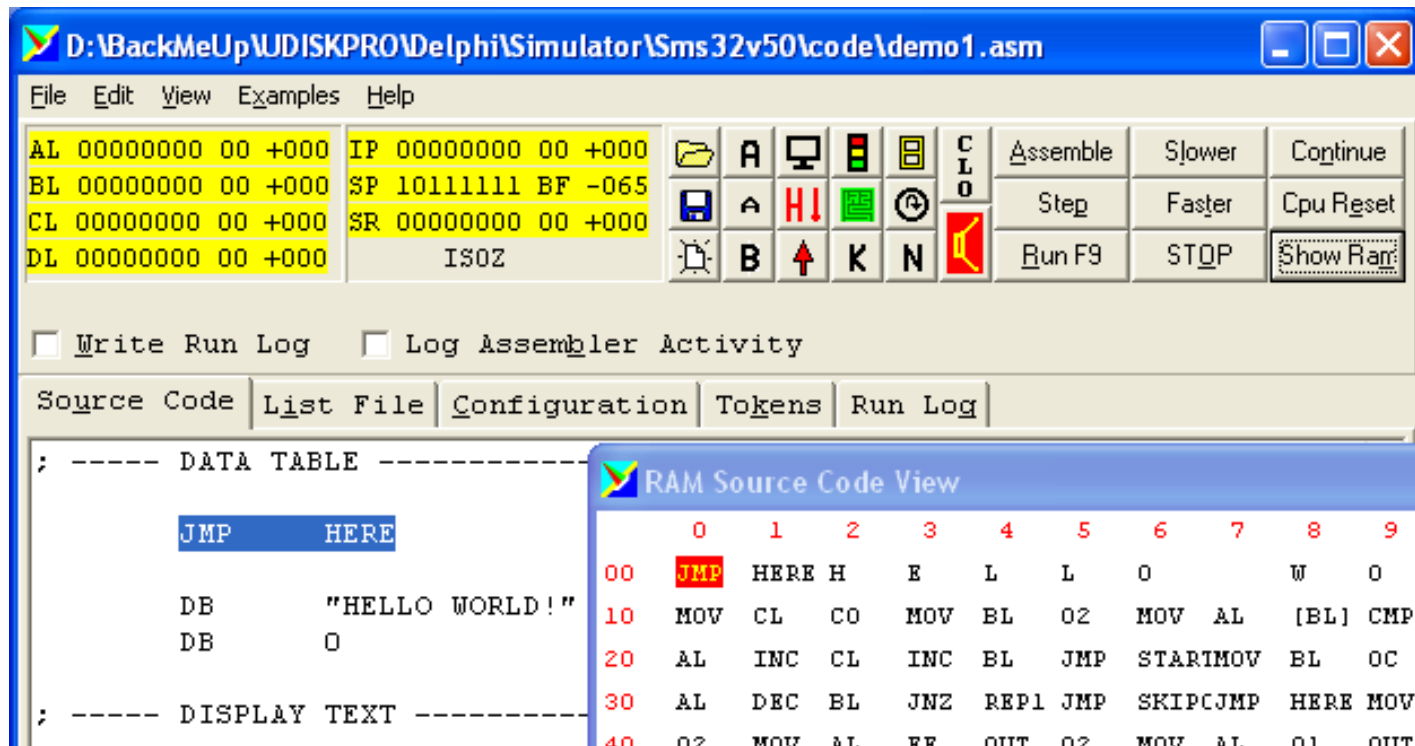
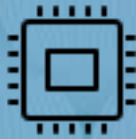




**Uniwersytet
w Siedlcach**

Architektura Systemów Komputerowych

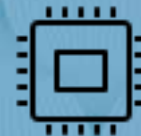
**dr Marcin
Stępnia**



- **Pomoc – Help** naciśnij klawisz **F1**
- **Piszemy Program.** Naciśnij **Alt+U**. Możesz teraz pisać swój program. Najlepiej w małych fragmentach kompilując i wykonując patrząc jednocześnie, co się dzieje. Możesz też metodą copy-paste wklejać program lub fragmenty programu. ALE ZDECYDOWANIE NAJLEPIEJ (ze względu na egzamin) PISAĆ KOD Z KLAWIATURY. Należy wstawiać komentarze, pomagają zapamiętać, o co w kodzie chodzi. Komentarze są pomijane podczas kompilacji. Zaczynają się of średnika „;” i są do końca linii.



Prosty kod



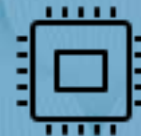
```
; ===== dodaj 2 =====  
      MOV      AL,0      ; wstaw 0 do rejestru AL  
REP:      ADD      AL,2      ; dodaj 2 do AL  
      JMP      REP      ; skocz do etykiety REP  
  
      END              ; koniec programu  
; =====
```

Wykonanie Programu



Step	Naciskając ten klawisz lub Alt+P wykonujesz pojedynczy krok programu
Run F9	Naciskając ten klawisz lub F9 lub Alt+R uruchamiasz cały program
Slower	Żeby lub zwolnić odpowiednio przyspieszyć wykonywanie programu, również przez Alt+L lub Alt+T
Faster	
STOP	Zatrzymaj wykonywanie programu, lub Alt+O lub Escape
Continue	Kontynuacja (po zatrzymaniu) wykonywania programu, również poprzez Alt+N
Cpu Reset	Restart od początku, również poprzez Alt+E
Show Ram	Otwórz okno z RAM, również poprzez Alt+M



