Lab 3. The Art of Assembly Language (II)

Dan Bruce, David Clark and Héctor D. Menéndez

Department of Computer Science
University College London

October 23, 2017

Modified from Xeno Kovah slides of Open Security Training

Introduction

There are two forms to modify control flow:

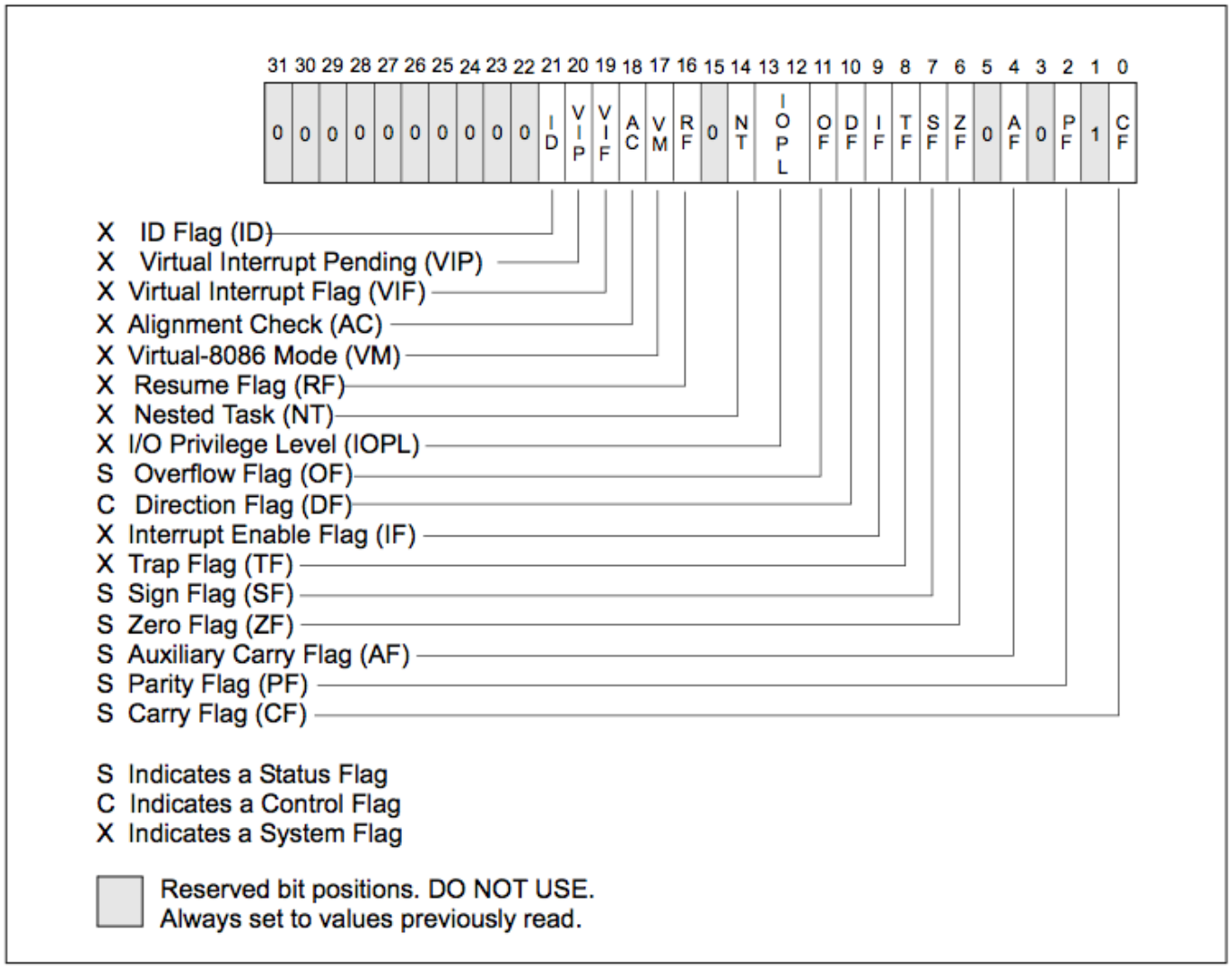
- **Conditional**: go somewhere if a condition is met. For example: “if”s, switches, loops.
- **Unconditional**: go somewhere directly. For example: Procedure calls, goto, exceptions, interrupts.

JMP - Jump

Change **EIP** to the given address (like goto). The main forms of the address:

- **Relative**: 1 byte displacement from end of the instruction (`jmp 0x0E`).
- **Absolute**: hardcoded address in instruction (`jmp 0x02348291`).
- **Absolute Indirect**: address calculated with r/m32 (`jmp eax + 0x0E`).

Flags


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 ID VIP VIF AC VM RF 0 NT IOPL OF DF IF TF SF ZF 0 AF 0 PF 1 CF
X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)
S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag
Reserved bit positions. DO NOT USE.
Always set to values previously read.

Jcc - Jump If Condition Is Met

There are several options for conditional jumps.

We will focused on those that corresponds with: ZF, CF, OF and SF.

Some Jcc Instructions

JZ/JE (Zero/Equal): if ZF == 1.

JNZ/JNE (No Zero/Equal): if ZF == 0.

JLE/JNG (Less or Equal - Signed): if ZF == 1 or SF != OF.

JGE/JNL (Greater or Equal -Signed): if SF == OF.

JBE (Below or Equal - Unsigned): if CF == 1 OR ZF == 1.

JAE (Above or Equal - Unsigned): if CF == 0 OR ZF == 1.

In the debugger you can just look at eflags and/or watch whether it takes a jump.

Example

We update the frame register with the stack register

OF	SF	ZF	PF	CF
0	0	0	0	0

```
something:
00401000   mov cx,3
00401001   sub ax,cx
00401003   jz main
00401008   jmp end
main:
00401010   mov ax,2
00401011   mov bx,3
00401013   sub ax,bx
00401018   add ax,ax
0040101D   jnz something
end:
0040101E   ret
```

EIP	0x00401010
AX	undefined
BX	undefined
CX	undefined

Example

We update the frame register with the stack register

OF	SF	ZF	PF	CF
0	0	0	0	0

```
something:
00401000   mov cx,3
00401001   sub ax,cx
00401003   jz main
00401008   jmp end
main:
00401010   mov ax,2
00401011   mov bx,3
00401013   sub ax,bx
00401018   add ax,ax
0040101D   jnz something
end:
0040101E   ret
```

EIP	0x00401011
AX	2
BX	undefined
CX	undefined

Example

We update the frame register with the stack register

OF	SF	ZF	PF	CF
0	0	0	0	0

```
something:
00401000   mov cx,3
00401001   sub ax,cx
00401003   jz main
00401008   jmp end
main:
00401010   mov ax,2
00401011   mov bx,3
00401013   sub ax,bx
00401018   add ax,ax
0040101D   jnz something
end:
0040101E   ret
```

EIP	0x00401013
AX	2
BX	3
CX	undefined

Example

We update the frame register with the stack register

OF	SF	ZF	PF	CF
0	1	0	1	0

```
something:
00401000   mov cx,3
00401001   sub ax,cx
00401003   jz main
00401008   jmp end
main:
00401010   mov ax,2
00401011   mov bx,3
00401013   sub ax,bx
00401018   add ax,ax
0040101D   jnz something
end:
0040101E   ret
```

EIP	0x00401018
AX	-1
BX	3
CX	undefined

Example

We update the frame register with the stack register

OF	SF	ZF	PF	CF
0	1	0	0	0

```
something:
00401000   mov cx,3
00401001   sub ax,cx
00401003   jz main
00401008   jmp end
main:
00401010   mov ax,2
00401011   mov bx,3
00401013   sub ax,bx
00401018   add ax,ax
0040101D   jnz something
end:
0040101E   ret
```

EIP	0x0040101D
AX	-2
BX	3
CX	undefined

Example

We update the frame register with the stack register

OF	SF	ZF	PF	CF
0	0	0	0	0

```
something:
00401000   mov cx,3
00401001   sub ax,cx
00401003   jz main
00401008   jmp end
main:
00401010   mov ax,2
00401011   mov bx,3
00401013   sub ax,bx
00401018   add ax,ax
0040101D   jnz something
end:
0040101E   ret
```

EIP	0x00401000
AX	-2
BX	3
CX	undefined

Example

We update the frame register with the stack register

OF	SF	ZF	PF	CF
0	0	0	0	0

```
something:
00401000   mov cx,3
00401001   sub ax,cx
00401003   jz main
00401008   jmp end
main:
00401010   mov ax,2
00401011   mov bx,3
00401013   sub ax,bx
00401018   add ax,ax
0040101D   jnz something
end:
0040101E   ret
```

EIP	0x00401001
AX	-2
BX	3
CX	3

Example

We update the frame register with the stack register

OF	SF	ZF	PF	CF
0	1	0	1	0

```
something:
00401000   mov cx,3
00401001   sub ax,cx
00401003   jz main
00401008   jmp end
main:
00401010   mov ax,2
00401011   mov bx,3
00401013   sub ax,bx
00401018   add ax,ax
0040101D   jnz something
end:
0040101E   ret
```

EIP	0x00401003
AX	-5
BX	3
CX	3

Example

We update the frame register with the stack register

OF	SF	ZF	PF	CF
0	0	0	0	0

```
something:
00401000   mov  cx,3
00401001   sub  ax,cx
00401003   jz main
00401008   jmp  end
main:
00401010   mov  ax,2
00401011   mov  bx,3
00401013   sub  ax,bx
00401018   add  ax,ax
0040101D   jnz  something
end:
0040101E   ret
```

EIP	0x00401008
AX	-5
BX	3
CX	3

Example

We update the frame register with the stack register

OF	SF	ZF	PF	CF
0	0	0	0	0

```
something:
00401000   mov cx,3
00401001   sub ax,cx
00401003   jz main
00401008   jmp end
main:
00401010   mov ax,2
00401011   mov bx,3
00401013   sub ax,bx
00401018   add ax,ax
0040101D   jnz something
end:
0040101E   ret
```

EIP	0x0040101E
AX	-5
BX	3
CX	3

Example

We update the frame register with the stack register

OF	SF	ZF	PF	CF
0	0	0	0	0

```
something:
00401000   mov cx,3
00401001   sub ax,cx
00401003   jz main
00401008 jmp end
main:
00401010   mov ax,2
00401011   mov bx,3
00401013   sub ax,bx
00401018   add ax,ax
0040101D   jnz something
end:
0040101E    ret
```

EIP	unknown
AX	-5
BX	3
CX	3

Flag setting

Before you can do a conditional jump, you need to set the flags.

