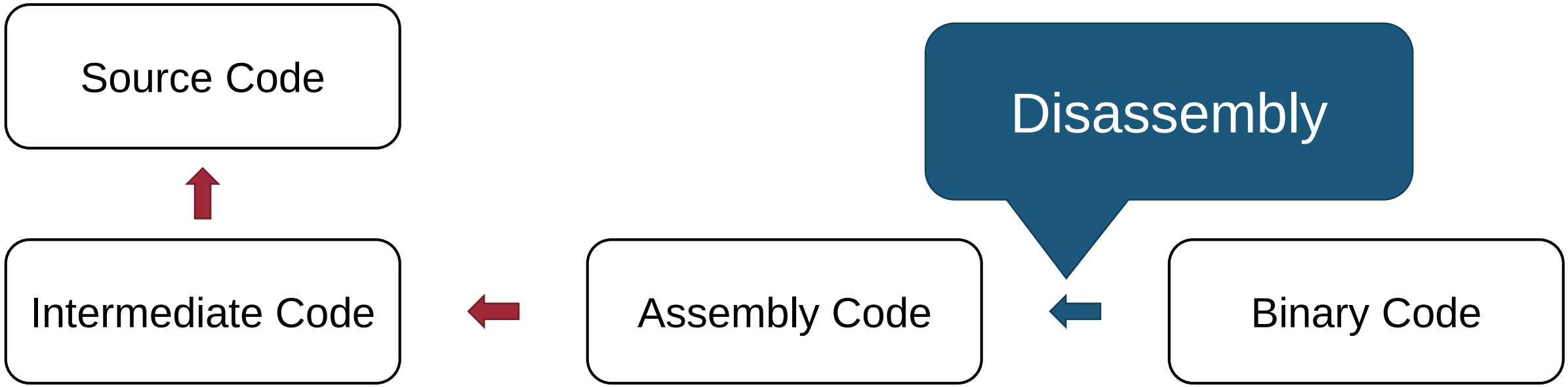
Storing a return value in eax

Recap

Compilation



Our Goal: Understanding Binary



GNU objdump

```
$ objdump –M intel –d a.out
```

Intel syntax

objdump Output

x86 uses variable-length encoding

```
00000000 <Red>:
 0:   55                      push   ebp
 1:   89 e5                   mov    ebp,esp
 3:   83 ec 28                sub    esp,0x28
 6:   c7 45 f4 00 00 00 00    mov    DWORD PTR [ebp-0xc],0x0
 d:   8b 45 08                mov    eax,DWORD PTR [ebp+0x8]
10:   83 e8 2a                sub    eax,0x2a
13:   89 04 24                mov    DWORD PTR [esp],eax
16:   e8 fc ff ff ff          call   17 <Red+0x17>
1b:   8b 55 f4                mov    edx,DWORD PTR [ebp-0xc]
1e:   01 d0                   add    eax,edx
20:   c9                      leave
21:   c3                      ret
```

Address | Binary Code | Disassembled Assembly Code

Use B2R2 instead of objdump

```
$ b2r2 dump <file name>
```

Question

Is perfect disassembly possible?

Key Concepts

- Compilation pipeline
- x86 architecture
- Assembly
- Disassembly

Debugging with GDB

GNU Debugger (GDB)

- Allows an analyst to inspect the program context at a certain point during execution

- Online manual: https://sourceware.org/gdb/onlinedocs/gdb/

First Configuration

- Use Intel syntax

- Add the following line to ~/.gdbinit file:
  `set disassembly-flavor intel`

Basics

- Repetition
  - Simply type a return key (without any command) will repeat the previous command
  - Some commands such as “**run**” will not repeat
- Executing shell commands
  - **shell** <cmds>
- Help
  - Always use ask for “help” ☺
  - **help** <command>

Running a Program under GDB

- Two ways to start
  - **start** <arguments>
  - **run** <arguments> (insert a temporary breakpoint to `main` function)

- Execution wrapper
  - **set exec-wrapper** <cmds>
  - Example: set exec-wrapper env -i <cmds>

Execution Environment

- Setting arguments
  - **set args** <args>
- Showing the current arguments
  - **show args**

Execution Environment (Cont'd)

- Setting an environment variable
  - **set environment <varname> = <value>**
- Unsetting an environment variable
  - **unset environment <varname>**
- Showing the current arguments
  - **show environment**

