Simple Instructions

Simple Instructions

- **mov destination, source**
  - Moves data from one location to another
- We use Intel format throughout the book, with destination first
- Remember indirect addressing
  - [ebx] means the memory location pointed to by EBX

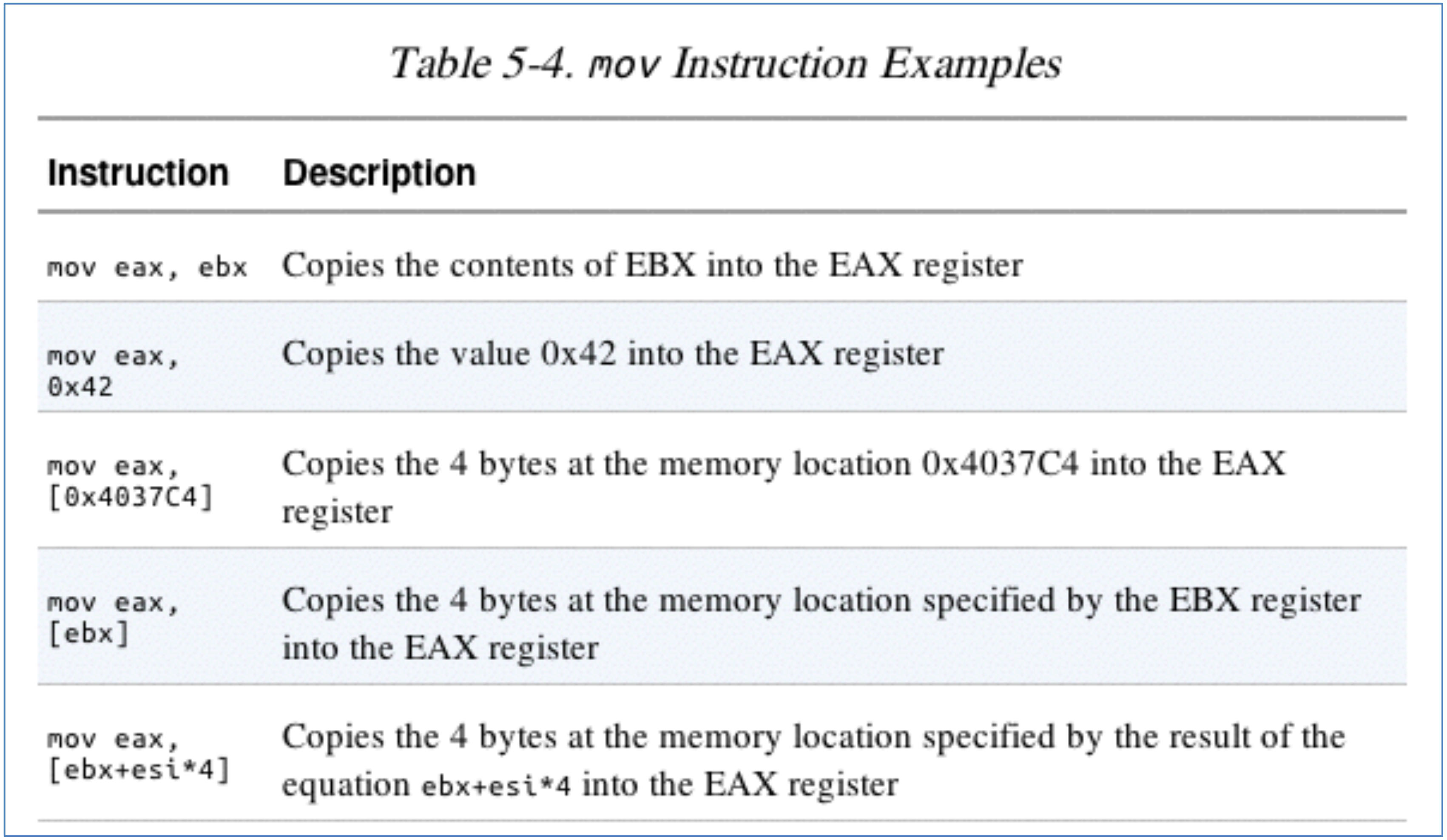


Table 5-4. `mov` Instruction Examples

Instruction	Description
`mov eax, ebx`	Copies the contents of EBX into the EAX register
`mov eax, 0x42`	Copies the value 0x42 into the EAX register
`mov eax, [0x4037C4]`	Copies the 4 bytes at the memory location 0x4037C4 into the EAX register
`mov eax, [ebx]`	Copies the 4 bytes at the memory location specified by the EBX register into the EAX register
`mov eax, [ebx+esi*4]`	Copies the 4 bytes at the memory location specified by the result of the equation `ebx+esi*4` into the EAX register

lea (Load Effective Address)