Common instructions are CMP, TEST, or any arithmetic or logic instructions having flag-setting side-effects

CMP - Compare Two Operands

The comparison is performed by subtracting the second operand from the first operand and then setting the status flags in the same manner as the SUB instruction.

With CMP the result is computed, the flags are set, but the result is discarded.

Modifies CF, OF, SF, ZF, AF, and PF.

TEST - Logical Compare

Computes the bit-wise logical AND of first operand and the second operand and sets the SF, ZF, and PF status flags according to the result.

Like CMP - sets flags, and throws away the result

AND - Logical AND

Destination operand can be r/m32 or register.

Source operand can be r/m32 or register or immediate.

No source and destination as r/m32.

and al, bl

	00110011b	(al - 0x33)
AND	01010101b	(bl - 0x55)
result	00010001b	(al - 0x11)

and al, 0x42

	00110011b	(al - 0x33)
AND	01000010b	(imm - 0x42)
result	00000010b	(al - 0x02)

OR - Logical Inclusive OR

Destination operand can be r/m32 or register.

Source operand can be r/m32 or register or immediate.

or al,bl

	00110011b	(al - 0x33)
OR	01010101b	(bl - 0x55)
result	01110111b	(al - 0x77)

or al, 0x42

	00110011b	(al - 0x33)
OR	01000010b	(imm - 0x42)
result	01110011b	(al - 0x73)

XOR - Logical Exclusive OR

Commonly used to zero a register.

xor al,al

	00110011b	(al - 0x33)
XOR	00110011b	(al - 0x33)
result	00000000b	(al - 0x00)

xor al, 0x42

	00110011b	(al - 0x33)
OR	01000010b	(imm - 0x42)
result	01110001b	(al - 0x71)

NOT - One's Complement Negation

Single source/destination operand can be r/m32.

not al

NOT	00110011b	(al - 0x33)
result	11001100b	(al - 0xCC)

not [al+bl]

al	0x10000000
bl	0x00001234
al+bl	0x10001234
[al+bl]	0 (assumed memory at 0x10001234)
NOT	00000000b
result	11111111b

SHL - Shift Logical Left

Can be explicitly used with the C << operator.

First operand (source and destination) is r/m32.

Second operand is either cl (lowest byte of ecx), or a 1 byte immediate. The 2nd operand is the number of places to shift.

Multiplies by 2 more efficiently.

Bits shifted off the left hand side set the carry flag (CF).

sh cl,2

	00110011b	(cl - 0x33)
result	11001100b	(cl - 0xCC) CF = 0

shl cl,3

	00110011b	(cl - 0x33)
result	10011000b	(cl - 0x98) CF = 1

SHR - Shift Logical Right

Can be explicitly used with the C >> operator.

First operand (source and destination) is r/m32.

Second operand is either cl (lowest byte of ecx), or a 1 byte immediate. The 2nd operand is the number of places to shift.

It divides the register by 2 for each place the value is shifted.

Bits shifted off the right hand side set the carry flag (CF).

shr cl,2

	00110011b	(cl - 0x33)
result	00001100b	(cl - 0x0C) CF = 1

shr cl,3

	00110011b	(cl - 0x33)
result	00000110b	(cl - 0x06) CF = 0

REP - Repeat Prefix

It is an instruction prefix.

Repeat a single instruction multiple times.

All rep operations use ecx register as a counter to determine how many times to loop. Each time it decrements ecx. Once ecx == 0, it continues.

REP STOS - Repeat Store String

Either moves one byte at a time or one dword at a time.

Either fill byte at [edi] with al or fill dword at [edi] with eax.

Moves the edi register forward one byte or one dword at a time, so that the repeated store operation is storing into consecutive locations.

So there are 3 things which must happen before the actual rep stos occurs: set edi to the start destination, eax/al to the value to store, and ecx to the number of times to store

REP STOS - Example

Set edi - the destination:

```
004113AC lea edi,[ebp-0F0h]
```

Set ecx - the count

```
004113B2 mov ecx,3Ch
```

Set eax - the value

```
004113B7 mov eax,0CCCCCCCCh
```

Start the repeated store

```
004113BC rep stos dword ptr [edi]
```

Stores 0x3C copies of the dword 0xCCCCCCCC starting at ebp-0xF0

REP MOVS - Repeat Move Data String to String

Either moves one byte at a time or one dword at a time.

Either move byte at [esi] to byte at [edi] or move dword at [esi] to dword at [edi].

Moves the esi and edi registers forward one byte or one dword at a time, so that the repeated store operation is storing into consecutive locations.

So there are 3 things which must happen before the actual rep movs occurs: set esi to the start source, set edi to the start destination, and set ecx to the number of times to move

ENTER - High Level Procedure Init

Pushes EBP and then sets EBP to ESP.

Reserves memory for local variables in the stack.

```
1   enter 20,0
```

LEAVE - High Level Procedure Exit

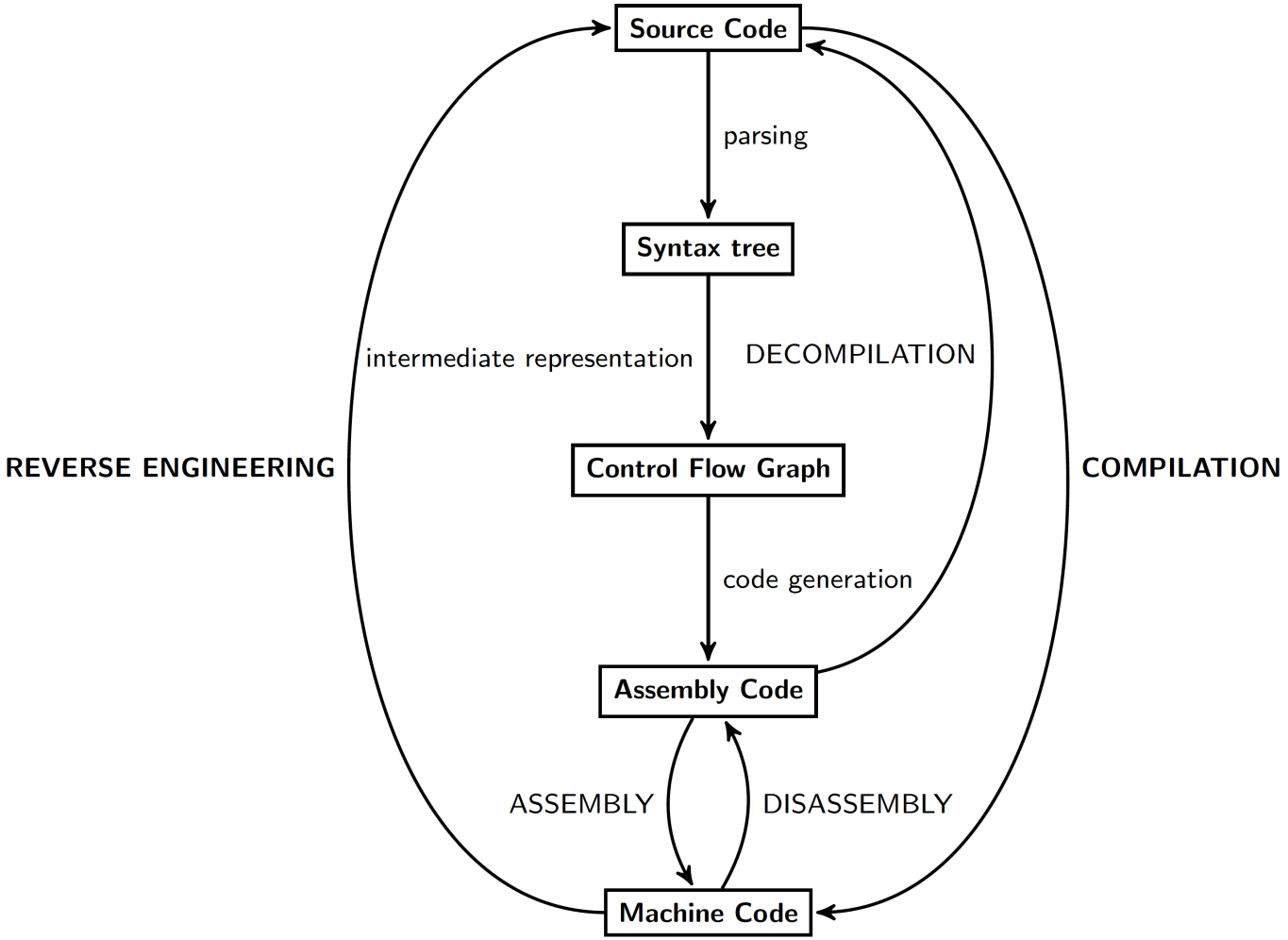
Set ESP to EBP, then pop EBP.