Prosty kod



Asembler

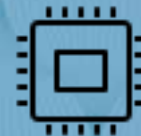
	Programy są pisane w asemblerze. Następnie są kompilowane do kodu maszynowego poprzez naciśnięcie tego klawisza, lub poprzez <u>Alt+A</u>
<input type="checkbox"/> Log Assembler Activity	Możesz zobaczyć jak to się dokonuje zaznaczając ten <u>box</u> .
	Jeśli naciskasz ten klawisz po raz pierwszy (od zmiany w kodzie), to program jest również kompilowany do kodu maszynowego

Asembler-kompilacja

1. **zapisuje kod**
2. **parsuje** i ew. zwraca listę błędów
3. **oblicza skoki**

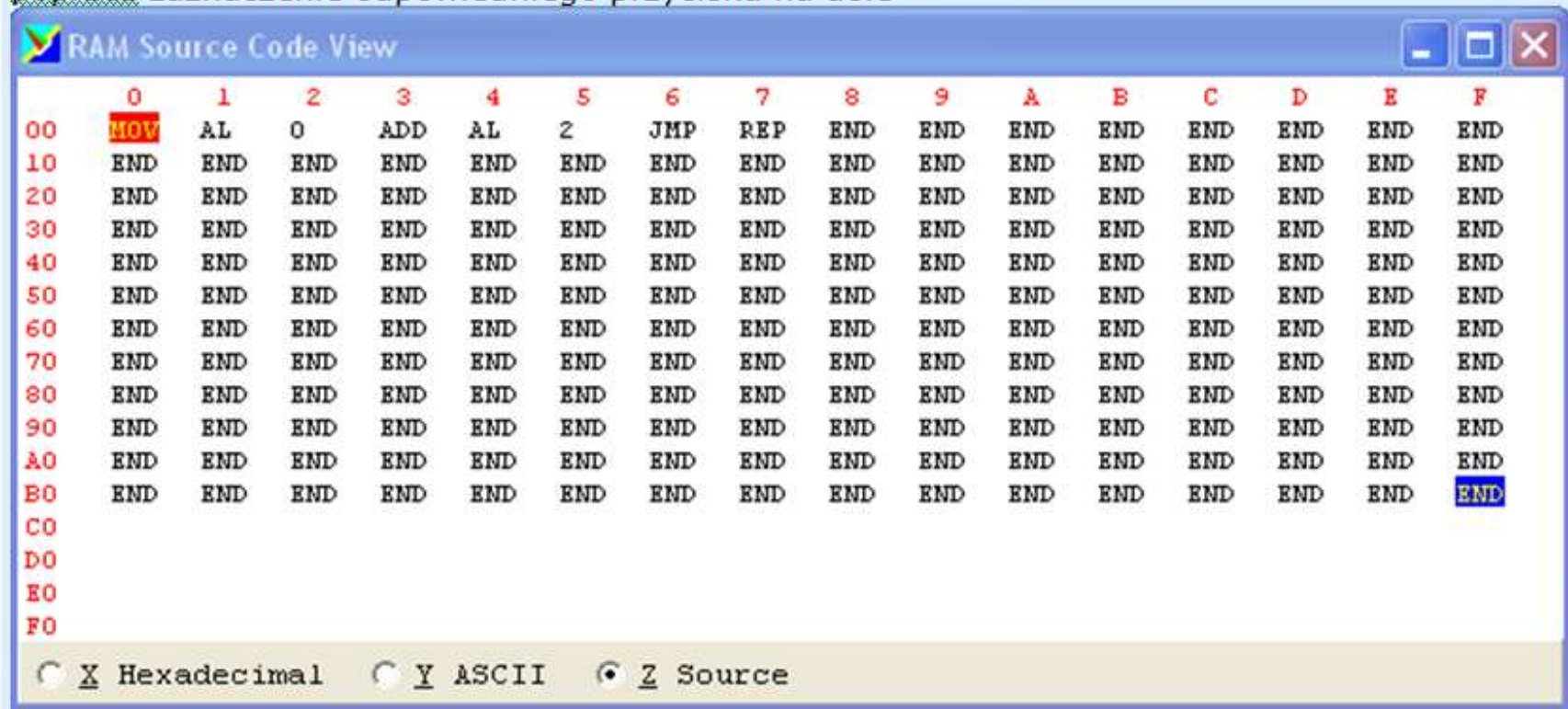


Prosty kod



Można zobaczyć kod maszynowy zapisany w RAM

poprzez zaznaczenie odpowiedniego przycisku na dole



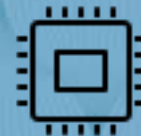
Hexadecimal – odpowiada kodowi binarnemu, na jakim pracuje CPU.

ASCII – wyświetla dane tekstowe, (jeśli są) w RAM.

Source Code – kod asemblera. Można zobaczyć gdzie zapisany jest program



Prosty kod



List File

Przycisk „List File” służy do listowania oryginalnego kodu.

Liczby w nawiasach kwadratowych, np. [1C] oznaczają adresy w RAM gdzie poszczególne instrukcje są zapisywane.

Po prawej stronie jest pokazany kod maszynowy.

Typowa linia.

```
MOV CL,C0      ; [10] D0 02 C0      ; bazowy adres Video w RAM
```

Instrukcja powoduje wstawienie liczby C0 do rejestru CL

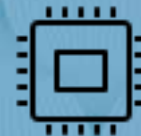
Kod maszynowy tej instrukcji jest zapisany w RAM pod adresem [10].

