```
0801: 00
0802: 00
0803: 00
0804: FF    (Nowe dane zastąpiły poprzedni wpis)
0805: A7
```

8.5. Lista rozkazów oraz tryby adresowania jednostki CPU12

Komputery doskonale radzą sobie z wykonywaniem prostych instrukcji, takich jak szybkie i dokładne dodawanie liczby zapisanej w danym miejscu pamięci do zawartości określonego rejestru. Komputery są zdolne do bardzo wyrafinowanych i pozornie inteligentnych zachowań dzięki wykonywaniu sekwencji instrukcji zwanych **programami** lub **oprogramowaniem**. Programy te są przygotowywane przez człowieka – programistę.

Znaczna część wysiłku związanego z projektowaniem sterownika opartego na mikrokontrolerze polega na pisaniu oprogramowania.

Niestety, nawet najmniejsze niedopatrzenie ze strony programisty może sprawić, że program stanie się bezużyteczny, dopóki błąd nie zostanie naprawiony. Znaczna część wysiłku związanego z projektowaniem sterownika opartego na mikrokontrolerze polega na pisaniu oprogramowania. Aby było ono skuteczne, programista musi znać wszystkie szczegóły zestawu instrukcji dla używanego MCU. Naszym celem w tym i następnym podrozdziale jest przedstawienie krótkiego przeglądu, a nie uczynienie z czytelnika eksperta programisty.

Ogólnie rzecz biorąc, zestawy instrukcji są podobne w różnych typach MCU, ale różnią się szczegółami. Po opanowaniu programowania danej maszyny o wiele łatwiej jest nauczyć się i dobrze wykorzystać zestaw instrukcji innego procesora. Również w tym przypadku za przykład posłuży procesor CPU12. Więcej szczegółów na jego temat można łatwo znaleźć w Internecie.

Instrukcje procesora CPU12

(A) reprezentuje *zawartość* rejestru A.

Wybrany zestaw instrukcji dla CPU12 przedstawiono w tabeli 8.1. W pierwszej kolumnie w tabeli znajduje się kod mnemoniczny każdej instrukcji, a druga kolumna zawiera krótki opis i równoważne wyrażenie logiczne dla danej instrukcji. Na przykład, instrukcja ABA dodaje zawartość rejestru B do zawartości rejestru A, a wynik znajduje się w rejestrze A. Operację tę możemy oznaczyć jako

$$(A) + (B) \rightarrow A$$

co pokazano w drugiej kolumnie tabeli.

Mnemoniki są łatwe do zapamiętania dla ludzi. Jednak w pamięci mikrokomputera instrukcje są przechowywane jako kody maszynowe lub kody operacyjne (tzw. *op codes* – przyp. tłum.) składające się z jednej lub więcej liczb 8-bitowych, z których każda jest przedstawiona w tabeli jako dwucyfrowa liczba szesnastkowa. Na przykład w wierszu dotyczącym instrukcji ABA widzimy, że kodem op jest 1806. Zatem instrukcja ABA pojawia się w pamięci jako liczby binarne 00011000 i 00000110.

