Ćwiczenie

„Instrukcje arytmetyczne i logiczne.

Przesuwanie i rotacja bitów”

Tematy ćwiczenia

- operacje arytmetyczne,
- instrukcje logiczne,
- przesuwanie i rotacja bitów

Sprawozdanie

Na każdym ćwiczeniu sporządza się sprawozdanie na bazie materiałów ćwiczenia. Bazowa zawartość sprawozdania musi być przygotowana w domu przed ćwiczeniem (sprawozdanie do ćwiczenia pierwszego jest przygotowywane w czasie ćwiczenia). W czasie ćwiczenia do sprawozdania są dodawane wyniki testowania.

Treść sprawozdania:

strona tytułowa,

spis treści sporządzony za pomocą *Word'*a,

dla każdego punktu rozdziały "Zadanie ", "Opracowanie zadania" (rozdział z tekstem programu i komentarzami), "Testowanie" (rozdział z opisem danych wejściowych i wynikami testowania, w tym zrzuty aktywnego okna).

Nazwa (bez polskich liter, żeby można było archiwizować) pliku ze sprawozdaniem musi zawierać nazwę przedmiotu, numer ćwiczenia i nazwisko studenta, na przykład "PN_...".

Pliki ze sprawozdaniem są przekazywane do archiwum grupy.

a) Operacje arytmetyczne

Zadanie

Napisać program, w którym obliczyć funkcję *y* od czterech argumentów całkowitych a, b, c, d według wzoru dla swojego zadania (patrz tabele wariantów)

Argumenty wprowadzać pojedynczo wyświetlając zachętę na wprowadzenie każdego argumentu.

Wyświetlić wynik obliczeń.

Opracowanie zadania

<<tekst programu>>

Testowanie

<<zrzut ekranu MS DOS po wykonaniu obliczeń>>

Tabela wariantów z operacjami arytmetycznymi

Np	Funkcja		Np	Funkcja		Np	Funkcja		Np	Funkcja
1	(a + b) * c / d		**9**	(a + b) * (c + d)		**17**	(a + b) * c – d		**25**	a + b * c - d
2	a –(b * c) / d		**10**	(a – b) * c + d		**18**	(a – b) * c / d		**26**	(a – b) * (c – d)
3	a * (b + c) / d		**11**	a * b * (c + d)		**19**	a *(b + c) - d		**27**	a * b * c – d
4	a / (b + c) * d		**12**	a / (b * c) + d		**20**	a / (b + c – d)		**28**	a / b * c - d
5	(a + b) – (c/ d)		**13**	a + b / c / d		**21**	a + (b – c) - d		**29**	a + b / c - d
6	(a - b * c) + d		**14**	(a – b) / c + d		**22**	a - b - c / d		**30**	a - b / c - d
7	a * (b – c) + d		**15**	a * b / c + d		**23**	a * (b - c – d)		**31**	a * b / c - d
8	a / b - c + d		**16**	a / b / (c + d)		**24**	a / (b - c)- d		**32**	a / b / c - d

b) Operacje logiczne

Zadanie

Napisać program, w którym obliczyć funkcję *y* od czterech argumentów logicznych a, b, c, d według wzoru dla swojego zadania (patrz tabele wariantów), na przykład:

y = (a | b) * c # d

Znaki operacji w tabelę wariantów:

* - mnożenie logiczne bitowe („AND”),
| - suma logiczna bitowa („OR”),
- suma symetryczna bitowa („XOR”),
~ - negacja logiczna bitowa („NOT”).

Argumenty wprowadzać pojedynczo wyświetlając zachętę na wprowadzenie każdego argumentu.
Wartości logiczne wprowadzać za pomocą liczb "0" i "1".
Wyświetlić wynik obliczeń.

Uwaga. W wyniku operacji NOT ma miejsce negacja we wszystkich bitach, w tym w starszym bicie, i liczba po negacji jest interpretowana jako ujemna.
Negacja bitowa liczby 0 daje w wyniku liczbę -1, a negacja bitowa liczby 1 daje w wyniku liczbę -2.
Jeśli po operacji NOT dodamy liczbę 2, to otrzymamy poprawny wynik:

```
mov   EAX, a ; logiczna zmienna a przypisana do rejestru EAX
not   EAX  ; negacja bitowa
add   EAX, 2 ; korekta wyniku negacji; w EAX negacja zmiennej a
```

Opracowanie zadania

<<tekst programu>>

Testowanie

<<zrzut ekranu MS DOS po wykonaniu obliczeń>>

Tabela wariantów z operacjami logicznymi

Np	Funkcja		Np	Funkcja		Np	Funkcja		Np	Funkcja
1	(a | b) * c # d		**9**	(a | b) * (c | d)		**17**	(a | b) * c * ~d		**25**	a | b * c * ~d
2	a * ~(b * c) # d		**10**	(a * ~b) * c | d		**18**	(a * ~b) * c # d		**26**	(a *~b) * (c *~d)
3	a * (b | c) # d		**11**	a * b * (c | d)		**19**	a *(b | c) * ~d		**27**	a * b * c * ~d
4	a # (b | c) * d		**12**	a # (b * c) | d		**20**	a # (b | c * ~d)		**28**	a # b * c * ~d
5	(a | b) * ~ (c# d)		**13**	a | b # c # d		**21**	a | (b * ~c) * ~d		**29**	a | b # c * ~d
6	(a * ~b * c) | d		**14**	(a * ~b) # c | d		**22**	a * ~b * ~c # d		**30**	a * ~b # c * ~d
7	a * (b * ~c) | d		**15**	a * b # c | d		**23**	a * (b * ~c * ~d)		**31**	a * b # c * ~d
8	a # b * ~c | d		**16**	a # b # (c | d)		**24**	a # (b * ~c)* ~d		**32**	a # b # c * ~d

Znaki operacji: * - mnożenie logiczne bitowe („AND”),
| - suma logiczna bitowa („OR”),
- suma symetryczna bitowa („XOR”),
~ - negacja logiczna bitowa („NOT”).

c) Przesuwanie i rotacja bitów

Zadanie

Napisać program, w którym wykonać operacje przesuwania lub rotacji bitów według swojego zadania (patrz tabele wariantów).