It finishes the execution.

Depends on compiler and options.

```
1  mov eax,dword ptr [ebp+8]
2  pop esi
3  pop edi
4  leave
5  ret
```

C to Assembly



IF / ELSE

```
1  if( a > 5)
2      printf("A is greater than 5");
3  else
4      printf("A is smaller or equal to 5");
5  return;
```

IF / ELSE

Imagine that 'a' value is in ax

```
 1       cmp ax,5
 2       jg greater
 3       jmp smaller
 4   greater:
 5       mov rdi,msgGreater
 6       mov rax,0
 7       call printf
 8       jmp end
 9   smaller:
10       mov rdi,msgSmaller
11       mov rax,0
12       call printf
13   end:
14       leave
```

FOR

```
1  for(i=0;i<10;i++)
2          printf("Hello");
```

FOR

```
 1        mov cx,0
 2   loop1:
 3        cmp cx,10
 4        jge end
 5        mov rdi,msg
 6        mov rax,0
 7        call printf
 8        add  cx,1
 9        jmp loop1
10   end:
11        mov rax,0
12        call exit
```

Compile to Assembly

The compiler is able to generate assembly code as an intermediate representation:

```
$ gcc -S -masm=intel -m32 test.c
```

This will generate a file named test.s containing the assembly code (in intel format) of the source code test.c

Disassembling: Objdump

We can recover the original assembly code using Objdump:

```
$ objdump -M intel -d test
```

This will show the assembly expressions after a disassemble process.

Disassembling: GDB

We can also recover them with GDB:

```
$ gdb test
$> break main
$> run
$> set disassembly-flavor intel
$> disassemble
```

In this case, we are running the program.

MAIN

```
1  int main(int argc, char * argv[]){
2     int a=3;
3     int b=5;
4     int c=a+b;
5     return 1;
6  }
```

MAIN

```
1  main:
2          push    ebp
3          mov     ebp, esp
4          sub     esp, 16
5          mov     DWORD PTR [ebp-4], 3
6          mov     DWORD PTR [ebp-8], 5
7          mov     eax, DWORD PTR [ebp-8]
8          mov     edx, DWORD PTR [ebp-4]
9          add     eax, edx
10         mov     DWORD PTR [ebp-12], eax
11         mov     eax, 1
12         leave
13         ret
```

Function Call

```
1  int mySum(int a, int b){
2    int c=3;
3    int total;
4    total=a+c+b;
5    return total;
6  }
7
8  int main(int argc, char * argv[]){
9    int a=3;
10   int b=5;
11   mySum(a,b);
12   return 1;
13 }
```

Function Call (main)

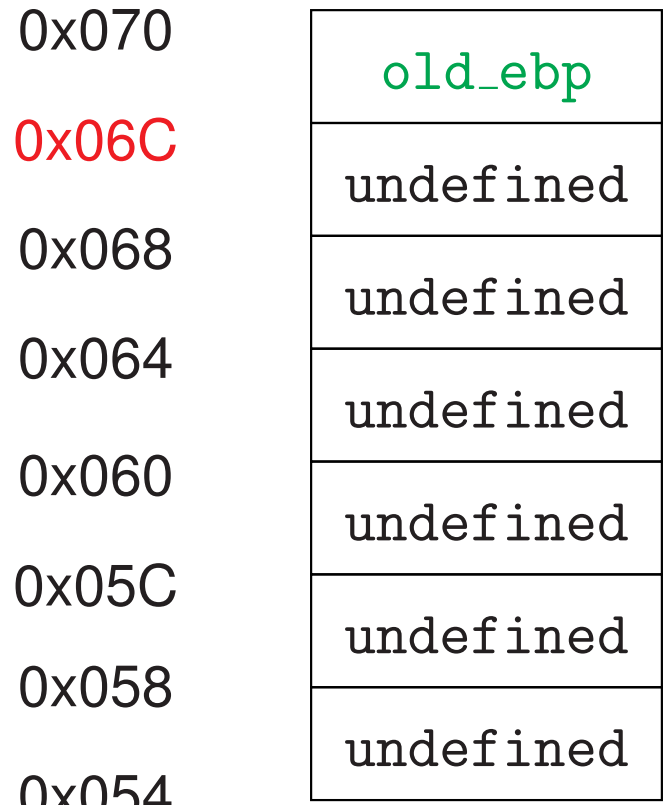
```
1  main:
2          push    ebp
3          mov     ebp, esp
4          sub     esp, 24
5          mov     DWORD PTR [ebp-4], 3
6          mov     DWORD PTR [ebp-8], 5
7          mov     eax, DWORD PTR [ebp-8]
8          mov     DWORD PTR [esp+4], eax
9          mov     eax, DWORD PTR [ebp-4]
10         mov     DWORD PTR [esp], eax
11         call    mySum
12         mov     eax, 1
13         leave
14         ret
```

Function Call (mySum)

```
1   mySum:
2               push    ebp
3               mov     ebp, esp
4               sub     esp, 16
5               mov     DWORD PTR [ebp-4], 3
6               mov     eax, DWORD PTR [ebp-4]
7               mov     edx, DWORD PTR [ebp+8]
8               add     edx, eax
9               mov     eax, DWORD PTR [ebp+12]
10              add     eax, edx
11              mov     DWORD PTR [ebp-8], eax
12              mov     eax, DWORD PTR [ebp-8]
13              leave
14              ret
```

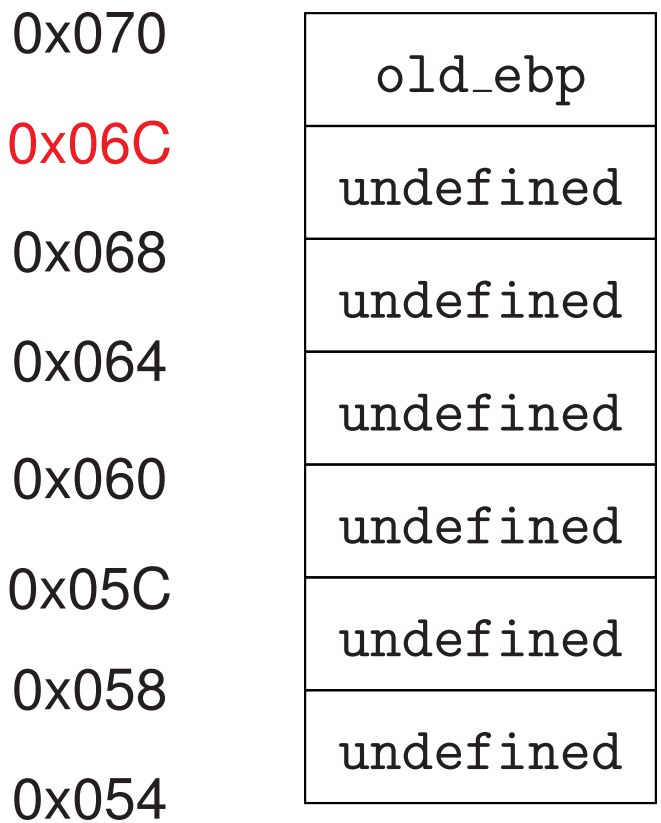
Example

```
main:
 push ebp
 mov     ebp, esp
 sub     esp, 24
 mov     DWORD PTR [ebp-4], 3
 mov     DWORD PTR [ebp-8], 5
 mov     eax, DWORD PTR [ebp-8]
 mov     DWORD PTR [esp+4], eax
 mov     eax, DWORD PTR [ebp-4]
 mov     DWORD PTR [esp], eax
 call    mySum
 mov     eax, 1
 leave
 ret
```



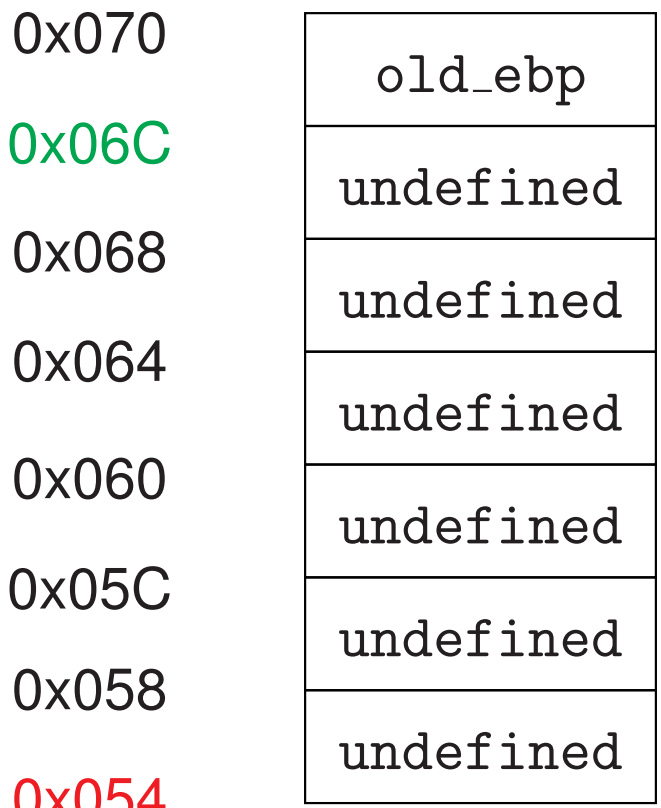
Example

```
main:
 push    ebp
 mov ebp, esp
 sub     esp, 24
 mov     DWORD PTR [ebp-4], 3
 mov     DWORD PTR [ebp-8], 5
 mov     eax, DWORD PTR [ebp-8]
 mov     DWORD PTR [esp+4], eax
 mov     eax, DWORD PTR [ebp-4]
 mov     DWORD PTR [esp], eax
 call    mySum
 mov     eax, 1
 leave
 ret
```