Tabela 8.1. Wybrane instrukcje procesora CPU12

Kod mnemoniczny	Wykonywana operacja	Tryb adresowania	Kod maszynowy	Kody stanu							
				S	X	H	I	N	Z	V	C
ABA	Dodaje akumulatory (A) + (B) → A	INH	18 06	-	-	↕	-	↕	↕	↕	↕
ADDA (opr)	Dodaje zawartość pamięci do A (A) + (M) → A	IMM DIR EXT IDX	8B ii 9B dd BB hh ll AB *	-	-	↕	-	↕	↕	↕	↕
ADDB (opr)	Dodaje zawartość pamięci do B (B) + (M) → B	IMM DIR EXT IDX	CB ii DB dd FB hh ll EB *	-	-	↕	-	↕	↕	↕	↕
ADDD (opr)	Dodaje zawartość pamięci do D (D) + (M: M+1) → D	IMM DIR EXT IDX	C3 jj kk D3 dd F3 hh ll E3 *	-	-	-	-	↕	↕	↕	↕
BSC (rel)	Rozgałęzienie jeśli wystąpiło przeniesienie (C = 1)	REL	25 rr	-	-	-	-	-	-	-	-
BEQ (rel)	Rozgałęzienie jeśli wystąpiła równość (Z = 1)	REL	27 rr	-	-	-	-	-	-	-	-
BLO (rel)	Rozgałęzienie jeśli wystąpiło mniejsze^{U} (C = 1)	REL	25 rr	-	-	-	-	-	-	-	-
BMI (rel)	Rozgałęzienie jeśli minus^{s} (C = 1)	REL	2B rr	-	-	-	-	-	-	-	-
BNE (rel)	Rozgałęzienie jeśli nie równe (Z = 0)	REL	26 rr	-	-	-	-	-	-	-	-
BPL (rel)	Rozgałęzienie jeśli plus^{s} (C = 1)	REL	2A rr	-	-	-	-	-	-	-	-
BRA (rel)	Rozgałęzienie zawsze	REL	20 rr	-	-	-	-	-	-	-	-
CLRA	Wyzeruj akumulator A $00 → A	INH	87	-	-	-	-	0	1	0	0
CLRB	Wyzeruj akumulator B $00 → B	INH	C7	-	-	-	-	0	1	0	0
COMA	Zaneguj bity akumulatora A	INH	41	-	-	-	-	↕	↕	0	1
INCA	Inkrementuj akumulator A (A) + $01 → A	INH	42	-	-	-	-	↕	↕	↕	-
INCB	Inkrementuj akumulator B (B) + $01 → B	INH	52	-	-	-	-	↕	↕	↕	-
INX	Inkrementuj rejestr indeksowy X (X) + $0001 → X	INH	08	-	-	-	-	-	↕	-	-
JMP (opr)	Rozkaz skoku Adres procedury → PC	EXT IDX	06 hh ll 05 *	-	-	-	-	-	-	-	-
JSR (opr)	Skok do podprogramu (wyjaśnienie w tekście)	DIR EXT IDX	17 dd 16 hh ll 15 *								
LDAA (opr)	Załaduj zawartość pamięci do akumulatora A (M) → A	IMM DIR EXT IDX	86 ii 96 dd B6 hh ll A6 *	-	-	-	-	↕	↕	0	-
LDAB (opr)	Załaduj zawartość pamięci do akumulatora B (M) → B	IMM DIR EXT IDX	C6 ii D6 dd F6 hh ll E6 *	-	-	-	-	↕	↕	0	-
LDD (opr)	Załaduj zawartość pamięci do akumulatora D (M): (M+1) → D	IMM DIR EXT IDX	CC jj kk DC dd FC hh ll EC *	-	-	-	-	↕	↕	0	-
LDX (opr)	Załaduj zawartość pamięci do rejestru indeksowego X (M): (M+1) → X	IMM DIR EXT IDX	CE jj kk DE dd FE hh ll EE *	-	-	-	-	↕	↕	0	-

Tabela 8.1. cd.

Kod mnemoniczny	Wykonywana operacja	Tryb adresowania	Kod maszynowy	Kody stanu							
				S	X	H	I	N	Z	V	C
LDY (opr)	Załaduj zawartość pamięci do rejestru indeksowego Y (M): (M+1) → Y	IMM DIR EXT IDX	CD jj kk DD dd FD hh ll ED *	-	-	-	-	↕	↕	0	-
MUL	Pomnóż A przez BU (A) × (B) → D	INH	12	-	-	-	-	-	-	-	
NOP	Brak operacji (cykl jałowy – przyp. Tłum.)	INH	A7	-	-	-	-	-	-	-	-
PSHA	Zapisz akumulator A na stosie (SP) – 1 ⇒ SP; (A) ⇒ $M_{(SP)}$	INH	36	-	-	-	-	-	-	-	-
PSHB	Zapisz akumulator B na stosie (SP) – 1 ⇒ SP; (B) ⇒ $M_{(SP)}$	INH	37	-	-	-	-	-	-	-	-
PSHX	Zapisz rejestr X na stosie (SP) – 2 ⇒ SP; (X_H:X_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$	INH	34	-	-	-	-	-	-	-	-
PSHY	Zapisz rejestr Y na stosie (SP) – 2 ⇒ SP; (Y_H:Y_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$	INH	35	-	-	-	-	-	-	-	-
PULA	Pobierz ze stosu do akumulatora A ($M_{(SP)}$) ⇒ A; (SP) + 1 ⇒ SP	INH	32	-	-	-	-	-	-	-	-
PULB	Pobierz ze stosu do akumulatora B ($M_{(SP)}$) ⇒ B; (SP) + 1 ⇒ SP	INH	33	-	-	-	-	-	-	-	-
PULX	Pobierz ze stosu do rejestru X ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ X_H:X_L; (SP) + 2 ⇒ SP	INH	30	-	-	-	-	-	-	-	-
PULY	Pobierz ze stosu do rejestru Y ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ Y_H:Y_L; (SP) + 2 ⇒ SP	INH	31	-	-	-	-	-	-	-	-
RTS	Powrót z podprogramu ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ PC; (SP) + 2 ⇒ SP	INH	3D	-	-	-	-	-	-	-	-
STAA (opr)	Zapisz akumulator A w pamięci (A) → M	DIR EXT IDX	5A dd 7A hh ll 6A *	-	-	-	-	↕	↕	0	-
STAB (opr)	Zapisz akumulator B w pamięci (B) → M	DIR EXT IDX	5B dd 7B hh ll 6B *	-	-	-	-	↕	↕	0	-
STD (opr)	Zapisz akumulator D w pamięci (A) → M; (B) → M + 1	DIR EXT IDX	5C dd 7C hh ll 6C *	-	-	-	-	↕	↕	0	-
STOP	Zatrzymaj zegary wewnętrzne. Jeśli bit kontrolny S = 1, instrukcja STOP jest wyłączona i działa jak NOP	INH	18 3E	-	-	-	-	-	-	-	-
TSTA	Test akumulatora A; (A) – 00	INH	97	-	-	-	-	↕	↕	0	0
TSTB	Test akumulatora A; (A) – 00	INH	D7	-	-	-	-	↕	↕	0	0

