15. Gilmore Ch.: *Wwiedienje w mikroprocessornuju tiechniku*. Moskwa 1984 Mir, s. 334. Przekład książki: Charles M. Gilmore: *Introduction to Microprocessors*. McGraw Hill 1981
16. *Kompiutiery. Sprawocznoje rukowodstwo w trioch tomach*. Moskwa 1986 Mir, s. 413, 440, 406. Tłumaczenie książki: *The McGraw Hill Computer Handbook*, 1983
17. Sacha Krzysztof, Rydzewski Andrzej: *Mikroprocesor w pytaniach i odpowiedziach*. Warszawa 1987 WNT, s. 287
18. Ruszczyc Jan: *Poznajemy FORTH*. Warszawa 1987 SOETO, s. 230.

ANEKSY

A1 Rozkazy 6502. Opis i zastosowania

Poniższe zestawienie obejmuje wszystkie rozkazy mikroprocesora 6502. W nagłówku każdego rozkazu od lewej: mnemonik, jego rozwinięcie angielskie oraz po prawej nazwy znaczników, na które wpływa dany rozkaz. W treści: opis działania rozkazu oraz omówienie najczęstszych zastosowań.

Przy rozkazie BCC obszerniej omówiono wspólne cechy wszystkich ośmiu rozkazów odgałęzień.

Zastosowano następujące oznaczenia:

`A` – akumulator

`X, Y` – rejestry indeksowe X i Y

`M` – bajt w pamięci

`DANE`, dane – bajt danych w pamięci lub operandzie trybu natychmiastowego.

Nazwy znaczników – takie jak w treści książki (rozdz. 2 i 4).

Dalsze informacje o rozkazach, ich kodach i trybach adresowania zawarte są w następnych aneksach.

`ADC` *add with carry* N V – B D I Z C

Opis: Dodaje bajt danych do A plus znacznik przeniesienia C. Ustawia C, gdy wynik przekroczył 255 (FF hex). Wynik zostawia w A.

Zastosowania: Jeżeli w chwili dodawania znacznik C był ustawiony (miał wartość 1), wynik będzie o 1 większy od sumy liczb. Dlatego przed rozpoczęciem każdej operacji dodawania należy skasować C z pomocą rozkazu CLC. Jeżeli po wykonaniu ADC znacznik C został ustawiony, oznacza to, że wynik przekroczył rozmiar bajtu czyli 255. W akumulatorze znajduje się wówczas osiem dolnych bitów wyniku, a w C – najbardziej znaczący. Umożliwia to

wykorzystanie wartości C w dodawaniu liczb dwu i wielobajtowych. Patrz punkt 3.5.

ADC wpływa również na inne znaczniki. Ustawienie znacznika nadmiaru V sygnalizuje, że nastąpiło przeniesienie z bitu b6 do b7. V wykorzystywany jest w dodawaniu i odejmowaniu ze znakiem.

Duże znaczenie ma Z, którego ustawienie sygnalizuje wynik 0. Ustawienie N oznacza, że bit b7 wyniku przybrał wartość 1, co w arytmetyce ze znakiem oznacza liczbę ujemną.

Po przejściu z pomocą SED na tryb binarno-dziesiętny rozkaz ADC wykonuje dodawania w tym trybie.

Jako jeden z dwóch rozkazów arytmetycznych ADC jest szeroko wykorzystywany we wszelkich obliczeniach, w tym w podprogramach mnożenia.

`AND` *logical AND* N V – B D I Z C

Opis: Wykonuje logiczne I (koniunkcję) na bitach danych i A. Wynik zostawia w A. Bity wyniku przybierają wartość 1 tylko wtedy, gdy oba odpowiednie bity danych i A mają wartość 1.

Zastosowania: Głównym zastosowaniem AND jest kasowanie części bitów z pomocą wzorcowego bajtu zwanego maską przy zachowaniu wartości pozostałych bitów. Oto przykład:

```
LDA NN      Przetwarzana liczba
AND #$7F    01111111 bin
```

spowoduje, że bit b7 w liczbie NN bez względu na jego poprzednią wartość przybierze 0, natomiast pozostałe bity nie ulegną zmianie.

`ASL` *arithmetic shift left* N V – B D I Z C

Opis: Przesuwa wszystkie bity w bajcie wskazanym przez operand o 1 w lewo i zostawia wynik w tym bajcie. W trybie akumulatora jest to A, w pozostałych trybach M. Skrajny lewy bit bajtu jest przenoszony do znacznika C, a do skrajnego prawego wprowadzane jest 0.