- lea destination, source
- lea eax, [ebx+8]
  - Puts ebx + 8 into eax
- Compare
  - mov eax, [ebx+8]
  - Moves the data at location ebx+8 into eax

Figure 5-5 shows values for registers EAX and EBX on the left and the information contained in memory on the right. EBX is set to 0xB30040. At address 0xB30048 is the value `0x20`. The instruction `mov eax, [ebx+8]` places the value `0x20` (obtained from memory) into EAX, and the instruction `lea eax, [ebx+8]` places the value `0xB30048` into EAX.

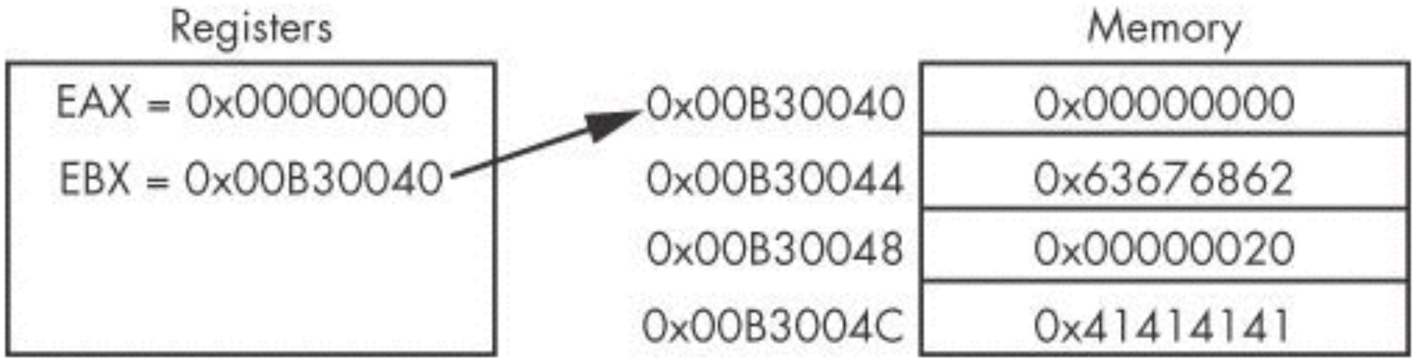


Figure 5-5. EBX register used to access memory

Arithmetic

- **sub** Subtracts
- **add** Adds
- **inc** Increments
- **dec** Decrements
- **mul** Multiplies
- **div** Divides

