

U N I W E R S Y T E T J A G I E L L O Ń S K I
INSTYTUT INFORMATYKI

SKRYPTY UCZELNIANE

Marek Skomorowski

**PODSTAWY UKŁADÓW
CYFROWYCH**



Kraków 1996

Spis treści

Przedmowa	5
1 SYSTEMY LICZBOWE I KODY	7
1.1 Systemy liczbowe	7
1.2 Uzupełnienia	10
1.3 Liczby dwójkowe ze znakiem	13
1.4 Podsumowanie	15
2 MINIMALIZACJA FUNKCJI LOGICZNYCH	17
2.1 Elementy algebry Boole’a	17
2.2 Minimalizacja funkcji logicznych	20
2.2.1 Metoda przekształceń formalnych	20
2.2.2 Metoda tablic Karnaugh’a	21
2.2.3 Metoda Quine’a–McCluskeya	28
2.3 Podsumowanie	32
3 UKŁADY KOMBINACYJNE	34
3.1 Wstęp	34
3.2 Bramki logiczne	34
3.3 Analiza układów kombinacyjnych	40
3.4 Projektowanie układów kombinacyjnych	42
3.4.1 Układy arytmetyczne	43
3.4.2 Dekodery	46
3.4.3 Kodery	48
3.4.4 Multipleksery	49
3.5 Pamięci ROM	50
3.6 Programowalne układy logiczne PLD	51
3.6.1 Programowalne struktury logiczne PLA	53
3.6.2 Programowalne struktury logiczne PAL	54
3.7 Podsumowanie	55
4 UKŁADY SEKWENCYJNE	57
4.1 Wstęp	57
4.2 Przerzutniki	57
4.2.1 Przerzutnik <i>SR</i>	57
4.2.2 Przerzutnik <i>D</i>	58
4.2.3 Przerzutnik <i>D</i> wyzwalany zboczem	59
4.2.4 Przerzutnik <i>D–MS</i>	59
4.2.5 Przerzutnik <i>JK</i>	61
4.2.6 Tablice wzbudzeń przerzutników <i>D</i> i <i>JK</i>	62

4.2.7	Przerzutnik T	62
4.3	Analiza synchronicznych układów sekwencyjnych	63
4.4	Projektowanie synchronicznych układów sekwencyjnych	65
4.5	Rejestry	70
4.6	Liczniki	74
4.7	Podsumowanie	75
5	PROCESOR	79
5.1	Wstęp	79
5.2	Mikrooperacje przesyłania między rejestrami	80
5.3	Mikrooperacje arytmetyczne i logiczne	82
5.4	Mikrooperacje przesuwania	82
5.5	Przesyłanie za pomocą szyn	83
5.6	Przesyłanie do pamięci i z pamięci	84
5.7	Projektowanie jednostki arytmetyczno-logicznej	85
5.8	Układ przesuwania	91
5.9	Przykładowy procesor	93
5.10	Podsumowanie	95
6	JEDNOSTKA STERUJĄCA	96
6.1	Słowo sterujące	96
6.2	Sterowanie sprzętowe	98
6.3	Sterowanie mikroprogramowane	106
6.4	Podsumowanie	110
7	PRZYKŁADOWY KOMPUTER	111
7.1	Wstęp	111
7.2	Formaty rozkazów	111
7.3	Lista rozkazów	111
7.4	Wykonywanie rozkazów	112
7.5	Generowanie sygnałów taktujących	117
7.6	Projektowanie jednostki sterującej	118
7.7	Podsumowanie	120

Przedmowa

Skrypt jest materiałem pomocniczym do przedmiotu "Układy cyfrowe", w ramach programu studiów informatycznych w Uniwersytecie Jagiellońskim.

Celem skryptu jest wprowadzenie do problematyki układów cyfrowych w zakresie niezbędnym do zrozumienia budowy i działania prostego, przykładowego komputera. Wszystkie układy cyfrowe są omówione wyłącznie w aspekcie ich struktury logicznej. Technologia, budowa i parametry układów cyfrowych nie są przedmiotem rozważań skryptu. Na dobór materiału i sposób jego prezentacji decydujący wpływ miały zarówno specyfika uniwersyteckich studiów informatycznych, cel, jakiemu skrypt ma służyć, jak również ograniczona liczba godzin wykładowych (30).

Skrypt jest przeznaczony przede wszystkim dla studentów uniwersyteckich studiów informatycznych, którzy pierwszy raz stykają się z problematyką układów cyfrowych.

W rozdziałach 1 i 2 zostały podane podstawowe wiadomości na temat systemów liczbowych, algebry Boole'a i minimalizacji funkcji logicznych. W rozdziałach 3 i 4 przedstawiono podstawowe wiadomości na temat układów kombinacyjnych i synchronicznych układów sekwencyjnych. W rozdziale 5 został zaprojektowany prosty przykładowy procesor.

W rozdziale 6 podano podstawowe wiadomości na temat sterowania sprzętowego (*hard-wired control*) i mikroprogramowego (*microprogrammed control*). W rozdziale 7 został zaprojektowany prosty przykładowy komputer.

Recenzentowi skryptu, Panu prof. drowi hab. inż. Janowi Zabrodzkiemu z Instytutu Informatyki Politechniki Warszawskiej, dziękuję za uwagi, które umożliwiły usunięcie usterek maszynopisu.

Panu mgrowi Władysławowi Kaczorowskiemu z Instytutu Informatyki Uniwersytetu Jagiellońskiego dziękuję za pomoc w składaniu tekstu i wykonaniu rysunków w systemie LaTeX.

Marek Skomorowski

Kraków, październik 1996

1 SYSTEMY LICZBOWE I KODY

1.1 Systemy liczbowe

W systemie dziesiętnym ciąg cyfr 123.45 oznacza liczbę o wartości:

$$1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}$$

Oznaczając podstawę systemu liczbowego przez p , a cyfry przez a_i , można każdą liczbę N zawierającą n -cyfrową część całkowitą i m -cyfrową część ułamkową, przedstawić w postaci szeregu ([1]):

$$N = a_{n-1}p^{n-1} + a_{n-2}p^{n-2} + \dots + a_0p^0 + a_{-1}p^{-1} + a_{-2}p^{-2} + \dots$$
$$\dots + a_{-m}p^{-m} = \sum_{i=-m}^{n-1} a_i p^i \quad (1.1)$$

lub w następującej postaci:

$$N = a_{n-1}a_{n-2} \dots a_0.a_{-1}a_{-2} \dots a_{-m} \quad (1.2)$$

W wyrażeniu (1.2) kropka jest używana do oddzielenia części całkowitej od części ułamkowej.

W systemie dwójkowym, stosowanym w układach cyfrowych, podstawa jest równa 2. W systemie dwójkowym cyframi są 0 i 1. W systemie dwójkowym ciąg cyfr 1001.01 oznacza liczbę o wartości

$$1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 9.25 \quad (1.3)$$

W celu odróżnienia liczb o odmiennych podstawach stosuje się notację polegającą na ujęciu zapisu liczby w nawiasy okrągłe () oraz podaniu podstawy systemu liczbowego jako indeksu. Na przykład:

$$(1001.01)_2 = (9.25)_{10}$$

Cyframi w systemie ósemkowym są: 0, 1, 2, 3, 4, 5, 6, 7. Cyframi w systemie szesnastkowym są: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Litery A, B, C, D, E, F, używane do reprezentowania cyfr szesnastkowych, oznaczają odpowiednio liczby dziesiętne: 10, 11, 12, 13, 14, 15. W systemie ósemkowym ciąg cyfr 123.4 oznacza liczbę o wartości

$$1 \cdot 8^2 + 2 \cdot 8^1 + 3 \cdot 8^0 + 4 \cdot 8^{-1} = 83.5 \quad (1.4)$$

W systemie szesnastkowym ciąg cyfr F5A.C oznacza liczbę o wartości

$$15 \cdot 16^2 + 5 \cdot 16^1 + 10 \cdot 16^0 + 12 \cdot 16^{-1} = 3930.75 \quad (1.5)$$

Wyrażenia (1.3), (1.4), (1.5) pokazują, w jaki sposób liczby dwójkowe, ósemkowe i szesnastkowe można zamienić na równoważne im liczby dziesiętne. W podobny sposób można dokonać konwersji liczby w systemie o dowolnej podstawie p na równoważną jej liczbę dziesiętną.

Zamiany liczby dziesiętnej na równoważną jej liczbę w systemie o podstawie p dokonuje się przez podział liczby na część całkowitą i część ułamkową, a następnie dokonanie konwersji każdej części oddzielnie. Zamiany całkowitej liczby dziesiętnej na liczbę w systemie o podstawie p dokonuje się w wyniku kolejnych dzieleni tej liczby przez p i zapamiętywania kolejnych reszt. Zamiany części ułamkowej na liczbę w systemie o podstawie p dokonuje się w wyniku kolejnych mnożeń przez liczbę p i zapamiętywania otrzymanych w ten sposób cyfr części całkowitej ([2, 3]).

Przykład 1.1. Zamiany liczby dziesiętnej 25.90625 na liczbę dwójkową dokonuje się po uprzednim podzieleniu jej na część całkowitą (25) i część ułamkową (0.90625), a następnie dokonaniu konwersji każdej części oddzielnie. Część całkowitą zamienia się przez dzielenie 25 przez 2, co daje w wyniku 12 i resztę 1. Wynik 12 jest znów dzielony przez 2, co daje kolejny wynik 6 i resztę 0. I tak dalej. Proces ten kończy się wtedy, kiedy kolejny wynik z dzielenia przez 2 jest równy 0. Cyframi części całkowitej liczby dwójkowej są uzyskane kolejno reszty, przy czym pierwsza reszta jest najmniej znaczącą cyfrą liczby dwójkowej. Proces ten został pokazany poniżej.

$$\begin{array}{l} 25|2 = 12 \text{ reszta} = 1 \text{ najmniej znacząca cyfra} \\ 12|2 = 6 \text{ reszta} = 0 \\ 6|2 = 3 \text{ reszta} = 0 \\ 3|2 = 1 \text{ reszta} = 1 \\ 1|2 = 0 \text{ reszta} = 1 \text{ najbardziej znacząca cyfra} \end{array}$$

$$(25)_{10} = (11001)_2$$

Dalej będziemy korzystać ze skróconego zapisu, to znaczy:

$$\begin{array}{l|l} 25 & \\ 12 & 1 \text{ najmniej znacząca cyfra} \\ 6 & 0 \\ 3 & 0 \\ 1 & 1 \\ 0 & 1 \text{ najbardziej znacząca cyfra} \end{array}$$

$$(25)_{10} = (11001)_2$$

Część ułamkową zamienia się przez mnożenie 0.90625 przez $p = 2$, co daje część całkowitą równą 1 i część ułamkową równą 0.8125. Nowa część ułamkowa jest znowu mnożona przez 2, co daje część całkowitą równą 1 i część ułamkową równą 0.625. I tak dalej. Dalsze mnożenia kończy się wtedy, kiedy część ułamkowa jest równa 0, lub wtedy, kiedy otrzyma się żadaną liczbę cyfr w liczbie dwójkowej. Cyframi części ułamkowej liczby

dwójkowej są, wyznaczone w wyniku kolejnych mnożeń, cyfry części całkowitej. Cyfra pierwszej wyznaczonej części całkowitej jest cyfrą najbardziej znaczącą (umieszczoną zaraz po kropce). Proces ten został pokazany poniżej.

$$\begin{aligned} 0.90625 \times 2 &= 1.8125 \text{ część całkowita}=1 \text{ najbardziej znacząca} \\ 0.81250 \times 2 &= 1.6250 \text{ część całkowita}=1 \text{ cyfra} \\ 0.62500 \times 2 &= 1.2500 \text{ część całkowita}=1 \\ 0.25000 \times 2 &= 0.5000 \text{ część całkowita}=0 \\ 0.50000 \times 2 &= 1.0000 \text{ część całkowita}=1 \text{ najmniej znacząca cyfra} \end{aligned}$$

$$(0.90625)_{10} = (0.11101)_2$$

Ostatecznie otrzymujemy

$$(25.90625)_{10} = (11001.11101)_2$$

Przykład 1.2. Zamiany ułamka dziesiętnego 0.345 na ułamek dwójkowy dokonuje się w następujący sposób:

$$\begin{aligned} 0.345 \times 2 &= 0.69 \text{ część całkowita} = 0 \text{ najbardziej znacząca cyfra} \\ 0.690 \times 2 &= 1.38 \text{ część całkowita} = 1 \\ 0.380 \times 2 &= 0.76 \text{ część całkowita} = 0 \\ 0.760 \times 2 &= 1.52 \text{ część całkowita} = 1 \\ 0.520 \times 2 &= 1.04 \text{ część całkowita} = 1 \\ 0.040 \times 2 &= 0.08 \text{ część całkowita} = 0 \end{aligned}$$

W tym przypadku proces mnożenia kolejnych części ułamkowych kończymy wtedy, kiedy liczba wyznaczonych cyfr daje wymaganą dokładność, na przykład:

$$(0.345)_{10} = (0.010110)_2$$

Przykład 1.3. Zamiany liczby dziesiętnej 1510.90625 na liczbę ósemkową dokonuje się po uprzednim podzieleniu jej na część całkowitą (1510) i część ułamkową (0.90625) oraz w wyniku dokonania konwersji każdej części oddzielnie:

$$\begin{array}{r|l} 1510 & \\ 188 & 6 \\ 23 & 4 \\ 2 & 7 \\ 0 & 2 \end{array} \quad (1510)_{10} = (2746)_8$$

$$\begin{aligned} 0.90625 \times 8 &= 7.25 \text{ część całkowita} = 7 \\ 0.25000 \times 8 &= 2.00 \text{ część całkowita} = 2 \end{aligned}$$

$$(0.90625)_{10} = (0.72)_8$$

Ostatecznie otrzymujemy

$$(1510.90625)_{10} = (2746.72)_8$$

Przykład 1.4. Zamiany liczby dziesiętnej 1951.65625 na liczbę szesnastkową dokonuje się po uprzednim podzieleniu jej na część całkowitą (1951) i część ułamkową (0.65625) oraz w wyniku dokonania konwersji każdej części oddzielnie:

$$\begin{array}{r|l} 1951 & \\ 121 & (15)_{10} = (F)_{16} \\ 7 & 9 \\ 0 & 7 \end{array} \quad (1951)_{10} = (79F)_{16}$$

$$\begin{aligned} 0.65625 \times 16 &= 10.5 & \text{część całkowita} &= (10)_{10} = (A)_{16} \\ 0.50000 \times 16 &= 8.00 & \text{część całkowita} &= 8 \end{aligned}$$

$$(0.65625)_{10} = (0.A8)_{16}$$

Ostatecznie otrzymujemy

$$(1951.65625)_{10} = (79F.A8)_{16}$$

Liczba dwójkowa może być bezpośrednio zamieniona na liczbę ósemkową. W tym celu należy podzielić ją na grupy 3-bitowe, poczynając od kropki w lewo i w prawo, a następnie zastąpić otrzymane grupy odpowiadającymi im cyframi ósemkowymi. Na przykład:

$$(010\ 111\ 100\ 110.111\ 010)_2 = (2746.72)_8$$

Liczba dwójkowa może być bezpośrednio zamieniona na liczbę szesnastkową. W tym celu należy podzielić ją na grupy 4-bitowe, poczynając od kropki w lewo i w prawo, a następnie zastąpić otrzymane grupy odpowiadającymi im cyframi szesnastkowymi. Na przykład:

$$(0111\ 1001\ 1111.1010\ 1000)_2 = (79F.A8)_{16}$$

1.2 Uzupełnienia

Uzupełnień używa się w układach cyfrowych do przedstawiania liczb ujemnych w celu uproszczenia operacji odejmowania ([2, 3, 4]). Istnieją dwa rodzaje uzupełnień dla każdego systemu liczbowego o podstawie p :

- uzupełnienie do $(p - 1)$,
- uzupełnienie do p .

Uzupełnienie do $(p - 1)$ n -cyfrowej liczby N o podstawie p jest zdefiniowane jako

$$(p^n - 1) - N \quad (1.6)$$

Dla liczb dziesiętnych $(p - 1) = 9$, dla dwójkowych $(p - 1) = 1$.

Przykład 1.5. Uzupełnieniem do 9 liczby $(3456)_{10}$ jest

$$(p^n - 1) - N = (10^4 - 1) - 3456 = 9999 - 3456 = 6543$$

Przykład 1.6. Uzupełnieniem do 1 liczby $(1001)_2$ jest

$$(p^n - 1) - N = (2^4 - 1) - 1001 = 1111 - 1001 = 0110$$

Uzupełnienie do 1 liczby dwójkowej uzyskuje się w wyniku odjęcia każdej jej cyfry od 1. Odejmując cyfry dwójkowe od 1 otrzymujemy $1 - 0 = 1$ lub $1 - 1 = 0$. W obu przypadkach cyfra otrzymana w wyniku odjęcia od 1 jest negacją jej wartości. Stąd, w celu otrzymania uzupełnienia do 1 danej liczby dwójkowej należy zanegować wszystkie jej cyfry.