Zastosowania: Umożliwia szybkie mnożenie przez 2. Liczby większe niż 255 można mnożyć przez 2 łącząc stosowanie ASL i ROL. Patrz punkt 3.10. ASL jest szeroko wykorzystywane w podprogramach mnożenia.

Innym zastosowaniem ASL jest przesuwanie dolnego półbajtu do górnego i wypełnienie dolnego zerami. Uzyskuje się to czterokrotnym ASL.

`BCC` *branch if carry clear* `N V - B D I Z C`

Opis: Grupa ośmiu rozkazów odgałęzień czyli skoków warunkowych wykonuje czynności na takich samych zasadach, które zostaną tu omówione.

1. Każdy rozkaz odgałęzienia reaguje przeskokiem pod nowy adres w programie na inny z czterech znaczników: N, V, Z i C, w rejestrze stanu procesora P oraz na inną wartość tego znacznika 0 lub 1… Gdy odpowiedni znacznik ma odmienną wartość, przeskok nie następuje i wykonywany jest rozkaz w programie następujący bezpośrednio po rozkazie odgałęzienia.
2. Zawsze w 1-bajtowym operandzie zawarte jest przesunięcie w stosunku do adresu rozkazu następującego bezpośrednio za rozkazem odgałęzienia traktowane jako liczba z e z n a k i e m, dodawana do owego adresu. Tak więc zasięg skoku warunkowego ograniczony jest do 127 bajtów naprzód i 128 bajtów wstecz licząc od następnego rozkazu czyli +129 i –126 licząc od miejsca, w którym umieszczamy rozkaz odgałęzienia.
3. Rozkazy nie wpływają na stan znaczników ani rejestrów.
4. W asemblerze podaje się adres docelowy lub jego etykietę, a program asemblujący sam wylicza właściwy 1-bajtowy operand.

Wracając do rozkazu BCC, wykonuje on skok, gdy znacznik przeniesienia C = 0.

Zastosowania: W oparciu o sprawdzenie wyniku CMP, ADC i innych rozkazów wpływających na C rozkaz BCC umożliwia wykonanie odgałęzienia czyli skoku warunkowego analogicznego do struktur IF … THEN lub ON … GOTO w Basicu. Przy porównaniach wykonuje odgałęzienie, gdy

```
DANE > A
```

`BCS` *branch if carry set* `N V - B D I Z C`

Opis: Działanie analogiczne jak BCC, ale gdy znacznik przeniesienia C jest ustawiony (C = 1). Gdy C = 0, wykonywany jest następny rozkaz.

Zastosowania: Podobne jak BCC. Przy porównaniach wykonuje odgałęzienie, gdy

```
DANE <= A
```

`BEQ` *branch if result zero* `N V - B D I Z C`

Opis: Wykonuje czynności analogiczne jak BCC, gdy znacznik wyniku zerowego Z jest ustawiony (Z = 1). Gdy Z = 0, wykonywany jest następny rozkaz.

Zastosowania: Umożliwia wykonanie odgałęzienia warunkowego do innego fragmentu programu w przypadku, gdy sprawdzenie Z wykazało równość dwóch liczb lub zerowy wynik operacji.

Przy porównaniach wykonuje odgałęzienie, gdy

```
DANE = A
```

`BIT` *test bits* N V – B D I Z C

Opis: Wykonuje logiczne AND na A i M, jednak nie zapisuje wyniku, natomiast wpływa na znaczniki. Znacznik Z ustawiony jest, gdy wynikiem logicznego AND jest 0, kasowany w przeciwnym wypadku. Ponadto bity b7 i b6 danych są k o p i o w a n e odpowiednio do znaczników N i V.

Zastosowania: Zaletą BIT jest to, że przy sprawdzaniu nie zmienia wartości A ani M. BIT upraszcza sprawdzanie bitów b6 i b7 w danych. Użyteczność rozkazu ogranicza to, że nie można stosować go w trybie natychmiastowym ani w adresowaniu indeksowym. Dlatego zastępuje się go często innymi sprawdzeniami, np. CMP i AND.

`BMI` *branch if minus* N V – B D I Z C

Opis: Wykonuje czynności analogiczne jak BCC, gdy znacznik wyniku ujemnego N jest ustawiony (N = 1). Gdy N = 0, wykonywany jest następny rozkaz.

Zastosowania: Pozwala wykonać odgałęzienie, gdy bit b7 jest ustawiony tzn. bajt ma wartość większą niż 127. Wykorzystuje się to m.in. w przeszukiwaniu tablic słów kluczowych języków programowania w interpretatorach ustawiając b7 w ostatnim znaku słowa. Poza tym możliwe do wykorzystania głównie w arytmetyce ze znakiem.