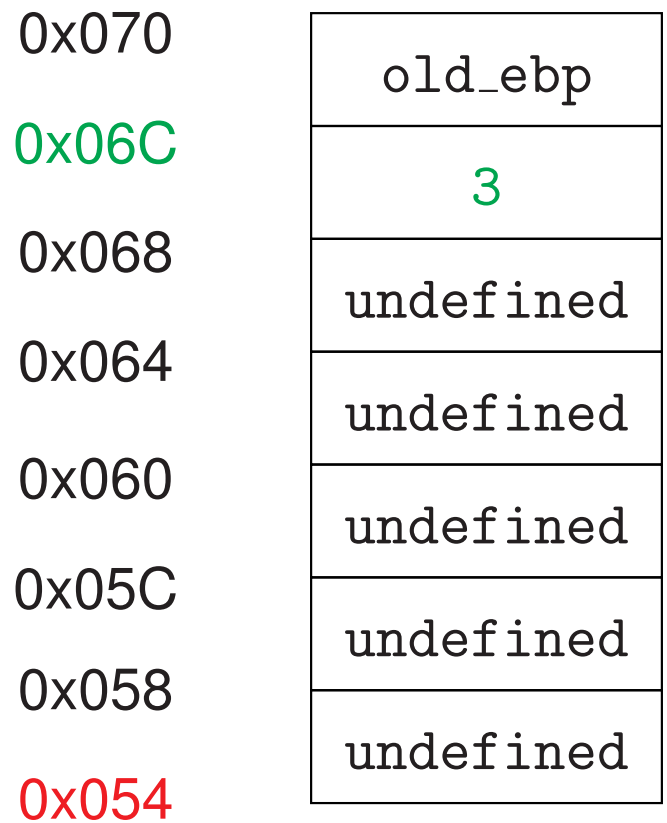
Example

```
main:
 push    ebp
 mov     ebp, esp
 sub esp, 24
 mov     DWORD PTR [ebp-4], 3
 mov     DWORD PTR [ebp-8], 5
 mov     eax, DWORD PTR [ebp-8]
 mov     DWORD PTR [esp+4], eax
 mov     eax, DWORD PTR [ebp-4]
 mov     DWORD PTR [esp], eax
 call    mySum
 mov     eax, 1
 leave
 ret
```



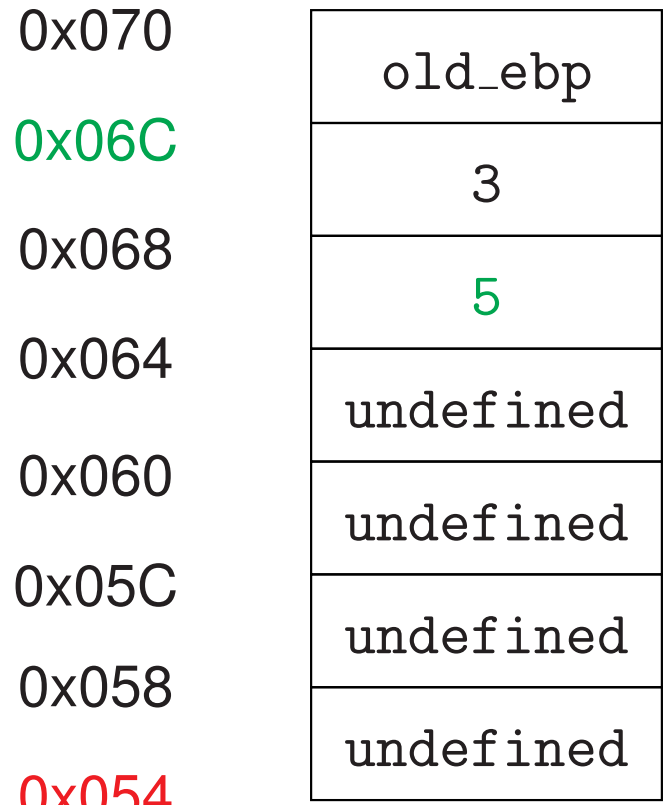
Example

```
main:
 push    ebp
 mov     ebp, esp
 sub     esp, 24
 mov DWORD PTR [ebp-4], 3
 mov     DWORD PTR [ebp-8], 5
 mov     eax, DWORD PTR [ebp-8]
 mov     DWORD PTR [esp+4], eax
 mov     eax, DWORD PTR [ebp-4]
 mov     DWORD PTR [esp], eax
 call    mySum
 mov     eax, 1
 leave
 ret
```



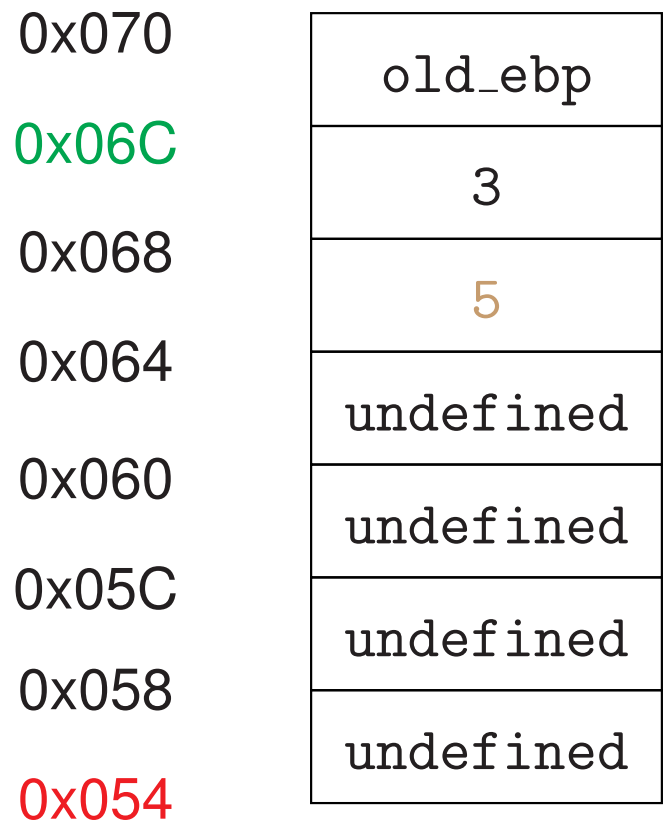
Example

```
main:
 push    ebp
 mov     ebp, esp
 sub     esp, 24
 mov     DWORD PTR [ebp-4], 3
 mov DWORD PTR [ebp-8], 5
 mov     eax, DWORD PTR [ebp-8]
 mov     DWORD PTR [esp+4], eax
 mov     eax, DWORD PTR [ebp-4]
 mov     DWORD PTR [esp], eax
 call    mySum
 mov     eax, 1
 leave
 ret
```



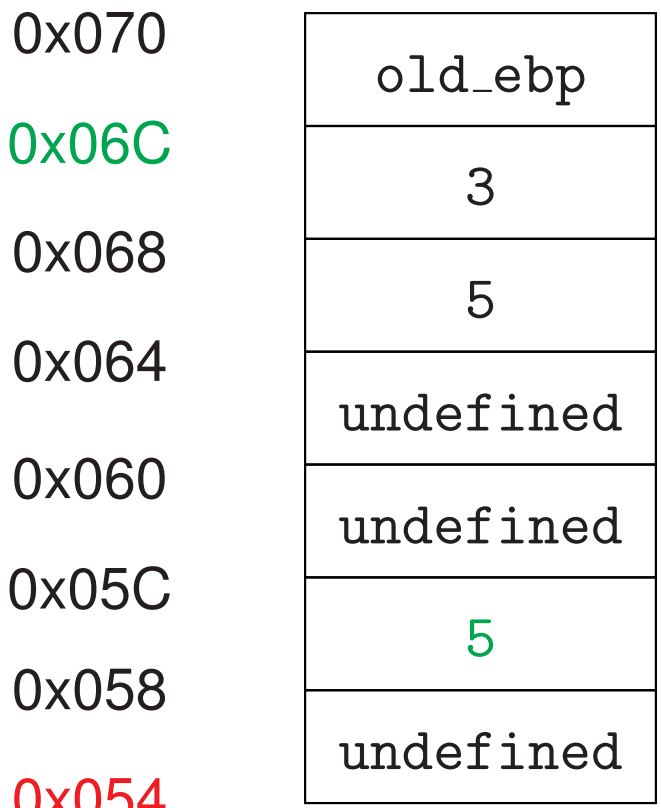
Example

```
main:
 push    ebp
 mov     ebp, esp
 sub     esp, 24
 mov     DWORD PTR [ebp-4], 3
 mov     DWORD PTR [ebp-8], 5
 mov eax, DWORD PTR [ebp-8]
 mov     DWORD PTR [esp+4], eax
 mov     eax, DWORD PTR [ebp-4]
 mov     DWORD PTR [esp], eax
 call    mySum
 mov     eax, 1
 leave
 ret
```



