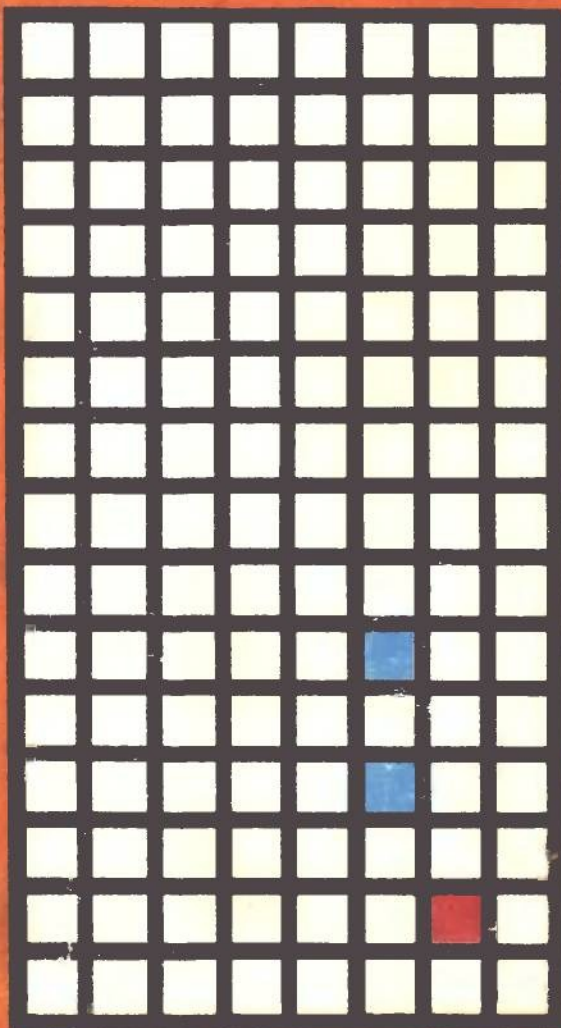


MIKROPROCESSOR 6502

*i jego
rodzina*





Scanned by Gozdek
Classic Computers OnLine
www.cconline.com.pl



**Henryk Kruszyński
Krzysztof Kulpa**

MIKROPROCESOR 6502 *i jego rodzina*

Scanned by Gozdek
Classic Computers OnLine
www.cconline.com.pl

Wydawnictwo Czasopism
i Książek Technicznych



Warszawa 1989

Okladkę i stronę tytułową projektował *Andrzej Radziejewski*
Redaktor *Andrzej Markowski*
Redaktor techniczny *Władysław Nasternak*
Korektor *Anna Awilowska*

W książce przedstawiono rozwiązania układowe rodziny mikroprocesorów 65XX. Omówiono także układy wspomagające systemy mikrokomputerowe oparte na tych mikroprocesorach. Książka jest przeznaczona dla wszystkich interesujących się techniką mikroprocesorową.

© Copyright by Wydawnictwo Czasopism i Książek Technicznych NOT-SIGMA,
Warszawa 1989

ISBN 83-85001-58-1

Scanned by Gozdek
Classic Computers OnLine
www.cconline.com.pl

Spis treści

1. Wstęp	7
Część I. Mikroprocesor	11
2. Informacje ogólne o mikroprocesorze	11
3. Lista rozkazów	15
3.1. Wstępne wiadomości o rozkazach	15
3.2. Rozkazy przesłań	15
3.3. Rozkazy arytmetyczne	17
3.4. Rozkazy dodatkowe	19
3.5. Rozkazy działań logicznych	20
3.6. Rozkazy przesunięć	21
3.7. Rozkazy sterujące	23
3.8. Rozkazy związane z podprogramami	26
3.9. Obsługa przerw	31
3.10. Tryby adresowania	33
3.11. Uwagi dodatkowe	39
4. Budowa i działanie mikroprocesora	42
4.1. Zasilanie	42
4.2. Zegar	43
4.3. Magistrale	46
4.4. Przerwania i zerowanie	47
4.5. Sygnały dodatkowe	49
4.6. Cykl rozkazowy	50
4.7. Inne typy mikroprocesorów	52
Część II. Mikrokomputer	53
5. Organizacja mikrokomputera	53

6. Elementy składowe mikrokomputera	58
6.1. Dołączanie pamięci stałej (ROM)	58
6.2. Dołączanie pamięci zapisywalnej (RWM)	63
6.2.1. Dołączanie pamięci statycznej	63
6.2.2. Dołączanie pamięci dynamicznej	66
6.3. Układy wejścia i wyjścia	73
6.3.1. Układ równoległego wejścia i wyjścia	73
6.3.2. Układ szeregowego wejścia i wyjścia	96
6.3.3. Moduł wytwarzania sygnału wizyjnego	103
6.3.4. Inne układy peryferyjne	108
6.4. Obsługa przerwań	109
6.5. Buforowanie	114
7. Wskazówki dotyczące uruchamiania i diagnozowania mikrokomputera	119
7.1. Pułapka	119
7.2. Praca krokowa	120
7.3. Testowanie oscyloskopem	122
8. Zakończenie	124
Dodatki	
1. Lista rozkazów zestawiona alfabetycznie według mnemoników	126
2. Lista rozkazów zestawiona według kodów szesnastkowych	128
3. Schematy cykli rozkazowych	130
4. Arytmetyka	147
5. Różnice między mikroprocesorem 6502 a 65C02	150
6. Przykłady programów	151
7. Przykłady porównania szybkości pracy różnych mikroprocesorów	154
8. Rozkład wyprowadzeń wybranych układów scalonych rodziny 65XX	157
Literatura	159

Oddajemy w Twoje ręce, Czytelniku, książkę poświęconą mikroprocesorowi 6502 i jego rodzinie. Powstał on w 1975 r. w amerykańskiej firmie MOS Technology i jest w chwili obecnej najbardziej chyba rozpowszechnionym w świecie zachodnim procesorem 8-bitowym.

W Polsce, gdzie najczęściej są używane wyroby firmy Intel lub ich odpowiedniki, mikroprocesor 6502 jest mniej znany.

W momencie powstania procesora 6502 rynek mikroprocesorowy był podzielony między dwie firmy: Intel z procesorem 8080 i Motorola z procesorem 6800. Aby móc się dostać na ten rynek, procesor 6502 musiał być lepszy od swoich konkurentów — i takim też był (i jest nadal). Zespół projektantów tego procesora (a byli w nim również projektanci, którzy poprzednio zajmowali się procesorem MC 6800) dokładnie określił, jakie warunki ma spełniać nowy mikroprocesor.

Po pierwsze, miał to być procesor tani, a zarazem bardzo szybki. Wymagania takie wynikały z przewidywanych głównych dziedzin zastosowania, a były nimi automatyka, kontrola i sterowanie oraz „inteligentne” przyrządy i aparatura; do zastosowań tych powstały też wersje z ograniczoną do 4 KB przestrzenią adresową — w obudowie DIL 28.

Po drugie, miał to być procesor łatwy do zastosowania, a więc łatwy do poznania i zrozumienia, a także dobrze pasujący do istniejących opracowań (dlatego też powstała wersja 6501, która była sygnałowo całkowicie zgodna z procesorem 6800, tak że procesor 6501 można było stosować w konstrukcjach projektowanych dla MC 6800, a jedyną wymaganą zmianą była zmiana oprogramowania).

W wyniku prac projektowych powstał procesor 8-bitowy z 16-bitową magistralą adresową (jako chyba jedyny procesor całkowicie 8-bitowy, bez

rejestrów 16-bitowych, poza licznikiem rozkazów, i bez żadnych rozkazów 16-bitowych), częściowo wzorowany na procesorze 6800, lecz wolny od wszystkich jego wad. W procesorze tym zastosowano kilka rewelacyjnych, jak na tamte lata, rozwiązań powodujących uproszczenie budowy procesora, a jednocześnie zwiększających jego szybkość.

Pierwszym z tych rozwiązań była nowoczesna architektura realizująca pracę potokową, „na zakładkę” (ang. *pipe-line*). Polega to na rozdzieleniu funkcji zewnętrznych i wewnętrznych procesora. Funkcje zewnętrzne (odczyt i zapis pamięci) wyprzedzają funkcje wewnętrzne (dekodowanie rozkazów i wykonywanie obliczeń) o kilka (od jednego do trzech) taktów zegara, co powoduje znaczne przyspieszenie pracy procesora. Mówiąc prościej, procesor czyta pewne dane „na zapas”, nie wiedząc, czy będą mu potrzebne (w jaki sposób to robi, dowiesz się, Czytelniku, w rozdz. 4). Gdy dane okazują się potrzebne, procesor już je ma i nie musi tracić czasu na ich czytanie.

Drugim świetnym rozwiązaniem było zastosowanie dość małej liczby rozkazów (tylko 56). Z analizy innych rozwiązań, w tym z analizy działania dużych komputerów, wynikało, że zastosowanie dużej liczby różnorodnych rozkazów tylko pozornie zwiększa moc obliczeniową procesora. W praktyce z większości złożonych rozkazów korzysta się niezmiernie rzadko, a nawet wcale, tak że praktycznie jedynym efektem istnienia tych rozkazów jest rozbudowa (i to często dość znaczna) samego procesora. To, co naprawdę jest przydatne, to duża liczba sposobów adresowania (a więc łatwość manipulowania adresem, a przez to danymi), których to sposobów w naszym procesorze programista ma do dyspozycji aż 13 (w niektórych wersjach nawet 15). Pod tym względem przez długie lata procesor 6502 był nie do pokonania i dopiero procesor 6809 ma ich więcej.

Trzecim rozwiązaniem, może trochę bardziej kontrowersyjnym, było zastosowanie zaledwie 3 rejestrów wewnętrznych procesora (akumulator i dwa rejestry indeksowe *X* i *Y*), tylko 8-bitowych. Rozwiązanie to upraszcza budowę samego procesora, rezygnacja zaś z rejestrów 16-bitowych przyspiesza znacznie procesor (ze względu na krótszą propagację przeniesienia we wszystkich operacjach arytmetycznych). Kontrowersyjność tego rozwiązania polega na tym, że duża liczba rejestrów, szczególnie 16-bitowych, jest bardzo przydatna przy manipulacjach na dużych obszarach danych, co ma miejsce zwłaszcza przy stosowaniu języków wyższych poziomów, ale nie do tego był projektowany ten mikroprocesor.

Czwartym wreszcie rozwiązaniem była rezygnacja z wyodrębnienia w pamięci oddzielnego obszaru wejścia-wyjścia (co ma miejsce w procesorach 8080 i Z80). Umieszczenie urządzeń wejścia-wyjścia w jednolitej przestrzeni adresowej ma dwie istotne zalety: upraszcza budowę procesora, a także pozwala na stosowanie wszystkich rozkazów procesora w operacjach wejścia i wyjścia.

Procesor całkowicie spełnił pokładane w nim nadzieje i w krótkim czasie podbił rynek mikrokomputerowy. Było to spowodowane, poza wymienionymi

już zaletami, bardzo małą liczbą elementów potrzebnych do stworzenia mikrokomputera – wystarczyły cztery elementy: rezonator kwarcowy, procesor, układ COMBO – np. 6530 – zawierający pamięci RWM (RAM) i ROM, układ wejścia-wyjścia oraz licznik-czasomierz (ang. *timer*) i kilka bramek logicznych, np. układ 7400, oraz jedno napięcie zasilania (+5 V). Szybko okazało się, że procesor, projektowany do prostych zastosowań, również bardzo dobrze nadaje się do systemów dużych. Powstała cała gama komputerów biurowych i osobistych wykorzystująca ten procesor, a w tym komputery serii Apple, traktowane jako sprzęt profesjonalny używany do obliczeń naukowych, komputery edukacyjne BBC-micro, a także komputery domowe firm Commodore i Atari, poczynając od najprostszego – PET, przez VIC20 i C64, Atari 600 i Atari 800, kończąc na C128. Dopiero wprowadzenie na rynek procesorów 16-bitowych (Intel 8086, Motorola 68000 i innych) ograniczyło zastosowanie procesora 6502 w mikrokomputerach uniwersalnych, nie ograniczając jednak zastosowań procesorów rodziny 65XX w układach automatyki i sterowania.

Niniejsza książka, prezentująca rodzinę procesorów 65XX, składa się z dwóch części. W pierwszej jest omówiony mikroprocesor 6502, jego budowa, zasady działania i sposoby programowania. Część druga, wymagająca od Czytelnika pewnego zasobu wiedzy z dziedziny elektroniki, a szczególnie układów logicznych (cyfrowych), omawia budowę mikrokomputera, którego sercem jest procesor z rodziny 65XX, a także działanie poszczególnych jego bloków i sposób programowania.

Książka ta może być przydatna osobom, które samodzielnie pragną skonstruować mikrokomputer lub tylko dokładnie zrozumieć szczegóły budowy gotowych komputerów.

Jeśli któryś z rozdziałów wyda Ci się, Czytelniku, trudny, niezbyt zrozumiały lub nudny, opuść go i czytaj dalej. Na pewno stanie się zrozumiały, gdy wrócisz do niego ponownie, bogatszy o przeczytane inne rozdziały.

Książka jest wydrukowana jednakową czcionką: taką, jak ta właśnie. Jednak w niektórych miejscach pojawia się taki rodzaj druku. Oznacza on miejsca, dotyczące jednego z wariantów mikroprocesora 6502, mianowicie wersji CMOS oznaczonej 65C02. Te miejsca można oczywiście opuścić, chyba że masz Czytelniku właśnie do czynienia z 65C02 i chcesz znać wszystkie jego właściwości.

W książce przyjęto oznaczenia sposobów zapisu liczb:

- dziesiętnego – bez dodatkowych znaków, np. 12,
- szesnastkowego – ze znakiem \$ przed liczbą, np. \$0C,
- dwójkowego – ze znakiem % przed liczbą, np. %1100.

(Na rysunkach, dla ich większej przejrzystości, pominięto te dodatkowe znaki).

Ponadto tzw. argument bezpośredni (co to jest, wyjaśnimy dalej) będzie oznaczany dodatkowo znakiem #. Nawiasy w części adresowej rozkazu

oznaczają adresowanie pośrednie, np. (ABS) oznacza adres wzięty z komórek o numerach ABS i ABS+1. Oznaczenie ZPG dotyczy komórki na stronie zerowej; w przypadku adresowania pośredniego indeksowego po przecinku podaje się nazwę rejestru (X lub Y), którego zawartość służy do modyfikacji adresu argumentu, np. (ZPG), X.

Do takich oznaczeń warto się przyzwyczaić, gdyż są one stosowane w asemblerach przeznaczonych dla procesora 6502.

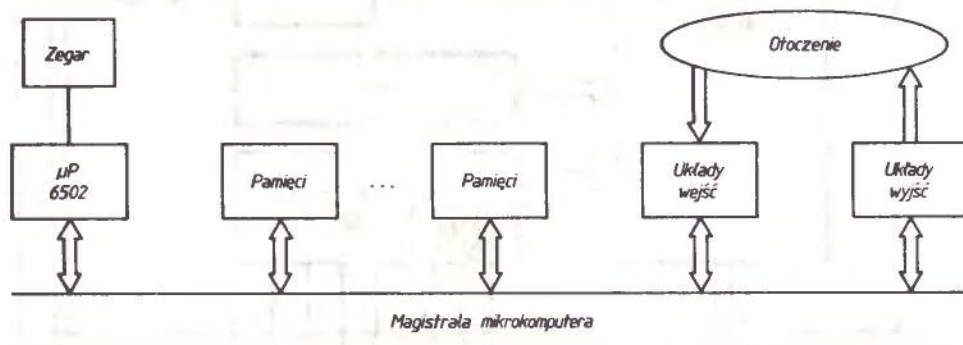
MIKROPROCESOR

W pierwszej części książki jest omówiona budowa i działanie mikroprocesora 6502 oraz innych jego wariantów. Pominięto szczegóły technologiczne, natomiast wyjaśniono dokładnie działanie wszystkich rozkazów i sygnałów mikroprocesora. Czytelnicy zainteresowani tylko programowaniem mogą opuścić (przynajmniej na początku) prawie cały rozdz. 4, natomiast programiści, piszący programy silnie uzależnione czasowo, powinni dokładnie przestudiować p. 3.11.

2

Informacje ogólne o mikroprocesorze

Zanim przyjrzymy się dokładniej strukturze mikroprocesora, musimy uświadomić sobie jego miejsce w mikrokomputerze. Rysunek 2.1 pokazuje typową strukturę mikrokomputera. Składa się on z mikroprocesora, kilku bloków pamięci oraz urządzeń wejścia i wyjścia. Elementy składowe mikrokomputera są ze sobą połączone za pomocą zespołu połączeń zwanych **magistralą**. Szczegóły budowy magistrali i innych elementów mikrokomputera są opisane w dalszych rozdziałach. Funkcje tych elementów są dość oczywiste: urządzenia wejścia i wyjścia służą do kontaktowania się mikrokomputera z otoczeniem (np. z człowiekiem albo z silnikiem samochodu, który jest sterowany przez mikrokomputer), a pamięci służą do przechowywania programów i danych.



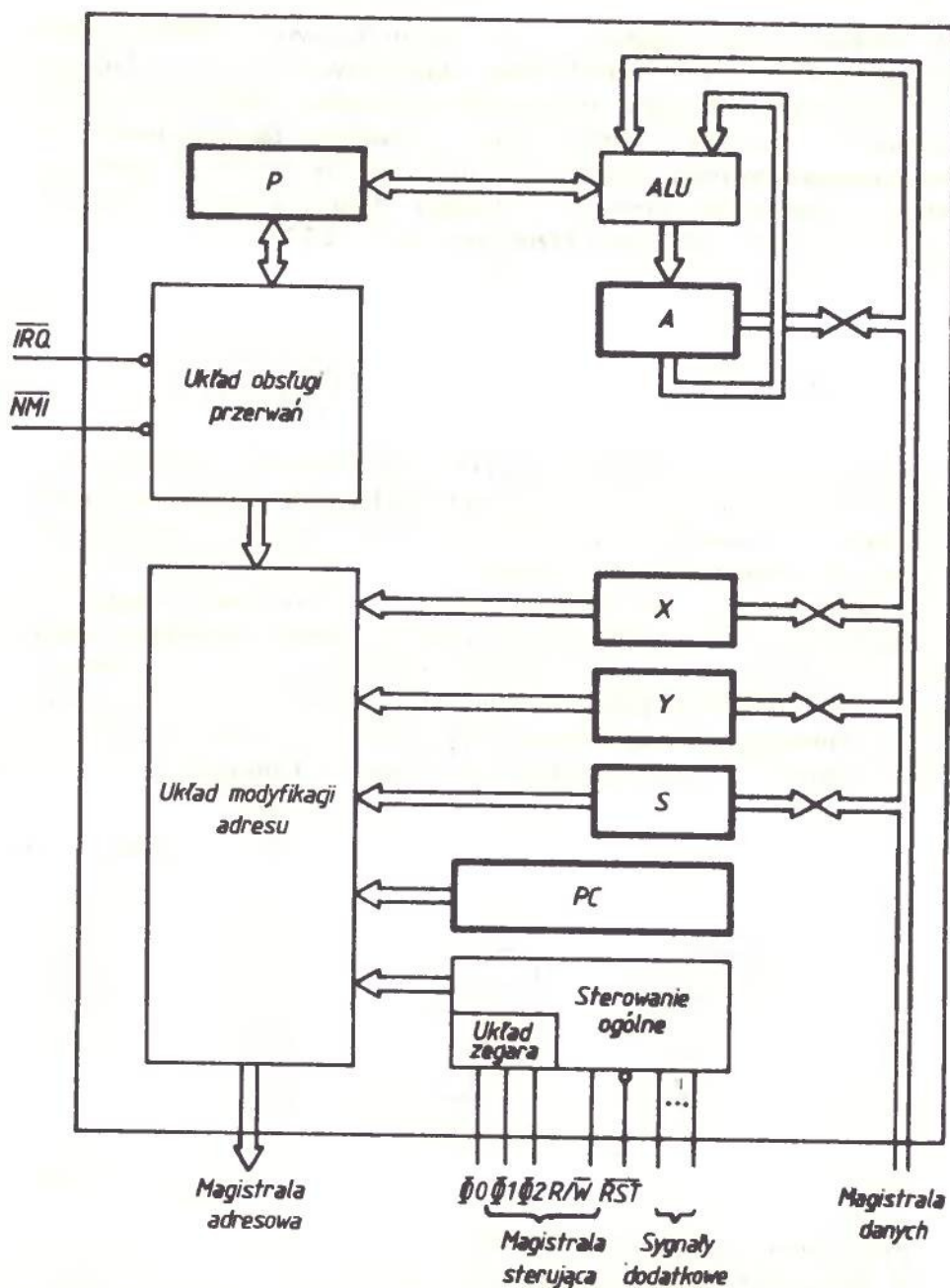
Rys. 2.1. Struktura mikrokomputera

Pracą całości steruje mikroprocesor.

W kolejnych rozdziałach zajmiemy się mikroprocesorem 6502 traktując go szczegółowo bądź z punktu widzenia programisty (a zaniedbując inne aspekty),

bądź z punktu widzenia sprzętu, tj. współpracy z innymi elementami mikrokomputera (nie mówiąc już tak wiele o programowaniu).

Mikroprocesor 6502 jest mikroprocesorem prawdziwie 8-bitowym. Wszystkie jego rejestry (z jednym wyjątkiem) i jednostka arytmetyczno-lo-



Rys. 2.2. Schemat funkcjonalny mikroprocesora 6502

giczna są 8-bitowe. Z otoczeniem komunikuje się on za pomocą 8-bitowej dwukierunkowej magistrali danych, 16-bitowej jednokierunkowej magistrali adresowej i 3-bitowej (lub 2-bitowej) magistrali sterującej.

Spójrzmy teraz na rys. 2.2. Przedstawia on schemat funkcjonalny mikroprocesora 6502. Jest to schemat celowo uproszczony: przedstawiono tu tylko rejestry dostępne dla programisty (tj. te, do których może wpisywać i odczytywać dane) – pogrubioną linią. Aby nie gmatwać rysunku, opuszczono wiele dróg przesyłania informacji, pozostawiając tylko najważniejsze połączenia. Cienką linią są zaznaczone układy sterujące i inne, które umożliwiają właściwą pracę mikroprocesora. Są one bardzo ważne, ale programiście wystarczy skupić uwagę tylko na rejestrach.

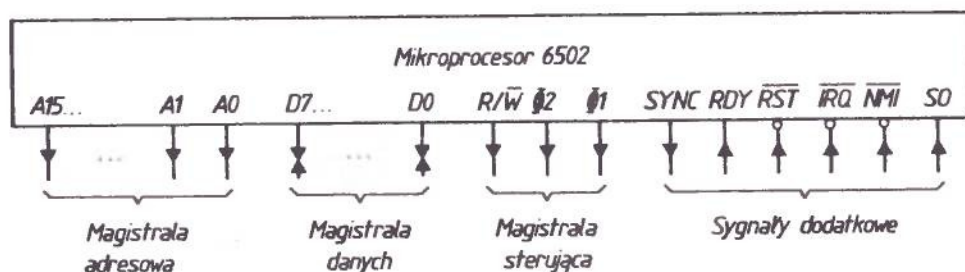
Głównym rejestrem mikroprocesora 6502 jest **akumulator A** (8-bitowy). Służy do przechowywania danych (np. pobranych z pamięci) oraz do wykonywania operacji na tych danych. Z akumulatorem jest związana **jednostka arytmetyczno-logiczna** (ang. *ALU* – *Arithmetic-Logic Unit*) wraz z rejestrem znaczników *P* (ang. *Processor status register*).

Ponadto procesor 6502 ma **rejestry indeksowe X i Y** (8-bitowe) używane do realizacji różnych trybów adresowania. Mogą one też być używane jako rejestry pomocnicze do przechowywania danych, odliczania pętli itp.

Rejestr stosu S (8-bitowy) służy do adresowania pamięci stosu, która znajduje się na stronie 1 (adresy \$01FF ÷ \$0100).

Od tej chwili często będziemy stosowali zapis szesnastkowy. Liczba \$01FF oznacza $0 \cdot 16^3 + 1 \cdot 16^2 + 15 \cdot 16^1 + 15 \cdot 16^0 = 511$. Liczba \$0100 oznacza $0 \cdot 16^3 + 1 \cdot 16^2 + 0 \cdot 16^1 + 0 \cdot 16^0 = 256$. Przypominamy, że \$A = 10, \$B = 11, \$C = 12, \$D = 13, \$E = 14, \$F = 15. Zechciej sprawdzić, Czytelniku, że %10100011 = \$A3 = 163 oraz %11111100 = \$FC = 252.

Licznik rozkazów PC (ang. *Program Counter*) ma 16 bitów i wskazuje na komórkę pamięci, w której znajduje się wykonywany aktualnie rozkaz,



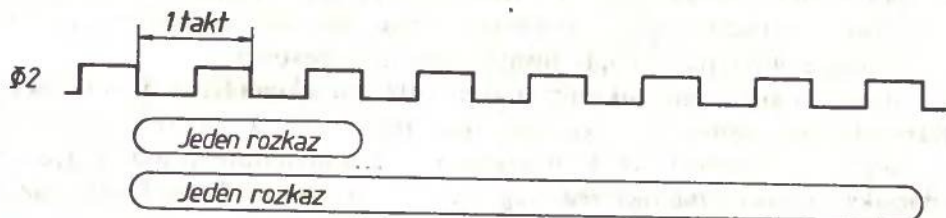
Rys. 2.3. Sygnały mikroprocesora 6502

Programista ma tylko pośredni dostęp do tego rejestru (będzie to omówione dalej). Magistrale i inne sygnały, przedstawione szkicowo na rys. 2.2, są przedstawione w pełni na rys. 2.3.

Magistrala adresowa jest 16-bitowa, o sygnałach ponumerowanych od A0 do A15. Umożliwia to zaadresowanie $2^{16} = 65536 = \$10000$ komórek pamięci, ponumerowanych od 0 do 65535 (tj. od \$0 do \$FFFF).

Ośmiobitowa magistrala danych, o sygnałach oznaczonych D0...D7, umożliwia dwukierunkowe przesyłanie danych między mikroprocesorem a innymi elementami mikrokomputera.

Rolą **magistrali sterujących** jest sterowanie kierunkiem przepływu danych oraz wyznaczanie chwil wykonania operacji na magistrali danych.



Rys. 2.4. Sygnał zegara i cykle rozkazowe

Są poza tym sygnały dodatkowe: wejścia przerwań, gotowości, zerowania mikroprocesora oraz wyjście sygnalizujące początek wykonania każdego rozkazu (ściślej: pobieranie kodu rozkazu z pamięci). Wszystkie te sygnały są omówione dokładnie w rozdz. 4. Tutaj zwróćmy tylko uwagę na jedną cechę pracy magistrali (i mikroprocesora): jeden takt wyznaczany przez sygnał zegara $\Phi 2$ (liczony od tylnego zbocza sygnału do następnego tylnego zbocza) – jest **cyklem maszynowym** (najmniejszym przedziałem czasu, w którym wykonują się elementarne operacje). Kilka takich kolejnych taktów (od 2 do 7) tworzy **cykl rozkazowy**, tj. przedział czasu, w którym jest wykonywany całkowicie jeden rozkaz (rys. 2.4).

Lista rozkazów

3.1. Wstępne wiadomości o rozkazach

Lista rozkazów mikroprocesora 6502 obejmuje wiele typów rozkazów: przesłań, różnych operacji na akumulatorze i innych rejestrach oraz rozkazów sterujących i innych. Ogółem jest 56 (66) rozkazów, a 151 (178) różnych kodów rozkazów; 13 (15) trybów adresowania rozkazów daje programiście dosyć duże możliwości.

W następnych punktach rozkazy są opisane według ich funkcji. Końcowe punkty omawiają tryby adresowania i zagadnienia dodatkowe. W Dodatkach D1 ÷ D3 podano zestawienia wszystkich rozkazów.

3.2. Rozkazy przesłań

Najczęściej wykonywanymi rozkazami są rozkazy przesyłania danych. Jeśli przez M oznaczymy adres pewnej komórki pamięci, to możemy symbolicznie zapisać operację przesłania danych z tej komórki do akumulatora: $M \rightarrow A$. Rozkaz wykonujący takie przesłanie ma skrót mnemoniczny LDA (ang. *Load Accumulator* – ładuj akumulator). Przesłanie w przeciwnym kierunku można oznaczyć: $A \rightarrow M$. Mnemonikiem rozkazu jest STA (ang. *Store Accumulator* – zapamiętaj akumulator).

Możliwe jest też (w mikroprocesorze 65C02) wyzerowanie komórki pamięci. Służy temu rozkaz STZ (ang. *Store Zero* – zapamiętaj zero).

Przesłania wewnątrz mikroprocesora mają inne mnemoniki: TAX (ang. *Transfer Accumulator to register X* – prześlij zawartość akumulatora do rejestru X), oznaczenie $A \rightarrow X$.

Przypominamy, że przesłanie danych z jednego miejsca do drugiego nie niszczy danych w ich źródle: po wykonaniu rozkazu STA zawartość akumulatora pozostaje bez zmian.

Tablica 3.1 podaje możliwe operacje przesłań oraz mnemoniki rozkazów. Zauważmy, że nie można bezpośrednio wykonać załadowania rejestru S (należy użyć dwóch rozkazów: LDX i TXS). Ponadto nie są możliwe bezpośrednie przesłania pomiędzy rejestrami X oraz Y.

Tablica 3.1 Rozkazy przesłań

Przesłanie	Mnemonic rozkazu
$M \rightarrow A$	LDA
$M \rightarrow X$	LDX
$M \rightarrow Y$	LDY
$A \rightarrow M$	STA
$X \rightarrow M$	STX
$Y \rightarrow M$	STY
$0 \rightarrow M$	STZ (65C02)
$A \rightarrow X$	TAX
$A \rightarrow Y$	TAY
$X \rightarrow A$	TXA
$Y \rightarrow A$	TYA
$X \rightarrow S$	TXS
$S \rightarrow X$	TSX

Rejestr P zdaje się być na razie w ogóle niedostępny, ale można zmodyfikować jego zawartość przez stos i bezpośrednio pewnymi rozkazami, co jest opisane w p. 3.8, oraz korzystać z jego zawartości rozkazami skoków warunkowych (p. 3.7).

Akumulator jest rejestrem szczególnie ważnym, gdyż na jego zawartości można wykonywać najwięcej operacji: dodawanie, odejmowanie, operacje logiczne oraz operacje przesuwania w prawo i w lewo (a także operacje zwiększania oraz zmniejszania zawartości o 1).

Aby wykonać dodawanie dwóch liczb, dostarcza się jedną z nich poprzez magistralę danych do lewego wejścia ALU, natomiast drugą — z akumulatora — do prawego wejścia ALU. Wynik z wyjścia ALU jest wpisywany do akumulatora.

Czytelnikowi nie obytemu z synchronicznymi układami cyfrowymi należy się tu wyjaśnienie, że jest możliwe równoczesne pobieranie danych z rejestru (tutaj A) i wpisywanie doń nowych. W układach synchronicznych (a tak jest właśnie wykonany akumulator) nie ma niebezpieczeństwa kolizji nowych danych ze starymi. Identycznie działają inne rejestry procesora.

Zauważmy w tym miejscu, że mikroprocesor 6502 (jak większość mikroprocesorów) jest maszyną jednoadresową, tj. przy wykonywaniu operacji dwuargumentowych podaje się adres tylko jednego argumentu: drugi znajduje się w umówionym miejscu (w akumulatorze), a wynik zostanie przesłany do umówionego miejsca (też do akumulatora).

Czytelnika chcącego przypomnieć sobie zasady arytmetyki odsyłamy do [7] oraz do Dodatku 4. Tutaj zauważmy tylko, że 6502 jest w stanie wykonywać: dodawanie i odejmowanie (z przeniesieniem lub pożyczką) w naturalnym kodzie dwójkowym (NB), w kodzie uzupełnień do dwóch (U2) oraz w kodzie BCD (ang. *binary coded decimal* – dziesiętny kodowany dwójkowo) oraz trzy operacje logiczne: sumę (ORA), iloczyn (AND) oraz sumę modulo 2 (EOR) na bitach (o jednakowych numerach) obu argumentów.

Dodawanie dwóch liczb 8-bitowych jest wykonywane tak, że mikroprocesor traktuje je równocześnie jako liczby w kodzie NB i U2. Jest to możliwe, bo wynik dodawania w obu kodach różni się tylko bitem przeniesienia (przepełnienia). Wyjaśnią to przykłady z rys. 3.1.

	Dodawanie:	jako NB	jako U2
a)	$\begin{array}{r} 01100000 \\ + 01000000 \\ \hline 10100000 \end{array}$	$\begin{array}{r} 96 \\ + 64 \\ \hline = 160 \end{array}$	$\begin{array}{r} (+96) \\ + (-64) \\ \hline = (+160) \\ = (-96) + (+256) \end{array}$
b)	$\begin{array}{r} 01100000 \\ + 11000000 \\ \hline 00100000 \end{array}$	$\begin{array}{r} 96 \\ + 192 \\ \hline = 288 \\ = 32 + 256 \end{array}$	$\begin{array}{r} (+96) \\ + (-64) \\ \hline = (+32) \end{array}$
c)	$\begin{array}{r} 10000000 \\ + 10000001 \\ \hline 00000001 \end{array}$	$\begin{array}{r} 128 \\ + 129 \\ \hline = 257 \\ = 1 + 256 \end{array}$	$\begin{array}{r} (-128) \\ + (-127) \\ \hline = (-255) \\ = (+1) + (-256) \end{array}$

Rys. 3.1. Dodawanie liczb w kodach NB i U2

Jak widać, te same słowa (reprezentujące inne liczby w kodach NB i U2) po dodaniu w sumatorze dają wyniki, które są bądź poprawne, bądź złe. W przykładzie z rys. 3.1a wynik %10100000 w kodzie NB oznacza liczbę 160 i jest poprawny, ale w kodzie U2 oznacza -96, a powinien wynosić +160. Dzieje się tak dlatego, że zakres 8-bitowych liczb w kodzie NB wynosi: 0...255, a w U2: (-128)...(+127). W przykładzie z rys. 3.1b wynik %00100000 oznacza liczbę +32 w obu kodach, ale w NB powinno być 288. Wreszcie w przykładzie z rys. 3.1c wynik jest %00000001, czyli +1 w obu kodach. Jest on w obu kodach niepoprawny: za duży lub za mały o 256 (nastąpiło przepełnienie). Mikroprocesor 6502 wykonuje dodawanie liczb dwójkowych tak, jak to opisano wyżej, sygnalizując przepełnienie dwoma znacznikami: *C* (ang. *Carry* – przeniesienie) dla kodu NB oraz *V* (ang. *overflow* – przepełnienie) dla kodu U2. Jeśli programista dodaje liczby w kodzie NB i obawia się złego wyniku, to powinien po dodawaniu sprawdzić znacznik *C*, a znacznik *V* zignorować. Jeśli pracuje w U2, to na odwrót: trzeba sprawdzić *V*, a zignorować *C*.

Jak to zrobić, jest opisane dalej. Tymczasem zauważmy, że podobnie jest przy odejmowaniu (Czytelnikowi proponujemy sprawdzić na podobnych przykładach), tyle że znacznik *C* odgrywa rolę negacji pożyczki (tzn. jeśli po odejmowaniu w NB będzie $C = 1$, to wszystko w porządku, a jeśli $C = 0$, to nastąpiła pożyczka – czyli przepełnienie – i wynik w akumulatorze jest o 256 za duży). Znacznik *V* zachowuje się tak, jak przy dodawaniu.

Ponadto trzeba pamiętać, że przed dodawaniem należy wyzerować znacznik *C* (lub być pewnym, że jest wyzerowany), a przed odejmowaniem – ustawić go na 1 (żeby nie było przypadkowego przeniesienia ani pożyczki na początku). Służą do tego rozkazy CLC i SEC. Może się to wydawać dziwne, ale jest korzystne przy tzw. arytmetyce wielokrotnej precyzji (Dodatek 4).

Po każdej operacji arytmetycznej (i po wielu innych) sumator zmienia jeszcze dwa znaczniki, *Z* i *N*, w następujący sposób: znacznik *Z* zostaje ustawiony na 1, jeśli wynik działania jest równy zero (chodzi o 8 bitów wyniku, bez bitu nadmiaru), a wyzerowany w przeciwnym przypadku. Znacznik *N* (ang. *Negative* – ujemny) staje się równy bitowi 7 wyniku, tj. równy 1, gdy wynik w kodzie U2 jest ujemny bądź gdy w kodzie NB przekracza 127, a jest zerowany w przeciwnych przypadkach.

Sumator mikroprocesora 6502 może też pracować w arytmetyce dziesiętnej. Przed przystąpieniem do takiej pracy należy rozkazem SED (ang. *SEt Decimal* – ustaw tryb dziesiętny) przełączyć znacznik *D* na 1, potem można wrócić do pracy dwójkowej rozkazem CLD (ang. *CLear Decimal mode* – wyzeruj tryb dziesiętny). Sumator pracuje wtedy w kodzie BCD (dwie cyfry dziesiętne kodowane dwójkowo), a wynik nie wymaga żadnej korekcji. Ilustrują to przykłady (rys. 3.2). Przykład b) pokazuje, że w przypadku nadmiaru (a teraz zakresem liczb jest 0...99) otrzymamy końcówkę wyniku (tzn. 34), a znacznik $C = 1$ poinformuje o przepełnieniu. W przykładzie c) na skutek odejmowania większej liczby od mniejszej występuje pożyczka (tzn. $C = 0$), a wynik pojawia się w dziwnej postaci tzw. uzupełnienia dziesiątkowego. Zalety tego rozwiązania są opisane w Dodatku 4.

a)	59	0101.1001
	+ 35	+0011.0101
	= 94	1001.0100
b)	59	0101.1001
	+ 75	+0111.0101
	= 134	0011.0100
c)	59	0101.1001
	- 75	-0111.0101
	= (-16)	1000.0100
	= (+84 - 100)	

Rys. 3.2. Dodawanie i odejmowanie liczb dziesiętnych

Przy pracy dziesiętnej sumatora należy ignorować znacznik *V*. Liczby ujemne nie dają się kodować bezpośrednio (tylko pośrednio przez kod

Jak się później przekonamy, rozkaz ten jest szczególnie wygodny do szybkiego sprawdzania stanu wyłączników, klawiszy i innych sygnałów dochodzących do mikrokomputera z zewnątrz (także do szybkiego określania urządzenia, które zgłasza przerwanie).

Procesor 65C02 uzupełniono o dwa podobne rozkazy: TSB (ang. Test and Set Bits — sprawdź i ustaw bity) i TRB (ang. Test and Reset Bits — sprawdź i wyzeruj bity)¹.

Pierwszy z nich ustawia znaczniki tak, jak BIT: $M7 \rightarrow N$ i $M6 \rightarrow V$, a potem oblicza sumę logiczną akumulatora i komórki pamięci. Wynik przesyła do pamięci ($A \vee M \rightarrow M$), a znacznik Z ustawia na 1, jeśli ten wynik jest zerowy.

Rozkaz TRB ustawia N i V tak samo. Potem oblicza iloczyn logiczny negacji zawartości akumulatora z komórką pamięci, a wynik przesyła do pamięci ($\bar{A} \wedge M \rightarrow M$). Znacznik Z zachowuje się tak samo.

UWAGA: w obu rozkazach znaczniki N i V zależą od starej zawartości komórki pamięci, a Z — od nowej.

3.5. Rozkazy działań logicznych

Mikroprocesor 6502 może wykonywać trzy działania logiczne: iloczyn, sumę i różnicę symetryczną (sumę modulo 2).

Rysunek 3.4a pokazuje wynik działania rozkazu AND (iloczyn logiczny): w wyniku są jedynki tylko na tych pozycjach (tutaj bity 2 i 3), na których były jedynki w obu argumentach. To działanie może służyć do „wycinania” (lub „maskowania”) jedynek. W tym przykładzie pierwszy argument „wyciął” z drugiego dwie jedynki na pozycjach 4 i 5.

	76543210
a)	00001111
	00111100

	00001100
b)	00001111
	00111100

	00111111
c)	00001111
	00111100

	00110011

Rys. 3.4. Przykłady wykonania rozkazów operacji logicznych:
a) AND; b) OR; c) EOR

¹ Te rozkazy wspomagają pracę w systemie wieloprocesorowym dzięki temu, że umożliwiają jednoczesne testowanie i ustawianie semaforów programowych. Umożliwia to budowanie systemów wieloprocesorowych z tym procesorem, aczkolwiek rzadko się go do tych celów wykorzystuje.

Rysunek 3.4b ilustruje działanie rozkazu ORA (suma logiczna) dla tych samych danych: w wyniku są jedynki na tych pozycjach, na których były jedynki choć w jednym argumencie. To działanie w tym przykładzie dodało jedynki do drugiego słowa na pozycjach 0 i 1.

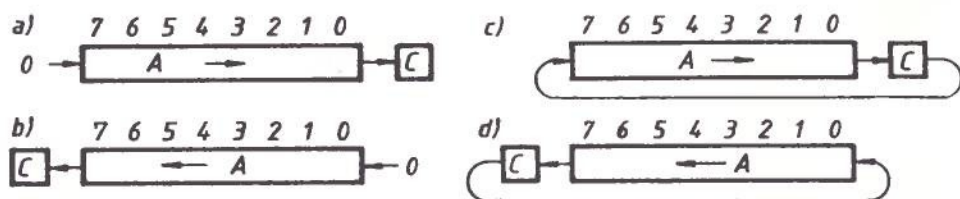
Rysunek 3.4c objaśnia trzeci rozkaz: EOR (ang. *Exclusive OR* – suma wyłączająca). Wynik różni się od jednego z argumentów na tych pozycjach, na których drugi argument ma jedynkę. Jeśli jeden argument zawiera same jedynki, to drugi zmienia się na swoją negację.

Oczywiście nie musimy dodawać, że jeden z argumentów tych rozkazów musi znajdować się w akumulatorze, drugi jest pobierany z komórki pamięci, a wynik zostanie zapisany do akumulatora (niszcząc starą wartość pierwszego argumentu).

Po wykonaniu rozkazów AND, ORA, EOR zmieniają się tylko znaczniki Z i N. Znacznik Z staje się 1, gdy wynik jest zerem, a staje się 0 w przeciwnym przypadku; N powtarza bit 7 wyniku. Inne znaczniki pozostają nie zmienione.

3.6. Rozkazy przesunięć

Zawartość akumulatora może być przesuwana (jak na rys. 3.5). Rozkaz LSR (ang. *Logical Shift Right* – logiczne przesunięcie w prawo) powoduje wpisanie bitu z pozycji 0 do znacznika C, bitu z pozycji 1 do pozycji 0 itd., a bit Z jest zerowany (rys. 3.5a). Rozkaz ASL (ang. *Arithmetic Shift Left* – arytmetyczne

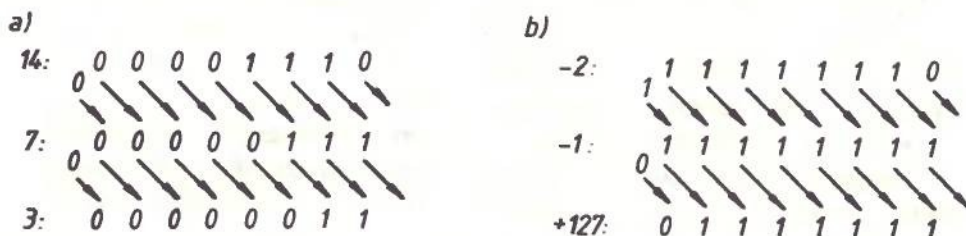


Rys. 3.5. Przykłady wykonania rozkazów przesunięć: a) LSR; b) ASL; c) ROR; d) ROL

przesunięcie w lewo) działa symetrycznie (rys. 3.5b). Dwa następne rozkazy to ROR (ang. *ROtate Right* – przesunij cyklicznie w prawo) i ROL (ang. *ROtate Left* – przesunij cyklicznie w lewo). Działają one podobnie jak opisane wyżej, ale zamiast zera wpisuje się stara zawartość znacznika C (rys. 3.5c i d). Przed użyciem tych rozkazów należy mieć pewność, że znacznik C ma taką wartość, jaką chcemy (można ją zmienić rozkazami CLC i SEC); w przeciwnym przypadku przesunięta zawartość akumulatora będzie przypadkowa na bicie 7 lub 0.

Każdy z tych rozkazów zmienia jeszcze stan znaczników Z i N w znany sposób: Z informuje o zerowym wyniku (jak zwykle chodzi o 8 bitów akumulatora, bez bitu C), a N mówi o tym, że bit 7 wyniku stał się równy 1.

Operacje przesunięcia mogą mieć znaczenie arytmetyczne albo logiczne. Na rys. 3.6a pokazano liczbę 14 (w kodzie NB) przesuniętą w prawo (otrzymaliśmy jej połowę, tj. 7) i jeszcze raz przesuniętą w prawo. Tym razem wynikiem jest 3, czyli tzw. „mniejsza połowa”, tj. wynik dzielenia zaokrąglony w dół.



Rys. 3.6. Arytmetyczne znaczenie rozkazu LSR: a) dla kodu NB; b) dla kodu U2

Dla kodu U2 ten sam przykład jest dobry, ale tylko dla liczb nieujemnych. Rysunek 3.6b pokazuje, że w czasie przesunięcia w prawo liczby ujemnej należy spowodować ustawienie bitu 7 na jeden. (Zaniechanie tego da błędny wynik, tu: 127). Trzeba użyć dwóch rozkazów: SEC i ROR.

Przesuwanie może też mieć znaczenie czysto logiczne. Jeśli pewna komórka pamięci przechowuje dane, które mają obrazować słupkę diod świecących (np. jako zobrazowanie pomiaru temperatury) i założymy, że 1 oznacza świecenie diody, a 0 — zgaszenie, to rys. 3.7 pokazuje przykład wzrostu temperatury

00000000
10000000
11000000
11100000

11111110
11111111
11111110

11000000
10000000
00000000

Rys. 3.7. Przykłady wykorzystania rozkazów przesunięć do zobrazowania wzrostu i spadku temperatury

(świecenie diod rozszerza się w prawo) i jej spadku (świecący rząd kurczy się w lewo). Czytelnikowi pozostawiamy jako ćwiczenie napisanie, jakich rozkazów należałoby użyć do zobrazowania wzrostu temperatury i jej spadku. Zanim jednak to zrobisz, Czytelniku, zwróć uwagę na to, że wyżej omówione rozkazy mogą dotyczyć także dowolnej komórki pamięci. Wykonując taki rozkaz, mikroprocesor dokonuje samoczynnie następujących czynności:

- przesłanie danych z komórki pamięci do pewnego rejestru (nazwijmy go

- przesunięcie $A1$ według treści rozkazu i ustawienie znaczników N , Z , C ,
- przesłanie zawartości $A1$ z powrotem do tej samej komórki pamięci.

Dotychczas założyliśmy milcząco (a właściwie liczyliśmy, że się tego, Czytelniku, domyślisz), że rozkazy umieszczone kolejno w pamięci mikrokomputera są przez mikroprocesor wykonywane po kolei. Tak jest rzeczywiście, ale istnieją rozkazy, które mogą zmienić tę kolejność.

PC	Instruction
0501	INX
0502	STX
0503	03
0504	07
0505	LDX
0506	00
0507	10
0508	JMP
0509	00
050A	06
050B	
...	
0600	
0601	

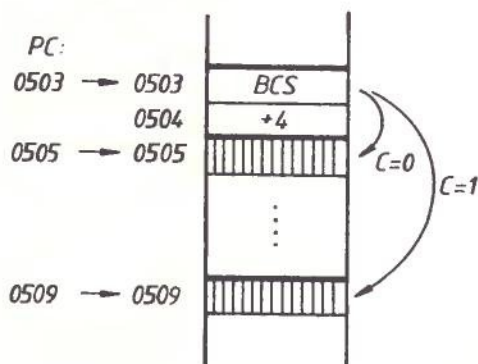
Rys. 3.8. Przykład rozmieszczenia w pamięci kilku kolejnych rozkazów i wykonania rozkazu skoku (JMP)

Rozkaz INX (zwiększa zawartość rejestru X o 1) zajmuje zawsze jedną komórkę (tutaj komórkę o numerze \$0501). Przed jego wykonaniem zawartość PC jest równa \$0501 i wskazuje na tę komórkę. W czasie wykonywania tego rozkazu mikroprocesor zwiększa PC o 1 po to, aby następny rozkaz pobrać z komórki \$0502. Ścisłej: pobrać pierwszy bajt tego rozkazu, bo tym razem jest to rozkaz STX (zapamiętaj rejestr X) i zajmuje 3 bajty. Poleca on mikroprocesorowi zapamiętać zawartość rejestru X w komórce o adresie \$0703 (dlaczego na rys. 3.8 liczby \$07 i \$03 są przestawione — wyjaśnimy wkrótce). W czasie wykonywania rozkazu STX liczba pamiętana w liczniku rozkazów zwiększa się o 3 (czyli wynosi \$0505) i wskazuje na pierwszy bajt następnego rozkazu. I w tym przypadku rozkaz LDX (ładuj rejestr X) jest trzybajtowy (poleca on przesłać nowe dane do rejestru X z komórki pamięci o numerze \$1000), a więc w czasie jego wykonania PC zwiększa się o 3.

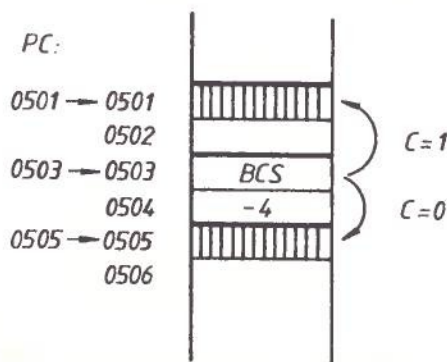
W komórce \$0508 znajduje się kod rozkazu JMP (ang. *Jump* — skocz), a w dwóch następnych — adres skoku, tutaj liczba \$0600. Zamiast więc zwiększania zawartości PC mikroprocesor wpisuje ten adres do PC . To oznacza, że pierwszego bajtu następnego rozkazu mikroprocesor będzie szukał pod adresem \$0600.

Tu jeszcze uwaga dotycząca numeracji komórek: po liczbie \$0509 nie następuje \$0510, lecz \$050A, bo używamy tu kodu szesnastkowego, a nie dziesiętnego. Można stosować dowolną konwencję, ale wszyscy przyzwyczaili się do szesnastkowej, więc i Ty, Czytelniku, też musisz. Zresztą cóż by był za pożytek z pisania numeru 4096 (dziesiętnie), skoro szesnastkowo wynosi to \$1000, a po rozłożeniu na dwie części: bardziej znaczącą \$10 i mniej znaczącą \$00, musi być w tej właśnie postaci umieszczony w pamięci mikroprocesora (jak adres w rozkazie LDX w komórkach \$0507 i \$0506)?

Istnieją jeszcze rozkazy skoków warunkowych: BCC (ang. *Branch on Carry Clear* — skocz, jeśli C wyzerowane), BCS (ang. *Branch on Carry Set* — skocz, jeśli C ustawione), BEQ (ang. *Branch on Equal* — skocz, jeśli równe; czyli gdy $Z = 0$), BNE (ang. *Branch on Not Equal* — skocz, jeśli nierówne, czyli gdy



3.9. Przykład wykonania rozkazu skoku warunkowego BCS w przód



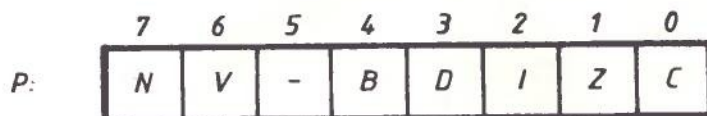
Rys. 3.10. Przykład wykonania rozkazu skoku warunkowego BCS w tył

$Z = 1$), BPL (ang. *Branch on Plus* – skocz, jeśli dodatnie, czyli gdy $N = 0$), BMI (ang. *Branch on Minus* – skocz, jeśli ujemne, czyli gdy $N = 1$), BVC (ang. *Branch on V Clear* – skocz, jeśli V wyzerowane), BVS (ang. *Branch on V Set* – skocz, jeśli V ustawione).

Każdy z tych rozkazów poleca zbadać stan odpowiedniego znacznika (C , Z , N albo V odpowiednio dla każdej pary rozkazów) i wykonanie skoku do nowego adresu, o ile warunek, wynikający z treści rozkazu, jest spełniony. W przeciwnym przypadku nie dzieje się nic poza przejściem do następnego rozkazu. Przykład z rys. 3.9 wyjaśnia to dla rozkazu BCS zapisanego w komórkach \$0503 i \$0504: jeśli przed jego wykonaniem było $C = 0$, to warunek nie jest spełniony i PC zwiększy się o 2 wskazując na następny rozkaz (w komórce \$0505). Jeśli jednak było $C = 1$, to PC zwiększy się o 2 i jeszcze o 4: nowy rozkaz będzie pobrany z komórki \$0509. *Procesor 65C02 ma jeszcze rozkaz BRA (ang. BRanch Always – skacz zawsze), który wykonuje się zawsze – bez sprawdzania warunków.*

Zauważmy, że rozkazy skoków warunkowych nie podają pełnego adresu skoku, lecz jedynie poprawkę względem adresu następnego rozkazu. Poprawka (czyli tzw. **adres względny**, o czym jeszcze będzie mowa) jest w kodzie U2 (czyli z zakresu $-128 \dots +127$), a zatem może dotyczyć także komórek wcześniejszych. Rysunek 3.10 pokazuje przykład skoku do tyłu.

Do rozkazów sterujących należą też rozkazy zmieniające stan znaczników: CLC (ang. *CLear Carry* – wyzeruj C), SEC (ang. *SEt Carry* – ustaw C), CLD (ang. *CLear D* – wyzeruj D), SED (ang. *SEt D* – ustaw D), CLV (ang. *CLear V* – wyzeruj V), CLI (ang. *CLear I* – wyzeruj I), SEI (ang. *SEt I* – ustaw I).



Rys. 3.11. Rejestr znaczników

Rysunek 3.11 pokazuje cały rejestr znaczników. Przypomnijmy krótko ich role: znacznik N pokazuje, czy ostatnio wykonana operacja lub przesłanie dały w wyniku liczbę ujemną (tj. powtarza bit 7 wyniku). Znacznik C sygnalizuje nadmiar (przepełnienie) dla liczb w kodzie NB, a V – przepełnienie dla liczb traktowanych w kodzie U2. Znacznik Z informuje, że wynik operacji jest zerem. Znacznik D sprawia, że sumator pracuje dziesiętnie, a nie dwójkowo.

Rozkazy SEI oraz CLI sterują znacznikiem I (ang. *Interrupt disable* – blokowanie przerwań IRQ) i zostaną omówione w p. 3.9 i 4.4 (podobnie znacznik B – *Break*).