`BNE` *branch on not equal to zero* N V – B D I Z C

Opis: Wykonuje czynności analogiczne jak BCC, gdy znacznik wyniku zerowego jest skasowany (Z = 0). Gdy Z = 1, wykonywany jest następny rozkaz.

Zastosowania: Powoduje odgałęzienie tylko wtedy, gdy porównywane liczby są nierówne. Obok zastosowań analogicznych jak w konstrukcjach Basicu IF … THEN i ON … GOTO rozkaz BNE jest również szeroko wykorzystywany w budowie pętli liczonych podobnych do konstrukcji Basicu: FOR I=1 TO N: sekwencja instrukcji: NEXT I. Na przykład:

```
        LDX #$20
        LDA #0
CYKL    STA $4000,X
        DEX
        BNE CYKL
```

Powrót do etykiety CYKL następować będzie, dopóki w X nie pojawi się 0, czyli 32 ($20) razy.

Przy porównaniach BNE wykonuje odgałęzienie, gdy

```
DANE <> A
```

`BPL` *branch if plus* N V – B D I Z C

Opis: Wykonuje czynności analogiczne jak BCC, gdy znacznik wyniku ujemnego jest skasowany (N = 0). Gdy N = 1 wykonywany jest następny rozkaz.

Zastosowania: Może być użyty do sprawdzenia, czy bajty mają najwyższy bit skasowany, reprezentują zatem wartości mniejsze niż 128, co może być istotne przy kodach znaków. Poza tym możliwe do wykorzystania głównie w arytmetyce ze znakiem.

`BRK` break N V – B D I Z C

Opis: Wywołuje przerwanie programowe. Ustawia znacznik B, po czym wstawia kolejno na stos licznik programu (mniej znaczący bajt jako pierwszy) i rejestr stanu procesora P. W PCL i PCH umieszczone zostają zawartości komórek o adresach odpowiednio FFFE i FFFF. W przeciwieństwie do innych przerwań BRK zapisuje stan PC zwiększony o 2, choć jest rozkazem 1-bajtowym.

Zastosowania: BRK wykorzystywane jest przede wszystkim przy uruchamianiu programów w JM w celu ustanawiania punktów przerwań i testowania poprzedzających je odcinków programów.

Klawisz BREAK wywołuje wykonanie BRK. Rozkaz ten powoduje czynności identyczne jak instrukcja STOP w Basicu.

`BVC` *branch if overflow clear* N V – B D I Z C

Opis: Wykonuje czynności analogiczne jak BCC, gdy znacznik nadmiaru V jest skasowany (V = 0). Gdy V = 1, wykonywany jest następny rozkaz.

Zastosowania: Tylko w arytmetyce ze znakiem. Umożliwia korygowanie wyników w przypadku nadmiaru.

`BVS` *branch if overflow set* N V – B D I Z C

Opis: Wykonuje czynności jak BCC, gdy znacznik V jest ustawiony (V = 1). W przeciwnym wypadku wykonywany jest następny rozkaz.

Zastosowania: Jak BVC.

`CLC` *clear carry* N V – B D I Z C

Opis: Kasuje znacznik przeniesienia C (C = 0) w rejestrze P.

Zastosowania: Niezbędny jest przed każdą operacją dodawania. W przypadku dodawania liczb wielobajtowych należy zastosować CLC tylko przed, wykonywanym jako pierwsze, dodaniem najmniej znaczących bajtów. 6502 nie ma rozkazu dodawania bez przeniesienia, stąd konieczność stosowania CLC przed ADC.

`CLD` *clear decimal mode* N V – B D I Z C

Opis: Kasuje znacznik trybu binarno-dziesiętnego BCD w rejestrze P wprowadzając tryb binarny dla wszystkich rozkazów ADC i SBC.

Zastosowania: Gdy stosujemy arytmetykę liczb binarnych, celowe jest wprowadzenie tego rozkazu raz na początku programu, by zapobiec przypadkowemu przejściu 6502 na tryb binarno-dziesiętny BCD i dezorganizacji obliczeń. Basic używa kodu BCD w reprezentowaniu liczb zmiennopozycyjnych.

`CLI` *clear interrupt disable* N V – B D I Z C

Opis: Kasuje znacznik zakazu przerwań I w rejestrze P. Oznacza to zezwolenie na wszelkie przerwania łącznie z maskowalnymi.