Liczbę testową 10100110001110000111100000111110b zapisać w sekcji danych korzystając z formatu liczb binarnych.

Wyświetlić liczbę testową oraz liczby po każdemu przesuwaniu.

Opracowanie zadania

<<tekst programu>>

Testowanie

<<zrzut ekranu MS DOS>>

Tabela wariantów z operacjami przesuwania i rotacji bitów

Np	Funkcja		Np	Funkcja
1	w lewo 4; w prawo 2		**17**	w lewo 8; w prawo 2
2	w lewo a4; w prawo 2		**18**	w lewo a8; w prawo 2
3	w lewo c4; w prawo 2		**19**	w lewo c8; w prawo 2
4	w lewo CFc4; w prawo 2		**20**	w lewo CFc8; w prawo 2
5	w prawo 4; w lewo 2		**21**	w prawo 8; w lewo 2
6	w prawo a4; w lewo 2		**22**	w prawo a8; w lewo 2
7	w prawo c4; w lewo 2		**23**	w prawo c8; w lewo 2
8	w prawo CFc4; w lewo 2		**24**	w prawo CFc8; w lewo 2
9	w lewo 4; w prawo a2		**25**	w lewo 8; w prawo a2
10	w lewo a4; w prawo a2		**26**	w lewo a8; w prawo a2
11	w lewo c4; w prawo a2		**27**	w lewo c8; w prawo a2
12	w lewo CFc4; w prawo a2		**28**	w lewo CFc8; w prawo a2
13	w prawo 4; w lewo a2		**29**	w prawo 8; w lewo a2
14	w prawo a4; w lewo a2		**30**	w prawo a8; w lewo a2
15	w prawo c4; w lewo a2		**31**	w prawo c8; w lewo a2
16	w prawo CFc4; w lewo a2		**32**	w prawo CFc8; w lewo a2

Oznaczenia: a – arytmetyczne, c – cykliczne; CFc – cykliczne przez znacznik CF, „liczba” to liczba bitów.

Wskazówki

Kodowanie rozkazów procesorów Intel jest przedstawione w tabelach przytoczonych niżej, gdzie oznaczono:
`A` – rejestr-akumulator EAX (AX, AH, AL),
`EA` – adres efektywny,
`Num` – numer portu,
`param8/16/32` – parametr 8-, 16- lub 32-bitowy,
`przes8/16/32` – przesunięcie 8-, 16- lub 32-bitowe,
`P` – pamięć,
`R` – rejestr,
`r8/16/32` – rejestr 8-, 16- lub 32-bitowy,
`R/P` – rejestr lub pamięć,
`Rs` – rejestr segmentowy.

Grupa `Arithmetic` (operacje arytmetyczne)

Wykonując rozkaz arytmetyczny (tab. 10) procesor operuje danymi w dwóch rejestrach, w tym w akumulatorze, lub w rejestrze i komórkę pamięci. Jako jeden z dwóch operandów może być zastosowana stała. Wynik operacji procesor może zapisać do rejestru lub do komórki pamięci. Dana może być 8-, 16- lub 32 bitowa.

Do sumowania są przeznaczone rozkazy „`add`” i „`adc`”, przy czym rozkaz „`adc`” uwzględnia przeniesienie, które jest przechowywane w znaczniku CF.

Rozkazy odejmowania „sub” i „sbb” są podobne strukturalnie do rozkazów sumowania „add” i „adc”. Rozkaz „sbb” uwzględnia pożyczkę, która jest przechowywana tak samo jak przeniesienie w znaczniku CF.

Przy wykonaniu rozkazów „inc” i „dec” zawartość rejestru lub komórki pamięci jest zwiększana lub zmniejszana odpowiednio o 1.

Rozkaz „cmp” służy do porównania dwóch wartości. Procesor odejmuje operandy, przy tym nie zapisuje wynik odejmowania, ale ustawia znaczniki operacji: AF, CF, OF, PF, SF i ZF.

Do zmiany znaku operandu stosuje się rozkaz „neg”.

Mnożenie operandów wykonują rozkazy „mul” i „imul”, przy czym rozkaz „mul” służy do mnożenia bez znaku, a rozkaz „imul” – do mnożenia ze znakiem.

Mnożenie „mul” dwóch liczb bez znaku przewiduje, że operand pierwszy jest domyślnym i znajduje się w rejestrze AL, AX lub EAX, a operand drugi - w rejestrze lub komórce pamięci. Wynik mnożenia jest zapisywany do rejestru AX, jeśli mnożna i mnożnik 8-bitowe, do rejestrów DX:AX, jeśli mnożna i mnożnik 16-bitowe, do rejestrów EDX:EAX, jeśli mnożna i mnożnik 32-bitowe.

Mnożenie liczb ze znakiem przez rozkaz „imul” ma wiele wariantów.

W wariancie, gdy w kodzie operacji znajduje się jeden operand, pierwszy operand operacji jest domyślnym i mieści się w rejestrze AL, AX lub EAX, a operand drugi - w rejestrze lub komórce pamięci. Podobnie do rozkazu „mul” wynik mnożenia jest zapisywany do rejestru AX, jeśli mnożna i mnożnik 8-bitowe, do rejestrów DX:AX, jeśli mnożna i mnożnik 16-bitowe, do rejestrów EDX:EAX, jeśli mnożna i mnożnik 32-bitowe.

Jeśli w kodzie operacji znajdują się dwa operandy, to operand pierwszy musi być w rejestrze, operand drugi - w rejestrze, komórce pamięci lub może być stałą, a wynik jest umieszczany w rejestrze.

W przypadku rozkazu „imul” z trzema argumentami, pierwszy z argumentów wskazuje na miejsce rozmieszczenia wyniku, argument drugi jest pierwszym operandem operacji i może być w rejestrze lub komórce pamięci, a argument trzeci jest stała (drugim operandem operacji).