Brak rozkazu ustawiania znacznika V jest spowodowany tym, że w programie istnieje potrzeba tylko sprawdzenia jego stanu i zerowania go. Pierwotnie przewidywano możliwość współpracy mikroprocesora 6502 z zewnętrznym koprocesorem arytmetycznym, który sygnalizowałby swoją gotowość ustawiając znacznik V sygnałem przez nóżkę 38 (ang. *set overflow*).

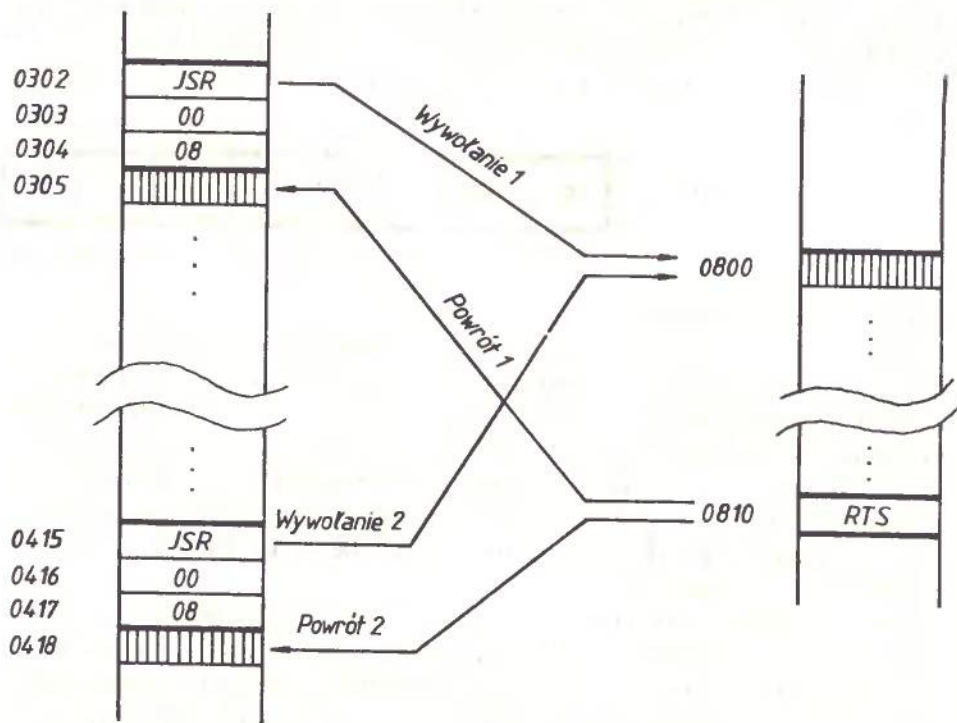
Na razie taki koprocesor nie jest produkowany. Bit 5 w rejestrze znaczników nie jest wykorzystany: jego wartość jest przypadkowa. W procesorze 65C02 jest równa 1.

W tej grupie rozkazów można też umieścić NOP (ang. *No OPeration*). Nie wykonuje on nic, stosuje się go zaś bądź do rezerwacji miejsca w programie (przewidzianego na przyszłe zmiany), bądź do zastąpienia zbędnych już fragmentów programu.

W przypadku 65C02 rozkazami NOP są wszystkie nie wykorzystane kombinacje bitów (szczegóły w Dodatku 5).

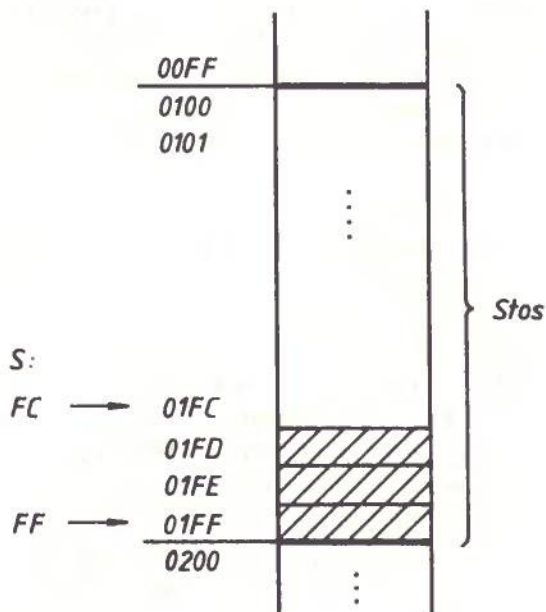
3.8. Rozkazy związane z podprogramami

Jeśli w programie jest fragment, którego trzeba użyć kilka razy, to wygodnie jest nie pisać go kilka razy, lecz tylko raz, a główny program zorganizować tak, żeby mógł korzystać z tego fragmentu (zwanego **podprogramem**) kilka razy. Istotne jest to, żeby po wykorzystaniu podprogramu wrócić za każdym razem do rozkazu następnego po tym, który podprogram wywołał. Widać stąd, że rozkazy skoku JMP są tu nieprzydatne. Mikroprocesor 6502 dysponuje za to dwoma innymi: JSR (ang. *Jump to SubRoutine* – skocz do podprogramu) oraz RTS (ang. *ReTurn from Subroutine* – powrót z podprogramu).



Rys. 3.12. Wywołanie podprogramu i powrót do programu głównego

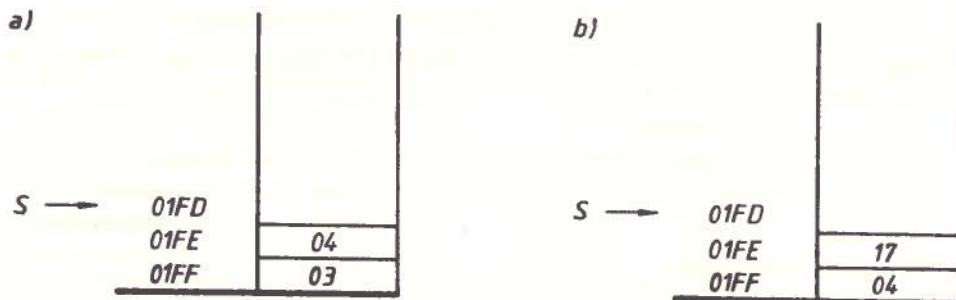
Rysunek 3.12 pokazuje fragment programu z podprogramem, który zaczyna się w komórce \$0800, a kończy w \$0810. Program główny wywołuje go po raz pierwszy rozkazem JSR z komórek \$0302...\$0304, a po raz drugi rozkazem JSR z komórek \$0415...\$0417. Podprogram po wykonaniu swoich obliczeń wykonuje rozkaz RTS, który w pierwszym przypadku realizuje powrót do komórki \$0305, a w drugim — do \$0418. Jest to możliwe dzięki tzw. **stosowi**. Stos (ang. *stack*) jest to pewien obszar pamięci zapisywalnej, zorganizowany tak, jak stos kartek na biurku. Widzimy zawsze treść kartki leżącej na wierzchu; możemy dołożyć nową kartkę na wierzch; możemy też zdjąć kartkę z wierzchu i odczytać treść leżącą niżej. W mikroprocesorze 6502 stosem jest obszar pamięci o adresach \$0100...\$01FF (rys. 3.13). Dnem stosu jest komórka \$01FF (odpowiednik kartki leżącej bezpośrednio na biurku).



Rys. 3.13. Przykład pracy stosu

Aktualnym wierzchem stosu jest komórka o adresie o 1 większym od zawartości rejestru wskaźnika stosu *S* (ang. *stack pointer*). Na rys. 3.13 *S* zawiera liczbę \$FC, co oznacza, że pierwszą wolną komórką stosu jest \$01FC, a trzy następne, tj. \$01FD, \$01FE i \$01FF, zawierają jakieś istotne informacje. Jeśli wskaźnik stosu *S* zawiera liczbę \$FF, oznacza to, że stos jest pusty. Do wskaźnika stosu *S* można zresztą zapisać na początku dowolną liczbę (z zakresu \$00 ÷ FF) i tam znajdzie się wtedy dno stosu.

Użycie stosu do wywoływania podprogramów ilustruje rys. 3.14. Przy wywołaniu podprogramu mikroprocesor umieszcza na stosie adres powrotu zmniejszony o 1 (tj. w omawianym przykładzie \$0304), czyli adres następnego rozkazu po JSR, potem zmniejsza zawartość rejestru *S* o 2, a do licznika rozkazów wpisuje adres początku podprogramu (\$0800). To oznacza, że następnym rozkazem będzie rozkaz rozpoczynający się od komórki \$0800. W



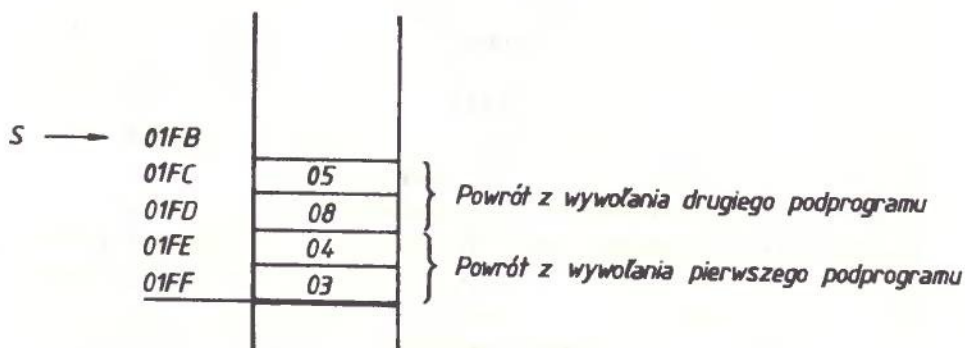
Rys. 3.14. Przykład zawartości stosu po wywołaniu podprogramu według rys. 3.12: a) za pierwszym razem; b) za drugim razem

tym momencie stan stosu jest pokazany na rys. 3.14. Gdy wykonanie podprogramu dojdzie do końca, to rozkaz RTS spowoduje przesłanie adresu ze stosu do licznika rozkazów i zwiększenie zawartości S o 2. Zmiana zawartości PC oznacza skok, a więc nastąpi powrót do komórki \$0305 (bo w czasie wykonywania ostatniego cyklu rozkazu RTS zawartość PC jest zwiększana o 1).

Po drugim wywołaniu podprogramu stos zapełni się jak na rys. 3.14b. Powrót nastąpi więc do komórki \$0418 (tj. następnego rozkazu po rozkazie JSR).

Gdybyśmy chcieli w podprogramie wywołać jakiś inny podprogram (np. w komórce \$0803 umieścili rozkaz JSR z odpowiednim adresem), to oczywiście można to zrobić. Jeśli rozkaz JSR zajmuje komórki \$0803, \$0804 i \$0805, to stan stosu po tym drugim wywołaniu pokazuje rys. 3.15. Rejestr S wskazuje na pierwszą pustą komórkę (tj. \$01FB), a pod nią, w komórkach \$01FC i D znajduje się ślad powrotu z drugiego podprogramu (tj. do komórki \$0806), a jeszcze głębiej – w komórkach \$01FE i F – ślad powrotu z pierwszego podprogramu do programu głównego (tj. do komórki \$0305).

Z informacji, że pamięć stosu ma 256 komórek, wynika, że można aż 128 razy zagłębiać się w wywoływanie podprogramów. Większość programistów używa kilku, wyjątkowo kilkunastu, zagłębień.

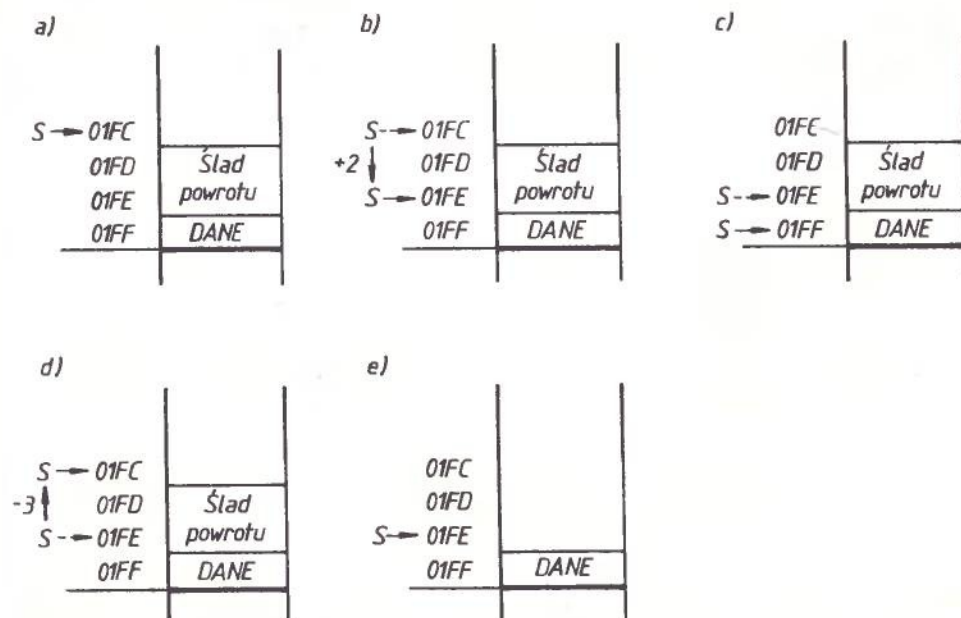


Rys. 3.15. Przykład stanu stosu po podwójnym zagłębieniu się w podprogramy

Rejestr *S* jest licznikiem modulo 256, a zatem tak naprawdę stos jest strukturą cykliczną i nigdy nie wychodzi poza stronę pierwszą pamięci. Ma to tę zaletę, że przy uruchamianiu programów nie ma możliwości zniszczenia programu głównego przez omyłkowe zagłębianie się wywołań podprogramów.

Wadą jest niemożność bezpośredniej realizacji rekursji (bo wtedy 256 komórek już nie wystarcza), ale też 6502 nie był pierwotnie projektowany do takich zastosowań.

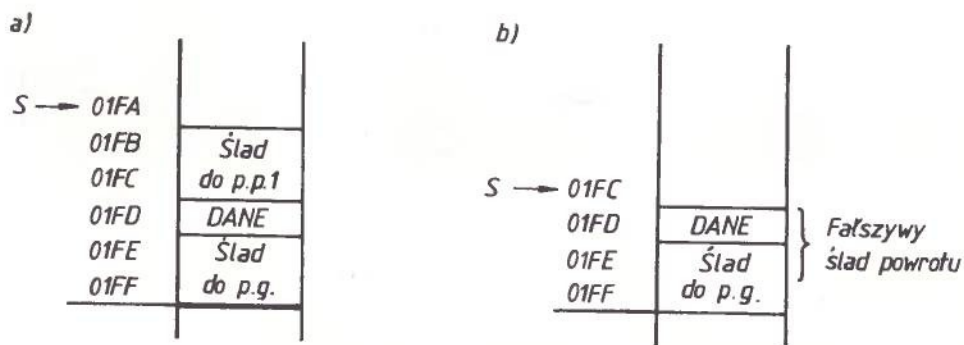
To jeszcze nie wszystkie możliwości współpracy ze stosem. Istnieją jeszcze cztery rozkazy: *PHA* (ang. *Push Accumulator to stack* – umieść akumulator na stosie), *PHP* (ang. *Push Processor status to stack* – umieść rejestr *P* na stosie), *PLA* (ang. *PuLL Accumulator from stack* – odczytaj akumulator ze stosu), *PLP* (ang. *PuLL Processor status from stack* – odczytaj rejestr *P* ze stosu). Dwa pierwsze powodują zapamiętanie zawartości akumulatora (bądź rejestru *P*) na stosie (i zmniejszenie zawartości *S* o 1). Dwa ostatnie wykonują operacje przeciwne.



Rys. 3.16. Przykład pracy przy przekazywaniu danych do podprogramu przez stos: a) po wykonaniu skoku do podprogramu; b) przygotowanie do odczytu danych; c) po odczycie danych; d) odtworzenie zawartości rejestru *S* – przygotowanie do powrotu; e) po wykonaniu powrotu

Te rozkazy mogą służyć do przekazywania danych do podprogramów. Jeśli program główny ma przekazać do podprogramu jeden bajt, to można to zorganizować następująco: przed wykonaniem skoku do podprogramu należy zapamiętać na stosie dane rozkazem *PHA*. Po wykonaniu rozkazu *JSR* stan stosu przedstawia rys. 3.16a. W podprogramie, aby odczytać komórkę *DANE*, należy ominąć w jakiś sposób dwie komórki zawierające ślad powrotu do

programu głównego. Należy zwiększyć zawartość rejestru *S* o dwa (jak na rys. 3.16b). Wskazuje on wprowadzić na komórkę \$01FE, a nie na \$01FF, w której są interesujące nas *DANE*, ale o to właśnie chodzi, gdyż użyty teraz rozkaz *PLA* samoczynnie powiększa zawartość rejestru *S* o 1 przed przesłaniem zawartości stosu do akumulatora. Po wykonaniu tego rozkazu stos wygląda jak na rys. 3.16c. Zauważmy przy tym, że po raz pierwszy rysujemy tutaj zawartość stosu ponad miejscem wskazywanym przez *S*. Powinno się tak robić zawsze wtedy, gdy mamy zamiar powrócić jeszcze do jego zawartości (tak, jakbyśmy kartki ze stosu na biurku odkładali na bok zamiast wyrzucać je do kosza). Porównując rys. c) i a) można dojść do wniosku, że po tych wszystkich operacjach trzeba przywrócić jego stan według rys. d) po to, aby rozkaz *RTS* (który zakończy podprogram) mógł spowodować powrót do właściwego miejsca w programie głównym. Pozostawiamy Czytelnikowi do sprawdzenia, że ciąg rozkazów: *TSX*, *DEX*, *DEX*, *TXS* wykonuje operację według rys. d). Lepiej jednak nie zmieniać zawartości rejestru *S*, a dostęp do komórki *DANE* uzyskać ciągiem rozkazów: *TSX*, *DEX*, *DEX*, *DEX*, *LDA* \$1000, *X*. Po wykonaniu powrotu *RTS* sytuacja będzie jak na rys. 3.16e (tu już zapominamy o śladzie powrotu, bo do niego nie wrócimy).



Rys. 3.17. Przykład niepoprawnego użycia stosu: a) stan po dwukrotnym zagłębieniu się z przekazaniem danych; b) stan po pierwszym powrocie

Wydawałoby się, że wszystko jest w porządku, ale program główny powinien jeszcze zwiększyć zawartość *S* o 1 aby „unicestwić” *DANE* (np. rozkazem *PLA*). Jest to konieczne nie tylko dla uzyskania elegancji, ale także dla uniknięcia pomyłki jak na rys. 3.17a. Przedstawiono tu stan stosu po wywołaniu podprogramu *PP1* przez program główny, oraz po przekazaniu danych i wywołaniu podprogramu drugiego stopnia *PP2*. Po powrocie do *PP1* (ale bez „skasowania” komórki *DANE*) sytuacja wygląda jak na rys. 3.17b. Użycie w tym momencie rozkazu *RTS* (z intencją powrotu do programu głównego) da oślakane skutki, bo mikroprocesor weźmie ślad powrotu z komórek \$01FD i \$01FE, zamiast z \$01FE i \$01FF.

65C02 ma jeszcze cztery podobne rozkazy: *PHX* (ang. *PusH X on stack* – zapisz *X* na stos), *PHY* (ang. *PusH Y on stack* – zapisz *Y* na stos), *PLX* (ang.

PuLL X from stack — odczytaj *X* ze stosu), *PLY*(ang. *PuLL Y from stack* — odczytaj *Y* ze stosu). Umożliwiają one szybkie zapamiętanie stanu tych rejestrów na stosie — bez pośrednictwa akumulatora.

3.9. Obsługa przerw

Każdy nowoczesny komputer (a także mikroprocesor) musi mieć tzw. **system przerw**. Można go porównać do dzwonka u drzwi wejściowych mieszkania, który umożliwia oderwanie gospodarza od jego aktualnego zajęcia i obsłużenie przybysza, np. listonosza. Gdyby dzwonka nie było, to właściciel takiego mieszkania musiałby stać cały dzień w drzwiach mieszkania (pod groźbą utraty np. emerytury).

Podobnie działa mikroprocesor, jeśli ma obsługiwać kilka zadań (np. dwa), przy czym jedno z nich pojawia się sporadycznie, ale za to wymaga szybkiej obsługi.

6502 ma nóżkę nr 4 nazwaną $\overline{IRQ}$ (ang. *Interrupt ReQuest* — żądanie przerwania), która wykrywa stan zera logicznego (szczegóły w rozdz. 4). Gdy się ten stan pojawi (tak jak naciśnięcie guzika dzwonka), mikroprocesor wykonuje następujący ciąg działań:

- kończy wykonanie bieżącego rozkazu (tak, jakby żadnego zgłoszenia na razie nie było),
 - przesyła na stos adres powrotu do programu (tj. adres początku następnego rozkazu),
 - przesyła na stos zawartość rejestru *P*,
- (powyższe operacje oczywiście powodują zmniejszenie zawartości *S* o 3),
- do *PC* przesyła tzw. **wektor przerwania** *IRQ*, tj. zawartość komórki \$FFFE (mniej znaczący bajt) i \$FFFF (bardziej znaczący),
 - ustawia bit *I* w rejestrze stanu *P* na 1,
 - przechodzi do wykonywania następnego rozkazu.

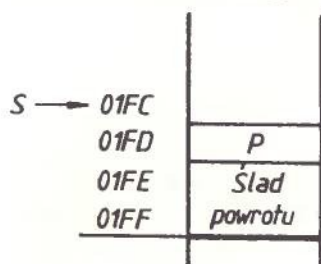
To, co się dzieje dalej, zależy już od programisty. Powinien on napisać odpowiedni program, który zaczynałby się w tej komórce, której adres podano w komórkach \$FFFE i \$FFFF, bo to będzie właśnie pierwszy rozkaz programu obsługi przerwania (*PC* zawiera właśnie ten adres). Program powinien obsłużyć przerwanie, tzn. wykonać wszystkie czynności (np. obliczenia, wysłanie danych, odczyt danych), których żąda ten sygnał przerwania.

Mikroprocesor ustawia bit *I* w swoim rejestrze stanu po to, aby sygnał przerwania (który jakiś czas trwa) nie przerywał programu obsługującego to przerwanie, ale aby procesor go wykonał do końca.

Jest jeszcze jedna możliwość wykorzystania znacznika *I*: rozkazem *SEI* (ang. *SEt I* — ustaw *I*) można go ustawić w dowolnej chwili i wtedy przerwania nie są przyjmowane (właściciel mieszkania gaszący płonąca firankę też nie reaguje na dzwonek u drzwi!).

Rozkazem *CLI* (ang. *CLear I* — wyzeruj *I*) można wyzerować ten znacznik,

zezwalając znowu na przyjęcie przerwania. Przyjęto tu umowę, że urządzenie zgłaszające wysyła sygnał przerwania *IRQ* tak długo, aż zostanie obsłużone: do tej chwili sygnał *IRQ* nie znika.



Rys. 3.18. Stan stosu po przyjęciu przerwania

Po zakończeniu programu obsługi należy wrócić do przerwanego programu głównego. Służy temu rozkaz *RTI* (ang. *ReTurn from Interrupt* – powrót z przerwania). Z rys. 3.18, który pokazuje stan stosu po przyjęciu przerwania, wynika, co ten rozkaz wykonuje:

- przesyła starą zawartość rejestru *P* ze stosu do tego rejestru (tutaj zwykle następuje wyzerowanie bitu *I* oraz przywrócenie wartości innych znaczników na takie, jakie były przed przyjęciem przerwania),
- przesyła ślad powrotu ze stosu do licznika rozkazów *PC*,
- próbuje przejść do wykonania następnego rozkazu.

(Po tych czynnościach zawartość rejestru *S* zwiększa się o 3).

Jak widać, rozkaz *RTI* jest bardzo podobny do *RTS* (rozszerzony o manipulację z rejestrem *P*).

Komentarza wymaga jeszcze sformułowanie „próbuję przejść do wykonania następnego rozkazu”. Po wykonaniu *RTS* w liczniku rozkazów znajduje się już adres tego rozkazu, który jako pierwszy nie został wykonany, bo zostało przyjęte przerwanie. Cóż więc może przeszkodzić mikroprocesorowi w jego wykonaniu? Otóż – następne przerwanie! Jeśli się ono zgłosi przed wykonaniem rozkazu powrotu *RTI*, a w czasie wykonywania tego rozkazu zostanie przywrócona stara wartość *I* równa 0, to mikroprocesor zamiast dokończenia programu głównego wykona jeszcze raz opisaną wyżej procedurę przyjęcia przerwania i zacznie wykonywać od nowa program obsługi przerwania.

Czy to ma sens? Tak. Czasem mogą się zdarzyć zgłoszenia tego samego typu bardzo blisko siebie. Częściej jednak bywa tak, że do tej samej nóżki *IRQ* procesora jest dołączonych kilka urządzeń, z których np. dwa zażądają obsługi w krótkim odstępie czasu. W tym przypadku to drugie musi trochę poczekać na obsługę. (Jeszcze jedna ważna rzecz: program obsługi musi w tym przypadku zbadać, które urządzenie – lub które urządzenia – zgłaszają się i tylko te obsługiwać!).

W mikroprocesorze istnieje jeszcze nóżka nr 6, nazwana *NMI* (ang. *Non-Maskable Interrupt* – przerwanie niemaskowalne). Sygnał dochodzący tutaj traktowany jest podobnie jak *IRQ*, ale:

- jeśli zgłoszą się oba, to mikroprocesor obsługuje *NMI* (a *IRQ* czeka),
- na *NMI* nie działa maskującą ani wskaźnik *I*, ani żaden inny,
- wejście *NMI* jest wrażliwe nie na poziom sygnału (równy 0), ale na opadające zbocze (skok sygnału z 1 na 0 powoduje zgłoszenie przerwania); trwające dalej 0 nie będzie rozpoznane po raz drugi,
- adres początku programu obsługi (tzw. wektor *NMI*) należy umieścić w komórkach \$FFFA i \$FFFB.

Do kompletu rozkazów dotyczących przerwania należy jeszcze *BRK* (ang. *force BRaK* – wymuś przerwanie). Działa on tak, jak rozkaz *NOP* (nic), w czasie którego zgłosiło się przerwanie *IRQ*, przy czym oprócz ustawienia znacznika *I* ustawia się też znacznik *B*. (Tu ostrzeżenie dla Czytelników związanych z procesorem Motorola 6800: adres początku programu jest brany z tych samych komórek, co dla przerwania *IRQ*, tzn. \$FFFE i \$FFFF).

3.10. Tryby adresowania

Różnych trybów adresowania (tj. sposobów obliczenia rzeczywistego adresu argumentu) ma 6502 aż 13, a 65C02 – 15.

Zacznijmy od najmniej skomplikowanych. **Adresowanie niejawne** (ang. *implied addressing*) polega na tym, że już z treści rozkazu (pierwsza i jedyna komórka rozkazu) wynika, co i gdzie należy wykonać. (Dalej będziemy używać skrótu *IMP*). Są to rozkazy: *BRK*, *CLC*, *CLD*, *CLV*, *DEX*, *DEY*, *INX*, *INY*, *NOP*, *PHA*, *PHP*, *PHX*, *PHY*, *PLA*, *PLP*, *PLX*, *PLY*, *RTI*, *RTS*, *SEC*, *SED*, *SEI*, *TAX*, *TAY*, *TSX*, *TXA*, *TXS*, *TYA*. Nie mają one już innych trybów adresowania.

Jeśli, Czytelniku, powrócisz do poprzednich punktów, aby dokładnie przeanalizować działanie tych rozkazów, to stwierdzisz, że w każdym z tych przypadków rzeczywiście już nic nie trzeba określać: operacja wykona się jednoznacznie.

Adresowanie akumulatora (ang. *accumulator addressing*, skrót *ACC*) polega na tym, że argumentem ma być zawartość akumulatora.

Rozkazy *DEA*, *INA* mają tylko jeden tryb adresowania: adresowanie akumulatora, więc można by je włączyć do grupy z adresowaniem niejawnym.

Są jednak rozkazy: *ASL*, *LSR*, *ROL*, *ROR*, które mogą dotyczyć bądź pamięci, bądź akumulatora i dlatego ten tryb adresowania wyodrębniono jako osobny.

Adresowanie natychmiastowe lub proste (ang. *immediate addressing*, skrót *IMM*) polega na tym, że argument jest brany z drugiego bajtu rozkazu (czyli w rozkazie nie jest podany adres, lecz od razu argument) – rys. 3.19. Ten tryb adresowania mają rozkazy: *ADC*, *AND*, *BIT*, *CMP*, *CPX*, *CPY*, *EOR*, *ORA* i *SBC* oraz *LDA*, *LDX* i *LDY*. Wykorzystują one jednostkę arytmetyczno-logiczną mikroprocesora, a więc jednym z argumentów jest zawartość rejestru, o

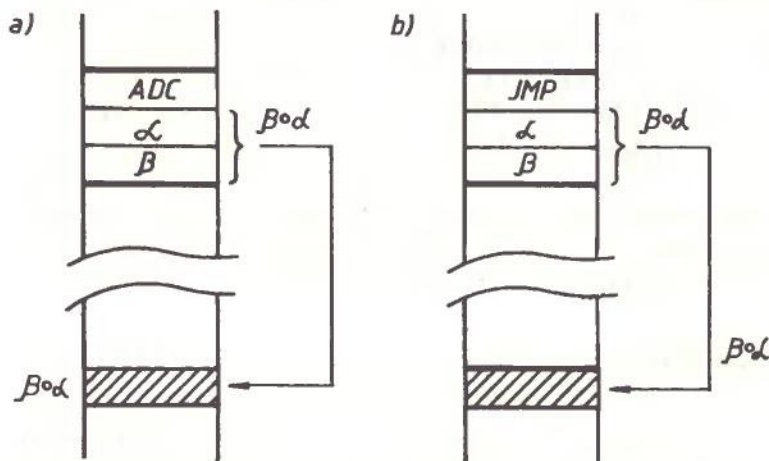
którym mowa w treści rozkazu (tj. *A*, *X* lub *Y*), a drugim właśnie argument bezpośredni.

Zwykle w ten sposób wprowadza się do programu parametry stałe.



Rys. 3.19. Format rozkazu i położenie argumentu dla adresowania natychmiastowego

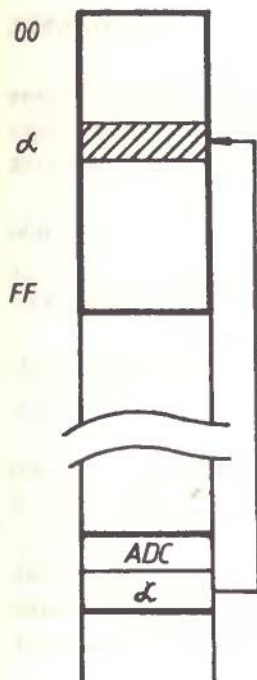
Adresowanie bezwzględne (ang. *absolute addressing*, skrót *ABS*) polega na wzięciu z drugiego (mniej znaczącej części) i trzeciego bajtu (bardziej znaczącej części) adresu argumentu (rys. 3.20). Na rysunkach zaznaczono ukośnymi kreskami argument brany do działań arytmetycznych, logicznych itp. a kreskami pionowymi – pierwszy bajt rozkazu, do którego się skacze. Ten tryb adresowania mają: *ADC*, *AND*, *ASL*, *BIT*, *CMP*, *CPX*, *CPY*, *DEC*, *EOR*, *INC*, *JMP*, *JSR*, *LDA*, *LDX*, *LDY*, *LSR*, *ORA*, *ROL*, *ROR*, *SBC*, *STA*, *STX*, *STY*, *STZ*, (*TRB* i *TSB*).



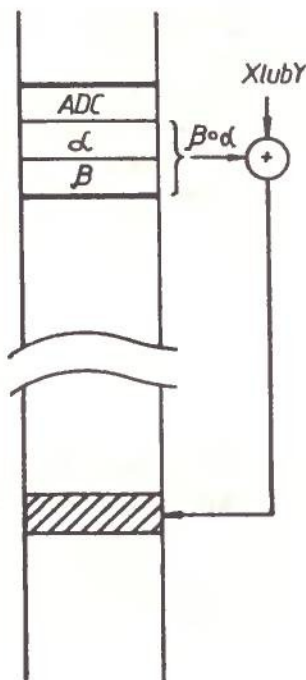
Rys. 3.20. Format rozkazu i położenie argumentu dla adresowania bezwzględnego: a) rozkaz *ADC*; b) rozkaz *JMP*

Jest to najliczniejsza grupa rozkazów. Ten tryb adresowania umożliwia dostęp do każdej komórki przestrzeni adresowej mikrokomputera, gdyż dwa bajty dają adres 16-bitowy i taki jest też rozmiar magistrali adresowej mikroprocesora.

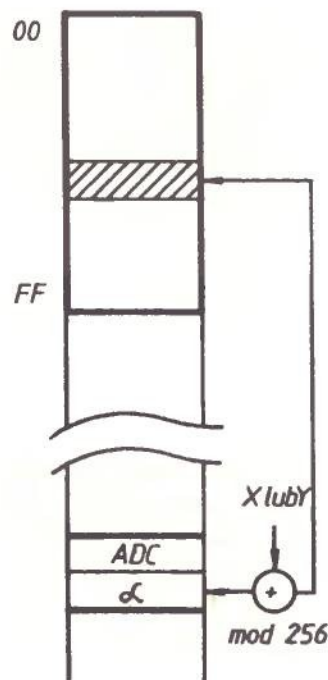
Adresowanie na stronie zerowej (ang. *zero page addressing*, skrót *ZPG*) polega na tym, że zamiast dwóch bajtów adresu w rozkazie podaje się tylko jeden. Umożliwia to zaadresowanie tylko 256 komórek pamięci (bo $2^8 = 256$):



Rys. 3.21. Format rozkazu i położenie argumentu dla adresowania na stronie zerowej



Rys. 3.22. Format rozkazu i położenie argumentu dla adresowania indeksowego



Rys. 3.23. Format rozkazu i położenie argumentu dla adresowania indeksowego na stronie zerowej

o adresach \$00 ÷ \$FF (rys. 3.21). Zaletą jest skrócenie programu i szybsze jego wykonywanie, o ile programista umieści większość komórek roboczych swojego programu na stronie zerowej (czyli w obszarze \$0000 ÷ \$00FF).

Strona zerowa jest zresztą używana do innych trybów adresowania (patrz niżej); w pewnym sensie jest ona rozszerzeniem wewnętrznych rejestrów mikroprocesora.

Grupa rozkazów wyposażona w ten tryb adresowania jest taka sama jak wyżej, za wyjątkiem JMP i JSR (bo na stronie zerowej zwykle się nie umieszcza programów).

Adresowanie indeksowe bezwzględne (ang. *absolute indexed addressing*, skróty ABS X lub Y) jest podobne do bezwzględnego, jednakże 16-bitowy adres wzięty z 2 i 3 bajtu rozkazu jest modyfikowany przez dodanie zawartości rejestru indeksowego X bądź rejestru indeksowego Y (rys. 3.22). Umożliwia to łatwy dostęp do tablicowanych danych: wtedy w rozkazie podaje się adres początku tablicy, a w rejestrze X (bądź Y) należy przedtem zapamiętać numer elementu tablicy (liczony od 0, a nie od 1!).

Ten sposób dostępu do tablic jest możliwy dla tablic o wielkości do 256 elementów. Przy większych tablicach należy użyć nieco innych sposobów. Są dwie grupy rozkazów z tym trybem adresowania: z użyciem rejestru X: ADC, AND, ASL, BIT, CMP, DEC, EOR, INC, LDA, LDY, LSR, ORA, ROL,

ROR, SBC, STA (i STZ). Grupa z użyciem rejestru Y jest mniejsza: ADC, AND, CMP, EOR, LDA, LDX, ORA, SBC i STA.

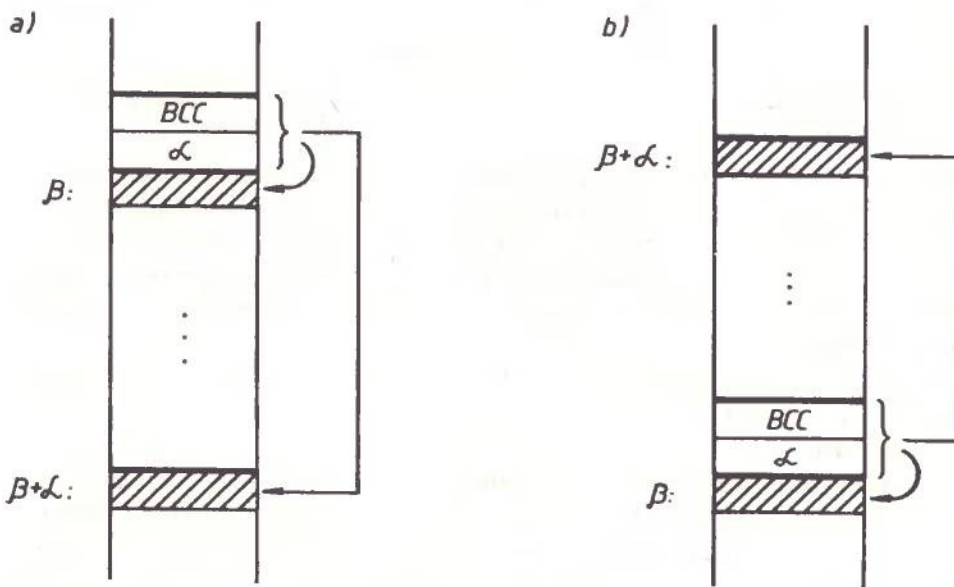
Jeśli ograniczyć się do strony zerowej, to powstanie **adresowanie indeksowe na stronie zerowej** (skrót ZPG, X i Y). Polega ono na tym, że do jednego bajtu adresu wziętego z rozkazu dodaje się zawartość rejestru X (bądź Y), a wynik obcina się do ośmiu bitów, tj. do zakresu $\$00 \div \FF (rys. 3.23).

Napis „mod 256” pod symbolem dodawania oznacza dodawanie modulo 256, tzn. obcinanie wyniku do zakresu $0 \div 255$ ($7+8 = 5_{\text{mod } 10}$; $255+2 = 1_{\text{mod } 256}$).

Zalety i wady są takie same jak przy wszystkich trybach adresowania strony zerowej.

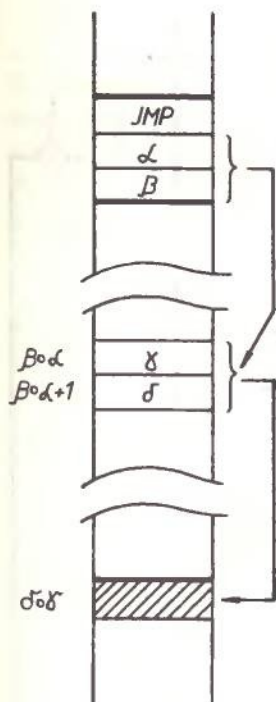
Grupa rozkazów z użyciem rejestru X jest taka sama jak poprzednio, ale bez STY. Do grupy rozkazów z użyciem rejestru Y należą tylko dwa: LDX i STX.

Adresowanie względne (ang. *relative addressing*, skrót REL) mają tylko rozkazy skoków: BCC, BCS, BEQ, BMI, BNE, BPL, (BRA), BVC i BVS. Drugi bajt rozkazu (w kodzie U2, a więc w zakresie $-128 \dots +127$) podaje adres skoku względem początku następnego rozkazu (rys. 3.24). Zaletą jest (jak zwykle) skrócenie programu i jego szybsze wykonanie, a wadą niemożliwość skakania dalej niż pół strony. (Ale rozkazem JMP można skoczyć wszędzie).

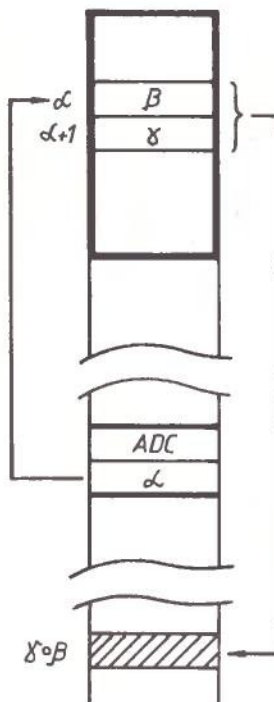


Rys. 3.24. Format rozkazu i położenie następnego wykonywanego rozkazu dla adresowania względnego

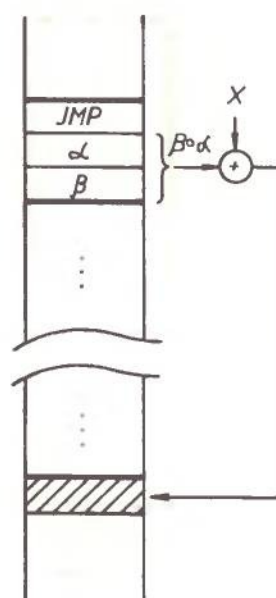
Adresowanie pośrednie (ang. *absolute indirect addressing*, skrót ABS) ma tylko rozkaz JMP. Polega to na tym, że dwa bajty wzięte z rozkazu skoku wyznaczają adres komórki, z której procesor pobiera mniej znaczący bajt (a z następnej — bardziej znaczący) miejsca w pamięci, do którego skacze (rys. 3.25).



Rys. 3.25. Format rozkazu i położenie następnego wykonywanego rozkazu dla adresowania pośredniego



Rys. 3.26. Format rozkazu i położenie argumentu dla adresowania pośredniego na stronie zerowej



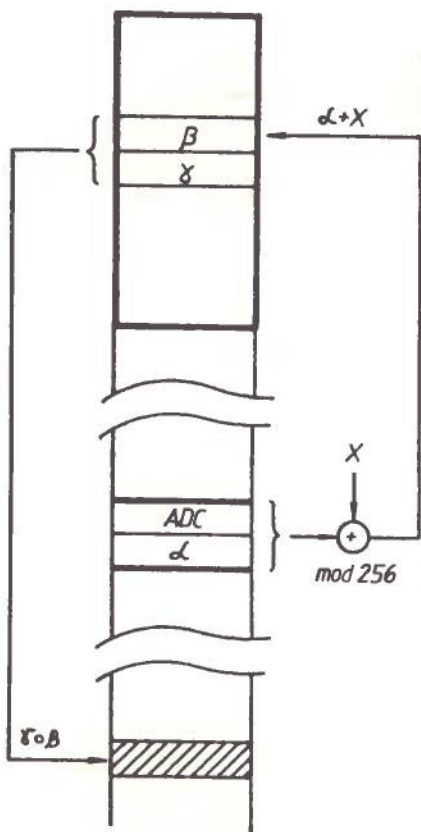
Rys. 3.27. Format rozkazu i położenie następnego wykonywanego rozkazu dla adresowania indeksowego pośredniego

Adresowanie pośrednie na stronie zerowej (ang. zero page indirect addressing, skrót (ZPG)) polega na wzięciu drugiego bajtu z rozkazu. Wskazuje on na pewną komórkę na stronie zerowej. Zawartość tej i następnej komórki daje pełny adres argumentu (rys. 3.26). Rozkazy mające ten tryb adresowania to: ADC, AND, CMP, EOR, ORA, SBC (dwuargumentowe) oraz LDA i STA (jednoargumentowe). W rozkazach dwuargumentowych drugi argument (i wynik) jest w akumulatorze.

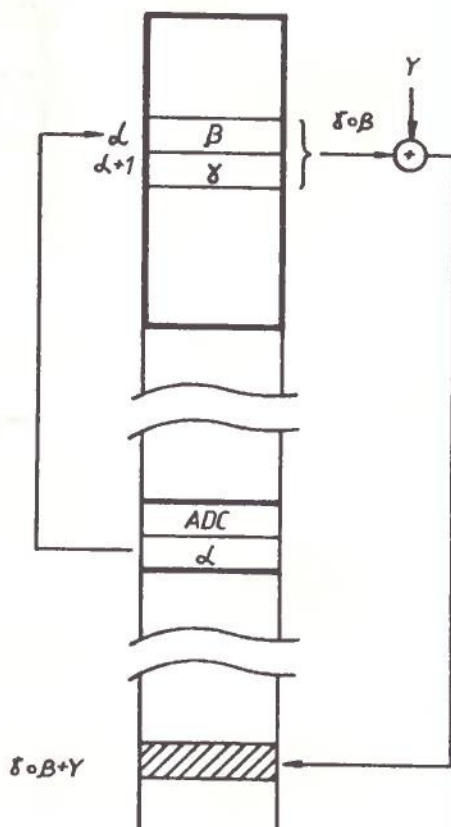
Adresowanie indeksowe pośrednie (ang. absolute indexed indirect addressing, skrót (ABS, X)) ma tylko rozkaz JMP. W stosunku do pośredniego różnica polega na tym, że dwa bajty z rozkazu tworzą adres, do którego dodaje się zawartość rejestru X i tak powstały adres wskazuje na komórkę, z której procesor pobiera mniej znaczący bajt (a z następnej — bardziej znaczący) adresu, dokąd wykona skok (rys. 3.27).

Istnieją jeszcze dwa tryby adresowania służące do łatwego dostępu do tablic i podobnych struktur danych.

Adresowanie indeksowe pośrednie na stronie zerowej (ang. zero page indexed indirect addressing, skrót (ZPG, X)). Polega ono na tym, że drugi bajt rozkazu, po dodaniu do niego zawartości rejestru X i obcięciu wyniku do zakresu strony



Rys. 3.28. Format rozkazu i położenie argumentu dla adresowania indeksowego pośredniego na stronie zerowej



Rys. 3.29. Format rozkazu i położenie argumentu dla adresowania pośredniego indeksowego

zerowej, wskazuje na komórkę (i następną), gdzie znajduje się adres argumentu (rys. 3.28). Ten tryb¹ adresowania mają rozkazy: ADC, AND, CMP, EOR, LDA, ORA, SBC i STA.

Zupełnie inaczej działa **adresowanie pośrednie indeksowe** (ang. *indirect indexed addressing*, skrót (ZPG, Y)). Drugi bajt rozkazu wskazuje na pewną komórkę na stronie zerowej. Zawartość tej i następnej komórki tworzy pewien adres, który jest zwiększany o zawartość rejestru Y. Tak powstały adres wskazuje na argument – rys. 3.29. Zbiór rozkazów dysponujących tym trybem jest identyczny jak poprzednio.²

¹ Jest on użyteczny np. przy pośrednim adresowaniu urządzeń. Do kolejnych par komórek pewnej tablicy należy wpisać adresy urządzeń (nie muszą być kolejne ani w żaden sposób ze sobą powiązane). Rozkazem LDA (POCZT, X) można teraz odczytać dane z urządzenia, o ile POCZT jest adresem początku tablicy, a w rejestrze X jest zapisany podwojony numer urządzenia.

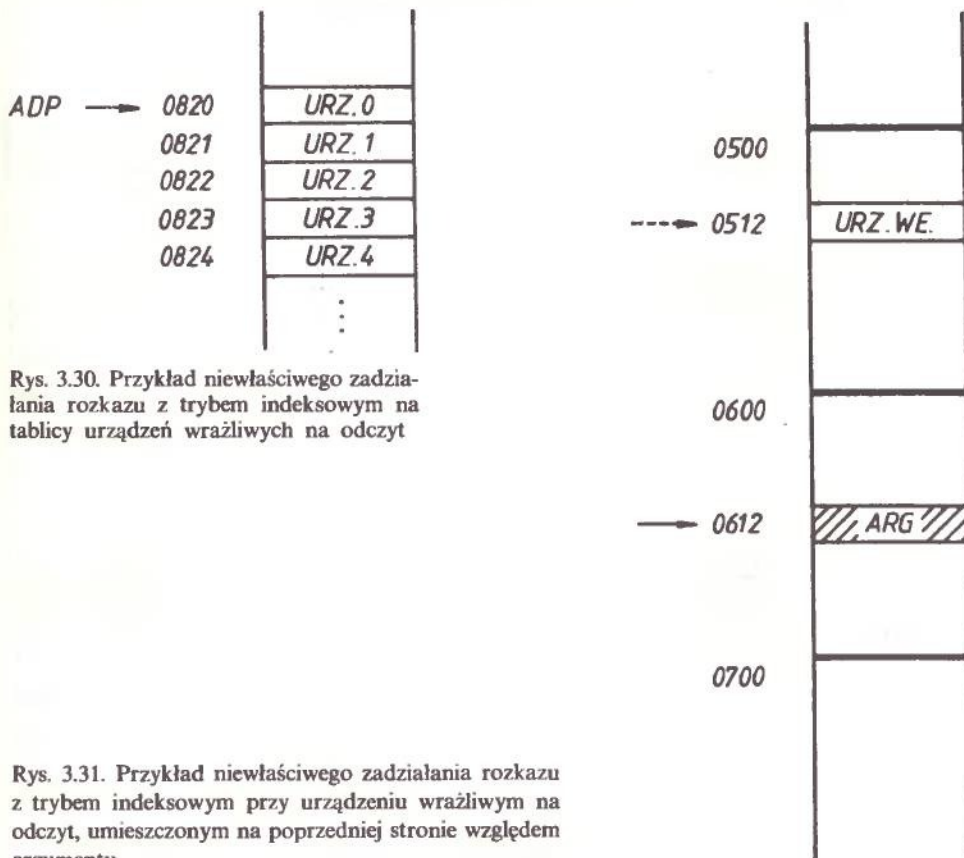
² Ten tryb adresowania jest przydatny przy stabilizowanych danych. Dla małych tablic (do 256 elementów) dostęp do danych uzyskuje się np. rozkazem LDA (PT, Y, gdzie PT jest adresem początku tablicy, a Y zawiera numer elementu. Dla dużych tablic (powyżej 256 elementów) należy zastosować sposób opisany w Dodatku 6.

3.11. Uwagi dodatkowe

Ze schematów pokazanych w poprzednim punkcie wynika, że 6502 czasem czyta pamięć niepotrzebnie. Dzieje się tak dlatego, że jest on pozbawiony sygnału *VMA* (ang. *valid memory address* – właściwy adres pamięci) w stosunku do mikroprocesora Motorola 6800, a praca na zakładkę powoduje czasem wysłanie po magistrali adresowej nie całkiem dobrego adresu.

Przypadki mogą być następujące:

1. Czytanie argumentu bez modyfikacji zawartości rejestru (tryby ZPG, X i Y, (ZPG, X)).
2. Czytanie argumentu o stronę niżej przy przekroczeniu granic strony (tryby ABS, X i Y, (ZPG), Y).
3. Czytanie o stronę niżej lub wyżej (REL).
4. Zbędne czytanie ze stosu (ABS 11, IMP 12, 13, 7, MD 3).
5. Czytanie ostatniego bajtu rozkazu (ABS 10, ZPG 10).
6. Zbędne czytanie następnego kodu rozkazu (tryby IMP, ACC, IMP 12, 13, 14, HD 3, REL).
7. Zbędny zapis starego argumentu (tryby X4 – bez ACC4).



Rys. 3.30. Przykład niewłaściwego zadziałania rozkazu z trybem indeksowym na tablicy urządzeń wrażliwych na odczyt

Rys. 3.31. Przykład niewłaściwego zadziałania rozkazu z trybem indeksowym przy urządzeniu wrażliwym na odczyt, umieszczonym na poprzedniej stronie względem argumentu

W niektórych przypadkach używanie pewnych rozkazów może doprowadzić do błędnego działania mikrokomputera.

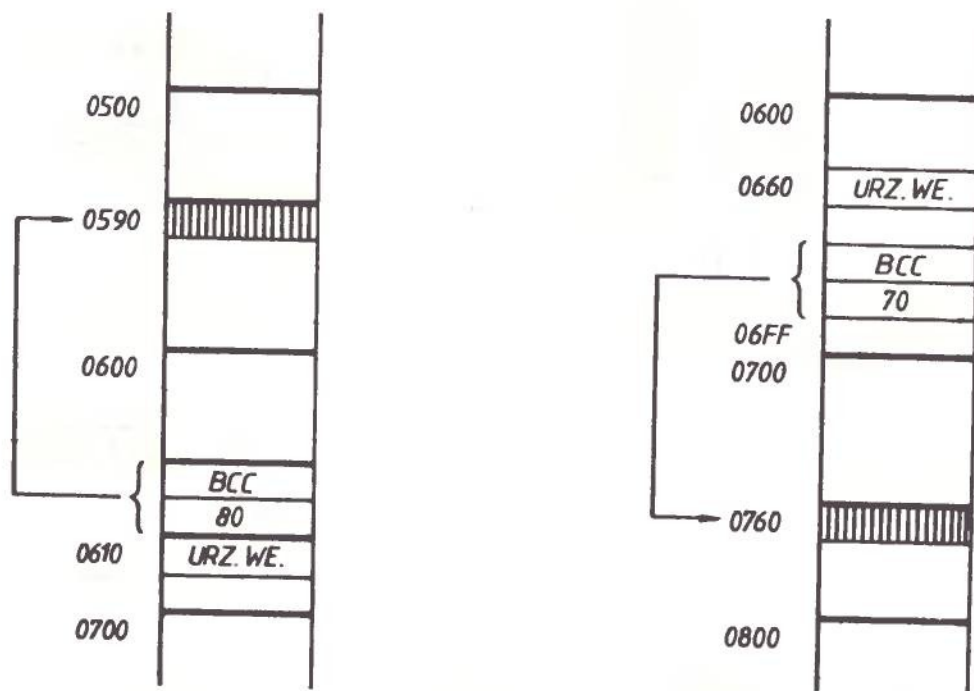
Ad 1. Jeśli urządzenia wejściowe są ułożone w ciąg (tablicę) i ich adresowanie jest dokonywane np. rejestrem X poczynając od numeru 0, i jeśli urządzenie o numerze 0 ma przerzutnik zgłoszenia automatycznie kasowany w czasie odczytu danych spod tego adresu (albo zalicza liczbę odczytów danych), to odczyt danych z innych urządzeń rozkazem typu ZPG X i Y lub ZPG, X skasuje także zgłoszenie urządzenia 0 (rys. 3.30). Tutaj odczyt urządzenia 3 rozkazem LDA ADP, X spowoduje odczyt zarówno urządzenia 0, jak urządzenia 3.

Ad 2. Jeśli poniżej strony z komórkami roboczymi programu znajduje się strona z urządzeniami wejściowymi wrażliwymi na odczyt, i jeśli adresem bazowym jest adres ze strony, na której znajduje się urządzenie peryferyjne, to nastąpi złe działanie (rys. 3.31).

Ad 3. Jeśli na tej samej stronie, co rozkaz skoku warunkowego, znajduje się urządzenie wrażliwe na odczyt (bardzo nieprawdopodobne) — rys. 3.32.

Ad 4, 5. Nie dają szkodliwych efektów.