Przykład 1.7. Uzupełnieniem do 1 liczby dwójkowej 10101001 jest 01010110.

Uzupełnienie do p dla n -cyfrowej liczby N o podstawie p jest zdefiniowane jako

$$p^n - N \quad (1.7)$$

Wyrażenie (1.7) możemy zapisać w postaci

$$p^n - N = [(p^n - 1) - N] + 1 \quad (1.8)$$

Porównując (1.6), (1.7) i (1.8) można zauważyć, że uzupełnienie do p liczby N otrzymujemy przez dodanie 1 do uzupełnienia do $(p - 1)$ liczby N .

Uzupełnienie do 2 można również otrzymać, pozostawiając, od prawej strony do lewej, wszystkie zera i pierwszą jedynekę bez zmian i negując pozostałe cyfry.

Przykład 1.8. Pozostawiając w liczbie 10010100, od prawej strony do lewej, wszystkie zera i pierwszą jedynekę bez zmian, a następnie negując pozostałe cyfry

$$\begin{array}{c|c} 10010 & 100 \\ \text{negacja} & \text{bez zmian} \end{array}$$

otrzymujemy jej uzupełnienie do 2 równe 01101100.

Odejmowanie dwóch liczb n -bitowych bez znaku, $M - N$, w systemie o podstawie p może być wykonane w następujący sposób:

1. Do odjemnej M należy dodać uzupełnienie do p odjemnika N

$$M + (p^n - N) = M - N + p^n. \quad (1.9)$$

2. Jeżeli $M \geq N$, w wyrażeniu (1.9) należy pominąć p^n , co daje wynik $M - N$.

3. Jeżeli $M < N$, wyrażenie (1.9) można zapisać w postaci:

$$M + (p^n - N) = p^n - (N - M) \quad (1.10)$$

będącej uzupełnieniem do p różnicy $(N - M)$.

Biorąc uzupełnienie do p wyrażenia (1.10) i poprzedzając je znakiem minus otrzymujemy:

$$- \{p^n - [p^n - (N - M)]\} = -(N - M) = M - N$$

Przykład 1.9. Niech $M = (1987)_{10}$ i $N = (1958)_{10}$. W tym przypadku $p = 10$, $n = 4$, $M > N$, a różnicę $(M - N)$ otrzymujemy w następujący sposób:

$M =$	1987
$p^n - N = 10^4 - 1958 =$	+ 8042
suma=	1 0029
$p^n = 10^4$	- 1 0000
wynik=	0 0029

Przykład 1.10. Niech $M = (1958)_{10}$ i $N = (1958)_{10}$. W tym przypadku $p = 10$, $n = 4$, $M = N$, a różnicę $(M - N)$ otrzymujemy w następujący sposób:

$M =$	1958
$p^n - N = 10^4 - 1958 =$	+ 8042
suma=	1 0000
$p^n = 10^4$	- 1 0000
wynik=	0 0000

Przykład 1.11. Niech $M = (1958)_{10}$ i $N = (1987)_{10}$. W tym przypadku $p = 10$, $n = 4$, $M < N$, a różnicę $(M - N)$ otrzymujemy w następujący sposób:

$M =$	1958
$p^n - N = 10^4 - 1987 =$	+ 8013
suma=	9971

$$-(p^n - \text{suma}) = -(10^4 - 9971) = -29$$

Przykład 1.12. Niech $M = (11001)_2$ i $N = (1010)_2$. W tym przypadku $p = 2$, $n = 5$, $M > N$, a różnicę $(M - N)$ otrzymujemy w następujący sposób:

$$\begin{array}{r} M = 11001 \\ p^n - N = 2^5 - 1010 = 10110 \\ \hline \text{suma} = 110111 \\ p^n = 2^5 100000 \\ \hline \text{wynik} = 01111 \end{array}$$

Przykład 1.13. Niech $M = (1010)_2$ i $N = (1010)_2$. W tym przypadku $p = 2$, $n = 4$, $M = N$, a różnicę $(M - N)$ otrzymujemy w następujący sposób:

$$\begin{array}{r} M = 1010 \\ p^n - N = 2^4 - 1010 = 0110 \\ \hline \text{suma} = 10000 \\ p^n = 2^4 10000 \\ \hline \text{wynik} = 00000 \end{array}$$

Przykład 1.14. Niech $M = (1010)_2$ i $N = (11001)_2$. W tym przypadku $p = 2$, $n = 5$, $M < N$, a różnicę $(M - N)$ otrzymujemy w następujący sposób:

$$\begin{array}{r} M = 01010 \\ p^n - N = 2^5 - 11001 = 00111 \\ \hline \text{suma} = 10001 \\ -(p^n - \text{suma}) = -(2^5 - 10001) = -01111 \end{array}$$

1.3 Liczby dwójkowe ze znakiem

Przyjęto, że plus jest reprezentowany przez 0, minus natomiast przez 1 ($\{1, 2, 3, 4\}$). Zatem do przedstawienia n -bitowej liczby dwójkowej z uwzględnieniem jej znaku potrzeba $n + 1$ bitów. Liczby dwójkowe ze znakiem mogą być reprezentowane w następujący sposób:

1. znak-moduł,
2. znak-uzupełnienie do 1,
3. znak-uzupełnienie do 2.

Zapis liczby dodatniej jest taki sam w każdym z wymienionych sposobów. W zapisie znak-moduł liczby ujemnej wartość bezwzględna liczby następuje bezpośrednio po ujemnym znaku. W pozostałych dwóch zapisach liczba ujemna jest reprezentowana przez

uzupełnienie do 1 lub do 2 jej wartości. Na rysunku 1.1 są przedstawione 4-bitowe liczby dwójkowe ze znakiem.

Zapis dziesiętny	Zapis znak- -uzupełnienie do 2	Zapis znak- -uzupełnienie do 1	Zapis znak-moduł
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	–	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011

Rys. 1.1. Liczby dwójkowe ze znakiem

Zaletą zapisu znak-uzupełnienie do 2, w porównaniu z pozostałymi zapisami, jest istnienie tylko jednej reprezentacji zera (rys. 1.1).

Przykład 1.15. Na rysunku 1.2 zostały podane przykłady dodawania dwóch 8-bitowych liczb dwójkowych ze znakiem (i ich równoważników dziesiętnych), przy czym liczby ujemne są przedstawione w zapisie znak-uzupełnienie do 2.

+5	00000101	-5	11111011
+9	00001001	+9	00001001
<u>+14</u>	<u>00001110</u>	<u>+4</u>	<u>11111000</u>
+5	00000101	-5	11111011
-9	11110111	-9	11110111
<u>-4</u>	<u>11111100</u>	<u>-14</u>	<u>11110010</u>

Rys. 1.2. Dodawanie liczb dwójkowych ze znakiem

W podanych przykładach bit przeniesienia z najstarszej pozycji jest pomijany, a otrzymany wynik, w przypadku gdy jest ujemny, jest przedstawiony w zapisie znak-uzupełnienie do 2.

Jeżeli dodając dwie n -bitowe liczby bez znaku otrzymujemy wynik na $n + 1$ bitach, to mówimy, że wystąpił nadmiar lub przepełnienie (*overflow*). Pojęcie nadmiaru wyjaśnimy na następującym przykładzie.

Przykład 1.16

$$\begin{array}{r}
 150 \quad \vdots 10010110 \\
 + 190 \quad \vdots 10111110 \\
 \hline
 340 \quad \vdots 101010100
 \end{array}$$

W przykładzie tym przyjęty format danych obejmuje 8 bitów (liczby bez znaku). Zakres liczb bez znaku zapisanych na ośmiu bitach obejmuje liczby od 0 do 255. Wynik dodawania $150 + 190 = 340$ wykracza poza ten zakres. Bit przeniesienia z najbardziej znaczącej pozycji wykracza poza przyjęty format danych i jest bitem nadmiaru.

W przypadku liczb ze znakiem nadmiar może wystąpić wtedy, kiedy dodajemy dwie liczby dodatnie lub ujemne. Ilustruje to następujący przykład.

Przykład 1.17

$$\begin{array}{r}
 50 \quad 00110010 \\
 + 90 \quad 01011010 \\
 \hline
 140 \quad 10001000
 \end{array}
 \qquad
 \begin{array}{r}
 -50 \quad 11001110 \\
 - 90 \quad 10100110 \\
 \hline
 -140 \quad 101110110
 \end{array}$$

W przykładzie tym przyjęty format danych obejmuje 8 bitów (liczby ze znakiem). Najbardziej znaczący bit jest bitem znaku. Zakres liczb ze znakiem zapisanych na ośmiu bitach obejmuje liczby od $+127$ do -128 . Widzimy, że zapisany na ośmiu bitach wynik dodawania liczb dodatnich $50 + 90 = 140$ ma znak ujemny (1 na bicie znaku). Błąd ten spowodowany jest tym, że liczba $50 + 90 = 140$ wykracza poza zakres od $+127$ do -128 . Widzimy również, że zapisany na ośmiu bitach wynik dodawania liczb ujemnych $-50 + (-90) = -140$ ma znak dodatni (0 na bicie znaku). Błąd ten spowodowany jest tym, że liczba $-50 + (-90) = -140$ wykracza poza zakres od $+127$ do -128 .

1.4 Podsumowanie

Systemy liczbowe, kody i arytmetyka binarna zostały przedstawione w rozdziale 1 w zakresie niezbędnym do zrozumienia materiału prezentowanego w dalszej części skryptu.

Systemy liczbowe, kody i arytmetyka binarna stanowią materiał na odrębny wykład. Wiadomości na temat systemów liczbowych, kodów i algorytmów stosowanych w operacjach arytmetycznych na liczbach stałoprzecinkowych i zmiennoprzecinkowych są obszernie przedstawione, na przykład w [5, 6].

Literatura

- [1] Kalisz J.: *Podstawy elektroniki cyfrowej*, WKŁ, 1993.
- [2] Mano M.M.: *Architektura komputerów*, WNT, 1980.

- [3] Mano M.M.: *Computer engineering: hardware design*, Prentice-Hall, 1988.
- [4] Mano M.M.: *Computer system architecture*, Prentice-Hall, 1993.
- [5] Flores I.: *Arytmetyka maszyn cyfrowych*, WNT, 1970.
- [6] Biernat J.: *Arytmetyka komputerów*, PWN, 1996.

2 MINIMALIZACJA FUNKCJI LOGICZNYCH

2.1 Elementy algebry Boole'a

Algebra Boole'a jest działem matematyki wykorzystywanym do projektowania i analizy układów cyfrowych. Zmienne boolowskie (logiczne) mogą przyjmować wartości tylko ze zbioru $\{0, 1\}$. W algebrze Boole'a do przedstawienia iloczynu logicznego AND zmiennych X i Y będziemy stosować zapis $X \cdot Y = XY$. Do przedstawienia sumy logicznej zmiennych X i Y będziemy stosować zapis $X + Y$. Negację zmiennej X będziemy zapisywać jako \overline{X} . Na rysunku 2.1 są przedstawione (za pomocą tablicy prawdy) definicje następujących funkcji logicznych: AND, OR i NOT.

AND			OR			NOT	
X	Y	$X \cdot Y$	X	Y	$X + Y$	X	\overline{X}
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Rys. 2.1. Tablice prawdy funkcji AND, OR, NOT

Funkcjami boolowskimi (funkcjami logicznymi) nazywamy funkcje, których zmienne i one same przyjmują wartości tylko ze zbioru $\{0, 1\}$. Rozważmy funkcję logiczną

$$f(x, y, z) = xy + xz + yz$$

Zależność między wartością funkcji a wartościami jej zmiennych można przedstawić w tablicy prawdy pokazanej na rysunku 2.2.

Algebra Boole'a ([1, 2, 3, 4]) jest algebrą aksjomatyczną, przyjmującą aksjomaty przedstawione na rysunku 2.3. W algebrze Boole'a obowiązuje tzw. zasada dualizmu. Zasada ta może być sformułowana w następujący sposób ([2]): zastępując w dowolnej tożsamości algebry Boole'a symbol OR (+) symbolem AND (\cdot) i symbol AND (\cdot) symbolem OR (+) oraz zastępując jedynkę zerem, a zero jedynką (jeśli występują w wyrażeniu), otrzymamy również tożsamość (rys. 2.3). Własności (16) i (17) są nazywane prawami de Morgana. Prawa de Morgana uogólnione na n zmiennych przyjmują postać:

$$\overline{x_1 + x_2 + \dots + x_n} = \overline{x_1} \cdot \overline{x_2} \cdot \dots \cdot \overline{x_n}$$

$$\overline{x_1 \cdot x_2 \cdot \dots \cdot x_n} = \overline{x_1} + \overline{x_2} + \dots + \overline{x_n}$$

Funkcje logiczne mogą być przedstawione w postaci opisu słownego, tablic prawdy i w sposób analityczny.

Funkcja n zmiennych jest przedstawiana za pomocą tablicy prawdy o $n + 1$ kolumnach i 2^n wierszach. W kolejnych wierszach wpisuje się wszystkie kombinacje zmiennych niezależnych. Ostatnia kolumna jest przeznaczona do zapisania wartości funkcji dla poszczególnych kombinacji zmiennych niezależnych. Aby wszystkie możliwe kombinacje zostały uwzględnione, wpisujemy je w taki sposób, by tworzyły kolejne liczby (rys. 2.2).

i	x	y	z	$f(x, y, z) = xy + xz + yz$
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Rys. 2.2. Tablica prawdy funkcji $f(x, y, z) = xy + xz + yz$

1. $x + 0 = x$	2. $x \cdot 1 = x$
3. $x + 1 = 1$	4. $x \cdot 0 = 0$
5. $x + x = x$	6. $x \cdot x = x$
7. $x + \bar{x} = 1$	8. $x \cdot \bar{x} = 0$
9. $\bar{\bar{x}} = x$	
10. $x + y = y + x$	11. $x \cdot y = y \cdot x$
12. $x + (y + z) = (x + y) + z$	13. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
14. $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$	15. $x + (y \cdot z) = (x + y) \cdot (x + z)$
16. $\overline{x + y} = \bar{x} \cdot \bar{y}$	17. $\overline{x \cdot y} = \bar{x} + \bar{y}$

Rys. 2.3. Aksjomaty algebry Boole'a

Definicja 2.1. Iloczynem pełnym n zmiennych nazywamy taki iloczyn tych zmiennych (lub ich negacji), w którym każda zmienna (lub jej negacja) występuje dokładnie jeden raz.

Definicja 2.2. Sumą pełną n zmiennych nazywamy taką sumę tych zmiennych (lub ich negacji), w której każda zmienna (lub jej negacja) występuje dokładnie jeden raz.

Na rysunku 2.4 są przedstawione iloczyny pełne m_i i sumy pełne M_i dla trzech zmiennych x, y, z .

x	y	z	Iloczyny pełne m_i	Sumy pełne M_i
0	0	0	$\bar{x}\bar{y}\bar{z}$ m_0	$x + y + z$ M_0
0	0	1	$\bar{x}\bar{y}z$ m_1	$x + y + \bar{z}$ M_1
0	1	0	$\bar{x}y\bar{z}$ m_2	$x + \bar{y} + z$ M_2
0	1	1	$\bar{x}yz$ m_3	$x + \bar{y} + \bar{z}$ M_3
1	0	0	$x\bar{y}\bar{z}$ m_4	$\bar{x} + y + z$ M_4
1	0	1	$x\bar{y}z$ m_5	$\bar{x} + y + \bar{z}$ M_5
1	1	0	$xy\bar{z}$ m_6	$\bar{x} + \bar{y} + z$ M_6
1	1	1	xyz m_7	$\bar{x} + \bar{y} + \bar{z}$ M_7

Rys. 2.4. Iloczyny i sumy pełne dla trzech zmiennych x, y, z

Iloczyn pełny przyjmuje wartość 1 tylko dla jednej kombinacji wartości zmiennych. Suma pełna przyjmuje wartość 0 tylko dla jednej kombinacji wartości zmiennych.

Twierdzenie 2.1. Dowolną funkcję logiczną n zmiennych można jednoznacznie przedstawić w postaci sumy iloczynów pełnych:

$$f(x_1, x_2, \dots, x_n) = \sum_{i=0}^{2^n-1} \alpha_i m_i, \quad \alpha_i = f(i)$$

gdzie $f(i)$ oznacza wartość funkcji $f(x_1, x_2, \dots, x_n)$ dla i -tej kombinacji zmiennych.