Dzielenie liczb bez znaku wykonuje rozkaz „div”. Wskazany w rozkazie dzielnik może znajdować się w rejestrze lub w komórce pamięci. Dzielna musi być przygotowana w rejestrze AX, jeśli dzielnik 8-bitowy, w rejestrach DX:AX, jeśli dzielnik 16-bitowy, oraz w rejestrach EDX:EAX, jeśli dzielnik 32-bitowy. Po wykonaniu rozkazu iloraz znajduje się w rejestrach AL, AX lub EAX w zależności od rozmiaru dzielnika, a reszta odpowiednio w rejestrach AH, DX lub EDX.

Tabela 10

Kodowanie rozkazów grupy Arithmetic (operacje arytmetyczne)

Mnemonik i operandy	Bajt 0	Bajt 1	Od bajta 2

add R/P, R
add R, R/P
add R/P, dana
add A, dana
	0000000w
	0000001w
	100000sw
	0000010w
		mod_reg_r/m
		mod_reg_r/m
		mod_000_r/m
		dana
			
			dana

adc R/P, R adc R, R/P adc R/P, dana adc A, dana	0001000w 0001001w 100000sw 0001010w	mod_reg_r/m mod_reg_r/m mod_010_r/m dana	 dana

sub R/P, R
sub R, R/P
sub R/P, dana
sub A, dana
	0010100w
	0010101w
	100000sw
	0010110w
		mod_reg_r/m
		mod_reg_r/m
		mod_101_r/m
		dana
			
			dana

sbb R/P, R sbb R, R/P sbb R/P, dana sbb A, dana	0001100w 0001101w 100000sw 0001110w	mod_reg_r/m mod_reg_r/m mod_011_r/m dana	 dana
inc R/P inc R	1111111w 01000reg	mod_000_r/m	
dec R/P dec R	1111111w 01001reg	mod_001_r/m	

cmp R/P, R
cmp R, R/P
cmp R/P, dana
cmp A, dana
	0011100w
	0011101w
	100000sw
	0011110w
		mod_reg_r/m
		mod_reg_r/m
		mod_111_r/m
		dana
			
			dana

neg R/P	1111011w	mod_011_r/m	
mul R/P	1111011w	mod_100_r/m	
imul R/P imul R, R/P imul R, R/P, dana	1111011w 00001111 011010s1	mod_101_r/m 10101111 mod_reg_r/m	 mod_reg_r/m dana
div R/P	1111011w	mod_110_r/m	
idiv R/P	1111011w	mod_111_r/m	

aaa	00110111		
aas	00111111		
aad	11010101	00001010	
aam	11010100	00001010	
daa	00100111		
das	00101111		
cbw	10011000		
cwde	10011000		
cwd	10011001		
cdq	10011001		

Rozkaz „idiv” dzieli liczby ze znakiem. W kodzie rozkazu jest wskazywany dzielnik. Dzielna musi być przygotowana w rejestrze AX, jeśli dzielnik 8-bitowy, w rejestrach DX:AX, jeśli dzielnik 16-bitowy, oraz w rejestrach EDX:EAX, jeśli dzielnik 32-bitowy. Po wykonaniu rozkazu iloraz znajduje się w rejestrach AL, AX lub EAX w zależności od rozmiaru dzielnika, a reszta odpowiednio w rejestrach AH, DX lub EDX.

Grupa rozkazów „aaa”, „aas”, „aad” i „aam” jest potrzebna do korekty wyników operacji w rozpakowanym kodzie BCD (Binary-Coded Decimal [Interchange Code]).

Procesory Intel zorientowane są na przetwarzanie metodą „cyfra za cyfrą” dziesiętnych liczb w kodzie BCD.

Pojedyncza cyfra liczby może być zapisana w kodzie dwójkowym w odrębnym bajcie lub w 4-bitowej połówce bajta. Zapis jednej dziesiętnej liczby w kodzie dwójkowym w bajcie nazywamy *rozpakowanym kodem BCD*. W tym przypadku wykorzystujemy tylko 10 kombinacji kodu dwójkowego spośród 256.

Zapis dwóch dziesiętnych liczb w kodzie dwójkowym w 4-bitowych połówkach bajta nazywamy *spakowanym kodem BCD*. W tym przypadku wykorzystujemy 100 kombinacji kodu dwójkowego spośród 256.

Rozkaz „aaa” wprowadza korektę do wyniku sumowania w rejestrze AL dwóch cyfr rozpakowanego kodu BCD. Suma w rejestrze AL, która nie może być większą niż 18, jest rozdzielana przez rozkaz „aaa” na dwie cyfry dwójkowo-dziesiętne zapisywane do rejestrów AH i AL.

Rozkaz „aas” służy do przekształcenia wyniku w AL po operacji odejmowania cyfr dwójkowo-dziesiętnych rozpakowanego kodu BCD. Jeżeli wynik odejmowania w rejestrze AL jest ujemny, to ten wynik jest zapisany w kodzie uzupełniającym do 256. Po korekcie w rejestrze AL znajduje się wynik w kodzie BCD uzupełniającym do 10, a w rejestrze AH – znak (00h dla liczby dodatniej, FFh dla liczby ujemnej). Przed korektą rejestr AH musi być wyzerowany.

Po operacji mnożenia cyfr dwójkowo-dziesiętnych rozkaz „aam” przekształca wynik mnożenia w AL. Liczba dwójkowa w AL jest rozdzielana na dwie cyfry dwójkowo-dziesiętne zapisywane do rejestrów AH i AL. Chociaż przy mnożeniu dwóch dziesiętnych cyfr nie może powstać wynik większy niż 81, rozkaz „aam” poprawnie działa do wartości 99.

Rozkaz „aad” przekształca dzielną w AX przed operacją dzielenia liczb dwójkowo-dziesiętnych. Dwie cyfry dwójkowo-dziesiętne dzielnej powinny być zapisane do rejestrów AH i AL Wynik przekształcenia zapisuje się do rejestru AL.