Ad 6. Jeśli rozkaz tego typu znajdzie się na końcu obszaru zajętego przez pamięć programu, a następna komórka należy do peryferii wrażliwych na niepożądany odczyt (patrz 1), to nastąpi złe działanie (przykład na rys. 3.33).



Rys. 3.32. Przykład niewłaściwego oddziaływania rozkazu skoku względnego na urządzenie wejściowe umieszczone na tej samej stronie: przy skoku \$80 (= -128) w tył oraz przy skoku w przód \$70 (= 112)

Może to się zdarzyć tylko wtedy, gdy blok programu kończy się instrukcją skoku warunkowego.

Ad 7. Jeśli są urządzenia wrażliwe na zbędny zapis (np. wysyłające na zewnątrz sygnał gotowości po cyklu zapisu, to użycie rozkazu typu X 4 spowoduje błędne działanie.

Ponadto rozkaz JMP (XX, FF) nie wykonuje się prawidłowo, bo mniej znaczącą część adresu pośredniego pobiera z komórki \$XXFF, ale bardziej znaczącą z \$XX00, zamiast z (\$XX+1), \$00 (brak przeniesienia przy zwiększaniu adresu o 1).

07FC	BCS
07FD	A0
07FE	BCC
07FF	D0
0800	URZ. WE.

Rys. 3.33. Złe oddziaływanie na urządzenie wejściowe na skutek odczytania na zapas kodu następnego rozkazu

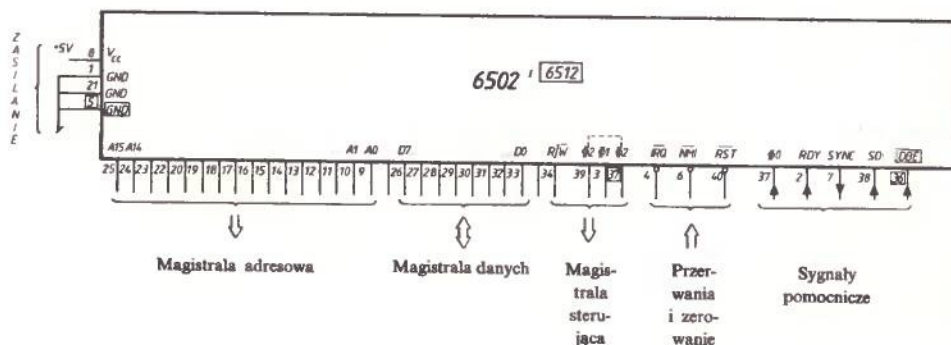
Niektóre ciągi bitów, jeśli 6502 potraktuje je jako kody rozkazów, mogą spowodować złe działanie. Są to wszystkie ciągi, którym odpowiadają puste kratki w tablicy w Dodatku 2 — 6502 traktuje takie kody jako jakieś rozkazy (ale nikt nie wie — jakie). Co gorsza, po napotkaniu kodu X2 procesor już nie wraca do normalnej pracy (nawet jeśli potem trafi na poprawne rozkazy). Trzeba wówczas użyć sygnału RST. W normalnych warunkach takie sytuacje nie występują, ale przy błędach w programie może się to zdarzyć.

65C02 nie ma tej wady. Wszystkie nie wykorzystane kody wykonują się jako NOP, tyle że zajmują różną liczbę komórek w pamięci i mają różne czasy wykonania (Dodatek 5). Wynika to z uproszczenia budowy dekodera rozkazów (wewnątrz mikroprocesora).

4

Budowa i działanie mikroprocesora

Na rys. 2.1 została już przedstawiona struktura systemu mikroprocesorowego (mikrokomputera). Pora teraz dokładnie objaśnić sygnały tam występujące. Rysunek 4.1 pokazuje mikroprocesor 6502 (oraz jego mutację 6512) i jego



Rys. 4.1. Rozkład wyprowadzeń mikroprocesorów 6502 i 6512

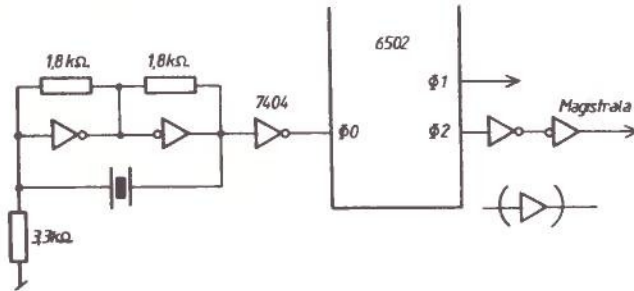
sygnały wraz z numerami nóżek i ich nazwami. W kolejnych punktach omówimy kolejno ich rolę. Sygnały w prostokątnych ramkach występują tylko w procesorze 6512.

4.1. Zasilanie

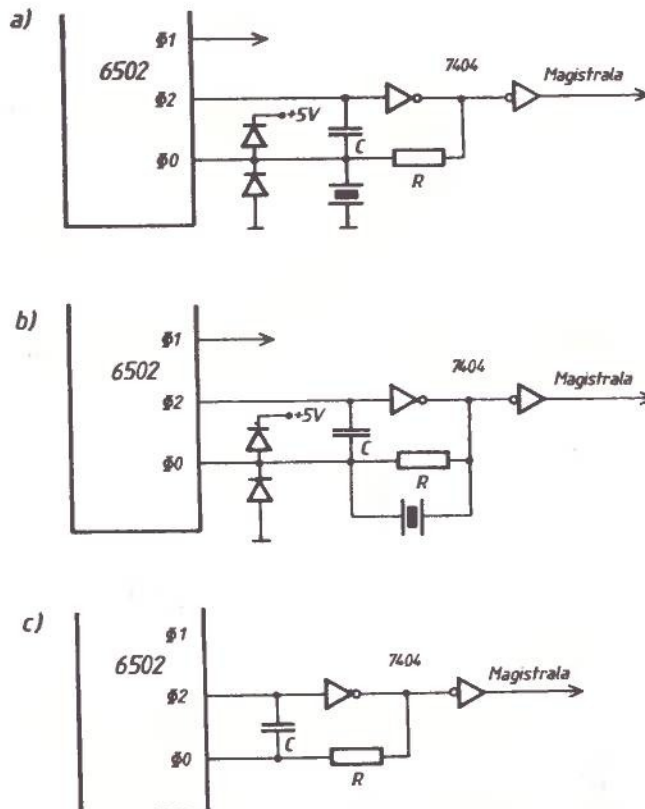
Mikroprocesor 6502 wymaga tylko jednego napięcia zasilającego: +5V (nóżka 8) w stosunku do masy (nóżki 1 i 21 — obie muszą być zwarte). Pobór prądu zależy od częstotliwości zegara i do 2 MHz wynosi typowo 90 mA (max 140 mA), a przy 3 MHz — 100 mA (max 160 mA).

4.2. Zegar

Do poprawnej pracy procesora (i innych elementów systemu) wymagane jest wytwarzanie sygnału taktującego (zegara) o częstotliwości 1 MHz (albo więcej – w zależności od wariantu, bo wielu producentów produkuje także wersje szybsze: z oznaczeniem 6502A ma maksymalną częstotliwość 2 MHz, a z oznaczeniem B – 3 MHz).



Rys. 4.2. Sposób wykorzystania sygnałów $\Phi 0$, $\Phi 1$ i $\Phi 2$



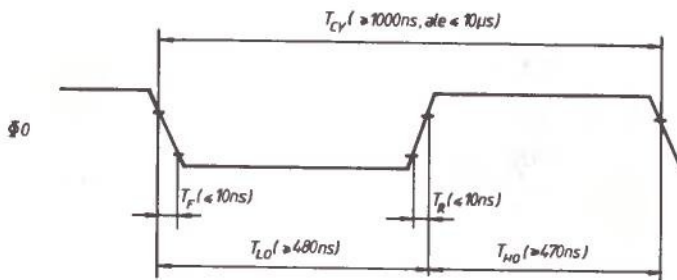
Rys. 4.3. Inne sposoby wykonania zegara systemu mikroprocesorowego: a) z równolegle włączonym rezonatorem kwarcowym; b) z szeregowo włączonym rezonatorem kwarcowym; c) bez rezonatora kwarcowego

Mikroprocesory 6502... 6507 (różnice między nimi będą opisane później) mają trzy nóżki służące do wytworzenia sygnałów zegarowych: $\Phi 0$ (wejście), oraz $\Phi 1$ i $\Phi 2$ (wyjścia). Sposób ich wykorzystania może być dwojaki:

— do wejścia $\Phi 0$ doprowadza się prostokątny sygnał o odpowiedniej częstotliwości, a $\Phi 2$ (czasem także $\Phi 1$) są dołączone do magistrali systemu (rys. 4.2) — ten sposób jest najlepszy;

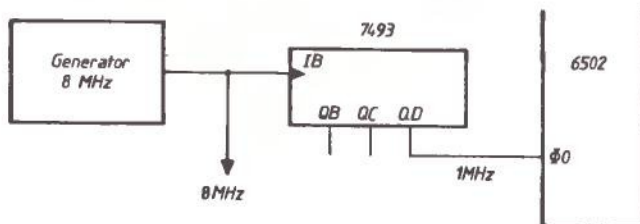
— generator kwarcowy (lub RC) jest utworzony z bramki negacji i wewnętrznych bramek mikroprocesora między sygnałem $\Phi 0$ a $\Phi 2$ (rys. 4.3).

Generator nie musi być taki, jak na rys. 4.2: wystarczy, że spełnia warunki czasowe z rys. 4.4 oraz ma poziomy sygnałów zgodnie ze standardem TTL.



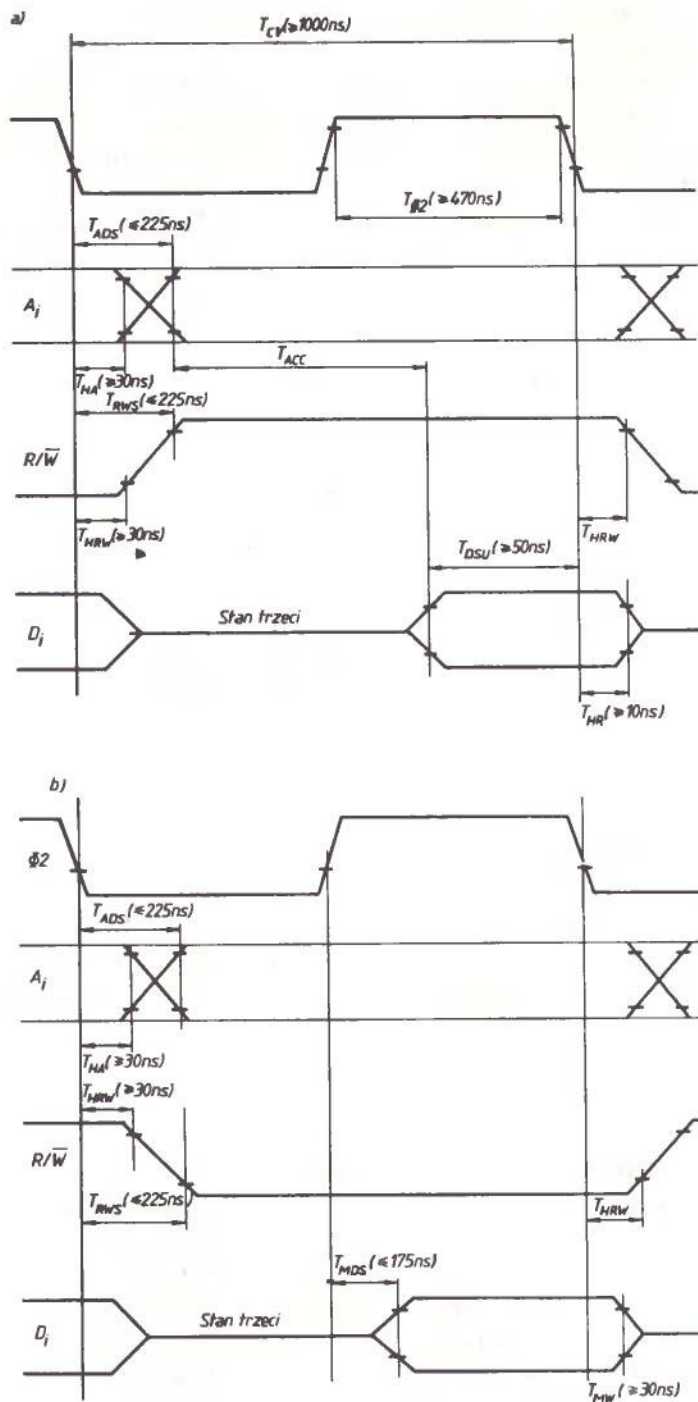
Rys. 4.4. Wymagania czasowe na wejściowy sygnał zegara $\Phi 0$

Jeżeli w urządzeniu, w którego skład wchodzi mikroprocesor, jest już generator o potrzebnej częstotliwości (albo jej wielokrotności), to można go wykorzystać (ewentualnie dzieląc tę częstotliwość), o ile sygnał dochodzący do wejścia $\Phi 0$ ma wypełnienie 1/2. Przykład pokazuje rys. 4.5. Rozwiązania generatorów z rys. 4.3 różnią się sposobem dołączenia rezonatora kwarcowego (lub jego brakiem). To ostatnie rozwiązanie jest odpowiednie wtedy, gdy system nie musi mieć dokładnych ani stabilnych częstotliwości, a ma być tani. Wartości C i R należy określić posługując się katalogiem tego producenta, którego mikroprocesorów używamy.



Rys. 4.5. Przykład wykorzystania zewnętrznego generatora o większej częstotliwości

Wspólną cechą układów z rys. 4.2 i 4.3 są dwa negatory wtrącone między sygnał $\Phi 2$ z mikroprocesora a magistralę. Służą one do wzmocnienia tego sygnału, gdyż mikroprocesor wytrzymuje obciążenie jedną bramką TTL i 130 pF. Jako wzmacniacz może służyć też bramka nienegująca (np. 7408, 7432). Można te bramki pominąć, jeżeli system jest mały i układy scalone nie wnoszą obciążenia większego niż jeden układ TTL ani 130 pF. W żadnym jednak przypadku opóźnienie wnoszone przez wzmacniacze sygnału $\Phi 2$ nie może być zbyt duże — użycie więcej niż dwóch bramek TTL nie jest dopuszczalne.

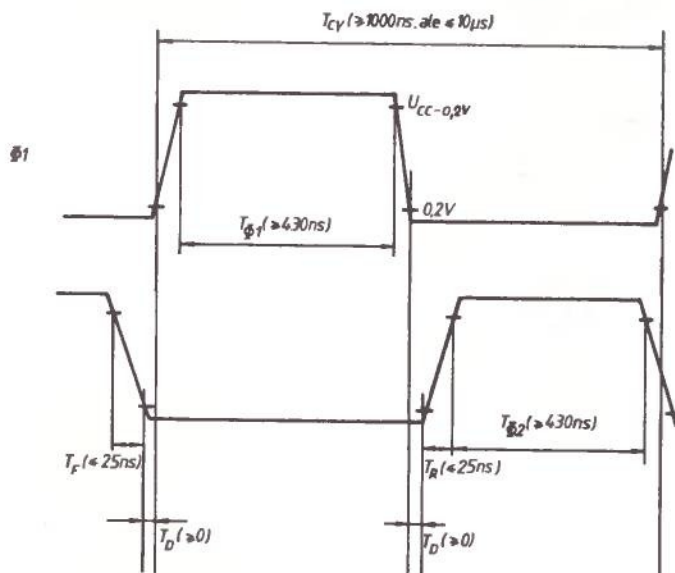


Rys. 4.6. Zależności czasowe magistrali (dla wersji 1 MHz): a) cykl odczytu; b) cykl zapisu

Sygnał $\Phi 1$ podlega podobnym ograniczeniom, ale w praktyce rzadko jest używany.

Na rys. 4.6 są podane czasy dla cyklu odczytu (a) oraz zapisu (b).

Mikroprocesory o numerach zwiększonych o 10 (tzn. 6512...17) nie mają wewnątrz układu zegara. Zamiast tego sygnały $\Phi 1$ i $\Phi 2$ muszą być doprowadzone do mikroprocesora z zewnątrz (tym razem muszą one mieć amplitudę większą niż sygnał TTL — rys. 4.7). Jest to wygodne w systemach wielo-



Rys. 4.7. Wymagania czasowe na wejściowe sygnały zegara $\Phi 1$ i $\Phi 2$ mikroprocesorów 6512 ÷ 6515

procesorowych albo gdy ktoś chce zastąpić mikroprocesor Motorola 6800 procesorem 6512 (są one zamienne z dokładnością do nóżki — ang. *pin compatible*, a ponadto należy jeszcze wymienić pamięci stałe z programami).

4.3. Magistrale

Parametry obciążeniowe sygnałów magistrali: adresowej, danych i sterującej — są identyczne jak dla układów TTL (obciążenie jedną jednostką) z wyjątkiem wejść, których prądy upływności są rzędu 10 μA (dokładnych danych, jak zwykle, trzeba szukać w katalogu właściwego producenta).

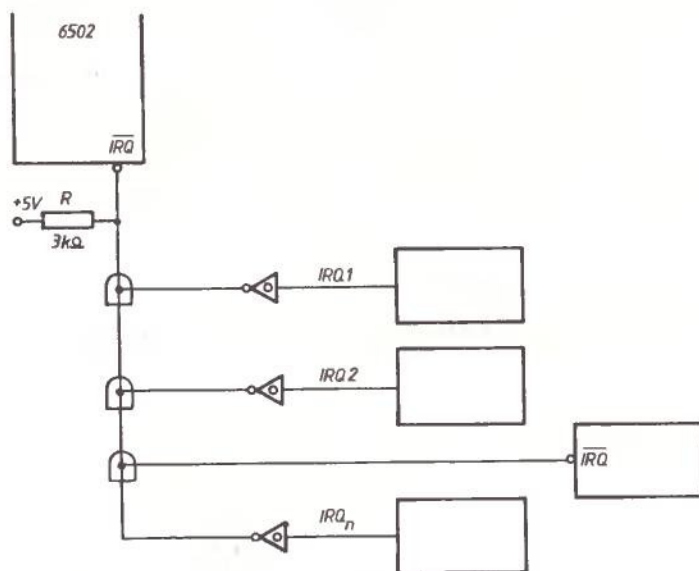
Parametry czasowe tych sygnałów wynikają już z rys. 4.5. Należy tu jednak zwrócić uwagę, aby przy sprawdzaniu zależności czasowych nie pominąć czasów opóźnień wnoszonych np. przez dekodery adresu itp. (trzeba je doliczyć do T_{AS} lub T_{RWS}) oraz przez wzmacniacze (bufory) danych (wtedy trzeba je dodać do T_{DSU} oraz do T_{MDS}). Przykłady stosownych projektów podają następne rozdziały.

Tym, którzy znają mikroprocesor Motorola 6800, przypominamy, że magistrale 6502 i 6800 są identyczne z dokładnością do sygnału VMA , którego tu brak. Zaletą tego rozwiązania jest uproszczenie dekodatorów adresów (bo nie trzeba do nich dołączać tego sygnału). Na magistrali adresowej nie pojawiają się „fałszywe” adresy w cyklach zapisu, tylko czasem w cyklach odczytu (patrz p. 3.11).

Sygnał DBE (ang. *data bus enable*) występuje tylko w 6512 i służy do uaktywniania bramek trójstanowych na wyjściach danych w cyklach zapisu. Z tego wynika, że należy tam dołączyć sygnał $\Phi 2$ albo $\Phi 2$ pomnożony logicznie przez negację tego sygnału, który ma blokować magistralę danych w pewnych cyklach.

4.4. Przerwania i zerowanie

O tym, jak jest obsługiwane przerwanie, pisaliśmy już w rozdz. 3. Wejście IRQ jest zaopatrzone w negator, a więc zgłoszenie przerwania polega na wymuszeniu na tym wejściu zera. Jeśli kilka urządzeń ma być podłączonych do tego wejścia, to należy to zrobić na przykład tak jak na rys. 4.8. Kilka źródeł sygnałów — $IRQ1, IRQ2, \dots, IRQ_n$ — jest dołączonych do wejścia IRQ poprzez negatory z otwartym kolektorem. Opornik R (zwykle $3\text{ k}\Omega$) jest konieczny i wymusza powrót sygnału do stanu 1 po zniknięciu wszystkich sygnałów IRQ . Połączenie wyjść bramek ze sobą realizuje w technice TTL iloczyn logiczny; dlatego właśnie zastosowano negatory, które według praw de



Rys. 4.8. Przykład dołączenia kilku urządzeń zgłaszających przerwania IRQ do wejścia IRQ mikroprocesora

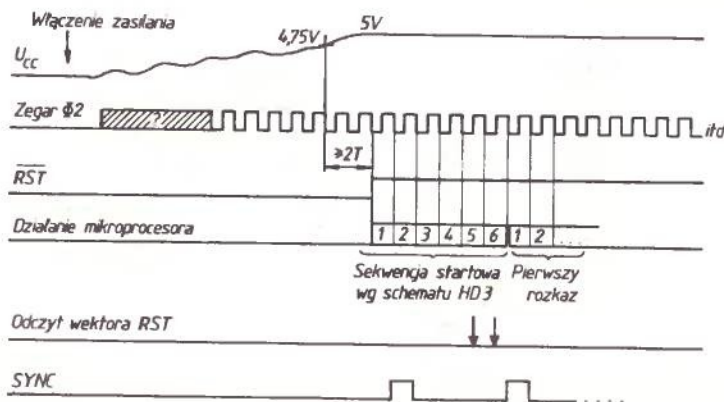
Morgana umożliwiają uzyskanie sumy logicznej sygnałów. Na rys. 4.8 bezpośrednie połączenia przewodów oznaczono symbolami bramek iloczynu – dla zaznaczenia ich funkcji. W literaturze amerykańskiej na takie połączenie mówi się *wired-and* (iloczyn montażowy) albo *wired-or* (suma montażowa). Jeśli układ scalony ma wyjście z negatorem ($\overline{IRQ}$) i z otwartym kolektorem, to należy je dołączyć bezpośrednio.

To rozwiązanie (jak już było powiedziane wcześniej) wymaga od programu obsługi przerwań, aby dokonywał przepatrywania stanu urządzeń po każdym zgłoszeniu przerwania (mikroprocesor „nie wie” od razu, kto je zgłosił). To wymaga czasu, a jeśli program ma być szybki, to można zrobić układ opisany w rozdz. 5.

Istnieje jeszcze drugie wejście przerwań: $\overline{NMI}$. Wejście $\overline{NMI}$ jest wrażliwe na zbocze (opadające), a nie na poziom i nie jest maskowalne. Z tego wynika, że układ nie może mieć zwykłej bramki sumy w drodze sygnału $\overline{NMI}$ (bo jedna jedynka logiczna „zatka” tę bramkę). Ponadto trzeba starannie przeanalizować, czy następne przerwanie nie przeszkodzi w obsłudze przerwania $\overline{IRQ}$. Do $\overline{NMI}$ dołącza się zwykle tylko jeden sygnał (o bardzo wysokiej randze), np. sygnał zaniku zasilania albo sygnał pracy krokowej (do przerywania pracy programu po każdym rozkazie).

Ze schematów HD1, 2 i 3 (zamieszczonych w Dodatku 3) wynika duże podobieństwo działania mikroprocesora po wyzerowaniu go sygnałem $\overline{RST}$ do działania po zgłoszeniu przerwań. Konieczność istnienia i używania tego sygnału stanie się oczywista, jeśli sobie uświadomić, że po włączeniu zasilania zawartość wszystkich rejestrów wewnętrznych mikroprocesora może być (i jest) przypadkowa. To samo dotyczy pamięci zapisywalnych itp. urządzeń (ale nie dotyczy pamięci stałych).

Jeśli zatem można by tuż po włączeniu zasilania procesora załadować do jego rejestru PC jakąś stałą liczbę, to potraktowałby ją jako adres początku programu „rozruchowego”. Tak też się dzieje: w takcie 6 i 7 sekwencji $\overline{RST}$ procesor czyta spod adresów $\$FFFC$ i D . Dane tam umieszczone nie mogą być



Rys. 4.9. Zachowanie się mikroprocesora po włączeniu zasilania

przypadkowe (i musi to być pamięć stała). Zostaną one odczytane i system zacznie pracę od podanego adresu (cały program „rozruchowy” też musi być umieszczony w pamięci stałej, żeby nie zniknął po wyłączeniu komputera).

Rysunek 4.9 pokazuje przebiegi w układzie po włączeniu zasilania. Napięcie U_{CC} rośnie (dość powoli w tej skali czasu) do 5 V. Układ zegara zaczyna działać jeszcze przed osiągnięciem 5 V, a na pewno działa poprawnie przy 4,75 V. Tak samo mikroprocesor: przy 4,75 V jest już gotów do normalnej działalności. Od tej chwili powinien jeszcze przez co najmniej dwa takty dostawać sygnał zerowania (tj. $\overline{RST} = 0$). Po zniknięciu tego sygnału wykonuje się sekwencja według schematu HD3, a pierwszy takt po niej należy już do pierwszego rozkazu programu rozruchowego. Sygnał $\overline{SYNC}$ (równy 1 w taktach pobierania kodu rozkazu) pojawia się dodatkowo w drugim takcie sekwencji startowej. Sygnał $\overline{RST}$ można uaktywnić w dowolnej chwili w pracującym mikrokomputerze (bez wyłączania zasilania). Spowoduje to rozpoczęcie pracy systemu od początku.

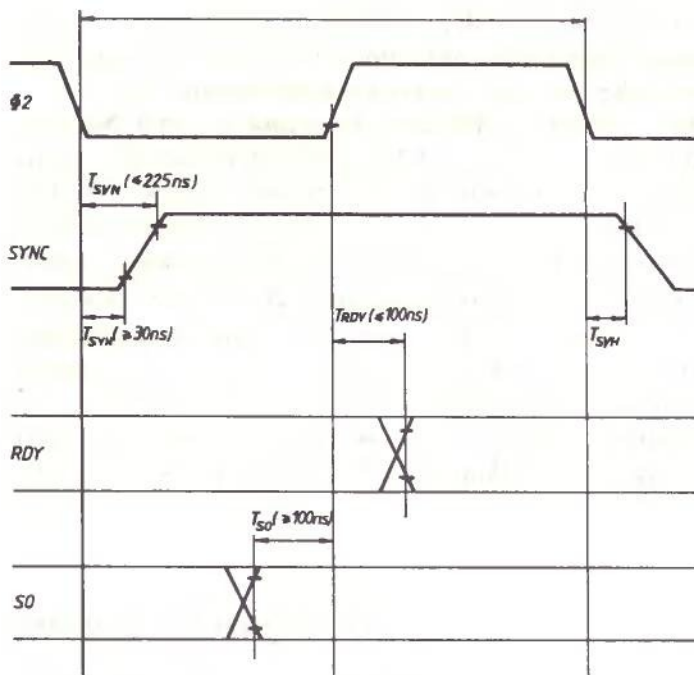
4.5. Sygnały dodatkowe

Osobnego omówienia wymagają sygnały dodatkowe: $\overline{SYNC}$, $\overline{RDY}$ i $\overline{SO}$. Ich zależności czasowe pokazuje rys. 4.10. Sygnał $\overline{SYNC}$ jest równy 1 w tych cyklach, w których następuje pobranie kodu rozkazu. W pozostałych cyklach jest równy 0 (z wyjątkiem sekwencji $\overline{RST}$). Często, w gotowym mikrokomputerze, ten sygnał nie jest używany — za to oddaje cenne usługi przy uruchamianiu i testowaniu (patrz rozdz. 7).

Sygnał $\overline{RDY}$ (ang. *ready*) oznacza gotowość urządzeń współpracujących do zakończenia przesyłania danych. Często w gotowym systemie ten sygnał jest na stałe równy 1 i nie wpływa na działanie mikroprocesora. Jeśli jednak w czasie cyklu odczytu (w 65C02 — w dowolnym cyklu) w jego pierwszej fazie sygnał $\overline{RDY}$ stanie się równy 0, to praca mikroprocesora zostaje zatrzymana (tzn. ten sam cykl powtarza się aż do zniknięcia stanu $\overline{RDY} = 0$). Wszelkie zmiany sygnał $\overline{RDY}$ może wykonywać tylko w czasie $\Phi 1$ oraz przez czas T_{RDY} poczynając od początku $\Phi 2$ (rys. 4.10). Takie działanie sygnału $\overline{RDY}$ umożliwia współpracę z pamięciami, które mają czas dostępu dłuższy niż T_{ACC} obliczony według rys. 4.6 (patrz p. 6) albo pracę krokową mikrokomputera (szczegóły w rozdz. 7).

Zauważmy jeszcze tu, że nie można inaczej zatrzymać mikroprocesora, np. przez zatrzymanie zegara (bo odstęp między kolejnymi zboczami $\Phi 0$ nie może przekroczyć 10 μs). (W mikroprocesorze 65C02 jest to możliwe).

Sygnał $\overline{SO}$ (ang.: *set overflow*) służy do ustawiania znacznika V. Służy on do celów specjalnych i odradzamy Ci, Czytelniku, pochopne jego używanie. $\overline{SO}$ może się zmieniać tylko w pierwszej fazie cyklu (do czasu T_{SO} przed rozpoczęciem $\Phi 2$) — rys. 4.10.



Rys. 4.10. Zależności czasowe dla sygnałów SYNC, RDY

4.6. Cykl rozkazowy

Zasady wykonywania każdego cyklu rozkazowego mikroprocesorów 65XX są następujące:

- każdy takt zegara $\Phi 2$ jest cyklem dostępu do pamięci;
- pierwszym taktem wykonania każdego rozkazu jest pobranie kodu tego rozkazu (wtedy pojawia się sygnał SYNC);
- w drugim takcie następuje dekodowanie rozkazu a równocześnie odczyt następnej komórki pamięci (dla rozkazów dwu- i trzabajtowych jest to pożyteczne, a dla jednobajtowych dane nie są wykorzystywane);
- w zależności od treści rozkazu są wykonywane ewentualne dalsze takty (do 5), w których mikroprocesor czyta pamięć, zapisuje do niej i wykonuje operacje wewnętrzne;
- co tylko się da, mikroprocesor wykonuje nie czekając na wynik poprzednich operacji (praca na zakładkę).

Szczegółowe opisy wykonania każdego rozkazu znajdują się w Dodatku 3 w postaci kilkudziesięciu schematów (wraz z dokładnym opisem oznaczeń).

Tutaj – tytułem przykładu – są pokazane trzy rozkazy, wykonane jeden po drugim – rys. 4.11. W komórkach \$310...\$315 są zapisane rozkazy: LDX # \$05, INX i AND \$1205. Jeślibyśmy wycieli odpowiadające im

schematy (IMM2, IMP5 i ABS4) i złożyli je w jeden ciąg, to powstałby rys. 4.11b.

Widać tu wyraźnie pracę potokową: kolumna OPERACJA WEWNĘTRZNA jest obsunięta w dół o jeden wiersz: w czasie pobrania kodu bieżącego rozkazu mikroprocesor kończy wykonanie poprzedniego rozkazu. W drugim takcie każdej sekwencji rozkazowej pobiera zawartość następnej komórki: wartość jest bądź zbędna (tutaj w rozkazie INX), bądź jest wykorzystana w czasie wykonania rozkazu (jako argument w rozkazie LDX lub jako dolna część adresu w rozkazie AND). Trzeci (i następne) takty są dodawane w cyklu rozkazowym w zależności od rozkazu, który jest dekodowany w układzie sterowania mikroprocesora w takcie 2. Na przykład rozkaz AND z rys. 4.11 wymaga jeszcze taktu 3, w którym pobiera z następnej komórki bardziej znaczącą część adresu (ADH) oraz czwartego taktu, w którym pobiera argument operacji z pamięci (z komórki o adresie złożonym z ADH i ADL).

a)

Adres	Zawartość	Rozkaz	Schemat
\$0310	\$A2	LDX #\$05	IMM 2
\$0311	\$05		
\$0312	\$E8	INX	IMP 5
\$0313	\$2D	AND \$1205	ABS 4
\$0314	\$05		
\$0315	\$12		
\$0316			
...	...	...	...
\$1205	\$0F	ARG	

b)

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0310)	+1	KOD (\$A2)	0	Dek. rozk., przesł. ARG do X
2	AP+1 (\$0311)	+1	ARG (\$05)	0	
1	(AP+2)=AP (\$0312)	+1	KOD (\$E8)	0	Nic
2	AP+1 (\$0313)	0	nKOD (\$2D)	X	Dekodowanie rozkazu
1	(AP+1)=AP (\$0313)	+1	KOD (\$2D)	0	Wykonanie X: =X+1
2	AP+1 (\$0314)	+1	ADL (\$05)	0	Dek. rozk., zapam. ADL
3	AP+2 (\$0315)	+1	ADH (\$12)	0	Zapamiętanie ADH
4	ADH, ADL (\$1205)	0	ARG (\$0F)	0	Odczyt i zapamiętanie ARG
1	AP+3 (\$0316)				Wykonanie mnożenia log.

Rys. 4.11. Umieszczenie w pamięci (a) i szczegółowe wykonanie sekwencji (b) przykładowych trzech rozkazów

Z rys. 4.11 widać też, że mikroprocesor nie traci czasu na wykonanie niektórych operacji: wykonuje je (np. zwiększenie zawartości rejestru X w rozkazie INX) w czasie pobierania kodu rozkazu następnego.

Dokładne omówienie cykli rozkazowych wszystkich rozkazów we wszystkich trybach adresowania znajduje się w Dodatku 3. Tam też są umieszczone sekwencje obsługi przerw IRQ , NMI oraz zerowania RST (nie będące wykonaniem rozkazów).

4.7. Inne typy mikroprocesorów

Oprócz mikroprocesora 6502 istnieje jeszcze kilka jego odmian o skromniejszych możliwościach (mają obudowę 28-nóżkową, czyli mniej sygnałów) i o niższej cenie (czasem to nieprawda). Oznaczone są numerami 6503, 6504, 6505, 6506, 6507 oraz 6513, 6514 i 6515. Schematy wyprowadzeń podaje tabl. 4.1. Zauważmy, że ta sama nóżka może pełnić różne role w różnych

Tablica 4.1. Rozkłady wyprowadzeń innych mikroprocesorów rodziny 65XX

	A12	A11	A10	...	A1	A0	D7...D0	R/W	$\Phi 2$	$\Phi 1$	$\Phi 0$	IRQ	NMI	PSI	$SYNC$	RDY	V_{cc}	GND
6503	—	17	16		7	6	18 25	26	28	—	27	3	4	1	—	—	5	2
6504	—	17	16		7	6	18 25	26	28	—	27	4	—	1	—	3	5	2
6505	17	16	15		6	5	18 25	26	28	—	27	3	—	1	—	—	4	2
6506	—	17	16		7	6	18 25	26	28	3	27	4	—	1	—	—	5	2
6507	17	16	15		6	5	18 25	26	28	—	27	—	—	1	—	3	4	2
6513	—	17	16		7	6	18 25	26	27	2	—	3	4	28	—	—	5	1
6514	17	16	15		6	5	18 25	26	27	2	—	3	—	28	—	—	4	1
6515	—	17	16		7	6	18 25	26	27	3	—	4	—	28	—	2	5	1

mikroprocesorach (czyli nie są one zamienne w układzie!) oraz, że niektórych sygnałów brak: na przykład 6504 może zaadresować tylko 4 K (tj. 4096) komórek pamięci, ale ma oba sygnały przerw, a 6507 może zaadresować 8 K komórek, może być wstrzymywany sygnałem RDY , ale nie ma w ogóle wejść przerw. Którego użyć — zależy od projektanta. Z punktu widzenia programowania nie ma między nimi różnic.

MIKROKOMPUTER

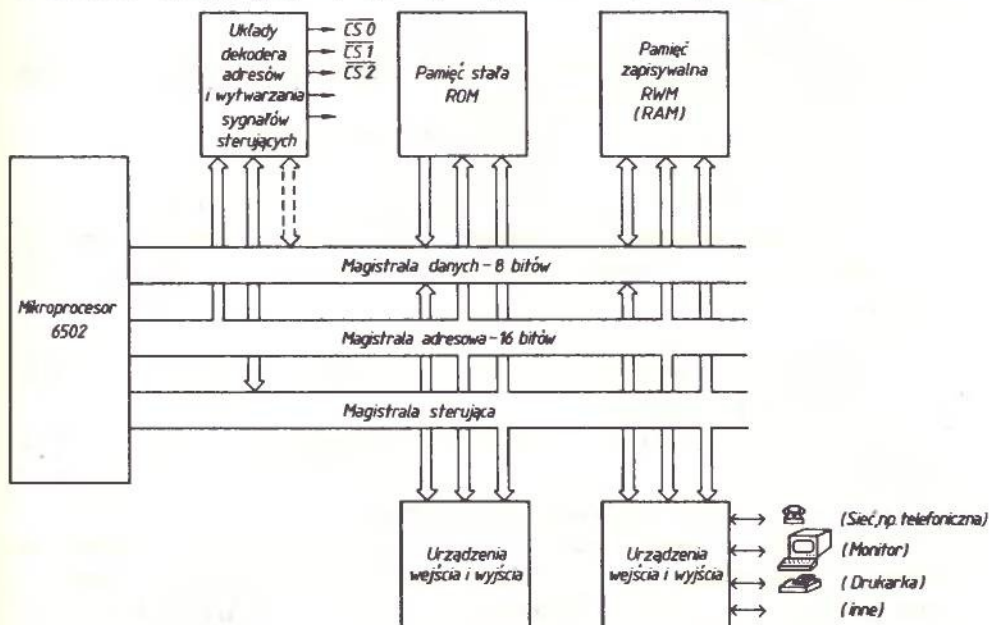
W drugiej części książki zostanie omówiona budowa typowych mikrokomputerów, których sercem jest procesor 6502. Część ta jest trudniejsza od poprzedniej i adresowana głównie do tych, dla których elektornika nie jest „czarną magią”. Czytelnikowi, który chce się tylko zorientować w problemach sprzętowych bez wnikania w szczegóły techniczne, zalecamy opuszczenie trudniejszych fragmentów opisujących szczegółowo scalone układy wejścia-wyjścia.

5

Organizacja mikrokomputera

Typowy mikrokomputer, bez względu na to, czy jest zbudowany z wykorzystaniem procesora 6502, czy innego, składa się z kilku podstawowych bloków. Są nimi (por. rys. 5.1):

Mikroprocesor, który jest sercem mikrokomputera (a raczej mózgiem), wykonuje wszystkie operacje, a w tym operacje odczytu i zapisu pamięci i urządzeń peryferyjnych, operacje logiczne i arytmetyczne.



Rys. 5.1. Schemat blokowy mikrokomputera

Pamięć stała, tylko do odczytu (ang. *Read Only Memory*, w skrócie ROM — zapamiętaj to, Czytelniku, bo w wielu miejscach spotkasz ten skrót). Pamięć ta zawiera w sobie „wiedzę” mikrokomputera, a mówiąc precyzyjniej — programy i stałe dane (np. że $\pi = 3,1415\dots$).

Pamięć zapisywalna. W języku angielskim są spotykane dwie nazwy: *Random Acces Memory* — w skrócie RAM — nazwa bardzo rozpowszechniona, lecz niepoprawna, bo oznacza pamięć o dostępie swobodnym (a więc nie sekwencyjnym), a taką jest również pamięć ROM. Drugą nazwą, poprawniejszą, jest *Read-Write Memory*, w skrócie RWM, co dokładnie oznacza pamięć odczytywalno-zapisywalną i tę nazwę (skrót RWM) będziemy spotykać na kartkach tej książki. Pamięć tego typu służy do przechowywania aktualnie przetwarzanych informacji, zmiennych, wyników częściowych, a także programów ładowanych z pamięci masowych (np. z dysków elastycznych).

Urządzenia wejścia i wyjścia, lub inaczej urządzenia peryferyjne. Są to „oczy” i „uszy” mikrokomputera, a nawet „ręce” i „nogi”. Za pomocą tych urządzeń mikrokomputer komunikuje się ze światem zewnętrznym, otrzymuje polecenia (np. z klawiatury), powiadamia o wyniku ich wykonania (np. na ekranie monitora) i steruje innymi urządzeniami.

W zależności od zastosowania mikrokomputera, a także od pomysłowości konstruktorów, urządzenia wejścia i wyjścia mogą się dość znacznie różnić między sobą; można je jednak, ze względu na wykonywane funkcje, podzielić na kilka grup:

- a) urządzenia do współpracy z człowiekiem:
 - klawiatura,
 - ekran alfanumeryczny lub graficzny,
 - manipulatory, np. myszka, kula (ang. *trackball*), drążek sterowniczy (ang. *joystick*), pióro świetlne itp.,
 - wyświetlacze cyfrowe (7-segmentowe) i alfanumeryczne,
 - syntezatory i analizatory dźwięku, a w tym mowy;
- b) urządzenia do współpracy z innymi komputerami i peryferiami:
 - sprzęgi (ang. *interface*) czytników, perforatorów, drukarek, pamięci masowych (np. dysków elastycznych i twardych, taśmy magnetycznej itp.),
 - sprzęgi szeregowo i modemy telefoniczne,
 - sprzęgi sieci komputerowych, pomiarowych itp. (np. IEC 625, ETHERNET i inne);
- c) urządzenia specjalizowane do sterowania i kontroli np. silników ramienia robota itp.

Oczywiście nie każdy mikrokomputer ma wszystkie te urządzenia peryferyjne. Na przykład mikrokomputery specjalizowane do celów pomiaru i sterowania prędkości samochodu są wyposażone tylko w urządzenia umożliwiające pomiar prędkości, wyświetlanie wartości prędkości na wskaźniku i kontrolę położenia pedałów przyspieszenia i hamulca oraz kilka klawiszy wybierających

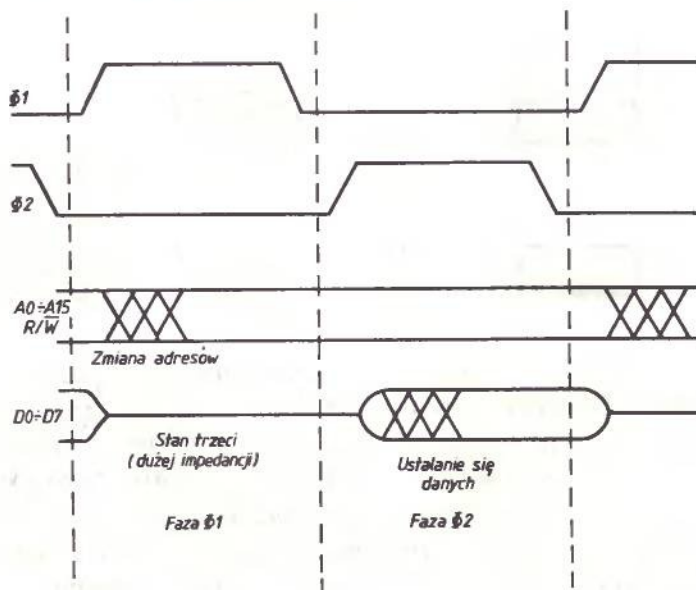
funkcje mikrokomputera (np. funkcję utrzymywania stałej prędkości jazdy).

Typowe są też podstawowe działania mikroprocesora. Są nimi:

- pobieranie danych wejściowych z urządzeń wejścia-wyjścia (np. w kasie sklepowej – ciągu liczb do zsumowania),
- przetwarzanie danych (w przykładzie z kasą – sumowanie liczb),
- wyprowadzanie wyników działania programu na zewnątrz (drukowanie sumy na paragonie).

Przechodząc do szczegółów związanych z budową mikrokomputerów wyposażonych w procesory rodziny 65XX (w tym 6502) przypomnijmy w skrócie, główne informacje związane z omawianym procesorem.

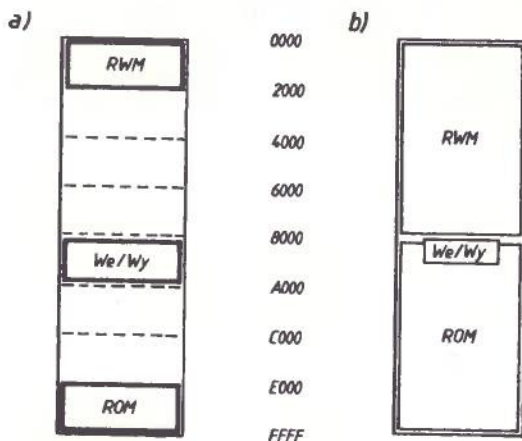
1. 6502 jest procesorem 8-bitowym z 16-bitową magistralą adresową. Konstruktor ma do dyspozycji przestrzeń adresową o rozmiarach 64 KB (dokładnie 65536 adresów), w której musi zmieścić wszystkie dołączone bloki (pamięci RWM, ROM, urządzenia wejścia-wyjścia i inne).



Rys. 5.2. Cykl maszynowy (zegarowy) procesora 65XX

2. Każdy cykl zegarowy (rys. 5.2) jest cyklem maszynowym tj. cyklem, w którym następuje dostęp do pamięci (odczyt lub zapis).
3. Procesor traktuje wszystkie swoje zasoby jak pamięć: nie ma sygnału odróżniającego rzeczywistą pamięć od urządzeń peryferyjnych.
4. Pierwsza część cyklu zegarowego (faza $\Phi 1$) jest przeznaczona na ustalenie się adresów oraz sygnału odczyt/zapis (R/W). Pozostają one nie zmienione przez całą fazę $\Phi 2$.
5. Druga część cyklu zegarowego (faza $\Phi 2$) jest przeznaczona na ustalenie się danych. Są one odczytywane lub zapisywane na opadającym zboczku sygnału $\Phi 2$.

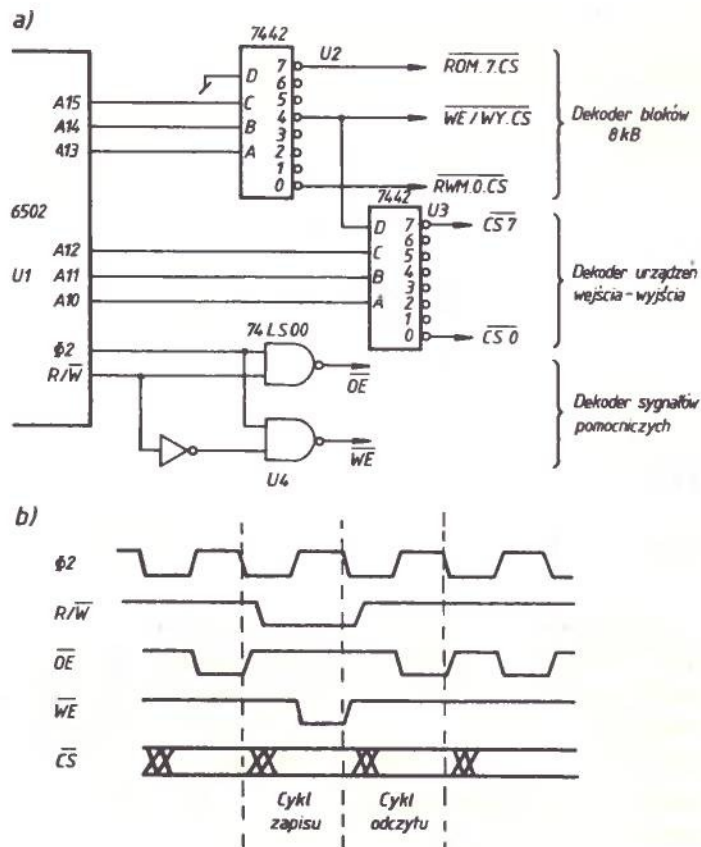
6. Aby nie dopuścić do konfliktu na liniach danych w czasie fazy $\Phi 1$, linie danych nie powinny być sterowane — wszystkie bufony danych (procesora, pamięci, urządzeń peryferyjnych) są wtedy w stanie trzecim — dużej impedancji.
7. Ze względu na umieszczenie wektorów przerwań (*NMI*, *IRQ*, *RESET*) w obszarze przestrzeni adresowej o numerach komórek $\$FFFA \div \$FFFF$, zaś stosu — w obszarze o numerach $\$0100 \div \$01FF$, konieczne jest umieszczenie w odpowiednich obszarach pamięci ROM i pamięci RWM. Pozostałe wolne miejsca przeznacza się zwykle na obszar urządzeń wejścia-wyjścia oraz dodatkowe bloki pamięci.



Rys. 5.3. Przykłady map pamięci mikrokomputera: a) bloki 8 KB (pamięci typu 2764, 6264); b) bloki 32 KB (pamięci typu 27256, 62256) z blokowaniem pamięci stałej (ROM) w obszarze wejścia-wyjścia

O tym, jak wygląda mapa pamięci mikrokomputera (por. rys. 5.3), decyduje blok dekodera adresów, którego zadaniem jest dostarczanie sygnałów uaktywniających poszczególne bloki. Zwykle do dekodera dodaje się układ służący do wytwarzania potrzebnych sygnałów sterujących, a w tym sygnałów $\overline{OE}$ (ang. *Output Enable* — zezwolenie wysłania danych na magistralę) i $\overline{WE}$ (ang. *Write Enable* — zezwolenie na zapis danych z magistrali), używanych do sterowania buforami 3-stanowymi a także funkcjami odczytu i zapisu pamięci RWM i ROM.

Przykład bloku sterowania, składający się dekodera adresów i układu wytwarzania sygnałów $\overline{OE}$ i $\overline{WE}$, jest przedstawiony na rys. 5.4. Układ U2 (7442) jest dekodерem ośmiu bloków o wielkości 8 KB każdy. Sygnał $RWM\ 0 \cdot CS$ jest sygnałem uaktywniającym pamięć zapisywalną, będzie więc ona znajdować się w obszarze przestrzeni adresowej $\$0000 \div \$1FFF$. Sygnał $ROM\ 7 \cdot CS$ jest sygnałem uaktywniającym pamięć stałą znajdującą się w obszarze $\$E000 \div \$FFFF$; sygnał ten, podany na układ U3, powoduje wytworzenie sygnałów uaktywniających poszczególne urządzenia ($CS0 \div CS7$), tak więc urządzenie o numerze 0 będzie zajmowało obszar $\$8000 \div \$83FF$, o numerze 1 — odpowiednio $\$8400 \div \$87FF$ i tak dalej — do



Rys. 5.4. Układ wytwarzania sygnałów sterujących: a) schemat ideowy; b) przebiegi czasowe

urządzenia nr 7 zajmującego obszar $\$9C00 \div \$9FFF$. Sygnały $\overline{OE}$ i $\overline{WE}$ są wytwarzane przez bramki układu U4 (7400). Sygnał $\overline{OE}$ (zezwolenie odczytu) jest zanegowanym iloczynem sygnałów $\Phi 2$ i $R/\overline{W}$, zaś sygnał $\overline{WE}$ (zezwolenie zapisu) jest zanegowanym iloczynem sygnałów $\Phi 2$ i $R/\overline{W}$.

6

Elementy składowe mikrokomputera

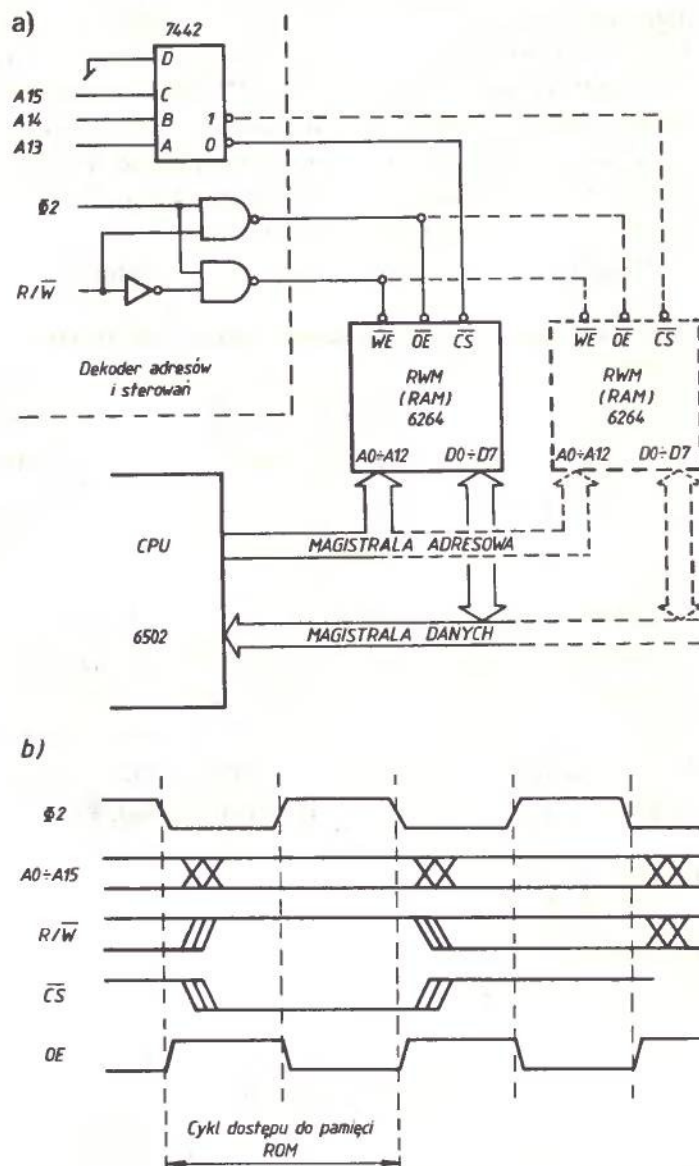
Z rozdziału poprzedniego dowiedziałeś się, Czytelniku, co trzeba dołączyć do mikroprocesora, by powstał mikrokomputer, w tym rozdziale zaś opowiemy dokładniej – jak to zrobić. W rozdz. 2 podaliśmy, jak wytworzyć sygnał zegarowy procesora, w rozdz. 5 – jak podzielić przestrzeń adresową i wytworzyć podstawowe sygnały sterujące – sygnały selekcji obszaru pamięci (z dekodera adresów) oraz sygnały pomocnicze *OE* i *WE*. Teraz zaczniemy dołączać pozostałe elementy mikrokomputera. Zacznijmy od najprostszego, tj. pamięci.

6.1. Dołączanie pamięci stałej (ROM)

W niewielu dziedzinach techniki można zaobserwować tak daleko posunięte ujednolicenie wyrobów, jak to się obserwuje w odniesieniu do pamięci ROM (EPROM). Dołączenie pamięci do procesora omówimy na przykładzie pamięci typu 2764 o organizacji $8\text{ K} \times 8$ bitów, jednak dołączanie innych typów jest praktycznie identyczne, gdyż pamięci te różnią się tylko wielkością (liczbą bajtów) i czasem dostępu, nie zaś sposobem sterowania. Przytoczone poniżej zasady są praktycznie biorąc wystarczające do projektowania układów mikrokomputera. Nie można jednak zapomnieć o sprawdzeniu, czy w zaprojektowanym układzie są spełnione wszystkie wymagania czasowe. Wymagania te są częściowo omówione w p. 4.2, jednak przy samodzielnym projektowaniu gorąco polecamy posługiwanie się katalogami firmowymi, gdyż elementy różnych producentów mogą się różnić niektórymi parametrami czasowymi.