Example

```
main:
 push    ebp
 mov     ebp, esp
 sub     esp, 24
 mov     DWORD PTR [ebp-4], 3
 mov     DWORD PTR [ebp-8], 5
 mov     eax, DWORD PTR [ebp-8]
 mov DWORD PTR [esp+4], eax
 mov     eax, DWORD PTR [ebp-4]
 mov     DWORD PTR [esp], eax
 call    mySum
 mov     eax, 1
 leave
 ret
```

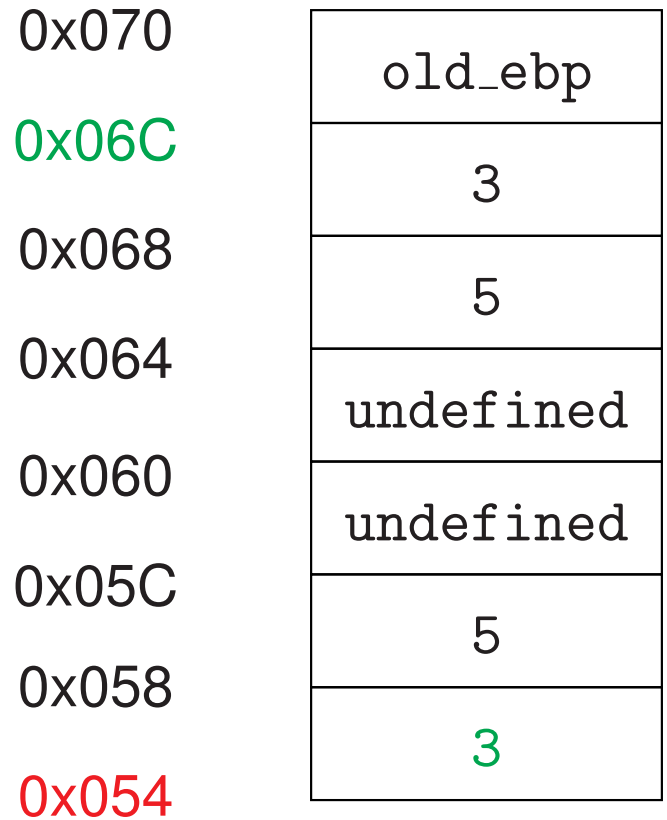


Example

```
main:
 push    ebp
 mov     ebp, esp
 sub     esp, 24
 mov     DWORD PTR [ebp-4], 3
 mov     DWORD PTR [ebp-8], 5
 mov     eax, DWORD PTR [ebp-8]
 mov     DWORD PTR [esp+4], eax
 mov eax, DWORD PTR [ebp-4]
 mov     DWORD PTR [esp], eax
 call    mySum
 mov     eax, 1
 leave
 ret
```

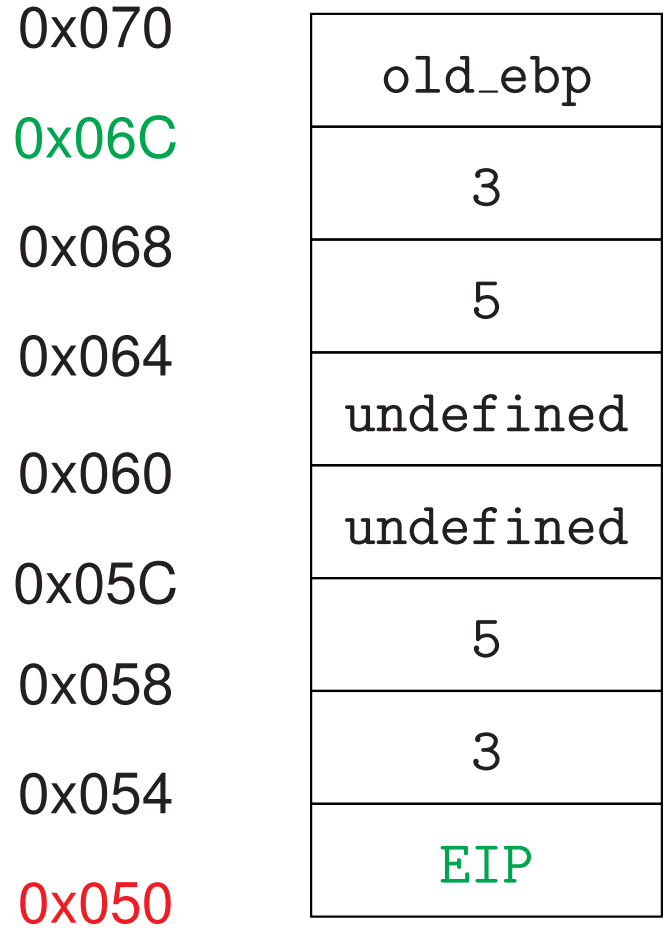
Example

```
main:
 push    ebp
 mov     ebp, esp
 sub     esp, 24
 mov     DWORD PTR [ebp-4], 3
 mov     DWORD PTR [ebp-8], 5
 mov     eax, DWORD PTR [ebp-8]
 mov     DWORD PTR [esp+4], eax
 mov     eax, DWORD PTR [ebp-4]
 mov DWORD PTR [esp], eax
 call    mySum
 mov     eax, 1
 leave
 ret
```



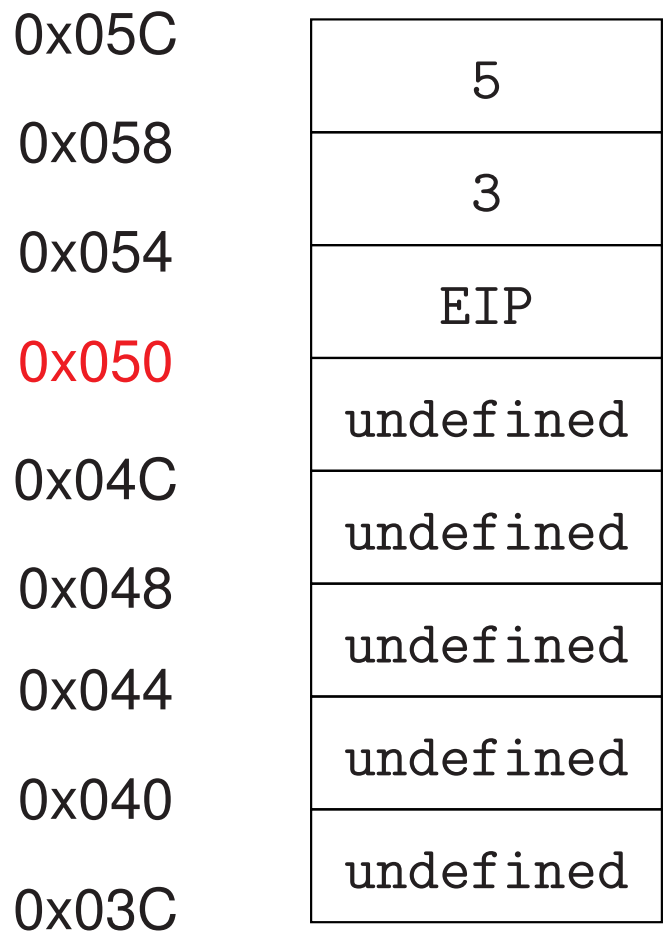
Example

```
main:
  push    ebp
  mov     ebp, esp
  sub     esp, 24
  mov     DWORD PTR [ebp-4], 3
  mov     DWORD PTR [ebp-8], 5
  mov     eax, DWORD PTR [ebp-8]
  mov     DWORD PTR [esp+4], eax
  mov     eax, DWORD PTR [ebp-4]
  mov     DWORD PTR [esp], eax
  call mySum
  mov     eax, 1
  leave
  ret
```



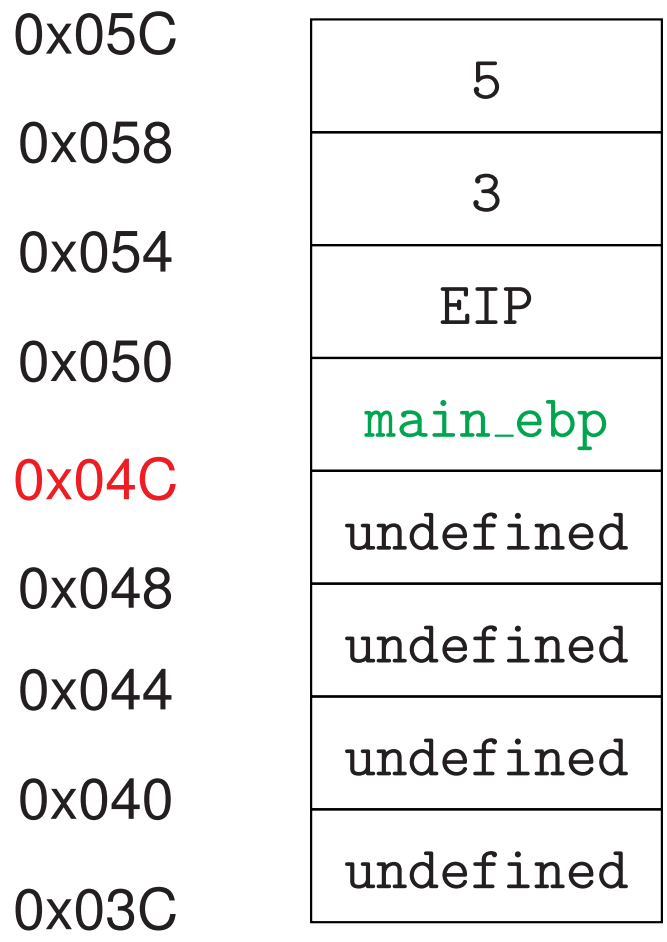
Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