Korekcja liczb przy dodawaniu lub odejmowaniu w spakowanym kodzie BCD jest prowadzona za pomocą rozkazów „daa” i „das”.

Rozkaz „daa” poprawia sumę w AL przy wykonaniu operacji w spakowanym kodzie BCD. Przeniesienie jest zapisywane do znacznika CF.

Rozkaz „das” służy do korekcji w rejestrze AL wyniku operacji odejmowania liczb w spakowanym kodzie BCD. Jeżeli wynik odejmowania jest ujemny, to każda liczba dwójkowo-dziesiętna jest zapisywana w kodzie uzupełniającym do 10. Pożyczka jest zapisywana do znacznika CF.

Rozkazy „cbw”, „cwd”, „cwde” i „cdq” rozszerzają znak operandu z rejestru-akumulatora:

- „cbw” rozszerza znak bajta w AL na słowo w AX,
- „cwd” rozszerza znak słowa w AX na słowo w DX,
- „cwde” rozszerza znak słowa w AX na słowo w EAX,
- „cdq” rozszerza znak słowa w EAX na słowo w EDX.

Rozszerzenie znaku jest realizowane przez kopiowanie bitu znakowego do starszej części słowa lub do wszystkich bitów rejestru DX (EDX). Na przykład w wyniku wykonania rozkazu „cbw” bit znakowy rejestru AL będzie skopiowany do wszystkich bitów rejestru AH.

Grupa `Logic` (operacje logiczne bitowe)

Operacje logiczne są realizowane procesorami Intel tylko jako operacje bitowe. Rozkazy „`and`”, „`or`”, „`xor`” i „`not`” (tab. 11) powodują wykonanie operacji mnożenia logicznego bitowego, sumy logicznej bitowej, sumy symetrycznej bitowej i negacji logicznej bitowej odpowiednio.

Tabela 11

Kodowanie rozkazów grupy `Logic` (operacje logiczne bitowe)

Mnemonik i operandy	Bajt 0	Bajt 1	Od bajta 2
and R/P, R and R, R/P and R/P, dana and dana	0010000w 0010001w 100000sw 0010010w	mod_reg_r/m mod_reg_r/m mod_100_r/m dana	 dana
test R/P, R; test R, R/P test R/P, dana test dana	1000010w 100000sw 1010100w	mod_reg_r/m mod_000_r/m dana	 dana
or R/P, R or R, R/P or R/P, dana or dana	0000100w 0000101w 100000sw 0000110w	mod_reg_r/m mod_reg_r/m mod_001_r/m dana	 dana
xor R/P, R xor R, R/P xor R/P, dana xor dana	0011000w 0011001w 100000sw 0011010w	mod_reg_r/m mod_reg_r/m mod_110_r/m dana	 dana
not R/P	1111011w	mod_010_r/m	

Rozkaz „`test`” jest wykonywany jako mnożenie logiczne bitowe z tej różnicą, że wynik nie jest zapisywany, a tylko są ustawiane znaczniki operacji.

Grupa Shift/Rotate (przesuwanie bitowe)

Procesory Intel mają liczną grupę rozkazów (tab. 12) do przesuwania i rotacji bitów. Operand może być w rejestrze lub komórce pamięci. Parametr wskazuje na liczbę przesunięć i może znajdować się w rejestrze CL lub mieć wartość stałą.

Rozkazy „rcl" i „rcr" wywołują przesuwanie bitowe cykliczne w lewo (w stronę starszych bitów) lub w prawo (w stronę młodszych bitów) przez znacznik CF.

Rozkazy „rol" i „ror" też wywołują przesuwanie bitowe cykliczne w lewo lub w prawo, ale nie przez znacznik CF. Cykl jest zamykany między starszym i młodszym bitami. Do znacznika CF jest zapisywany aktualny starszy bit w przypadku przesuwania w lewo (rozkaz „rol") lub młodszy bit w przypadku przesuwania w prawo (rozkaz „ror").

Tabela 12

Kodowanie rozkazów grupy Shift/Rotate (przesuwanie bitowe)

Mnemonik i operandy	Bajt 0	Bajt 1	Bajt 2	Od bajta 3
rcl R/P, 1 rcl R/P, CL rcl R/P, param8	1101000w 1101001w 1100000w	mod_010_r/m mod_010_r/m mod_010_r/m	 param8	
rcr R/P, 1 rcr R/P, CL rcr R/P, param8	1101000w 1101001w 1100000w	mod_011_r/m mod_011_r/m mod_011_r/m	 param8	
rol R/P, 1 rol R/P, CL rol R/P, param8	1101000w 1101001w 1100000w	mod_000_r/m mod_000_r/m mod_000_r/m	 param8	
ror R/P, 1 ror R/P, CL ror R/P, param8	1101000w 1101001w 1100000w	mod_001_r/m mod_001_r/m mod_001_r/m	 param8	
sal R/P, 1 sal R/P, CL sal R/P, param8	1101000w 1101001w 1100000w	mod_100_r/m mod_100_r/m mod_100_r/m	 param8	
sar R/P, 1 sar R/P, CL sar R/P, param8	1101000w 1101001w 1100000w	mod_111_r/m mod_111_r/m mod_111_r/m	 param8	
shl R/P, 1 shl R/P, CL shl R/P, param8	1101000w 1101001w 1100000w	mod_100_r/m mod_100_r/m mod_100_r/m	 param8	
shr R/P, 1 shr R/P, CL shr R/P, param8	1101000w 1101001w 1100000w	mod_101_r/m mod_101_r/m mod_101_r/m	 param8	
shld R/P, R, CL shld R/P, R, param8	00001111 00001111	10100101 10100100	mod_reg_r/m mod_reg_r/m	 param8
shrd R/P, R, CL shrd R/P, R, param8	00001111 00001111	10101101 10101100	mod_reg_r/m mod_reg_r/m	 param8

W procesie wykonania rozkazu „sar" ma miejsce przesuwanie bitowe arytmetyczne w prawo (w stronę młodszych bitów). Przesuwanie bitów jest nazywane *arytmetycznym*, jeśli bit znakowy (starszy bit) nie zmienia wartości. Przy przesuwaniu w prawo (rozkaz „sar") do znacznika CF jest przekazywany młodszy bit operandu. Przesuwanie arytmetyczne w prawo o jeden bit jest ekwiwalentne dzieleniu na 2.