Ten kod to D0 00 C0.

Komentarze są po prawej stronie.



Przykład niepoprawnego kodu



Poniższy kod jest przykładem jak można kod kompletnie nieczytelny.
Komentarze są ważne.

Przykład - 99nasty.asm

```
; ----- jak nie należy kodować -----  
_: Mov BL,C0 Mov AL,3C Q: Mov [BL],AL CMP AL,7B  
JZ Z INC AL INC BL JMP Q Z: MOV CL,40 MOV AL,20  
MOV BL,C0 Y: MOV [BL],AL INC BL DEC CL JNZ Y JMP  
_ END ; naciśnij List File, będzie trochę lepiej!  
; -----
```

Niepoprawny komentarz

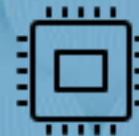
INC BL ; dodaj jeden do BL

Komentarz, który pomoże zrozumieć kod

INC BL ; zwiększ adres w BL, żeby wskazywał na następną komórkę w video RAM



Poprawny kod



A to jest kod dobrze napisany i odpowiednimi komentarzami

```
; ----- Program pokazujący kody ASCII dla niektórych znaków --
; ----- Dobrze napisany, z komentarzami -----
; ----- Można zrozumieć kod -----
; ----- Etykiety mają odpowiednie nazwy --

Start:
    Mov BL,C0      ; zapisz w BL adres do początku video RAM
    Mov AL,3C      ; 3C jest kodem ASCII dla symbolu 'większy'

Here:
    Mov [BL],AL    ; zapisz ten kod ASCII (z AL) do RAM pod adres wskazany przez BL
    CMP AL,7B      ; porównaj AL z '{'
    JZ Clear       ; jeśli AL zawiera '{' to skocz do etykiety Clear:
    INC AL         ; wstaw następny kod ASCII do AL
    INC BL         ; zwiększ adres w BL, aby wskazywał następną komórkę w video RAM
    JMP Here       ; wróć do Here

Clear:
    MOV CL,40      ; ma być 40 (hex) powtórzeń
    MOV AL,20      ; kod ASCII dla spacji wpisz do AL
    MOV BL,C0      ; w BL jest adres początku video RAM

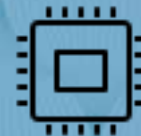
Loop:
    MOV [BL],AL    ; wstaw kod ASCII dla spacji (z AL) do video RAM pod adres z BL
    INC BL         ; zwiększ adres w BL, aby wskazywał na następną komórkę w video RAM
    DEC CL         ; zmniejsz ilość powtórzeń w CL o jeden
    JNZ Loop       ; jeśli CL nie jest zerem to skocz do etykiety Loop
    JMP Start      ; CL jest zero, a więc cofnij się do etykiety Start

    END

; -----
```




Instrukcje



Instrukcja MOV. Flagi nie są ustawiane.

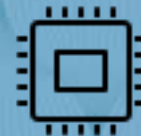
Asembler		Kod maszynowy		Opis
MOV	AL,15	<u>D0</u> <u>00</u> <u>15</u>	AL = 15	Wstaw 15 do AL
MOV	BL,[15]	<u>D1</u> <u>01</u> <u>15</u>	BL = [15]	Skopiuj i wstaw RAM[15] do BL
MOV	[15],CL	<u>D2</u> <u>15</u> <u>02</u>	[15] = CL	Skopiuj i wstaw CL do RAM[15]
MOV	DL,[AL]	<u>D3</u> <u>03</u> <u>00</u>	DL = [AL]	Skopiuj i wstaw RAM[AL] do DL
MOV	[CL],AL	<u>D4</u> <u>02</u> <u>00</u>	[CL] = AL	Skopiuj i wstaw AL do RAM[CL]

Bezpośrednie instrukcje arytmetyczno-logiczne. Flagi mogą być ustawiane.

Asembler		Kod maszynowy	
<u>ADD</u>	<u>AL,BL</u>	<u>A0</u> <u>00</u> <u>01</u>	AL = AL + BL
<u>SUB</u>	<u>BL,CL</u>	<u>A1</u> <u>01</u> <u>02</u>	BL = BL - CL
<u>MUL</u>	<u>CL,DL</u>	<u>A2</u> <u>02</u> <u>03</u>	CL = CL * DL
<u>DIV</u>	<u>DL,AL</u>	<u>A3</u> <u>03</u> <u>00</u>	DL = DL / AL
<u>INC</u>	<u>DL</u>	<u>A4</u> <u>03</u>	DL = DL + 1
<u>DEC</u>	<u>AL</u>	<u>A5</u> <u>00</u>	AL = AL - 1
<u>AND</u>	<u>AL,BL</u>	<u>AA</u> <u>00</u> <u>01</u>	AL = AL AND BL
<u>OR</u>	<u>CL,BL</u>	<u>AB</u> <u>03</u> <u>02</u>	CL = CL OR BL
<u>XOR</u>	<u>AL,BL</u>	<u>AC</u> <u>00</u> <u>01</u>	AL = AL XOR BL
<u>NOT</u>	<u>BL</u>	<u>AD</u> <u>01</u>	BL = NOT BL
<u>ROL</u>	<u>AL</u>	<u>9A</u> <u>00</u>	Rotuj bity w lewo.
<u>ROR</u>	<u>BL</u>	<u>9B</u> <u>01</u>	Rotuj bity w prawo.
<u>SHL</u>	<u>CL</u>	<u>9C</u> <u>02</u>	Przesuń bity w lewo.
<u>SHR</u>	<u>DL</u>	<u>9D</u> <u>03</u>	Przesuń bity w prawo.