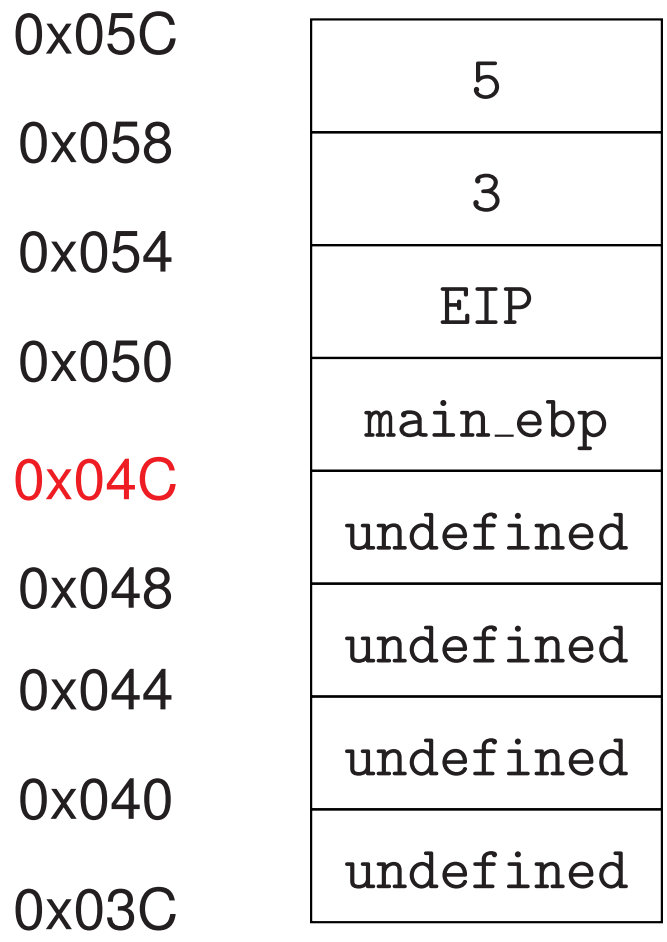
Example

```
mySum:
  push ebp
  mov     ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



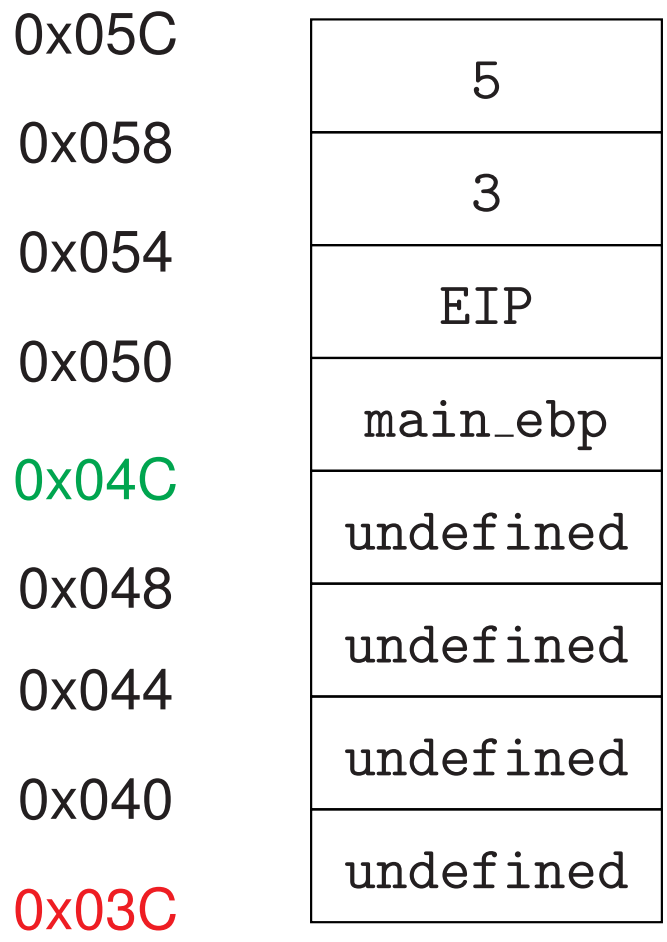
Example

```
mySum:
  push    ebp
  mov ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



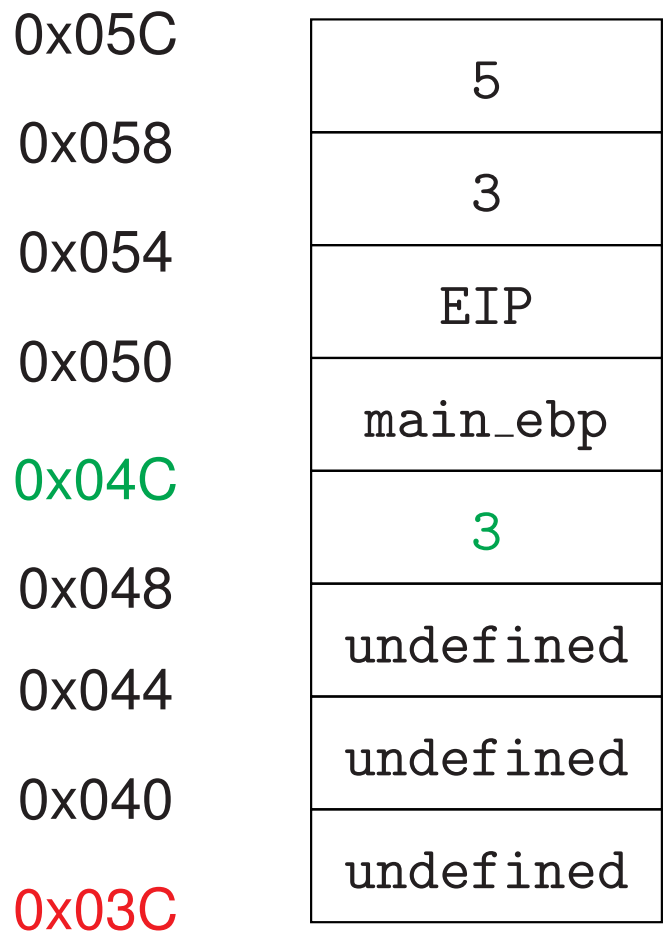
Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



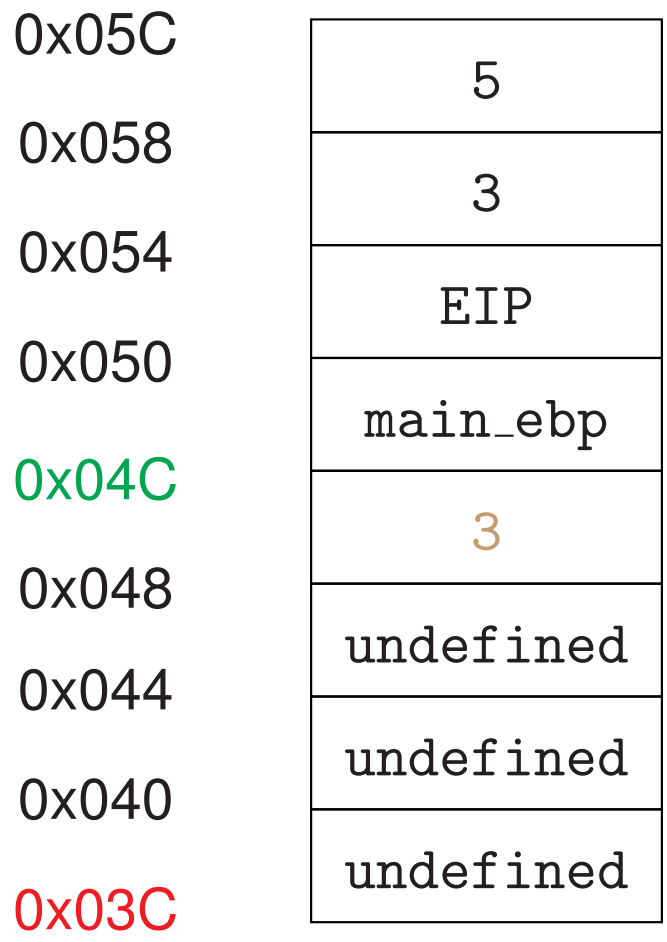
Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub     esp, 16
  mov DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



