Practical Foundations of Debugging

Chapter 6

Frame Pointer and Local Variables – Part 2

Designed, written and illustrated by Dmitry Vostokov

```
LocalVariables!main:
push    ebp                         ; establishing stack frame
mov     ebp,esp                     ;
sub     esp,0xd8                    ; creating stack frame for locals
push    ebx                         ; saving registers that might be used
push    esi                         ;   outside
push    edi                         ;
lea     edi,[ebp-0xd8]              ; getting lowest address of stack frame
mov     ecx,0x36                    ; filling stack frame with 0xCC
mov     eax,0xcccccccc              ;
rep     stosd                       ;
mov     dword ptr [ebp-0x8],0x1     ; [a] = 1       ([ebp-0x8])
mov     dword ptr [ebp-0x14],0x1    ; [b] = 1       ([ebp-0x14])
mov     eax,[ebp-0x14]              ; eax := [b]
add     eax,[ebp-0x8]               ; eax := eax + [a]
mov     [ebp-0x14],eax              ; [b] := eax    (b = b + a)
mov     eax,[ebp-0x8]               ; eax := [a]
add     eax,0x1                     ; eax := eax + 1
mov     [ebp-0x8],eax               ; [a] := eax    (++a)
mov     eax,[ebp-0x8]               ; eax := [a]
imul    eax,[ebp-0x14]              ; eax := eax * [b]
mov     [ebp-0x14],eax              ; [b] := eax    (b = a * b)
xor     eax,eax                     ; eax := 0      (return value)
pop     edi                         ; restoring registers
pop     esi                         ;
pop     ebx                         ;
mov     esp,ebp                     ; restoring previous stack pointer
pop     ebp                         ; restoring previous stack frame
ret                                 ; return 0
```