Typowa pamięć stała jest zasilana napięciem $+5\text{ V}$, ma 8 wyjść danych, od 10 do 16. wejść adresowych (w zależności od wielkości pamięci – pamięć 2764

ma ich 13) i dwa wejścia sterujące odczytem: wejście uaktywniające układ $\overline{CS}$ (ang. *Chip Select*) i wejście zezwolenia odczytu $\overline{OE}$. Dodatkowo pamięci programowalne (PROM, EPROM) mają wejścia sterujące programowaniem. Wejść tych nie używa się w gotowym mikrokomputerze, należy tylko pamiętać o dołączeniu ich do napięcia zalecanego przez producenta (w pamięci 2764 wejście V_{pp} dołącza się do napięcia +5 V).



Rys. 6.1. Sposób dołączania pamięci stałej (ROM): a) schemat; b) przebiegi czasowe

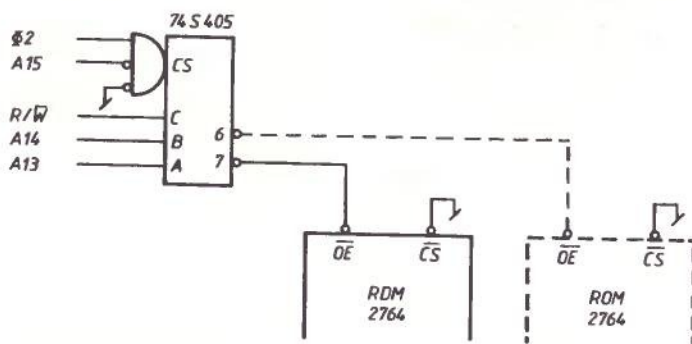
Uważny Czytelnik zapyta w tym miejscu, po co aż dwa wejścia sterujące odczytem, gdy tak naprawdę wystarczy jedno. Producenci zastosowali takie rozwiązanie z kilku powodów. Po pierwsze, zastosowanie kilku wejść sterujących może uprościć i przyspieszyć układ sterujący. Po drugie, funkcje obu tych wejść są różne, różne też są czasy uzyskania poprawnych danych wyjściowych, liczone od momentu podania danego sygnału. Wejście $\overline{CS}$ powoduje uaktywnienie całego układu. Jeśli jest na nim sygnał jedynki logicznej, to zwykle układ jest nie tylko nieaktywny, ale wręcz odłączony od zasilania (wewnętrznie). Czas dostępu do pamięci liczony od tego sygnału jest najdłuższy (w naszym przykładzie 250 ns) i praktycznie taki sam, jak czas dostępu dla linii adresowych. Wejście $\overline{OE}$ steruje tylko 3-stanowymi buforami wyjściowymi pamięci, umożliwiając pojawienie się odczytywanych danych na liniach $D0 \div D7$, dlatego też czas dostępu liczony od tego sygnału jest znacznie mniejszy (w naszym przykładzie 100 ns).

W tabl. 6.1 są przedstawione funkcje sygnałów $\overline{OE}$ i $\overline{CS}$.

Tablica 6.1. Znaczenie wejść sterujących pamięci stałej (ROM); X – wartość dowolna

$\overline{CS}$	$\overline{OE}$	Wyjścia $D0 \div D7$	Pobór mocy
1	X	Z – duża impedancja	Mały
0	1	Z – duża impedancja	Normalny
0	0	D_o – dane z pamięci	

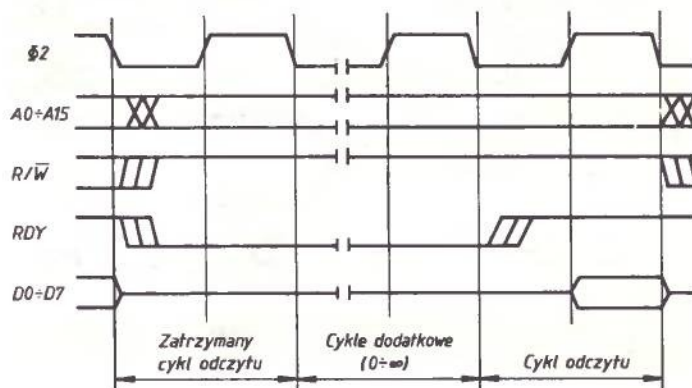
Dołączając pamięć stałą do procesora 6502 (rys. 6.1), najczęściej na wejście $\overline{CS}$ podaje się sygnał z dekodera adresów, zaś na wejście $\overline{OE}$ sygnał będący zanegowanym iloczynem logicznym $\Phi 2$ i $R/\overline{W}$ (sygnał $\overline{OE}$). Jeśli chcemy dołączyć więcej układów pamięci, to wszystkie sygnały adresowe i sygnały danych łączymy równolegle, na wejście $\overline{OE}$ podajemy wspólny sygnał, wejściami $\overline{CS}$ zaś wybieramy odpowiedni układ. Rozwiązanie takie stosuje się



Rys. 6.2. Inny sposób dołączania pamięci stałej (ROM) – sterowanie tylko sygnałem $\overline{OE}$

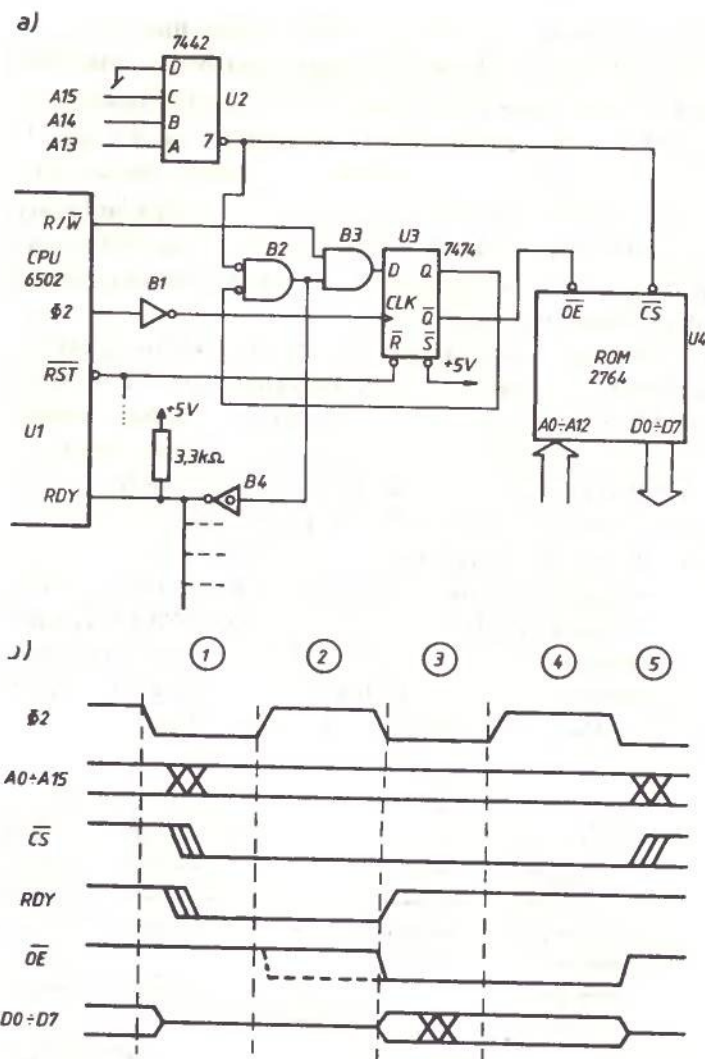
jednak coraz rzadziej, gdyż łatwiej obecnie jest wziąć odpowiedniej wielkości pamięć (nawet $64\text{ K} \times 8\text{ b}$ — typu 27512). Można również sterować układ tylko jednym z sygnałów $\overline{OE}$ i $\overline{CS}$, na drugie z wejść podając na stałe 0. Rozwiązanie takie stosuje się czasem, gdy chce się do minimum zredukować czas dostępu do danych, oszczędzając na czasie dekodowania adresów. Steruje się wtedy tylko linią $\overline{OE}$, podając na nią iloczyn sygnałów $R/\overline{W}$, $\Phi 2$ i odpowiedniej kombinacji adresów, np. tak jak przedstawia to rys. 6.2. Układ taki ma tę dodatkową zaletę, że nie występują duże zmiany prądu zasilania przy uaktywnianiu układu (ale za to średni prąd zasilania jest większy).

Przy stosowaniu szybkich wersji procesora przedstawione powyżej rozwiązania mogą okazać się niewystarczające. Na przykład procesor wersji C, pracujący z zegarem 4 MHz, wymaga od współpracujących z nim pamięci czasu dostępu 110 ns, co zwykle jest warunkiem zbyt wygórowanym w stosunku do pamięci ROM, dla których typowe czasy dostępu mają wartości w granicach 150 — 450 ns. Dołączenie wolniejszych pamięci stałych staje się możliwe, jeśli stosuje się zasadę powtórzenia cyklu odczytu, a więc przedłużenia go z jednego taktu zegara do dwóch, a nawet więcej. Po podaniu na linię $\overline{RDY}$ stanu 0 procesor powtarza cykl odczytu (dla wersji CMOS każdy cykl) tak długo, dopóki nie stwierdzi w fazie $\Phi 2$ stanu 1 na tej linii (rys. 6.3). Najczęściej wystarcza przedłużenie cyklu odczytu o jeden takt zegara, co przy częstotliwości 4 MHz powiększa czas dostępu z 110 na 360 ns.



Rys. 6.3. Przedłużenie cyklu odczytu sygnałem $\overline{RDY}$

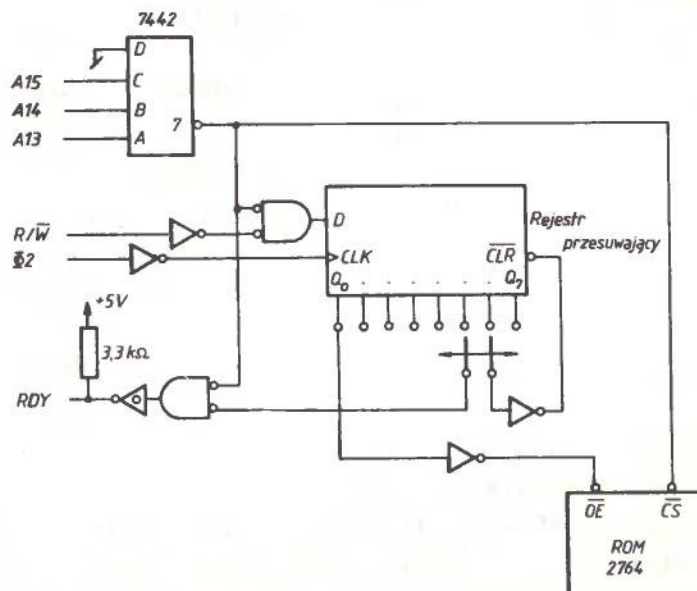
Przykład rozwiązania układu według omawianej zasady przedstawia rys. 6.4. W przypadku odczytu danych z wolnej pamięci ROM (U_4) w fazie $\Phi 1(1)$, bezpośrednio po zdekodowaniu adresu układu U_4 przez dekodery U_2 , na wejście $\overline{RDY}$ zostanie podany, poprzez bramki B_2 i B_4 , stan 0. Bramka B_4 jest bramką z otwartym kolektorem, co umożliwia sterowanie linią $\overline{RDY}$ przez większą liczbę układów mikrokomputera. W fazie $\Phi 2(2)$ procesor stwierdzi stan zero na linii $\overline{RDY}$ i powtórzy cykl odczytu. Jednocześnie na opadającym



Rys. 6.4. Układ wydłużający czas odczytu pamięci ROM o jeden cykl zegarowy: a) schemat ideowy; b) przebiegi czasowe

zbozcu sygnału $\Phi 2(2)$ do przerzutnika typu D (U3) zostanie wpisana jedynka (wytworzona z sygnału $R/\overline{W}$ i $\overline{CS}$ przez bramki B2 i B3). W fazach $\Phi 2(3)$ i $\Phi 2(4)$, jedynka, wpisana do przerzutnika, spowoduje ustawienie linii RDY ponownie w stan 1 (poprzez zablokowanie bramki B2), a więc odczytanie danych w fazie $\Phi 2(4)$. Linia $\overline{OE}$ jest sterowana z wyjścia Q przerzutnika, a więc stan zero na wejściu $\overline{OE}$ układu U4 trwać będzie przez cały cykl zegarowy ($\Phi 1(3) + \Phi 2(4)$). Opadające zbocze sygnału $\Phi 2(4)$ spowoduje wpisanie do przerzutnika D zera, a więc powrót do stanu spoczynkowego. Należy zwrócić uwagę, że zachodzi tu konieczność nie tylko podania odpowiedniego sygnału na linię $READY$

procesora, lecz także zmiany sposobu generacji sygnału $\overline{OE}$ dochodzącego do pamięci ROM, tak by długość tego sygnału spełniała wszystkie wymagania czasowe pamięci.



Rys. 6.5. Układ do współpracy z wolnymi pamięciami stałymi (ROM), gdy wymagany czas dostępu wynosi od 3 do 7 cykli zegara

W przypadkach, gdy dodanie jednego taktu zegarowego jest niewystarczające, układ współpracy staje się bardziej skomplikowany. W przypadku konieczności dodania kilku do kilkunastu cykli najprostszym rozwiązaniem jest zastosowanie rejestru przesuwającego (rys. 6.5), powyżej — liczników i dekodów.

6.2. Dołączanie pamięci zapisywalnej (RWM)

Pamięć zapisywalna RWM umożliwia wykonanie dwóch operacji: zapisu danych lub odczytu danych, do lub z dowolnej komórki pamięci (o dowolnym adresie).

Istnieją dwa zasadnicze rodzaje pamięci zapisywalnej: statyczna i dynamiczna. Różnią się one znacznie budową podstawowej komórki pamięci, co pociąga za sobą różnicę w sygnałach sterujących, a zatem i w sposobie dołączania.

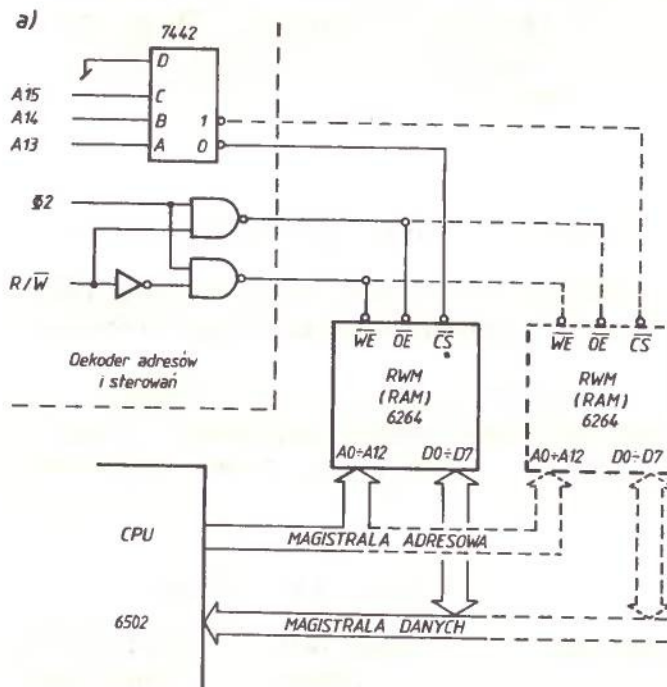
6.2.1. Dołączanie pamięci statycznej

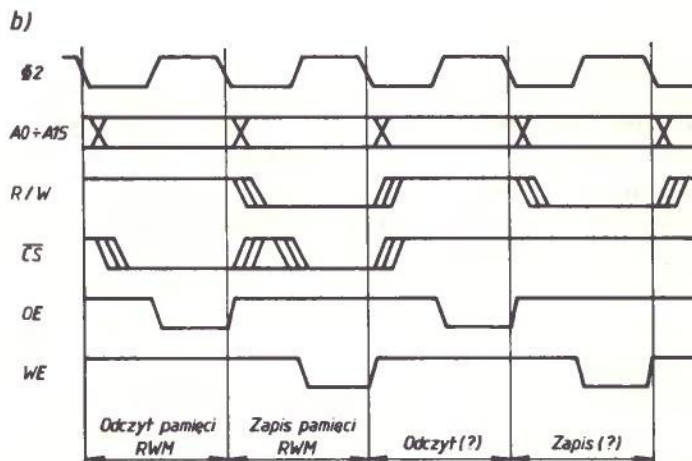
Dołączanie pamięci zapisywalnej statycznej (ang. SRWM, SRAM) niewiele różni się od dołączania pamięci stałej. Obecnie produkowane pamięci poza

Tablica 6.2. Funkcje sygnałów sterujących pamięci RWM; X – wartość dowolna

$\overline{CS}$	$\overline{OE}$	$\overline{WE}$	Linie $D0 \div D7$	Operacja
1	X	X	Z – duża impedancja	Układ nieaktywny, zwykle zmniejszony pobór mocy
0	1	1	Z – duża impedancja	Układ aktywny, przygotowanie do zapisu lub odczytu
0	0	1	D_o – dane odczytywane	Odczyt
0	1	0	D_i – dane zapisywane	Zapis
0	0	0	D_i – dane zapisywane	Kombinacja różnie traktowana przez różnych wytwórców; dla niektórych pamięci może to być operacja zapisu, w innych może być zabroniona

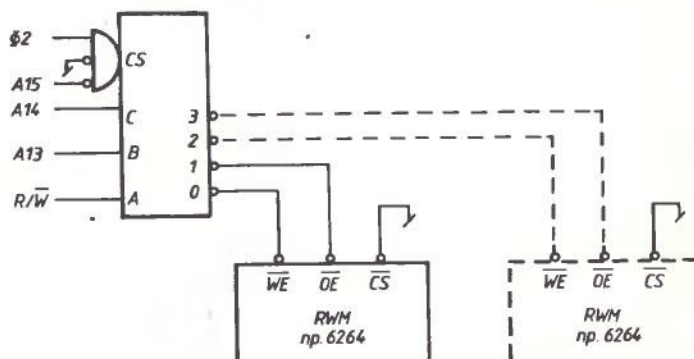
wejściem adresowym i dwukierunkowymi liniami danych mają najczęściej trzy wejścia sterujące: $\overline{CS}$, $\overline{OE}$ i $\overline{WE}$. Znaczenie wejść $\overline{CS}$, $\overline{OE}$ jest identyczne jak dla pamięci ROM, wejście $\overline{WE}$ steruje operacją zapisu (patrz tabl. 6.2). Najczęściej spotykany sposób dołączania jest przedstawiony na rys. 6.6. Do wejścia $\overline{WE}$ (zezwolenie zapisu) jest doprowadzony zanegowany iloczyn sygnałów $\Phi 2$ i $R/\overline{W}$, do wejścia $\overline{OE}$ (zezwolenie odczytu) – zanegowany iloczyn sygnałów





Rys. 6.6. Dołączanie pamięci zapisywalnej (RWM): a) schemat; b) przebiegi czasowe

$\Phi 2$ i $R/\overline{W}$, zaś do wejścia $\overline{CS}$ – sygnał z dekodera adresów. W przypadku, gdy zależy nam na zmniejszeniu czasu dostępu, można, podobnie jak w przypadku pamięci ROM, zrezygnować z użycia sygnału $\overline{CS}$, a przez to wyeliminować czas opóźnienia wprowadzony przez dekodery adresowy. Przykład takiego rozwiązania jest przedstawiony na rys. 6.7. Ponieważ procesor nie ma możliwości powtórzenia cyklu zapisu, przy dołączaniu pamięci RWM nie stosuje się przedłużania cyklu dostępu. Nie miałoby to również większego sensu. (Pomimo istnienia takiej możliwości w wersji CMOS, praktycznie nie jest ona do tego wykorzystywana).



Rys. 6.7. Dołączanie pamięci zapisywalnej (RWM) bez użycia sygnału $\overline{CS}$

Pamięć RWM jest jak zwykle najszybszym elementem mikrokomputera, jeśli jednak szybszy byłby mikroprocesor, to w takim przypadku byłby powtarzany każdy cykl dostępu do pamięci, więc prostszym rozwiązaniem jest zmniejszenie częstotliwości zegara.

6.2.2. Dołączanie pamięci dynamicznej

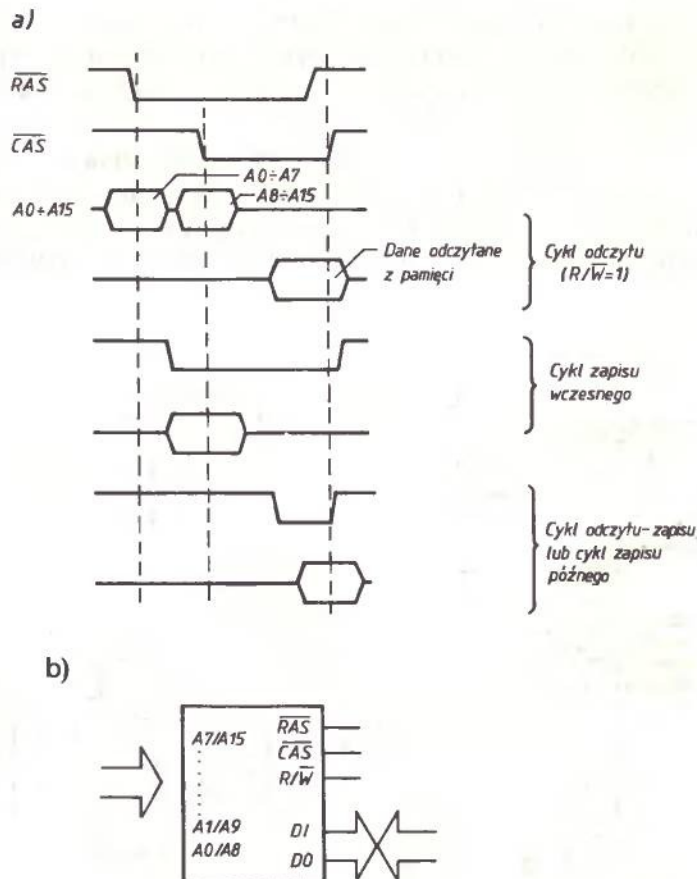
W obecnym stanie techniki coraz mniej celowe staje się stosowanie pamięci dynamicznych w układach z procesorami 8-bitowymi, a w tym z procesorem 6502. Całą 64-kilobajtową przestrzeń adresową procesora można obecnie zapisać za pomocą dwóch układów scalonych – jednego układu pamięci ROM typu 27256 (32 KB) oraz jednego układu pamięci statycznej RAM typu 62256 (32 KB). Pamięci dynamiczne (ang. DRWM, DRAM) należy stosować tylko tam, gdzie wielkość pamięci przekracza znacznie 64 KB i zysk związany z mniejszym kosztem pamięci dynamicznej jest większy od kosztów dość skomplikowanego układu sterowania pamięci. W przypadkach konieczności stosowania pamięci o wielkości powyżej 64 KB należy jednak raczej zastosować jeden z dostępnych procesorów 16-bitowych (np. Motorola 68000, Intel 80286 lub inny), niż stosować układy stronicowania pamięci. Ze względu na dostępność środków uruchomieniowych (sprzętu uruchomieniowego i oprogramowania) stosowanie pamięci dynamicznych do procesora 6502 nie zostało jednak całkowicie zarzucone, dlatego też pokrótce zostaną omówione sposoby dołączania tych pamięci.

Przebiegi czasowe i wymagania pamięci dynamicznych. Podstawowa komórka pamięci dynamicznej dość znacznie różni się od komórki pamięci statycznej. Informacja w niej nie jest przechowywana jako stan układu bistabilnego, lecz jako ładunek kondensatora typu MOS. Dlatego też są wymagane zupełnie inne sygnały sterujące pamięcią oraz konieczne jest odświeżanie pamięci, podczas którego uzupełniany zostaje ładunek kondensatora komórki pamięci.

Opis sposobu dołączania pamięci dynamicznych, ze względu na wielkość przestrzeni adresowej procesora, opierać się będzie na pamięci typu 4164 (64 Kb). Pamięć dynamiczna jest sterowana następującymi sygnałami: $\overline{RAS}$ (ang. *Row Address Strobe*) – sygnał strobuujący adresu wiersza, $\overline{CAS}$ (ang. *Column Address Strobe*) – sygnał strobuujący adresu kolumny, $R/\overline{W}$ (ang. *Read/Write*) – sygnał zapisu-odczytu. Wejścia adresowe w pamięciach dynamicznych są wejściami multipleksowanymi – na ujemnym zboczach sygnału $\overline{RAS}$ podaje się adres wiersza ($A0 \div A7$), zaś na ujemnym zboczach $\overline{CAS}$ – adres kolumny ($A8 \div A15$). Przykłady cykli zapisu i odczytu są pokazane na rys. 6.8. Odświeżanie pamięci polega bądź na odczycie ze wszystkich adresów wierszy co okres odświeżania (2 ms), bądź na stosowaniu specjalnych (osobnych) cykli odświeżania (ang. *RAS-only*, *hidden refresh*).

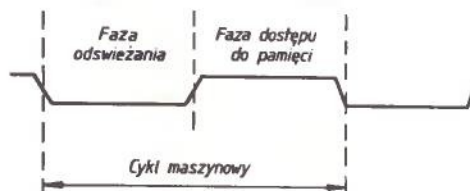
Dołączając pamięci dynamiczne do procesora 6502, należy rozwiązać dwa problemy: problem odświeżania i problem wygenerowania sygnałów sterujących oraz ich synchronizacji z procesorem. Spośród wielu możliwych sposobów odświeżania przedstawione zostaną trzy najbardziej rozpowszechnione w zastosowaniach z procesorem 6502.

Jeśli pamięć ma czas dostępu znacznie krótszy od wymaganego przez



Rys. 6.8. Pamięć dynamiczna: a) przebiegi czasowe; b) rozkład wyprowadzeń

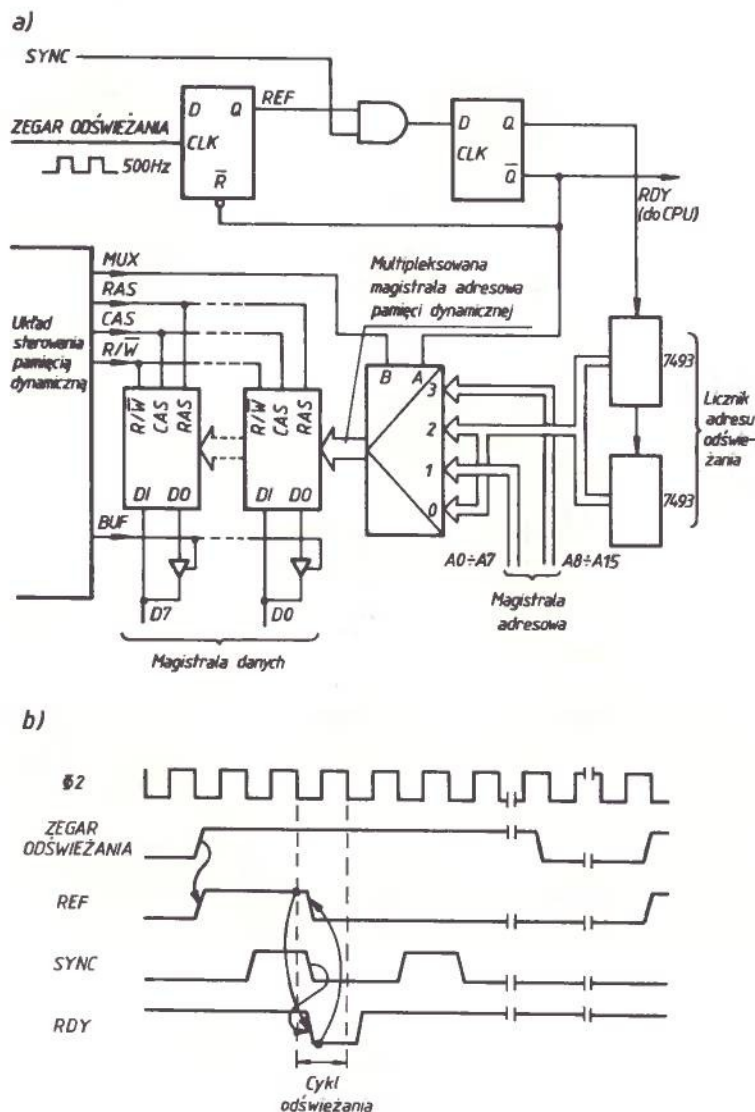
procesor, można zastosować odświeżanie w przeciwfazie (rys. 6.9). W fazie $\Phi 2$ procesora ($\Phi 2 = 1$) następuje dostęp do pamięci (operacja odczytu lub zapisu), natomiast w fazie $\Phi 1$ procesora ($\Phi 2 = 0$) następuje odczyt pamięci przez układ odświeżający, którego zadaniem jest wygenerowanie w kolejnych cyklach kolejnych adresów pamięci. Dość często układem odświeżającym jest tzw. sterownik VIDEO (układ wytwarzający sygnał wizyjny na ekranie monitora), który w fazie $\Phi 1$ odczytuje dane wizyjne. Ten typ odświeżania stosuje się głównie przy częstotliwościach zegara około 1 MHz, w mikrokomputerach z



Rys. 6.9. Odświeżanie w przeciwfazie

wbudowanym układem wyświetlania danych na ekranie. Zapewnia on bez-konfliktowy dostęp do pamięci zarówno procesora, jak i układu wyświetlania (zwykle zrealizowanego z wykorzystaniem układu 6545 lub 6847, który jest opisany w dalszej części książki).

W przypadku zastosowania tzw. **odświeżania z kradnięciem cyklu** (ang. *cycle stealing*) układ odświeżający zatrzymuje procesor na jeden cykl maszynowy (sygnałem *READY*) i w tym czasie dokonuje odczytu pamięci. Dla typowych pamięci dynamicznych, których czas między odświeżeniami wynosi



Rys. 6.10. Układ odświeżania pracujący w trybie „kradnięcia” cyklu:
a) schemat ideowy; b) przebiegi czasowe

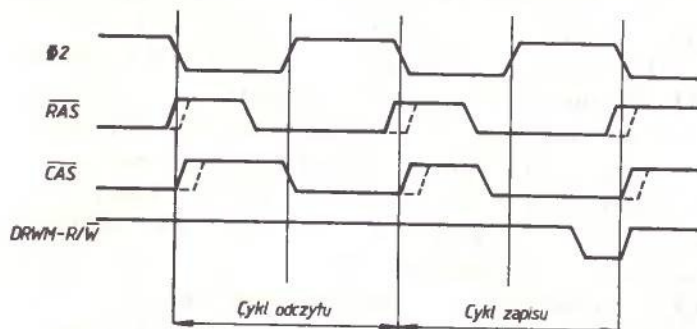
2 ms, a liczba odświeżanych adresów rzędów wynosi 128 (7 bitów), czas między poszczególnymi cyklami odświeżania (rozłożonymi równomiernie w czasie) wynosi około 16 μ s. Ze względu na fakt, że procesor 6502 zatrzymuje się tylko w cyklu odczytu, zwykle „kradnie się” pierwszy cykl po cyklu pobrania kodu operacyjnego, korzystając z faktu, że jest to zawsze cykl odczytu, w dodatku z obszaru pamięci programu. Przykład układu odświeżania z „kradnięciem cyklu” jest przedstawiony na rys. 6.10. Dodatkowo zbocze zegara odświeżania (pochodzącego np. z podziału sygnału $\Phi 2$ przez 16 — dla wersji 1 MHz lub przez 32 w wersji 2 MHz) powoduje pojawienie się sygnału *REF* (żądanie odświeżania). Sygnał *REF*, wymnożony logicznie z sygnałem procesora *SYNC* (przez bramkę 1) i podawany na przerzutnik synchronizowany opadającym zboczem sygnału $\Phi 2$, powoduje ustawienie sygnału *READY* na zero na czas jednego cyklu zegara następującego po impulsie *SYNC*, a co za tym idzie — zatrzymanie procesora na jeden cykl zegara. W tym cyklu następuje przełączenie multipleksera (*MUX*) z magistrali adresowej na wyjścia licznika adresu odświeżania (inkrementowanego dodatnim zboczem sygnału *READY*) i odświeżanie jednego wiersza pamięci dynamicznej. Opisany układ powoduje zmniejszenie efektywnej szybkości procesora o około 6,3% w przypadku częstotliwości zegara 1 MHz i 1,6% dla 4 MHz. Omówiony sposób odświeżania stosuje się w układach o dużych szybkościach procesora (w wersjach 2, 3 i 4 MHz) w przypadku, gdy jest wymagany krótki czas reakcji procesora na zdarzenia zewnętrzne (przerwania).

Jeżeli nie jest wymagana od mikrokomputera szybka reakcja na przerwania, to można zastosować **odświeżanie programowe**. Co 2 ms przerywa się normalną pracę mikroprocesora (najczęściej podając przebieg prostokątny o częstotliwości 500 Hz na wejście przerwania niemaskowalnego *NMI*) i wykonuje program odświeżania. W programie tym dokonuje się odczytu 128 kolejnych adresów pamięci dynamicznej. Postać programu odświeżającego może być różna, jednak najprostszym rozwiązaniem jest wielokrotne wykorzystanie rozkazu *JMP*, który trwa 3 cykle maszynowe i powoduje odczyt z kolejnych 3 adresów pamięci.

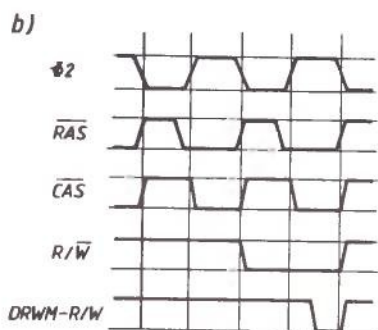
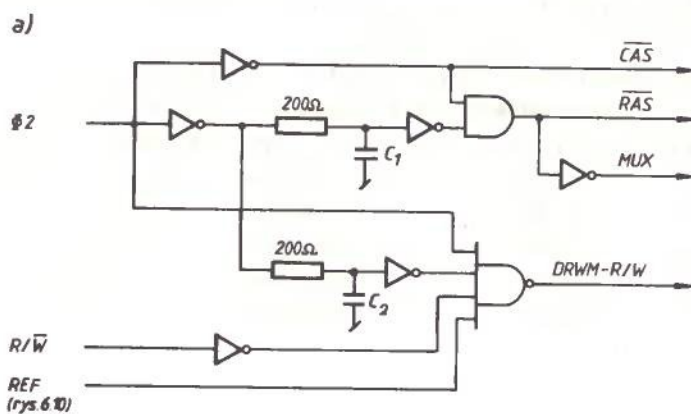
Typowy program odświeżania pamięci ma postać:

Adres	Rozkaz
\$FCFC	JMP \$FCFF
\$FCFF	JMP \$FD02
\$FD02	JMP \$FD05
\$FD05	JMP \$FD08
.	.
\$FD71	JMP \$FD74
\$FD74	JMP \$FD77
\$FD77	NOP
\$FD78	RTI

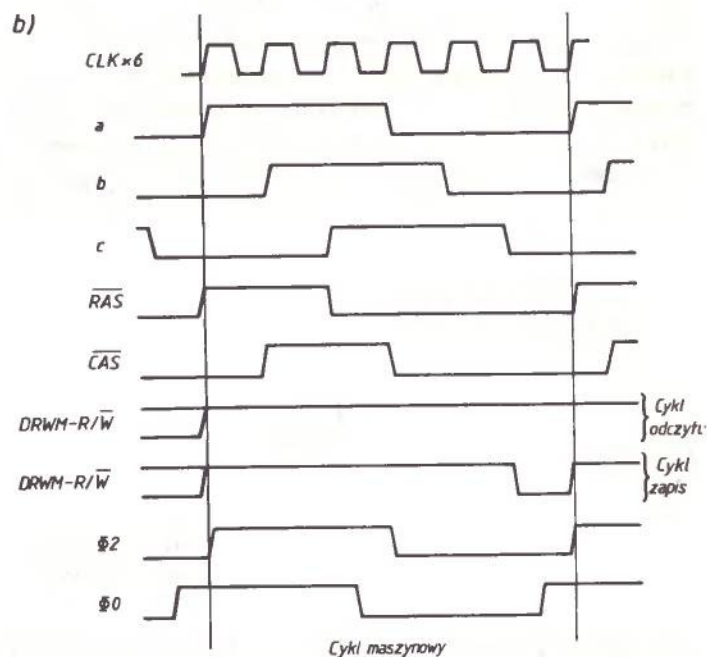
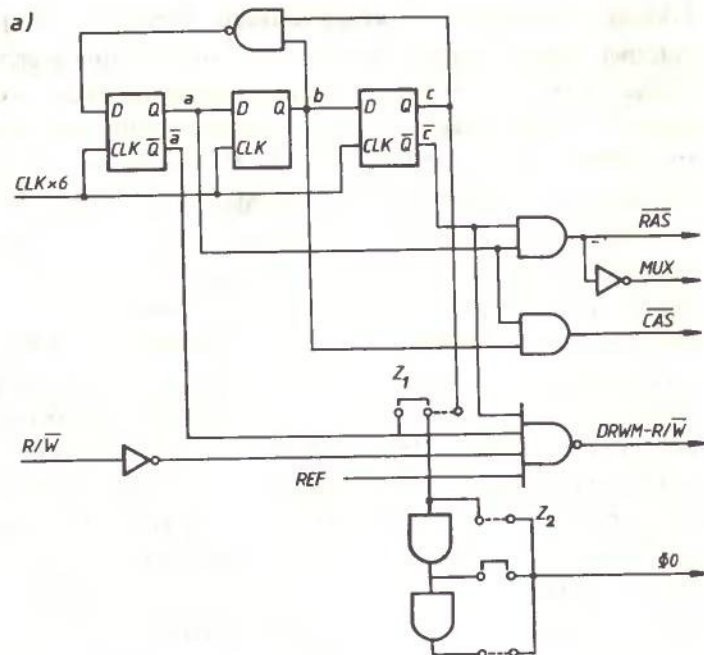
Wektor NMI (\$FFFA, \$FFFB) wskazuje na adres startu procedury odświeżania (\$FCFC). Czas wykonania procedury wynosi 134 cykle zegara, tak więc proces odświeżania programowego wymaga 6 cykli zegarowych więcej niż proces odświeżania sprzętowego, wymaga również dodatkowego programu o długości 126 bajtów, daje jednak znaczne uproszczenie układu sterowania pamięci dynamicznej oraz multiplekserów adresowych.



Rys. 6.11. Zalecane przebiegi czasowe do sterowania pamięci dynamicznej

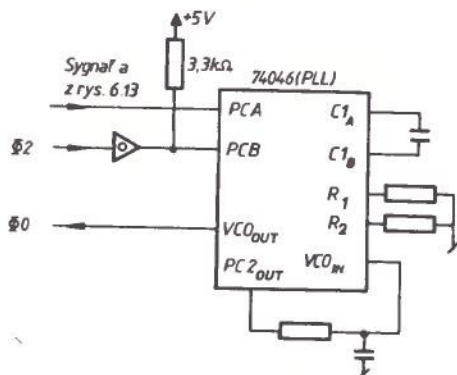


Rys. 6.12. Przykład asynchronicznego wytwarzania sygnałów $\overline{RAS}$, $\overline{CAS}$, MUX i $\overline{DRAM-R/W}$: a) schemat ideowy; b) przebiegi czasowe



Rys. 6.13. Synchroniczny układ sterowania pamięci dynamicznej: a) schemat ideowy (Z_1 i Z_2 — zwory doboru wyprzedzenia sygnału Φ_0 względem $\overline{RAS}$); b) przebiegi czasowe

Układy wytwarzania i synchronizacji sygnałów sterujących pamięci dynamicznej. Układy te mają za zadanie wygenerowanie sygnałów RAS , CAS i R/W , będących w odpowiednich zależnościach czasowych z sygnałem $\Phi 2$ procesora (rys. 6.11). Najprostszym układem generującym powyższe sygnały jest układ z obwodami opóźniającymi RC . Kondensator C_1 określa czas pomiędzy opadającym zboczem $\Phi 2$ a opadającym zboczem RAS . Sygnał CAS jest zanegowanym sygnałem $\Phi 2$. Kondensator C_2 określa czas pomiędzy narastającym zboczem $\Phi 2$ a opadającym zboczem sygnału $DRAM-R/W$ w cyklu zapisu. Sygnał REF blokuje możliwość zapisu w cyklu odświeżania (zwykle nie jest wykorzystany, gdyż odświeżanie wykonuje się zatrzymując procesor w cyklu odczytu — R/W w stanie „1”). Wadą takiego układu jest wrażliwość na zmiany czasu trwania cyklu zegarowego oraz na zmiany parametrów układu. Pewniejsze w działaniu są układy synchroniczne, które korzystają z zegara o częstotliwości od 4 do 16 razy większej od częstotliwości pracy procesora. W tych układach wszystkie zmiany sygnałów odbywają się synchronicznie z sygnałem zegara. Poza sygnałami RAS , CAS , $DRAM-R/W$ i MUX układ sterujący wytwarza sygnał $\Phi 0$, który jest zewnętrznym zegarem procesora. Przykładowy układ synchroniczny i jego przebiegi są przedstawione na rysunku 6.13. Układ ten jest sterowany z zegara o częstotliwości sześciokrotnie większej od częstotliwości $\Phi 0$. Składa się on z licznika pierścieniowego modulo 6 oraz dekodera sygnałów RAS , CAS , $DRAM-R/W$. Sygnał $\Phi 2$ powinien być zgodny w fazie z sygnałem $\bar{a}$. Ponieważ jednak istnieje przesunięcie czasowe między sygnałem $\Phi 0$ a $\Phi 2$ w procesorze, należy więc dobrać odpowiedni sygnał do sterowania procesora — za pomocą zwor pokazanych na rys. 6.13. Aby uniknąć kłopotliwego dobierania opóźnień w torze $\Phi 0$ (w celu zapewnienia zgodności czasowej) można zastosować układ



Rys. 6.14. Układ synchronizowania sygnału $\Phi 2$ z sygnałem odniesienia za pomocą układu pętli fazowej (PLL)

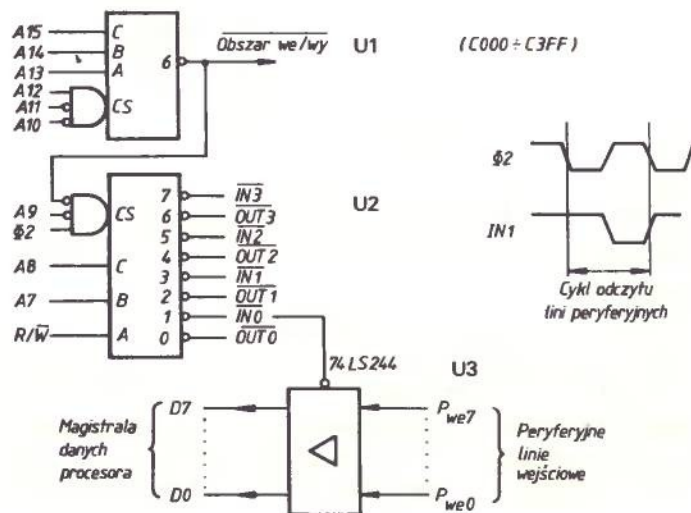
pętli fazowej, sprowadzającej automatycznie przesunięcie czasowe między sygnałem $\Phi 2$ i sygnałem odniesienia ($\bar{a}$) do zera. Przykład takiego rozwiązania jest pokazany na rys. 6.14. Podobne rozwiązanie jest zastosowane na przykład w mikrokomputerze Commodore C64.

6.3. Układy wejścia i wyjścia

Każdy mikrokomputer, poza pamięcią, musi mieć dołączone układy wejścia-wyjścia, służące do komunikacji ze światem zewnętrznym. Procesor 6502, w przeciwieństwie do 8080 i Z80, nie ma sprzętowo wydzielonego obszaru adresów urządzeń peryferyjnych, nie ma też specjalizowanych rozkazów wejścia-wyjścia. Wszystkie układy wejścia-wyjścia są traktowane przez procesor na równi z komórkami pamięci, można im więc przypisać adresy w dowolnie wybranym miejscu przestrzeni adresowej. Zwykle jednak wydziela się pewien obszar przestrzeni adresowej (np. blok wielkości 1 KB lub 8 KB), w którym umieszcza się adresy wszystkich urządzeń peryferyjnych. Czasem w celu przyspieszenia obsługi urządzeń peryferyjnych umieszcza się ich adresy (a przynajmniej ich część) na stronie zerowej. Należy jednak czynić to z daleko posuniętą ostrożnością, zwracając uwagę, by dodatkowe odczyty występujące w niektórych instrukcjach nie spowodowały odczytu rejestru urządzenia peryferyjnego, co może doprowadzić do utraty danych (w przypadku stosowania automatycznego potwierdzenia, tak jak w układach 6520 i 6522).

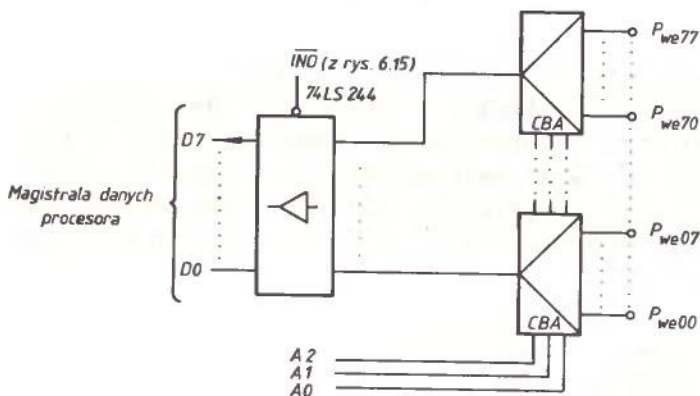
6.3.1. Układ równoległego wejścia i wyjścia

Omówienie układów wejścia-wyjścia rozpoczniemy od najprostszych – zestawu peryferyjnych linii wejściowych (których stan może być odczytywany przez procesor) oraz wyjściowych (które mogą być ustawiane przez procesor w stan 1 lub 0). Linie takie można stosować do odczytywania stanu przełączników lub danych przesyłanych z innych urządzeń (linie wejściowe) oraz do sterowania przekaźnikami, lampkami kontrolnymi i innymi urządzeniami (sygnały wyjściowe).



Rys. 6.15. Układ ośmiu peryferyjnych sygnałów wejściowych

Rys. 6.16. Układ 64 peryferyjnych sygnałów wejściowych

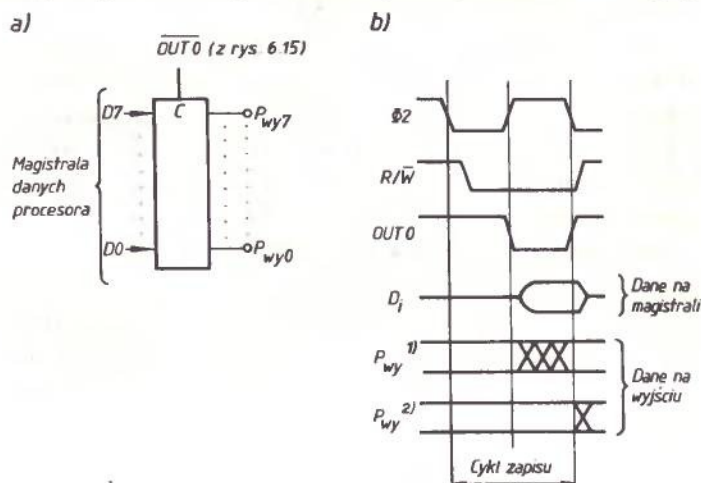


Przeciwny kierunek przepływu danych uzyskuje się stosując zamiast buforów 3-stanowych rejestr typu D, sterowany sygnałem $\overline{OUT} 0$ z dekodera (rys. 6.17).

W tym miejscu należy przypomnieć, że istnieją dwa typy przerzutników: sterowane poziomem sygnału i sterowane zboczem sygnału. Przerzutniki sterowane poziomem (ang. *latch*, np. 7475, 74373) są tańsze i zwykle mają mniejszy pobór mocy, lecz ich wadą jest to, że są przezroczyste w czasie zapisu (tzn. w całej fazie $\Phi 2$), co prowadzi do niestałości danych w cyklu zapisu (patrz rys. 6.17). Wszędzie tam, gdzie zależy nam na stałości danych wyjściowych (np.

gdy sygnał wyjściowy steruje urządzeniem czułym na zbocze, takim jak licznik) konieczne staje się stosowanie przerzutników sterowanych zboczem (np. 74174, 74374).

Często od urządzeń peryferyjnych wymaga się bardziej skomplikowanych funkcji niż tylko prosty zapis i odczyt informacji o postaci równoległej.



Rys. 6.17. Układ ośmiu peryferyjnych sygnałów wyjściowych: a) schemat; b) przebiegi czasowe; 1 – wyjście układu sterowanego poziomem (np. 74373), 2 – wyjście układu sterowanego zboczem (np. 74374)

Przykładem takich wymagań jest transmisja równoległa z potwierdzeniem (ang. *handshake*), stosowana np. w drukarkach (a także w rzadko stosowanych obecnie czytnikach i perforatorach taśmy papierowej) oraz bardziej złożone metody przesyłania danych (np. transmisja szeregową, synchroniczną lub asynchroniczną itp.). W takich przypadkach zalecane jest stosowanie układów scalonych dużego stopnia scalenia (MOS-VLSI), specjalizowanych do tego zadania. Poniżej zostaną opisane pokrótce najbardziej popularne z nich.

Układy transmisji równoległej. W rodzinie procesora 6502 są trzy układy peryferyjne, służące do przesyłania danych równoległych:

6520 – zawiera dwa równoległe porty 8-bitowe. Każda z linii peryferyjnych może być indywidualnie programowana jako wejściowa lub wyjściowa. Każdy z portów ma dodatkowo dwie linie sterujące, które mogą służyć bądź jako wejścia czułe na zbocze, bądź jako linie do potwierdzania przesyłania danych.

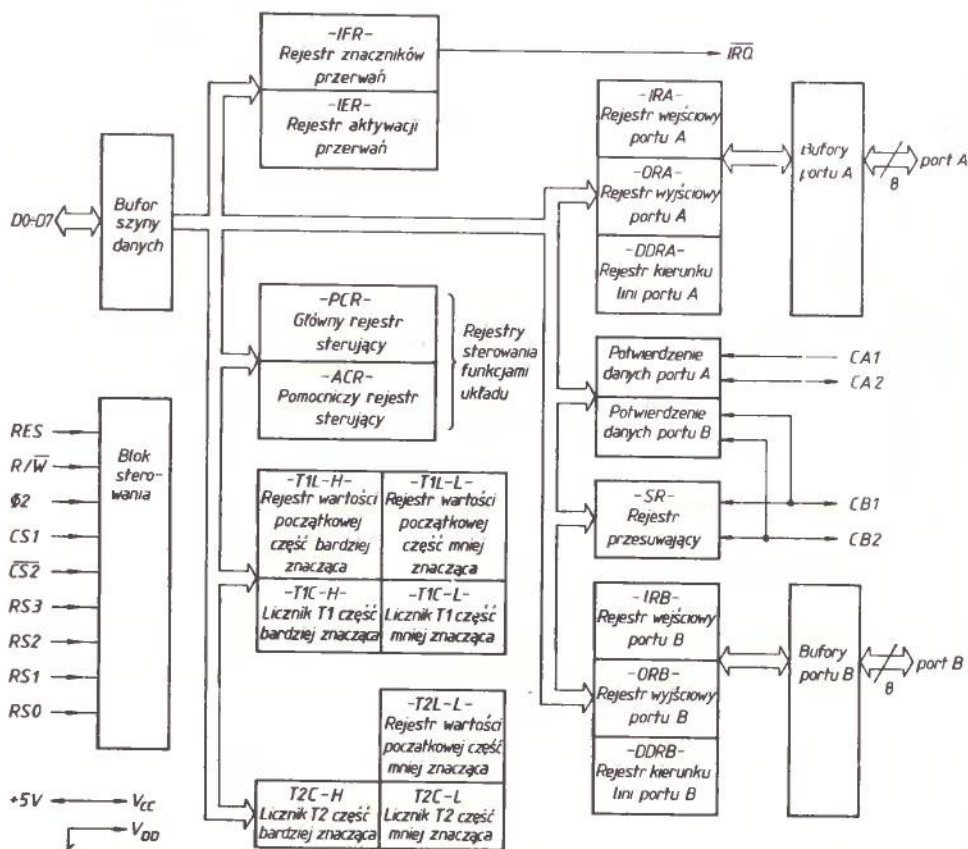
6522 – zawiera 2 równoległe porty 8-bitowe, o liniach indywidualnie programowanych jako wejściowe lub wyjściowe; porty te są wyposażone w linie sterujące. Dodatkowo układ jest wyposażony w dwa 16-bitowe liczniki (ang. *timer*), służące do odmierzania czasu bądź do zliczania impulsów, oraz 8-bitowy rejestr przesuwający stosowany do wprowadzania lub wyprowadzania danych szerego-

wych, a także generowania przebiegów prostokątnych, w tym sygnału akustycznego.

- 6532 — zawiera dwa 8-bitowe porty o liniach indywidualnie programowanych jako wejściowe lub wyjściowe (bez linii sterujących), jeden 8-bitowy licznik-czasomierz służący do odmierzania czasu oraz 128 bajtów pamięci (RWM).

W tej książce opiszemy dokładnie tylko układ 6522, jako najbardziej uniwersalny i najbardziej popularny. Układ 6520 jest zbliżony do 6522, ma jednak mniejsze możliwości, zaś układ 6532 opracowany dla małych systemów, w których wystarcza 128 bajtów pamięci RWM, praktycznie wyszedł z użycia, gdyż obecnie tak małe systemy są realizowane z użyciem mikrokomputerów jednoukładowych (np. R 6500/1, Z8 itp.).

Jak uprzednio było sygnalizowane, w skład układu 6522, stanowiącego uniwersalne urządzenie peryferyjne (ang. — *Versatile Interface Adapter*, VIA), wchodzi dwa 8-bitowe porty równoległe, dwa 16-bitowe liczniki oraz rejestr przesuwający. Schemat blokowy układu jest przedstawiony na rys. 6.18.



Rys. 6.18. Schemat blokowy układu 6522

Opis wyprowadzeń układu 6522:

RES (ang. *reset*) – wejście zerowania. Podanie na wejście zerowania stanu zero powoduje wyzerowanie wszystkich wewnętrznych rejestrów układu (poza rejestrami liczników i rejestrem przesuwającym), co programuje wszystkie linie peryferyjne jako wejścia, blokuje przerwania, zatrzymuje liczniki i rejestr przesuwający.