Zastosowania: CLI służy do przywrócenia normalnego trybu obsługi przerwań po czasowym jego wyłączeniu z pomocą SEI.

`CLV` *clear overflow* N V – B D I Z C

Opis: kasuje znacznik nadmiaru V (V = 0) w rejestrze P.

Zastosowania: W arytmetyce ze znakiem.

CMP *compare data and accumulator* N V – B D I Z C

Opis: Odejmuje dane od akumulatora (A – DANE), ale nie zapisuje wyniku, wpływa natomiast na znaczniki N, Z i C odpowiednio do tego, czy wynik jest dodatni, równy zeru lub ujemny. Wartość A pozostaje niezmieniona.

Z jest ustawiany, gdy oba bajty są sobie równe, kasowany w przeciwnym przypadku. N przybiera wartość bitu b7 wyniku, C jest ustawiany, gdy A >= DANE, kasowany, gdy A < DANE.

Zastosowania: Jest to ważny rozkaz, odgrywający kluczową rolę w konstrukcjach typu: IF … THEN, ON … GOTO i FOR … NEXT. Wraz z następującymi po nim rozkazami odgałęzień CMP umożliwia wybór alternatywnych dróg dalszego wykonywania programu, zależnie od wyniku porównania.

Często poprzednie czynności wpływają na znaczniki w sposób umożliwiający zastosowanie rozkazu odgałęzienia bez CMP.

Na przykład, rozkaz LDA #20 skasuje znaczniki N i Z, toteż zastosowanie BPL CEL zawsze spowoduje przeskok do CEL.

Jednakże gdy warunkiem odgałęzienia jest określona wartość niezerowa danych, można to ustalić tylko z pomocą CMP. Rozkazy odgałęzień następująco reagują skokiem na poszczególne relacje danych i zawartości akumulatora.

```
DANE >  A  BCC
DANE <= A  BCS
DANE =  A  BEQ
DANE <> A  BNE
```

Ograniczoną rolę po CMP spełniają BPL i BMI, ponieważ stan bitu b7 nie zawsze prawidłowo określa relację danych i A.

CPX *compare data and X* N V – B D I Z C

Opis: Odejmuje dane od rejestru X (X – DANE), ale nie zapisuje wyniku, wpływa natomiast na znaczniki N, Z i C, odpowiednio do wyniku porównania. Zawartość X pozostaje niezmieniona.

Zastosowania: CPX można stosować zamiennie z CMP wobec identycznego działania obu rozkazów. Można zatem wykorzystać informacje podane przy CMP zastępując oznaczenia A przez X.

Ponieważ rejestr X jest często wykorzystywany do sterowania pętlami i organizacji tablic, w tych dziedzinach CPX odgrywa znaczną rolę. Rozkaz ten, podobnie jak CMP, może być często pominięty, gdy poprzednie czyn-

ności określają wartości znaczników. Np. kolejne DEX w pętli doprowadzają w końcu do wyzerowania X i zakończenia działania BNE bez potrzeby stosowania CPX.

`CPY` *compare data and Y* N V – B D I Z C

Opis: Odejmuje dane od rejestru Y (Y – DANE), ale nie zapisuje wyniku, wpływa natomiast na znaczniki N, Z i C odpowiednio do wyniku porównania. Zawartość Y pozostaje niezmieniona.

Zastosowania: Takie same jak CPX. Rejestr Y jest częściej stosowany w wygodnych trybach adresowania pośredniego indeksowanego i wówczas CPY znajduje często zastosowanie.

`DEC` *decrement memory* N V – B D I Z C

Opis: Zmniejsza o 1 wartość bajtu pamięci zostawiając wynik w tym bajcie (M = M – 1). Wpływa na znaczniki N i Z. N i e wpływa na znacznik przeniesienia C.

Zastosowania: Użytecznie zastępuje SBC przy odejmowaniu niewielkich liczb. Na przykład (obok to samo w SBC):

```
DEC 2000          LDA 2000
DEC 2000          SEC
                  SBC #2
                  STA 2000
```

Pierwsze rozwiązanie jest krótsze i szybsze.

Z pomocą DEC i komórki pamięci, zwłaszcza na stronie zerowej, można zorganizować licznik pętli, gdy rejestry X i Y są przeciążone.

`DEX` *decrement X register* N V – B D I Z C

Opis: Zmniejsza o 1 zawartość rejestru X (X = X – 1). Nie wpływa na C.