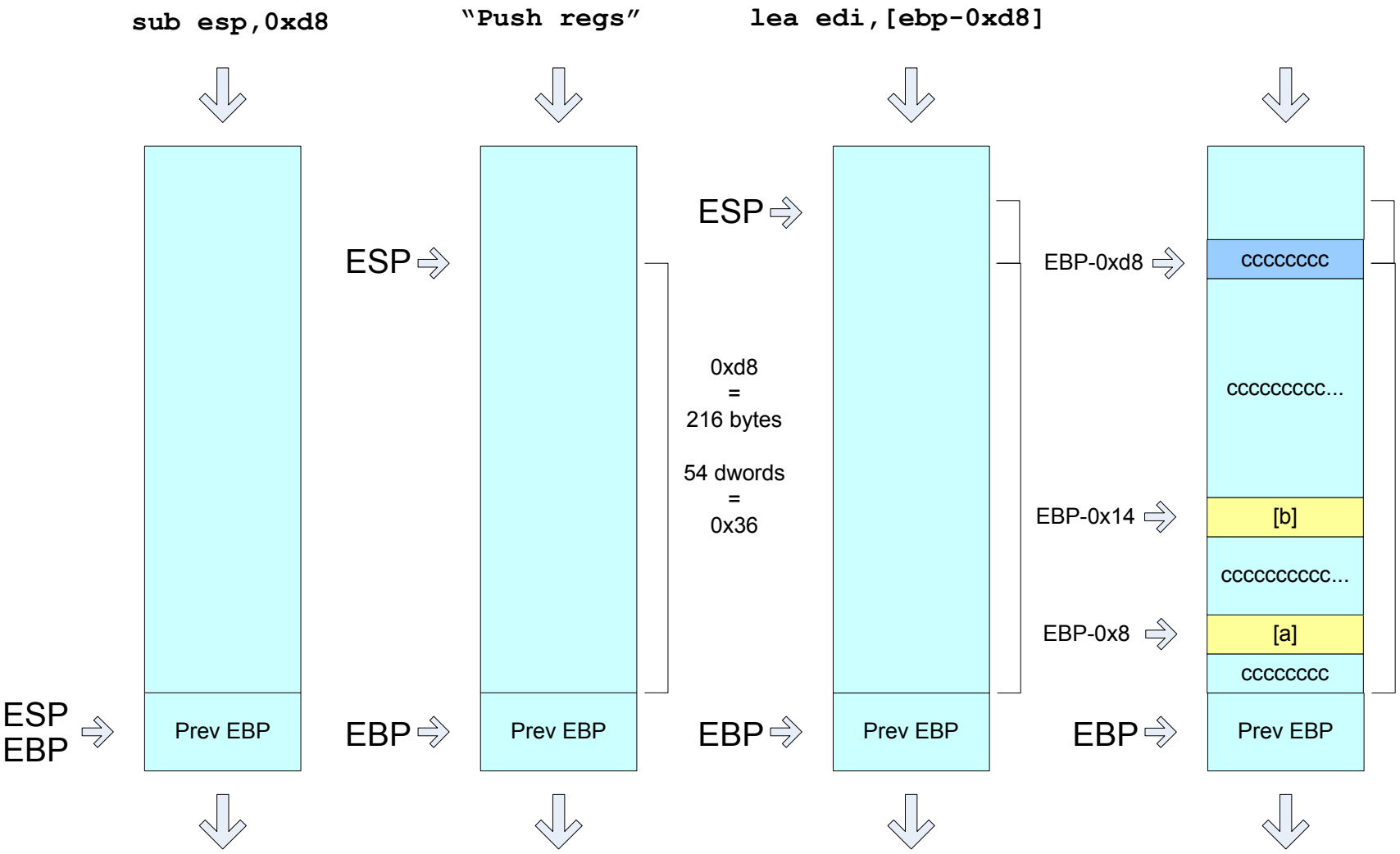
Comments

- The function that called main may use EDI register (group 1) so it is saved
- Other group 1 registers are saved because without optimization compiler emits standard code (saves all group 1 registers, standard prolog/epilog, etc.)

```
int mainCRTStartup (/*…*/)
{
   // may use group 1 registers (EDI, …)
   main(/*…*/);
   // may continue use group 1 registers (EDI, …)
}

int main(/*…*/)
{
   // saves group 1 registers if it uses them (EDI, …)
   // uses group 1 registers
   // restores group 1 registers (if saved)
}
```

```
sub     esp,0xd8
push    ebx
push    esi
push    edi
lea     edi,[ebp-0xd8]
mov     ecx,0x36
mov     eax,0xcccccccc
rep     stosd
```



Disassembly for optimized executable (Release configuration)

```
LocalVariables!main:
00401000 33c0        xor     eax,eax
00401002 c3          ret

int main(int argc, char* argv[])
{
   return 0;
}
```

Where is all the code?

How to preserve the code under optimization?

Modifiers in C and C++

- static

static int a, b;

- const

const int a = 1, b = 1;

- volatile

volatile int a, b;

static const volatile a = 1, b = 1;

Using volatile modifier

```
int main(int argc, char* argv[])
{
   volatile int a, b;

   a = 1;
   b = 1;

   b = b + a;
   ++a;
   b = a * b;

   return 0;
}
```

FPO

- Frame Pointer omission

Optimization where ESP is used to address local variables and parameters and EBP is used as a general purpose register (like EAX, ECX, etc.)

```
LocalVariables!main:
sub     esp,0x8             ; allocating stack space for locals
mov     eax,0x1             ; eax := 1
mov     [esp],eax           ; [a] := eax  ([esp])
mov     [esp+0x4],eax       ; [b] := eax  ([esp+0x4])
mov     eax,[esp+0x4]       ; eax := [b]
add     eax,[esp]           ; eax := eax + [a]
mov     [esp+0x4],eax       ; [b] := eax  (b = b + a)
mov     ecx,[esp]           ; ecx := [a]
inc     ecx                 ; ecx += 1
mov     [esp],ecx           ; [a] := ecx  (++a)
mov     edx,[esp+0x4]       ; edx := [b]
mov     eax,[esp]           ; eax := [a]
imul    edx,eax             ; edx := edx * eax
mov     [esp+0x4],edx       ; [b] := edx  (b = b * a)
xor     eax,eax             ; eax := 0
add     esp,0x8             ; clearing stack space for locals
ret                         ; return 0
```

What's next?

- Passing parameters to functions
- Accessing function parameters
- "Arithmetical" project with function parameters