Twierdzenie 2.2. Dowolną funkcję logiczną n zmiennych można jednoznacznie przedstawić w postaci iloczynu sum pełnych:

$$f(x_1, x_2, \dots, x_n) = \prod_{i=0}^{2^n-1} (\alpha_i + M_i), \quad \alpha_i = f(i)$$

gdzie $f(i)$ oznacza wartość funkcji $f(x_1, x_2, \dots, x_n)$ dla i -tej kombinacji zmiennych.

Dowody tych twierdzeń są przedstawione między innymi w [5].

Przykład 2.1. Przedstawić funkcję $f(x, y, z)$ zadaną przez tablicę prawdy na rysunku 2.2 w postaci sumy iloczynów pełnych. Jak wynika z tablicy prawdy, funkcja ta przyjmuje wartość 1 w wierszach: 3, 5, 6 i 7, a wartość 0 w pozostałych. Zatem

$$\begin{aligned} f(x, y, z) &= \sum_{i=0}^{2^n-1} \alpha_i m_i = 0 \cdot m_0 + 0 \cdot m_1 + 0 \cdot m_2 + \\ &+ 1 \cdot m_3 + 0 \cdot m_4 + 1 \cdot m_5 + 1 \cdot m_6 + 1 \cdot m_7 \end{aligned}$$

Każdy składnik $0 \cdot m_i = 0$ może być wyeliminowany. Zatem funkcja przyjmuje postać

$$f(x, y, z) = m_3 + m_5 + m_6 + m_7 = \\ = \overline{x}yz + x\overline{y}z + xy\overline{z} + xyz$$

Przykład 2.2. Przedstawić funkcję $f(x, y, z)$ zadaną przez tablicę prawdy na rysunku 2.2 w postaci iloczynu sum pełnych. Jak wynika z tablicy prawdy, funkcja ta przyjmuje wartość 0 w wierszach: 0, 1, 2, 4, a wartość 1 w pozostałych. Zatem

$$f(x, y, z) = \prod_{i=0}^{2^n-1} (\alpha_i + M_i) = (0 + M_0) \cdot (0 + M_1) \cdot (0 + M_2) \cdot \\ \cdot (1 + M_3) \cdot (0 + M_4) \cdot (1 + M_5) \cdot (1 + M_6) \cdot (1 + M_7)$$

Każdy czynnik $(1 + M_i) = 1$ może być wyeliminowany. Zatem funkcja przyjmuje postać

$$f(x, y, z) = M_0 \cdot M_1 \cdot M_2 \cdot M_4 = \\ = (x + y + z)(x + y + \overline{z})(x + \overline{y} + z)(\overline{x} + y + z)$$

Funkcja NAND (Not AND) jest negacją funkcji AND. Funkcja NOR (Not OR) jest negacją funkcji OR. Tablice prawdy funkcji NAND i NOR (dla dwóch zmiennych) zostały przedstawione na rysunku 2.5.

NAND			NOR		
x	y	$\overline{x \cdot y}$	x	y	$\overline{x + y}$
0	0	1	0	0	1
0	1	1	0	1	0
1	0	1	1	0	0
1	1	0	1	1	0

Rys. 2.5. Tablice prawdy funkcji NAND i NOR

Systemem funkcjonalnie pełnym nazywamy taki zbiór funkcji logicznych, który umożliwia przedstawienie dowolnej funkcji logicznej ([5]). Zbiór zawierający funkcje: AND, OR, NOT tworzy podstawowy system funkcjonalnie pełny. Jednoelementowe zbiory zawierające funkcje NAND lub NOR są systemami funkcjonalnie pełnymi.

2.2 Minimalizacja funkcji logicznych

2.2.1 Metoda przekształceń formalnych

Funkcje logiczne można upraszczać korzystając z podstawowych własności algebry Boole'a (rys. 2.3).

Przykład 2.3. Korzystając z podstawowych własności algebry Boole’a, uprościmy funkcję: $f(x, y, z) = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$

$$\begin{aligned} f(x, y, z) &= \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz = \\ &= \bar{x}yz + xyz + x\bar{y}z + xyz + xy\bar{z} + xyz = \\ &= yz(\bar{x} + x) + xz(\bar{y} + y) + xy(\bar{z} + z) = yz + xz + xy \end{aligned}$$

2.2.2 Metoda tablic Karnaugh

Funkcja n zmiennych przedstawiona za pomocą tablicy prawdy ma 2^n iloczynów pełnych równoważnych 2^n liczbom dwójkowym otrzymanym z n cyfr. Funkcja logiczna dla pewnych iloczynów pełnych jest równa 1, dla innych jest równa 0. Informację zawartą w tablicy prawdy można przedstawić podając dziesiętny odpowiednik tych iloczynów pełnych, dla których funkcja przyjmuje wartość 1. Na przykład tablicę prawdy funkcji z rysunku 2.2 można przedstawić w następujący sposób:

$$f(x, y, z) = \sum(3, 5, 6, 7)$$

Liczby w nawiasie przedstawiają zmienne dwójkowe w kolejności ich pojawiania się w tablicy prawdy. Symbol \sum oznacza sumę iloczynów pełnych wyszczególnionych w nawiasie. Iloczyny pełne, dla których funkcja przyjmuje wartość 1, są podane za pomocą ich dziesiętnych odpowiedników. Brakujące iloczyny pełne to te, dla których funkcja przyjmuje wartość 0.

Tablica Karnaugh ([1, 2, 3, 4, 5, 6]) jest tablicą składającą się z kwadratów, przy czym każdemu kwadratowi odpowiada jeden iloczyn pełny. Liczba kwadratów w tablicy n zmiennych jest równa 2^n . Liczby odpowiadające iloczynom pełnym wpisuje się do tablicy w taki sposób, aby iloczyny pełne w sąsiednich (mających wspólną krawędź) kwadratach różniły się od siebie tylko jedną zmienną (na jednej pozycji). Porządek taki jest charakterystyczną właściwością tablicy Karnaugh, wykorzystywaną do przeprowadzenia uproszczeń na podstawie zależności:

$$xy + x\bar{y} = x(y + \bar{y}) = x$$

gdzie x, y są zmiennymi logicznymi. Zatem zmienną, która w sąsiednich kwadratach przyjmuje różne wartości, można pominąć. Tablice Karnaugh dla funkcji dwóch, trzech i czterech zmiennych są pokazane odpowiednio na rysunkach 2.6, 2.7, 2.8.

	y	
	00	01
x {	10	11

	y	
	0	1
x {	2	3

	y	
	$\bar{x}\bar{y}$	$\bar{x}y$
x {	$x\bar{y}$	xy

Rys. 2.6. Tablica Karnaugh dla funkcji dwóch zmiennych (x, y)

		y			
		000	001	011	010
x		100	101	111	110
	z				

		y			
		0	1	3	2
x		4	5	7	6
	z				

		y			
		$\bar{x}\bar{y}\bar{z}$	$\bar{x}\bar{y}z$	$\bar{x}yz$	$\bar{x}y\bar{z}$
x		$x\bar{y}\bar{z}$	$x\bar{y}z$	xyz	$xy\bar{z}$
	z				

Rys. 2.7. Tablica Karnaugh dla funkcji trzech zmiennych
(x, y, z)

Iloczyn pełne z sąsiednich kwadratów są identyczne, z wyjątkiem jednej zmiennej. Zgodnie z tą definicją sąsiedztwa, kwadraty znajdujące się na końcach tego samego rzędu też uważa się za sąsiednie. Podobnie, kwadraty znajdujące się na końcach tej samej kolumny również uważa się za sąsiednie.

		y			
		0000	0001	0011	0010
w		0100	0101	0111	0110
		1100	1101	1111	1110
		1000	1001	1011	1010
	z				

		y			
		0	1	3	2
w		4	5	7	6
		12	13	15	14
		8	9	11	10
	z				

Rys. 2.8. Tablica Karnaugh dla funkcji czterech zmiennych
(w, x, y, z)

W procesie minimalizacji funkcję logiczną opisaną za pomocą tablicy prawdy przedstawia się w postaci tablicy Karnaugh, wpisując 1 w te kwadraty, dla których wartość funkcji jest równa 1.

Liczba kwadratów łączonych w grupy musi być równa całkowitej potęgze 2. Każdej grupie kwadratów odpowiada jedno wyrażenie, a suma logiczna tych wyrażeń jest uproszczonym przedstawieniem funkcji.

Przykład 2.4. Uprościć funkcję $f(x, y, z) = \sum(2, 4, 5, 6, 7)$, korzystając z tablicy Karnaugh.

Tablica Karnaugh dla tej funkcji jest pokazana na rysunku 2.9.

			1
1	1	1	1

Rys. 2.9. Tablica Karnaugh dla funkcji

$$f(x, y, z) = \sum(2, 4, 5, 6, 7)$$

W tablicy tej jest 5 kwadratów z wpisaną jedynką, po jednej dla każdego iloczynu pełnego, dla którego funkcja przyjmuje wartość 1. W kolumnie 4 zostały połączone dwa sąsiednie kwadraty. Kolumna ta należy zarówno do y , jak i do \bar{y} , zatem odpowiadające jej wyrażenie jest równe $y\bar{y}$. W wierszu drugim zostały połączone 4 sąsiednie kwadraty. Wiersz ten należy do x , zatem odpowiadające mu wyrażenie jest równe x . Uproszczona funkcja jest zatem równa

$$f(x, y, z) = \sum(2, 4, 5, 6, 7) = x + y\bar{y}$$

Przykład 2.5. Uprościć funkcję $f(x, y, z) = \sum(1, 4, 5, 6)$, korzystając z tablicy Karnaugh.

Tablica Karnaugh dla tej funkcji jest pokazana na rysunku 2.10.

	1		
1	1		1

Rys. 2.10. Tablica Karnaugh dla funkcji

$$f(x, y, z) = \sum(1, 4, 5, 6)$$

W kolumnie 2 zostały połączone dwa sąsiednie kwadraty, dając wyrażenie $\bar{y}z$. Dwa kwadraty z 1 na obu końcach drugiego wiersza są sąsiednie i zostały połączone, dając wyrażenie $x\bar{z}$. Uproszczona funkcja jest zatem równa

$$f(x, y, z) = \sum(1, 4, 5, 6) = x\bar{z} + \bar{y}z$$

Przykład 2.6. Uprościć funkcję $f(x, y, z) = \sum(0, 2, 4, 6, 7)$, korzystając z tablicy Karnaugh.

Tablica Karnaugh dla tej funkcji jest pokazana na rysunku 2.11.

				y
	1		1	
x	1		1	
		1	1	z

Rys. 2.11. Tablica Karnaugh dla funkcji

$$f(x, y, z) = \sum(0, 2, 4, 6, 7)$$

Cztery kwadraty w pierwszej i czwartej kolumnie są sąsiednie i zostały połączone, dając wyrażenie \bar{z} . Kwadraty odpowiadające iloczynom pełnym 6 i 7 zostały połączone, dając wyrażenie xy . Uproszczona funkcja jest zatem równa

$$f(x, y, z) = \sum(0, 2, 4, 6, 7) = xy + \bar{z}$$

Otrzymane w poprzednich przykładach funkcje miały postać sumy iloczynów. Czasami jest wygodnie przedstawić funkcję w postaci iloczynu sum. Jedynki w tablicy Karnaugh przedstawiają iloczyny pełne, dla których wartość funkcji f jest równa 1. Kwadraty nie zaznaczone jedynkami określają iloczyny pełne, dla których wartość funkcji f jest równa 0. Wpisując w puste kwadraty zera i łącząc je w grupy sąsiednich kwadratów otrzymujemy negację funkcji f , to znaczy \bar{f} . W dalszym ciągu, korzystając z praw de Morgana i negując \bar{f} , otrzymujemy $\bar{\bar{f}} = f$. Otrzymana w ten sposób funkcja (f) przyjmuje postać iloczynu sum.

Przykład 2.7. Uprościć funkcję

$$f(a, b, c, d) = \sum(2, 3, 8, 10, 11, 12, 14, 15)$$

do postaci sumy iloczynów i do postaci iloczynu sum, korzystając z tablicy Karnaugh.

Tablica Karnaugh dla tej funkcji jest pokazana na rysunku 2.12.

		c	
		1	1
		0	0
a	b	1	0
		1	0
		d	

		c	
		0	0
		1	1
a	b	0	0
		1	1
		d	

Rys. 2.12. Tablica Karnaugh dla funkcji

$$f(a, b, c, d) = \sum(2, 3, 8, 10, 11, 12, 14, 15)$$

Jedynki w tej tablicy odpowiadają iloczynom pełnym, dla których funkcja f przyjmuje wartość 1. Kwadraty z zerami odpowiadają iloczynom pełnym nie zawartym w f i określają dopełnienie funkcji f , to znaczy \bar{f} . Łącząc sąsiednie kwadraty z jedynkami otrzymujemy uproszczoną funkcję w postaci sumy iloczynów

$$f(a, b, c, d) = \sum(2, 3, 8, 10, 11, 12, 14, 15) = ac + a\bar{d} + \bar{b}c$$

Łącząc sąsiednie kwadraty z zerami, otrzymujemy uproszczoną postać negacji funkcji f , to znaczy

$$\bar{f} = \bar{a}b + \bar{c}d + \bar{a}\bar{c}$$

Korzystając z prawa de Morgana, otrzymujemy uproszczoną postać funkcji w postaci iloczynu sum

$$f = \bar{\bar{f}} = \overline{\bar{a}b + \bar{c}d + \bar{a}\bar{c}} = \overline{\bar{a}b} \cdot \overline{\bar{c}d} \cdot \overline{\bar{a}\bar{c}} = (a + \bar{b}) \cdot (c + \bar{d}) \cdot (a + c)$$

Jedynki w tablicy Karnaugh reprezentują iloczyny pełne, dla których funkcja przyjmuje wartość 1. Zera w tablicy Karnaugh reprezentują iloczyny pełne, dla których funkcja przyjmuje wartość 0. Są jednak sytuacje, kiedy nie ma znaczenia, czy dla danego iloczynu pełnego funkcja przyjmuje wartość 0, czy 1. Iloczynny pełny, dla których wartość funkcji może być dowolna (0 lub 1), nazywa się iloczynami pełnymi nieokreślonymi i oznacza się w tablicy Karnaugh na przykład przez x. Iloczynny pełny nieokreślony mogą być używane do upraszczania funkcji. Wybierając dla danej funkcji sąsiednie kwadraty w tablicy Karnaugh, iloczynom pełnym nieokreślonym można przypisać wartość 0 lub 1, w zależności od tego, czy umożliwi to uproszczenie funkcji.

Przykład 2.8. Uprościć funkcję

$$f(x, y, z) = \sum(0, 4, 6)$$

nieokreślona dla następujących iloczynów pełnych:

$$n(x, y, z) = \sum n(1, 3, 5, 7),$$

korzystając z tablic Karnaugh'a.

Powyższą funkcję można przedstawić w postaci

$$f(x, y, z) = \sum (0, 4, 6) + \sum n(1, 3, 5, 7)$$

Tablica Karnaugh'a dla tej funkcji jest pokazana na rysunku 2.13.

		y	
x	1	x	x
	1	x	x
		z	

Rys. 2.13. Tablica Karnaugh'a dla funkcji

$$f(x, y, z) = \sum (0, 4, 6) + \sum n(1, 3, 5, 7)$$

Dla iloczynów pełnych nieokreślonych $\sum n(1, 3, 5, 7)$, oznaczonych przez x, funkcja może przyjmować wartość 0 lub 1. Przyjęcie wartości 1 dla iloczynów pełnych nieokreślonych 1, 5, 7 umożliwia uproszczenie funkcji. Kwadraty 1, 5, 7 zostały połączone z sąsiednimi tak, aby uzyskać maksymalną liczbę kwadratów. Iloczyn pełny nieokreślony 3 nie umożliwia uproszczenia funkcji i dlatego nie został połączony z sąsiednimi kwadratami. Uproszczona funkcja jest zatem równa

$$f(x, y, z) = \sum (0, 4, 6) + \sum n(1, 3, 5, 7) = x + \bar{y}$$

Przykład 2.9. Uprościć funkcję

$$f(a, b, c, d, e) = \sum (0, 1, 5, 7, 13, 15, 16, 17, 25, 27, 29, 31)$$

korzystając z tablicy Karnaugh'a.

Tablica Karnaugh'a dla funkcji pięciu zmiennych jest pokazana na rysunku 2.14.