Arithmetic

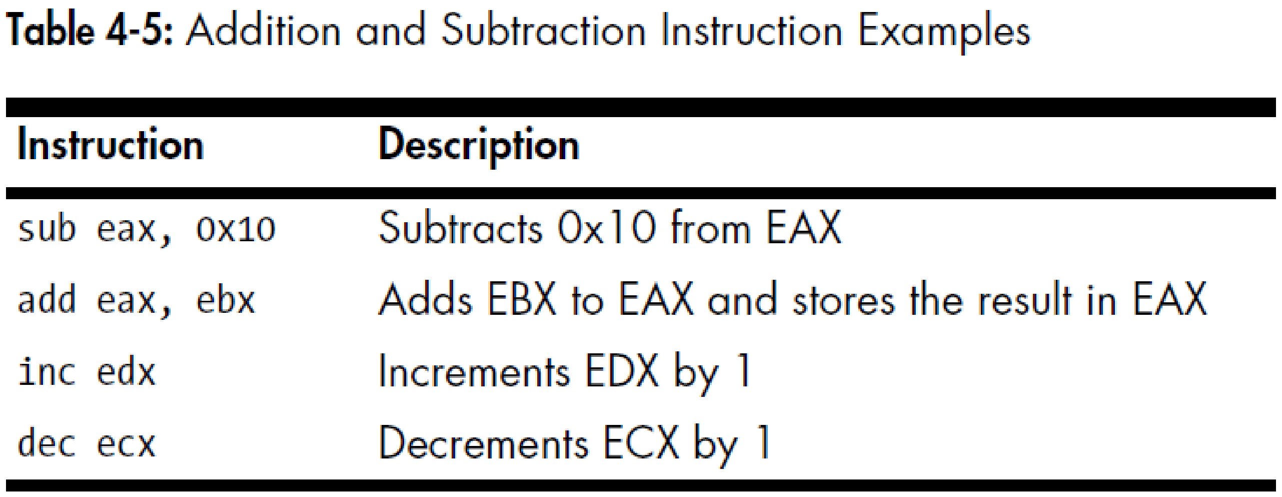


Table 4-5: Addition and Subtraction Instruction Examples

Instruction	Description
`sub eax, 0x10`	Subtracts 0x10 from EAX
`add eax, ebx`	Adds EBX to EAX and stores the result in EAX
`inc edx`	Increments EDX by 1
`dec ecx`	Decrements ECX by 1

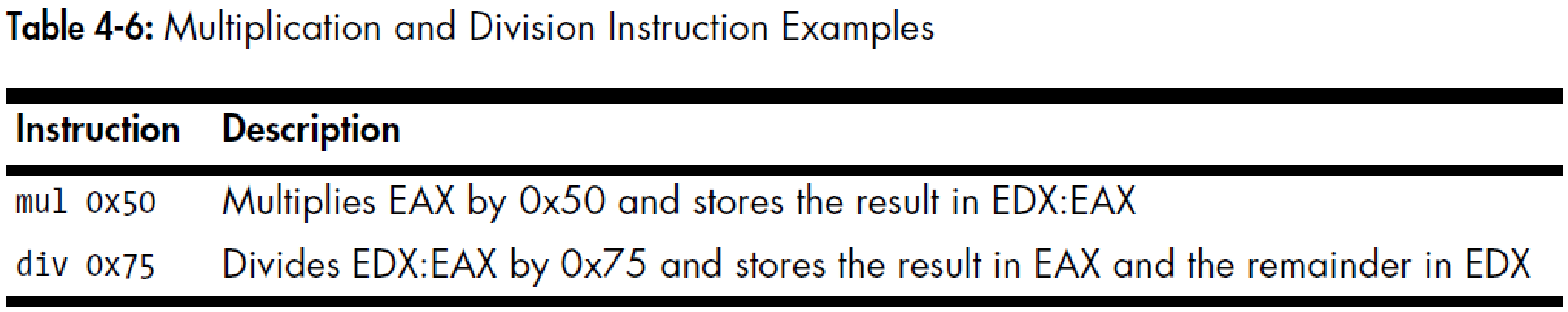


Table 4-6: Multiplication and Division Instruction Examples

Instruction	Description
`mul 0x50`	Multiplies EAX by 0x50 and stores the result in EDX:EAX
`div 0x75`	Divides EDX:EAX by 0x75 and stores the result in EAX and the remainder in EDX