W przypadku rozkazu „sal" ma miejsce przesuwanie bitowe w lewo nie arytmetyczne, a logiczne, tj. bez zachowania znaku. W przypadku przesuwania w lewo (rozkaz „sal") z prawej strony wsuwa się 0, a kopia starszego bitu jest zapisywana do znacznika CF.

Rozkazy „shl" i „shr" powodują przesuwanie bitowe logiczne w lewo (w stronę starszych bitów) lub w prawo (w stronę młodszych bitów). W przypadku rozkazu „shl" starszy bit wysuwa się w znacznik CF, a na miejsce młodszego bitu z prawej strony wsuwa się 0. Dla rozkazu „shr" do znacznika CF jest wysuwany młodszy bit, a z lewej strony jest wsuwane 0.

Rozkazy „shld" i „shrd" mają dodatkowy operand – rejestr, który można rozpatrywać jako łańcuch wsuwanych bitów, ponieważ dodatkowy operand – rejestr też jest przesuwany. Przy przesuwaniu w lewo (rozkaz „shld") starszy bit dodatkowego operandu jest wsuwany do młodszego bitu operandu bazowego. Podobnie temu przy przesuwaniu w prawo (rozkaz „shrd") młodszy bit dodatkowego operandu jest wsuwany do starszego bitu operandu bazowego. Dla rozkazów „shld" i „shrd" wartość parametru przesunięć jest ograniczona liczbą 31. Wysuwany bit jest zapisywany do znacznika CF.

Przesuwania bitowe logiczne

Logiczne przesuwania na wskazaną ilość bitów są wykonywane na 8-, 16- i 32-bitowych rejestrach. Na miejsce zwolnione jest wpisywane 0.

Przesuwanie bitowe logiczne w lewo (w stronę starszych bitów) powoduje instrukcja `SHL`, a w prawo (w stronę młodszych bitów) - instrukcja `SHR`.

SHL	**Operand**																**CF**
Do operacji	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0
Po operacji	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0	1
SHR	**Operand**																**CF**
Do operacji	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0
Po operacji	0	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0

Niezależnie od rodzaju instrukcji bit wysuwany trafia do znacznika CF, a na miejsce zwolnione jest wpisywana wartość 0. Ilość przesunięć jest wskazywana przez operand drugi.

Instrukcje można użyć do mnożenia lub dzielenia liczb całkowitych. Przesuwanie o jeden bit w prawo oznacza dzielenie na 2. Przesuwanie o jeden bit w lewo oznacza mnożenie razy 2.

Poniżej przedstawiono kilka przykładów użycia instrukcji `SHL` i `SHR` do przesunięć bitowych:

```
.286
mov AL, 00110011b
shl AL, 2 ;przesuwanie w lewo o dwa bity, tj. mnożenie razy 4
;AL = 11001100b
mov CL, 4 ;przygotowanie liczby przesunięć
shr AL, CL     ;przesuwanie w prawo o 4 bity, tj. dzielenie na 16
;AL ==> 00001100b
```

Za pomocą przesuwania o jeden bit w prawo a później w lewo można przekształcić liczbę na liczbę parzystą, na przykład:

```
mov AL, 00110011b        ;tj. AL = 51
shr AL, 1 ;przesuwanie w prawo o jeden bit
;AL = 00011001b, tj. AL = 25
shl AL, 1 ;przesuwanie w lewo o jeden bit
;AL = 00110010b, tj. AL = 50
```

Instrukcja SHL może być użyta do mnożenia liczby o 10, ponieważ iloczyn można obliczać według wzoru:

```
y = x *10 = x * 8 + x * 2
```

Przykładowy program wygląda następująco:

```
.DATA
x          DW    0
y          DW    0
.CODE
...
mov  AX, x          ;operacja AX = x
```

```
mov  CL, 3          ;przygotowanie liczby przesunięć równej 3
shl  AX, CL    ;mnożenie razy 8
mov  BX, AX    ;w BX x*8
mov  AX, x          ;operacja AX = x
shl  AX, 1          ;mnożenie x o 2
add  AX, BX    ;suma x * 8 + x * 2
mov  y,  AX    ;wynik
...
```

Kopiowanie przesuwanego bitu do znacznika CF można wykorzystać do analizowania zawartości zmiennej. W następnym przykładzie jest pokazane, w jaki sposób wyświetlić na ekranie wartość zmiennej w binarnym formacie:

```
.DATA
x          DW   0
.CODE
...
 mov AX, x          ;przygotowanie cyklu
 mov CX, 16    ;przygotowanie cyklu (16 = ilość bitów)
cykl:
 shl AX, 1          ;przesuwanie w lewo, w CF zawsze starszy bit
 jc     bit_1       ;skok jeśli CF==1
 mov DL, '0'   ;operacja DL = '0'
 jmp dalej          ;skok bezwarunkowy
bit_1:
 mov DL, '1'   ;operacja DL = '1'
 jmp dalej          ;skok bezwarunkowy
dalej:
 push     AX        ; AX na stos
 mov AX, 0200h ;funkcja INT21h/2
 int 21h       ;wyświetlenie zawartości DL
 pop AX        ;przywrócenie AX
 loop     cykl      ;zmniejszenie CX o 1 i skok jeśli CX > 0
```

Przesuwania bitowe ze wstawianiem

Instrukcje `SHLD` i `SHRD` (wprowadzone najpierw dla procesorów Intel386) są bardzo mocne. Instrukcje zawierają trzy operandy.

Liczbę przesunięć określa operand trzeci, a operand drugi wskazuje na źródło do zapełnienia zwolnionego bitu.