Tablica 6.3. Dostępne programowo rejestry układu 6522

Nr rejestru	RS3	RS2	RS1	RS0	Nazwa	Zapis	Odczyt
0	0	0	0	0	ORB/IRB	Rejestr wyjściowy portu B (output register B)	Rejestr wejściowy portu B (input register B)
1	0	0	0	1	ORA/IRA	Rejestr wyjściowy portu A (output register A)	Rejestr wejściowy portu A (input register A)
2	0	0	1	0	DDRB	Rejestr kierunku linii portu B (data direction register B)	
3	0	0	1	1	DDRA	Rejestr kierunku linii portu A (data direction register A)	
4	0	1	0	0	T1 C-L	Mniej znaczący bajt rejestru wartości początkowej licznika T1 (T1 low order latches)	Mniej znaczący bajt licznika T1 (T1 low order counter)
5	0	1	0	1	T1 C-H	Bardziej znaczący bajt licznika T1 (T1 high order counter)	
6	0	1	1	0	T1 L-L	Mniej znaczący bajt rejestru wartości początkowej licznika T1 (T1 low order latches)	
7	0	1	1	1	T1 L-H	Bardziej znaczący bajt rejestru wartości początkowej licznika T1 (T1 high order latches)	
8	1	0	0	0	T2 L-L	Mniej znaczący bajt rejestru wartości początkowej licznika T2 (T2 low order latches)	Mniej znaczący bajt licznika T2 (T2 low order counter)
9	1	0	0	1	T2 C-H	Bardziej znaczący bajt licznika T2 (T2 high order counter)	
10 (\$A)	1	0	1	0	SR	Rejestr przesuwający (shift register)	
11 (\$B)	1	0	1	1	ACR	Dodatkowy rejestr sterujący (auxiliary control register)	
12 (\$C)	1	1	0	0	PCR	Główny rejestr sterujący (peripheral control register)	
13 (\$D)	1	1	0	1	IFR	Rejestr znaczników przerwań (interrupt flag register)	
14 (\$E)	1	1	1	0	IER	Rejestr uaktywnienia przerwań (interrupt enable register)	
15 (\$F)	1	1	1	1	ORA/IRA	Rejestr portu A bez wpływu na automatyczne potwierdzanie danych.	

$\Phi 2$ – wejście zegarowe. Na wejście to podaje się sygnał zegara systemowego (wyjście $\Phi 2$ procesora). Służy ono do synchronizowania wszystkich operacji wewnętrznych układu.

$R/\overline{W}$ – wejście „odczyt/zapis”. Wejście to steruje kierunkiem przesyłania danych (stan $R/\overline{W} = 1$ odpowiada odczytowi danych przez procesor) i jest dołączone do wyjścia $R/\overline{W}$ procesora.

$D0 \div D7$ – linie danych. Osiem dwukierunkowych linii danych służy do wymiany danych między procesorem a układem peryferyjnym. Dołącza się je do linii danych procesora.

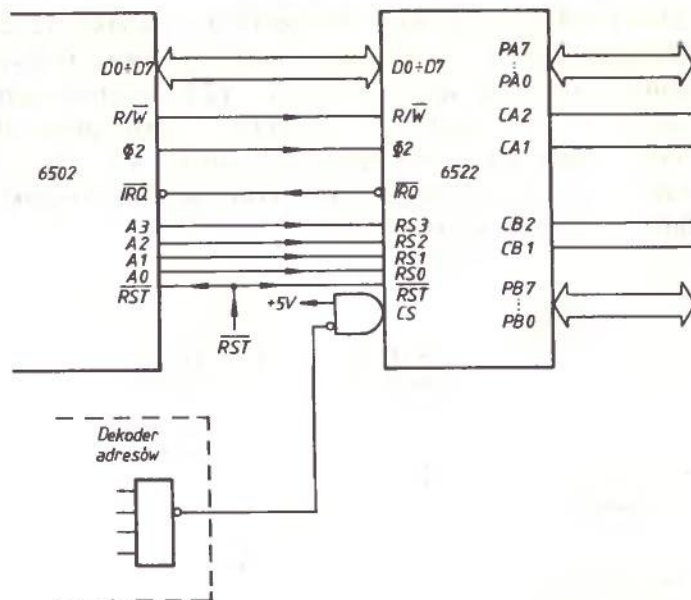
$CS1, CS2$ (ang. *chip select*) – wejścia wyboru układu. Są to wejścia służące do wyboru układu. Procesor ma dostęp do rejestrów wewnętrznych układu w tych cyklach zegarowych, w których $CS1 = 1$ i $\overline{CS2} = 0$. Dołącza się je do dekodera adresów (zwykle wykorzystując tylko linię $\overline{CS2}$, zaś linię $CS1$ dołączając do $+5V$).

$RS0 \div RS3$ (ang. *register select*) – linie wyboru rejestru. Cztery linie wyboru rejestru umożliwiają procesorowi dostęp do jednego z 16 wewnętrznych rejestrów układu 6522, jak przedstawiono w tabl. 6.3. Linie te zwykle dołącza się do najmniej znaczących linii adresowych procesora ($A0 \div A3$).

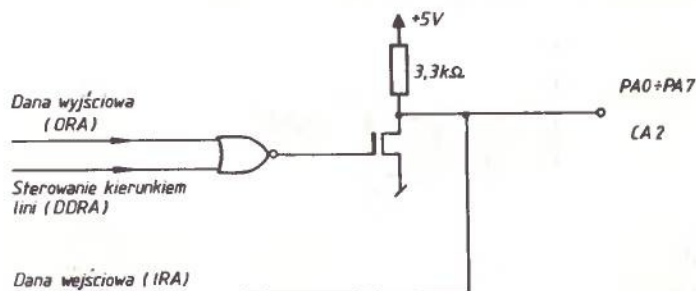
$\overline{IRQ}$ (ang. *interrupt request*) – linia zgłoszenia przerwania. Za pomocą tej linii układ zgłasza procesorowi żądanie obsługi. Ma to miejsce, gdy zostanie ustawiony znacznik w rejestrze przerwania i gdy odpowiadające mu znaczniki w rejestrze zezwolenia przerwania są ustawione (stan 1). Linia ta jest linią wyjściową typu „otwarty dren” (to samo, co wyjście typu „otwarty kolektor”), co umożliwia dołączenie bezpośrednie kilku urządzeń do wejścia $\overline{IRQ}$ procesora (realizując funkcję iloczynu montażowego).

Układ 6522 pochodzi z rodziny 65XX, a więc dołącza się go wprost do odpowiednich linii procesora, tzn. do linii danych ($D0 \div D7$), linii sterujących $RST, R/\overline{W}$ i $\Phi 2$, linii przerwania $\overline{IRQ}$. Układ uaktywnia się ustawiając linię $CS1$ w stan 1, a linię $\overline{CS2}$ w stan 0. Sposób dołączenia pokazano na rys. 6.19.

$PA0 \div PA7$ (ang. *peripheral port A*) – linie wejścia-wyjścia portu A. Port A składa się z 8 linii, które można indywidualnie programować jako wejściowe lub wyjściowe. Układy wyjściowe poszczególnych linii są typu „otwarty dren” (z wbudowanym opornikiem, rys. 6.20), co umożliwia realizację iloczynu montażowego (ang. *wired-or*) i np. traktowanie linii jako wejściowych i wyjściowych jednocześnie. Wewnętrzny rejestr wejściowy może być zaprogramowany jako „przezroczysty” (i wtedy przy operacji odczytu rejestru $\overline{IRA}$ odczytujemy stan poszczególnych linii portu) lub jako rejestr sterowany zboczem. W drugim przypadku zapamiętuje on stan linii portu A, jaki był w chwili przyścia aktywnego zbocza sygnału $CA1$. Aktywne zbocze sygnału $CA1$ programuje się w rejestrze PCR, zaś rodzaj pracy rejestru wejściowego – w rejestrze ACR. Linie portu A jako wejścia reprezentują sobą jedno standardowe obciążenie TTL i mogą (jako wyjścia)ysterować jedno wejście TTL.



Rys. 6.19. Dołączenie układu 6522 do procesora



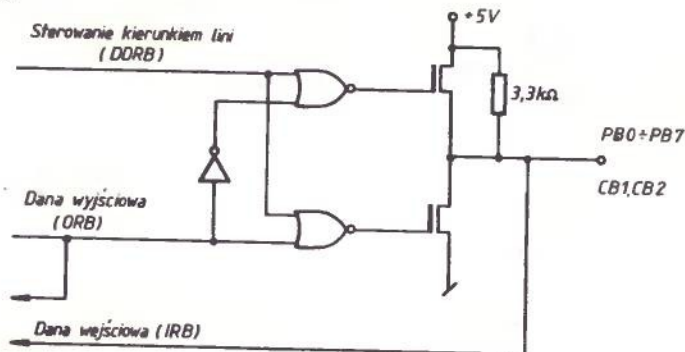
Rys. 6.20. Układ wyjściowy portu A

CA1 i CA2 — linie sterujące portu A. Linie sterujące portu A mogą służyć jako dwie niezależne linie wejściowe czułe na zbocze lub dwie realizujące funkcje automatycznego potwierdzania danych (typu *handshake*). W tym przypadku linia CA1 jest wejściowa, zaś CA2 — wyjściowa. Dodatkowo linia CA1 może sterować funkcją zapamiętywania danych w rejestrze wejściowym IRA.

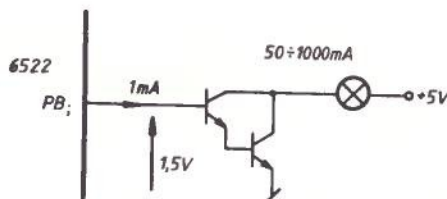
PB0 ÷ PB7 — linie wejścia-wyjścia portu B. Port B, podobnie do portu A, składa się z 8 indywidualnie programowanych linii wejścia-wyjścia, jednak układy wyjściowe są inne niż w porcie A (rys. 6.21). Operacja odczytu rejestru IRB dostarcza informacji o stanie tylko tych linii, które są zaprogramowane jako wejściowe. Dla linii zaprogramowanej jako wyjście jest czytany stan rejestru ORB, tzn. na wynik odczytu nie ma wpływu aktualne napięcie na linii (np. zwarcie jej do masy). Rejestr IRB, podobnie jak IRA, może pracować w dwóch trybach — jako przezroczysty i jako zapamiętujący — sterowany linią

CB1. Każda linia portu *B* reprezentuje sobą jako wejście jedno standardowe obciążenie TTL, jako wyjście może sterować jednym obciążeniem TTL. Zamiast obciążyć wyjście układem TTL, można zastosować tranzystorowy wzmacniacz w układzie Darlingtona. W tym przypadku wyjście będące w stanie 1 daje prąd 1 mA (przy napięciu 1,5 V) – rys. 6.21. Dodatkowo linia PB7 może być linią wyjściową sterowaną przez licznik T1, zaś linia PB6 – linią wejściową licznika T2.

a)



b)



Rys. 6.21. Układ wyjściowy portu *B* (a) i jego współpraca z układem Darlingtona (b)

CB1 i CB2 – linie sterujące portu *B*. Linie sterujące portu *B*, podobnie do linii CA1, CA2, mogą być zaprogramowane bądź jako dwie linie wejściowe czułe na zbocze, bądź jako linie sterujące przy automatycznym potwierdzaniu danych. Dodatkowo linie CB1 i CB2 mogą być zaprogramowane jako linie sterujące rejestrem przesuwającym (linia CB1 jako linia zegarowa, CB2 jako linia danych).

Programowanie kierunku transmisji linii peryferyjnych odbywa się przez wpisanie odpowiednich danych do rejestru kierunku portu. Ustawienie bitu rejestru kierunku (stan 1) odpowiada zaprogramowaniu linii odpowiadającej danemu bitowi jako linii wyjściowej, wpisanie zera – zaprogramowaniu linii jako wejściowej. Jeśli chce się linie PB1, PB3 i PB4 zaprogramować jako wyjściowe, należy wykonać następującą sekwencję programową:

```
LDA # %00011010 ; jedynki na pozycjach 1, 3, 4
STA DDRB        ; zapis do rejestru DDRB
```


Jeśli natomiast uprzednio zaprogramowana linia wyjściowa *PB3* ma być ustawiona w stan 1 (np. ma być zapalona lampka sterowana przez linię) bez zmiany stanu pozostałych linii, należy wykonać program:

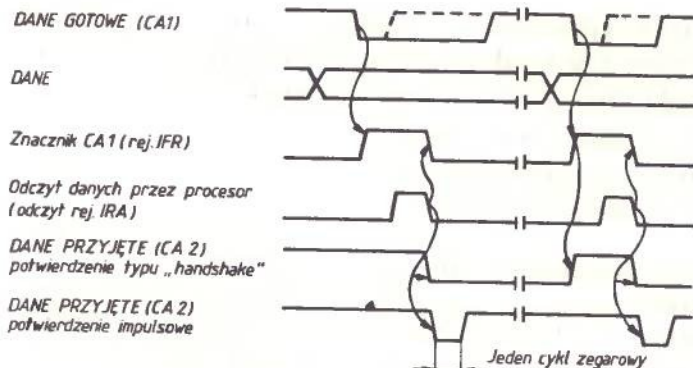
```
LDA ORB          ; odczyt stanu portu B
AND # %00001000 ; ustawienie bitu 3
STA ORB          ; zapis nowej wartości portu
```

Automatyczne sterowanie przesyłaniem danych z potwierdzeniem. Układ 6522 umożliwia automatyczne sterowanie przesyłaniem danych za pomocą sygnałów „DANE GOTOWE” i „DANE PRZYJĘTE” przesyłanych liniami sterującymi *CA1*, *CA2*, *CB1*, *CB2*. Port *A* (przy użyciu linii sterujących *CA1* i *CA2*) umożliwia automatyczne sterowanie zarówno wysyłaniem danych jak i odbiorem. Port *B* (przy użyciu linii sterujących *CB1* i *CB2*) umożliwia tylko automatyczne sterowanie przy wysyłaniu danych. Użycie portu *B* do odbioru danych jest również możliwe, lecz funkcje sterujące w tym przypadku należy realizować programowo. Przy odbiorze danych (przesyłaniu danych z innych urządzeń do mikrokomputera) urządzenie wysyłające dane sygnalizuje fakt wysłania sygnałem „DANE GOTOWE” (zwykle stanem 0) podawanym na wejście *CA1* (tylko port *A* ma automatyczne sterowanie przy odbiorze). Układ 6522 po rozpoznaniu aktywnego zbocza tego sygnału ustawia znacznik linii *CA1* w rejestrze znaczników przerwań *IFR*. Znacznik ten może bądź powodować przerwanie (jeśli jest ustawiony znacznik linii *CA1* rejestru uaktywnienia przerwań *IER*), bądź może być odczytany przez procesor w programowej pętli oczekiwania na dane. Odpowiednie przebiegi czasowe przedstawiono na rys. 6.22, a program pętli oczekiwania ma postać:

```
LOOP  LDA  IFR          ; ładowanie rejestru znaczników przer-
                        ; wania
      AND  # %00000010 ; wybór znaczników linii CA1
      BEQ  LOOP         ; testowanie znacznika
      LDA  IRA          ; odebranie danej z automatycznym
                        ; wysłaniem sygnału „DANE PRZY-
                        ; JĘTE”
```

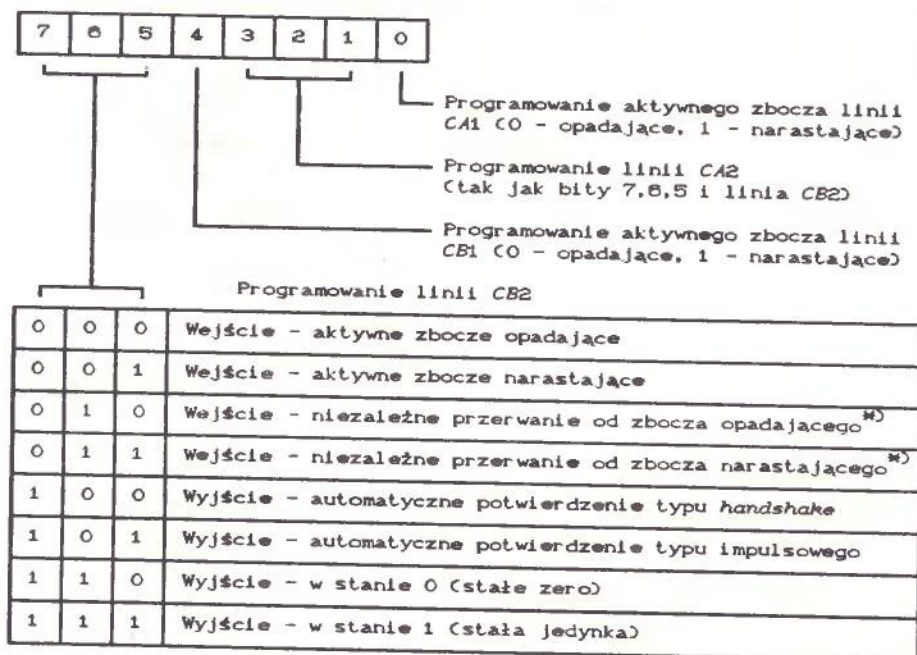
Po rozpoznaniu, że dana została przesłana, procesor odczytuje rejestr portu *A*, co powoduje automatyczne wysłanie sygnału „DANE PRZYJĘTE”.

W przypadku danych wysyłanych (przesłaniach w drugim kierunku — od mikrokomputera do urządzeń zewnętrznych) sytuacja jest podobna, przy czym wykorzystywać można zarówno port *A* jak i port *B*. Zapis danych do rejestru wyjściowego (*ORA* lub *ORB*) powoduje wysłanie sygnału „DANE GOTOWE” na linię *CB2* lub *CA2*. Dołączone urządzenie sygnalizuje przyjęcie danych wysyłając impuls potwierdzający — „DANE PRZYJĘTE” — na linię *CB1* (*CA1*), a to powoduje ustawienie znacznika linii *CB1* (*CA1*) w rejestrze znaczników przerwań i — jeżeli przerwanie to jest aktywne (tzn. jeżeli jest ustawiony znacznik linii *CB1* w rejestrze uaktywnienia przerwania) —



Rys 6.22. Odbiór danych z programową pętlą oczekiwania i automatycznym potwierdzeniem

zgłoszenie na linii $\overline{IRQ}$ żądania obsługi procesora. Programowanie rodzaju pracy linii sterujących portów (aktywnego zbocza linii wejściowych, tzn. czy znacznik przerwania ustawia się przy zboczach opadającym, czy narastającym) oraz trybu pracy linii wyjściowych odbywa się przez odpowiednie ustawienie bitów głównego rejestru sterującego PCR – patrz rys. 6.23. Na przykład, aby



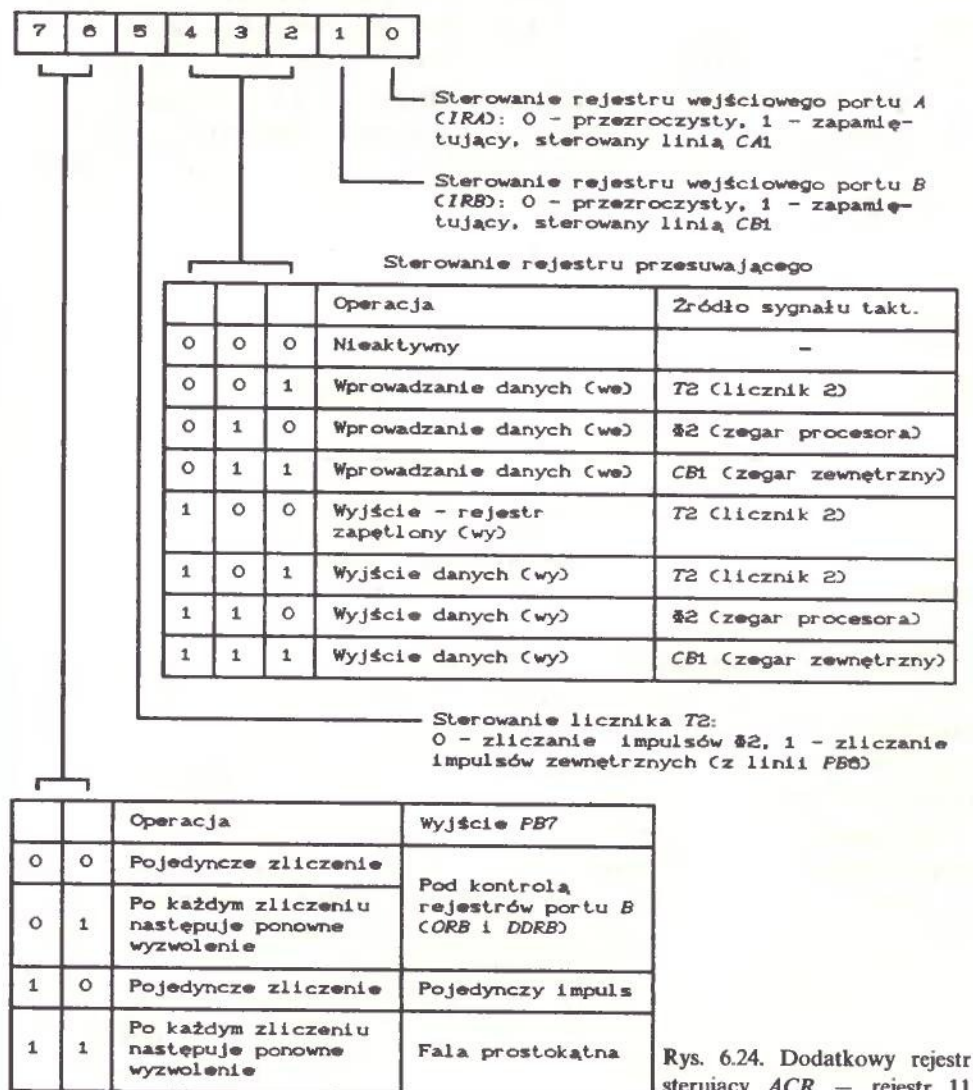
^{*)} Przy zaprogramowaniu linii CA2 (CB2) jako niezależnego wejścia kasowanie znacznika przerwania od linii CA2 (CB2) odbywa się tylko za pomocą rozkazu kasowania przerwania (odpowiedniego wpisu do rejestru znaczników przerwania), nie zaś przy zapisie (odczytanie) portu.

Rys. 6.23. Główny rejestr sterujący – PCR

zaprogramować linie sterujące portu A jako linie spełniające funkcje automatycznego potwierdzenia (typu *handshake*) z aktywnym zboczem opadającym, a linie sterujące portu B jako dwie niezależne linie wejściowe z aktywnymi zboczami narastającymi, należy wykonać następujący program:

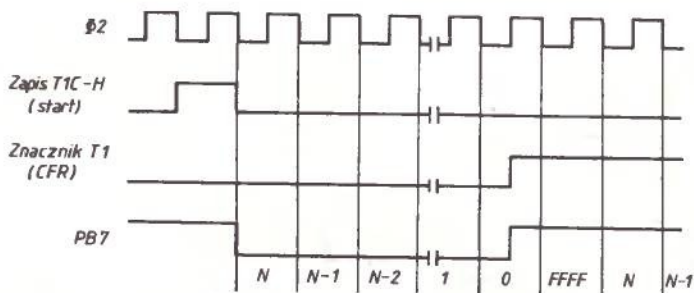
```
LDA # %01111000 ; słowo programujące
STA PCR          ; zapis do rejestru 12
```

W skład układu 6522 wchodzi także dwa 16-bitowe liczniki, mogące służyć do zliczania czasu, generowania przerwań w stałych odstępach czasu, zliczania impulsów zewnętrznych itp. Tryb pracy liczników jest programowany ustawieniem bitów w dodatkowym rejestrze sterującym (ARC – rejestr nr 11), którego funkcje pokazano na rys. 6.24.



Rys. 6.24. Dodatkowy rejestr sterujący ACR – rejestr 11

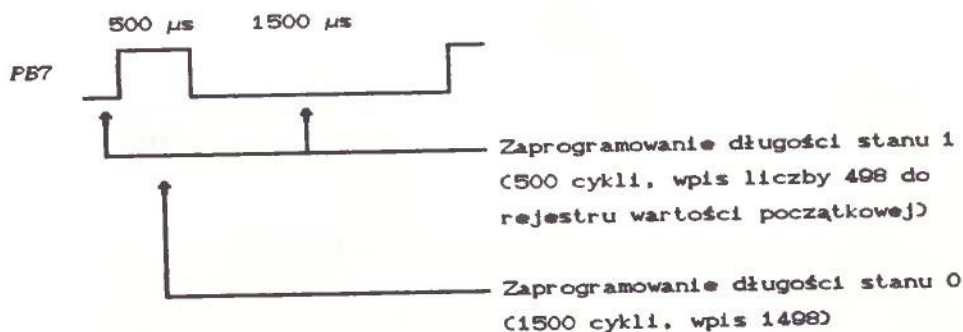
Licznik $T1$ jest taktowany zawsze zegarem systemowym (zliczając impulsy $\Phi 2$). Pracować może w dwóch rodzajach pracy: z pojedynczym impulsem lub z powtarzaniem. W pierwszym przypadku, po załadowaniu licznika $L1$ (zapis mniej znaczącego bajtu do rejestru $T1\ L-L$ lub $T1\ C-L$, a następnie zapis bardziej znaczącego bajtu do rejestru $T1\ C-H$) następuje zliczanie w dół z częstotliwością $\Phi 2$ do osiągnięcia zera. Po osiągnięciu zera jest ustawiany znacznik przerwania od $T2$ w rejestrze znaczników przerwań (IFR – rejestr 13) i, jeżeli przerwanie jest aktywne (tzn. jest ustawiony znacznik $T2$ w rejestrze uaktywniania przerwań IER), następuje zgłoszenie żądania obsługi do procesora (podanie zera na linię IRQ). Jeżeli bit $b6$ dodatkowego rejestru rozkazowego jest ustawiony, to na linii $PB7$ pojawi się pojedynczy impuls prostokątny o czasie trwania równym $N + 1,5$ cyklu zegarowego (rys. 6.25). Po zliczeniu do zera proces zliczania nie jest zatrzymywany (powtarza się, rozpoczynając od poprzednio zaprogramowanej wartości N), jednak nie ma on już wpływu na znacznik $T2$ oraz sygnały IRQ i $PB7$. Przedstawionego trybu pracy używa się do generacji pojedynczych impulsów prostokątnych oraz do zliczania czasu koniecznego do zajścia pewnych zdarzeń (np. czasu zadziałania przekładników).



Rys. 6.25. Licznik $T1$ pracujący w trybie pojedynczego zliczania, sterujący linią $PB7$

Jedną z podstawowych właściwości licznika $T1$ jest to, że poza dwoma rejestrami licznika ($T1\ C-L$ i $T1\ C-H$) ma on dwa rejestry wartości początkowej ($T1\ L-L$ i $T1\ L-H$), do których można wprowadzić dane, które następnie będą automatycznie ładowane do licznika za każdym razem, gdy licznik osiągnie zero (a dokładniej -1). W przypadku ustawienia bitu $b8$ dodatkowego rejestru sterującego otrzymujemy drugi tryb pracy – z powtarzaniem (ang. *free-run*), który to tryb daje możliwość generowania przerwań procesora co stały odcinek czasu. Ze względu na obecność rejestrów wartości początkowej – na czas między przerwaniami nie wpływa zmienny czas wykonania przez procesor programu obsługi przerwań (zależny od czasu trwania i stopnia zaawansowania wykonywania bieżącego rozkazu), można więc licznik $T1$ stosować jako zegar czasu rzeczywistego, o dokładności ograniczonej jedynie stabilnością częstotliwości zegara systemowego $\Phi 2$. W

tym trybie pracy, po ustawieniu bitu *b7* dodatkowego rejestru sterującego (*ACR*) na linii *PB7* pojawia się sygnał prostokątny o okresie równym $2 \cdot (N + 2)$ cykli zegarowych i wypełnieniu równym $1/2$. Stosując licznik *T1* w tym trybie pracy, można (bez angażowania procesora) uzyskiwać sygnał o częstotliwości od 7,63 Hz do 250 kHz (dane te dotyczą pracy z zegarem 1 MHz). W trybie pracy z powtarzaniem zawartość rejestrów wartości początkowej jest przepisywana do licznika w chwili rozpoczęcia procesu zliczania do zera, natomiast dostęp procesora do tych rejestrów nie ma bezpośredniego wpływu na proces zliczania. Jeśli więc przed kolejnym zliczaniem do zera do rejestru wartości początkowej wpisze się nową wartość, to następny cykl zliczania będzie miał inną długość. Można to wykorzystać na przykład do generacji fali prostokątnej o współczynniku wypełnienia innym niż 0,5. Niżej jest przedstawiony program służący do generowania na linii *PB7* fali prostokątnej o częstotliwości 500 Hz i wypełnieniu 0,25, a na rys. 6.26 kształt przebiegu czasowego.



Rys. 6.26. Wytwarzanie przebiegu prostokątnego o częstotliwości 500 Hz i wypełnieniu $1/4$ — przebieg czasowy

		; inicjalizacja licznika
INIT	LDA ARC	; pobranie poprzedniego stanu pomocniczego rejestru sterującego
	AND # %00111111	; wyzerowanie bitów sterujących licznik T1
	ORA # %11000000	; tryb pracy <i>free run</i> , linia <i>PB7</i> sterowana licznikiem
	STA ARC	; zaprogramowanie licznika T2
	LDA # 218	; mniej znacząca część liczby 1498
	STA T1.L-L	; wpisanie do rejestru 6
	LDA # 5	; bardziej znacząca część liczby 1498 (1500 cykli)
	STA T1.L-H	; wpisanie do rejestru 7
	STA T1.C-H	; wpisanie do rejestru 5, uruchomienie licznika

			; program główny
LOOP	JSR	WAIT	; oczekiwanie na zliczenie
	LDA	ORB	; odczytanie stanu linii PB7 (całego portu B)
	BMI	PB7.ON	; skok, jeśli linia jest w stanie 1
PB7.OFF	LDA	# 242	; programowanie zliczenia do 500
	STA	T1.L-L	; długość stanu 1
	LDA	# 1	; bardziej znacząca część liczby 498 (500-2)
	STA	T1.L-H	
	JMP	LOOP	; przejście do następnego stanu
PB7.ON	LDA	# 218	; programowanie zliczania do 1500
	STA	T1.L-L	; długość stanu 0
	LDA	# 5	; bardziej znacząca część liczby 1498 (1500-2)
	STA	T1.L-H	
	JMP	LOOP	
			; procedura oczekiwania na zliczenie
WAIT	BIT	IFR	; testowanie znacznika T1
	BVC	WAIT	; oczekiwanie na ustawienie znacznika
	LDA	T1.C-L	; skasowanie znacznika T1
	RTS		; koniec podprogramu

Licznik T2 został zaprojektowany do pracy w dwóch podstawowych trybach pracy: z pojedynczym impulsem lub ze zliczaniem impulsów zewnętrznych. Pierwszy typ pracy jest identyczny jak dla licznika T1. Jedyną różnicą jest to, że po doliczeniu do -1 do licznika nie wpisuje się wartości ostatnio zaprogramowanej (N), lecz wartość -2 (\$FFFE).

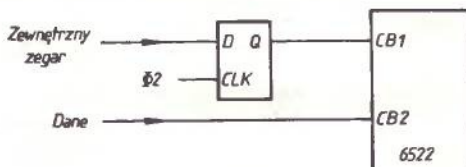
W trybie ze zliczaniem impulsów zewnętrznych (z linii PB6) — po zaprogramowaniu licznika (wpisaniu wartości początkowych do rejestrów T2 L-L i T2 C-H) każdy impuls na linii PB6 jest traktowany jako sygnał zmniejszenia zawartości licznika o jeden (modulo $2^{16} = 65536$). Znacznik przerwania zostaje ustawiony po osiągnięciu przez licznik stanu 0. Znacznik ten jest zerowany przez odczyt T2 C-L lub zapis T2 C-H. Do ponownego ustawienia znacznika przerwania konieczne jest ponowne zaprogramowanie licznika wpisem do T2 C-H.

Układ 6522 jest układem synchronicznym taktowanym zegarem $\Phi 2$, dlatego też szybkość zmian wszystkich sygnałów peryferyjnych jest ograniczona (nie dotyczy to sygnałów służących do łączenia układu 6522 z mikroprocesorem). Aby stan sygnału (0 lub 1, a co za tym idzie — odpowiednie zbocza) mógł być poprawnie rozpoznany, stan ten musi trwać co najmniej jeden pełny cykl zegara, więc praktycznie największa częstotliwość sygnałów peryferyjnych (a w tym sygnału wprowadzanego na licznik 2 poprzez linię PB6) wynosi 250 kHz przy zegarze 1 MHz.

Dodatkowym trybem pracy licznika T2 jest wytwarzanie symetrycznej fali prostokątnej sterującej rejestrem przesuwającym. Ponieważ licznik T2 ma rejestr wartości początkowej tylko dla mniej znaczącego bajtu danych, więc częstotliwości generowane przez ten licznik mieszczą się w przedziale $3,9 \text{ kHz} \div 500 \text{ kHz}$ (dla częstotliwości zegara 1 MHz).

Układ 6522 jest wyposażony w 8-bitowy **rejestr przesuwający**, sterowany wewnętrznym licznikiem modulo 8. Rejestr ten umożliwia odczyt danych szeregowych podawanych na linię CB2 lub też wysyłanie tych danych (też przez linię CB2). Zegarem taktującym operacje wejścia-wyjścia z rejestru przesuwającego może być (w zależności od słowa sterującego umieszczonego na bitach 2, 3, 4 dodatkowego rejestru sterującego ACR) zegar systemowy $\Phi 2$, wyjście licznika T2 (sygnał $\Phi 2$ podzielony przez zawartość licznika) lub sygnał zewnętrzny podawany na linię CB1. Jeśli sygnałem sterującym jest sygnał wewnętrzny, to pojawia się on również na linii CB1 (traktowanej jako wyjściowa). Jeżeli sygnałem sterującym licznik jest sygnał zewnętrzny, to należy

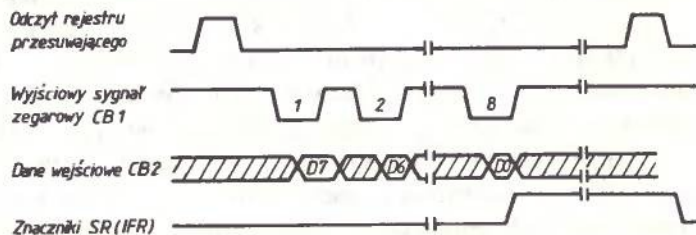
Rys. 6.27. Synchronizacja zewnętrznego przebiegu zegarowego rejestru przesuwającego z zegarem procesora



zapewnić jego synchronizację z zegarem $\Phi 2$ (rys. 6.27) np. za pomocą pojedynczego przerzutnika typu D. Rejestr przesuwający ma praktycznie cztery sposoby pracy:

1. Rejestr wyłączony (bity b2, b3, b4 rejestru 11 są w stanie 0). Nie jest dokonywana żadna operacja na rejestrze.

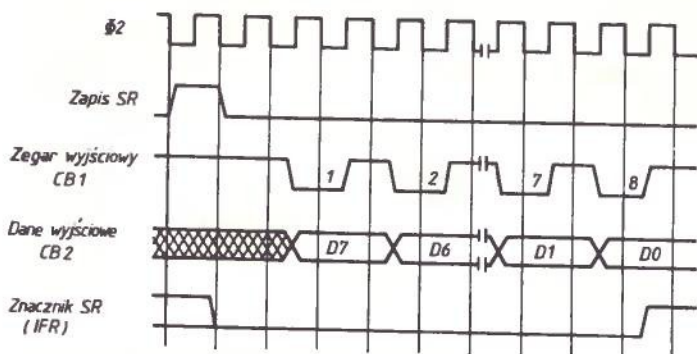
2. Rejestr w trybie wejściowym. Po uaktywnieniu rejestru (przez zapis lub odczyt rejestru SR) następuje wczytanie 8 kolejnych bitów informacji (stanów linii CB2) poczynając od bitu bardziej znaczącego, po czym zostaje ustawiony znacznik SR i wygenerowane przerwanie. Źródłem sygnałów taktujących dane może być zegar $\Phi 2$ lub znacznik T2 (i w tym przypadku na linii CB1 pojawi się 8 impulsów; na dodatnich zboczach tych impulsów jest realizowany zapis danych — rys. 6.28). Źródłem sygnału taktu przesunięcia może być także zegar zewnętrzny, dołączony do układu przez linię CB1. W tym przypadku, po



Rys. 6.28. Wprowadzanie danych za pomocą rejestru przesuwającego (z wewnętrznym zegarem)

przeczytaniu 8 bitów danych, jest ustawiany znacznik przerwania, lecz nie jest zatrzymywany rejestr przesuwający.

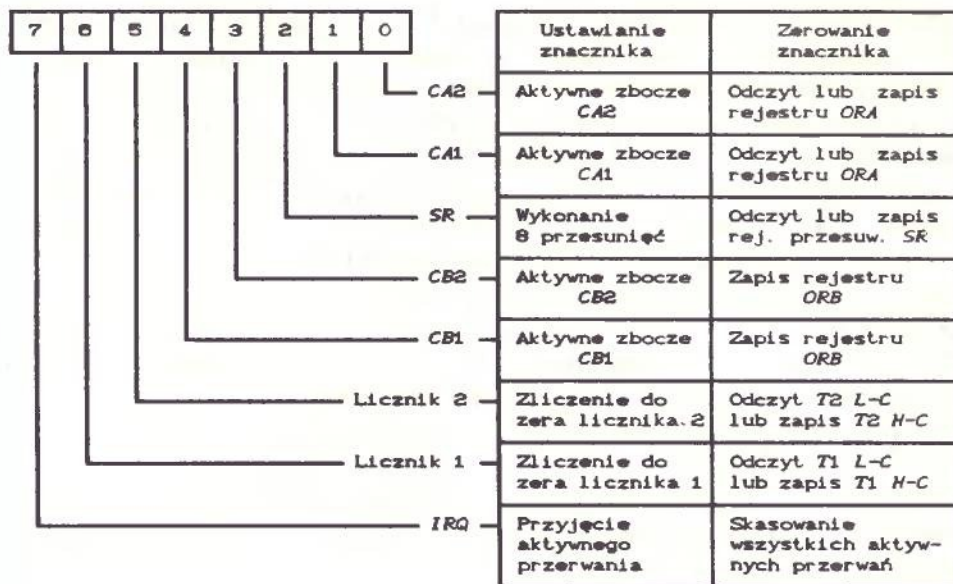
3. Rejestr w trybie wyjściowym. W tym trybie pracy rejestr przesuwający wysyła informację szeregową na zewnątrz (poczynając od bitu najbardziej znaczącego). Jeśli operacje przesłania są sterowane sygnałami wewnętrznymi, to rejestr przesuwający zatrzymuje się po wysłaniu 8 bitów. Jeśli zaś sterowanie jest realizowane poprzez podanie sygnału zewnętrznego na linię CB1, to po wysłaniu 8 bitów jest ustawiany znacznik przesuwu SR, lecz operacje wysyłania nie są przerwane (rejestr przesuwający jest zapętlony — dana wychodząc na zewnątrz jest jednocześnie kierowana do najmniej znaczącego bitu rejestru — rys. 6.29).



Rys. 6.29. Praca rejestru przesuwającego w trybie wyjściowym, sterowana zegarem systemowym ($\Phi 2$)

4. Rejestr w trybie wyjściowym z powtarzaniem. Ten tryb pracy jest bardzo zbliżony do trybu wyjściowego sterowanego licznikiem T2. Jedyną różnicą jest to, że po wysłaniu 8 bitów danych rejestr przesuwający nie jest zatrzymywany, lecz proces powtarza się (z zapętleniem danych).

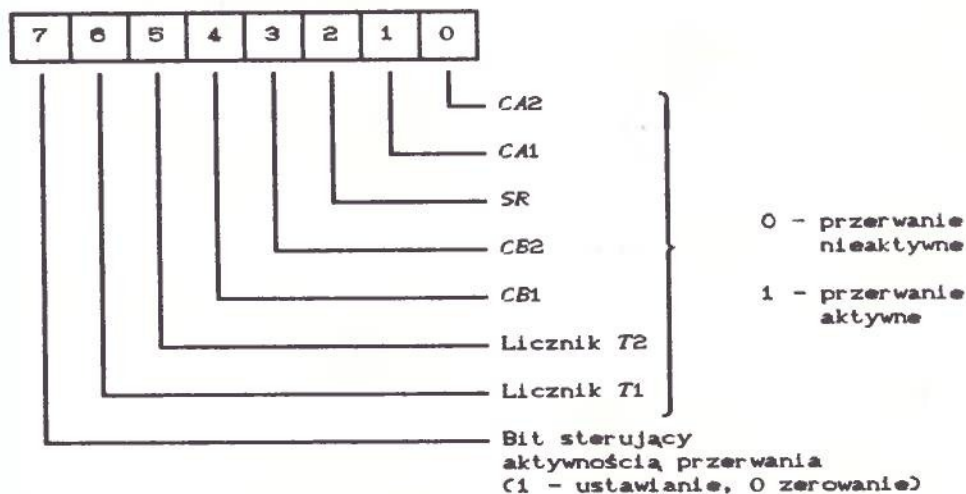
Rejestry przerw i obsługa przerw. Układ wytwarzania sygnału przerwania $\overline{IRQ}$ jest sterowany zawartością rejestru znaczników przerw (IFR — rejestr 13) lub rejestru uaktywnienia przerw (IER — rejestr 14). Linia $\overline{IRQ}$ jest w stanie 0, gdy jednocześnie jest ustawiony znacznik przerwania i odpowiadający mu znacznik uaktywnienia przerwania. Wszystkie znaczniki przerw IFR są przedstawione na rys. 6.30. Znaczniki są ustawiane w wyniku zajścia zdarzenia zewnętrznego (np. wystąpienia aktywnego zbocza na linii CA1) lub wewnętrznego (np. zliczenia licznika T1 do zera). Kasowanie znaczników przerw dokonuje się bądź automatycznie, przy odczycie odpowiedniego rejestru, bądź przez wpisanie do odpowiednich bitów rejestru znaczników przerw wartości 1. Bit 7 rejestru IFR nie jest takim samym znacznikiem jak poprzednie. Sygnalizuje on, że układ jest źródłem aktywnego przerwania (co wykorzystuje się w programie obsługi przerwania) i wraca do stanu zerowego dopiero po obsłudze wszystkich aktywnych przerw układu



Rys. 6.30. Rejestr IFR

6522. Bit 7 rejestru IFR jest sumą logiczną wszystkich niezamaskowanych (czyli aktywnych) znaczników przerw. Dlaczego takie rozwiązanie jest wygodne? Widać to doskonale na rys. 6.33, a także wspominaliśmy o tym przy omawianiu rozkazu BIT (rozdz. 3).

Ostatnim z omawianych rejestrów układu 6522 jest rejestr uaktywnienia przerw (IER, rejestr nr 14 – rys. 6.31). Steruje on, wspólnie z rejestrem IFR, stanem linii $\overline{IRQ}$ ($\overline{IRQ} = IFR6 \cdot IER6 + IFR5 \cdot IER5 + \dots + IFR0 \cdot IER0$). Rejestr ten ma dość specyficzną cechę: inaczej zachowuje się przy zapisie, inaczej



Rys. 6.31. Rejestr IER

przy odczycie. Przy odczycie poszczególne bity oznaczają aktywność przerwań (1 – przerwanie aktywne), zaś bit 7 jest zawsze równy 1. Operacja zapisu do tego rejestru bardzo różni się od operacji zapisu pozostałych rejestrów. Zapis do rejestru *IER* powoduje wykonanie jednego z dwóch rozkazów: selektywnego uaktywnienia przerwań lub selektywnego ich kasowania. O tym, która z tych dwóch operacji jest wykonywana, decyduje wartość wpisywana do bitu *b7* rejestru *IER*. Jeśli jest to jedynka, to dokonuje się selektywnego uaktywnienia przerwań, co oznacza, że wszystkie znaczniki *IER*, do których wpisuje się jedynkę, będą ustawione (przerwania będą aktywne). Jeśli zaś bit 7 ma przy zapisie wartość 0, to dokonuje się selektywnego zerowania znaczników *IER*. Zerowane są tylko te znaczniki, do których wpisuje się jedynkę. W obu przypadkach wpis zera nie powoduje zmiany stanu aktywności przerwania. Jeśli na przykład mają być uaktywnione przerwania od wejścia *CA1* i od rejestru przesuwającego, to należy do rejestru *IER* wpisać wartość \$85 (%10000110). Przykład programu rozpoznawania źródła przerwania w układzie 6522 przedstawiono poniżej. W konkretnym zastosowaniu program ten można uprościć, usuwając z niego fragmenty obsługi tych źródeł przerwań, które nigdy nie będą aktywne.

	IRQ. 6522	LDA	IFR		; odczyt rejestru znaczników przerwań
		BPL	IRQ. NEXT		; nie ma przerwania od układu
		AND	IER		; testowanie tylko przerwań aktywnych
		ROR	A		; testowanie przerwania od linii CA2
		BCC	ET1	}	program obsługi przerwania pochodzącego od linii CA 2
		:			
		:			
ET1		ROR	A	}	program obsługi przerwania pochodzącego od linii CA 1
		BCC	ET2		
		:			
		:			
ET2		ROR	A	}	program obsługi przerwania pochodzącego od rejestru przesuwającego
		BCC	ET3		
		:			
		:			
ET3		ROR	A	}	program obsługi przerwania pochodzącego od linii CB2
		BCC	ET4		
		:			
		:			

ET4	ROR	A	
	BCC	ET5	
	:		
	:		
			} program obsługi przerwania pochodzącego od linii CB1
ET5	ROR	A	
	BCC	ET6	
	:		
	:		
			} program obsługi przerwania pochodzący od licznika T1
ET6	ROR	A	
	BCC	IRQ. NEXT	
	:		
	:		
			} program obsługi przerwania pochodzący od licznika T2
IRQ. NEXT	:		
	:		
	:		
			} program testowania innych możli- wych źródeł przerwań (poza ukła- dem 6522)

Zastosowanie układu 6522. Ze względu na dużą liczbę funkcji, które realizuje układ 6522, znalazł on (i jego dalsze wersje, np. 6526) szerokie zastosowanie w urządzeniach mikroprocesorowych (nie tylko opartych na procesorze 6502).

Poniżej są opisane przykłady najpopularniejszych zastosowań układu. Aby przykłady były możliwie czytelne, podano nie tylko schematy elektryczne, ale również programy bądź ich fragmenty. Najczęściej układ 6522 znajdziemy w sprzęgach równoległych, np. do drukarki. Układ zawiera dwa porty wejścia-wyjścia, umożliwia więc dołączenie dwóch urządzeń wyposażonych w sprzęg równoległy, lecz w tym przypadku liczba linii sterująco-kontrolnych wynosi 2 („DANE GOTOWE” i „DANE POBRANE”), co zwykle jest niewystarczające. Poza liniami danych i dwiema liniami sterującymi przesyłaniem danych w sprzęgach drukarek występują zazwyczaj inne linie sterująco-kontrolne, takie jak zewnętrzne zerowanie urządzenia (*reset*), kontrola sprawności (np. linia „BRAK PAPIERU”), aktywności urządzenia itp. Przykład schematu dołączenia drukarki jest przedstawiony na rys. 6.32, a przykład programów obsługi — poniżej.

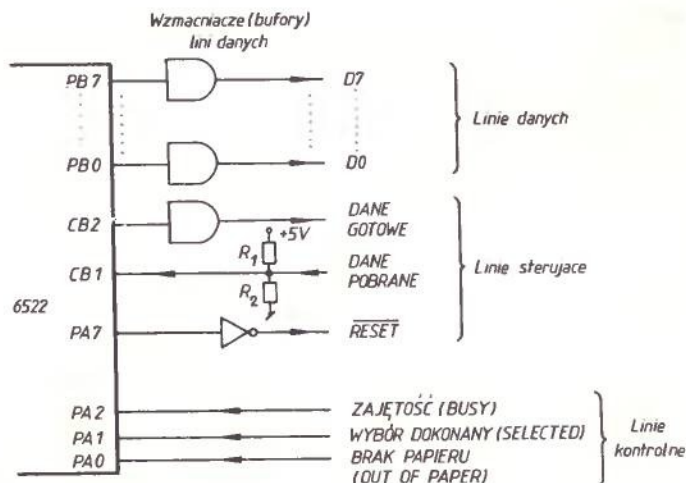
```

; inicjalizacja
INIT  LDA  PCR                ; zaprogramowanie trybu
      AND  # %00001111      ; handshake z aktywnym
      ORA  # %10000000      ; opadającym zboczem
      STA  PCR
      LDA  # $FF            ; zaprogramowanie portu B jako
      STA  DDRB              ; wyjścia
      LDA  DDRA              ; zaprogramowanie linii PA7 ja-
      AND  # %01111000      ; ko wyjścia, PA0 ÷ PA2 jako

```

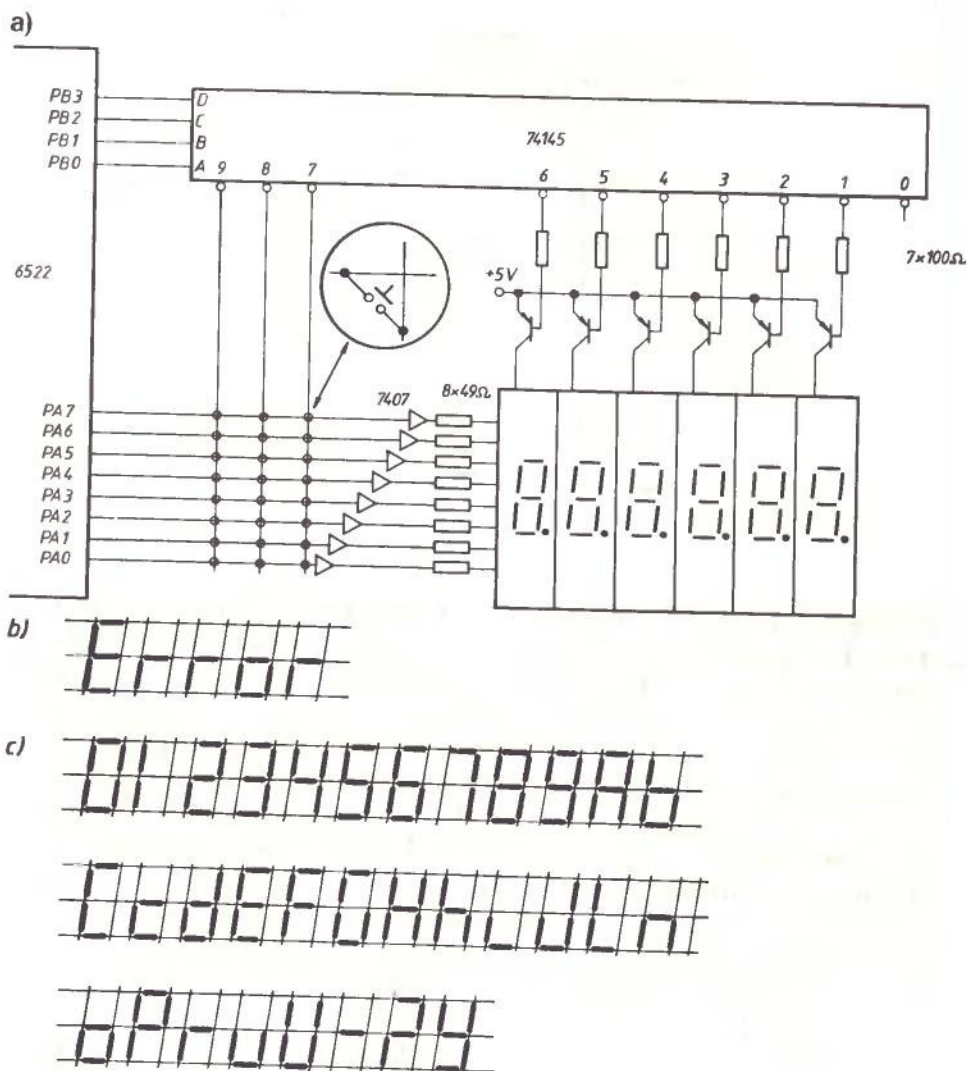

	ORA	# %10000000	; wejścia, bez zmiany kierunku
	STA	DDRA	; linii PA3 ÷ PA6
	LDA	ORA	
	ORA	# %10000000	; wysłanie rozkazu RESET
	STA	ORA	
	AND	# %01111111	
	STA	ORA	; zakończenie rozkazu RESET
LOOP.INIT	LDA	IRA	; oczekiwanie na gotowość —
	AND	# %00000100	; zgłoszone stanem 0 na linii
			; „BUSY”
	BNE	LOOP.INIT	
	LDA	# \$13	; wysłanie rozkazu CR —
	STA	ORB	; powrót głowicy do początku
	RTS		; linii
SEND	TAY		; wysłanie znaku z akumulatora
	LDA	ORA	; sprawdzenie, czy linie kontrolne
	AND	# %00000111	; są w odpowiednim stanie
	CMP	# %00000000	
	BNE	ERROR	
LOOP.SEND	LDA	IFR	; testowanie, czy pobrany został
	AND	# %00010000	; przez drukarkę poprzednio wysłany
	BEQ	LOOP.SEND	; znak
	STY	ORB	; wysłanie znaku
	RTS		
ERROR			; procedura obsługi błędu
	.		
	.		
	.		

Należy zwrócić uwagę na fakt, że zwykle zachodzi konieczność buforowania wyjść układu 6522 (co oznacza włączenie bramek, zwykle nienegujących, między wyjścia układu a wejścia dołączonych urządzeń). Spowodowane jest to tym, że obciążalność układu 6522 wynosi tylko 1 TTL (tzn. można sterować tylko jedną bramkę TTL), co zwykle jest niewystarczające do odpowiedniego wysterowania wejść drukarki (lub innych urządzeń), szczególnie przy większej długości kabla pomiędzy mikrokomputerem a drukarką. Należy również zwrócić uwagę na dzielnik rezystancyjny R_1 , R_2 , który jest stosowany w celu zapewnienia dopasowania falowego kabla z wejściem sygnału „DANE POBRANE”. Dopasowanie to jest szczególnie ważne w układach czułych na zbocze, w których pojawienie się sygnału odbitego może być błędnie zinterpretowane jako nowy sygnał. Bardziej szczegółowe rozwiązania problemów buforowania i dopasowań Czytelnik znajdzie w innych książkach [9, 14].