Arithmetic

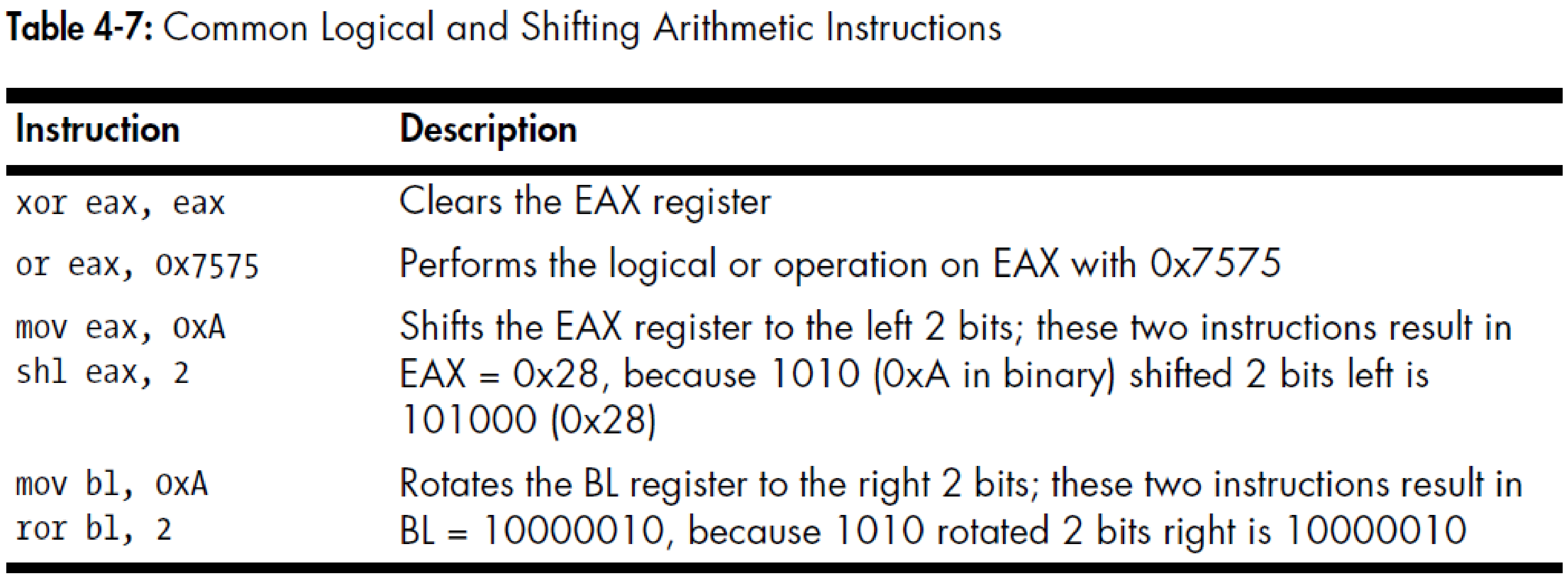


Table 4-7: Common Logical and Shifting Arithmetic Instructions

Instruction	Description
`xor eax, eax`	Clears the EAX register
`or eax, 0x7575`	Performs the logical or operation on EAX with 0x7575

`mov eax, 0xA`
`shl eax, 2`
	Shifts the EAX register to the left 2 bits; these two instructions result in EAX = 0x28, because 1010 (0xA in binary) shifted 2 bits left is 101000 (0x28)

`mov bl, 0xA`
`ror bl, 2`
	Rotates the BL register to the right 2 bits; these two instructions result in BL = 10000010, because 1010 rotated 2 bits right is 10000010

NOP

- Does nothing
- 0x90
- Commonly used as a **NOP Sled**
- Allows attackers to run code even if they are imprecise about jumping to it

The Stack

- Memory for functions, local variables, and flow control
- Last in, First out
- ESP (Extended Stack Pointer) – top of stack
- EBP (Extended Base Pointer) – bottom of stack
- PUSH puts data on the stack
- POP takes data off the stack

Other Stack Instructions

- To enter a function
  - Call or Enter
- To exit a function
  - Leave or Ret

Function Calls

- Small programs that do one thing and return, like printf()
- Prologue
  - Instructions at the start of a function that prepare stack and registers for the function to use
- Epilogue
  - Instructions at the end of a end of a function that restore the stack and registers to their state before the function was called