\bar{a}

		d		
	0	1	3	2
	4	5	7	6
b {	12	13	15	14
	8	9	11	10
	e			

a

		d		
	16	17	19	18
	20	21	23	22
b {	28	29	31	30
	24	25	27	26
	e			

Rys. 2.14. Tablica Karnaugh dla funkcji pięciu zmiennych (a, b, c, d, e)

Tablica Karnaugh dla funkcji

$$f(a, b, c, d, e) = \sum(0, 1, 5, 7, 13, 15, 16, 17, 25, 27, 29, 31)$$

jest pokazana na rysunku 2.15.

Diagram of a 4x4 grid labeled \bar{a} . The grid contains four '1's in a 2x2 pattern. The top row has '1's in the first two columns. The middle two rows have '1's in the second and third columns. Brackets indicate dimensions: d for the top row, c for the middle two rows, b for the first column, and e for the bottom two columns.

Diagram of a 4x4 grid labeled a . The grid contains four '1's in a 2x2 pattern. The top row has '1's in the first two columns. The middle two rows have '1's in the second and third columns. Brackets indicate dimensions: d for the top row, c for the middle two rows, b for the first column, and e for the bottom two columns.

Rys. 2.15. Tablica Karnaugh dla funkcji

$$f(a, b, c, d, e) = \sum(0, 1, 5, 7, 13, 15, 16, 17, 25, 27, 29, 31)$$

Uproszczona funkcja jest równa

$$\begin{aligned}
 f(a, b, c, d, e) &= \sum(0, 1, 5, 7, 13, 15, 16, 17, 25, 27, 29, 31) = \\
 &= \bar{a}ce + abe + \bar{a}\bar{b}\bar{c}\bar{d} + a\bar{b}\bar{c}\bar{d} = \bar{a}ce + abe + (\bar{a} + a)\bar{b}\bar{c}\bar{d} = \\
 &= \bar{a}ce + abe + \bar{b}\bar{c}\bar{d}
 \end{aligned}$$

2.2.3 Metoda Quine’a–McCluskeya

Metodę Quine’a–McCluskeya ([1, 2, 3, 4, 5, 6]), zwaną również metodą implikantów prostych, przedstawimy na przykładach:

Przykład 2.10. Uprościć funkcję

$$f(a, b, c, d) = \sum(5, 7, 8, 9, 10, 11, 13, 15)$$

Minimalizację rozpoczniemy od uporządkowania zbioru iloczynów pełnych funkcji w taki sposób, aby poszczególne grupy zawierały iloczyny pełne o takiej samej liczbie jedynek. Poszczególne grupy zapisuje się w postaci kolumny, rozpoczynając od grupy z najmniejszą liczbą jedynek. Grupy te oddziela się od siebie poziomą kreską. Z lewej strony kolumny wypisujemy dziesiętne odpowiedniki liczb dwójkowych.

W omawianym przykładzie otrzymujemy tablicę pokazaną na rysunku 2.16

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
8	1	0	0	0
5	0	1	0	1
9	1	0	0	1
10	1	0	1	0
7	0	1	1	1
11	1	0	1	1
13	1	1	0	1
15	1	1	1	1

Rys. 2.16. Szeregowanie iloczynów pełnych według liczby jedynek (przykład 2.10)

Następnie porównujemy każdą kombinację należącą do danej grupy z każdą kombinacją należącą do grupy następnej. Jeżeli porównywane kombinacje różnią się od siebie tylko na jednej pozycji, to łączymy je w nową kombinację, zastępując różniące się pozycje dowolnym znakiem, na przykład znakiem —. Otrzymane w ten sposób kombinacje zapisujemy w nowej kolumnie. Z lewej strony kolumny wypisujemy dziesiętne odpowiedniki składników nowej kombinacji.

W omawianym przykładzie otrzymujemy tablicę pokazaną na rysunku 2.17.

	a	b	c	d
8,9	1	0	0	–
8,10	1	0	–	0
5,7	0	1	–	1
5,13	–	1	0	1
9,11	1	0	–	1
9,13	1	–	0	1
10,11	1	0	1	–
7,15	–	1	1	1
11,15	1	–	1	1
13,15	1	1	–	1

Rys. 2.17. Łączenie kombinacji różniących się na jednej pozycji (przykład 2.10)

Kontynuujemy procedurę łączenia, usuwając powtarzające się kombinacje. Procedurę łączenia kończymy wtedy, kiedy nie ma już możliwości dokonywania dalszych łączeń. Każda kombinacja nie podlegająca dalszemu łączeniu jest nazywana implikantem prostym.

W omawianym przykładzie otrzymujemy tablicę pokazaną na rysunku 2.18.

	a	b	c	d
8,9,10,11	1	0	–	–
5,7,13,15	–	1	–	1
9,11,13,15	1	–	–	1

Rys. 2.18. Łączenie kombinacji różniących się na jednej pozycji (przykład 2.10)

Następnie rysujemy siatkę składającą się z linii pionowych i poziomych. Liczba linii pionowych jest równa liczbie iloczynów pełnych minimalizowanej funkcji. Liczba linii poziomych jest równa liczbie otrzymanych implikantów prostych. Liniom pionowym przypisujemy liczby dziesiętne odpowiadające iloczynom pełnym, liniom poziomym natomiast implikanty proste. Analizując siatkę stawiamy dowolny znak, na przykład kropkę (\bullet), w miejscach, w których linia odpowiedniego implikanta prostego określonego przez dane liczby przecina się z liniami iloczynów pełnych odpowiadających tym liczbom.

W omawianym przykładzie otrzymujemy siatkę pokazaną na rysunku 2.19

	5	7	8	9	10	11	13	15	
8,9,10,11			\bullet	\bullet	\bullet	\bullet			$a\bar{b}$
5,7,13,15	\bullet	\bullet					\bullet	\bullet	bd
9,11,13,15				\bullet		\bullet	\bullet	\bullet	ad

Rys. 2.19. Tablica implikantów prostych (przykład 2.10)

Uproszczona funkcja, równoważna funkcji minimalizowanej, może być otrzymana w postaci sumy wybranych implikantów prostych. Wybór implikantów prostych jest przeprowadzony tak, aby wszystkie iloczyny pełne występujące w funkcji minimalizowanej były reprezentowane w implikantach prostych. Liczba wybranych implikantów prostych powinna być jak najmniejsza. W omawianym przykładzie otrzymujemy następujące funkcje uproszczone, równoważne funkcji minimalizowanej:

$$f_1(a, b, c, d) = a\bar{b} + bd$$

$$f_2(a, b, c, d) = a\bar{b} + bd + ad$$

Najmniejsza liczba implikantów prostych występuje w funkcji

$$f_1(a, b, c, d)$$

Ostatecznie otrzymujemy

$$f(a, b, c, d) = \sum(5, 7, 8, 9, 10, 11, 13, 15) = a\bar{b} + bd$$

Przykład 2.11. Uprościć funkcję

$$f(a, b, c, d) = \sum(0, 1, 4, 6, 8, 12, 14) + \sum n(5, 10, 13, 15)$$

Porządkując zbiór wszystkich iloczynów pełnych w grupy o takiej samej liczbie jedynek, otrzymujemy tablicę pokazaną na rysunku 2.20.

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
0	0	0	0	0
1	0	0	0	1
4	0	1	0	0
8	1	0	0	0
5	0	1	0	1
6	0	1	1	0
10	1	0	1	0
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Rys. 2.20. Szeregowanie iloczynów pełnych według liczby jedynek (przykład 2.11)

Łącząc różniące się na jednej pozycji kombinacje z sąsiednich grup otrzymujemy tablicę pokazaną na rysunku 2.21

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
0,1	0	0	0	–
0,4	0	–	0	0
0,8	–	0	0	0
1,5	0	–	0	1
4,5	0	1	0	–
4,6	0	1	–	0
4,12	–	1	0	0
8,10	1	0	–	0
8,12	1	–	0	0
5,13	–	1	0	1
6,14	–	1	1	0
10,14	1	–	1	0
12,13	1	1	0	–
12,14	1	1	–	0
13,15	1	1	–	1
14,15	1	1	1	–

Rys. 2.21. Łączenie kombinacji różniących się na jednej pozycji (przykład 2.11)

Kontynuując procedurę łączenia otrzymujemy tablicę pokazaną na rysunku 2.22.

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
0,1,4,5	0	–	0	–
0,8,4,12	–	–	0	0
4,5,12,13	–	1	0	–
4,6,12,14	–	1	–	0
8,10,12,14	1	–	–	0
12,13,14,15	1	1	–	–

Rys. 2.22. Łączenie kombinacji różniących się na jednej pozycji (przykład 2.11)

Ponieważ nie ma już możliwości dokonywania dalszych łączeń, rysujemy siatkę pokazaną na rysunku 2.23

	0	1	4	6	8	12	14	
0,1,4,5	•	•	•					$\bar{a} \bar{c}$
0,8,4,12	•		•		•	•		$\bar{c} \bar{d}$
4,5,12,13			•			•		$b \bar{c}$
4,6,12,14			•	•		•	•	$b \bar{d}$
8,10,12,14					•	•	•	$a \bar{d}$
12,13,14,15						•	•	$a b$

Rys. 2.23. Tablica implikantów prostych (przykład 2.11)

Otrzymujemy następujące funkcje uproszczone, równoważne funkcji minimalizowanej:

$$f_1(a, b, c, d) = \bar{a} \bar{c} + b \bar{d} + \bar{c} \bar{d}$$

$$f_2(a, b, c, d) = \bar{a} \bar{c} + b \bar{d} + a \bar{d}$$

2.3 Podsumowanie

Opisane w rozdziale 2 metody minimalizacji funkcji logicznych są metodami podstawowymi, z których będziemy korzystali w dalszej części skryptu. Są również inne metody minimalizacji funkcji logicznych, na przykład metoda bezpośredniego przeszukiwania ([4]). Osobną grupę stanowią metody minimalizacyjne układu funkcji logicznych odnoszące się do układów kombinacyjnych wielowyjściowych (układy kombinacyjne będą omawiane w rozdziale 3). W tym przypadku poza wymaganiem minimalnej złożoności każdej funkcji logicznej wymaga się również, aby miały jak najwięcej elementów wspólnych ([1]). Opisywane w literaturze ([4, 6]) metody wykorzystują w tym przypadku zmodyfikowany algorytm Quine'a-McCluskeya.

Literatura

- [1] Kalisz J.: *Podstawy elektroniki cyfrowej*, WKŁ, 1993.
- [2] Majewski W.: *Układy logiczne*, WNT, 1993.
- [3] Mano M.M.: *Computer engineering: hardware design*, Prentice-Hall, 1988.
- [4] Traczyk W.: *Układy cyfrowe. Podstawy teoretyczne i metody syntezy*, WNT, 1986.
- [5] Bromirski J.: *Teoria automatów*, WNT, 1969.
- [6] McCluskey E.: *Logic design principles*, Prentice-Hall, 1986

Z a d a n i a

Zadanie 2.1. Uprościć następujące funkcje logiczne:

$$f_1(A, B, C, D) = \sum(0, 2, 6, 8, 10, 14)$$

$$f_2(A, B, C, D) = \sum(0, 1, 5, 8, 9, 13)$$

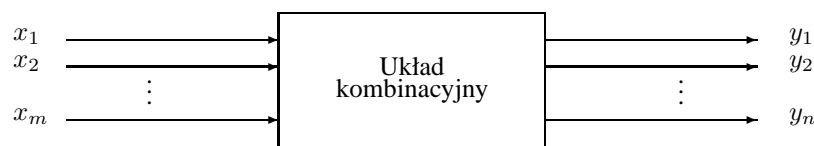
$$f_3(A, B, C, D) = \sum(0, 2, 6, 8) + \sum n(10, 11, 12, 13, 14, 15)$$

$$f_4(A, B, C, D) = \sum(0, 1, 3, 4, 5, 6, 7, 8, 9) + \sum n(10, 11, 12, 13, 14, 15)$$

3 UKŁADY KOMBINACYJNE

3.1 Wstęp

Układem kombinacyjnym ([1, 2, 3, 4, 5]) nazywamy taki układ logiczny, w którym każda kombinacja wartości zmiennych wejściowych jednoznacznie określa kombinację wartości zmiennych wyjściowych. Oznaczmy przez X zbiór wszystkich możliwych wartości zmiennych wejściowych, przez Y zaś zbiór wszystkich możliwych wartości zmiennych wyjściowych. Rozważmy układ kombinacyjny o m wejściach i n wyjściach pokazany na rysunku 3.1.



Rys. 3.1. Schemat blokowy układu kombinacyjnego

Działanie układu kombinacyjnego opisuje funkcja logiczna

$$Y = f(X)$$

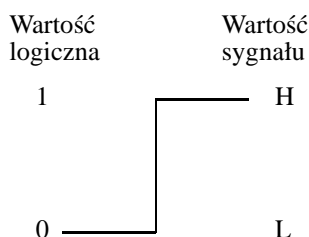
równoważna układowi następujących funkcji logicznych:

$$\begin{aligned} y_1 &= f_1(x_1, x_2, \dots, x_m) \\ &\vdots \\ y_n &= f_n(x_1, x_2, \dots, x_m) \end{aligned}$$

Układ kombinacyjny można opisać również za pomocą tablicy prawdy. Najprostszymi układami kombinacyjnymi są bramki logiczne (*digital logic gates*).

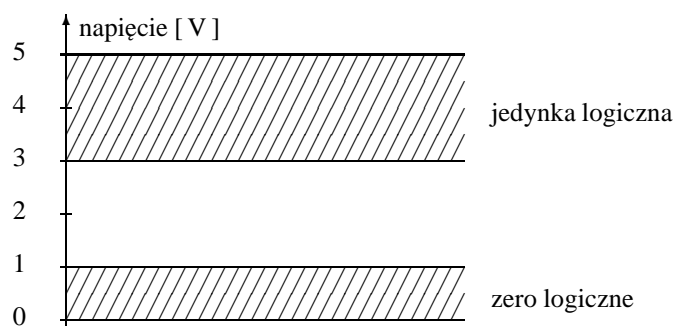
3.2 Bramki logiczne

W układach cyfrowych informację dwójkową przedstawia się najczęściej za pomocą napięcia elektrycznego w jednej z dwóch rozróżnialnych wartości: poziom niski L (*low*) i poziom wysoki H (*high*). W logice dodatniej (*positive logic*) poziom niższy reprezentuje zero logiczne, poziom wyższy natomiast jedynkę logiczną (rys. 3.2).



Rys. 3.2. Sygnały dwójkowe w logice dodatniej

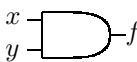

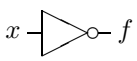
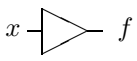
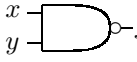



W praktyce zamiast poziomów określa się dwa przedziały napięć, wewnątrz których powinien znajdować się poziom sygnału reprezentującego logiczne 0 i 1. Przykładowe przedziały napięć są pokazane na rysunku rys. 3.3.



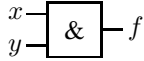
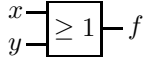
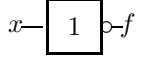
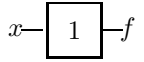
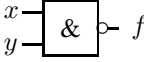
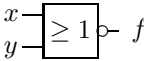
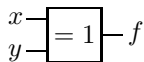
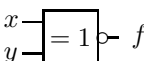
Rys. 3.3. Przykładowe przedziały napięć reprezentujące zero logiczne i jedynkę logiczną

Bramkami logicznymi nazywamy układy elektroniczne realizujące funkcje logiczne jednej lub wielu zmiennych. Na rysunku 3.4 są przedstawione nazwy, symbole graficzne, funkcje logiczne i tablice prawdy dla bramek logicznych o jednym lub dwóch wejściach ([3, 6]). Wszystkie bramki logiczne, z wyjątkiem NOT i Bufora, mogą mieć większą liczbę wejść.

Na rysunku 3.5 są pokazane symbole graficzne bramek logicznych zgodne ze standardem ANSI/IEEE ([3]). Dalej będziemy używać symboli graficznych pokazanych na rysunku 3.4.