Working with I/O

- Similar to regular command line shells
- Output file redirection
  - **run** > output

- Input file redirection
  - **run** < input

Attaching an Already-Running Process

- When debugging an already-running process
  - **attach <pid>**

- Detach from the current process
  - **detach**

Multithreaded Applications

- Show threads
  - **info threads**
- Change the current thread
  - **thread <tid>**
- Apply commands to multiple threads
  - **thread apply all <cmds>**

Debugging Forked Process

- By default, GDB follows the parent process after forking
- Set GDB to follow the child process
  - **set follow-fork-mode child**

Breakpoints

- Setting a breakpoint:
  - **break** *<address>
  - **break** <location>
    - Only works with a debugging symbol
- Continue after a break
  - **continue**
  - **continue <ignore count>**
- Conditional breakpoints
  - **break <location> if <cond>**
  - `break foo if ((int)strcmp(x, "hello")) == 0`

H/W Breakpoints

Use `hbreak` instead of `break`

Watchpoints

- Break when a memory value is modified
  - **watch *<addr>**
  - **watch <location>**

- Break when a memory value is ***accessed***
  - **rwatch *<addr>**
  - **rwatch <location>**

- Break when a memory value is either modified or accessed
  - **awatch *<addr>**
  - **awatch <location>**

Catchpoints

- Break at program events such as exceptions, syscall invocations, etc.
  - catch throw
  - catch exception
  - catch syscall
  - catch syscall <syscall name or number>
  - catch fork
  - catch signal

Breakpoints, Watchpoints, and Catchpoints

- Show information about breakpoints/watchpoints/catchpoints
  - **info breakpoints**
- Disabling
  - **disable <id>**
- Enabling
  - **enable <id>**
- Deleting
  - **delete <id>**

Stepping Source Lines

- Continue until the next source line
  - **step**
  - **step <num lines>**
- Continue until the next source line (do not follow function calls)
  - **next**
  - **next <num lines>**
- Continue until the current stack frame returns
  - **finish**
- Continue until the program reaches a source line greater than the current
  - **until**

Stepping Binary Instructions

- Continue until the next instruction
  - **stepi**
  - **stepi <num instructions>**

- Continue until the next instruction (do not follow call instruction)
  - **nexti**
  - **nexti <num instructions>**

Skipping

- When stepping, skip a specified function (or a line, or a file)
  - **skip <line>**
  - **skip <function name>**
  - **skip file <filename>**

- Show skipping information
  - **info skip**

- Delete skipping information
  - **skip delete <id>**

Thread-Specific Breakpoints

- **break <location> thread <thread id>**
- **break <location> thread <thread id> if <cond>**

Examining Stack

- Stack backtrace
  - **backtrace**
  - **backtrace <number of frames>**
- Select a stack frame
  - **frame <num>**
- Move a stack up/down
  - **up**
  - **down**
- Print current stack arguments and local variables
  - **info args**
  - **info locals**

Examining Data (with Source)

- Printing a data
  - print <expr>
  - print/<format> <expr>

- Expressions?
  - Global variables or local variables of the current frame
  - Arrays
    print *array@len
    (@ sign is useful when the array variable is just a simple pointer)

Array Example

```
int main(){
  int *a;
  int b[3] = {1,2,3};
  a = b;
  return 0;
}
```

Printing Formats

print/<**format**> <expr>

Format	Meaning
x	Hexadecimal integer
d	Decimal integer
u	Unsigned decimal integer
o	Octal integer
c	Character
f	Floating point
s	C-String
t	Binary

Examining Data without Source

- Examine a given memory address
  - **x/nfu <addr>**

- Optional parameters n, f, u
  - n = the repeat count (default 1)
  - f = the display format (default ‘x’)
    - Format is the same as in the print command, but there is one extra format ‘i’
    - i = print a machine instruction
  - u = the unit size (default ‘w’)
    - b = bytes
    - h = half words (2 bytes)
    - w = words (4 bytes)
    - g = giant words (8 bytes)

Examining Registers

- Show register values
  - **info register**
  - **info register <regname>**
- Print a register value
  - **print/<format> $regname**
- Print the instruction to be executed
  - **x/i $eip**
  - **x/10i $eip**

TUI

- Start TUI
  - **tui reg general**
- Change layout
  - **layout asm**
  - **layout reg**

Questions?