Stack Frames

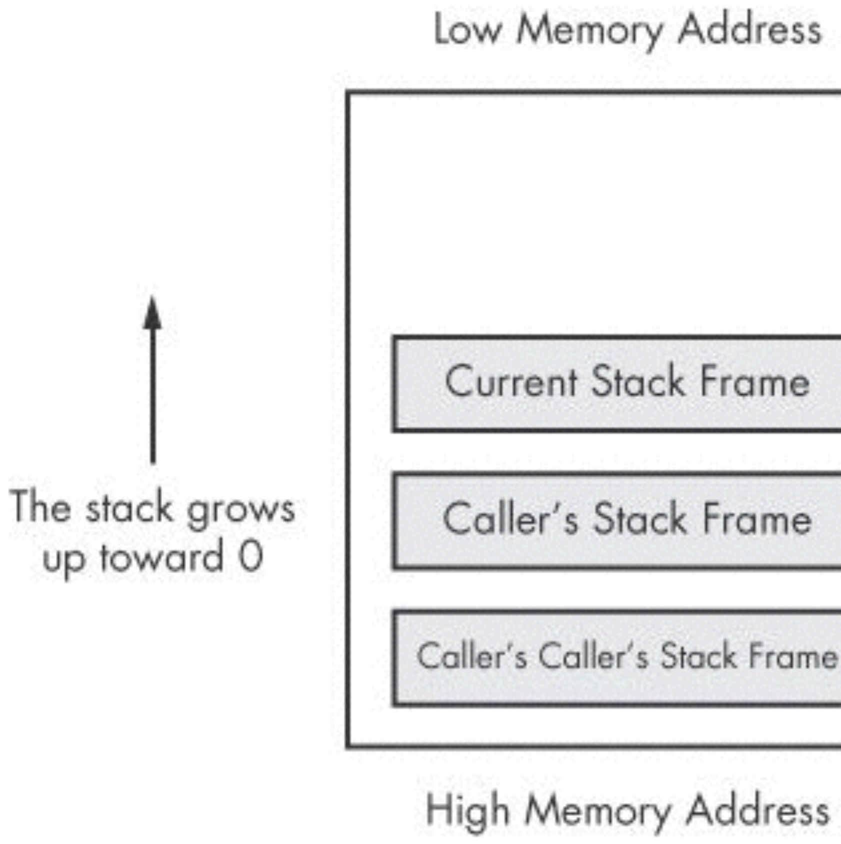


Figure 5-7. x86 stack layout

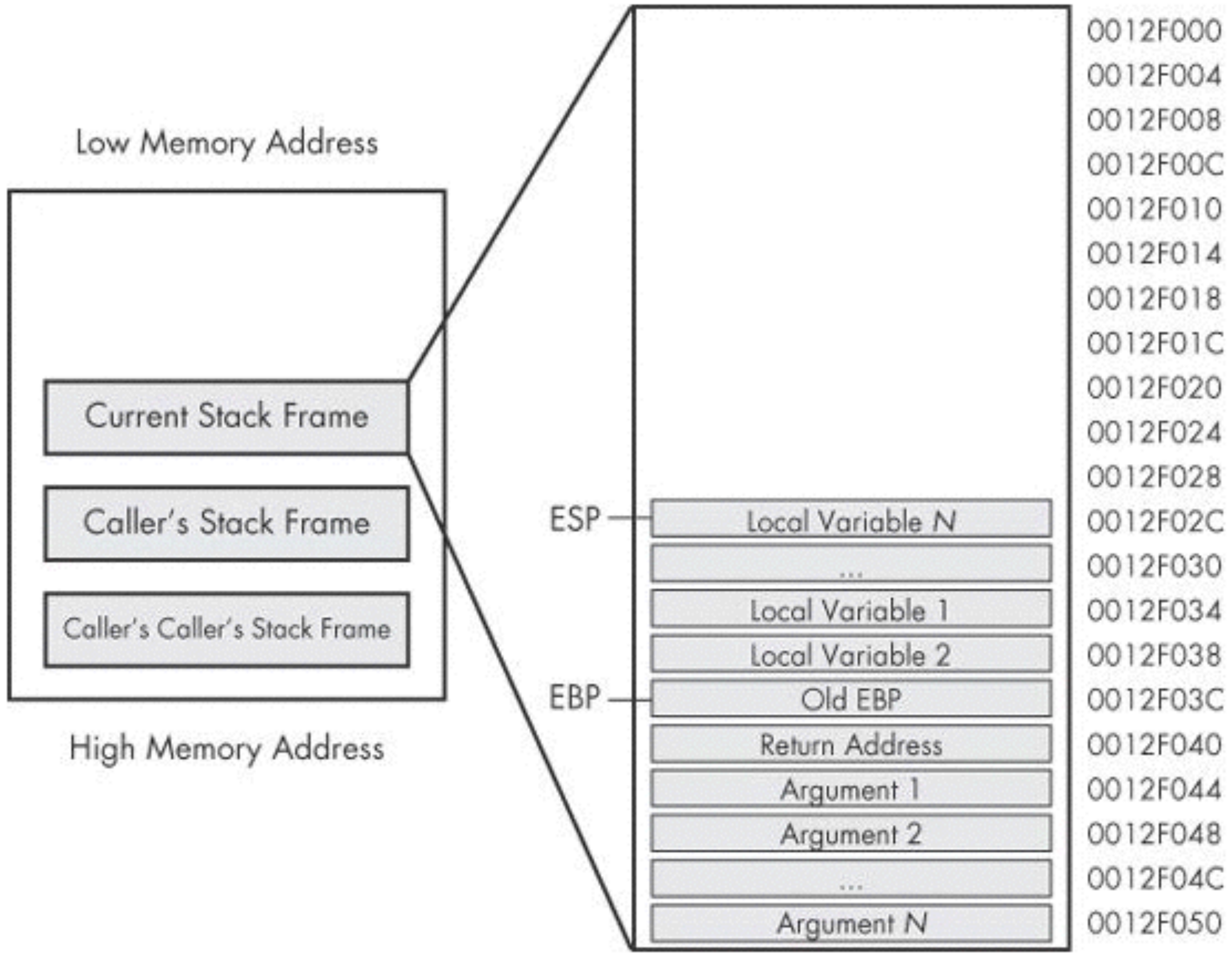
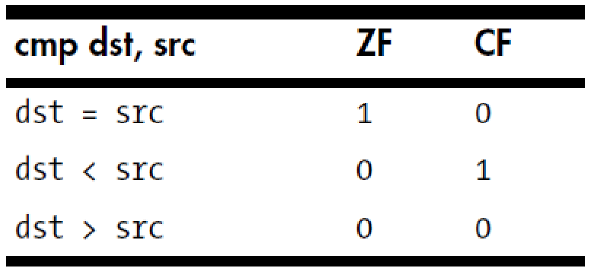