Nazwa	Symbol graficzny	Funkcja logiczna	Tablica prawdy															
AND		$f = x \cdot y = xy$	<table><tr><th>x</th><th>y</th><th>f</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	f	0	0	0	0	1	0	1	0	0	1	1	1
x	y	f																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$f = x + y$	<table><tr><th>x</th><th>y</th><th>f</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	f	0	0	0	0	1	1	1	0	1	1	1	1
x	y	f																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT (Inverter)		$f = \overline{x}$	<table><tr><th>x</th><th>f</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	f	0	1	1	0									
x	f																	
0	1																	
1	0																	
Bufor		$f = x$	<table><tr><th>x</th><th>f</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	f	0	0	1	1									
x	f																	
0	0																	
1	1																	
NAND (Not AND)		$f = \overline{xy}$	<table><tr><th>x</th><th>y</th><th>f</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	f	0	0	1	0	1	1	1	0	1	1	1	0
x	y	f																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR (Not OR)		$f = \overline{x + y}$	<table><tr><th>x</th><th>y</th><th>f</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	f	0	0	1	0	1	0	1	0	0	1	1	0
x	y	f																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR (Exclusive-OR)		$f = x \oplus y$	<table><tr><th>x</th><th>y</th><th>f</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	f	0	0	0	0	1	1	1	0	1	1	1	0
x	y	f																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
XNOR (Exclusive-NOR)		$f = \overline{x \oplus y}$	<table><tr><th>x</th><th>y</th><th>f</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	f	0	0	1	0	1	0	1	0	0	1	1	1
x	y	f																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Rys. 3.4. Symbole graficzne bramek logicznych

Nazwa	Symbol graficzny	Funkcja logiczna
AND		$f = x \cdot y = xy$
OR		$f = x + y$
NOT (Inverter)		$f = \overline{x}$
Bufor		$f = x$
NAND (Not AND)		$f = \overline{xy}$
NOR (Not OR)		$f = \overline{x + y}$
XOR (Exclusive-OR)		$f = x \oplus y$
XNOR (Exclusive-NOR)		$f = \overline{x \oplus y}$

Rys. 3.5. Symbole graficzne bramek logicznych

Przykład 3.1. Narysować schemat logiczny funkcji

$$f(a, b, c, d) = ab + cd$$

korzystając z bramek NAND.

Z praw de Morgana, otrzymujemy

$$f(a, b, c, d) = ab + cd = \overline{\overline{ab} + \overline{cd}} = \overline{\overline{ab} \cdot \overline{cd}}$$

Rozwiązanie przykładu jest pokazane na rysunku 3.6.

Przykład 3.2. Narysować schemat logiczny funkcji

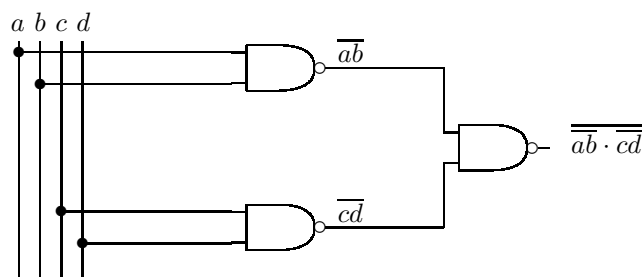
$$f(a, b, c, d) = (a + b) \cdot (c + d)$$

korzystając z bramek NOR.

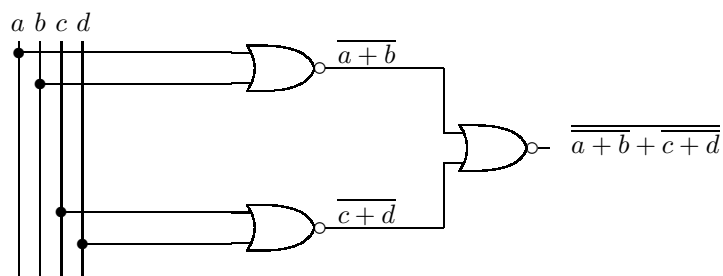
Z praw de Morgana, otrzymujemy

$$f(a, b, c, d) = (a + b) \cdot (c + d) = \overline{\overline{(a + b)} \cdot \overline{(c + d)}} = \overline{\overline{a + b} + \overline{c + d}}$$

Rozwiązanie przykładu jest pokazane na rysunku 3.7.



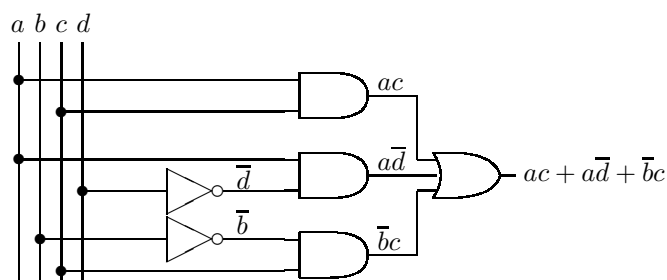
Rys. 3.6. Realizacja funkcji $f(a, b, c, d) = ab + cd$ za pomocą bramek NAND (przykład 3.1)



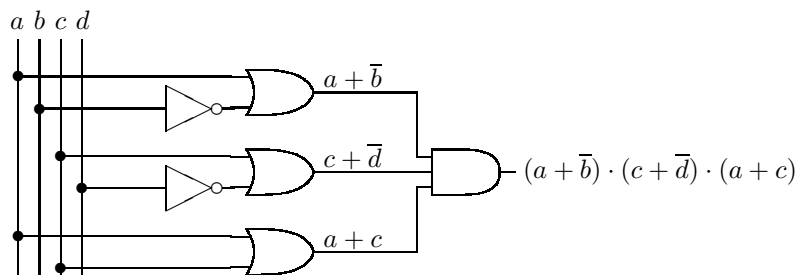
Rys. 3.7. Realizacja funkcji $f(a, b, c, d) = (a + b) \cdot (c + d)$ za pomocą bramek NOR (przykład 3.2)

Przykład 3.3. Narysować schematy logiczne funkcji z przykładu 2.7.

Rozwiązanie przykładu jest pokazane na rysunkach 3.8 i 3.9.



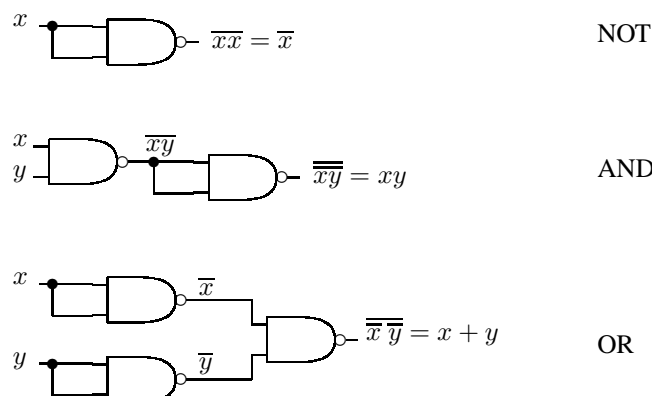
Rys. 3.8. Schemat logiczny funkcji (w postaci sumy iloczynów) z przykładu 2.7 (przykład 3.3)



Rys. 3.9. Schemat logiczny funkcji (w postaci iloczynu sum) z przykładu 2.7 (przykład 3.3)

Przykład 3.4. Udowodnić, że zbiór składający się z jednej funkcji NAND jest systemem funkcjonalnie pełnym.

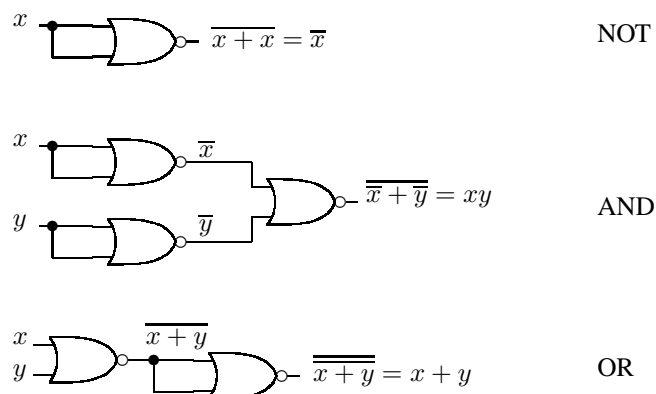
Wystarczy udowodnić, że za pomocą funkcji NAND można zrealizować funkcje: NOT, AND i OR. Rozwiązanie przykładu jest pokazane na rysunku 3.10.



Rys. 3.10. Realizacja NOT, AND i OR za pomocą NAND (przykład 3.4)

Przykład 3.5. Udowodnić, że zbiór składający się z jednej funkcji NOR jest systemem funkcjonalnie pełnym.

Wystarczy udowodnić, że za pomocą funkcji NOR można zrealizować funkcje: NOT, AND i OR. Rozwiązanie przykładu jest pokazane na rysunku 3.11.

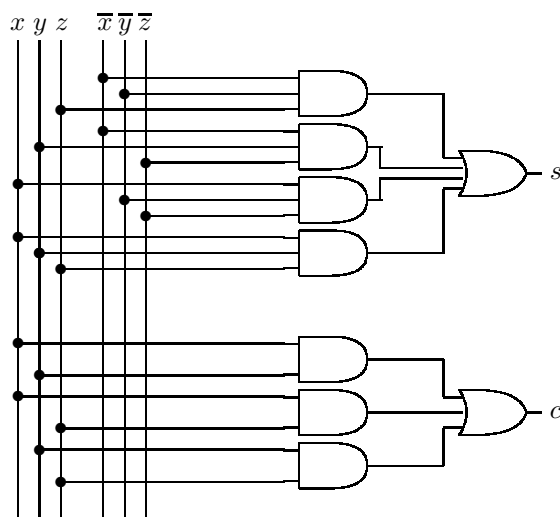


Rys. 3.11. Realizacja NOT, AND i OR za pomocą NOR (przykład 3.5)

3.3 Analiza układów kombinacyjnych

Analiza układu kombinacyjnego polega na określeniu relacji pomiędzy wartościami jego wejść i wyjść.

Przykład 3.6. Przeprowadzić analizę układu kombinacyjnego pokazanego na rysunku 3.12.



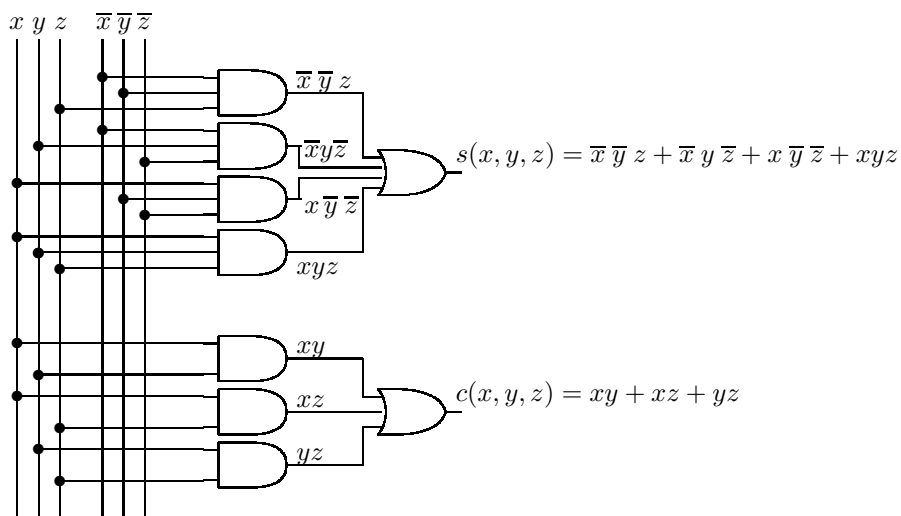
Rys. 3.12. Układ kombinacyjny (przykład 3.6)

Śledząc sygnały na wejściach i wyjściu każdej bramki otrzymujemy następujące funkcje logiczne pokazane na rysunku 3.13:

$$s(x, y, z) = \bar{x} \bar{y} z + \bar{x} y \bar{z} + x \bar{y} \bar{z} + xyz$$

$$c(x, y, z) = xy + xz + yz$$

Aby opisać analizowany układ za pomocą tablicy prawdy, należy określić wartości wyjść (s, c) dla wszystkich możliwych kombinacji wejść (x, y, z). Tablica prawdy analizowanego układu jest pokazana na rysunku 3.14.



Rys. 3.13. Układ kombinacyjny (przykład 3.6).

Wejścia			Wyjścia	
x	y	z	s	c
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Rys. 3.14. Tablica prawdy układu z rysunku 3.12 (przykład 3.6)

3.4 Projektowanie układów kombinacyjnych

Projektowanie układów kombinacyjnych omówimy na przykładach.

Przykład 3.7. Za pomocą bramek NAND zrealizować układ opisany tablicą prawdy pokazaną na rysunku 3.15:

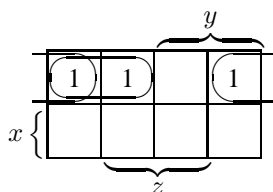
Wejścia			Wyjścia
x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Rys. 3.15. Tablica prawdy układu (przykład 3.7)

Na podstawie tablicy prawdy otrzymujemy funkcję logiczną

$$f(x, y, z) = \sum(0, 1, 2)$$

która po minimalizacji



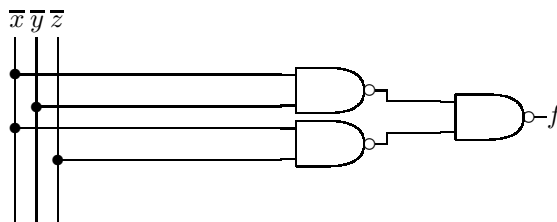
przyjmuje postać

$$f(x, y, z) = \sum(0, 1, 2) = \bar{x} \bar{y} + \bar{x} \bar{z}$$

Korzystając z praw de Morgana, otrzymujemy funkcję

$$f(x, y, z) = \bar{x} \bar{y} + \bar{x} \bar{z} = \overline{\overline{\bar{x} \bar{y}} + \overline{\bar{x} \bar{z}}} = \overline{\overline{\bar{x} \bar{y}} \cdot \overline{\bar{x} \bar{z}}}$$

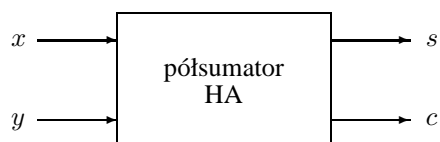
na podstawie której rysujemy schemat logiczny układu pokazany na rysunku 3.16.



Rys. 3.16. Rozwiązanie przykładu 3.7.

3.4.1 Układy arytmetyczne

Układ kombinacyjny dodający dwie cyfry dwójkowe jest nazywany półsumatorem (*half adder*). Zmiennymi wejściowymi półsumatora są bity składników sumy (x, y) . Zmiennymi wyjściowymi są bity sumy s i przeniesienia c . Schemat blokowy i tablica prawdy półsumatora są pokazane na rysunku 3.17.



Wejścia		Wyjścia	
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

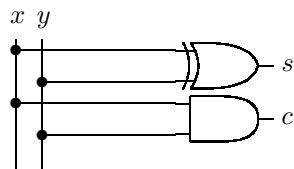
Rys. 3.17 Schemat blokowy i tablica prawdy półsumatora

Z tablicy prawdy półsumatora otrzymujemy następujące funkcje logiczne:

$$s(x, y) = \sum(1, 2) = \bar{x}y + x\bar{y} = x \oplus y$$

$$c(x, y) = \sum(3) = xy$$

na podstawie których rysujemy schemat logiczny półsumatora pokazany na rysunku 3.18.



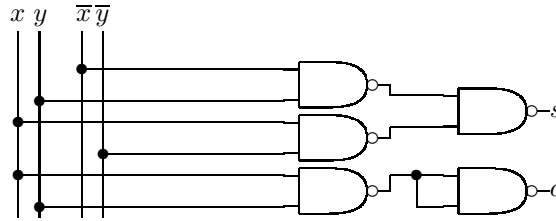
Rys. 3.18. Schemat logiczny półsumatora

Korzystając z praw de Morgana można zapisać funkcje $s(x, y)$ i $c(x, y)$ w postaci

$$s(x, y) = \overline{x}y + x\overline{y} = \overline{\overline{\overline{x}y + x\overline{y}}} = \overline{\overline{x}y} \cdot \overline{x\overline{y}}$$

$$c(x, y) = xy = \overline{\overline{xy}}$$

na podstawie których rysujemy schemt logiczny półsumatora, zrealizowanego za pomocą bramek NAND, pokazany na rysunku 3.19.



Rys. 3.19. Schemat logiczny półsumatora zrealizowanego za pomocą bramek NAND

Układ kombinacyjny dodający trzy cyfry dwójkowe jest nazywany sumatorem (*full adder*). Ma trzy wejścia. Dwie ze zmiennych wejściowych (x, y) reprezentują bity składników sumy. Trzecie wejście (z) reprezentuje przeniesienie z poprzedniej, mniej znaczącej pozycji. Zmiennymi wyjściowymi są bity sumy s i przeniesienia c . Schemat blokowy i tablica prawdy sumatora są pokazane na rysunku 3.20.