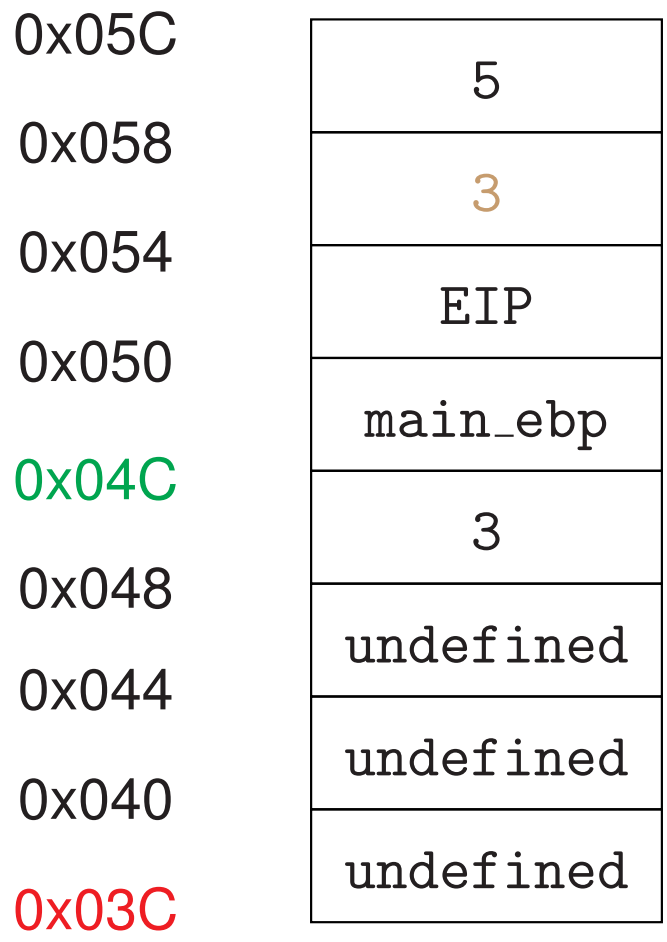
Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



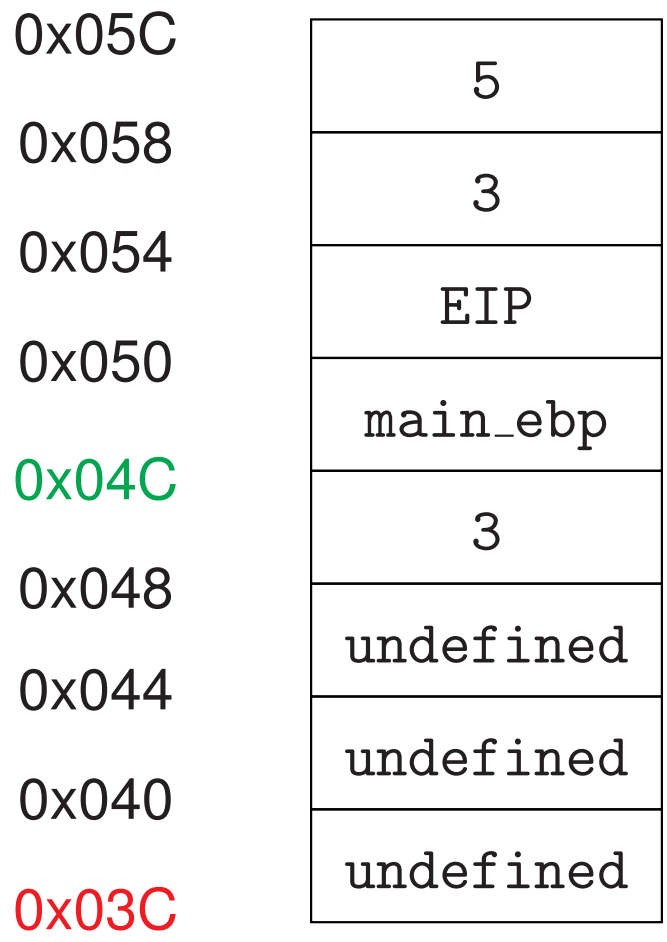
Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



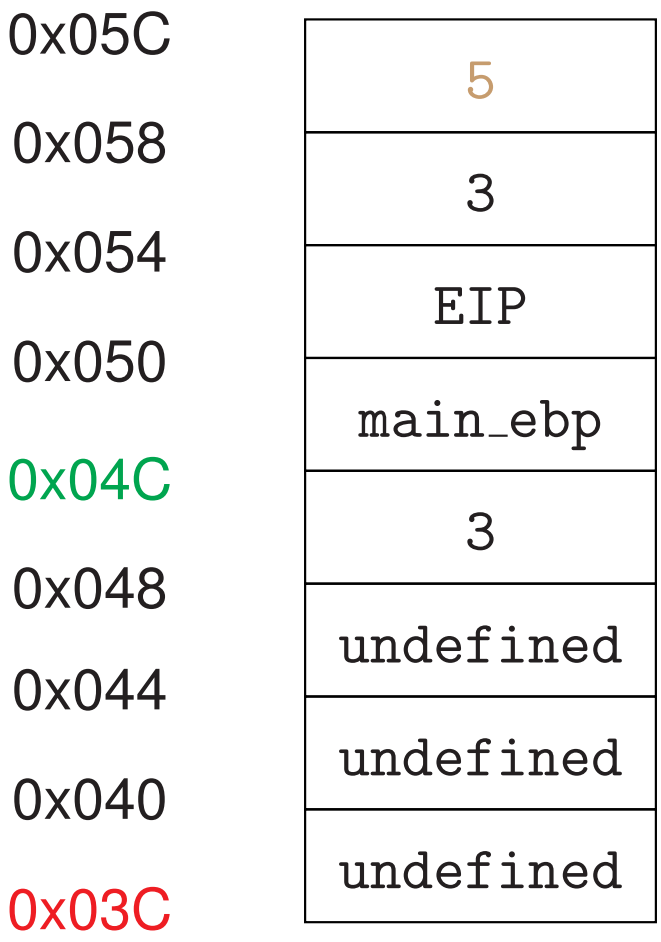
Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



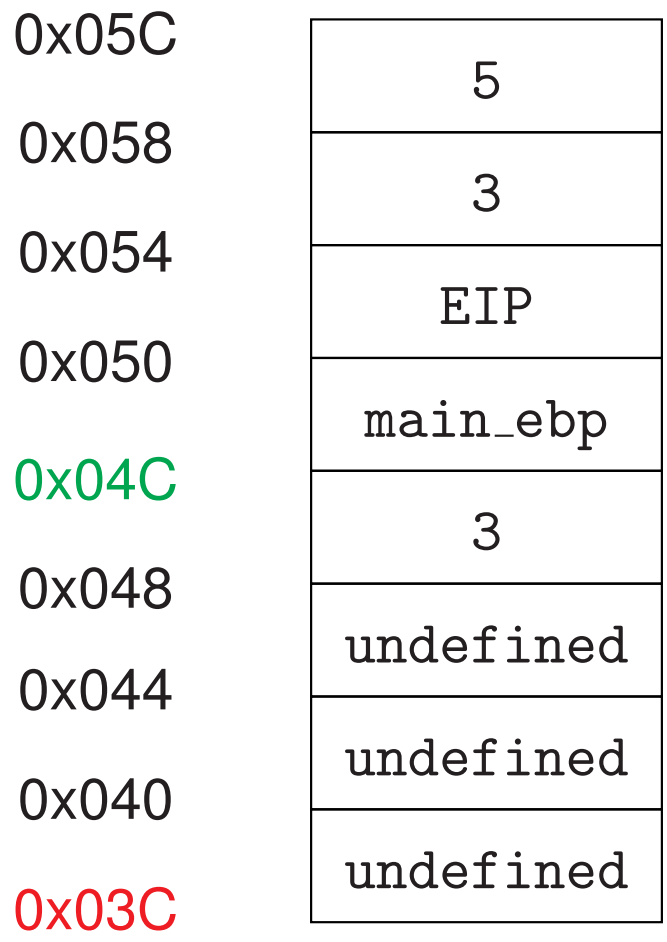
Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



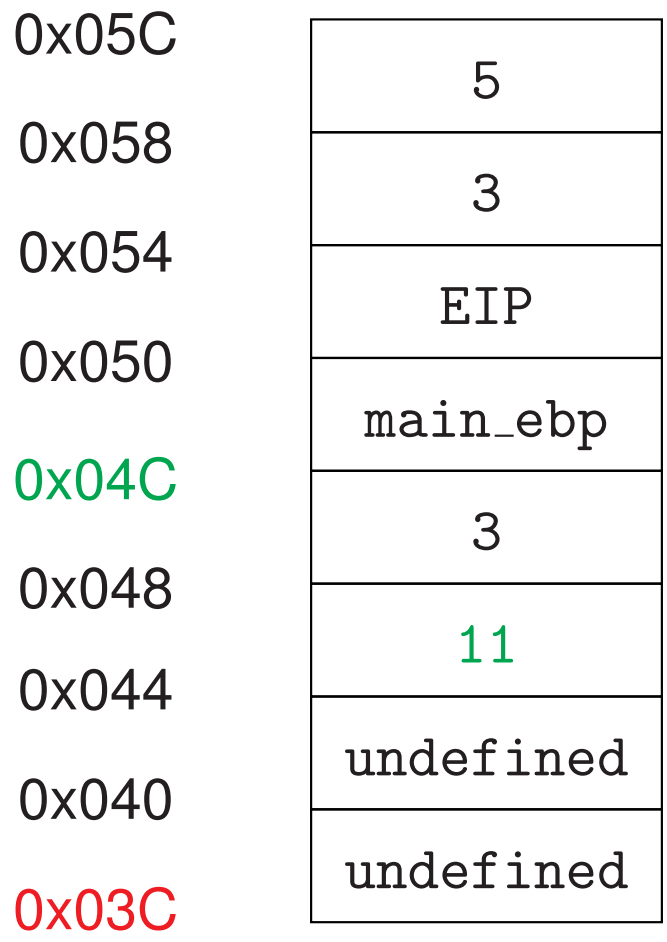
Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