Figure 5-8. Individual stack frame

Conditionals

- test
  - Compares two values the way AND does, but does not alter them
  - test eax, eax
    - Sets Zero Flag if eax is zero
- cmp eax, ebx
  - Sets Zero Flag if the arguments are equal

Table 4-8: cmp Instruction and Flags



cmp dst, src	ZF	CF
dst = src	1	0
dst < src	0	1
dst > src	0	0

Branching

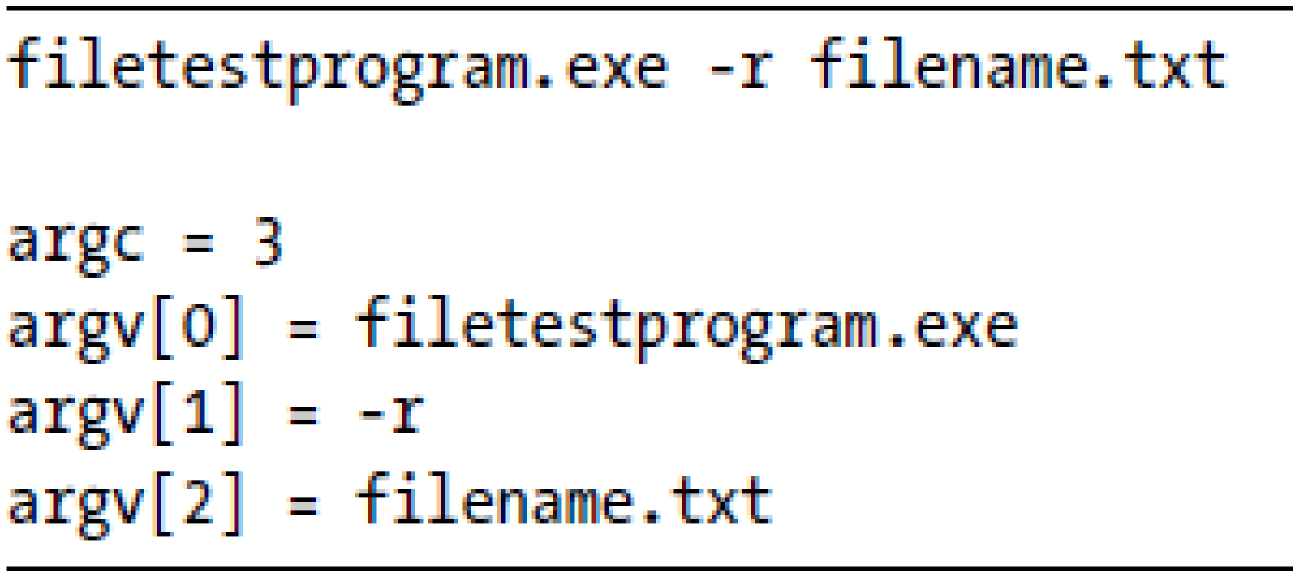
- jz loc
  - Jump to loc if the Zero Flag is set
- jnz loc
  - Jump to loc if the Zero Flag is cleared

