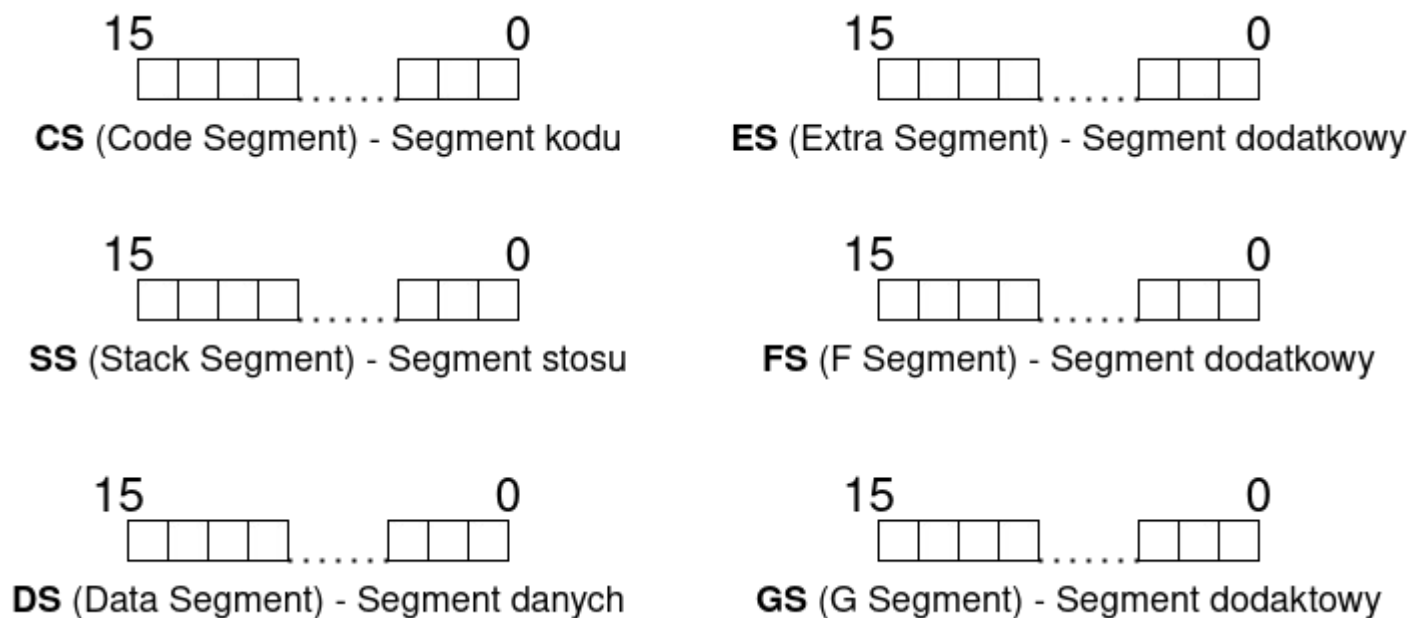


Rejestry Segmentowe

Przeznaczenie konkretnych rejestrów segmentowych jest zależne od systemu operacyjnego. Niezależnie od procesora mają one rozmiar 16-bitów (rysunek 2.2.13).



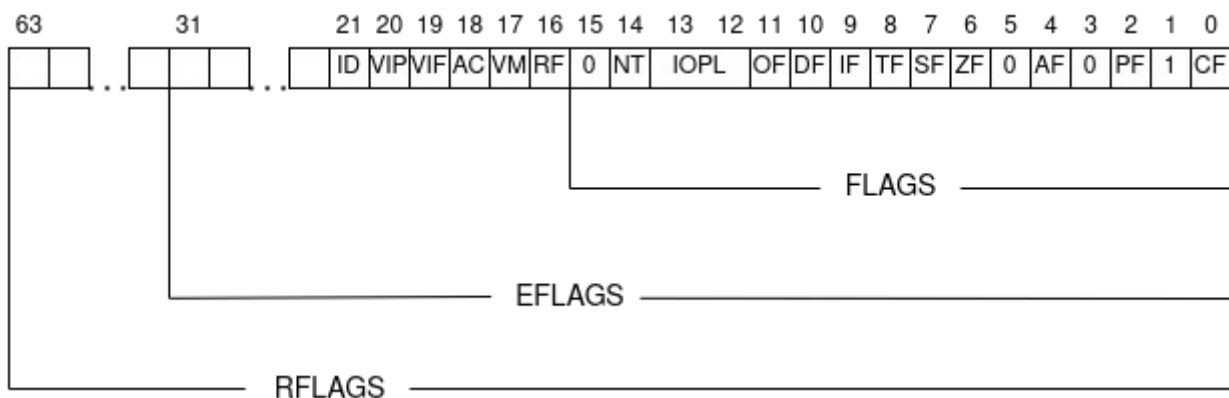
Rysunek 2.2.13. Rejestry segmentowe x86-64

Ważna kwestia dotycząca działania rejestrów segmentowych jest związana z wykorzystywanym modelem pamięci. Do dokładniejszego opisu i wizualizacji tych modeli przejdziemy w rozdziale **Pamięć Operacyjna**, tam też wyjaśnię wszystkie zawiłości związane z rejestrami segmentowymi.