Rys. 6.32. Dołączenie drukarki za pomocą układu 6522

Zastosowanie układu 6522 do dołączenia klawiatury i wyświetlaczy 7-segmentowych. Do współpracy człowieka z komputerem stosuje się zwykle dwa podstawowe urządzenia: klawiaturę do przesyłania informacji w kierunku do mikrokomputera oraz układ wyświetlania informacji bądź na ekranie telewizora, bądź na zestawie wskaźników alfanumerycznych. Dokładniejsze omówienie tego typu układów Czytelnik może znaleźć w innych książkach [1, 14], tu zaś chcemy przedstawić najprostsze rozwiązanie, bardzo charakterystyczne dla układu 6522. Rysunek 6.33 przedstawia schemat układu umożliwiającego sterowanie wyświetlaczem siedmiosegmentowym, zapewniającego wyświetlenie sześciu cyfr oraz odczyt stanu 24-klawiszowej klawiatury. Proces wyświetlania jest oparty na zasadzie podziału czasowego. W danej chwili jest sterowane tylko jedno pole wyświetlacza, jedna cyfra. Efekt świecenia wszystkich pól uzyskuje się dzięki szybkiemu przełączaniu kolejno świecących się cyfr. Licznik T1 układu 6522 programuje się tak, aby wytwarzał aktywne przerwanie co około 4 ms i tyle też wynosi czas świecenia każdego pola cyfry. Program obsługi przerwań przy każdym przerwaniu zmienia kolejno świecące się pole cyfry, zapisując do czterech mniej znaczących bitów portu B numer aktualnie świecącej się cyfry. O tym, co jest wyświetlane, decyduje stan linii portu A. Uważny Czytelnik zauważy w tym miejscu, że zamiast ośmiu linii portu A można by zastosować tylko cztery oraz dekodery kodu BCD na kod wskaźnika 7-segmentowego (np. układ 7447). Rozwiązanie to byłoby niewątpliwie oszczędniejsze, jednak ograniczyłoby to zastosowanie wyświetlacza tylko do wyświetlania liczb (a poza tym cyfry 6 i 9 nie miałyby „ogonków”). W naszym rozwiązaniu wyświetlić można nie tylko cyfry, ale też większość liter, co może znacznie ułatwić „dogadanie” się komputera z człowiekiem. Zapisanie na 4 mniej znaczących bitach portu B liczby z przedziału od 1 do 6 powoduje włączenie pola kolejnej cyfry, natomiast liczb 7, 8 i 9 powoduje

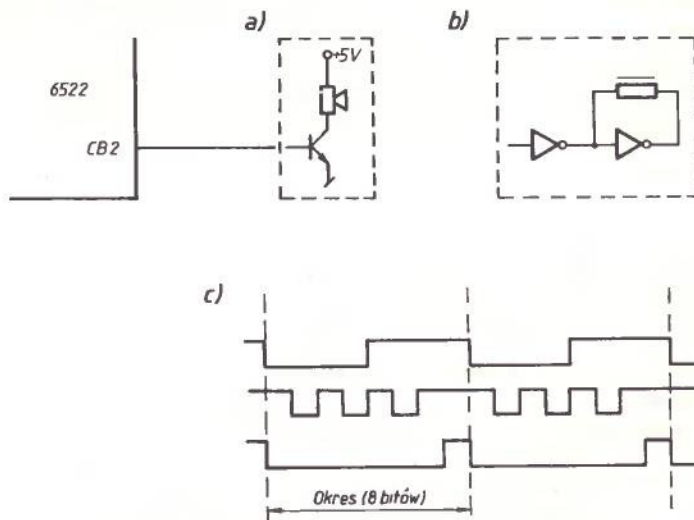


Rys. 6.33. Schemat dołączenia małej klawiatury (24 klawisze) i wyświetlacza 7-segmentowego (sześć cyfr) do mikrokomputera za pomocą układu 6522: a) schemat ideowy; b) przykład możliwości zastosowania wyświetlacza do ograniczonego wyświetlania tekstów; c) zestaw najczęściej używanych znaków

uaktywnienie jednej z kolumn klawiatury. W rozwiązaniu tym korzysta się z faktu, że port A jest zbudowany z układów typu „otwarty dren”. Linie portu A, zaprogramowane jako wyjściowe, służą przy wyświetlaniu cyfr jako wyjścia, zaś przy obsłudze klawiatury jako wejścia. Do rejestru wyjściowego portu A wpisuje się wtedy wartość \$FF, uaktywnia pierwszą kolumnę klawiatury (podając 0 na kolumnę 7 przez wpis liczby 7 na liniach PB0 do PB3), a następnie odczytuje port A. Jeśli był naciśnięty klawisz w tym rzędzie, to wartość odpowiadającego mu bitu jest równa zero i w ten sposób rozpoznaje się naciśnięcie klawisza (analogicznie postępując dla

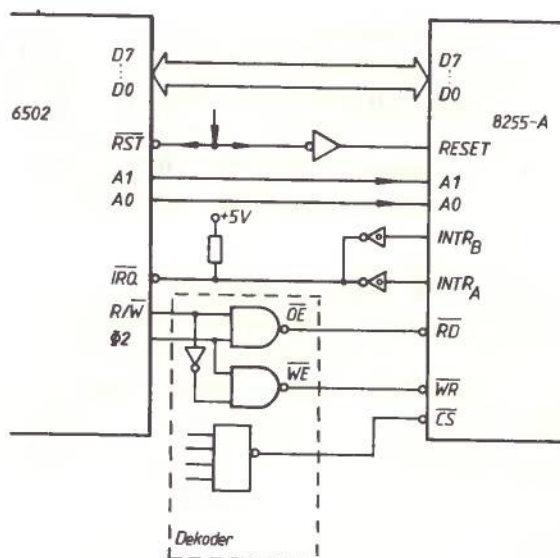
pozostałych kolumn 8 i 9). W tego typu klawiaturze dopuszczalne jest naciśnięcie tylko jednego klawisza, gdyż przy jednoczesnym naciśnięciu dwóch może wystąpić błąd wyświetlania. Program realizujący sterowanie wyświetlaniem oraz odczytu stanu klawiatury jest dość długi i z tego powodu nie jest tutaj przytoczony. Możesz go, Czytelniku, spróbować napisać jako ćwiczenie, oczywiście pod warunkiem, że dobrze opanowałeś pierwszą część książki.

Wytwarzanie dźwięku przy użyciu układu 6522. Układ 6522, jak to opisywaliśmy wcześniej, ma 3 linie, które można zaprogramować tak, by pojawiały się na nich przebiegi czasowe (bardziej lub mniej skomplikowane fale prostokątne) mogące mieć zastosowanie do wytwarzania dźwięku. Pierwszą z nich jest linia *PB7* sterowana przez licznik *T1*, umożliwiającą uzyskiwanie symetrycznego sygnału prostokątnego o częstotliwości od ok. 7 Hz do 250 kHz (dla zegara 1 MHz). Pozostałymi są zaś linie *CB1* i *CB2* sterowane układem rejestru przesuwającego, pracującego w trybie wyjściowym z zapętleniem (ang. *free run*), przy sterowaniu licznikiem *T2*. Szczególnie wygodna do generacji dźwięku jest linia *CB2* (wyjście rejestru przesuwającego), która umożliwia uzyskiwanie przebiegu wyjściowego o częstotliwościach od 243 Hz do 100 kHz (dla zegara 1 MHz), przy czym regulacji może podlegać nie tylko częstotliwość, ale i kształt przebiegu, a więc barwa dźwięku (rys. 6.34). I tutaj napisanie odpowiednich programów pozostawiamy chętnym Czytelnikom.



Rys. 6.34. Generacja dźwięku przy użyciu układu 6522: a) dołączenie głośnika dynamicznego; b) dołączenie przetwornika piezoelektrycznego; c) uzyskiwanie różnych barw dźwięku przez dobór kształtu przebiegu

Inne układy równoległe. Procesor 6502 dość dobrze współpracuje z większością dostępnych na rynku urządzeń peryferyjnych (poza rodziną Z80), a w tym z układami sprzęgów równoległych. Dołączenie układów innych rodzin po-

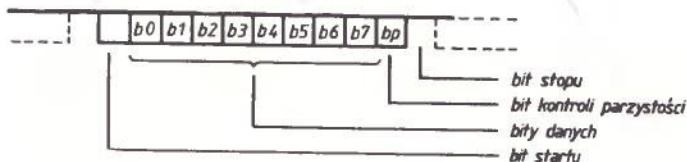


Rys. 6.35. Dołączenie układu Intel 8255-A do procesora 6502

kazujemy na przykładzie najpowszechniejszego w kraju układu 8255 A – rys. 6.35. Do dołączenia tego układu konieczne jest wytworzenie sygnałów $\overline{OE}$ i $\overline{WE}$ (opisywanych w p. 6.1) oraz zanegowanie sygnału RST , a także sygnałów wyjściowych $INTR_A$ i $INTR_B$. Przy dołączaniu wszelkich urządzeń peryferyjnych nie należących do rodziny 65XX należy zachować ostrożność i sprawdzić dokładnie wymagania czasowe i logiczne dołączonego układu (zalecenie to można znacznie złagodzić w przypadku układów peryferyjnych rodziny Motorola 68XX, które to zwykle dołączają się wprost, szczególnie układy 6820 i 6821).

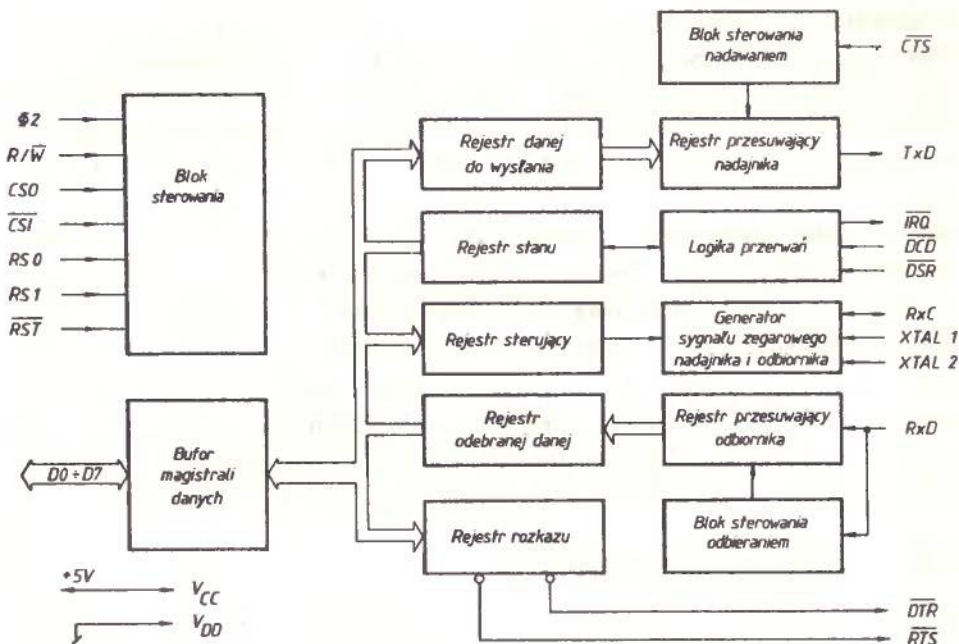
6.3.2. Układ szeregowego wejścia i wyjścia

Spośród dostępnych układów sprzęgu szeregowego najbardziej rozpowszechnione są układy transmisji asynchronicznej (według standardu V24). W tym typie transmisji poszczególne bity słowa są przesyłane kolejno, poczynając od najmniej znaczącego (ang. LSB – *least significant bit*). Każde słowo danych (najczęściej bajt) jest poprzedzane bitem startu (o wartości 0), zakończone zaś bitem stopu (1), często poprzedzonym bitem kontroli parzystości (rys. 6.36).



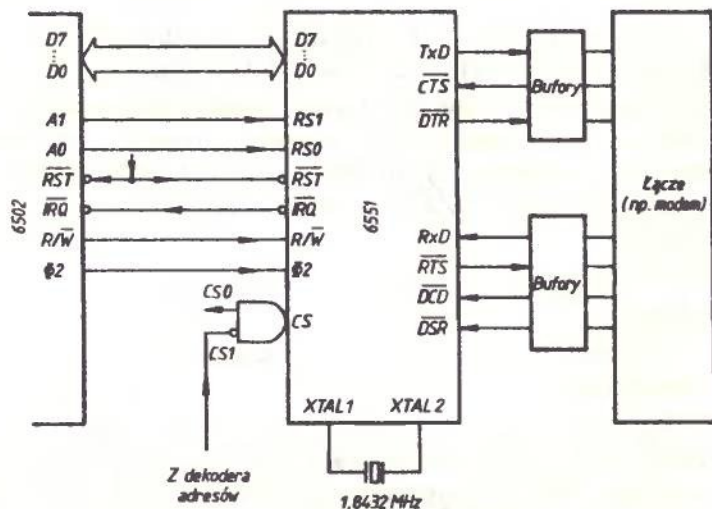
Rys. 6.36. Format danych w asynchronicznej transmisji szeregowej

Transmisję szeregową stosuje się dość często, gdy zachodzi konieczność niezbyt szybkiego przesyłania danych na większe odległości (np. przez linię tele-



Rys. 6.37. Schemat blokowy układu 6551

foniczną). Aby nie obciążać procesora wysyłaniem i odbieraniem poszczególnych bitów, stosuje się scalone układy transmisji szeregowej. W rodzinie 65XX jest układ 6551 (ang. ACIA — *Asynchronous Communication Interface Adapter*) realizujący wszystkie funkcje związane z szeregowym asynchronicznym przesyłaniem danych, w tym niezależne wysyłanie i odbiór danych, kontrola poprawności transmisji (za pomocą bitu kontroli parzystości), sterowanie



Rys. 6.38. Schemat dołączenia układu 6551 do procesora

modemem (urządzeniem umożliwiającym przesyłanie danych przy użyciu sieci telefonicznej) i inne. Schemat blokowy układu 6551 jest przedstawiony na rys. 6.37. Układ ten dołącza się do procesora 6502 w identyczny sposób jak pozostałe układy rodziny (rys. 6.38), zaś do innych urządzeń (modemy, inne komputery itp.) zazwyczaj przez odpowiednie bufory, np. 75150 i 75154.

Opis sygnałów układu 6551. Sygnały $D0 \div D7$, $\overline{IRQ}$, $\overline{RST}$, $\Phi2$ są typowymi sygnałami rodziny, nie wymagają więc dalszego komentarza. Wejścia CS_0 i CS_1 służą do wybierania (zaadresowania) układu: na wejście CS_0 musi być podana jedynka, na CS_1 — zero. Zwykle wejście CS_0 jest dołączone na stałe do napięcia +5 V, zaś CS_1 — do dekodera adresów. Wejścia $RS0$ i $RS1$ służą do zaadresowania odpowiedniego rejestru wewnętrznego układu (tabl. 6.4). Dołącza się je zwykle do linii A0 i A1 procesora. Dłuższego opisu wymagają linie przeznaczone do dołączania sygnałów związanych z transmisją.

Tablica 6.4. Rejestry programowe układu 6551

Adres	Stan linii		Operacja	
	RS1	RS0	Zapis	Odczyt
0	0	0	Rejestr danej do wysłania (<i>transmit data register</i>)	Rejestr danej odebranej (<i>receive data register</i>)
1	0	1	Zerowanie programowe — dane są nieważne (<i>programmed reset</i>)	Rejestr stanu (<i>status register</i>)
2	1	0	Rejestr rozkazów (<i>command register</i>)	
3	1	1	Rejestr sterujący (<i>control register</i>)	

$XTAL1$, $XTAL2$ — do tych linii zwykle dołącza się bezpośrednio rezonator kwarcowy o częstotliwości 1,8432 MHz. Służą one do wewnętrznej generacji przebiegu zegarowego, który po odpowiednim podzieleniu steruje odbiorem i nadawaniem danych. Jeśli zachodzi potrzeba stosowania niestandardowej szybkości transmisji, to można na linię $XTAL1$ podać sygnał prostokątny o częstotliwości 16 razy większej od częstotliwości bitowej (szybkości transmisji).

TxD (ang. *Transmit Data*) — wyjście danych szeregowych — sygnał z nadajnika.

RxD (ang. *Receive Data*) — wejście danych szeregowych — wejście odbiornika.

RxC (ang. *Receive Clock*) — linia ta może być użyta bądź jako wyjście sygnału zegarowego o częstotliwości 16 razy większej od częstotliwości bitowej, bądź jako wejście zegarowe odbiornika, również o częstotliwości 16 razy większej od częstotliwości bitowej. Z linii tej, zaprogramowanej jako wejście,

korzysta się głównie w układach, w których szybkości wysyłania i odbierania danych są różne.

RTS (ang. *Request To Send*) – wyjście służące do sterowania blokiem nadawczym modemu; gdy jest włączony nadajnik układu 6551, linia ta jest w stanie 0.

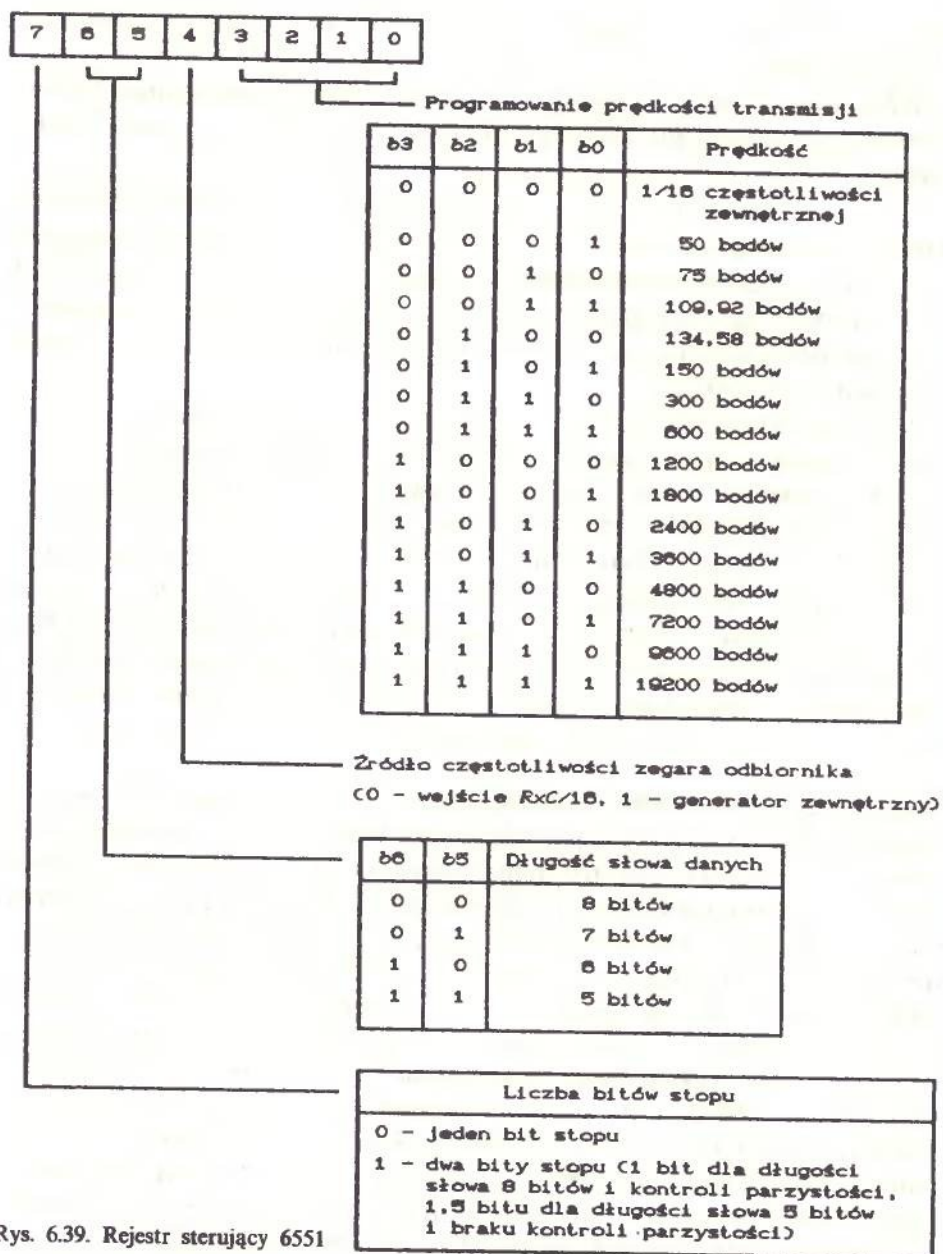
CTS (ang. *Clear To Send*) – wejście służące do sterowania nadajnikiem układu; pojawienie się stanu 1 na tej linii powoduje blokowanie nadajnika.

DTR (ang. *Data Terminal Ready*) – wyjście sygnalizujące stan układu 6551. Stan 0 na tej linii wskazuje, że układ jest aktywny i jest w stanie odbierać i nadawać dane; linia ta jest sterowana (przez negator) zawartością zerowego bitu rejestru sterującego.

DSR (ang. *Data Set Ready*) – wejście służące do odebrania informacji o stanie modemu (zero wskazuje, że modem jest „gotowy”). Stan tej linii nie wpływa na operacje przesłań danych, wpływa tylko na wartość bitu szóstego rejestru stanu i jeśli układ jest aktywny (tzn. bit zerowy rejestru sterującego jest w stanie 1), to każda zmiana stanu tej linii powoduje zgłoszenie przerwania.

DCD (ang. *Data Carrier Detected*) – wejście służące do odebrania informacji o wykryciu przez modem nośnika danych w linii transmisyjnej. Stan tej linii steruje operacją odbierania danych przez układ (stan 1 powoduje zablokowanie odbiornika) oraz zawartością bitu piątego rejestru stanu. Jeśli układ jest aktywny, to każda zmiana stanu linii powoduje przerwanie.

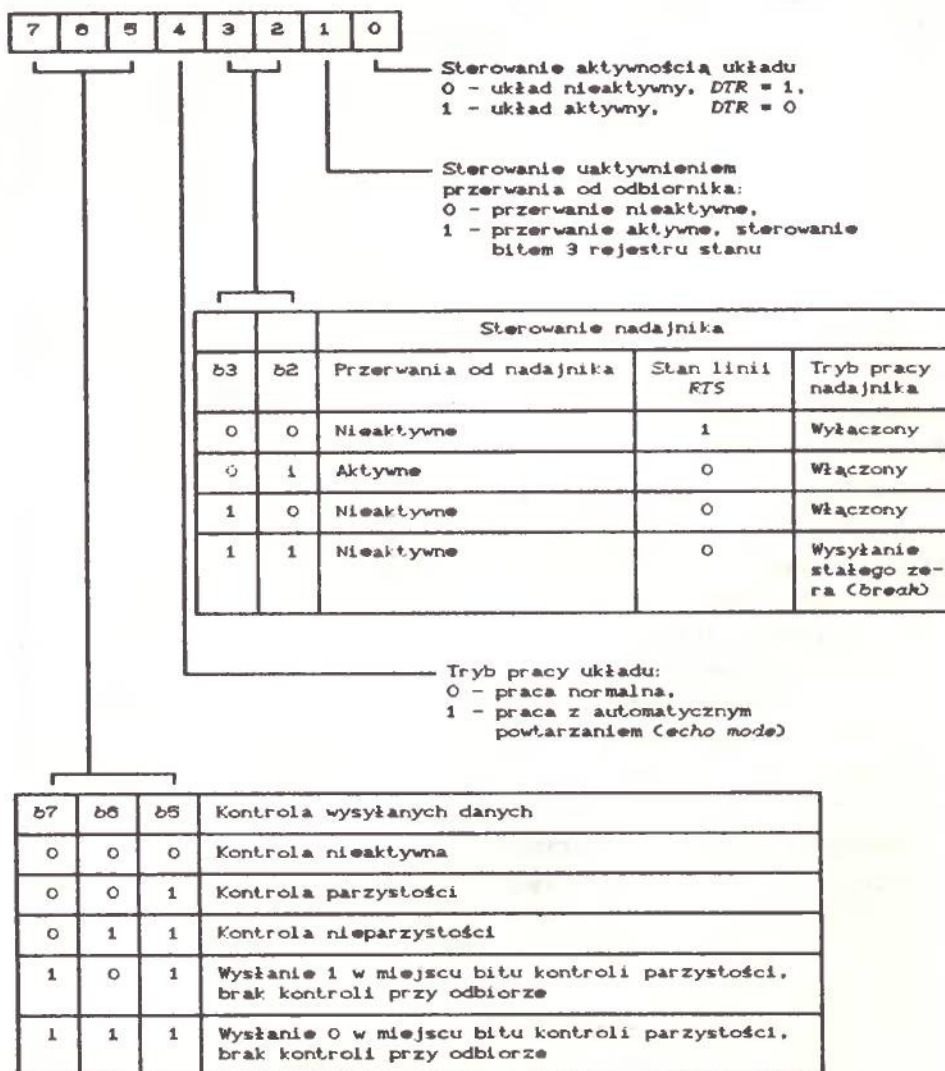
Programowanie układu 6551. Pierwszą czynnością, którą trzeba wykonać przy programowaniu układu 6551, jest wykonanie programowego zerowania przez zapis dowolnej danej do rejestru stanu (rejestru nr 1). Następnie należy określić szybkość transmisji (nadawania i odbioru), długość przesyłanego słowa danych (5, 6, 7 lub 8 bitów) oraz liczbę bitów stopu – przez odpowiednie ustawienie bitów rejestru sterującego (rys. 6.39). Na przykład, aby uzyskać transmisję o szybkości 9600 bodów (bitów na sekundę), długości słowa 8 bitów i jednym bicie stopu, należy do rejestru sterującego wpisać wartość \$0F. Ostatnią czynnością przed rozpoczęciem przesyłania danych jest zaprogramowanie trybu pracy układu, a w tym aktywności (lub nieaktywności) sygnału przerwania pochodzącego od odbiornika, trybu pracy nadajnika i sposobu kontroli przesyłanych danych (kontroli parzystości). Dodatkowo można zaprogramować układ 6551 do pracy w trybie automatycznego powtarzania odebranych danych (ang. *echo mode*). W trybie tym odbiór danej z linii *RxD* powoduje automatyczne jej wysłanie na linię *TxD* (bez udziału procesora). Tryby pracy programuje się ustawiając odpowiednio bity rejestru rozkazów – rys. 6.40. Po zaprogramowaniu układu można przystąpić do normalnej pracy. Wysłanie danych przez układ odbywa się automatycznie po dokonaniu wpisu do rejestru nadajnika (rejestru nr 0). Przed wysłaniem następnej danej należy upewnić się, czy wolny jest już rejestr nadajnika (testując bit rejestru stanu – patrz rys. 6.44). W celu przyspieszenia pracy układu zarówno w nadajniku jak



Rys. 6.39. Rejestr sterujący 6551

i odbiorniku zastosowano podwójne buforowanie, co powoduje, że nową daną można wpisać w trakcie nadawania starej, a więc nadajnik (i odbiornik) nie muszą czekać na procesor. Przy odbiorze danych sytuacja jest bardziej skomplikowana, gdyż poza odczytaniem odebranej danej, co jest możliwe po stwierdzeniu, że dana jest w rejestrze odbiornika (wymaga to testowania bitu 3 rejestru stanu), należy sprawdzić, czy nie nastąpił błąd w odbiorze, testując

znaczniki błędu parzystości (bit 0), błędu bitów stopu (bit 1) i znacznik zgubienia danych (bit 2). Znaczenie poszczególnych bitów rejestru stanu jest pokazane na rys. 6.41. Należy zwrócić uwagę, że układy transmisji szeregowej są układami silnie uzależnionymi czasowo, dlatego też układy tego typu należy obsługiwać z użyciem techniki przerwań (szczególnie część odbiorczą). Do takiej też obsługi jest przystosowany opisywany układ. Uaktywnianie układu (ustawienie bitu 0 w rejestrze rozkazów) powoduje uaktywnianie przerwań pochodzących od zmian stanu linii *DCD*, *DSR* i *CTS*. Przerwania pochodzące od nadajnika i odbiornika należy uaktywniać ustawiając bity *b2* (dla nadajnika) i *b4* (dla odbiornika) w rejestrze rozkazów. Uważny Czytelnik



Rys. 6.40. Rejestr rozkazów

7 6 5 4 3 2 1 0	Wskaźnik	Znaczenie	Kasowanie
	Błąd parzystości przy odbiorze	0 - bezbłędnie 1 - błąd	Automatycznie ¹⁾
	Błąd odbioru bitów stopu	0 - bezbłędnie 1 - błąd	Automatycznie ¹⁾
	Błąd zgubienia odbieranej danej	0 - bezbłędnie 1 - błąd	Automatycznie ¹⁾
	Dana w rejestrze odbiornika	0 - rej. pusty 1 - pełny	Odczyt rejestru danej odbieranej
	Pusty rejestr nadajnika	0 - dane w rej. 1 - pusty	Zapis rejestru danej do wysłania
	DCD ²⁾	0 - $DCD = 0$ 1 - $DCD = 1$	Niekasowalny, pokazuje stan linii
	DSR ²⁾	0 - $DSR = 0$ 1 - $DSR = 1$	Niekasowalny, pokazuje stan linii
	IRQ	0 - brak przerwania 1 - przerwanie z układu 6551	Odczyt rejestru stanu

¹⁾ Znacznik kasowany po odczycie rejestru danej odbiornika i poprawnym przesłaniu następnej danej. Znaczniki te nie wytwarzają sygnału przerwań.

²⁾ Nie jest możliwe wyłączenie przerw od linii wejściowych DCD i DSR bez wyłączenia całego układu 6551.

Rys. 6.41. Rejestr stanu

zauważy, że w rejestrze stanu nie ma bitu informującego o stanie linii wejściowej CTS , jak wobec tego można się dowiedzieć o zmianach stanu tego wyjścia? Niestety, trzeba zgadywać. Producent podaje, jak to zrobić:

1. Odczytać (i zapamiętać) rejestr stanu układu. Odczyt ten automatycznie skasuje znacznik IRQ (bit b7). Wszystkie następne operacje wykonuje się na wartości zapamiętanej.
2. Sprawdzić znacznik IRQ (b7). Jeżeli ma on wartość 0, oznacza to, że źródłem przerwania jest inny układ.
3. Sprawdzić, czy źródłem przerwania jest linia $\overline{DCD}$. W tym celu należy porównać obecną wartość znacznika DCD (b5) z wartością poprzednią, zapamiętaną przez mikrokomputer.
4. Sprawdzić, czy źródłem przerwania jest linia $\overline{DSR}$, porównując obecną wartość znacznika DSR (b6) z wartością poprzednią. Jeśli źródłem przerwania nie są linie $\overline{DCD}$ ani $\overline{DSR}$, to należy przejść do testowania nadajnika i odbiornika.
5. Sprawdzić znacznik odebranych danych (b3). Jeśli jest on w stanie 1, to należy odczytać przesłaną daną, a następnie sprawdzić jej poprawność testując znaczniki błędów parzystości, błędów bitu stopu i błędów zgubienia danej (bity b0, b1, b2).

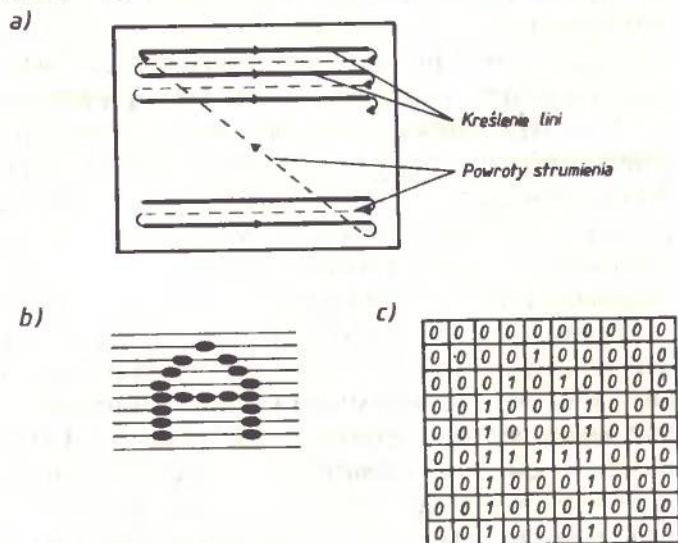
6. Sprawdzić znacznik nadajnika (bit b_4 — pusty rejestr nadajnika). Jeśli jest on w stanie jeden, to można wysłać następną daną, wpisując ją do rejestru nadajnika.
7. Jeśli nie zaszedł żaden z opisanych przypadków, należy wywnioskować, że stan linii $\overline{CTS}$ zmienił się z 0 na 1.

W układzie 6551 nadajnik i odbiornik są niezależne, można więc jednocześnie nadawać i odbierać dane.

Inne układy do transmisji szeregowej. Bardzo często do obsługi transmisji szeregowej jest wykorzystywany układ 6850 (z rodziny Motorola), który jak wszystkie układy rodziny 68XX bardzo łatwo łączy się do procesora 6502 (praktycznie tak samo jak układy rodziny 65XX). Układ ten ma podobne funkcje jak opisywany powyżej, nie zawiera jednak wewnętrznego generatora zegara transmisji ani wewnętrznych dzielników do wyboru szybkości transmisji, dlatego też wymaga do dołączenia większej liczby elementów zewnętrznych. Na wzmiankę zasługują również układy 2661 oraz 8551A, które umożliwiają zarówno stosowanie transmisji synchronicznej, jak i asynchronicznej, ale nie będziemy ich tutaj omawiali.

6.3.3. Moduł wytwarzania sygnału wizyjnego

Bardzo często głównym urządzeniem wyjściowym mikrokomputera jest ekran monitora alfanumerycznego lub graficznego. Monitor taki działa na tej samej zasadzie co zwykły telewizor. Aby na ekranie monitora powstał obraz, należy



Rys. 6.42. Tworzenie obrazu na ekranie monitora: a) sposób wyświetlania na ekranie (kreślenia linii); b) fragment ekranu z wyświetloną literą A; c) fragment pamięci ekranu zawierający dane do wyświetlenia litery A (w trybie graficznym)

wytworzyć (i odpowiednio doprowadzić do monitora) tzw. **zespółony sygnał wizyjny**, który zawiera impulsy synchronizacji poziomej i pionowej oraz informację, które punkty na ekranie mają być rozjaśnione, a które ciemne (albo informację o tym, jakiego koloru ma być każdy punkt).

W tej książce nie będziemy opisywać dokładnie, jak obraz jest tworzony (jest to schematycznie przedstawione na rys. 6.42), ani też dokładnie opisywać scalonych sterowników obrazu wizyjnego (ang. *video controller*). Chcemy natomiast zasygnalizować, jakie występują problemy przy dołączaniu do procesora 6502 układów do generacji obrazu wizyjnego i jak się je rozwiązuje. Aby jednak nie odchodzić za daleko od praktyki, posłużmy się przykładem najprostszego sterownika wizyjnego — układem 6847.

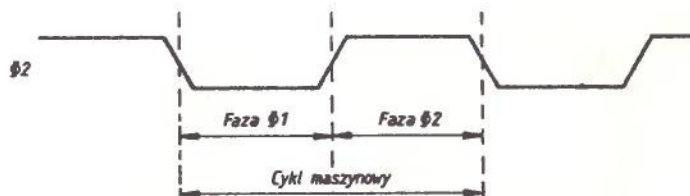
Głównym problemem, który należy rozwiązać przy projektowaniu układów generacji sygnałów wizyjnych, jest problem dwudostępności pamięci wizyjnej. Z pamięci tej korzysta zarówno sterownik wizyjny, odczytując z niej dane o wyświetlanym obrazie (np. kody znaków alfanumerycznych), jak i procesor przygotowujący te dane. Ponieważ jednak w danej chwili z pamięci korzystać może tylko jedno urządzenie, należy więc opracować sposób bezkolizyjnego dostępu do niej.

Możliwości rozwiązań jest wiele, podstawowe są omówione poniżej:

1. Wyższy priorytet sterownika wizyjnego. W tym rozwiązaniu zezwala się procesorowi na dostęp do pamięci wizyjnej tylko w czasie powrotu plamki na ekranie. Jeśli procesor próbuje wykonać operację na pamięci wizyjnej w czasie kreślenia linii, to zostaje on zatrzymany do końca kreślenia bieżącej linii. Rozwiązanie to, dość często stosowane w komputerach z innymi procesorami, jest kłopotliwe w realizacji w przypadku procesora 6502, który może się zatrzymać tylko w cyklach odczytu (*wady tej nie ma wersja 65C02*). Powyższe rozwiązanie powoduje znaczne zmniejszenie efektywnej szybkości procesora w zastosowaniach graficznych, co jest istotną wadą tego rozwiązania.

2. Wyższy priorytet procesora. W tym rozwiązaniu dostęp procesora do pamięci wizyjnej powoduje zablokowanie sterownika wizyjnego na czas dostępu. Rozwiązanie takie jest najprostsze, nie powoduje również zmniejszenia szybkości pracy procesora. Wadą tego rozwiązania jest to, że dostęp procesora do pamięci wizyjnej, realizowany w czasie kreślenia linii, powoduje zakłócenie procesu wyświetlania, tak zwane „śmiecenie na ekranie”.

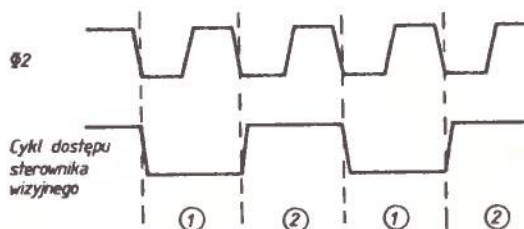
3. Dostęp w przeciwfazie. W tym rozwiązaniu i procesor, i sterownik wizyjny mają cały czas umożliwiony dostęp do pamięci. Jak to jest możliwe? Otóż 6502 wraz ze wszystkimi swoimi peryferiami pracuje synchronicznie w takt zegara $\Phi 2$ (tj. w czasie fazy $\Phi 2$ następuje dostęp procesora do pamięci obrazu — jeśli chce on zmienić treść obrazu). Natomiast w czasie $\Phi 1$ z danych zapisanych w pamięci korzysta sterownik wizyjny — rys. 6.43. Rozwiązanie takie zapewnia całkowicie bezkonfliktowy dostęp do pamięci (procesorowi nie przeszkadza sterownik wizyjny, a sterownikowi — procesor), jednak jest to okupione koniecznością stosowania pamięci dwa razy szybszej niż w przypadku współpracy pamięci tylko z procesorem.



Rys. 6.43. Zasada dostępu w przeciwfazie do pamięci wizyjnej; w fazie $\Phi 1$ dostęp do pamięci ma sterownik wizyjny, w fazie $\Phi 2$ — procesor

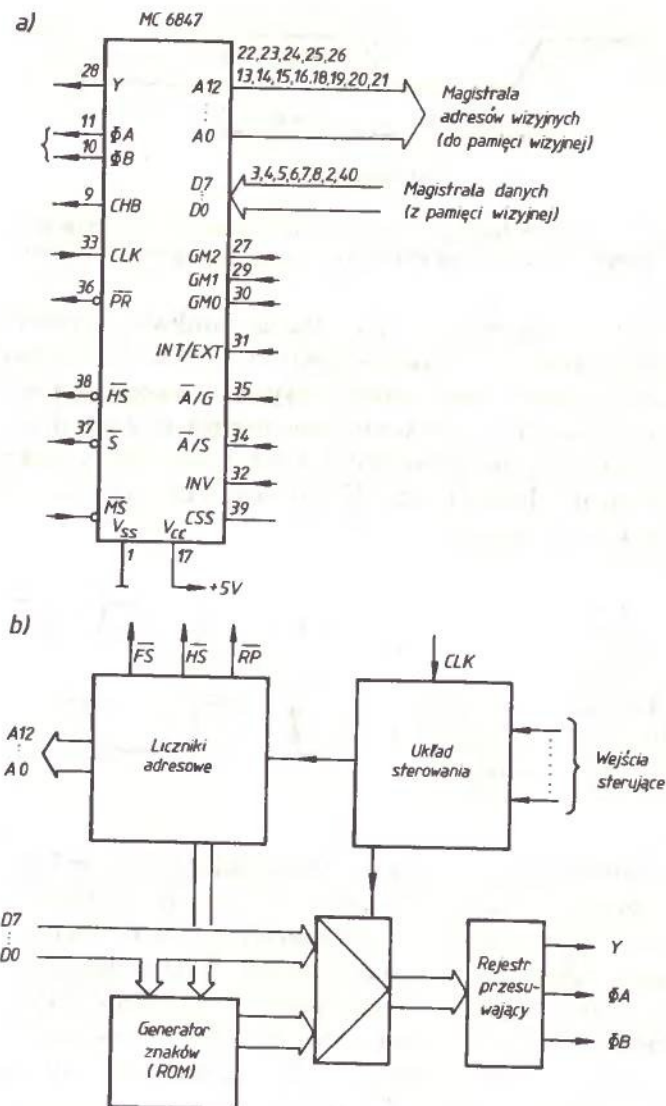
Rozwiązania takie stosuje się w praktyce dla częstotliwości zegara procesora od 1 do 2 MHz. Dla procesorów szybszych zwykle nie można zrealizować synchronizacji procesora i sterownika wizyjnego na częstotliwości podstawowej. Można jednak wtedy zastosować synchronizację z częstotliwością dwukrotnie większą (np. sterownik wizyjny 1,5 MHz, procesor 3 MHz) oraz pewną mieszaną strategię dostępu, nad której realizacją możesz się, Czytelniku, sam zastanowić (rys. 6.44).

Rys. 6.44. Synchronizacja dostępu do pamięci wizyjnej przy stosunku szybkości sterownika wizyjnego i procesora wynoszącym 1:2: 1 — faza dostępu procesora; 2 — faza dostępu sterownika wizyjnego — jeśli w tej fazie procesor żąda dostępu do pamięci wizyjnej, to zostaje zatrzymany na jeden cykl



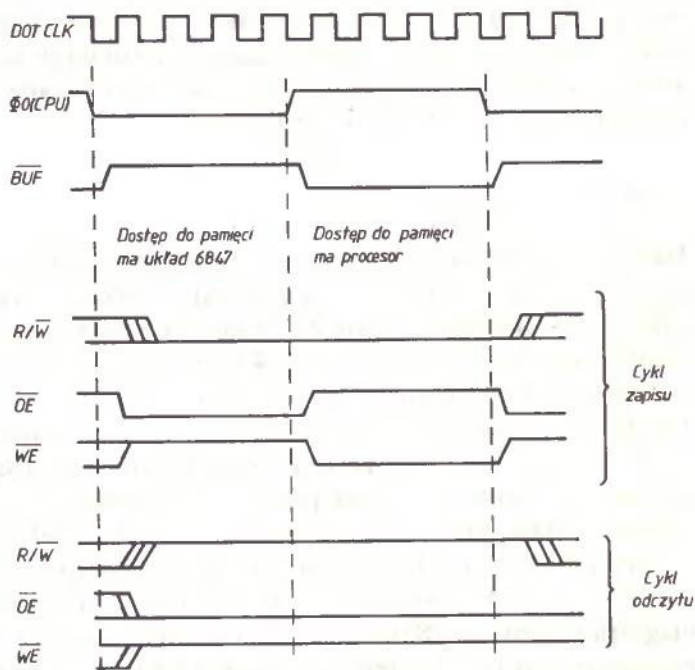
Przed przejściem do omawiania schematu ideowego układu wyświetlania słów kilka o samym sterowniku wizyjnym 6847 — rys. 6.45. Jest on najprostszym układem dostępnym na rynku, spełniającym jednak komplet funkcji sterownika wizyjnego. Zawiera on, poza typowym układem liczników adresów i dekoderek sygnałów synchronizacji, wewnętrzną pamięć stałą generatora znaków (64 podstawowe znaki kodu ASCII), rejestr przesuwający do zamiany danych z postaci równoległej na szeregową oraz układ wytwarzania sygnału chrominancji (sygnału kolorów).

Układ 6847 poza typową pracą w trybie alfanumerycznym (16 wierszy po 32 znaki w matrycy 8×12) umożliwia pracę w wielu trybach graficznych (dwu- lub czterokolorowych) z rozdzielczością do 256×192 punkty — por. tabl. 6.5. Do budowy całego systemu wyświetlania poza sterownikiem MC 6847 jest potrzebna pamięć wizyjna (RWM), bufor danych i adresów oraz modulator wytwarzający zespolony sygnał wizyjny (zwykle układ MC 1372). Przykład układu wyświetlania jest przedstawiony na rys. 6.46. Wszystkie przebiegi czasowe mikrokomputera (a w tym układu wyświetlania) są sterowane sygnałem prostokątnym *DOTCLK* o częstotliwości 3,58 MHz. Wejściowy sygnał zegarowy procesora ($\Phi 0$) otrzymuje się z sygnałem *DOTCLK* po podzieleniu go przez 4. Sygnał $\Phi 2$ procesora steruje dostępem do wspólnej



Rys. 645. Sterownik wizyjny MC 6847: a) rozkład wyprowadzeń; b) architektura wewnętrzna

pamięci według strategii dostępu w przeciwfazie. Jeżeli zaprogramujemy układ 6847 do pracy w trybie alfanumerycznym, podając na wejścia $\overline{INT/EXT}$ i $\overline{A/G}$ stan 0 (patrz rys. 6.45), to do pamięci należy wpisywać kody wyświetlanych znaków. Najbardziej znaczący bit danych, tj. $b7$, służy do rozróżnienia między znakami alfanumerycznymi ($b7 = 0$) i semigraficznymi (grafika o ograniczonej rozdzielczości – $b7 = 1$), zaś bit $b6$, dla znaków alfanumerycznych, przełącza między znakami pozytywowymi ($b6 = 0$) i negatywowymi ($b6 = 1$). Graficzny tryb pracy uzyskuje się przez podanie na wejście $\overline{A/G}$ stanu 1, zaś na wejścia od



Scanned by Gozdek
Classic Computers OnLine
www.cconline.com.pl

Tablica 6.5. Tryby pracy sterownika wizyjnego MC 6847

$\overline{A/G}$	$\overline{A/S}$	$\overline{INT/EXT}$	$\overline{INV}$	GM2	GM1	GM0	Pamięć [KB]	Tryb pracy
0	0	0	0	—	—	—	0,5	Alfanumeryczny wewnętrzny
0	0	0	1	—	—	—	0,5	Alfanumeryczny wewnętrzny negatywowy (z inwersją)
0	0	1	0	—	—	—	0,5	Alfanumeryczny zewnętrzny
0	0	1	1	—	—	—	0,5	Alfanumeryczny zewnętrzny negatywowy
0	1	0	—	—	—	—	0,5	Semigrafika 4
0	1	1	—	—	—	—	0,5	Semigrafika 6
1	—	—	—	0	0	0	1	Grafika kolorowa 64 × 64
1	—	—	—	0	0	1	1	Grafika 128 × 64
1	—	—	—	0	1	0	2	Grafika kolorowa 128 × 64
1	—	—	—	0	1	1	1,5	Grafika 128 × 96
1	—	—	—	1	0	0	3	Grafika kolorowa 128 × 96
1	—	—	—	1	0	1	3	Grafika 128 × 196
1	—	—	—	1	1	0	6	Grafika kolorowa 128 × 196
1	—	—	—	1	1	1	6	Grafika 256 × 192

GM0 do GM2 kodu trybu graficznego (wg rys. 6.45). W tym trybie pracy w pamięci umieszcza się dane o stanie poszczególnych punktów ekranu. W przypadku trybów graficznych „czarno-białych” każdy bit pamięci odpowiada jednemu punktowi na ekranie, a jedynka w pamięci odpowiada świeceniu punktu. Dla trybów „kolorowych” o kolorze punktu decydują dwa sąsiednie bity pamięci, wybierając jeden z czterech możliwych kolorów. Dokładniejszą analizę schematu z rys. 6.46 (a zwłaszcza analizę przepływu danych) pozostawiamy dociekliwym Czytelnikom.

6.3.4. Inne układy peryferyjne

Jak już wspominaliśmy, nie ma większych trudności z dołączaniem do procesora 6502 (i innych z tej rodziny) różnych układów peryferyjnych, z wyjątkiem układów rodziny Z80. I tak, na przykład do współpracy z dyskami elastycznymi często używa się układu 1771, jako przetwornika analogowo-cyfrowego, dołączanego wprost do magistral procesora — układu AD574, zaś do sterowania sprzęgiem IEC 625 (GPIB) — układu TMS9914.

Jedynie przy próbie zastosowania układu bezpośredniego dostępu do pamięci (lub ogólniej — przy pracy wieloprocesorowej) napotyka się większe trudności, gdyż nie do tego procesor ten był projektowany.

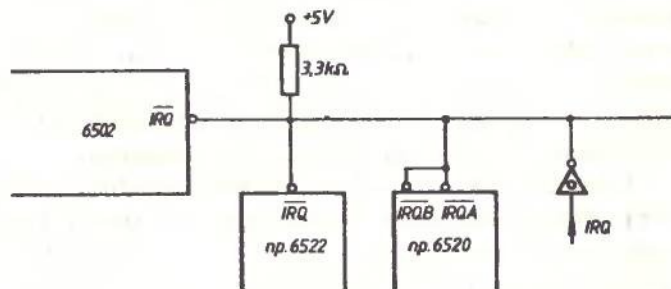
Procesor 6502 nie ma trójstanowych buforów ani na magistrali adresowej, ani na sygnałach sterujących. Nie ma też możliwości sterowania buforami magistrali danych. Dlatego też, chcąc zastosować go w układach wieloprocesorowych (np. łącznie z układem DMA — ang. *direct memory access*), należy zastosować trójstanowe buforowanie linii adresowych i sterowań, a także samemu zapewnić (za pomocą sygnału RDY) odpowiednią sekwencję

zatrzymań i ponownych uruchomień procesora. Ze względu na możliwość odczytu peryferii w trakcie przełączania między procesorem a układem DMA zalecane jest zatrzymywanie procesora w cyklu pobierania kodu rozkazu lub w cyklu bezpośrednio po nim następującym.

6.4. Obsługa przerwań

W pierwszej części książki zostały omówione sygnały przerwań i ich działanie. Obecnie słów kilka powiemy o programach i rozwiązaniach sprzętowych obsługi przerwań.

Procesor ma dwa wejścia przerwań: niemaskowalne *NMI* i maskowalne *IRQ*. Przerwanie niemaskowalne jest czułe na zbocze, dlatego też w praktyce stosuje się je do obsługi tylko jednego zdarzenia o największym priorytecie. Może to być sygnał zaniku napięcia zasilania, może być sygnał odświeżania pamięci dynamicznej, zaś w systemach uruchomieniowych zwykle jest to sygnał pochodzący od klawisza *NMI*, wykorzystywanego do podglądania stanu systemu. Wszystkie pozostałe sygnały przerwań są podawane na wejście *IRQ*, zwykle w sposób przedstawiony na rys. 6.47 (iloczyn montażowy sygnałów *IRQ* od poszczególnych urządzeń, czyli suma sygnałów *IRQ*).



Rys. 6.47. Sposób dołączania układów peryferyjnych do linii *IRQ*

Rozwiązanie to jest najprostsze sprzętowo, lecz nieco bardziej skomplikowane programowo, gdyż w poniższym programie obsługi przerwań trzeba sprawdzić, które z dołączonych urządzeń jest źródłem przerwania, co zwykle jest dość czasochłonne.

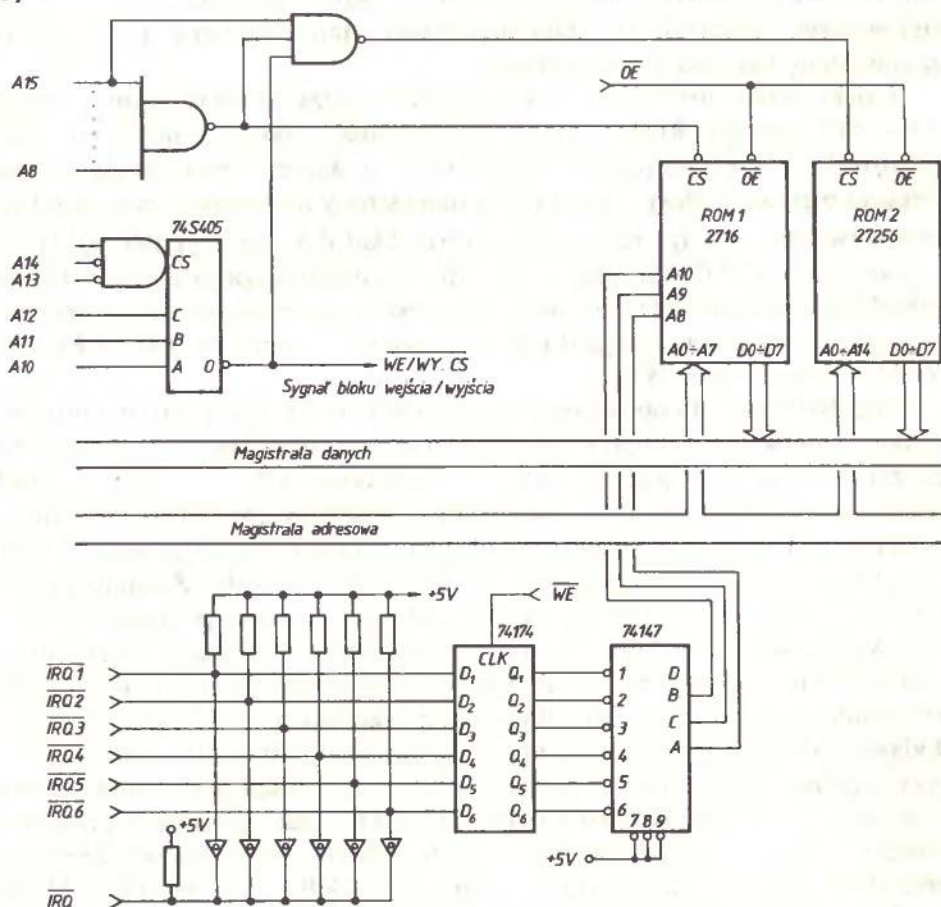
IRQ	PHA		; przesłanie zawartości rejestrów na
	TYA		; stos
	PHA		
	TXA		
	PHA		
	LDA	U1	; testowanie pierwszego urządzenia
	BPL	IRQ.1	
	:	}	; program obsługi urządzenia 1
	:		

IRQ.1	LDA	U2	; testowanie drugiego urządzenia
	BPL	IRQ.2	
	:	}	; program obsługi urządzenia 2
	:		
IRQ.2	:	}	; testy i programy obsługi pozostałych urządzeń
	:		
	:		
IRQ.EXIT	PLA		; zakończenie procedury
	TAX		; obsługi przerwań
	PLA		; odtworzenie rejestrów
	TAY		
	PLA		
	RTI		; wyjście z programu obsługi przerwania

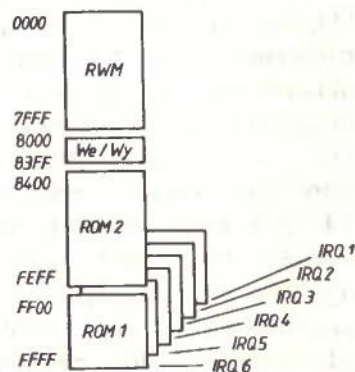
Rozwiązaniem szybszym jest tzw. **wektoryzowanie przerw** — technika polegająca na dostarczaniu procesorowi gotowego wektora obsługi przerwania pochodzącego od aktualnie zgłoszonego urządzenia, tzn. 2 bajtów adresu początku procedury obsługi aktualnego przerwania. Polega to na tym, że w wydzielonym obszarze przestrzeni adresowej znajduje się przełączana pamięć ROM, sterowana sygnałami przerw pochodzących od poszczególnych urządzeń. Przełączenie następuje automatycznie po zgłoszeniu przerwania o priorytecie wyższym od obecnie obsługiwanego.