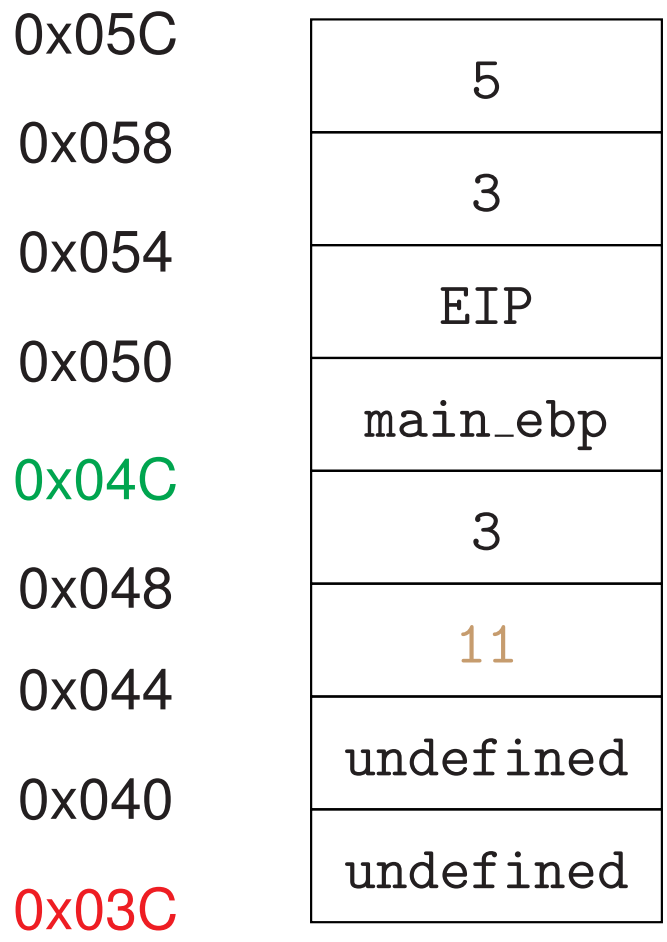
Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



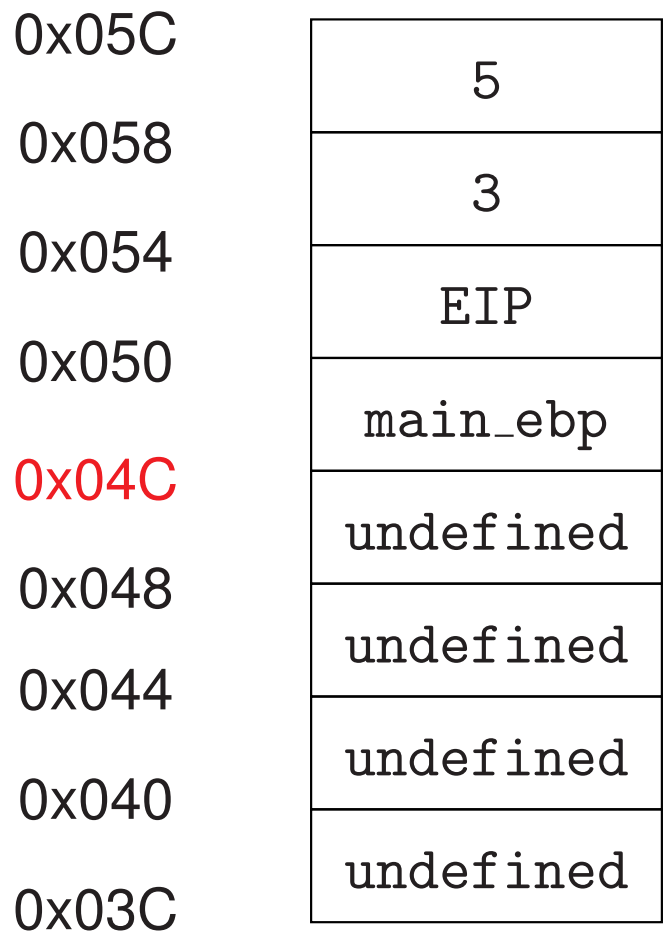
Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov eax, DWORD PTR [ebp-8]
  leave
  ret
```



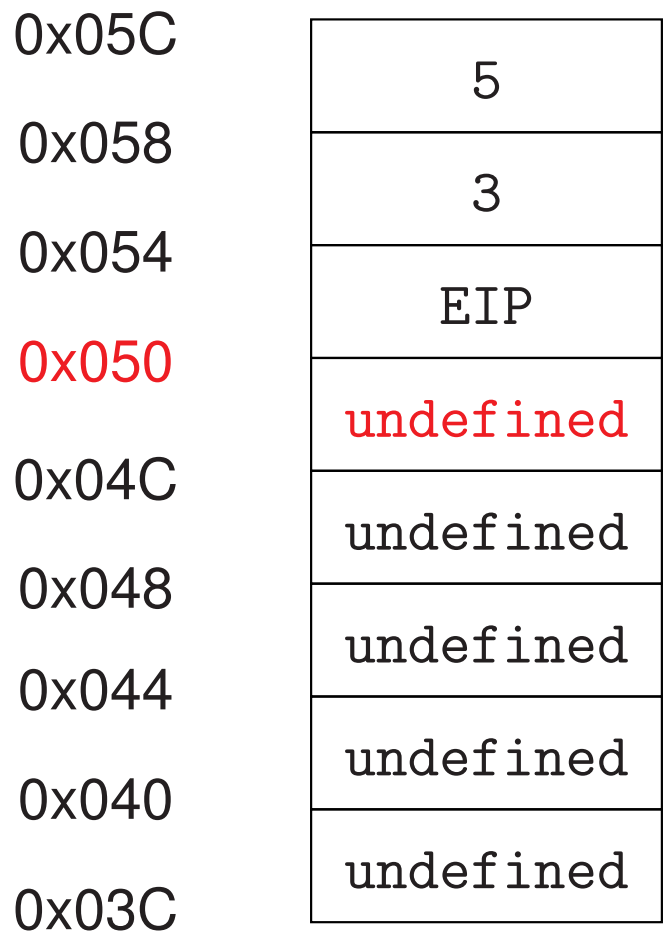
Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



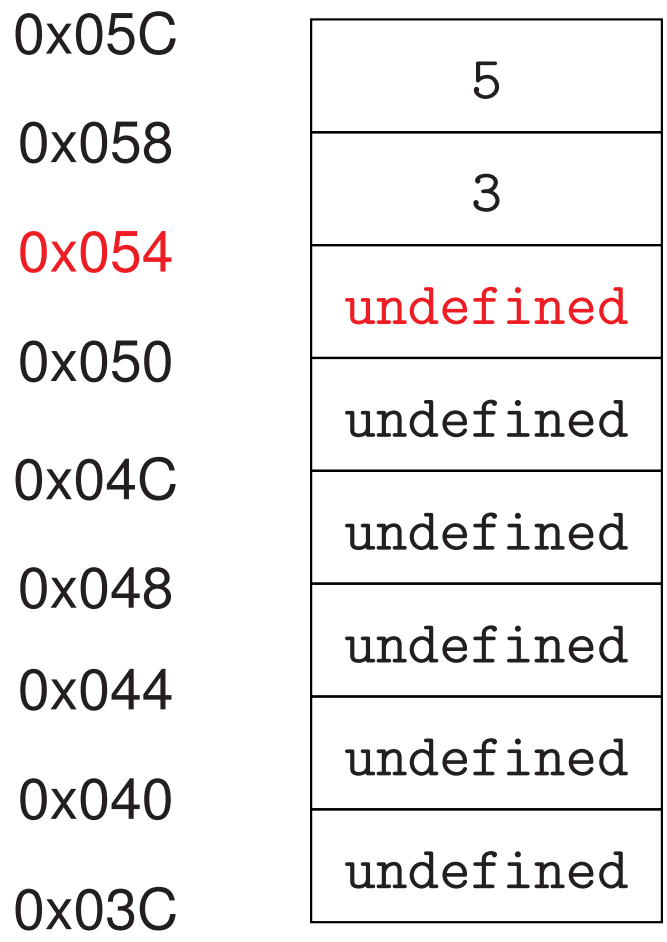
Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



Example

```
mySum:
  push    ebp
  mov     ebp, esp
  sub     esp, 16
  mov     DWORD PTR [ebp-4], 3
  mov     eax, DWORD PTR [ebp-4]
  mov     edx, DWORD PTR [ebp+8]
  add     edx, eax
  mov     eax, DWORD PTR [ebp+12]
  add     eax, edx
  mov     DWORD PTR [ebp-8], eax
  mov     eax, DWORD PTR [ebp-8]
  leave
  ret
```



Exercises

5. Modify exercise 4 and include AND, OR and XOR operations.
6. Ask the user to introduce two numbers and print the value of the greater and smaller.
7. Ask the user how many numbers he wants to introduce. Latter, ask him every number (the maximum will be 100). Print the maximum and minimum of these numbers.
8. Modify exercise 4 to use the stack for controlling the variables.