Ponieważ dodatkowy operand – rejestr też jest przesuwany, to przy przesuwaniu w lewo (rozkaz „`shld`") starszy bit dodatkowego operandu jest przesuwany na miejsce młodszego bitu operandu bazowego.

Podobnie przy przesuwaniu w prawo (rozkaz „`shrd`") młodszy bit dodatkowego operandu jest przesuwany na miejsce starszego bitu operandu bazowego.

Dla rozkazów „`shld`" i „`shrd`" wartość parametru przesunięć jest ograniczona liczbą 31.

Wysuwany bit jest zapisywany do znacznika CF.

SHLD	**Operand 1**																**Operand 2**																**CF**
Do operacji	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0
Po operacji	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0	1
SHRD	**Operand 1**																**Operand 2**																**CF**
Do operacji	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0
Po operacji	0	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0

Poniżej przedstawiono przykład użycia instrukcji SHLD i SHRD do przesunięć bitowych:

```
.386
.DATA
zmienna DW 1234h
zmienna2 DD 0FEDCBA98h
.CODE
 mov AX, 5678h
 shld zmienna, AX, 4 ;przesuwanie w lewo o 4 bity i
;wstawienie cyfry 5,  zmienna = 2345h
 mov CL, 4     ;przygotowanie liczby przesunięć
 shrd zmienna, AX, CL ;przesuwanie w prawo o 4 bity i
; wstawienie cyfry 8, zmienna = 8234h
 mov EAX, 12345678h
 shld EAX, zmienna2, 1 ;przesuwanie w lewo o bit i
; wstawienie 1, EAX = 2468ACF1h
 mov CL, 1     ;przygotowanie liczby przesunięć
 shrd EAX, zmienna2, CL ;przesuwanie w prawo o bit i
; wstawienie cyfry 8, EAX = 12345678h
```

Przesuwania bitowe arytmetyczne

Przesuwanie bitowe arytmetyczne różni się od logicznego tym, że bit znakowy (bit starszy) nie zmienia wartości.

Przesuwanie arytmetyczne w prawo o jeden bit jest ekwiwalentne dzieleniu przez 2.

Instrukcja SAR to instrukcja do przesuwania bitowego arytmetycznego w prawo (w stronę młodszych bitów).

Starszy bit jako bit znakowy nie zmienia wartości. Bit wysuwany trafia do znacznika CF.

SAL	**Operand**																**CF**
Do operacji	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0
Po operacji	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0	1
SAR	**Operand**																**CF**
Do operacji	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0
Po operacji	1	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0

***Uwaga*. W procesorach Intel w przypadku rozkazu „`sal`” ma miejsce przesuwanie bitowe w lewo nie arytmetyczne, a logiczne, tj. bez zachowania znaku.**

W przypadku przesuwania w lewo (rozkaz „`sal`”) z prawej strony wsuwa się 0, a kopia starszego bitu jest zapisywana do znacznika CF.

Dla instrukcji SAL i SAR Ilość przesunięć jest wskazywana przez operand drugi: rejestr CL lub stałą.

Poniżej przedstawiono przykład użycia instrukcji SAL i SAR do przesunięć bitowych arytmetycznych:

```
mov AL, 00110011b   ; AL=51
sal AL, 1 ;przesuwanie w lewo o jeden bit (mnożenie o 2)
;AL = 01100110b, tj. 102
mov CL, 4 ;przygotowanie liczby przesunięć
sar AL, CL     ;przesuwanie w prawo o 4 bity (dzielenie na 16)
;AL = 00000110b, tj. 6
```

Przesuwania bitowe cykliczne

Przesuwania bitowe cykliczne często jest nazywane rotacją bitów. Rotację wykonują instrukcje ROL i ROR.

Instrukcja ROL przesuwa bity w lewo cyklicznie, tj. wysuwany starszy bit zapisuje się na zwolnione miejsce w młodszym bicie.

Znacznik CF zawiera kopię wysuwanego starszego bitu.

Instrukcja ROR przesuwa bity w prawo cyklicznie, tj. wysuwany młodszy bit zapisuje się na zwolnione miejsce w starszym bicie.

Znacznik CF zawiera kopię wysuwanego młodszego bitu.

ROL	**Operand**																**CF**
Do operacji	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0
Po operacji	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	1	1
ROR	**Operand**																**CF**
Do operacji	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0
Po operacji	0	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0

Poniżej przedstawiono przykład użycia instrukcji ROL i ROR do przesunięć bitowych cyklicznych:

```
mov AL, 01110011b
rol AL, 2 ;przesuwanie w lewo o dwa bity
;AL = 11001101b
ror AL, 4 ;przesuwanie w prawo o 4 bity
;AL = 11011100b
```

Instrukcje ROL lub ROR można użyć do zamiany miejscami polówek zmiennej, na przykład:

```
.386
.DATA
zmienna DD 0FEDCBA98h
.CODE
rol zmienna, 16 ;przesuwanie cykliczne w lewo o 16 bitów
; zmienna = 0BA98FEDCh
ror zmienna, 16 ;przesuwanie cykliczne w prawo o 16 bitów
; zmienna = 0FEDCBA98h
```

Przesuwanie bitowe cykliczne z zastosowaniem znacznika CF

Instrukcje RCL i RCR wykonują przesuwanie bitowe cykliczne z zastosowaniem znacznika CF.

Wykonanie instrukcji RCL można opisać następująco:
do bitu młodszego przytacza się z prawej strony bit znacznika CF; bity są przesuwane w lewo tak, że bit CF jest przyjmowany do młodszego bitu słowa; wysuwany bit starszy jest zapisywany do znacznika CF.

Instrukcja RCR jest wykonywana następująco:
bity są przesuwane w prawo; znacznik CF jest wsuwany do bitu starszego, a wysuwany bit młodszy jest zapisywany do znacznika CF.