Zastosowania: W organizowaniu pętli z pomocą rejestru X. Na przykład:

```
      LDX #20
CYKL  sekwencja rozkazów
      DEX
      BNE CYKL
```

`DEY` *decrement Y register* N V – B D I Z C

Opis: Zmniejsza o 1 zawartość rejestru Y (Y = Y – 1). Nie wpływa na C.

Zastosowania: Takie jak DEX, przy czym rejestr Y jest częściej stosowany ze względu na bardzo popularny tryb adresowania pośredniego postindeksowanego Y.

`EOR` *exclusive OR* N V – B D I Z C

Opis: Wykonuje logiczne ALBO (nierównoważność) na bitach danych i A. Bity wyniku przybierają wartość 1 tylko wtedy, gdy odpowiednie bity danych i A mają r ó ż n e wartości. 1 EOR 1 i 0 EOR 0 dają 0.

Zastosowania: Jednym z nich jest powodowanie zmiany wartości najwyższego bitu w kodzie znaku, co w Atari powoduje inwertowanie jego obrazu (zamianę barw znaku i tła). EOR jest użyteczne w technikach wyszukiwania z pomocą map bitowych.

`INC` *increment memory* N V – B D I Z C

Opis: Zwiększa o 1 wartość bajtu w pamięci (M = M + 1).

Zastosowania: Podobne jak DEC z tą różnicą, że odlicza w górę.

`INX` *increment X register* N V – B D I Z C

Opis: Zwiększa o 1 zawartość rejestru X (X = X + 1).

Zastosowania: Podobne jak DEX, z tym, że odlicza w górę.

`INY` *increment Y register* N V – B D I Z C

Opis: Zwiększa o 1 zawartość rejestru Y (Y = Y + 1).

Zastosowania: Podobne jak DEY z tym, że odlicza w górę.

`JMP` *jump to new location* N V – B D I Z C

Opis: Wykonuje skok bezwarunkowy w programie pod nowy adres. Nadaje licznikowi programu PC wartość podaną bezpośrednio lub pośrednio w operandzie. Jedyny rozkaz 6502 dostępny w trybie nieindeksowanym.

Zastosowania: Wykonywanie dalekich skoków w programie. Spełnia funkcje analogiczne jak GOTO w Basicu.

Skok pośredni może być użyteczny w przypadku wykorzystywania adresów systemu operacyjnego dostępnych jako wektory w określonych komórkach pamięci. W Atari jednym z wielu tego przykładów jest adres uruchomienia programu umieszczany w komórkach 2E0-2E1.

`JSR` *jump to subroutine* N V - B D I Z C

Opis: Wykonuje skok do podprogramu pod dowolny adres w pamięci. W celu zapewnienia powrotu z podprogramu adres następnego po JSR rozkazu pomniejszony dla uproszczenia pracy 6502 o 1 zostaje wstawiony na stos. Kończący podprogram obowiązkowy rozkaz RTS zdejmuje ten adres ze stosu i zwiększa o 1 , co zapewnia kontynuację programu.

Zastosowania: Jako bezpośredni równoważnik GOSUB Basicu rozkaz JSR znajduje szerokie zastosowanie, gdy sekwencja rozkazów może być wielokrotnie użyta w programie. JSR pozwala także wykorzystywać podprogramy systemu operacyjnego, np. wykonujące wyświetlanie znaków oraz ich odczytywanie z klawiatury.

Jeżeli po zakończeniu podprogramu chcemy wykonać skok do innego miejsca w programie, należy pamiętać o zdjęciu ze stosu adresu umieszczonego tam przez JSR z pomocą PLA, PLA. W przeciwnym wypadku można łatwo przepełnić stos i spowodować załamanie się programu.

`LDA` *load accumulator* N V - B D I Z C

Opis: Ładuje czyli wpisuje bajt danych do akumulatora.

Zastosowania: LDA jest jednym z najszerzej stosowanych rozkazów przesłania danych. Wynika to z roli akumulatora jako najważniejszego rejestru CPU, za którego pośrednictwem następuje na ogół kierowanie danych do ALU i odbiór wyników.

Duża rola LDA wynika z braku możliwości bezpośredniego przesłania danych pamięć-pamięć i konieczności posłużenia się pośrednictwem akumulatora, jakkolwiek dzięki symetrii rozkazów można do tego wykorzystać również rejestry X i Y.

`LDX` *load X register* N V - B D I Z C

Opis: Ładuje czyli wpisuje bajt danych do rejestru X.

Zastosowania: Podobne jak w przypadku LDA, ponieważ X może spełniać część funkcji akumulatora, a jednocześnie jest przydatny jako licznik pętli i indeks tablic.