Rejestr Flag

Rejestr flag przechowuje sporo informacji na temat ostatniej wykonanej operacji arytmetycznej. Poszczególne flagi mają rozmiar jednego bita, analogicznie, mogą przyjąć wartość **0** lub **1**. Cały rejestr flag jest przedstawiony na rysunku 2.2.14. Rejestr RFLAGS jest dostępny tylko w trybie 64-bitowym.

Patrząc na rysunek 2.2.14. możesz zastanawiać się czym są najmłodsze bity: 15, 5, 3 i 1. Są to pewne archaizmy z procesorów pentium, obecnie te flagi stanowią tylko numer w rejestrze, nie są dłużej wykorzystywane. Są tam tylko przez wzgląd na kompatybilność wsteczną.



Rysunek 2.2.14. Rejestr flag x86-64

Opiszę tutaj i przedstawię na przykładzie najczęściej używane flagi, czyli: **ZF**, **CF**, **SF**, **PF**, **OF**. Wszystkie przedstawione na zrzutach ekranów przykłady pochodzą z debuggera IDA.

ZF (Zero Flag) Flaga Zerowa – jest ustawiana, gdy wynik ostatniej operacji arytmetycznej lub logicznej, jest równy zero (rysunek 2.2.15.).

```
.text:0000000000401000
.text:0000000000401000 ; Attributes: no
.text:0000000000401000
.text:0000000000401000 public _start
.text:0000000000401000 _start proc near
.text:0000000000401000 mov ah, 8
.text:0000000000401002 sub ah, 8
.text:0000000000401005
.text:0000000000401005 exit:
.text:0000000000401005 mov eax, 3Ch
.text:000000000040100A mov edi, 0
.text:000000000040100F
```

RBP	00000	IOPL	0
RSP	00007	OF	0
RIP	00000	DF	0
R8	00000	IF	1
R9	00000	TF	0
R10	00000	SF	0
R11	00000	ZF	1
R12	00000	AF	0
R13	00000	PF	1
R14	00000	CF	0

Rysunek 2.2.15. Przykład Zero Flag

CF (Carry Flag) Flaga Przeniesienia – jest ustawiana, gdy w wyniku ostatniej operacji arytmetycznej pojawiło się przeniesienie na najbardziej znaczący bit lub też pożyczanie z bitu wyniku operacji. Czyli **CF**, wskazuje na to, że wynik operacji zawiera się w większej, niż dostępna liczba bitów w rejestrze.

Scenariusz 1: Przeniesienie najbardziej znaczącego bitu (rysunek 2.2.16).

Przykład (dla 8-bitowego rejestru AL):

$$11111111 + 00000001 = 00000000$$

.text:0000000000401000	RIP 00000	IF 1
.text:0000000000401000 ; Attributes: noret	R8 00000	TF 0
.text:0000000000401000	R9 00000	SF 0
.text:0000000000401000 public _start	R10 00000	ZF 1
.text:0000000000401000 _start proc near	R11 00000	AF 1
.text:0000000000401000 mov al, 0FFh	R12 00000	PF 1
.text:0000000000401002 add al, 1	R13 00000	CF 1
.text:0000000000401004	R14 00000	
.text:0000000000401004 exit:	R15 00000	
.text:0000000000401004 mov eax, 3Ch ;	EFL 00000	
.text:0000000000401009 mov edi, 0		
.text:000000000040100E syscall		

Rysunek 2.2.16. Przykład scenariusz 1 Carry Flag

Scenariusz 2: Pożyczenie najbardziej znaczącego bitu przy odejmowaniu dwóch liczb (rysunek 2.2.17).

Przykład (dla 8-bitowego rejestru AL):

$$0000\ 0000 - 0000\ 0001 = 1111$$

.text:0000000000401000	RIP 00000	IF 1
.text:0000000000401000	R8 00000	TF 0
.text:0000000000401000 ; Attributes: noret	R9 00000	SF 1
.text:0000000000401000	R10 00000	ZF 0
.text:0000000000401000 public _start	R11 00000	AF 1
.text:0000000000401000 _start proc near	R12 00000	PF 1
.text:0000000000401000 mov al, 0	R13 00000	CF 1
.text:0000000000401002 sub al, 1	R14 00000	
.text:0000000000401004	R15 00000	
.text:0000000000401004 exit:	EFL 00000	
.text:0000000000401004 mov eax, 3Ch ;		
.text:0000000000401009 mov edi, 0		
.text:000000000040100E syscall		