Rysunek 6.48 przedstawia przykład układu zapewniającego wektoryzowanie przerw. Przestrzeń adresowa bloku pamięci stałej została podzielona na dwie części. Część (ROM2) o niższych numerach komórek jest częścią stałą, natomiast część o wyższych numerach — jedna strona o adresach od \$FFF0 do \$FFFF (ROM1) jest przełączana przez sygnały przerw. Zgłoszenia przerw $IRQ1 \div IRQ6$ pochodzące od poszczególnych urządzeń peryferyjnych są podawane na koder priorytetu (np. układ 74147), którego wyjście (numer przerwania o najwyższym priorytecie) steruje trzema wejściami adresowymi pamięci ROM1, wybiera więc jedną z ośmiu stron tej pamięci (nie wszystkie muszą być wykorzystane). Strony te różnią się między sobą tylko wartością wektora przerw IRQ , więc obsłużone będzie tylko to przerwanie, którego wektor odczytał procesor. Rozwiązanie to może wydać się rozrzutne — dajemy w układzie pamięć ROM1 o pojemności 2 KB, z czego tak naprawdę wykorzystujemy tylko 16 bajtów — jest jednak rozwiązaniem najtańszym. Ponieważ stan linii $IRQ1 \div IRQ6$ może zmieniać się w trakcie cyklu zegara, może zmieniać się też między dwoma cyklami pobrania wektora przerwania, należy więc zastosować pewien mechanizm zapewniający poprawną pracę pamięci ROM1. Jeśli przyjrzymy się wykonywanej przez procesor sekwencji przejścia do obsługi przerw (schemat HD1), to

a)



b)



Rys. 6.48. Układ wektoryzowania przerwań:
a) schemat ideowy; b) mapa pamięci

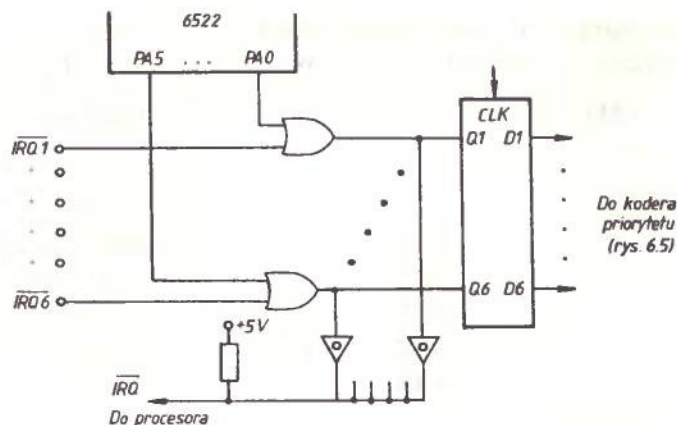
zauważymy, że po zaakceptowaniu przerwania procesor dokonuje najpierw zapisu na stos zawartości licznika rozkazów i rejestru stanu, a następnie odczytuje dwa bajty wektora obsługi przerwania *IRQ*. Najprostszym mechanizmem zapewniającym poprawne działanie mechanizmu wektoryzowa-

nia jest więc zastosowanie rejestru złożonego z przerzutników typu D, sterowanych zboczem, do zapamiętywania stanu zgłoszeń przerwań po zakończeniu każdego cyklu zapisu.

Dzięki temu upiekliliśmy dwie pieczenie przy jednym ogniu: adresy $A8 \div A10$ pamięci ROM1 mogą się zmieniać tylko na początku cyklu zegarowego i nie zaistnieje nigdy sytuacja, w której mniej znaczącą część wektora przerwań odczytywałoby się z innej strony niż bardziej znaczącą. Przy okazji zwrócimy uwagę na przedstawiony układ dekodera sygnału *ROM CS* (wyboru pamięci ROM2). Sygnał ten byłby po prostu negacją adresu *A15*, aby jednak nie wystąpiło zderzenie danych w obszarze wejścia-wyjścia i w obszarze ostatniej strony pamięci, sygnał ten jest blokowany sygnałami *I/O* i *VECTOR ROM CS* (w bramce B2).

Przejdźmy teraz do oprogramowania. Na stronie 109 przedstawiono program obsługi przerwania z programowym rozpoznawaniem źródła przerwań. Na początku programu jest 5 rozkazów (wykonujących się w 13 cyklach zegarowych) służących do zapamiętania rejestrów procesora na stosie, następnie jest część rozpoznawania i obsługi przerwań, na końcu zaś jest tylko 5 rozkazów odtwarzających zawartość rejestrów (16 cykli). Zapamiętywanie i odtwarzanie rejestrów umożliwia swobodne ich użycie w programie obsługi przerwań (a więc przerwanie nie niszczy obliczeń przerwanego programu – i tak powinno być zawsze) jednak wymaga dużo czasu (łącznie z wejściem do programu obsługi przerwania i wyjściem z przerwania – 42 cykle). Do tego dodajmy czas rozpoznawania źródła przerwań (7 do 20 cykli) i zauważmy, że przy większej liczbie urządzeń peryferyjnych czasy obsługi przerwania stają się dość duże (np. dla 10 urządzeń około 112 cykli + czas wykonania programu obsługi), co ma znaczny wpływ na ograniczenie maksymalnej zdolności przesyłania danych z użyciem przerwań ($5 \div 10$ KB/s przy zegarze 1 MHz).

Zastosowanie wektoryzowania przerwań sprawę znacznie poprawia. Odpada zupełnie czas rozpoznania źródła przerwania (chyba, że trzeba rozpoznać jedno z możliwych źródeł przerwań występujących w jednym układzie, np. 6522), poprawić też można czas związany z wejściem do programu obsługi przerwania i wyjściem z niego. Nie wszystkie programy obsługi przerwań używają wszystkich rejestrów procesora. Rezygnacja z jednego rejestru (np. z rejestru *X*) zmniejsza czas wejścia i wyjścia z przerwania z 42 do 31 cykli, zaś rezygnacja z dwóch rejestrów (*X* i *Y*) – do 20 cykli. Mogą również istnieć (dzięki instrukcji *BIT*) programy nie korzystające z żadnych rejestrów wewnętrznych poza rejestrem stanu, który jest automatycznie zapamiętywany i odtwarzany. Opisane rozwiązanie umożliwia dość sprawne odnalezienie źródła przerwań i w dodatku przypisanie poszczególnym urządzeniom priorytetu. Omówiony mechanizm działa jednak tylko przy wchodzeniu do obsługi przerwań. Co należy jednak zrobić, by urządzenie o wyższym priorytecie mogło przerwać program obsługi przerwania o niższym priorytecie? Jak w większości przypadków, istnieją tu dwa rozwiązania – sprzętowe i programowe.



Rys. 6.49. Modyfikacja układu wektoryzowania przerwań

Zacznijmy od programowego. Przy wejściu do programu obsługi przerwania procesor automatycznie ustawia znacznik *I* w stan 1 (maskuje przerwanie). Obsługując więc przerwanie o niskim priorytecie, należy najszybciej jak to możliwe zezwolić na kolejne przerwanie – przez rozkaz CLI. Można go jednak wykonać dopiero po zaawansowaniu obsługi przerwania do tego stopnia, że nastąpi skasowanie żądania przerwania obsługiwanego urządzenia (np. po odczytaniu danych z portu układu 6522 pracującego w trybie automatycznego potwierdzania danych). Należy również w jakiś sposób zabezpieczyć się przed błędami spowodowanymi przyjściem następnego przerwania z tego samego źródła przed ukończeniem obsługi poprzedniego przerwania, co może na przykład zmienić kolejność odebranych danych.

Rozwiązanie sprzętowe jest trochę bardziej skomplikowane. Ponieważ procesor nie ma linii potwierdzenia przyjęcia zgłoszenia przerwania, więc w rozwiązaniu całkowicie sprzętowym należałoby rozpoznać chwilę wejścia w procedurę obsługi przerwania (wykryć odczyt spod adresu \$FFFF lub \$FFFE) i wyjścia z procedury (rozkaz RTI) oraz zbudować bardzo skomplikowany układ zapamiętujący priorytety kolejnych przerwań i zezwalający na przerwanie o wyższym priorytecie. Jednak w takim przypadku wygodniej jest chyba użyć procesora 68000, który ma wbudowany blok priorytetowej obsługi przerwań. Pewnym kompromisem między efektywnością a komplikacją układu jest rozwiązanie przedstawione na rys. 6.49, będące modyfikacją poprzedniego układu. Cechą różniącą go od rozwiązania poprzedniego jest to, że sygnały zgłoszenia poszczególnych przerwań są podawane przez bramki sterowane liniami peryferyjnymi układu 6522, co umożliwia blokowanie pewnych przerwań. Po wejściu do programu obsługi przerwania i po zapamiętaniu koniecznych rejestrów – zapamiętuje się na stosie informacje, które przerwania są aktualnie zablokowane, po czym blokuje się przyjęte przerwanie i wszystkie przerwania o niższym priorytecie (stałą *MASK*, która ma jedynki w miejscach blokowanych przerwań) i wykonuje rozkaz CLI, umożliwiając tym przyjęcie przerwań o wyższym priorytecie. Po zakończeniu obsługi przerwania

odtwarza się stan rejestru blokującego przerwania oraz stan rejestrów procesora. Program obsługi przerwania ma następującą postać:

IRQ	PHA		; zapamiętanie akumulatora na stosie
	:	}	; zapamiętanie innych rejestrów, jeśli konieczne
	:		
	:		
	LDA	PA	; zapamiętanie aktualnego stanu
	PHA		; maskowania przerwań
	ORA	# MASK	; ustawienie nowego stanu maskowa-
	STA	PA	; nia przerwań
	CLI		; odblokowanie przerwania
	:	}	; program obsługi przerwania
	:		
	:		
	PLA		; odtworzenie początkowego stanu
	SEI		; maskowania przerwań
	STA	PA	
	:	}	; odtworzenie zawartości rejestrów procesora
	:		
	:		
	PLA		
	RTI		

Rozwiązanie zwiększa długość programu obsługi przerwania o niskim priorytecie o 25 cykli, więc należy je stosować tylko w tych przypadkach, gdy długość programu obsługi jest wyraźnie większa. Procedury obsługi przerwań o najwyższych priorytetach należy stosować nie zmienione (tzn. bez odblokowywania przerwań).

Podsumowując — należy stwierdzić, że obsługa przerwań jest zwykle sprawą bardzo istotną, a nawet krytyczną. Dlatego też przed przystąpieniem do projektowania mikrokomputera należy dokładnie przemyśleć wymagania stawiane przed nim i stosownie do nich zaprojektować schemat układu i program obsługi przerwań.

6.5. Buforowanie

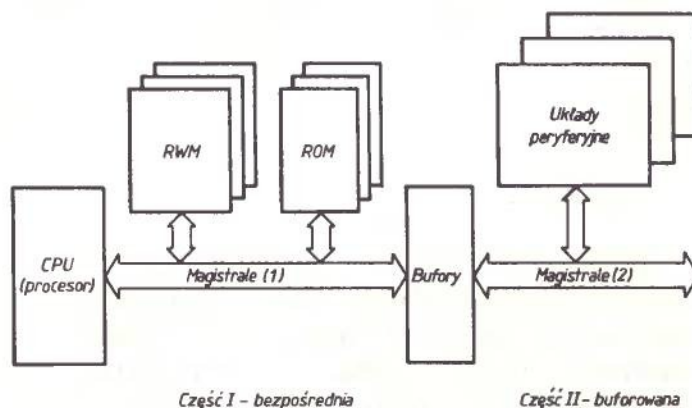
W poprzednich punktach przedstawiliśmy sposoby dołączania różnych elementów mikrokomputera do mikroprocesora. Rozważaliśmy jednak tylko dwa aspekty tej sprawy: prawidłowość dołączenia pod względem logicznym i odpowiedniość pod względem spełnienia wymagań czasowych. Obecnie zajmujemy się omówieniem wymagań elektrycznych.

Wszystkie sygnały występujące w rodzinie 65XX są elektrycznie zgodne ze standardem sygnałów TTL, co oznacza, że stanowi logicznemu 0 odpowiada napięcie od 0 do 0,8 V, zaś stanowi 1 — od 2,4 do 5 V. Inne są natomiast

wartości prądów wejściowych i wyjściowych. Większość wejść, a w tym wejścia adresowe, danych, $R/\overline{W}$, $\Phi 2$, $R\overline{ST}$, stanowią wejścia o dużej impedancji, praktycznie nie pobierające prądu (prądy wejściowe są mniejsze od $2,5 \mu A$), o pojemności wejściowej od 10 do 15 pF. Wyjścia układów mają niestety ograniczony prąd wyjściowy do około 1,6 mA, co powoduje, że wyjścia te mogą być obciążone tylko jednym układem TTL. Jeżeli zamiast serii podstawowej TTL używa się układów serii LS, to ze względu na mniejszy prąd wejściowy tych układów można dołączyć je cztery. Istnieje również ograniczenie liczby możliwych do dołączenia układów o wejściach wysokoimpedancyjnych – choć tym razem nie jest to ograniczenie wynikające bezpośrednio z bilansu prądów.

Aby każdy z układów mikrokomputera mógł pracować z pełną szybkością, pojemność obciążająca wyjście układu nie może być większa od 130 pF, co odpowiada pojemności około dziesięciu wejść połączonych równolegle. Przekroczenie wartości 130 pF nie uniemożliwia pracy układu, lecz pociąga za sobą konieczność zmniejszenia szybkości mikrokomputera (przez zmniejszenie częstotliwości zegara) ze względu na wydłużenie czasu ładowania pojemności wejściowych. Sytuacji takiej należy jednak unikać.

Jeżeli okaże się, że należy do procesora dołączyć więcej niż 10 układów (np. dużą liczbę urządzeń peryferyjnych), to należy zastosować technikę **buforowania**. Polega ona na wzmocnieniu sygnałów za pomocą bramek (buforów) o większych wydajnościach prądowych. Do tego celu stosuje się zwykle odpowiednie bramki serii TTL lub TTL-LS. Jakże zastosować bramki buforujące i w którym miejscu mikrokomputera je włączyć – zależy to od konkretnego przypadku. Postaramy się tu jednak przedstawić najbardziej typowe metody. Zaczniemy od problemu, gdzie włączać bufory. Jeżeli konstrukcja mikrokomputera jest jednopłytkowa, oparta głównie na układach typu MOS-LSI, a konieczność buforowania wynika z przekroczenia dopuszczalnej pojemności obciążającej linie (co oznacza, że należy zastosować więcej niż 10 układów), to dobre rezultaty daje podział mikrokomputera na

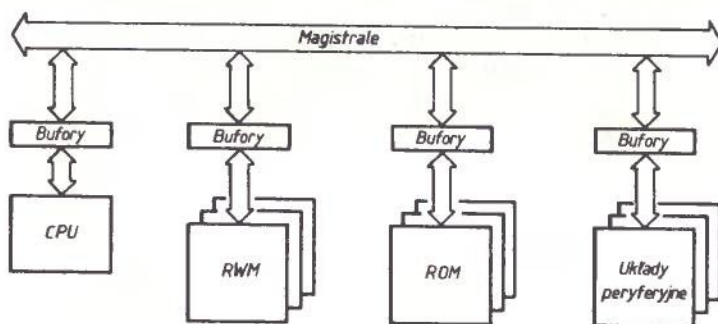


Rys. 6.50. Buforowanie przy podziale komputera na dwie części

dwie części, o ile układów nie jest więcej niż 20 (co zwykle jest spełnione). Ideę takiego podziału przedstawia rys. 6.50. Ważną rzeczą jest, w jaki sposób dokonuje się podziału. Należy zwrócić uwagę, że buforowanie zwiększa czas propagacji sygnałów co najmniej o podwójny czas propagacji przez bufor, co zwykle wynosi od 20 do 60 ns. Dokonując więc podziału, należy w części bezpośredniej, tuż przy procesorze, umieszczać układy wolniejsze, o większym czasie dostępu. W części buforowanej należy umieszczać szybsze. Zwykle są to urządzenia wejścia-wyjścia.

Podobne rozwiązanie można zastosować, gdy buforowanie wynika nie z liczby układów MOS, lecz z przekroczenia obciążalności w wyniku dołączenia dużej liczby układów TTL. W tym przypadku zasady podziału są jeszcze prostsze. W części bezpośredniej umieszcza się wszystkie układy MOS, w części buforowanej wszystkie układy TTL.

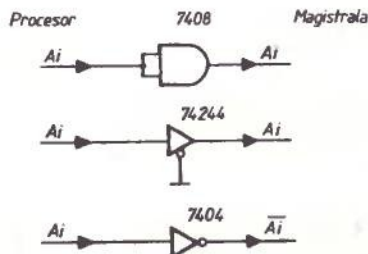
Jeżeli liczba układów dołączonych do procesora jest jeszcze większa (co zdarza się bardzo rzadko) lub gdy mikrokomputer ma mieć budowę modułową, to stosuje się odmienną filozofię buforowania. Mikrokomputer organizuje się wtedy wokół centralnej magistrali o dużej obciążalności, do której dołącza się wiele modułów, z których każdy jest wyposażony w zestawy buforów (rys. 6.51). Rozwiązanie takie umożliwia budowę mikrokomputera o dużej liczbie elementów, ma jednak dość dużą wadę — w każdej operacji zapisu lub odczytu biorą udział co najmniej 2 zestawy buforów, a więc sygnały propagują się przez cztery (dwa na adresach, dwa na danych) lub więcej buforów, co wprowadza opóźnienie od 40 do 120 ns. Rozwiązanie takie można więc stosować tylko przy niezbyt dużych szybkościach mikrokomputera — praktycznie do częstotliwości zegara równej 2 MHz.



Rys. 6.51. Buforowanie w systemie modułowym

Po omówieniu „filozofii” buforowania możemy przejść do szczegółów — które linie należy buforować, jakie bufor stosować i jak nimi sterować. Do linii jednokierunkowych, takich jak linie adresowe oraz linie sterujące $R/\overline{W}$ i $\Phi 2$ można zastosować dowolne bramki (rys. 6.52). Zwykle stosuje się bramki nie wnoszące negacji (np. układy 7408), lecz jest to bardziej kwestia przyzwyczajenia niż konieczność. Jeśli, na przykład, zastosować do buforowania

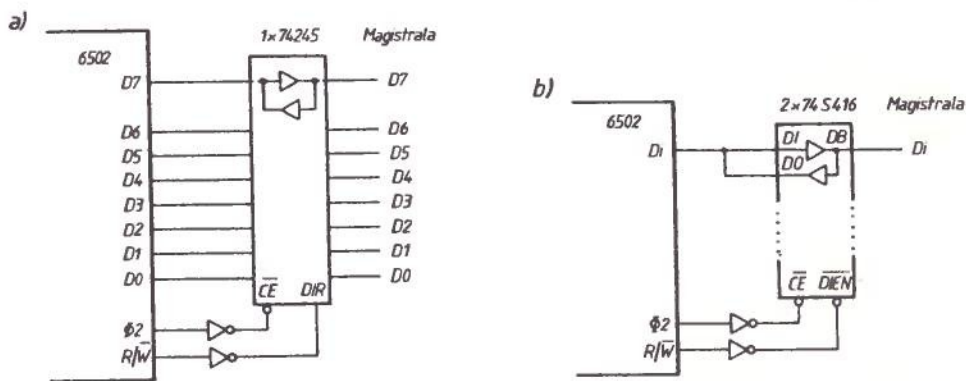
linii adresowych bramki negujące (np. układy 7404), to jedyną konsekwencją tego faktu będzie konieczność uwzględnienia tego przy pisaniu oprogramowania (odwrotna kolejność rejestrów urządzeń peryferyjnych), programowaniu pamięci ROM (należy programować ją wtedy „od końca”) i projektowaniu dekodera adresów. Dwukierunkowe linie danych należy buforować za pomocą bramek trójstanowych, takich jak 74LS244, 74LS245, 74S416, 74125, 74S482 lub podobnych. Bufory te należy sterować tak, by były otwarte tylko w fazie $\Phi 2$ (i to w odpowiednim kierunku), co zapobiega zderzeniom danych.



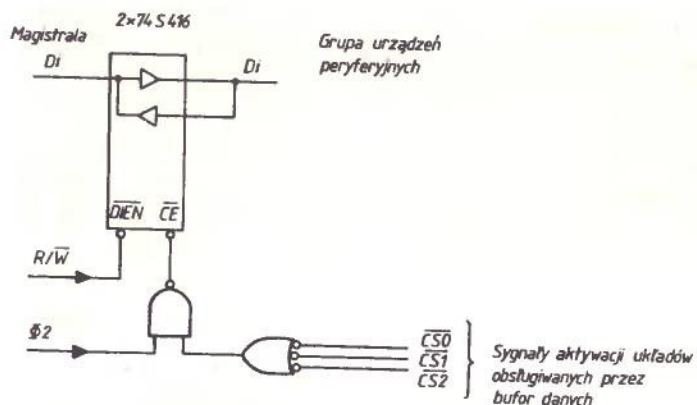
Rys. 6.52. Buforowanie linii adresowych

Przykład dołączenia buforów do linii danych procesora jest przedstawiony na rys. 6.53. Wejście *CE* (ang. *chip enable* — uaktywnienie układu) jest sterowane zanegowanym sygnałem $\Phi 2$ i zezwala na otwarcie się buforów tylko przy $\Phi 2 = 1$. O kierunku pracy buforów decyduje zanegowany sygnał $R/\bar{W}$, powodując, że w cyklach zapisu ($R/\bar{W} = 1$) otwierają się bufory w kierunku od procesora do magistrali, zaś w cyklach odczytu ($R/\bar{W} = 0$) w kierunku przeciwnym — do procesora.

W przypadku buforów obsługujących pamięć lub układy wejścia-wyjścia bardziej skomplikowane staje się sterowanie sygnałem *CE*, gdyż sygnał ten może pojawiać się tylko w cyklach dostępu do zasobów obsługiwanych przez dany bufor. Zrealizować to można w sposób przedstawiony na rys. 6.54.



Rys. 6.53. Buforowanie linii danych: a) przy użyciu układów 74245; b) przy użyciu układów 74S416



Rys. 6.54. Układ sterowania buforem danych obsługującym urządzenia peryferyjne lub pamięć

Bramka B2 realizuje sumę logiczną sygnałów wyboru układów, które wchodzi w skład buforowanego bloku. Suma ta, po wymnożeniu logicznym z sygnałem $\Phi 2$ i zanegowaniu, steruje otwarciem bramek trójstanowych bufora (w którą stronę — to zależy oczywiście od sygnału $R/\overline{W}$).

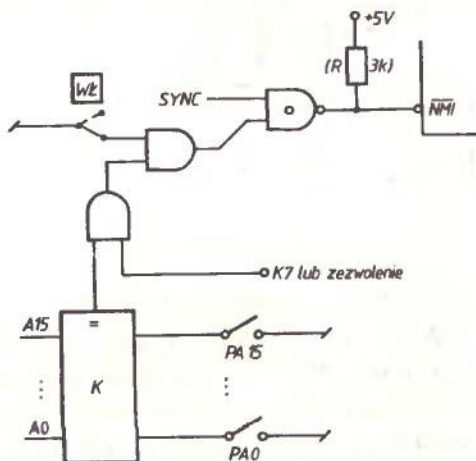
Wskazówki dotyczące uruchamiania i diagnozowania mikrokomputera

Nie sposób podać przepisu na wykrycie wszystkich uszkodzeń, które mogą wystąpić w mikrokomputerze. Podamy więc tutaj tylko niektóre wskazówki i trzy bardzo pomocne sposoby testowania. Ogólną wskazówką jest to, żeby rozumieć dokładnie, jak system działa, i analizować, czy jego fragmenty pracują poprawnie.

Kolejność testowania powinna być w zasadzie taka jak punkty rozdz. 4. Jeśli nie znajdziemy błędu, a mikrokomputer pomimo to nie działa, to należy zastosować poniższe metody.

7.1. Pułapka

Jeśli pułapki programowe gotowego mikrokomputera są niewystarczające (albo ich nie ma), to można skonstruować pułapkę sprzętową według rys. 7.1.

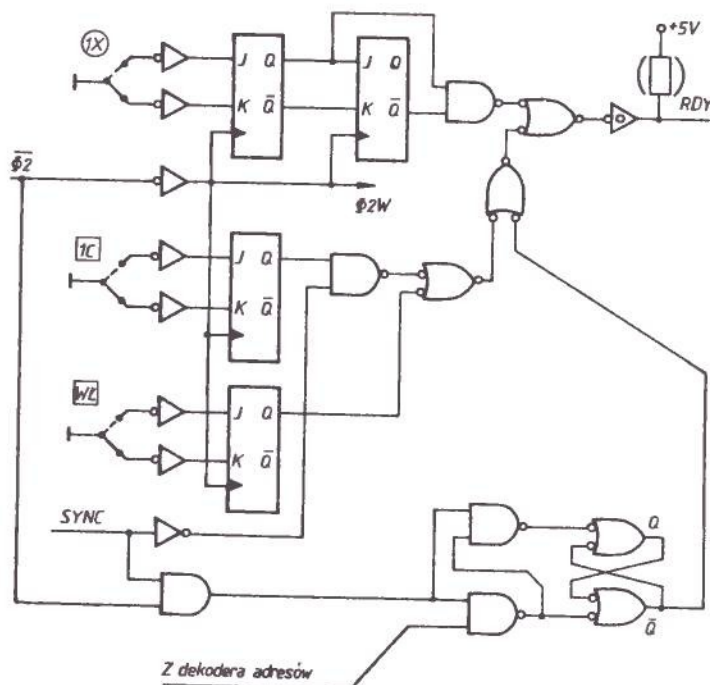


Rys. 7.1. Schemat pułapki sprzętowej

Po naciśnięciu przełącznika WŁ (stabilnego) najbliższy cykl, w którym mikroprocesor pobiera kod rozkazu spod adresu identycznego z nastawionym na przełącznikach PA0 ... PA15, spowoduje wytworzenie sygnału NMI, który z kolei spowoduje przerwanie (po zakończeniu bieżącego rozkazu). Wektor NMI (tj. zawartość komórek \$FFFA i \$FFFB) powinien zawierać adres początku programu obsługi przerwania, który by przesyłał zawartość wszystkich rejestrów mikroprocesora do pamięci. Te problemy pozostawiamy pomysłowości Czytelnika.

7.2. Praca krokowa

Dużą trudność sprawia fakt, że mikroprocesor działa bardzo szybko (minimalna częstotliwość zegara wynosi 100 kHz) i trzeba specjalnych zabiegów, żeby go zatrzymać. Należy w tym celu wykorzystać sygnały RDY oraz SYNC. Wypróbowany układ pokazuje rys. 7.2. Dokładną jego analizę pozostawiamy dociekliwym Czytelnikom, tu tylko podamy kilka zasad działania.

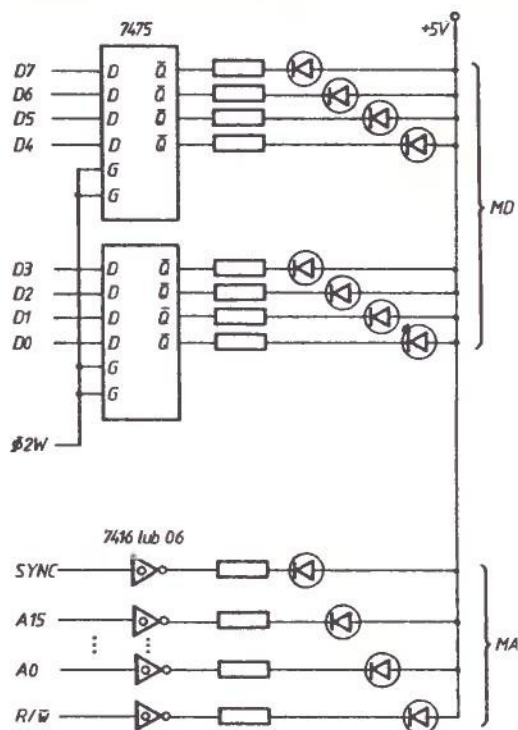


Rys. 7.2. Schemat układu pracy krokowej

W czasie normalnej pracy mikroprocesora (z pełną szybkością) należy utrzymywać $RDY = 1$. W celu zatrzymania go należy wymusić $RDY = 0$ na czas jednego taktu (po naciśnięciu klawisza 1×). Praca krokowa (po jednym rozkazie) polega na wytworzeniu $RDY = 1$ w kilku kolejnych taktach, aż do

pojawienia się sygnału *SYNC*, co oznacza rozpoczęcie nowego rozkazu. Wtedy sygnał *RDY* staje się z powrotem zerem; stąd wynikają funkcje klawiszy: kółko oznacza niestabilny, a prostokąt — stabilny. Naciśnięcie klawisza $1 \times$ powoduje wykonanie jednej akcji (przejsie do następnego cyklu maszynowego lub do następnego rozkazu).

Naciśnięcie klawisza *1C* decyduje o tym, że jedna akcja to jeden cykl maszynowy (uwaga! cykle zapisu są zawsze wykonywane, *ale w 6502 dają się wstrzymać*). Zwolnienie powoduje zatrzymanie się mikroprocesora na najbliższym cyklu pobrania rozkazu (tzn. gdy *SYNC* = 1). Zwolnienie klawisza *WŁ* wymusza stan *RDY* = 1 (praca krokowa wyłączona). Sygnał z dekodera (bez negacji) dotyczy tego obszaru pamięci, w którym program ma działać krokowo. Reszta programu będzie działała bez wstrzymywania.



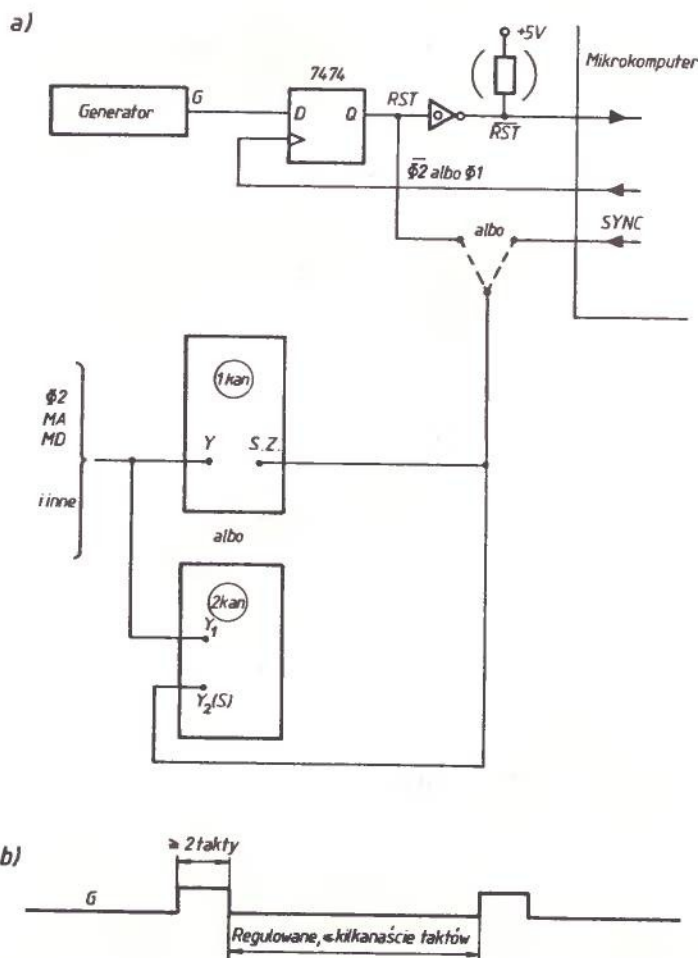
Rys. 7.3. Schemat układu wyświetlania stanu magistrali

Ale to nie wszystko: zatrzymywanie procesora ma sens, jeśli możemy obserwować jego działanie. Do środka i tak nie zajrzemy, ale „podglądanie” magistrali adresowej, magistrali danych oraz sygnału *SYNC* i sygnału *R/W* umożliwia zwykle dość szybkie znalezienie usterki. Rysunek 7.3 podaje schemat stosownego wyświetlacza: magistrala danych musi być „zatraskiwana” sygnałem $\Phi 2W$ (tj. $\Phi 2$ wzmocnione), ponieważ w pierwszej części cyklu dane są nie ustalone (najczęściej w trzecim stanie, co powoduje słabe świecenie diod zamiast ich zgaszenia).

7.3. Testowanie oscyloskopem

Jeśli mikrokomputer nie daje się uruchomić już żadnym sposobem, to należy sprawdzić, co się dzieje na samym początku, tj. po pojawieniu się i zniknięciu sygnału *RST*. Można to zrobić za pomocą wielokanałowego analizatora stanów logicznych (ale to drogie urządzenie i trudno je zdobyć w warunkach amatorskich) albo oscyloskopem z pamięcią (jeszcze droższy).

Można też użyć zwykłego oscyloskopu (dwukanałowy jest nieco wygodniejszy, ale jednokanałowy z wejściem sygnału synchronizacji zewnętrznej też wystarczy) oraz jakiegoś generatora przebiegu prostokątnego. Układ taki pokazano na rys. 7.4. Oscyloskop jednokanałowy należy synchronizować sygnałem *RST* (ujemnym zboczem), a obserwować sygnał zegara $\Phi 2$,



Rys. 7.4. Układ badania początkowego zachowania się mikroprocesora: a) schemat połączeń; b) sygnał generatora

magistralę danych, adresową itp. (przedtem — dla kalibracji czasowej — także sygnał na wejściu synchronizacji zewnętrznej *SZ*). Równie dobry do synchronizacji jest sygnał *SYNC*. Oscyloskop dwukanałowy jest o tyle lepszy, że drugi kanał pokazuje przebieg odniesienia *RST* lub *SYNC* (którym należy synchronizować pracę oscyloskopu) i w ten sposób unika się ryzyka przesunięcia odczytu na sąsiedni takt.

W obu przypadkach skalę czasu należy ustawić na 0,5, 1 albo 2 μ s/działkę. Przebieg generatora powinien wyglądać jak na rys. 7.4b. Wydłużenie stanu 0 przebiegu *G* ponad 20 taktów umożliwia wprowadzić obserwację dłuższej sekwencji początkowej pracy procesora, ale łatwo się wtedy pomylić i maleje jasność obrazu.

8

Zakończenie

Procesor 6502 powstał ponad 10 lat temu, więc nasuwa się pytanie, czy obecnie warto się nim zajmować. Nie jest on już nowością, nie stanowi szczytu możliwości technicznych. W tym wyprzedziły go procesory 16- i 32-bitowe. Z drugiej strony, w najprostszych zastosowaniach wypierają go mikrokomputery jednoukładowe, więc czy na tak zatłoczonym rynku mikroprocesorowym jest jeszcze dla niego miejsce? Odpowiedź brzmi: TAK!

Jak powiedzieliśmy na wstępie, jest on najlepszym mikroprocesorem 8-bitowym, pod względem szybkości wyprzedza on nawet procesor Z80 (który ma między innymi instrukcje blokowe — patrz Dodatek 7), zaś mikrokomputery budowane z jego wykorzystaniem są z reguły najtańsze — ze względu na niewielką liczbę wymaganych elementów. Do dziś znajduje on szerokie zastosowanie wszędzie tam, gdzie jest konieczna duża szybkość działania, a nie jest potrzebna duża moc obliczeniowa (głównie w automatyce, pomiarach i sterowaniu). Tam, gdzie nie jest wymagane zastosowanie dużej pamięci, obecnie zamiast układów mikroprocesorowych stosuje się mikrokomputery jednoukładowe. W tym kierunku poszedł też rozwój rodziny 65XX, a był on ułatwiony prostą budową procesora (a więc zajmowaniem małej powierzchni płytki krzemowej).

Firma Rockwell wypuściła całą rodzinę mikrokomputerów jednoukładowych zgodnych programowo z procesorem 6502, a więc mających dużą szybkość działania i łatwych do oprogramowania. Poza samym procesorem składają się one z pamięci ROM (od 64 do 196 bajtów), wielu linii wejścia-wyjścia (od 23 do 47 linii), liczników-czasomierzy a także układów asynchronicznej transmisji danych. Przykładami takich mikrokomputerów są układy: R6500/1, R6500/12, R6500/13 i inne.

Na podstawie tego procesora zbudowano również inteligentne układy

wejścia-wyjścia (np. R6500/42, R6500/43, R6541Q i inne), które w znacznej mierze mogą odciążyć główny procesor od żmudnych operacji wejścia-wyjścia.

Jeszcze dalej ze swoimi propozycjami poszła firma Syntertek. Uznała ona, że mikrokomputer jednoukładowy to dopiero połowa drogi. Użytkownik chce mieć układ, który zrealizuje dokładnie to, co jest potrzebne. Częścią tego układu może być mikrokomputer, ale oprócz niego są potrzebne i inne elementy składowe, takie jak przetworniki analogowo-cyfrowe i cyfrowo-analogowe, wzmacniacze, generatory itp. Użytkownik sam projektuje sobie układ w skali „makro”, używając do tego układów z biblioteki układów firmy — a w tym procesora 6502, pamięci, typowych bramek, przerzutników, liczników i innych. Schemat gotowego, przetestowanego układu biorą następnie w swoje ręce projektanci firmy Syntertek i (oczywiście z pomocą komputerów) po kilku tygodniach, metodą podobną do składania z klocków, wytwarzają wszystko na jednej małej płytce krzemu, dając użytkownikowi układ scalony „szyty na miarę” — specjalnie dla niego. I to jest chyba największa przyszłość tego procesora.

Dodatek 1

Lista rozkazów zestawiona alfabetycznie według mnemoników

TRYB ADRESOWANIA			ABS			ZPG			ABS. X			ABS. Y			ZPG. X			ZPG. Y		
MNEH	TRZEC ROZKAZU	UWAGI	KOD	B	T	KOD	B	T	KOD	B	T	KOD	B	T	KOD	B	T	KOD	B	T
ADC	$A \leftarrow A + C \rightarrow A$	1 4	0D	3	4	05	2	3	7D	3	4	79	3	4	75	2	4			
AND	$A \leftarrow A \wedge C \rightarrow A$	1a 1b	2D	3	4	25	2	3	3D	3	4	39	3	4	35	2	4			
ASL	Przes. w lewo	2	0E	3	6	06	2	5	1E	3	6				16	2	4			
BCC	Skok gdy C=0	2																		
BCS	Skok gdy C=1	2																		
BEQ	Skok gdy Z=1	2																		
BIT	$A \wedge B \rightarrow A$	3																		
BMI	Skok gdy N=1	2	2C	3	4	24	2	3	3C	3	4				34	2	4			
BNE	Skok gdy Z=0	2																		
BPL	Skok gdy N=0	2																		
BRA	Skok bezwar.	2																		
BRK	Przerw. progr.	2																		
BVC	Skok gdy V=0	2																		
BVS	Skok gdy V=1	2																		
CLC	$0 \rightarrow C$																			
CLB	$0 \rightarrow D$																			
CLI	$0 \rightarrow I$																			
CLV	$0 \rightarrow V$																			
CMF	$A \leftarrow M$	1	CD	3	4	C5	2	3	DD	3	4	DD	3	4	DD	2	4			
CPX	$X \leftarrow M$		EC	3	4	E4	2	3												
CPY	$Y \leftarrow M$		CC	3	4	C4	2	3												
DEA	$A \leftarrow 1 \rightarrow A$																			
DEC	$M \leftarrow M - 1$	1a 1b	CE	3	6	C6	2	5	DE	3	6				D6	2	6			
DEX	$X \leftarrow X - 1$																			
DEY	$Y \leftarrow Y - 1$																			
FOR	$A \leftarrow Y \rightarrow A$		4D	3	4	45	2	3	5D	3	4	59	3	4	55	2	4			
INA	$A \leftarrow 1 \rightarrow A$																			
INC	$M \leftarrow M + 1$	1a 1b	EE	3	6	E6	2	5	FE	3	6				F6	2	6			
INX	$X \leftarrow X + 1$																			
INY	$Y \leftarrow Y + 1$																			
JMP	Skok		4C	3	3															
JSR	Skok do podpr.	6	20	3	6															
LDA	$M \rightarrow A$	1	AD	3	4	A5	2	3	BD	3	4	B9	3	4	B5	2	4			
LDB	$M \rightarrow B$	1	AE	3	4	A6	2	3												
LDY	$M \rightarrow Y$	1	AC	3	4	A4	2	3	BC	3	4	DE	3	4	B4	2	4	Bo	2	4
LSK	Przes. w prawo	1a	4E	3	6	46	2	5	DE	3	6									
NOP	Nic																			
OKA	$A \wedge M \rightarrow A$	1	0D	3	4	05	2	3	1D	3	4	19	3	4	15	2	4			
PNA	Akum. na stos	1																		
PMP	P na stos	6																		
PHX	X na stos																			
PHY	Y na stos																			
PLA	Akum. ze stosu																			
PLP	P ze stosu																			
PLX	X ze stosu																			
PLY	Y ze stosu																			
R0L	Przes. cykl. w l.	1a 1b	2E	3	6	26	2	5	3E	3	6				36	2	6			
R0R	Przes. cykl. w p.	1a 1b	6E	3	6	66	2	5	7E	3	6				76	2	6			
RTL	Powrót z przerw.	9																		
RTS	Powrót z podpr.	8																		
SBC	$A \leftarrow A - C \rightarrow A$	1 4	ED	3	4	E5	2	3	FD	3	4	F9	3	4	F5	2	4			
SEC	$1 \rightarrow C$																			
SED	$1 \rightarrow D$																			
SEI	$1 \rightarrow I$																			
STA	$A \rightarrow M$								9D	3	5	99	3	5	95	2	4			
STX	$X \rightarrow M$		8E	3	4	86	2	3												
STY	$Y \rightarrow M$		BC	3	4	84	2	3												
STZ	$000 \rightarrow M$		9C	3	4	94	2	3	9E	3	5				94	2	4	96	2	4
TAX	$A \rightarrow X$																			
TAY	$A \rightarrow Y$																			
TRB	$A \wedge M \rightarrow M$	3	1C	3	6	14	2	5												
TSB	$A \wedge M \rightarrow M$	3	0C	3	6	04	2	5												
TSX	$S \rightarrow X$																			
TXA	$X \rightarrow A$																			
TXS	$X \rightarrow S$																			
TYA	$Y \rightarrow A$																			

Objaśnienia:

B - liczba bajtów rozkazu

T - liczba taktów (patrz Dodatek 3)

M7 - bit 7 z komórki pamięci

M6 - bit 6 z komórki pamięci

Pozycje zaznaczone jaśniejszym drukiem dotyczą 65C02.

Dodatek 2

Lista rozkazów zestawiona według kodów szesnastkowych

LSD MSD	0	1	2	3	4	5	6	7
0	BRK IMP	ORA (ZPG, X)			TSB ZPG	ORA ZPG	ASL ZPG	
1	BPL REL	ORA (ZPG), Y	ORA (ZPG)		TRB ZPG	ORA ZPG, X	ASL ZPG, X	
2	JSR ABS	AND (ZPG, X)			BIT ZPG	AND ZPG	ROL ZPG	
3	BHI REL	AND (ZPG), Y	AND (ZPG)		BIT ZPG, X	AND ZPG, X	ROL ZPG, X	
4	RTI IMP	EOR (ZPG, X)				EOR ZPG	LSR ZPG	
5	BVC REL	EOR (ZPG), Y	EOR (ZPG)			EOR ZPG, X	LSR ZPG, X	
6	RTS IMP	ADC (ZPG, X)			STZ ZPG, X	ADC ZPG	ROR ZPG	
7	BVS REL	ADC (ZPG), Y	ADC (ZPG)		STZ ZPG, X	ADC ZPG, X	ROR ZPG, X	
8	BRA REL	STA (ZPG, X)			STY ZPG	STA ZPG	STX ZPG	
9	BCC REL	STA (ZPG), Y	STA (ZPG)		STY ZPG, X	STA ZPG, X	STX ZPG, Y	
A	LDY IMM	LDA (ZPG, X)	LDX IMM		LDY ZPG	LDA ZPG	LDX ZPG	
B	BCS REL	LDA (ZPG), Y	LDA (ZPG)		LDY ZPG, X	LDA ZPG, X	LDX ZPG, Y	
C	CPY IMM	CMF (ZPG, X)			CPY ZPG	CMF ZPG	DEC ZPG	
D	BNE REL	CMF (ZPG), Y	CMF (ZPG)			CMF ZPG, X	DEC ZPG, X	
E	CPX IMM	SBC (ZPG, X)			CPX ZPG	SBC ZPG	INC ZPG	
F	BEQ REL	SBC (ZPG), Y	SBC (ZPG)			SBC ZPG, X	INC ZPG, X	
	0	1	2	3	4	5	6	7

LSD MSD	8	9	A	B	C	D	E	F
0	PHP IMP	ORA IMM	ASL ACC		TSB ABS	ORA ABS	ASL ABS	
1	CLC IMP	ORA ABS, Y	INC ACC		TRB ABS	ORA ABS, X	ASL ABS, X	
2	PLP IMP	AND IMM	ROL ACC		BIT ABS	AND ABS	ROL ABS	
3	SEC IMP	AND ABS, Y	DEC ACC		BIT ABS, X	AND ABS, X	ROL ABS, X	
4	PHA IMP	EOR IMM	LSR ACC		JMP ABS	EOR ABS	LSR ABS	
5	CLI IMP	EOR ABS, Y	PHY IMP			EOR ABS, X	LSR ABS, X	
6	PLA IMP	ADC IMM	ROR ACC		JMP IND	ADC ABS	ROR ABS	
7	SEI IMP	ADC ABS, Y	PLY IMP		JMP CABS, X	ADC ABS, X	ROR ABS, X	
8	DEY IMP	BIT IMM	TXA IMP		STY ABS	STA ABS	STX ABS	
9	TYA IMP	STA ABS, Y	TXS IMP		STZ ABS	STA ABS, X	STZ ABS, X	
A	TAY IMP	LDA IMM	TAX IMP		LDY ABS	LDA ABS	LDX ABS	
B	CLV IMP	LDA ABS, Y	TSX IMP		LDY ABS, X	LDA ABS, X	LDX ABS, Y	
C	INY IMP	CMP IMM	DEX IMP		CPY ABS	CMP ABS	DEC ABS	
D	CLD IMP	CMP ABS, Y	PHX IMP			CMP ABS, X	DEC ABS, X	
E	INX IMP	SBC IMM	NOP IMP		CPX ABS	SBC ABS	INC ABS	
F	SED IMP	SBC ABS, Y	PLX IMP			SBC ABS, X	INC ABS, X	
	8	9	A	B	C	D	E	F

Dodatek 3

Schematy cykli rozkazowych

D3.1. Zestawienie trybów adresowania i typów rozkazów

Tryb adresowania	Typ rozkazu			
	Przesł. do pam.	Przesł. z pam.	Dwuargumentowe	Czytanie mod. zap.
Bezwzględne <i>Absolute</i>	ABS 1	ABS 2	ABS 3	ABS 4
Na stronie zerowej <i>Zero page</i>	ZPG 1	ZPG 2	ZPG 3	ZPG 4
Indeksowe <i>Indexed</i>	ABS, XY 1 a i b	ABS, XY 2 a i b	ABS, XY 3 a i b	ABS, XY 4 a i b
Indeksowe na stronie zer. <i>Zero page indexed</i>	ZPG, XY 1	ZPG, XY 2	ZPG, XY 3	ZPG, XY 4
Pośrednie <i>Indirect</i>				
Pośrednie na stronie zer. <i>Zero page indirect</i>	(ZPG) 1	(ZPG) 2	(ZPG) 3	
Indeksowe pośrednie <i>Indexed indirect</i>				
Indeks. pośr. na str. zer. <i>Zero page indexed indirect</i>	(ZPG, X) 1	(ZPG, X) 2	(ZPG, X) 3	
Pośrednie indeksowe <i>Indirect indexed</i>	(ZPG), Y 1	(ZPG), Y 2	(ZPG), Y 3	
Względne <i>Relative</i>				
Akumulatora <i>Accumulator</i>				ACC 4
Natychmiastowe <i>Immediate</i>		IMM 2	IMM 3	
Niejawne <i>Implied</i>				
Sekwencje hardware'owe				

Typ rozkazu						
Operacje wewn.	Zapis na stos	Odczyt ze stosu	Skok	Skok względny	Oper. na bitach	Inne
			ABS 8		ABS 10	ABS 11 JSR
					ZPG 10	
			(ABS) 8			
			(ABS.X) 8			
				REL 9 a, b, c		
IMP 5	IMP 6	IMP 7				IMP1-RTS IMP2-RTI IMP3-BRK
						HD1-IRQ HD2-NMI HD3-RST

D3.2. Cykle rozkazowe

Oznaczenia:

A	— zawartość akumulatora,
M	— zawartość pamięci,
T	— numer taktu,
MA	— stan magistrali adresowej,
MD	— stan magistrali danych,
PC	— operacja na liczniku rozkazów: + 1: dodanie jedynki 0: brak zmiany stanu, Z: zapis nowej wartości,
W	— traktowanie danych: Z: zapis, O: odczyt, X: odczyt i ignorowanie danych,
KOD	— kod bieżącego rozkazu,
nKOD	— kod następnego rozkazu,
AP	— adres początku bieżącego rozkazu,
ADL	— mniej znaczący
ADH	— bardziej znaczący
PADL	— mniej znaczący
PADH	— bardziej znaczący
} bajt części adresowej rozkazu,	
} bajt adresu pośredniego,	
AW	— adres względny,
ARG	— argument operacji,
???	— przypadkowe lub nieistotne dane,
REJ	— zawartość wiadomego rejestru,
sREJ	— stara zawartość rejestru ze stosu,
sADL	— mniej znaczący
sADH	— bardziej znaczący
} bajt adresu ze stosu,	
sP	— stara zawartość rejestru P ze stosu.

Zapamiętanie danych z magistrali w rejestrach wewnętrznych procesora nie jest uwidocznione na schematach (jako oczywiste).

UWAGA: Schematy są prawdziwe dla pełnej szybkości procesora ($RDY = 1$). Przy pracy krokowej niektóre adresy mogą być obliczane wcześniej.

Sekwencje HD1 i HD2 (IRQ , NMI) wykonują się zamiast kolejnego rozkazu (po dokończeniu wykonania bieżącego). Nie pojawia się sygnał $SYNC$.

Sekwencja HD3 (RST) wykonuje się po pojawieniu się i zniknięciu sygnału $RST = 0$. Sygnał $SYNC$ pojawia się w drugim takcie.

ABS 1: STA \$7788

A=\$15

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$8D)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$88)	O	
3	AP+2 (\$0502)	+1	ADH (\$77)	O	
4	ADH, ADL (\$7788)	O	REJ (\$15)	Z	Zapis A→M\$7788
1	AP+3 (\$0503)				

ABS 2: LDX \$8899

M\$8899=\$15

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$AE)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$99)	O	
3	AP+2 (\$0502)	+1	ADH (\$88)	O	
4	ADH, ADL (\$8899)	O	ARG (\$15)	O	Przesłanie M\$8899→X
1	AP+3 (\$0503)				

ABS 3: AND \$1234

M\$1234=\$F0

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$2D)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$34)	O	
3	AP+2 (\$0502)	+1	ADH (\$12)	O	
4	ADH, ADL (\$1234)	O	ARG (\$F0)	O	
1	AP+3 (\$0503)				Wykonanie mnoż.log.

ABS 4: INC \$3456

M\$3456=\$07

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$EE)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$56)	O	
3	AP+2 (\$0502)	+1	ADH (\$34)	O	
4	ADH, ADL (\$3456)	O	ARG (\$07)	O	
5	ADH, ADL (\$3456)	O	ARG (\$07)	Z	Wyk.oper. (\$07+1=\$08)
6	ADH, ADL (\$3456)	O	nARG (\$08)	Z	
1	AP+3 (\$0503)				

Dla 65C02:

ABS 4: INC \$3456 MC(\$3456)=\$07

T	MA	PC	MD	W	
1	AP (\$0500)	+1	KOD (\$EE)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1	ADL (\$56)	O	Dekodowanie rozkazu
3	AP+2 (\$0502)	0	ADH (\$34)	O	
4	ADH, ADL (\$3456)	0	ARG (\$07)	O	
5	AP+2 (\$0502)	+1	???	X	Wyk. oper. (\$07+1=\$08)
6	ADH, ADL (\$3456)	0	nARG (\$08)	Z	
1	AP+3 (\$0503)				

??? oznacza niewiadome lub niepotrzebne dane, które mikroprocesor ignoruje.

ABS 8: tylko JMP \$0723.

T	MA	PC	MD	W	
1	AP (\$0500)	+1	KOD (\$4C)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1	ADL (\$23)	O	Dekodowanie rozkazu
3	AP+2 (\$0502)	Z	ADH (\$07)	O	
1	ADH, ADL (\$0723)				

Dla 65C02:

ABS 10: TSB \$2345 MC(\$2345)=\$0F A=\$11

T	MA	PC	MD	W	
1	AP (\$0500)	+1	KOD (\$0C)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1	ADL (\$45)	O	Dekodowanie rozkazu
3	AP+2 (\$0502)	0	ADH (\$23)	O	
4	ADH, ADL (\$2345)	0	ARG (\$0F)	O	
5	AP+2 (\$0502)	+1	???	X	Wyk. op. (\$0FV11=\$1F)
6	ADH, ADL (\$2345)	0	nARG (\$1F)	Z	
1	AP+3 (\$0503)				

ABS 11: tylko JSR \$0725.