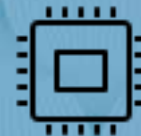
ROTACJE I PRZESUNIĘCIA LOGICZNE ORAZ ARYTMETYCZNE



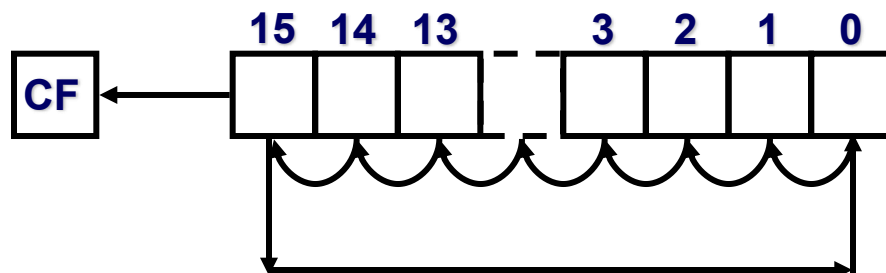
- Rozkazy rotacji:
 - ROL, ROR, RCL, RCR,
 - rozkazy przesunięć logicznych:
 - SAL/SHL, SHR
 - przesunięć arytmetycznych:
 - SAR,
- } **przemieszczają bity rejestru lub komórki pamięci 8- lub 16-bitowej o 1 bit (lub więcej) w prawo lub w lewo**
- Wynik rotacji lub przesunięcia wpisywany jest zawsze w miejsce argumentu (operandu).
 - Rotacja różni się tym od przesunięć, iż w przesunięciach następuje utrata bitów skrajnych przesuwanych rejestrów lub komórek pamięci, natomiast w rotacjach nie ma tej utraty.
 - Rozkazy przesunięć logicznych z uwagi na "pewne swoje pokrewieństwo w działaniu" do rozkazów mnożenia i dzielenia, używane są często zamiennie, tym bardziej, że wykonują się od nich o wiele szybciej
 - Rozkazy przesunięć oraz rozkazy rotacji oddziałują na znaczniki (flagi):
 - CF (flaga przeniesienia – nie ma jej w symulatorze)
 - OF (flaga przepełnienia , O w symulatorze).



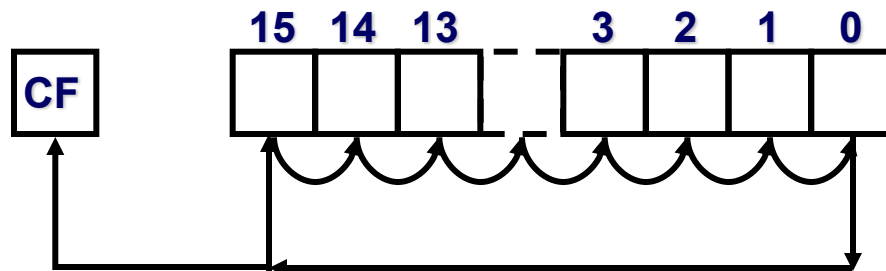
ROZKAZY ROTACJI (PRZESUNIĘCIA CYKLICZNEGO)



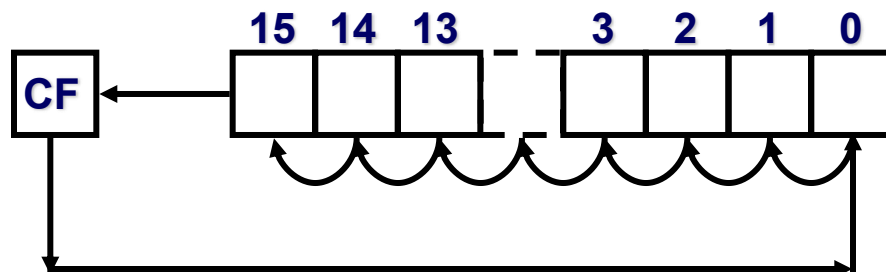
ROL x,wielkość



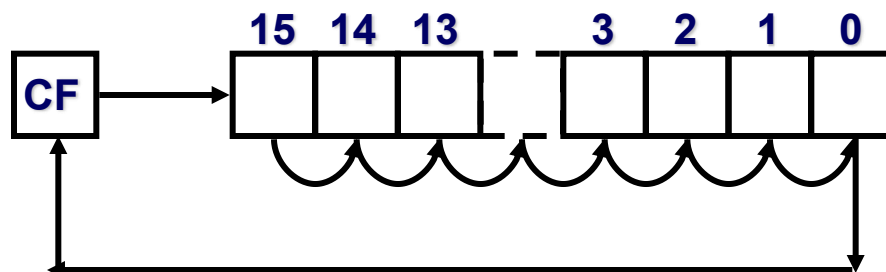
ROR x,wielkość



RCL x,wielkość

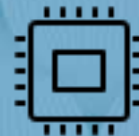


RCR x,wielkość





ROTACJA W LEWO - ROL



Rozkaz ROL dokonuje przesunięcia cyklicznego argumentu1 w lewo, o liczbę bitów, określoną argumentem2.