Rysunek 2.2.17. Przykład scenariusz 2 Carry Flag

SF (Sign Flag) Flaga Znaku – jest ustawiana względem najstarszego bitu (bitu znaku). Czyli jeśli wynik ostatniej operacji arytmetycznej jest ujemny (rysunek 2.2.18).

```

.text:000000000401000 RBP 00000 IOPL 0
.text:000000000401000 ; Attributes: none RSP 00007 OF 0
.text:000000000401000 RIP 00000 DF 0
.text:000000000401000 public _start R8 00000 TF 0
.text:000000000401000 _start proc near R9 00000 SF 1
.text:000000000401000 mov al, 64h ; R10 00000 ZF 0
.text:000000000401002 sub al, 65h ; R11 00000 AF 1
.text:000000000401004 exit: R12 00000 PF 1
.text:000000000401004 mov eax, 3Ch ; R13 00000 CF 1
.text:000000000401009 mov edi, 0 R14 00000
.text:00000000040100E svscall R15 00000

```

Rysunek 2.2.18. Przykład Sign Flag

PF (Parity Flag) Flaga Parzystości – jest ustawiana jeżeli najmniej znaczący bajt wyniku zawiera parzystą liczbę bitów (rysunek 2.2.19).

Przykład (dla 8-bitowego rejestru AL):

$$94_{16} - 4_{16} = 90_{16}$$

$$1001\ 0100 - 0100 = 1001\ 0000$$

```

.text:000000000401000 RSP 00007 OF 0
.text:000000000401000 ; Attributes: none RIP 00000 DF 0
.text:000000000401000 RIP 00000 IF 1
.text:000000000401000 public _start R8 00000 TF 0
.text:000000000401000 _start proc near R9 00000 SF 1
.text:000000000401000 mov al, 94h R10 00000 ZF 0
.text:000000000401002 sub al, 4 R11 00000 AF 0
.text:000000000401004 exit: R12 00000 PF 1
.text:000000000401004 mov eax, 3Ch ; R13 00000 CF 0
.text:000000000401009 mov edi, 0 R14 00000
.text:00000000040100E svscall R15 00000

```

Rysunek 2.2.19. Przykład Parity Flag

OF (Overflow Flag) Flaga Przepelnienia – jest ustawiana w wyniku przepelnienia, gdy wynik ostatniej operacji, jest **zbyt duzą** liczbą dodatnią lub **zbyt małą** liczbą ujemną, aby zmieścić ją w rejestrze docelowym (rysunek 2.2.20).

Przykład (dla 8-bitowego rejestru AL):

$$0100\ 0000 + 0100\ 0000 = 1000\ 0000$$

$$80_{16} + 80_{16} = 0$$

$$1000\ 0000 + 1000\ 0000 = 0000\ 0000$$

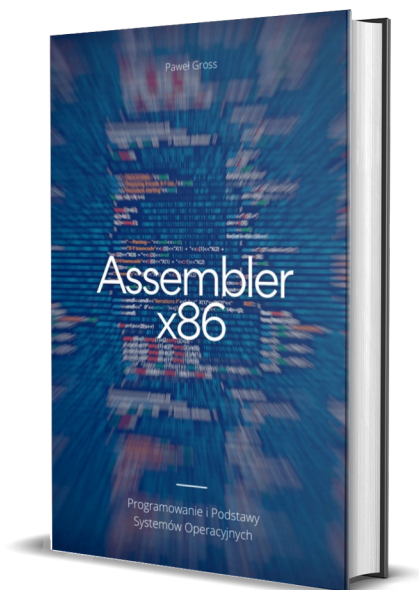
```
.text:0000000000401000  
.text:0000000000401000  
.text:0000000000401000 ; Attributes: none  
.text:0000000000401000 public _start  
.text:0000000000401000 _start proc near  
.text:0000000000401000 mov     al, 80h  
.text:0000000000401002 add     al, 80h  
.text:0000000000401004 exit:  
.text:0000000000401004 mov     eax, 3Ch  
.text:0000000000401009 mov     edi, 0  
.text:000000000040100E syscall
```

RDX	00000	AC	0
VM			0
RSI	00000	RF	0
RDI	00000	NT	0
RBP	00000	IOPL	0
RSP	00007	OF	1
RIP	00000	DF	0
R8	00000	IF	1
TF			0
R9	00000	SF	0
ZF			1
R10	00000	AF	0
R11	00000		

Rysunek 2.2.20. Przykład Overflow Flag

Pozostają do omówienia skoki warunkowe, ale do nich przejdziemy w rozdziale **Podstawowe instrukcje**.

Jeśli masz jakieś pytania, to pisz śmiało na naszym [serwerze discord](https://discord.gg/XFazfUXvCx) <https://discord.gg/XFazfUXvCx>.



Książka - Assembler x86 - Programowanie i Podstawy Systemów Operacyjnych

Wprowadzenie do brutalnego bitowego świata.

~~69.00zł~~

39.00zł

ZAMÓW

<https://assembler.techniczniej.pl/>