RCL	**Operand**																**CF**
Do operacji	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0
Po operacji	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0	1
RCR	**Operand**																**CF**
Do operacji	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0
Po operacji	0	1	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0

Poniżej przedstawiono przykłady użycia instrukcji RCL i RCR do przesunięć bitowych cyklicznych z zastosowaniem znacznika CF:

```
mov AL, 01110011b
stc  ; ustawienie na 1 znacznika CF
rcl AL, 2 ;przesuwanie w lewo o dwa bity
;AL = 11001110b
clc  ; zerowanie znacznika CF
rcr AL, 4 ;przesuwanie w prawo o 4 bity
;AL = 11001100b
```

Instrukcje RCL i RCR są często używane do rozgałęzień w programie w zależności od zawartości bajta (słowa). Zawartość znacznika CF wpływa na wykonanie instrukcji JC (skok, jeśli CF=1) i JNC (skok, jeśli CF=0).

W następującym przykładzie jest realizowane rozgałęzienie na 4 kierunki według wartości dwóch młodszych bitów zmiennej `stan`:

```
.DATA
stan  DB    0
rozg  DB    0     ;lokalna tymczasowa zmienna
.CODE
 mov  AL, stan    ; kopiowanie zmiennej „stan”
 mov  rozg, AL    ; kopiowanie zmiennej „stan”
 rcr  rozg, 1     ; W CF rozg[0] == stan[0]
 jc   etyk1       ; skok jeśli CF==1
 rcr  rozg, 1     ; W CF rozg[0] == stan[1]
 jc   etyk01      ; skok jeśli CF==1
; gałąź „stan[1]==0 && stan[0]==0”
 ...
 jmp dalej
etyk01:
; gałąź „stan[1]==1 && stan[0]==0”
 ...
 jmp dalej
etyk1:
 rcr  rozg, 1     ; W CF rozg[0] == stan[1]
 jc   etyk11      ; skok jeśli CF==1
; gałąź „stan[1]==0 && stan[0]==1”
 ...
 jmp dalej
etyk11:
; gałąź „stan[1]==1 && stan[0]==1”
 ...
 jmp dalej
dalej:
 ...
```

Instrukcje RCL i RCR często są używane do przesuwania bitowego w tablicach, gdy bity są przesuwane na odległość większą niż 32. Potrzeba takiego przesuwania często powstaje przy programowaniu aplikacji graficznych.

W następującym przykładzie jest pokazane przesuwanie na NN bitów w tablicy o rozmiarze 768 bajtów:

```
.DATA
ROZM =    768
tabl DB   ROZM dup (1)
NN   DW   0    ;ilość przesunięć
.CODE
 mov NN, 80    ; przykładowa wartość NN
 mov ECX, NN   ;przygotowanie cyklu przesunięć
cykl1:
 mov EBX, OFFSET tabl    ;przygotowanie cyklu w tablicy
 mov EDI, ROZM-1     ; przygotowanie cyklu w tablicy,
; w DI jest zapisany indeks
 clc      ;przygotowanie cyklu w tablicy, CF=0
cykl2:
 mov AL, [EBX+EDI]  ;przesłanie następnego bajta
 rcl AL, 1          ;przesuwanie w lewo, w CF zawsze starszy bit
 mov [EBX+EDI], AL  ;zapisywanie następnego bajta
 dec EDI
 jnz cykl2          ;skok jeśli ZF == 0, tj. DI != 0
 loop     cykl1          ;zmniejszenie CX o 1 i skok jeśli CX > 0
```

Grupa Bit Manipulation

Grupa rozkazów `Bit Manipulation` (tab. 14) służy do sprawdzania i zmiany wartości pojedynczych bitów danej.

Wykonując rozkazy „`bsf`” (nazwa od ang. *bit scan forward*) i „`bsr`” (nazwa od ang. *bit scan reverse*) procesor skanuje operand – źródło do pierwszego ustawionego bitu i zapisuje do operandu – miejsca przeznaczenia indeks tego bitu. Jeżeli jest znaleziony bit w stanie „1”, to jest ustawiany znacznik ZF. Rozkaz „`bsf`” skanuje od młodszego bitu, a rozkaz „`bsr`” – od starszego.

Tabela 14

Kodowanie rozkazów grupy `Bit Manipulation` (operacje bitowe)

Instrukcja	Bajt 0	Bajt 1	Bajt 2	Od bajta 3
`bsf R, R/P`	`00001111`	`10111100`	`mod_reg_r/m`	
`bsr R, R/P`	`00001111`	`10111101`	`mod_reg_r/m`	
`bt R/P, R` `bt R/P, dana`	`00001111` `00001111`	`10100011` `10111010`	`mod_reg_r/m` `mod_100_r/m`	 `dana8`
`btc R/P, R` `btc R/P, dana`	`00001111` `00001111`	`10111011` `10111010`	`mod_reg_r/m` `mod_111_r/m`	 `dana8`
`btr R/P, R` `btr R/P, dana`	`00001111` `00001111`	`10110011` `10111010`	`mod_reg_r/m` `mod_110_r/m`	 `dana8`
`bts R/P, R` `bts R/P, dana`	`00001111` `00001111`	`10101011` `10111010`	`mod_reg_r/m` `mod_101_r/m`	 `dana8`

Operand drugi rozkazu „`bt`” jest indeksem tego bitu w operandzie pierwszym, który jest kopiowany do znacznika CF.

Rozkaz „`btc`” jest wykonywany podobnie rozkazu „`bt`”, tylko z tą różnicą, że do znacznika CF jest kopiowana negacja bitu wskazanego przez indeks.

Przy wykonaniu rozkazów „`btr`” i „`bts`” drugi operand też jest indeksem bitu w operandzie pierwszym, a bit jest kopiowany do znacznika CF. Po zakończeniu kopiowania bit jest zerowany (rozkaz „`btr`”) lub ustawiany (rozkaz „`bts`”).

Grupa Conditional Byte Set

Rozkazy grupy `Conditional Byte Set` (tab. 17) przenoszą stany znaczników operacji do rejestru bajtowego lub do jednobajtowej zmiennej programu.