Z tablicy prawdy sumatora otrzymujemy następujące funkcje logiczne:

$$s(x, y, z) = \sum(1, 2, 4, 7)$$

$$c(x, y, z) = \sum(3, 5, 6, 7)$$

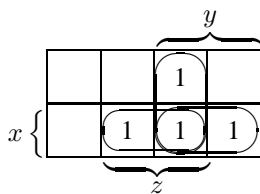
Z tablicy Karnaugh dla funkcji $s(x, y, z)$

		y	
		1	1
x	1		1
		1	
		z	

widać, że nie można jej zminimalizować. Zatem

$$s(x, y, z) = \sum(1, 2, 4, 7) = \overline{x}\overline{y}z + \overline{x}y\overline{z} + x\overline{y}\overline{z} + xyz$$

Minimalizując funkcję $c(x, y, z)$

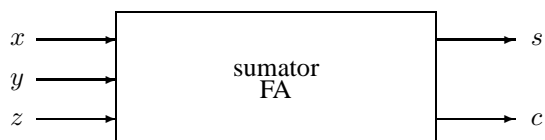


otrzymujemy

$$c(x, y, z) = xz + yz + xy$$

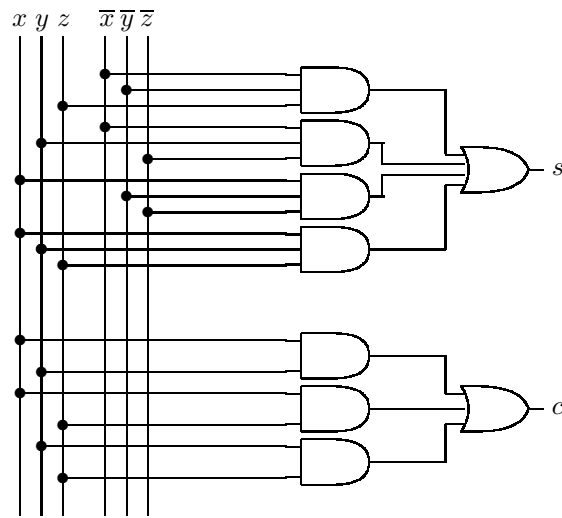
Na podstawie otrzymanych funkcji $s(x, y, z)$ i $c(x, y, z)$ rysujemy schemat logiczny sumatora pokazany na rysunku 3.21.

Sumę dwóch n -bitowych liczb dwójkowych można uzyskać w układzie n -bitowego sumatora równoległego, którego schemat blokowy dla $n = 4$ jest pokazany na rysunku 3.22 ([3, 6]).

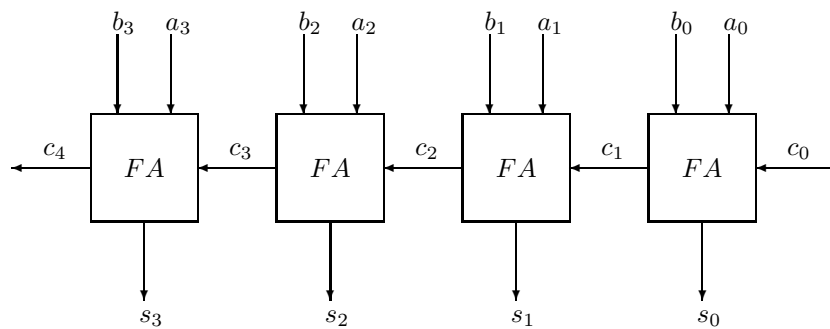


Wejścia			Wyjścia	
x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Rys. 3.20. Schemat blokowy i tablica prawdy sumatora



Rys. 3.21. Schemat logiczny sumatora



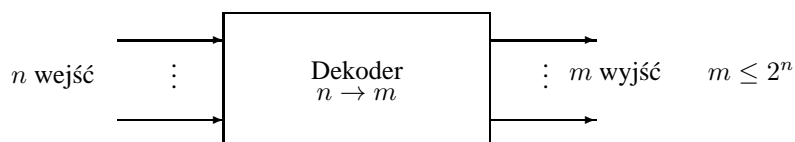
Rys. 3.22. Schemat blokowy 4-bitowego sumatora równoległego

Na rysunku tym a_i , $i = 0, \dots, 3$, oznaczają bity pierwszego składnika sumy; b_i – bity drugiego składnika sumy; c_i – bity przeniesienia, a s_i – bity sumy. Przeniesienie wejściowe jest oznaczone jako c_0 , przeniesienie wyjściowe jako c_4 .

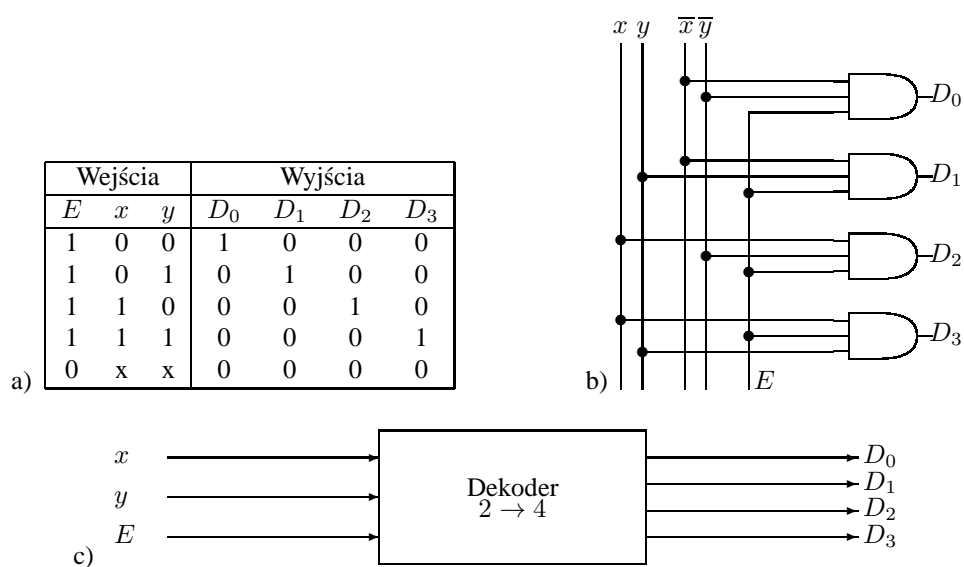
3.4.2 Dekodery

Dekoderem (*decoder*) nazywamy układ kombinacyjny mający n wejść i m wyjść, $m \leq 2^n$, w którym każdej kombinacji zmiennych wejściowych odpowiada pojawienie się jedynki logicznej tylko na jednym wyjściu, na pozostałych wyjściach pojawiają się wtedy zera logiczne. Schemat blokowy dekodera $n \rightarrow m$ jest pokazany na rysunku 3.23. Przykład dekodera $2 \rightarrow 4$ jest pokazany na rysunku 3.24. Dekodery generują 2^n iloczynów pełnych

dla n zmiennych wejściowych. Dowolna funkcja logiczna może być wyrażona jako suma iloczynów pełnych. Można zatem użyć dekodera do generowania iloczynów pełnych i bramek *OR* do tworzenia ich sum logicznych.



Rys. 3.23. Schemat blokowy dekodera $n \rightarrow m$



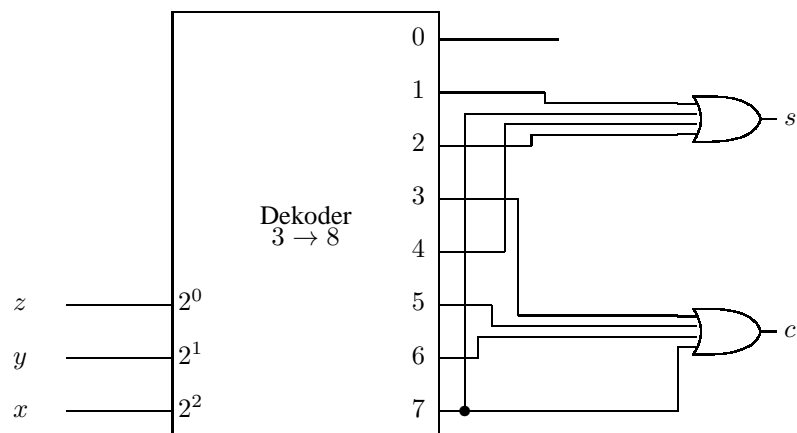
Rys. 3.24. Dekoder $2 \rightarrow 4$ z wejściem E (*enable*): tablica prawdy (a)), schemat logiczny (b)) i schemat blokowy (c))

Przykład 3.8 Zaprojektować jednobitowy sumator za pomocą dekodera.

Poprzednio (rozdział 3) otrzymaliśmy następujące funkcje logiczne dla sumatora:

$$s(x, y, z) = \sum(1, 2, 4, 7), \quad c(x, y, z) = \sum(3, 5, 6, 7)$$

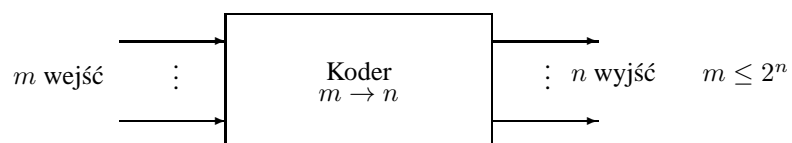
Ponieważ mamy 3 zmienne na wejściu i 8 iloczynów pełnych, zatem konieczny jest dekodek $3 \rightarrow 8$. Schemat logiczny zaprojektowanego sumatora jest pokazany na rysunku 3.25 ([3]).



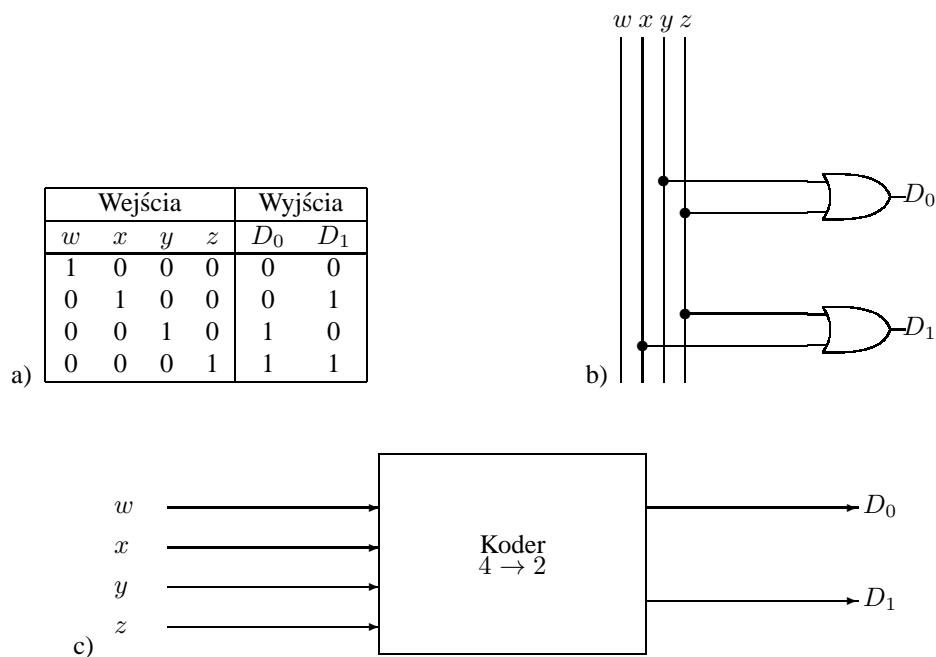
Rys. 3.25. Sumator zrealizowany za pomocą dekodera $3 \rightarrow 8$

3.4.3 Kodery

Koderem (*encoder*) nazywamy układ kombinacyjny działający odwrotnie do dekodera. Koder ma m wejść i n wyjść, $m \leq 2^n$. Schemat blokowy kodera $m \rightarrow n$ jest pokazany na rysunku 3.26. Przykład kodera $4 \rightarrow 2$ jest pokazany na rysunku 3.27.



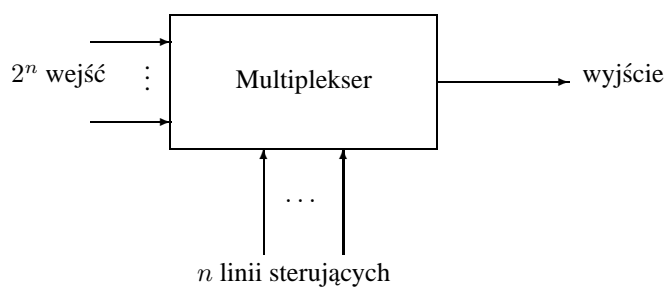
Rys. 3.26. Schemat blokowy kodera $m \rightarrow n$



Rys. 3.27. Koder $4 \rightarrow 2$: tablica prawdy (a)), schemat logiczny (b)) i schemat blokowy (c))

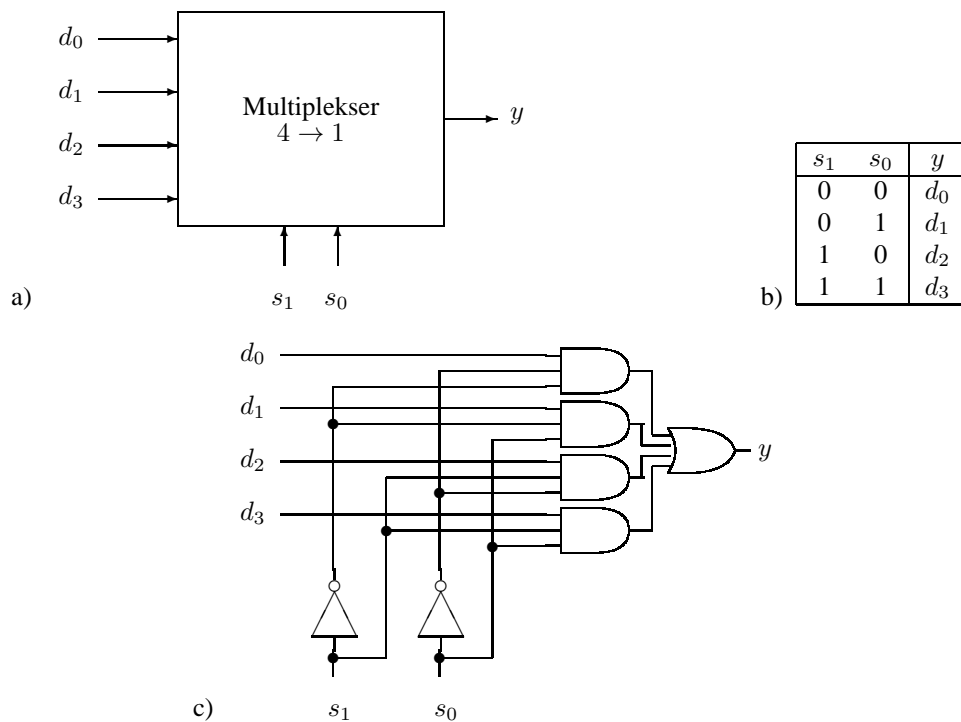
3.4.4 Multipleksery

Multiplekserem (*multiplexer*) nazywamy układ kombinacyjny wybierający informację dwójkową na jednej z linii wejściowych i kierujący ją na jedną linię wyjściową. Wybór linii wejściowej jest określany przez linie sterujące. Schemat blokowy multiplexera o 2^n liniach wejściowych i n liniach sterujących jest pokazany na rysunku 3.28. Przykład multiplexera $4 \rightarrow 1$ jest pokazany na rysunku 3.29.



Rys. 3.28. Schemat blokowy multiplexera $2^n \rightarrow 1$

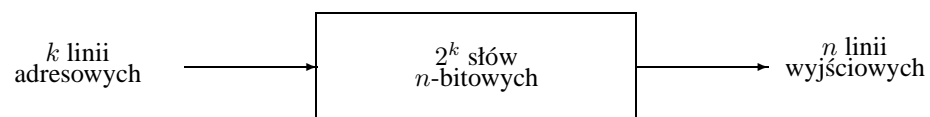
Linie wejściowe (d_0, d_1, d_2, d_3) są doprowadzone do bramek AND, których wyjścia są doprowadzone do bramki OR. W danej chwili tylko jedna linia wejściowa jest połączona z wyjściem (y). O wyborze wejścia połączanego z wyjściem decydują linie sterujące (s_0, s_1).



Rys. 3.29. Multiplexer $4 \rightarrow 1$: schemat blokowy (a)), tablica prawdy (b)) i schemat logiczny (c))

3.5 Pamięci ROM

Na rysunku 3.30 jest pokazany schemat blokowy pamięci stałej ROM (*read only memory*).



Rys. 3.30. Schemat blokowy pamięci stałej ROM

Pamięć ROM ([1, 2, 3, 4]) ma k linii adresowych umożliwiających wybór jednego z $2^k = m$ słów oraz n linii wyjściowych. W pamięciach ROM informacja dwójkowa jest wprowadzana na stałe podczas produkcji i nie można jej zmienić. Pamięci PROM (*programmable ROM*) są pamięciami, które mogą być programowane przez użytkownika. W takim przypadku raz wpisanej informacji nie można zmienić. Pamięci EPROM (*erasable PROM*) i EEPROM (*electrically erasable PROM*) mogą być wielokrotnie programowane przez użytkownika. Usuwanie zapisanych informacji przeprowadza się przez naświetlanie promieniami ultrafioletowymi (pamięci EPROM) lub elektrycznie (pamięci EEPROM).