ROL argument1,argument2

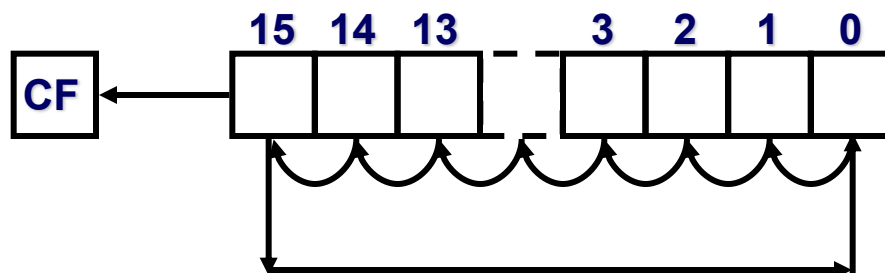
gdy dokonujemy przesunięcia o liczbę bitów większą niż 1 to argumentem2 jest rejestr CL, który uprzednio należy załadować liczbą, określającą o ile bitów należy przesunąć lewy argument.

W rotacji bit opuszczający rejestr (komórkę pamięci) z jednego końca, wchodzi do niego (do niej) z drugiego końca, a jego wartość kopiowana jest do rejestru znaczników, jako znacznik (flaga) przeniesienia, CF.

Jeśli podczas tej rotacji, o 1 bit w lewo, nie zmieni się zawartość bitu znaku, to wówczas znacznik OF jest zerowany, w przeciwnym razie znacznik OF przyjmuje wartość 1.

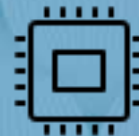
Przykłady:

```
.....  
ROL AL, 1  
.....  
MOV CL, 14  
ROL BL, CL  
.....
```





ROTACJA W PRAWO - ROR



Rozkaz ROR dokonuje przesunięcia cyklicznego argumentu1 w prawo, o liczbę bitów, określoną argumentem2.

ROR argument1,argument2

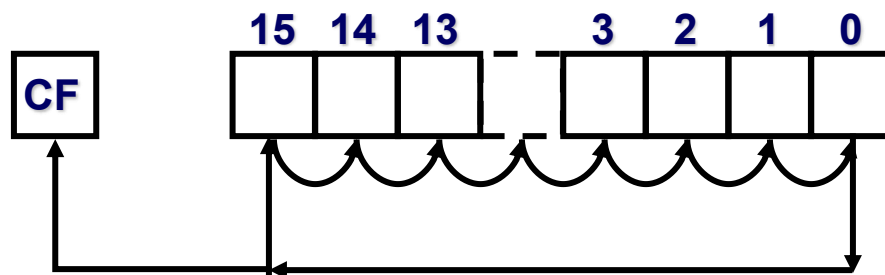
gdy dokonujemy przesunięcia o liczbę bitów większą niż 1 to argumentem2 jest rejestr CL, który uprzednio należy załadować liczbą, określającą o ile bitów należy przesunąć lewy argument.

W rotacji bit opuszczający rejestr (komórkę pamięci) z jednego końca, wchodzi do niego (do niej) z drugiego końca, a jego wartość kopiowana jest do rejestru znaczników, jako znacznik (flaga) przeniesienia, CF.

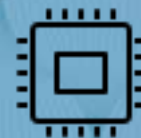
Jeśli podczas tej rotacji, o 1 bit w lewo, nie zmieni się zawartość bitu znaku, to wówczas znacznik OF jest zerowany, w przeciwnym razie znacznik OF przyjmuje wartość 1.

Przykłady:

```
.....  
ROR AL, 1  
.....  
MOV CL, 14  
ROR BX, CL  
.....
```

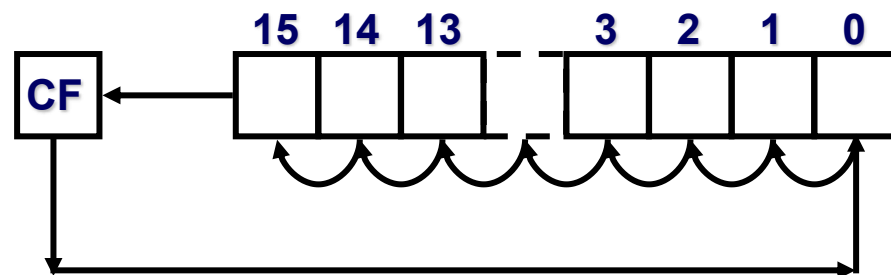


ROTACJA W LEWO (W PRAWO) Z PRZENIESIENIEM - RCL (RCR)



Rozkaz RCL dokonuje przesunięcia cyklicznego argumentu1 w lewo, o liczbę bitów, określoną argumentem2 (jeśli więcej niż 1 bit to użyć rejestru CL).

RCL argument1,argument2

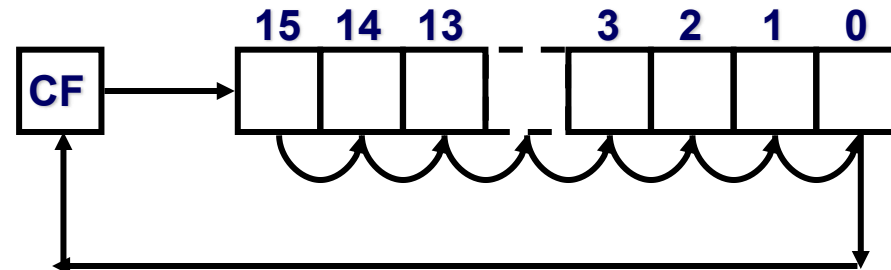


W tego rodzaju rotacji („rotacja długa”) bit CF jest łącznikiem między najmłodszym a najstarszym bitem rolowanego argumentu (operandu).