Tabela 17

Kodowanie rozkazów grupy `Conditional Byte Set` (warunkowe ustawienie bajta)

Instrukcja	Bajt 0	Bajt 1	Bajt 2	Bajt 3
`seto R/P`	`00001111`	`10010000`	`mod_000_r/m`	
`setno R/P`	`00001111`	`10010001`	`mod_000_r/m`	
`setb (setnae) R/P`	`00001111`	`10010010`	`mod_000_r/m`	
`setnb R/P`	`00001111`	`10010011`	`mod_000_r/m`	
`sete (setz) R/P`	`00001111`	`10010100`	`mod_000_r/m`	
`setne (setnz) R/P`	`00001111`	`10010101`	`mod_000_r/m`	
`sets R/P`	`00001111`	`10011000`	`mod_000_r/m`	
`setns R/P`	`00001111`	`10011001`	`mod_000_r/m`	
`setp (setpe) R/P`	`00001111`	`10011010`	`mod_000_r/m`	
`setnp (setpo) R/P`	`00001111`	`10011011`	`mod_000_r/m`	
`setbe (setna) R/P`	`00001111`	`10010110`	`mod_000_r/m`	
`setnbe (seta) R/P`	`00001111`	`10010111`	`mod_000_r/m`	
`setl (setnge) R/P`	`00001111`	`10011100`	`mod_000_r/m`	
`setnl (setge) R/P`	`00001111`	`10011101`	`mod_000_r/m`	
`setle (setng) R/P`	`00001111`	`10011110`	`mod_000_r/m`	
`setnle (setg) R/P`	`00001111`	`10011111`	`mod_000_r/m`	

Każdy z rozkazów „`seto`”, „`setb (setnae)`”, „`sete (setz)`”, „`sets`” i „`setp (setpe)`” kopiuję do operandu znacznik, a każdy z rozkazów „`setno`”, „`setnb`”, „`setne (setnz)`”, „`setns`” i „`setnp (setpo)`” kopiuję znacznik z negacją.

Kopiowane znaczniki:

- OF (znacznik nadmiaru) w rozkazach „`seto`” i „`setno`”,
- CF (znacznik przeniesienia) w rozkazach „`setb (setnae)`” i „`setnb`”,
- ZF (znacznik zera) w rozkazach „`sete (setz)`” i „`setne (setnz)`”,
- SF (znacznik znaku) w rozkazach „`sets`” i „`setns`”,
- PF (znacznik parzystości) w rozkazach „`setp (setpe)`” i „`setnp (setpo)`”.

Rozkazy „`setbe (setna)`”, „`setl (setnge)`” i „`setle (setna)`” analizują po dwa znaczniki i zapisują do operandu 1, jeśli warunek jest prawdziwy, i 0, jeśli warunek nie jest prawdziwy.

„Inwersyjne” rozkazy „`setnbe (seta)`”, „`setnl (setge)`” i „`setnle (seta)`” zapisują do operandu 0, jeśli warunek jest prawdziwy, i 1, jeśli warunek nie jest prawdziwy.

Procesor oblicza następujące warunki:

- (CF=1 lub ZF=1) w rozkazach „`setbe (setna)`” i „`setnbe (seta)`”,
- SF<>OF w rozkazach „`setl (setnge)`” i „`setnl (setge)`”,
- (ZF=1 lub SF<>OF) w rozkazach „`setle (setna)`” i „`setnle (seta)`”.

Znaczniki wyników operacji

Programowanie na poziomie asemblera jest możliwe tylko przy dokładnym śledzeniu za wartościami znaczników rejestru EFLAGS. W tab. 29 są pokazane zmiany znaczników rejestru EFLAGS po wykonaniu rozkazów. Zastosowano następujące zaznaczenia:

„+” w przypadku ustawienia wartości według wyniku operacji,
„0” lub ‘1’ w przypadku ustawienia konkretnej wartości znacznika,
„?” w przypadku nieprzewidywalnej zmiany wartości znacznika.

Tabela 29.

Stan znaczników rejestru EFLAGS po wykonaniu rozkazu

Mnemonic	Znaczniki								
	OF	DF	IF	TF	SF	ZF	AF	PF	CF
Rozkazy do przesyłania danych									
in, out, lahf, lds, lea, les, lods, lodsb, lodsw, mov, movs, movsb, movsw, pop, push, pushf, stos,stosb,stosw, xchg, xlat, xlatb									
popf	+	+	+	+	+	+	+	+	+
sahf					+	+	+	+	+
Rozkazy do operacji arytmetycznych									
aaa	?				?	?	+	?	+
aas, cbw, cwd									
aad, aam	?				+	+	?	+	?
adc, add, cmp	+				+	+	+	+	+
daa, das	?				+	+	+	+	+
dec, inc	+				+	+	+	+	
div	?				?	?	+	?	+
idiv	?				?	?	?	?	?
imul	+				?	?	?	?	+
mul	+				?	?	?	?	+
neg, ssb, sub	+				+	+	+	+	+
Rozkazy do operacji bitowych									
and, test, xor	0				+	+	?	+	0
not									
or	0				+	+	?	?	0
rcl, rcr, rol, ror, sal, sar, shl, shr	+								+
Rozkazy do sterowania przebiegiem programu									
clc									0
cmc									+
cld		0							
cli			0						
cmps,cmpsb,cmpsw	+				+	+	+	+	+
int, into					0	0			
iret	+	+	+	+	+	+	+	+	+
call, ja, jnbe, jae, jnb, jb, jnae, jbe, jna, jc, jz, jcxz, je, jz, jg, jnle, jge, jnl, jl, jnge, jnc, jne, jnz, jng, jle, jno, jnp, jpo, jns, jo, jp, jpe, js, jmp, loop, loope, loopz, loopne, loopnz, ret, retf									
rep, repe, repz, repne, repnz						+			
scas,scasb,scasw	+				+	+	+	+	+
stc									1
std		1							
sti			1						
Rozkazy do sterowania procesorem									
esc, hlt, lock, wait									