C Main Method

- Every C program has a main() function
- int main(int argc, char** argv)
  - argc contains the number of arguments on the command line
  - argv is a pointer to an array of names containing the arguments

C Main Method

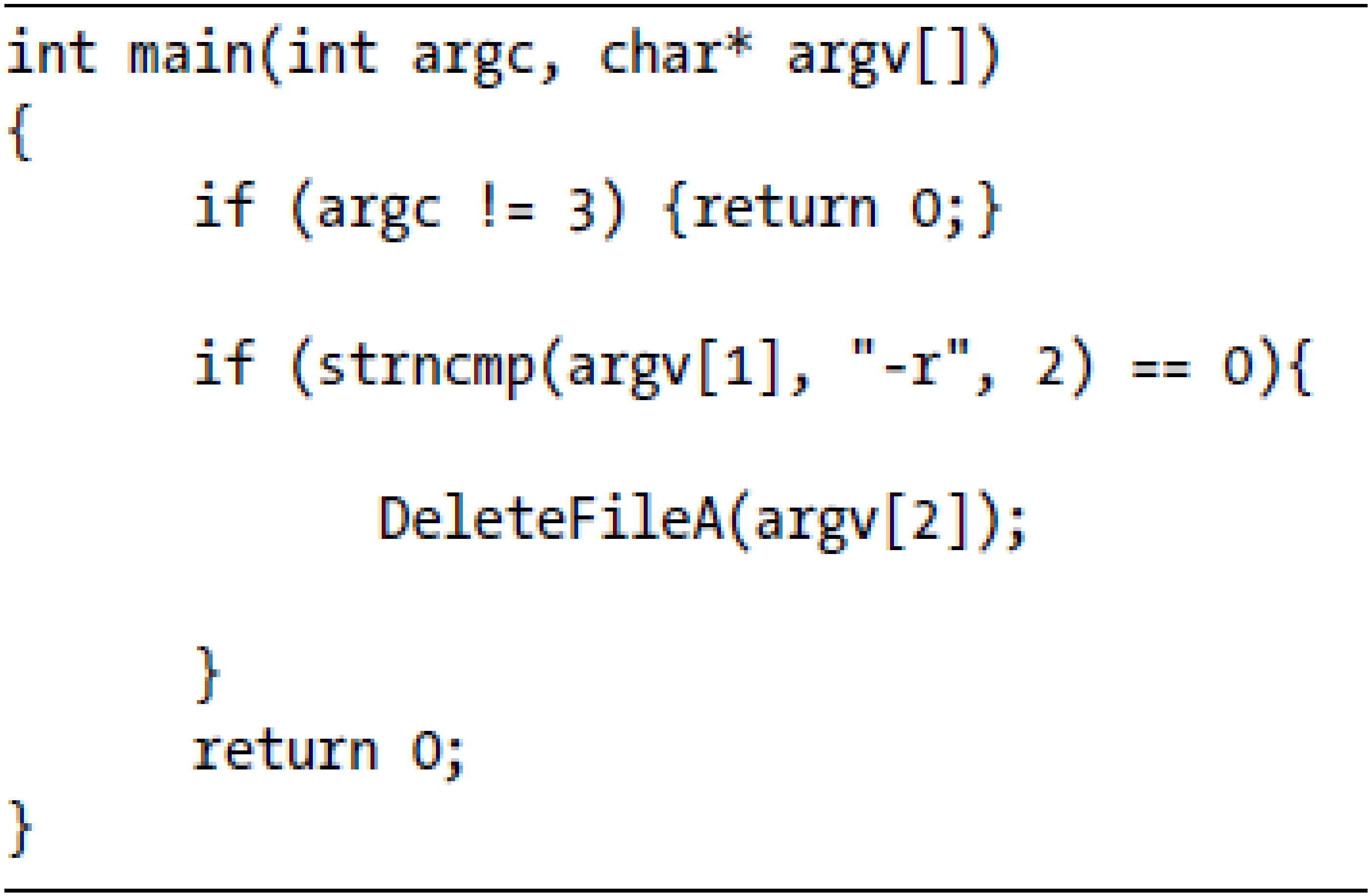
- Example



```
filetestprogram.exe -r filename.txt

argc = 3
argv[0] = filetestprogram.exe
argv[1] = -r
argv[2] = filename.txt
```