Skrajny bit po opuszczeniu rolowanego argumentu wchodzi na miejsce znacznika CF, podczas gdy poprzednia wartość CF wchodzi na brakujące miejsce po stronie przeciwnej tegoż argumentu.

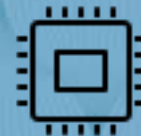
Rozkaz RCR działa analogicznie do rozkazu RCL lecz dokonuje przesunięcia w prawo

RCR argument1,argument2

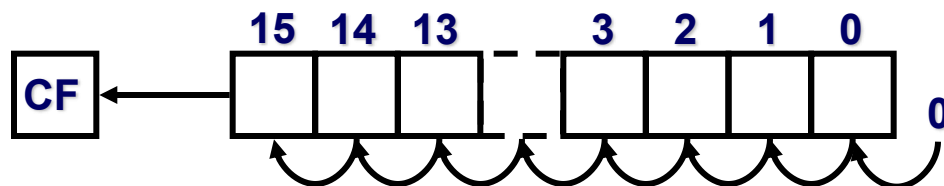




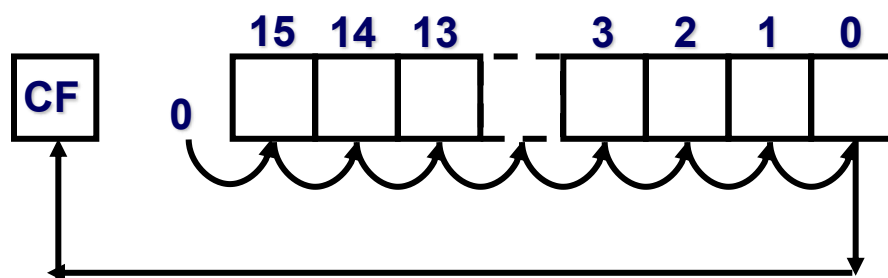
ROZKAZY PRZESUNIĘCIA LOGICZNEGO W LEWO (SHL) I PRAWO (SHR)



SHL x,wielkość



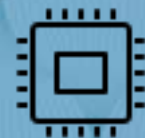
SHR x,wielkość



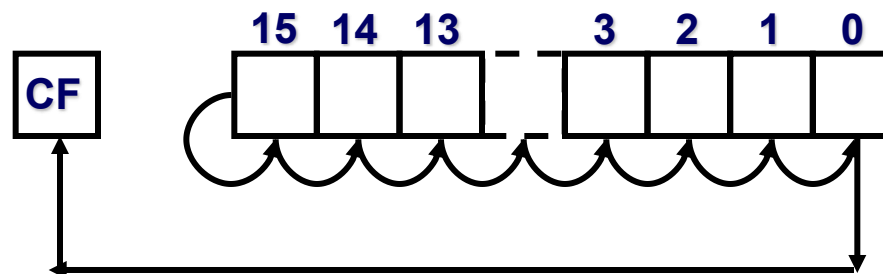
Rozkazy SHL i SHR używane są w programach jako rozkazy odpowiednio zastępujące: rozkaz mnożenia MUL i rozkaz dzielenia DIV, jak też w sytuacjach związanych z tablicami, z uzyskiwaniem dostępu do danego elementu tablicy.



ROZKAZY PRZESUNIĘCIA ARYTMETYCZNEGO W PRAWO (SAR)



SAR x, wielkość



W czasie przesuwania najmniej znaczące bity argumentu są wprowadzane kolejno w miejsce znacznika CF, natomiast najbardziej znaczący bit nie zmienia wartości.

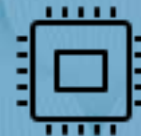
Arytmetyczne przesunięcie w lewo (SAL) nie istnieje. Gdy wartość danej zajmuje mniej bitów niż długość pojemnika danych zmniejszona o wielkość przesunięcia, to można użyć przesunięcia w lewo.

Działa jak przesunięcie logiczne w lewo (SHL).

W innych przypadkach bit znaku musi być programowo zapamiętany i potem przywrócony.



Instrukcje



Pośrednie instrukcje arytmetyczno-logiczne.

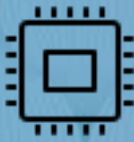
Flagi mogą być ustawiane.

Asembler		Kod maszynowy	
ADD	AL,12	B0 00 12	AL = AL + 12
SUB	BL,15	B1 01 15	BL = BL - 15
MUL	CL,03	B2 02 03	CL = CL * 3
DIV	DL,02	B6 03 02	DL = DL / 2
AND	AL,10	BA 00 10	AL = AL AND 10
OR	CL,F0	BB 02 F0	CL = CL OR F0
XOR	AL,AA	BC 00 AA	AL = AL XOR AA

Instrukcje porównania.

Flagi mogą być ustawiane.

Asembler		Kod maszynowy	Opis
CMP	AL,BL	DA 00 01	flaga 'Z' ustawiona, jeśli AL = BL flaga 'S' ustawiona, jeśli AL < BL
CMP	BL,13	DB 01 13	flaga 'Z' ustawiona, jeśli BL = 13. flaga 'S' ustawiona, jeśli BL < 13.
CMP	CL,[20]	DC 02 20	flaga 'Z' ustawiona, jeśli CL = [20]. flaga 'S' ustawiona, jeśli CL < [20].