Ze względu na to, że każda kombinacja wartości zmiennych wejściowych (linie adresowe) jednoznacznie określa kombinację wartości zmiennych wyjściowych (zawartość słowa pamięci), pamięć ROM jest zaliczana do układów kombinacyjnych. Pamięć ROM jest zbudowana z dekodera i bramek OR.

Rozważmy pamięć ROM o pojemności czterech słów, po cztery bity każde (4×4), pokazaną na rysunku 3.31 ([3]). Wejściami do pamięci ROM są linie adresowe (A_1, A_0). Wyjściami pamięci ROM są wyjścia czterech bramek OR (Y_3, Y_2, Y_1, Y_0). Rozważana pamięć jest pokazana na rysunku 3.32 ([3]) w uproszczonej postaci. Na rysunku 3.32 znak \times oznacza połączenie.

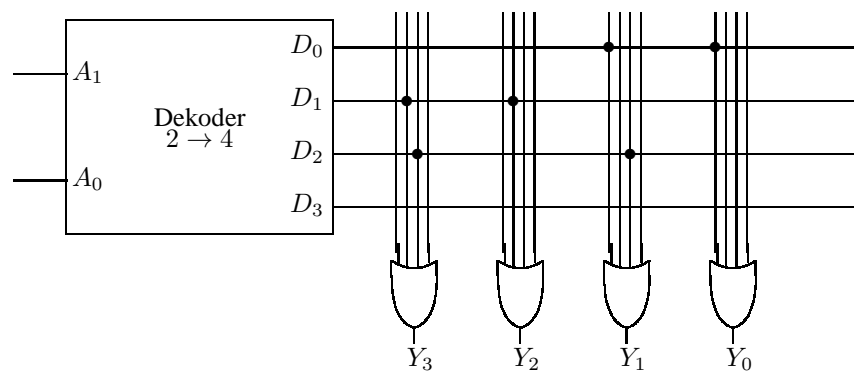
Dowolna funkcja logiczna może być wyrażona jako suma iloczynów pełnych. Można zatem użyć pamięci ROM do realizacji dowolnej funkcji logicznej.

Przykład 3.9. Zaprojektować jednobitowy sumator za pomocą pamięci ROM.

Na podstawie tablicy prawdy sumatora programujemy pamięć ROM 8×2 , tak jak na rysunku 3.33 ([3]).

3.6 Programowalne układy logiczne PLD

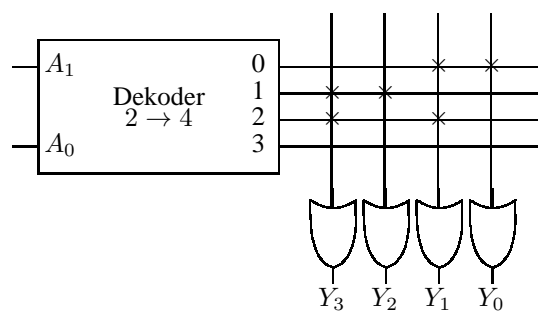
Układy scalone kombinacyjne są produkowane jako układy standardowe i jako układy specjalizowane ASIC (*application specific integrated circuits*), ([1, 2, 3]). Wśród układów specjalizowanych wyróżnia się między innymi programowalne układy logiczne PLD (*programmable logic devices*). Wśród układów PLD można wyróżnić 3 podstawowe typy: pamięci PROM, programowalne struktury logiczne PLA i programowalne struktury logiczne PAL.

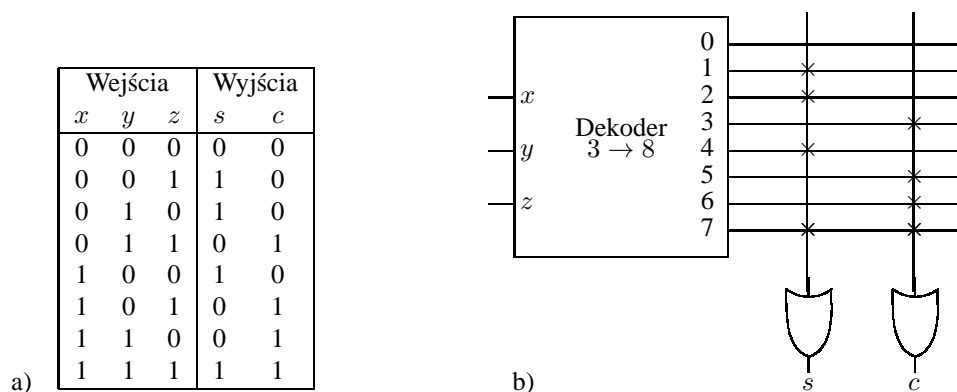


a) schemat logiczny

Wejścia		Wyjścia			
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	1	1
0	1	1	1	0	0
1	0	1	0	1	0
1	1	0	0	0	0

b) tablica prawdy

Rys. 3.31. Pamięć ROM 4×4 : schemat logiczny (a) i tablica prawdy (b))**Rys. 3.32.** Schemat logiczny pamięci ROM 4×4 z rysunku 3.31 w uproszczonej postaci



Rys. 3.33. Realizacja sumatora za pomocą pamięci ROM: tablica prawdy sumatora (a)) i pamięć ROM 8×2 (b))

3.6.1 Programowalne struktury logiczne PLA

Programowalna struktura logiczna PLA ([1, 2, 3]) składa się z programowalnej matrycy AND i programowalnej matrycy OR. Programowalna matryca AND umożliwia tworzenie iloczynów zmiennych wejściowych. Wyjścia bramek AND są połączone z programowalną matrycą bramek OR. Programowalna matryca OR umożliwia tworzenie sum złożonych z dowolnych iloczynów utworzonych w matrycy AND. Schemat logiczny przykładowego układu PLA o trzech wejściach i dwóch wyjściach jest pokazany na rysunku 3.34.

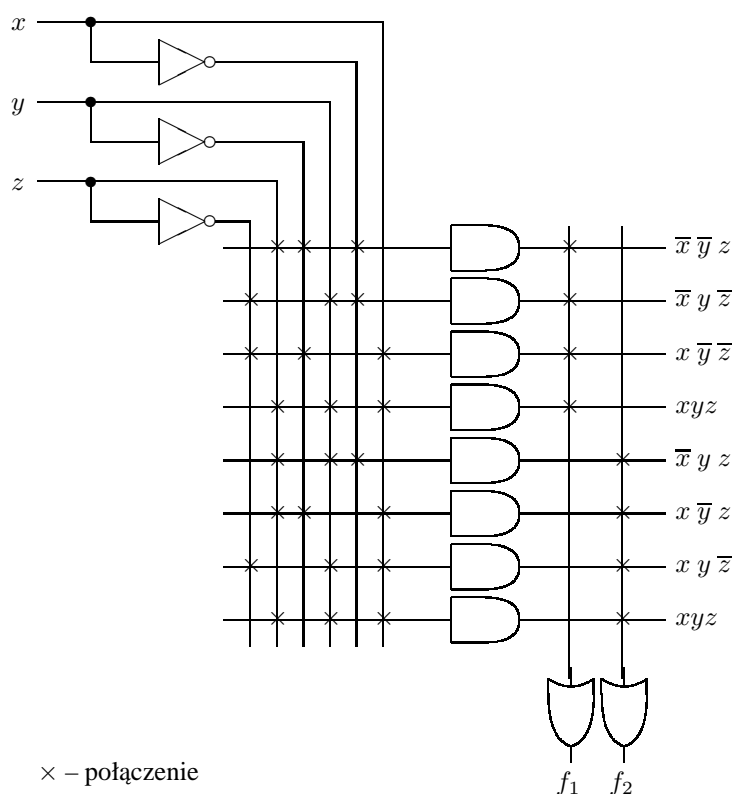
Przez realizację odpowiednich połączeń (programowanie) na wyjściach bramek AND uzyskuje się potrzebne (do realizacji funkcji logicznej) iloczyny zmiennych wejściowych (x, y, z). Przez realizację odpowiednich połączeń na wejściach bramek OR uzyskuje się sumę logiczną wybranych iloczynów.

Układ PLA zaprogramowany w taki sposób jak na rysunku 3.34 (połączenia są zaznaczone znakiem \times) realizuje następujące funkcje logiczne:

$$f_1(x, y, z) = \overline{x} \overline{y} z + \overline{x} y \overline{z} + x \overline{y} \overline{z} + xyz = \sum(1, 2, 4, 7)$$

$$f_2(x, y, z) = \overline{x} y z + x \overline{y} z + x y \overline{z} + xyz = \sum(3, 5, 6, 7)$$

będące funkcjami sumy (f_1) i przeniesienia (f_2) sumatora.



Rys. 3.34. Schemat logiczny przykładowego układu PLA

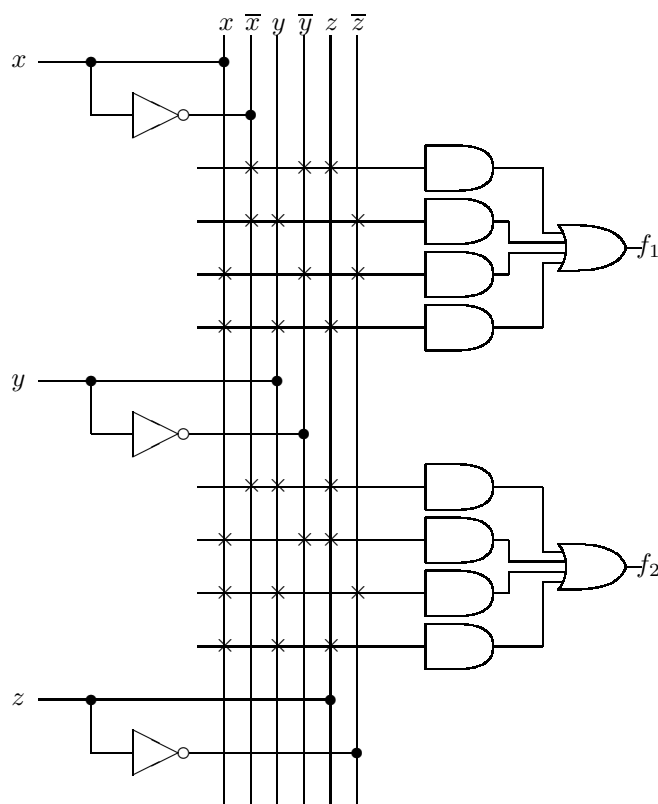
3.6.2 Programowalne struktury logiczne PAL

Programowalna struktura logiczna PAL ([1, 2, 3]) składa się z programowalnej matrycy AND i stałej matrycy OR. Programowalna matryca AND umożliwia tworzenie iloczynów zmiennych wejściowych. Wyjścia bramek AND są połączone na stałe z wejściami bramek OR, realizując funkcje sum iloczynów. Układy PAL są realizowane także ze sprzężeniem zwrotnym z niektórych wyjść matrycy OR na wejścia matrycy AND. Schemat logiczny przykładowego układu PAL jest pokazany na rysunku 3.35.

Układ PAL zaprogramowany w taki sposób jak na rysunku 3.35 realizuje funkcje logiczne przeniesienia (f_1) i sumy (f_2) sumatora:

$$f_1(x, y, z) = \bar{x} \bar{y} z + \bar{x} y \bar{z} + x \bar{y} \bar{z} + xyz = \sum(1, 2, 4, 7)$$

$$f_2(x, y, z) = \bar{x} y z + x \bar{y} z + x y \bar{z} + xyz = \sum(3, 5, 6, 7)$$



Rys. 3.35. Schemat logiczny przykładowego układu PAL

3.7 Podsumowanie

W rozdziale 3 zostały przedstawione elementarne układy kombinacyjne (bramki logiczne). Bramki logiczne są realizowane jako układy małego stopnia scalenia SSI (*small scale integration*) ([7]). Zaprojektowane zostały następujące układy kombinacyjne: sumator, dekodery, koder i multiplexer, niezbędne do zrozumienia materiału prezentowanego w dalszej części skryptu. W praktyce wymienione wyżej układy kombinacyjne są realizowane jako układy średniego stopnia scalenia MSI (*medium scale integration*) ([7]).

Podane zostały przykłady realizacji układów kombinacyjnych za pomocą programowalnych układów logicznych PLD i pamięci ROM. Programowalne układy logiczne PLD i pamięci ROM są realizowane jako układy wielkiego stopnia scalenia LSI (*large scale integration*). Schematy logiczne i parametry układów scalonych małego i średniego stopnia scalenia są przedstawione między innymi w [7].

Przedstawiona została klasyczna metoda projektowania układów kombinacyjnych wykorzystująca układy małego stopnia scalenia. Projektowanie układów cyfrowych ma inny przebieg w przypadku wykorzystania układów średniego i wielkiego stopnia scalenia ([2, 5, 8]).

Literatura

- [1] Kalisz J.: *Podstawy elektroniki cyfrowej*, WKŁ, 1993.
- [2] Majewski W.: *Układy logiczne*, WNT, 1993.
- [3] Mano M.M.: *Computer engineering: hardware design*, Prentice-Hall, 1988.
- [4] Pieńkos J., Turczyński J.: *Układy scalone TTL w systemach cyfrowych*, WKŁ, 1980.
- [5] Traczyk W.: *Układy cyfrowe. Podstawy teoretyczne i metody syntezy*, WNT, 1986.
- [6] Mano M.M.: *Computer system architecture*, Prentice-Hall, 1993.
- [7] Sasal W.: *Układy scalone serii UCY74LS i UCY74S. Parametry i zastosowania*, WKŁ, 1993.
- [8] Majewski W., Jasiński K., Luba T., Zbierchowski B.: *Programowalne moduły logiczne w syntezie układów cyfrowych*, WKŁ, 1992.

Z a d a n i a

Zadanie 3.1. Narysować schematy logiczne uproszczonych funkcji z zadania 2.1 korzystając z dowolnych bramek logicznych.

Zadanie 3.2. Narysować schematy logiczne uproszczonych funkcji z zadania 2.1 korzystając z bramek NAND.

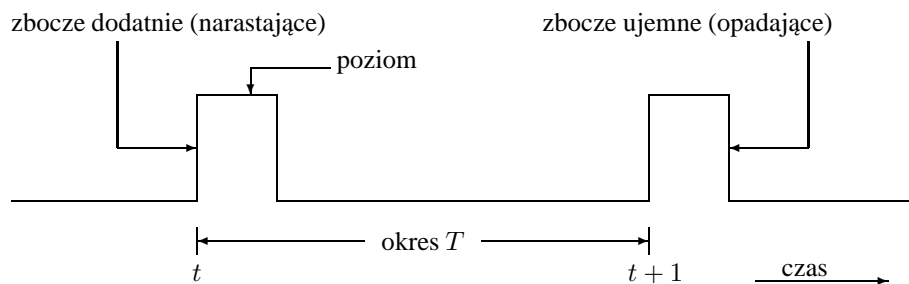
Zadanie 3.3. Narysować schematy logiczne uproszczonych funkcji z zadania 2.1 korzystając z bramek NOR.

4 UKŁADY SEKWENCYJNE

4.1 Wstęp

W układzie kombinacyjnym każda kombinacja wartości zmiennych wejściowych jednoznacznie określa kombinację wartości zmiennych wyjściowych. Układ kombinacyjny nie posiada pamięci. Układ sekwencyjny ([1, 2, 3, 4, 5]) posiada pamięć. W układzie sekwencyjnym wartości zmiennych wyjściowych zależą od wartości zmiennych wejściowych i stanu układu. Stan układu jest zdefiniowany przez zawartość poszczególnych elementów pamiętających. Najprostszym układem sekwencyjnym jest przerzutnik (*flip-flop*), który pamięta jeden bit (stan logiczny 0 lub 1).

Do dalszych rozważań wprowadzimy dyskretną oś czasu pokazaną na rysunku 4.1. Kolejne punkty na tej osi będziemy oznaczać liczbami naturalnymi $1, 2, \dots, t-1, t, t+1, \dots$. Punkty te są wyznaczane przez kolejne zbocza jednakowego typu (dodatnie, ujemne) impulsów zegarowych (*clock pulses*).



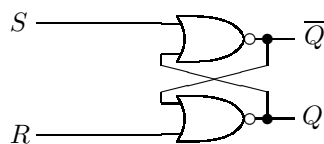
Rys. 4.1. Diagram czasowy impulsów zegarowych o okresie T

4.2 Przerzutniki

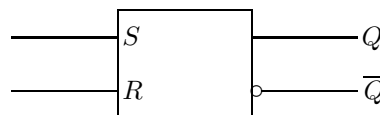
Elementarnymi układami pamiętającymi (przechowującymi) jeden bit są przerzutniki.