S oznacza instrukcje przeznaczone dla liczb w kodzie uzupełnienia do dwóch ze znakiem
U oznacza instrukcje przeznaczone dla liczb bez znaku
* Dla adresowania indeksowanego (IDX). Podawany jest tylko pierwszy bajt kodu maszynowego. Potrzebne są dodatkowe bajty od jednego do trzech. (Szczegóły wykraczają poza zakres naszej dyskusji.)
ii 8-bitowe dane natychmiastowe
dd niski bajt adresu bezpośredniego
hh ll wysoki i niski bajt adresu rozszerzonego
jj kk wysoki i niski bajt 16-bitowych danych natychmiastowych
rr 8-bitowe przesunięcie ze znakiem w instrukcji rozgałęzienia

Spójrzmy na wiersz dla instrukcji ADDA(opr), w którym (opr) oznacza adres miejsca w pamięci. Działanie tej instrukcji polega na dodaniu zawartości miejsca w pamięci do zawartości akumulatora A, a wynik zostanie zapisany w A. Przedstawia to wyrażenie

$$(A) + (M) \rightarrow A,$$

w którym (M) reprezentuje zawartość znajdującą się pod danym adresem w pamięci. W celu wybrania adresu pamięci, do którego mają mieć dostęp niektóre instrukcje, można użyć kilku **trybów adresowania**. Na przykład instrukcja ADDA może korzystać z jednego z kilku trybów adresowania. Tryby adresowania procesora CPU12 omówimy wkrótce.

Tabela 8.1 pokazuje również wpływ każdej instrukcji na zawartość CCR (rejestru kodu stanu). Znaczenie symboli przedstawionych dla każdego bitu kodu stanu jest następujące:

- \- bit nie jest zmieniany przez tę instrukcję,
- 0 bit jest zawsze kasowany przez tę instrukcję,
- 1 bit jest zawsze ustawiany przez tę instrukcję,
- ↕ bit jest ustawiany lub kasowany w zależności od wyniku.

CPU12 ma o wiele więcej instrukcji niż te wymienione w tabeli; podaliśmy tu tylko przykład różnych rodzajów. Następnie krótko opiszemy każdy z trybów adresowania używanych przez CPU12.

Adresowanie rozszerzone (EXT)

Przypomnijmy, że procesor CPU12 używa 16 bitów (zwykle zapisywanych jako cztery cyfry heksadecymalne) do adresowania pamięci. W adresowaniu rozszerzonym w instrukcji zawarty jest pełny adres operandu. Tak więc instrukcja

```
ADDA $CA01
```

dodaje zawartość znajdującą się pod adresem CA01 w pamięci do zawartości rejestru A. (Później zobaczymy, że do zamiany kodów mnemonicznych na kody operacji używany jest program zwany asemblerem. Znak $ wskazuje asemblerowi, że adres jest podany w postaci szesnastkowej). Kody operacyjne pojawiają się w trzech kolejnych miejscach pamięci jako:

- BB (kod operacyjny dla ADDA z adresowaniem rozszerzonym),
- CA (starszy bajt adresu),
- 01 (młodszy bajt adresu).

Zauważmy, że najpierw podawany jest starszy bajt adresu, a następnie młodszy bajt.

Adresowanie bezpośrednie (DIR)

W **adresowaniu bezpośrednim** podaje się tylko dwie najmniej znaczące cyfry adresu (w systemie szesnastkowym), a dwie najbardziej znaczące cyfry przyjmuje się jako zero. Dlatego efektywny adres mieści się w przedziale od 0000 do 00FF. Na przykład instrukcja

```
ADDA $A9
```

dodaje zawartość komórki o adresie 00A9 w pamięci do zawartości rejestru A. Instrukcja ta pojawia się w dwóch kolejnych miejscach pamięci jako