Uzupełnienie dwójkowe

bit pierwszy od lewej, to bit znaku

1 0 1 0 1 0 1 0

^

^ ten bit ma wartość -128 dziesiętnie oraz -80 hexadecymalnie

pozostałe siedem bitów są traktowane, jako liczby od 0 and 127.

Np.

1 1 1 1 1 1 1 1 - 128d + 127d = -1d

^

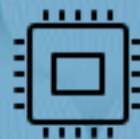
^ -128d

Liczba dziesiętna 127

0 1 1 1 1 1 1 1 0 + 127d = 127d

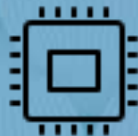
^

^ bit zerowy wskazuje na liczbę 0.



Liczby ujemne

Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
-128	80	-127	81	-126	82	-125	83	-124	84	-123	85	-122	86	-121	87
-120	88	-119	89	-118	8A	-117	8B	-116	8C	-115	8D	-114	8E	-113	8F
-112	90	-111	91	-110	92	-109	93	-108	94	-107	95	-106	96	-105	97
-104	98	-103	99	-102	9A	-101	9B	-100	9C	-099	9D	-098	9E	-097	9F
-096	A0	-095	A1	-094	A2	-093	A3	-092	A4	-091	A5	-090	A6	-089	A7
-088	A8	-087	A9	-086	AA	-085	AB	-084	AC	-083	AD	-082	AE	-081	AF
-080	B0	-079	B1	-078	B2	-077	B3	-076	B4	-075	B5	-074	B6	-073	B7
-072	B8	-071	B9	-070	BA	-069	BB	-068	BC	-067	BD	-066	BE	-065	BF
-064	C0	-063	C1	-062	C2	-061	C3	-060	C4	-059	C5	-058	C6	-057	C7
-056	C8	-055	C9	-054	CA	-053	CB	-052	CC	-051	CD	-050	CE	-049	CF
-048	D0	-047	D1	-046	D2	-045	D3	-044	D4	-043	D5	-042	D6	-041	D7
-040	D8	-039	D9	-038	DA	-037	DB	-036	DC	-035	DD	-034	DE	-033	DF
-032	E0	-031	E1	-030	E2	-029	E3	-028	E4	-027	E5	-026	E6	-025	E7
-024	E8	-023	E9	-022	EA	-021	EB	-020	EC	-019	ED	-018	EE	-017	EF
-016	F0	-015	F1	-014	F2	-013	F3	-012	F4	-011	F5	-010	F6	-009	F7
-008	F8	-007	F9	-006	FA	-005	FB	-004	FC	-003	FD	-002	FE	-001	FF

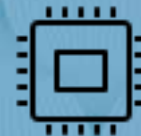


Liczby dodatnie

Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
+000	00	+001	01	+002	02	+003	03	+004	04	+005	05	+006	06	+007	07
+008	08	+009	09	+010	0A	+011	0B	+012	0C	+013	0D	+014	0E	+015	0F
+016	10	+017	11	+018	12	+019	13	+020	14	+021	15	+022	16	+023	17
+024	18	+025	18	+026	1A	+027	1B	+028	1C	+029	1D	+030	1E	+031	1F
+032	20	+033	21	+034	22	+035	23	+036	24	+037	25	+038	26	+039	27
+040	28	+041	29	+042	2A	+043	2B	+044	2C	+045	2D	+046	2E	+047	2F
+048	30	+049	31	+050	32	+051	33	+052	34	+053	35	+054	36	+055	37
+056	38	+057	39	+058	3A	+059	3B	+060	3C	+061	3D	+062	3E	+063	3F
+064	40	+065	41	+066	42	+067	43	+068	44	+069	45	+070	46	+071	47
+072	48	+073	49	+074	4A	+075	4B	+076	4C	+077	4D	+078	4E	+079	4F
+080	50	+081	51	+082	52	+083	53	+084	54	+085	55	+086	56	+087	57
+088	58	+089	59	+090	5A	+091	5B	+092	5C	+093	5D	+094	5E	+095	5F
+096	60	+097	61	+098	63	+099	63	+100	64	+101	65	+102	66	+103	67
+104	68	+105	69	+106	6A	+107	6B	+108	6C	+109	6D	+110	6E	+111	6F
+112	70	+113	71	+114	72	+115	73	+116	74	+117	75	+118	76	+119	77
+120	78	+121	79	+122	7A	+123	7B	+124	7C	+125	7D	+126	7E	+127	7F



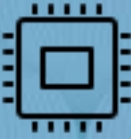
Instrukcje skoku



- Flaga "Z" (zera) jest ustawiana na jeden, jeśli obliczenia dały wynik zerowy.
- Flaga "S" (znaku) jest ustawiana na jeden, jeśli obliczenia dały wynik ujemny.
- Flaga "O" (przepełnienia) jest ustawiana, jeśli wynik był zbyt duży, aby zmieścić się w rejestrze. Gdy wartość rejestru osiągnie $7F_{(16)}$ lub $127_{(10)}$, to następna powinna być liczba 128, ale ze względu na sposób, w jaki numery są przechowywane w systemie binarnym, następną liczbą jest minus 128. Efekt ten nazywany jest przepełnieniem.
- Flaga "I" (przerwania) jest ustawiana, jeśli przerwania są włączone.