S=\$FF

T	MA	PC	MD	W	
1	AP (\$0500)	+1	KOD (\$20)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1	ADL (\$25)	O	Dekodowanie rozkazu
3	\$01, S (\$01FF)	0	???	X	
4	\$01, S (\$01FF)	0	PCH (\$05)	Z	S: =S-1 (= \$FE)
5	\$01, S (\$01FE)	0	PCL (\$02)	Z	S: =S-1 (= \$FD)
6	AP+2 (\$0502)	Z	ADH (\$07)	O	
1	ADH, ADL (\$0725)				

ZPG 1: STA \$0034

A=\$1F

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$85)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$34)	O	
3	\$00, ADL (\$0034)	O	REJ (\$1F)	Z	Zapis A→MC\$0034
1	AP+2 (\$0502)				

ZPG 2: LDA \$0034 MC\$0034)=\$E2

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$A5)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$34)	O	
3	\$00, ADL (\$0034)	O	ARG (\$E2)	O	Przesłanie MC\$0034)→A
1	AP+2 (\$0502)				

ZPG 3: AND \$0034 MC\$0034)=\$0F

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$25)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$34)	O	
3	\$00, ADL (\$0034)	O	ARG (\$0F)	O	Wykonanie operacji
1	AP+2 (\$0502)				

ZPG 4: DEC \$0034 MC\$0034)=\$0F

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$C8)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$34)	O	
3	\$00, ADL (\$0034)	O	ARG (\$0F)	O	Wyk.oper. (\$0F-1=\$0E)
4	\$00, ADL (\$0034)	O	ARG (\$0F)	Z	
5	\$00, ADL (\$0034)	O	nARG (\$0E)	Z	
1	AP+2 (\$0502)				

Dla 65C02:

ZPG 4: DEC \$0034 MC\$0034)=\$0F

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$C8)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	O	ADL (\$34)	O	
3	\$00, ADL (\$0034)	O	ARG (\$0F)	O	Wyk.oper. (\$0F-1=\$0E)
4	AP+1 (\$0501)	+1	???	X	
5	\$00, ADL (\$0034)	O	nARG (\$0E)	Z	
1	AP+2 (\$0502)				

Dla 85002:

ZPG 10: TSB \$0034 MC \$0034)=\$0F A=\$11

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1 KOD	(\$04)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1 ADL	(\$34)	O	
4	\$00, ADL (\$0034)	0 ARG	(\$0F)	O	
5	AP+2 (\$0502)	0 ???		X	Wyk.op. (\$0FA\$11=\$1F)
6	\$00, ADL (\$0034)	0 nARG	(\$01)	Z	
1	AP+2 (\$0502)				

ABS, XY 1a: STA \$12E4, X A=\$51 X=\$03

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1 KOD	(\$9D)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1 ADL	(\$E4)	O	
3	AP+2 (\$0502)	+1 ADH	(\$12)	O	
4	ADH, ADL+X (\$12E7)	0 REJ	(\$51)	Z	ADL+X (\$E4+\$03=\$0E7)
1	AP+3 (\$0503)				

ABS, XY 1b: STA \$12E4, X A=\$51 X=\$33

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1 KOD	(\$9D)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1 ADL	(\$E4)	O	
3	AP+2 (\$0502)	+1 ADH	(\$12)	O	
4	ADH, ADL+X (\$1217)	0 ???		X	ADL+X (\$E4+\$33=\$117)
5	ADH+1, ADL+X (\$1317)	0 REJ	(\$51)	Z	ADH+1 (\$12+1=\$13)
1	AP+3 (\$0503)				

Dla 85002

ABS, XY 1b: STA \$12E4, X A=\$51 X=\$33

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1 KOD	(\$9D)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1 ADL	(\$E4)	O	
3	AP+2 (\$0502)	0 ADH	(\$12)	O	
4	AP+2 (\$0502)	+1 ???		X	ADL+X (\$E4+\$33=\$117)
5	ADH+1, ADL+X (\$1317)	0 REJ	(\$51)	Z	ADH+1 (\$12+1=\$13)
1	AP+3 (\$0503)				

ABS, XY 2a: LDA \$12E4,X MC \$12E7)=\$61 X=\$03

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$9D)	O	
2	AP+1 (\$0501)	+1	ADL (\$E4)	O	Dekodowanie rozkazu
3	AP+2 (\$0502)	+1	ADH (\$12)	O	ADL+X (\$E4+\$03=\$0E7)
4	ADH, ADL+X (\$12E7)	0	ARG (\$61)	O	
1	AP+3 (\$0503)				

ABS, XY 2b: LDA \$12E4,X MC \$1317)=\$61 X=\$33

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$9D)	O	
2	AP+1 (\$0501)	+1	ADL (\$E4)	O	Dekodowanie rozkazu
3	AP+2 (\$0502)	+1	ADH (\$12)	O	ADL+X (\$E4+\$33=\$117)
4	ADH, ADL+X (\$1217)	0	???	X	ADH+1 (\$12+1=\$13)
5	ADH+1, ADL+X (\$1317)	0	ARG (\$61)	O	
1	AP+3 (\$0503)				

Dla 65C02

ABS, XY 2b: LDA \$12E4,X MC \$1317)=\$61 X=\$33

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$9D)	O	
2	AP+1 (\$0501)	+1	ADL (\$E4)	O	Dekodowanie rozkazu
3	AP+2 (\$0502)	0	ADH (\$12)	O	ADL+X (\$E4+\$33=\$117)
4	AP+2 (\$0502)	+1	ADH	X	ADH+1 (\$12+1=\$13)
5	ADH+1, ADL+X (\$1317)	0	ARG (\$61)	O	
1	AP+3 (\$0503)				

ABS, XY 3a: AND \$12E4,X MC \$12E7)=\$22 A=\$0F X=\$33

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$9D)	O	
2	AP+1 (\$0501)	+1	ADL (\$E4)	O	Dekodowanie rozkazu
3	AP+2 (\$0502)	+1	ADH (\$12)	O	ADL+X (\$E4+\$03=\$0E7)
4	ADH, ADL+X (\$12E7)	0	ARG (\$22)	O	
1	AP+3 (\$0503)				Wyk. oper. (\$0FA\$22=\$02)

ABS, XY 3b: AND \$12E4,X MC \$1317)=\$22 A=\$0F X=\$33

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$9D)	O	
2	AP+1 (\$0501)	+1	ADL (\$E4)	O	Dekodowanie rozkazu
3	AP+2 (\$0502)	+1	ADH (\$12)	O	ADL+X (\$E4+\$33=\$117)
4	ADH, ADL+X (\$1217)	0	???	X	ADH+1 (\$12+1=\$13)
5	ADH+1, ADL+X (\$1317)	0	ARG (\$22)	O	
1	AP+3 (\$0503)				Wyk. oper. (\$0FA\$22=\$02)

Dla 65C02

ABS, XY 3b: AND \$12E4, X MC\$1317)=\$22 A=\$0F X=\$33

T	MA	PC	MD	W	
1	AP (\$0500)	+1 KOD (\$9D)		O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1 ADL (\$E4)		O	Dekodowanie rozkazu
3	AP+2 (\$0502)	0 ADH (\$12)		O	ADL+X (\$E4+\$33=\$117)
4	AP+2 (\$0502)	+1 ???		X	ADH+1 (\$12+1=\$13)
5	ADH+1, ADL+X (\$1317)	0 ARG (\$22)		O	
1	AP+3 (\$0503)				Wyk. oper. (\$0FA\$22=\$02)

ABS, XY 4: DEC \$12E4, X MC\$1314)=\$10 A=11 X=\$33

T	MA	PC	MD	W	
1	AP (\$0500)	+1 KOD (\$DE)		O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1 ADL (\$E4)		O	Dekodowanie rozkazu
3	AP+2 (\$0502)	+1 ADH (\$12)		O	ADL+X (\$E4+\$33=\$117)
4	ADH, ADL+X (\$1217)	0 ??? lub ARG		X	ADH+C (\$12+1=\$13)
5	ADH+C, ADL+X (\$1317)	0 ARG (\$10)		O	
6	ADH+C, ADL+X (\$1317)	0 ARG (\$10)		Z	Wyk. oper. (\$10-1=\$0F)
7	ADH+C, ADL+X (\$1317)	0 nARG (\$0F)		Z	
1	AP+3 (\$0503)				Wykonanie operacji

C oznacza przeniesienie z dodawania ADL+X.

Dla 65C02

ABS, XY 4a: DEC \$12E4, X MC\$12E7)=\$10 A=11 X=\$03

T	MA	PC	MD	W	
1	AP (\$0500)	+1 KOD (\$DE)		O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1 ADL (\$E4)		O	Dekodowanie rozkazu
3	AP+2 (\$0502)	0 ADH (\$12)		O	ADL+X (\$E4+\$03=\$0E7)
4	ADH, ADL+X (\$12E7)	0 ARG (\$10)		O	ADH+C (\$12+1=\$13)
5	AP+2 (\$0502)	+1 ???		X	Wyk. oper. (\$10-1=\$0F)
6	ADH, ADL+X (\$12E7)	0 nARG (\$0F)		Z	
1	AP+3 (\$0503)				

Dla 65C02

ABS, XY 4b: DEC \$12E4, X MC\$1317)=\$10 A=11 X=\$33

T	MA	PC	MD	W	
1	AP (\$0500)	+1 KOD (\$DE)		O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1 ADL (\$E4)		O	Dekodowanie rozkazu
3	AP+2 (\$0502)	0 ADH (\$12)		O	ADL+X (\$E4+\$33=\$117)
4	ADH, ADL+X (\$1217)	0 ???		X	ADH+C (\$12+1=\$13)
5	ADH+1, ADL+X (\$1317)	0 ARG (\$10)		O	
6	AP+2 (\$0502)	+1 ???		X	Wyk. oper. (\$10-1=\$0F)
7	ADH+1, ADL+X (\$1317)	0 nARG (\$0F)		Z	
1	AP+3 (\$0503)				

ZPG, XY 1: STA \$0085, X A=\$55 X=\$82

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$95)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$85)	O	
4	\$00, ADL (\$0085)	0	???	X	(ADL+X) mod 256 (\$85+\$82=\$07)
5	\$00, ADL+X (\$0007)	0	REJ (\$55)	Z	
1	AP+2 (\$0502)				

ZPG, XY 2: LDA \$0085, X MC(\$0087)=\$44 X=\$02

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$85)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$85)	O	
4	\$00, ADL (\$0085)	0	???	X	(ADL+X) mod 256 (\$85+\$02=\$87)
5	\$00, ADL+X (\$0087)	0	ARG (\$44)	Z	
1	AP+2 (\$0502)				

ZPG, XY 3: AND \$0085, X MC(\$0087)=\$4F X=\$02 A=\$55

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$35)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$85)	O	
4	\$00, ADL (\$0085)	0	???	X	(ADL+X) mod 256 (\$85+\$02=\$87)
5	\$00, ADL+X (\$0087)	0	ARG (\$4F)	Z	
1	AP+2 (\$0502)				Wyk. oper. (\$55^\$4F=\$45)

ZPG, XY 4: LSR \$0085, X MC(\$0087)=\$C3 X=\$02

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$56)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$85)	O	
3	\$00, ADL (\$0085)	0	???	X	(ADL+X) mod 256 (\$85+\$02=\$87)
4	\$00, ADL+X (\$0087)	0	ARG (\$C3)	O	Wyk. oper. (\$C3→\$02)
5	\$00, ADL+X (\$0087)	0	ARG (\$C3)	Z	
6	\$00, ADL+X (\$0087)	0	nARG (\$02)	Z	
1	AP+2 (\$0502)				

Dla 65C02

ZPG, XY 4: LSR \$0085, X MC\$0087=\$C3 X=\$02

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$56)	O	Dekodowanie rozkazu (ADL+X) mod 255 (\$85+\$02=\$87)
2	AP+1 (\$0501)	0	ADL (\$85)	O	
3	\$00, ADL (\$0085)	0	???	X	
4	\$00, ADL+X (\$0087)	0	ARG (\$C3)	O	Wyk. oper. (\$C3→\$52)
5	AP+1 (\$0501)	+1	???	X	
6	\$00, ADL+X (\$0087)	0	nARG (\$52)	Z	
1	AP+2 (\$0502)				

(ABS) 8: tylko JMP (\$1234). MC\$1234=\$78 MC\$1235=\$6

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$6C)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$E4)	O	
3	AP+2 (\$0502)	Z	ADH (\$12)	O	
4	ADH, ADL (\$1234)	+1	PADL (\$78)	O	
5	ADH, ADL+1 (\$1235)	Z	PADH (\$56)	O	
1	PADH, PADL (\$5678)				

Dla 65C02

(ABS) 8: tylko JMP (\$12FF). MC\$12FF=\$78 MC\$1300=\$6

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$6C)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$E4)	O	
3	AP+2 (\$0502)	Z	ADH (\$12)	O	
4	ADH, ADL (\$12FF)	+1	PADL (\$78)	O	ADH+1 (\$12+1=\$13)
5	ADH, ADL+1 (\$1200)	+1	???	X	
6	ADH+1, ADL+1 (\$1300)	Z	PADH (\$56)	O	
1	PADH, PADL (\$5678)				

Dla 65C02

(ZPG) 1: STA (\$0025) MC\$0025=\$78 MC\$0026=\$56 A=\$34

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$92)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$25)	O	
3	\$00, ADL (\$0025)	0	ARG (\$78)	O	
4	\$00, ADL+1 (\$0026)	0	ARG (\$56)	O	ADL+1 (\$25+1=\$26)
5	PADH, PADL (\$5678)	0	REJ (\$34)	Z	
1	AP+2 (\$0502)				

Dla 65C02

(ZPG) 2: LDA (\$0025) HC\$0025)=\$78 HC\$0026)=\$56 HC\$5678)=\$33

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$92)	O	Dekodowanie rozkazu ADL+1 (\$25+1=\$26)
2	AP+1 (\$0501)	+1	ADL (\$25)	O	
3	\$00, ADL (\$0025)	O	ARG (\$78)	O	
4	\$00, ADL+1 (\$0026)	O	ARG (\$56)	O	
5	PADH, PADL (\$5678)	O	ARG (\$33)	O	
1	AP+2 (\$0502)				

Dla 65C02

(ZPG) 3: AND (\$0025) HC\$0025)=\$78 HC\$0026)=\$56 HC\$5678)=\$33 A=\$F1

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$92)	O	Dekodowanie rozkazu ADL+1 (\$25+1=\$26)
2	AP+1 (\$0501)	+1	ADL (\$25)	O	
3	\$00, ADL (\$0025)	O	ARG (\$78)	O	
4	\$00, ADL+1 (\$0026)	O	ARG (\$56)	O	
5	PADH, PADL (\$5678)	O	ARG (\$33)	O	
1	AP+2 (\$0502)				Wyk.oper. (\$F1^\$33=\$31)

Dla 65C02

(ABS, X) 8: tylko JMP (\$1284.X) X=\$82

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$7C)	O	Dekodowanie rozkazu ADL+X (\$84+\$82=\$106) ADH+C (\$12+1=\$13)
2	AP+1 (\$0501)	+1	ADL (\$84)	O	
3	AP+2 (\$0502)	O	ADH (\$12)	O	
4	AP+2 (\$0502)	Z ???		X	
5	(ADH, ADL)+X (\$1306)	+1	PADL (\$78)	O	
6	(ADH, ADL)+X+1 (\$1307)	Z	PADH (\$56)	O	
1	PADH, PADL (\$5678)				

(ZPG, X) 1: STA (\$23, X) HC\$002A)=\$56 HC\$002B)=\$34 X=\$07 A=\$11

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$81)	O	Dekodowanie rozkazu (ADL+X) mod 256 (\$23+\$07=\$2A) (ADL+X+1) mod 256 (\$2A+1=\$2B)
2	AP+1 (\$0501)	+1	ADL (\$23)	O	
3	\$00, ADL (\$0023)	O	???	X	
4	\$00, ADL+X (\$002A)	O	PADL (\$56)	O	
5	\$00, ADL+X+1 (\$002B)	O	PADH (\$34)	O	
6	PADH, PADL (\$3456)	O	REJ (\$11)	Z	
1	AP+2 (\$0502)				

(ZPG, X) 2: LDA (\$23, X) HC\$002A)=\$56 HC\$002B)=\$34 X=\$07 HC\$3456)=\$33

T	MA	PC	MD	W	
1	AP (\$0500)	+1 KOD	(\$81)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1 ADL	(\$23)	O	Dekodowanie rozkazu
3	\$00, ADL (\$0023)	O ???		X	(ADL+X) mod 256 (\$23+\$07=\$2A)
4	\$00, ADL+X (\$002A)	O PADL	(\$56)	O	(ADL+X+1) mod 256 (\$2A+1=\$2B)
5	\$00, ADL+X+1 (\$002B)	O PADH	(\$34)	O	
6	PADH, PADL (\$3456)	O ARG	(\$33)	O	
1	AP+2 (\$0502)				

(ZPG, X) 3: AND (\$23, X) HC\$002A)=\$56 HC\$002B)=\$34 X=\$07
HC\$3456)=\$33 A=\$1E

T	MA	PC	MD	W	
1	AP (\$0500)	+1 KOD	(\$81)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1 ADL	(\$23)	O	Dekodowanie rozkazu
3	\$00, ADL (\$0023)	O ???		X	(ADL+X) mod 256 (\$23+\$07=\$2A)
4	\$00, ADL+X (\$002A)	O PADL	(\$56)	O	(ADL+X+1) mod 256 (\$2A+1=\$2B)
5	\$00, ADL+X+1 (\$002B)	O PADH	(\$34)	Z	
6	PADH, PADL (\$3456)	O ARG	(\$33)	O	
1	AP+2 (\$0502)				Wyk.oper. (\$1E+\$33=\$12)

(ZPG, Y) 1: STA (\$23, Y) HC\$0023)=\$82 HC\$0024)=\$31 Y=\$83 A=\$55

T	MA	PC	MD	W	
1	AP (\$0500)	+1 KOD	(\$91)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1 ADL	(\$23)	O	Dekodowanie rozkazu
3	\$00, ADL (\$0023)	O PADL	(\$82)	O	ADL+1 (\$23+1=\$24)
4	\$00, ADL+1 (\$0024)	O PADH	(\$31)	O	PADL+Y (\$82+\$85=\$107)
5	PADH, PADL+Y (\$3107)	O ???		X	PADH+C (\$31+1=\$32)
6	PADH+C, PADL+Y (\$3207)	O REJ	(\$55)	Z	
1	AP+2 (\$0502)				

(ZPG, Y) 2: LDA (\$23, Y) HC\$0023)=\$82 HC\$0024)=\$31 Y=\$83
HC\$3207)=\$86

T	MA	PC	MD	W	
1	AP (\$0500)	+1 KOD	(\$91)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1 ADL	(\$23)	O	Dekodowanie rozkazu
3	\$00, ADL (\$0023)	O PADL	(\$82)	O	ADL+1 (\$23+1=\$24)
4	\$00, ADL+1 (\$0024)	O PADH	(\$31)	O	PADL+Y (\$82+\$85=\$107)
5	PADH, PADL+Y (\$3107)	O ???		X	PADH+C (\$31+1=\$32)
6	PADH+C, PADL+Y (\$3207)	O ARG	(\$86)	O	
1	AP+2 (\$0502)				

(ZP6), Y 3: AND (\$23),Y HC\$0023)=\$82 HC\$0024)=\$31 Y=\$83

HC\$3207)=\$66 A=\$5F

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0500)	+1	KOD (\$91)	O	Dekodowanie rozkazu
2	AP+1 (\$0501)	+1	ADL (\$23)	O	
3	\$00, ADL (\$0023)	O	PADL (\$82)	O	ADL+1 (\$23+1=\$24),
4	\$00, ADL+1 (\$0024)	O	PADH (\$31)	O	PADL+Y (\$82+\$85=\$107)
5	PADH, PADL+Y (\$3107)	O	???	X	PADH+C (\$31+1=\$32)
6	PADH+C, PADL+Y (\$3207)	O	ARG (\$56)	O	
1	AP+2 (\$0502)				Wyk.oper. (\$3FA\$66=\$56)

REL 9a: BMI #\$40 N=0

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0580)	+1	KOD (\$30)	O	Dekodowanie rozkazu
2	AP+1 (\$0581)	+1	AW (\$40)	O	
1	AP+2 (\$0582)				Spr.znacznika N *
					* oraz obliczenie na zapas PCL+AW (\$82+\$40=\$C2)

Skok nie wykonany.

REL 9b: BMI #\$40 N=1

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0580)	+1	KOD (\$30)	O	Dekodowanie rozkazu
2	AP+1 (\$0581)	+1	AW (\$40)	O	
3	AP+2 (\$0582)	Z	nKOD	X	Spr.znacznika N *
1	AP+2+AW (\$05C2)				Obl. PCH+C (zbędne)
					* oraz obliczenie PCL+AW (\$82+\$40=\$C2)

Skok na tę samą stronę.

REL 9c: BMI #\$7F N=1

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP (\$0580)	+1	KOD (\$30)	O	Dekodowanie rozkazu
2	AP+1 (\$0581)	+1	AW (\$7F)	O	
3	AP+2 (\$0582)	+1	nKOD	X	Spr.znacznika N *
4	AP+2+AW bp (\$0501)	Z	???	X	Obliczenie PCH+C
1	AP+2+AW (\$0601)				* oraz obliczenie PCL+AW (\$82+\$7F=\$101)

Skok na sąsiednią stronę.

UWAGA: W takcie 4 w schemacie REL 9c wykonuje się obliczenie nowej wartości PCH, a zapis jest przygotowywany (wykonuje się na końcu taktu).

Dla 65C02: W rozkazie BRA nie ma sprawdzania znacznika. REL 9a nie występuje.

ACC 4: LSR A A=\$F1

T	MA	PC	MD	W	
1	AP (\$0500)	+1	KOD (\$4A)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	0	nKOD	X	Dekodowanie rozkazu
1	AP+1 (\$0501)				Wyk.oper. (\$F1→\$78)

IMM 2: LDA #\$15

T	MA	PC	MD	W	
1	AP (\$0500)	+1	KOD (\$A9)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1	ARG (\$15)	O	Dekodowanie rozkazu
1	AP+2 (\$0502)				

IMM 3: AND #\$15 A=\$E8

T	MA	PC	MD	W	
1	AP (\$0500)	+1	KOD (\$29)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	+1	ARG (\$15)	O	Dekodowanie rozkazu
1	AP+2 (\$0502)				Wyk.oper. (\$E8∧\$15=\$04)

IMP 5: TAY

T	MA	PC	MD	W	
1	AP (\$0500)	+1	KOD (\$)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	0	nKOD	X	Dekodowanie rozkazu
1	AP+1 (\$0501)				Wykonanie operacji

IMP 6: PHA A=\$53 S=\$FF

T	MA	PC	MD	W	
1	AP (\$0500)	+1	KOD (\$48)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	0	nKOD	X	Dekodowanie rozkazu
3	\$01, S (\$01FF)	0	REJ (\$53)	Z	
1	AP+1 (\$0501)				S: =S-1 (\$FF-1=\$FE)

IMP 7: PLA HC\$01FE=\$53 S=\$FE

T	MA	PC	MD	W	
1	AP (\$0500)	+1	KOD (\$08)	O	OPERACJA WEWNĘTRZNA
2	AP+1 (\$0501)	0	nKOD	X	Dekodowanie rozkazu
3	\$01, S (\$01FE)	0	???	X	S: =S+1 (\$FE+1=\$FF)
4	\$01, S (\$01FF)	0	sREJ (\$53)	O	
1	AP+1 (\$0501)				

IMP 12: tylko RTS. $MC\$01FE) = \02 $S = \$FD$
 $MC\$01FF) = \05

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP ($\$0740$)	+1	KOD ($\$50$)	O	Dekodowanie rozkazu $S := S + 1$ ($\\$FD + 1 = \\FE) $S := S + 1$ ($\\$FE + 1 = \\FF)
2	AP+1 ($\$0741$)	0	nKOD	X	
3	$\$01, S$ ($\$01FD$)	0	???	X	
4	$\$01, S$ ($\$01FE$)	0	$\leq PCL$ ($\$02$)	O	
5	$\$01, S$ ($\$01FF$)	Z	$\leq PCH$ ($\$05$)	O	
6	$\leq PC$ ($\$0502$)	+1	???	X	
1	$\leq PC + 1$ ($\$0503$)				

IMP 13: tylko RTI. $MC\$01FD) = \$A3$ $S = \$FC$
Porównaj IMP 14 i 16. $MC\$01FE) = \30
 $MC\$01FF) = \06

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP ($\$0750$)	+1	KOD ($\$50$)	O	Dekodowanie rozkazu $S := S + 1$ ($\\$FC + 1 = \\FD) $S := S + 1$ ($\\$FD + 1 = \\FE) $S := S + 1$ ($\\$FE + 1 = \\FF)
2	AP+1 ($\$0751$)	0	nKOD	X	
3	$\$01, S$ ($\$01FC$)	0	???	X	
4	$\$01, S$ ($\$01FD$)	0	$\leq P$ ($\$A3$)	O	
5	$\$01, S$ ($\$01FE$)	0	$\leq PCL$ ($\$30$)	O	
6	$\$01, S$ ($\$01FF$)	Z	$\leq PCH$ ($\$06$)	O	
1	$\leq PC$ ($\$0630$)				

IMP 14: tylko BRK. $MC\$FFFE) = \20 $S = \$FF$ $P = \$A3$
Porównaj IMP 13. $MC\$FFFF) = \07

T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA
1	AP ($\$0630$)	+1	KOD ($\$00$)	O	Dekodowanie rozkazu $S := S - 1$ ($\\$FF - 1 = \\FE) $S := S - 1$ ($\\$FE - 1 = \\FD) $S := S - 1$ ($\\$FD - 1 = \\FC) $I := 1$ $B := 1$ ($P = \\$B7$)
2	AP+1 ($\$0631$)	0	nKOD	X	
3	$\$01, S$ ($\$01FF$)	0	PCH ($\$06$)	Z	
4	$\$01, S$ ($\$01FE$)	0	PCL ($\$30$)	Z	
5	$\$01, S$ ($\$01FD$)	Z	P ($\$A3$)	Z	
6		+1	PADL ($\$20$)	O	
7		Z	PADH ($\$07$)	O	
1	PAD ($\$0720$)				

HD 1: tylko $\neq$ IRQ w trakcie wykonywania rozkazu o ostatnim bajcie w komórce \$062F.

Porównaj IMP 13.

		MC\$FFFE)=\$20		MC\$FFF)=\$07		S=\$FF	P=\$A3
T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA		
1	ADR (\$0630)	O KOD		X			
2	ADR (\$0631)	O KOD		X			
3	\$01,S (\$01FF)	O PCH	(\$06)	Z	S: =S-1 (\$FF-1=\$FE)		
4	\$01,S (\$01FE)	O PCL	(\$30)	Z	S: =S-1(\$FE-1=\$FD)		
5	\$01,S (\$01FD)	Z P	(\$A3)	Z	S: =S-1(\$FD-1=\$FC)		
6	(\$FFFE)	+1 ADL	(\$20)	O	I: =1 (P=\$A7)		
7	(\$FFFF)	Z ADH	(\$07)	O			
1	ADH,ADL (\$0720)						

UWAGA: ADR i KOD dotyczą rozkazu, który by się wykonał, gdyby nie przerwanie. Sygnał SYNC nie pojawia się wcale.

HD 2: tylko $\neq$ NMI w trakcie wykonywania rozkazu o ostatnim bajcie w komórce \$062F.

Porównaj HD 1.

		MC\$FFFA)=\$20		MC\$FFFB)=\$07		S=\$FF	P=\$A3
T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA		
1	ADR (\$0630)	O KOD		X			
2	ADR (\$0631)	O KOD		X			
3	\$01,S (\$01FF)	O PCH	(\$06)	Z	S: =S-1 (\$FF-1=\$FE)		
4	\$01,S (\$01FE)	O PCL	(\$30)	Z	S: =S-1(\$FE-1=\$FD)		
5	\$01,S (\$01FD)	Z P	(\$A3)	Z	S: =S-1(\$FD-1=\$FC)		
6	(\$FFFA)	+1 ADL	(\$20)	O	I: =1 (P=\$A7)		
7	(\$FFFB)	Z ADH	(\$07)	O			
1	ADH,ADL (\$0720)						

UWAGA: ADR i KOD dotyczą rozkazu, który by się wykonał, gdyby nie przerwanie. Sygnał SYNC nie pojawia się wcale.

HD 3: tylko $\neq$ RST (po jego zniknięciu).

Porównaj HD 1.

		MC\$FFFC)=\$20		MC\$FFFD)=\$08		S=\$45	
T	MA	PC	MD	W	OPERACJA WEWNĘTRZNA		
1	ADR (\$0630)	O ???		X			
2	ADR (\$0631)	O ???		X			
3	\$01,S (\$0145)	O ???		X	S: =S-1 (\$45-1=\$44)		
4	\$01,S (\$0144)	O ???		X	S: =S-1(\$44-1=\$43)		
5	\$01,S (\$0143)	Z ???		X	S: =S-1(\$43-1=\$42)		
6	(\$FFFC)	+1 ADL	(\$20)	O	I: =1		
7	(\$FFFD)	Z ADH	(\$08)	O			
1	ADH,ADL (\$0820)						

UWAGA: ADR - adres przypadkowy. Sygnał SYNC pojawia się w T2, a nie w T1.

Dodatek 4

Arytmetyka (przykłady)

D4.1. Kod NB, dodawanie

	Wagi	128	64	32	16	8	4	2	1	
172 Argument1 (z akumulatora)		1	0	1	0	1	1	0	0	0
+ 154 +Argument2 (z pamięci)	+	1	0	0	1	1	0	1	0	
= 70 Suma (do akumulatora)		1	0	1	0	0	0	1	1	0
(+256) nadmiar										

↑
Przeniesienie końcowe (nadmiar) do C

D4.2. Kod NB, odejmowanie

	Wagi	128	64	32	16	8	4	2	1	
154 Argument1 (z akumulatora)		1	0	0	1	1	0	1	0	1
- 116 -Argument2 (z pamięci)	-	0	1	1	1	0	1	0	0	
= 38 Różnica (do akumulatora)		1	0	0	1	0	0	1	1	0

↑
Przeniesienia końcowe (nadmiar) po zanegowaniu do C

D4.3. Kod U2, dodawanie

	Wagi	-128	+64	+32	+16	+8	+4	+2	+1	
(+108) Argument1 (z akumulatora)		0	1	1	0	1	1	0	0	0
+ (-38) +Argument2 (z pamięci)	+	1	1	0	1	1	0	1	0	
= (+70) Suma (do akumulatora)		1	0	1	0	0	0	1	1	0

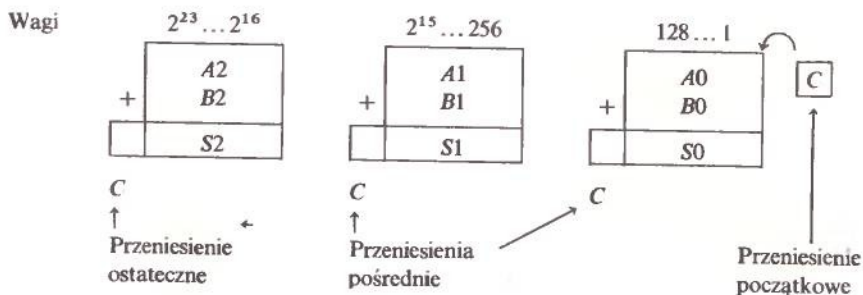
↑
Przeniesienie końcowe (nadmiar) do V

D4.4. Kod U2, odejmowanie

	Wagi	-128	+64	+32	+16	+8	+4	+2	+1	
(+64) Argument1 (z akumulatora)		0	1	0	0	0	0	0	0	1
- (-66) -Argument2 (z pamięci)	-	1	0	1	1	1	1	1	0	
= (+2) Różnica (do akumulatora)		1	0	0	0	0	0	0	1	0
(+128) nadmiar										

↑
Przeniesienie końcowe (nadmiar) do V

D4.5. Kod NB, dodawanie w potrójnej precyzji



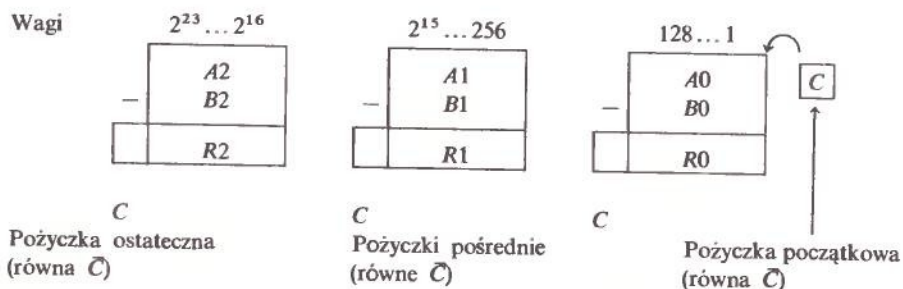
Program:

```

CLD          ; dwójkowa praca sumatora –można użyć wcześniej
CLC          ; zerowe przeniesienie początkowe
LDA  A0      ; pobranie najmniej znaczącej części argumentu A
ADC  B0      ; pobranie najmniej znaczącej części argumentu B
STA  S0      ; zapamiętanie najmniej znaczącej części sumy
              ; znacznik C stanowi przeniesienie pośrednie

LDA  A1      ;
ADC  B1      ; } jw., ale dla środkowej części
STA  S1      ;
              ;
LDA  A2      ;
ADC  B2      ; } jw., ale dla najbardziej znaczącej części
STA  C2      ;
              ;
              ; tu znacznik C mówi o nadmiarze całego wyniku
    
```

D4.6. Kod NB, odejmowanie w potrójnej precyzji



Program:

```

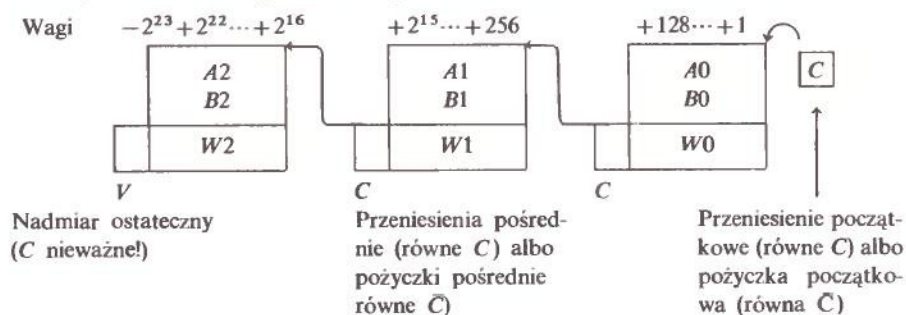
SEC          ; zerowa pożyczka początkowa =  $\bar{C}$ 
LDA  A0      ; pobranie najmniej znaczącej części argumentu A
SBC  B0      ; odjęcie najmniej znaczącej części argumentu B
STA  R0      ; zapamiętanie najmniej znaczącej części różnicy R
              ; znacznik C stanowi negację pożyczki pośredniej
    
```

```

LDA    A1    ;
SBC    B1    ;
STA    R1    ; } jw., ale dla środkowej części
;
;
;
;
LDA    A2    ;
SBC    B2    ; } jw., ale dla najbardziej znaczącej części
STA    R2    ;
;
; tu znacznik C stanowi negację pożyczki (czyli nad-
miaru) całego wyniku

```

D4.7. Kod U2, dodawanie i odejmowanie w potrójnej precyzji



Program:

```

CLC    albo    SEC    ; zerowe przeniesienie albo pożyczka
LDA     A0      ; pobranie najmniej znaczącej części
                ; argumentu A
ADC     albo    SBC    B0    ; dodanie albo odjęcie najmniej znaczącej
                ; części argumentu B
STA     W0      ; zapamiętanie najmniej znaczącej części
                ; wyniku
                ; C = przeniesienie lub negacja pożyczki
                ; pośredniej

LDA     A1      ;
ADC     albo    SBC    B1      ; } jw., ale dla środkowej części
STA     W1      ;
;
;
;
LDA     A2      ;
ADC     albo    SBC    B2      ; } jw., ale najbardziej znaczącej części
STA     W2      ;
;
; tutaj C jest nieważne
; V = 1 świadczy o przepełnieniu

```


D4.8. Kod BCD, dodawanie i odejmowanie w potrójnej precyzji – jak w NB, ale przed działaniami trzeba wykonać rozkaz SED

UWAGA 1: jeśli po wykonaniu odejmowania wystąpi przepełnienie (tj. $C = 0$), to wynik jest w kodzie uzupełnień do 10 (tak jak się liczy przy wydawaniu reszty), np. dla podwójnej precyzji

	5	0	0	0
–	6	5	4	4
0	8	4	5	6
C				

Wynik jest ujemny: taki jak wydanie reszty z 10000 do kwoty 8456, czyli –1544.

UWAGA 2: w argumentach w kodzie BCD muszą się znajdować cyfry z zakresu 0...9. W przeciwnym przypadku sumator przeliczy je na kod BCD, ale na ogół z błędem.

Dodatek 5

Różnice między mikroprocesorem 6502 a 65C02

Cecha	6502	65C02
Wykonanie nie wykorzystanych kodów	Operacje są nieokreślone. Niektóre można zakończyć tylko przez RST	Wszystkie działają jak NOP (rezerwa na przyszłość)
Rozkazy z adresowaniem indeksowym z przekraczaniem strony.	Zbędny odczyt spod złego adresu.	Zbędny odczyt ostatniego bajtu rozkazu.
Wykonanie rozkazów z grupy 4 (odczyt/modyfikacja/zapis)	Jeden cykl odczytu i dwa zapisu.	Dwa cykle odczytu i jeden zapisu.
Wykonanie rozkazów z grupy 4 i 10		W systemach wieloprocessorowych może się odbyć w kolejnych taktach dzięki sygnałowi ML (memory lock)
Skok pośredni (P8) z argumentem \$XXFF	Adres strony się nie zwiększa.	Adres strony się zwiększa (i wykonanie rozkazu wydłuża się o 1 cykl).
Tryby adresowania		Rozszerzono (patrz D1 i D2)
Znacznik D	Nieokreślony po RST.	Ustawia się na 0 po RST i przerwaniach.

Znaczniki po działaniach arytmetycznych BCD.	N , V i Z są nieokreślone.	Są określone. Wykonanie rozkazu jest dłuższe o 1 cykl
Przerwanie zgłoszone po pobraniu rozkazu BRK	Pobrany zostanie wektor przerwania zamiast wektora BRK.	BRK zostanie wykonany najpierw, przerwanie potem.
$RDY = 0$	Zatrzymuje procesor tylko w cyklach odczytu (gdy $R/\overline{W} = 1$)	Zatrzymuje procesor w każdym cyklu.
Zegar	Ma minimalną częstotliwość (ok. 100 kHz).	Można zatrzymać bez złych efektów.
Bit 5 w rejestrze P	Nieokreślony.	Równy 1.
Nie używane wejścia: $\overline{IRQ}$, $\overline{NMI}$, RDY , RST , SO	Trzeba dołączyć do stałej jedynki (ze względu na zakłócenia).	Są już zaopatrzone w oporniki (ok. 250 k Ω).

Dodatek 6

Przykłady programów

W dodatku tym zamieszczamy kilka przykładów prostych procedur (programów). Programy te są opatrzone tylko krótkim komentarzem, takim jaki znaleźć można w „prawdziwych” programach pisanych w języku maszynowym. Pełne zrozumienie działania tych programów pozostawiamy Czytelnikowi jako ćwiczenie zrozumienia zasad działania procesora 6502. Dodatkowym ćwiczeniem dla Czytelnika może być obliczanie czasu działania programu, a także próba modyfikacji tych programów, aby na przykład uzyskać większą precyzję obliczeń.

D6.1. Przepisywanie bloku danych o długości do 128 bajtów z jednego miejsca pamięci do drugiego

Rejestr Y = długość bloku - 1

Pomocnicze komórki na stronie zerowej:

BLOK1:

...

 — część mniej znacząca
 — część bardziej znacząca

 } adresu początku bloku,
 do którego wpisuje się dane

BLOK2:

...

 — część mniej znacząca
 — część bardziej znacząca

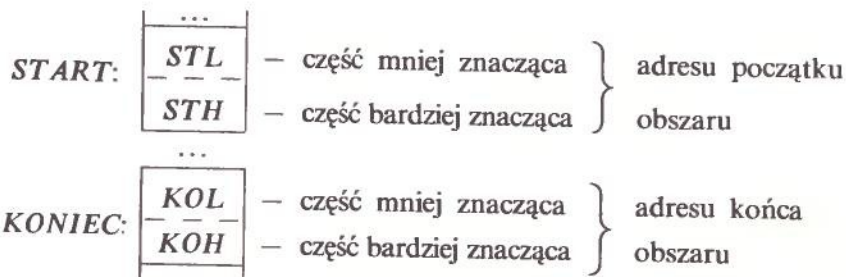
 } adresu początku bloku,
 z którego dane się czyta

KOPIUJ LDA (BLOK1),Y ; odczytanie danej z bloku 1

STA	(BLOK2),Y	; zapis danej do bloku 2
DEY		; zmniejszenie zawartości Y
BPL	KOPIUJ	; następny bajt
RTS		

D6.2. Zerowanie dużego obszaru pamięci (wersja najwolniejsza, lecz najprostsza)

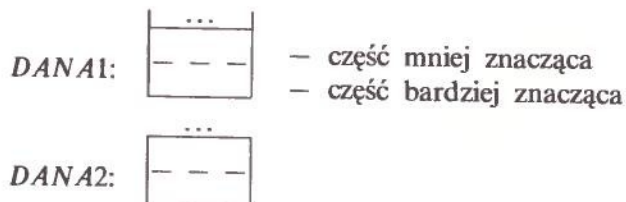
Pomocnicze komórki na stronie zerowej:

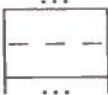


ZERUJ	LDY	# 00	
	LDA	# 00	
PĘTLA	STA	(START), Y	; wyzerowanie jednej komórki
	LDX	KONIEC	; czy ST = KO
	CMP	START	; porównanie części mniej znaczącej
	BNE	NASTĘPNY	; (czy STL = KOL)
	LDX	KONIEC+1	; porównanie części bardziej znaczącej
	CMP	START+1	; (czy STH = KOH)
	BEQ	ZROBIONE	
NASTĘPNY	INC	START	; START = START1
	BNE	PĘTLA	
	INC	START+1	; przekroczenie granicy strony
	JMP	PĘTLA	; następny adres
ZROBIONE	RTS		

D6.3. Dodawanie w podwójnej precyzji w kodzie U2

Dane i wynik umieszczone w pamięci:

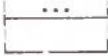


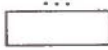
WYNIK: 

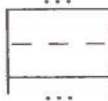
DODAJ	CLC		; wyzerowanie znacznika przeniesienia
	LDA	DANA1	
	ADC	DANA2	; dodanie mniej znaczących bajtów
	STA	WYNIK	
	LDA	DANA1+1	
	ADC	DANA2+1	; dodanie bardziej znaczących bajtów
	STA	WYNIK+1	; (z przeniesieniem)
	BVS	BŁĄD	; testowanie przekroczenia zakresu
	RTS		
BŁĄD	...		; procedura obsługi błędu przekroczenia zakresu

D6.4. Mnożenie liczb jednobajtowych bez znaku, z wynikiem 2-bajtowym

Dane i wynik w pamięci, na stronie zerowej:

DANA1: 

DANA2: 

WYNIK:  — część mniej znacząca
— część bardziej znacząca

MNOŻENIE	LDY	# 07	; licznik bitów
	LDA	# 00	; zerowanie wyniku
PĘTLA	ROR	DANA1	; testowanie kolejnych bitów
	BCC	ZERO	; DANEJ1
	CLC		
	ADC	DANA2	; dodanie DANEJ2
ZERO	ROR	A	
	ROR	WYNIK	
	DEY		
	BPL	PĘTLA	; następny bit
	STA	WYNIK+1	
	RTS		

Przykłady porównania szybkości pracy różnych mikroprocesorów

Najprostszą metodą porównania szybkości pracy różnych mikroprocesorów jest danie im do wykonania tego samego zadania i sprawdzenie, ile czasu zadanie to się wykonuje. W naszym przykładzie zadanie jest następujące:

Należy odebrać blok danych z urządzenia peryferyjnego, programowo testując przyjscie nowej danej i zapisać ten blok do pamięci.

Poniżej przedstawiono programy wykonujące to zadanie, ich długość i czas wykonania przypadający na odbiór jednej danej. Do analizy zostały wzięte najpopularniejsze procesory 8-bitowe: 6502, 6800, 6809, 8080 i Z80. Zestawienie najważniejszych parametrów tych procesorów zamieszczono na końcu tego Dodatku. We wszystkich zamieszczonych programach są używane następujące oznaczenia:

- POCZĄTEK** – adres początkowy zapisywanego bloku danych,
KONIEC – adres końcowy zapisywanego bloku danych,
RK – rejestr kontrolny urządzenia peryferyjnego: bit *b7* tego rejestru zawiera informację, że nowa dana jest gotowa do odczytu,
DANA – rejestr danych urządzenia peryferyjnego, pracującego w trybie automatycznego potwierdzania danych.

D7.1. Procesor 6502 – wersja 1 MHz

Adres urządzenia peryferyjnego jest umieszczony na stronie zerowej. Rejestr *Y* jest licznikiem danych, o wartości początkowej 0.

			Liczba bajtów rozkażu	Czas wykonania (liczba cykli zegara)
PĘTLA	LDA	RK	2	3
	BPL	PĘTLA	2	2/3
	LDA	DANA	2	3
	STA	POCZĄTEK, Y	3	5
	INY		1	2
	BPL	PĘTLA	2	2/3
Suma:			12	18

Czas odebrania danej 18 μ s, szybkość transmisji 55 KB/s.

D7.2. Procesor MC 6800 – wersja 1 MHz

Urządzenie peryferyjne na stronie zerowej (bezpośredniej). Rejestr *X* wskazuje miejsce pamięci, do którego należy wpisać daną. Wartość początkowa rejestru – **POCZĄTEK**.

			Liczba bajtów rozkażu	Czas wykonania (liczba cykli zegara)
PĘTLA	LDA A	RK	2	3
	BPL	PĘTLA	2	4
	LDA A	DANA	2	3
	STA A	0, X	2	6
	INX		1	4
	CPX	KONIEC	3	3
	BNE	PĘTLA	2	4
Suma: 14				27

Czas odebrania danej 27 μ s, szybkość transmisji 37 KB/s.

D7.3. Procesor MC 6809 – wersja 1 MHz

Uwagi jak dla procesora 6800.

			Liczba bajtów rozkażu	Czas wykonania (liczba cykli zegara)
PĘTLA	LDA A	RK	2	4
	BPL	PĘTLA	2	3
	LDA A	DANE	2	4
	STA A	X +	2	6
	CMP X	KONIEC	3	4
	BNE	PĘTLA	2	3
Suma: 13				24

Czas odebrania danej 24 μ s, szybkość transmisji 41 KB/s.

D7.4. Procesor 8080 – wersja 2 MHz

Urządzenie peryferyjne umieszczone w obszarze we-wy. Rejestr *HL* wskazuje miejsce w pamięci, do którego należy wpisać daną. Wartość początkowa tego rejestru – *POCZĄTEK*. Rejestr *B* jest licznikiem danych, o wartości początkowej 128.

			Liczba bajtów rozkażu	Czas wykonania (liczba cykli zegara)
PĘTLA	IN	RK	2	10
	ANA	A	1	4
	JP	PĘTLA	3	10
	IN	DANA	1	10
	MOV	A, M	1	5
	INX	H	1	5
	DCR	B	1	4
	JNZ	PĘTLA	3	10
Suma: 13				58

Czas odebrania danej 29 μ s, szybkość transmisji 34 KB/s.

D7.5. Procesor Z80 — wersja 2,5 MHz

Uwagi jak dla procesora 8080, wykorzystano instrukcję przesłania blokowego.

			Liczba bajtów rozkażu	Czas wykonania (liczba cykli zegara)
PĘTLA	IN	RK	2	11
	AND	A, A	1	4
	JP	P, PĘTLA	3	10
	INI		2	16
	JP	NZ, PĘTLA	3	10
Suma:			11	51

Czas odebrania danej 20,4 μ s. szybkość transmisji 49 KB/s.

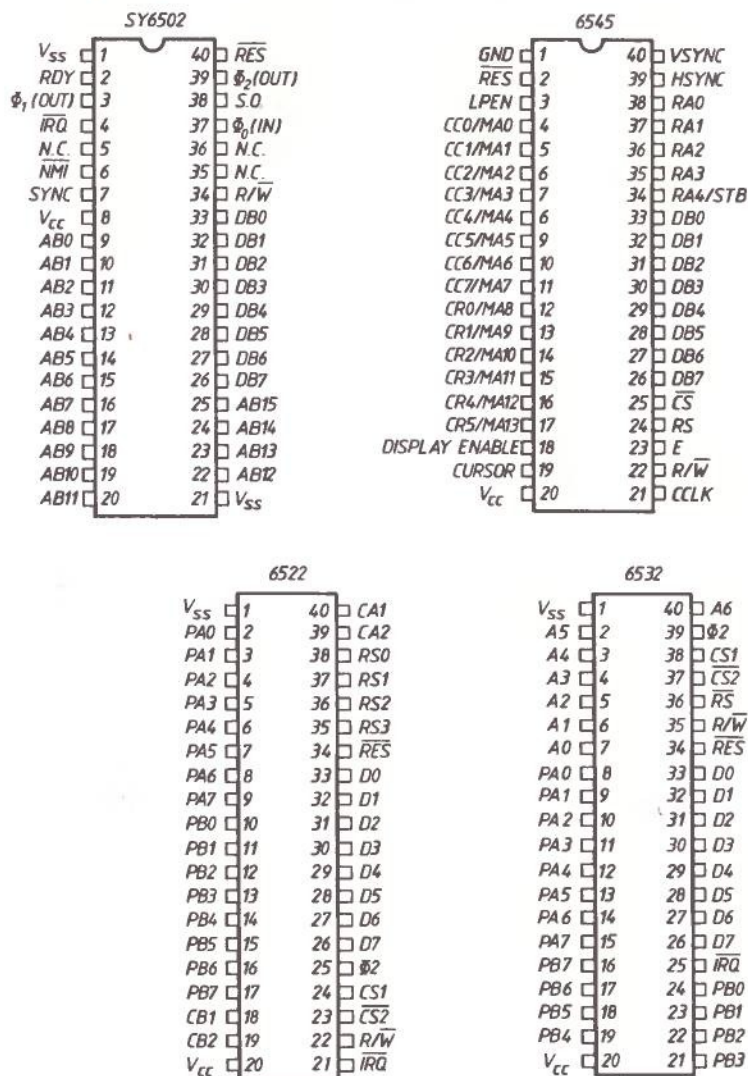
D7.6. Zestawienie podstawowych danych procesorów 8-bitowych

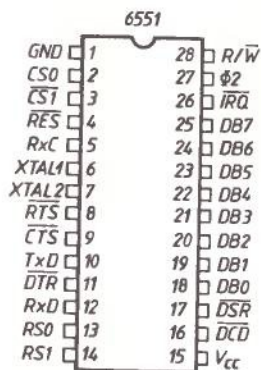
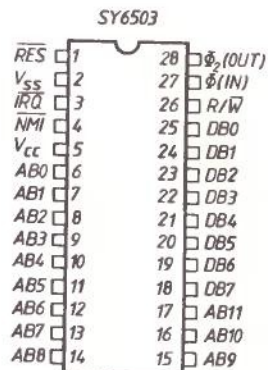
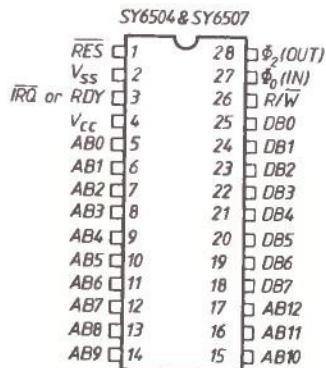
(dane wersji podstawowych są podane bez nawiasów, wersji najszybszych w nawiasach)

Procesor	6502	6800	6809	8080	Z80
Częstotliwość zegara [MHz]	1 (4)	1 (2)	1 (2)	2	2,5 (8)
Czas cyklu maszynowego [μ s]	1 (0,25)	1 (0,5)	1 (0,5)	1 ÷ 2	1,2 ÷ 1,6 (0,38 ÷ 0,5)
Wymagany czas dostępu pamięci (max) [ns]	650 (110)	530 (250)	690 (330)	200	600 (110)
Liczba rejestrów wewnętrznych/ w tym rejestrów 16-bitowych bez S i PC	3	3/1	6/3	7/3	9/5 dwa zestawy
Liczba rozkazów (mnemoników)	56	72	137	245	150
Liczba trybów adresowania	13 (15)	6	29	—	9
Długość wskaźnika stosu (bitów)	8	16	16	16	16
Maksymalna szybkość programo- wej transmisji danych z potwier- dzeniem [KB/s]	55 (222)	37 (74)	41 (83)	34	49 (147)

Dodatek 8

Rozkład wyprowadzeń wybranych układów scalonych rodziny 65XX





Literatura

1. *MCS 6500 Computer Family Hardware Manual*. MOS Technology Inc. 1976
2. *MCS 6500 Computer Family Programming Manual*. MOS Technology Inc. 1976
3. *KIM1 Microcomputer Module User Manual*. MOS Technology Inc. 1976
4. *Synertek Data Book*. SYNERTEK 1980
5. *6502 Macroassembler & text editor for PET, APPLE SYM & others*, JE&H 1979
6. Chu Y.: *Maszyny cyfrowe. Zasady projektowania*. PWN 1967
7. Flores I.: *Arytmetyka maszyn cyfrowych*. WNT 1970
8. Karpiński J., Wójcik P.: *Mikroprocesorowy system projektowy ISP*. PIE, kwiecień 1978
9. Łakomy M., Zabrodzki J.: *Cyfrowe układy scalone TTL*. PWN 1974
10. Misiurewicz P.: *Podstawy techniki cyfrowej*. WNT 1985
11. Misiurewicz P.: *Systemy mikrokomputerowe*. WSiP 1986
12. Misiurewicz P., Grzybek M.: *Półprzewodnikowe układy logiczne TTL*. WNT 1982
13. Pelka H.: *Od algebry połączeń do mikroprocesora*. WKŁ 1980
14. Pieńkos J., Turczyński I.: *Układy scalone TTL serii UCY74 i ich zastosowania*. WKŁ 1976
15. Rydzewski A., Sacha K.: *Mikrokomputer — elementy, budowa, działanie*. SIGMA 1986
16. Wagner F.: *Projektowanie urządzeń cyfrowych*. WNT 1978
17. Praca zbiorowa: *Układy mikroprocesorowe serii Intel 8080 i Motorola 6800*. Wydawnictwa Politechniki Warszawskiej 1985

Scanned by Gozdek
Classic Computers OnLine
www.cconline.com.pl

Printed in Poland

Wydawnictwo Czasopism i Książek Technicznych NOT-SIGMA
Warszawa 1989

Wydanie I. Nakład 19350 egz. Ark. wyd. 10,5. Ark. druk. 10.
Papier offset kl. III, 70g. 70 × 100 cm. Skład: Wrocławska Drukarnia
Naukowa. Druk: Olsztyńskie Zakłady Graficzne im. Seweryna
Pieniężnego, 10-417 Olsztyn. ul. Towarowa 2. U-61/30.

¹⁶⁵⁰
Cena zł 1400,—

Scanned by Gozdek
Classic Computers OnLine
www.cconline.com.pl

ISBN 83-85001-58-1