C Main Method



```
int main(int argc, char* argv[])
{
        if (argc != 3) {return 0;}

        if (strncmp(argv[1], "-r", 2) == 0){

                DeleteFileA(argv[2]);

        }
        return 0;
}
```

Listing 4-1: C code, main method example

C Main Method

```
004113CE        cmp     [ebp+argc], 3 ❶
004113D2        jz      short loc_4113D8
004113D4        xor     eax, eax
004113D6        jmp     short loc_411414
004113D8        mov     esi, esp
004113DA        push    2                 ; MaxCount
004113DC        push    offset Str2       ; "-r"
004113E1        mov     eax, [ebp+argv]
004113E4        mov     ecx, [eax+4]
004113E7        push    ecx               ; Str1
004113E8        call    strncmp ❷
004113F8        test    eax, eax
004113FA        jnz     short loc_411412
004113FC        mov     esi, esp ❸
004113FE        mov     eax, [ebp+argv]
00411401        mov     ecx, [eax+8]
00411404        push    ecx               ; lpFileName
00411405        call    DeleteFileA
```

1. `004113CE cmp [ebp+argc], 3`: `argc` (argüman sayısı) değerini 3 ile karşılaştırır.
2. `004113D2 jz short loc_4113D8`: Eğer `argc` 3'e eşitse, `loc_4113D8` etiketine atlar.
3. `004113D4 xor eax, eax`: EAX register'ını sıfırlar.
4. `004113D6 jmp short loc_411414`: Sıçrama komutu, `loc_411414` etiketine atlar.
5. `004113D8 mov esi, esp`: ESP değerini ESI register'ına kopyalar. Bu, bir sonraki işlem için yığındaki argümanları işaret eder.
6. `004113DA push 2`: 2 değerini yığına iter. Bu, `strncmp` fonksiyonuna geçirilecek olan ikinci parametredir (maxCount).
7. `004113DC push offset Str2`: `Str2` adresini yığına iter. Bu, `-r` dizisinin adresini `Str2` değişkenine atar.
8. `004113E1 mov eax, [ebp+argv]`: `argv` dizisinin adresini EAX register'ına yükler.
9. `004113E4 mov ecx, [eax+4]`: `argv[1]`'in adresini ECX register'ına yükler. İlk argüman argc'nin tutulduğu yer olduğu için bu `argv[1]`'in adresidir.
10. `004113E7 push ecx`: `argv[1]`'in adresini yığına iter. Bu, `strncmp` fonksiyonuna geçirilecek olan ilk parametredir (Str1).
11. `004113E8 call strncmp`: `strncmp` fonksiyonunu çağırır.
12. `004113F8 test eax, eax`: EAX register'ındaki değeri test eder. `strncmp` fonksiyonunun dönüş değerini kontrol eder.
13. `004113FA jnz short loc_411412`: Eğer EAX register'ındaki değer sıfır değilse (`strncmp` 0 döndürmediyse), `loc_411412` etiketine atlar.
14. `004113FC mov esi, esp`: ESP değerini ESI register'ına kopyalar. Bu, bir sonraki işlem için yığındaki argümanları işaret eder.
15. `004113FE mov eax, [ebp+argv]`: `argv` dizisinin adresini EAX register'ına yükler.
16. `00411401 mov ecx, [eax+8]`: `argv[2]`'nin adresini ECX register'ına yükler. İkinci argüman, `-r` seçeneğinin ardından gelen dosya adıdır.
17. `00411404 push ecx`: `argv[2]`'nin adresini yığına iter. Bu, `DeleteFileA` fonksiyonuna geçirilecek olan ilk ve tek parametredir (lpFileName).
18. `00411405 call DeleteFileA`: `DeleteFileA` fonksiyonunu çağırır.