Tabela 2. Instrukcje skoku symulowanego mikroprocesora

Instr.	Działanie
JMP	skok bezwarunkowy; zawsze skacze do wskazanej etykiety
JZ	skacze do wskazanej etykiety jeżeli flaga "Z" jest ustawiona
JNZ	skacze do wskazanej etykiety jeżeli flaga "Z" nie jest ustawiona
JS	skacze do wskazanej etykiety jeżeli flaga "S" jest ustawiona
JNS	skacze do wskazanej etykiety jeżeli flaga "S" nie jest ustawiona
JO	skacze do wskazanej etykiety jeżeli flaga "O" jest ustawiona
JNO	skacze do wskazanej etykiety jeżeli flaga "O" nie jest ustawiona



Procedures and Interrupts. Flags NOT set.

CALL, RET, INT and IRET are available only in the registered version.

Assembler

Machine Code

Explanation

CALL 30

CA 30

Save IP on the stack and jump to the procedure at address 30.

RET

CB

Restore IP from the stack and jump to it.

INT 02

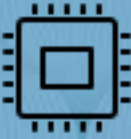
CC 02

Save IP on the stack and jump to the address (interrupt vector) retrieved from RAM[02].

IRET

CD

Restore IP from the stack and jump to it.



Stack Manipulation Instructions. Flags NOT set.

Assembler

Machine Code

Explanation

PUSH

BL

E0 01

BL is saved onto the stack.

POP

CL

E1 02

CL is restored from the stack.

PUSHF

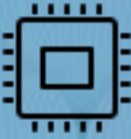
EA

SR flags are saved onto the stack.

POPF

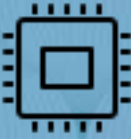
EB

SR flags are restored from the stack.



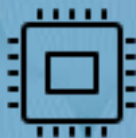
Input Output Instructions. Flags NOT set.

Assembler		Machine Code	Explanation
IN	07	F0 07	Data input from I/O port 07 to AL.
OUT	01	F1 01	Data output to I/O port 07 from AL.



Miscellaneous Instructions. CLI and STI set I flag.

Assembler		Machine Code	Explanation
CLO		FE	Close visible peripheral windows.
HALT		00	Halt the processor.
NOP		FF	Do nothing for one clock cycle.
STI		FC	Set the interrupt flag in the Status Register.
CLI		FD	Clear the interrupt flag in the Status Register.
ORG	40	Code origin	Assembler directive: Generate code starting from address 40.
DB	"Hello"	Define byte	Assembler directive: Store the ASCII codes of 'Hello' into RAM.
DB	84	Define byte	Assembler directive: Store 84 into RAM.



D:\BackMeUp\UDISKPRO\Delphi\Simulator\Sms32v50\code\foo.ASM

File Edit View Examples Help

AL 00000000 00 +000	IP 00000000 00 +000		A	
BL 00000000 00 +000	SP 10111111 BF -065		A	
CL 00000000 00 +000	SR 00000000 00 +000		B	
DL 00000000 00 +000	IS0Z		K	N

Keyboard Input

Waiting for keyboard input >

Run F9

STOP

Show Ram

Saved D:\BackMeUp\UDISKPRO\Delphi\Sim ... okens. Calculate jumps. Success.

☐ Write Run Log

☐ Log Assembler Activity

Source Code

List File

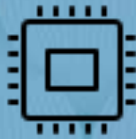
Configuration

Tokens

Run Log

in 00

end



; -----

; Input key presses from the keyboard until Enter is pressed.

; -----

CLO ; Close unwanted windows.

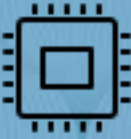
Rep: IN 00 ; Wait for key press - Store it in AL.

CMP AL, 0D ; sprawdź czy Enter? (ASCII 0D) J

JNZ Rep ; No - jump back. Yes - end.

END

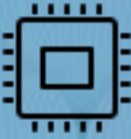
; patrz co się dzieje z rejestrem AL !!!



CLO

MOV BL, C0 ; wstaw do rejestru BL liczbę C0
; C0 to adres pierwszej komórki VDU

Rep: IN 00
MOV [BL], AL ; kopiuj z AL do komórki RAM-jakiej?
INC BL ; zwiększ BL o jeden
CMP AL, 0D ; sprawdź czy Enter? (ASCII 0D)
JNZ Rep
END



CLO

MOV DL, 00 ; pierwszy licznik

MOV BL, C0 ; wstaw do rejestru BL liczbę C0

Rep_1:

IN 00

PUSH AL ; skopiuj z AL i wstaw na stos

INC DL ; zwiększ licznik o 1

CMP AL, 0D ; Enter? (ASCII 0D)

JNZ Rep_1

Rep_2:

POP AL ; zdejmij ze stosu na AL

MOV [BL], AL

INC BL ; zwiększ BL o jeden

DEC DL ; zmniejsz DL o jeden

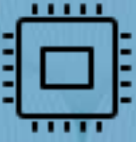
CMP DL, 00 ; porównaj czy DL = 0

JNZ Rep_2

END



Podsumowanie



- Był to opis (zgrubny) typowego języka maszynowego i jego instrukcji
 - Halt Instruction
 - Data Instructions
 - Arithmetic/Logic Instructions
 - Control Instructions
 - Memory Instructions

Dziękuję za uwagę!

Slajdy na podstawie wykładów prof. Stanisława Ambroszkewicza

Tło obrazka autorstwa rawpixel.com – pobrane z serwisu [Freepik](#)
[Memory Slot](#) icon by [Icons8](#)
[Electronics](#) icon by [Icons8](#)