4.2.1 Przerzutnik SR

Na rysunku 4.2 a) jest pokazany układ zrealizowany z bramek NOR, zwany przerzutnikiem SR.



a) schemat logiczny



b) symbol graficzny

Rys. 4.2. Przerzutnik SR: schemat logiczny (a)), symbol graficzny (b))

Przerzutnik SR ma dwa wejścia: R (*reset*) i S (*set*). Ma również dwa wyjścia: Q i \overline{Q} . Jeżeli $Q = 0$, to mówimy, że w przerzutniku jest przechowywane (pamiętane) zero. Jeżeli $Q = 1$, to mówimy, że w przerzutniku jest przechowywana (pamiętana) jedynka.

Kombinacja $S = 1, R = 0$ daje $Q = 1$.

Kombinacja $S = 0, R = 1$ daje $Q = 0$.

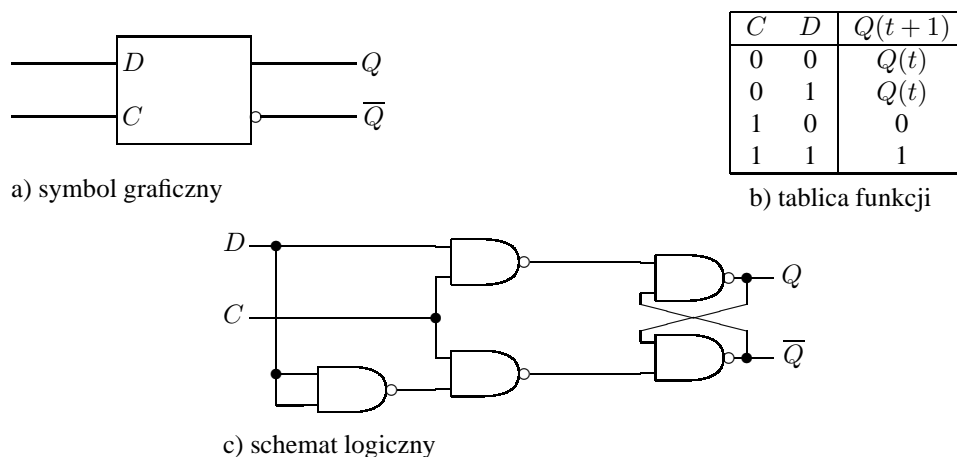
Kombinacja $S = 0, R = 0$ nie zmienia wartości wyjścia Q .

Kombinacja $S = 1, R = 1$ powodowałaby wystąpienie sytuacji, w której $Q = \overline{Q} = 0$, co nie ma sensu. Zatem kombinacja $S = 1, R = 1$ jest niedozwolona.

4.2.2 Przerzutnik D

Przerzutnik D (D latch) ma wejście danych D , wejście zegarowe C , wyjście Q dla wartości przechowywanego bitu i wyjście \overline{Q} dla jego negacji. Symbol graficzny, tablica funkcji opisująca działanie przerzutnika D i jego schemat logiczny są pokazane na rysunku 4.3 ([3]). W tablicy funkcji $Q(t)$ jest aktualnym stanem przerzutnika (w chwili t), $Q(t + 1)$ natomiast jest jego stanem następnym (w chwili $t + 1$).

Jeżeli na wejściu zegarowym C jest napięcie o poziomie jedynki logicznej ($C = 1$), wtedy wejście D oddziałuje bezpośrednio na wyjście Q . Wszystkie zmiany na wejściu D zachodzące w tym czasie (tzn. wtedy, kiedy $C = 1$) są natychmiast powtarzane na wyjściu Q . Kiedy stan wejścia zegarowego C zmieni się z 1 na 0, wyjście Q pozostaje w stanie logicznym, odpowiadającym stanowi wejścia D , występującemu bezpośrednio przed pojawieniem się zmiany 1 na 0 na wejściu zegarowym C . Stan wyjścia Q nie ulega zmianie tak długo, jak długo na wejściu zegarowym C jest napięcie o poziomie zera logicznego ($C = 0$). Wpisywanie informacji z wejścia D na wyjście Q odbywa się w czasie, kiedy na wejściu zegarowym C jest napięcie o poziomie jedynki logicznej.



Rys. 4.3. Przerzutnik D : symbol graficzny (a)), tablica funkcji (b)), schemat logiczny (c))

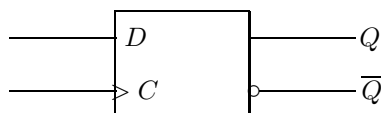
4.2.3 Przerzutnik D wyzwalany zboczem

Przerzutniki mogą być wyzwalane:

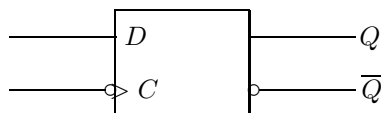
- zboczem dodatnim (*positive-edge-triggered*),
- zboczem ujemnym (*negative-edge-triggered*).

W przerzutniku D wyzwalanym zboczem dodatnim wpisywanie wartości z wejścia D na wyjście Q odbywa się w czasie zmiany napięcia na wejściu zegarowym z poziomu zera logicznego do poziomu jedynki logicznej. Symbol graficzny przerzutnika D wyzwalanego zboczem dodatnim jest pokazany na rysunku 4.4.

Wejście zegarowe C jest oznaczone trójkątem. Trójkąt ten wskazuje, że przerzutnik jest wyzwalany zboczem. W przerzutniku D wyzwalanym zboczem ujemnym wpisywanie wartości z wejścia D na wyjście Q odbywa się w czasie zmiany napięcia na wejściu zegarowym z poziomu jedynki logicznej do poziomu zera logicznego. Symbol graficzny przerzutnika D wyzwalanego zboczem ujemnym jest pokazany na rysunku 4.5.



Rys. 4.4. Symbol graficzny przerzutnika D wyzwalanego zboczem dodatnim (narastającym)

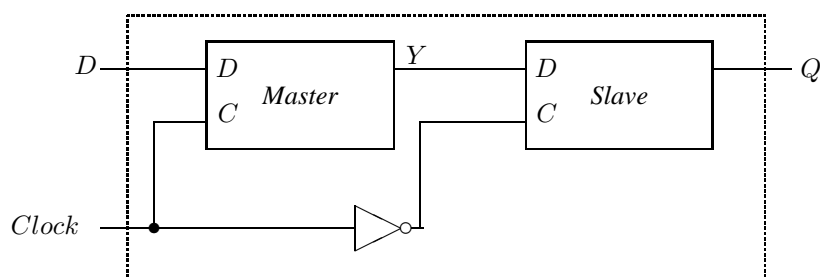


Rys. 4.5. Symbol graficzny przerzutnika D wyzwalanego zboczem ujemnym (opadającym)

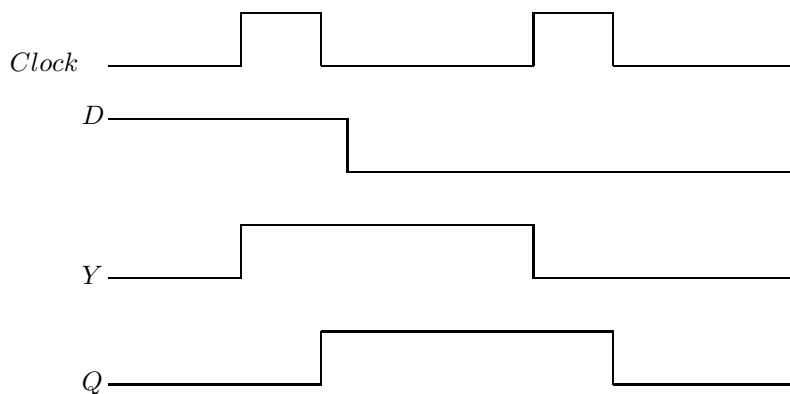
4.2.4 Przerzutnik D – MS

Schemat blokowy przerzutnika D – MS (*master–slave*) jest pokazany na rysunku 4.6. Przerzutnik ten składa się z przerzutnika *Master* (głównego), przerzutnika *Slave* (pomocniczego) i bramki NOT. Kiedy impuls zegara $Clock = 0$, na wejściu zegarowym przerzutnika *Slave* jest 1. W takim przypadku przerzutnik *Master* jest odcięty, tzn. wejście przerzutnika *Master* nie oddziałuje na jego wyjście, a przerzutnik *Slave* jest aktywny, tzn. wejście przerzutnika *Slave* oddziałuje na jego wyjście. Kiedy impuls zegara $Clock = 1$, przerzutnik *Master* jest aktywny, a przerzutnik *Slave* jest odcięty. Pokazane na rysunku 4.7 diagramy czasowe ilustrują kolejność zdarzeń zachodzących w przerzutniku D – MS .

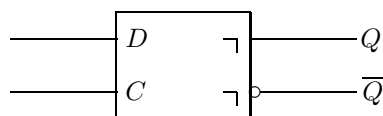
Założmy początkowo (rys. 4.7), że $Clock = 0$, $D = 1$, $Y = 0$ i $Q = 0$. W takim przypadku przerzutnik *Master* jest odcięty, a przerzutnik *Slave* jest aktywny i wartość jego wejścia ($Y = 0$) jest powtarzana na wyjściu ($Q = 0$). Stan wyjścia ($Q = 0$) nie ulega zmianie. Po zmianie impulsu zegarowego z $Clock = 0$ na $Clock = 1$ przerzutnik *Master* jest aktywny i wartość jego wejścia ($D = 1$) jest powtarzana na wyjściu ($Y = 1$), a przerzutnik *Slave* jest odcięty. Stan wyjścia ($Q = 0$) w dalszym ciągu nie ulega zmianie. Po zmianie impulsu zegarowego z $Clock = 1$ na $Clock = 0$ przerzutnik *Master* jest odcięty, a przerzutnik *Slave* jest aktywny i wartość jego wejścia ($Y = 1$) jest powtarzana na wyjściu ($Q = 1$). Stan wyjścia Q ulega zmianie z $Q = 0$ na $Q = 1$. W przerzutniku *D-MS* zmiana stanu przerzutnika *Master* zachodzi po zmianie impulsu zegarowego z $Clock = 0$ na $Clock = 1$, natomiast zmiana stanu przerzutnika *Slave* ma miejsce po zmianie impulsu zegarowego z $Clock = 1$ na $Clock = 0$. Symbol graficzny przerzutnika *D-MS* jest pokazany na rysunku 4.8.



Rys. 4.6. Schemat blokowy przerzutnika *D-MS*



Rys. 4.7. Diagramy czasowe ilustrujące kolejność zdarzeń zachodzących w przerzutniku *D-MS*



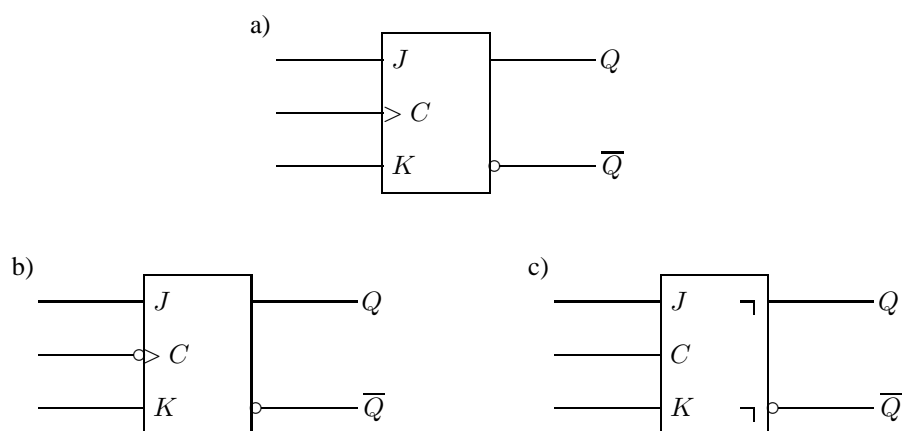
Rys. 4.8. Symbol graficzny przerzutnika D – MS

4.2.5 Przerzutnik JK

Przerzutnik JK ma dwa wejścia informacyjne (J, K). Tablica funkcji opisująca jego działanie jest pokazana na rysunku 4.9. W tablicy tej $Q(t)$ oznacza stan aktualny, a $Q(t + 1)$ stan następny. Symbole graficzne przerzutnika JK są pokazane na rysunku 4.10.

J	K	$Q(t + 1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

Rys. 4.9. Tablica funkcji przerzutnika JK



Rys. 4.10. Symbole graficzne przerzutnika JK : wyzwalanego zboczem dodatnim (a)), wyzwalanego zboczem ujemnym (b)) oraz symbol graficzny przerzutnika JK – MS (c))

4.2.6 Tablice wzbudzeń przerzutników D i JK

Tablica, w której są wyszczególnione kombinacje wejść przerzutnika dla danej zmiany stanu, nazywa się tablicą wzbudzeń (*excitation table*). Tablica wzbudzeń przerzutnika D jest pokazana na rysunku 4.11, a przerzutnika JK na rysunku 4.12. Symbol x w tablicach wzbudzeń oznacza, że odpowiednia zmienna wejściowa może przyjmować dowolną wartość, tzn. że nie ma żadnego znaczenia, czy jest równa 0, czy 1.

$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

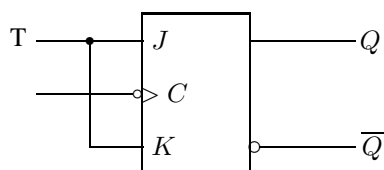
Rys. 4.11. Tablica wzbudzeń przerzutnika D

$Q(t)$	$Q(t+1)$	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Rys. 4.12. Tablica wzbudzeń przerzutnika JK

4.2.7 Przerzutnik T

Synchroniczny przerzutnik T można otrzymać z przerzutnika JK przez zwarcie wejść J i K w jedno wejście oznaczone jako T . Symbol graficzny przerzutnika T otrzymanego z przerzutnika JK jest pokazany na rysunku 4.13.

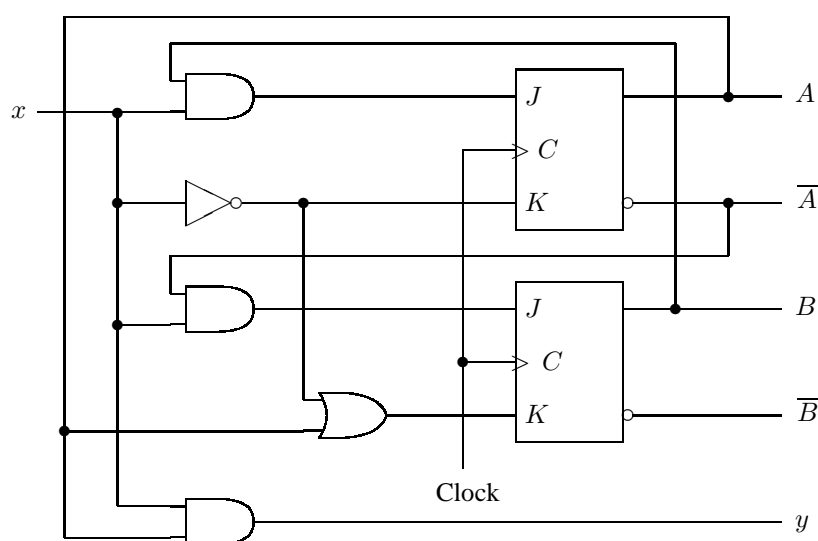


Rys. 4.13. Symbol graficzny przerzutnika T

4.3 Analiza synchronicznych układów sekwencyjnych

Układy sekwencyjne, w których zmiany stanów wyjść przerzutników są synchronizowane impulsami zegarowymi, są nazywane układami synchronicznymi.

Analiza układu sekwencyjnego jest procesem określającym relacje funkcjonalne istniejące między jego wejściami, wyjściami i stanami przerzutników. Jako przykład dokonajmy analizy układu sekwencyjnego pokazanego na rysunku 4.14.



Rys. 4.14. Przykład układu sekwencyjnego

Układ ten ma jedną zmienną wejściową x , jedną zmienną wyjściową y , dwa przerzutniki JK i 5 bramek logicznych.

Równania wejść przerzutników z rysunku 4.14 są następujące:

$$J_A = Bx \quad (\text{równanie dla wejścia } J \text{ przerzutnika } A),$$

$$K_A = \bar{x} \quad (\text{równanie dla wejścia } K \text{ przerzutnika } A),$$

$$J_B = \bar{A}x \quad (\text{równanie dla wejścia } J \text{ przerzutnika } B),$$

$$K_B = A + \bar{x} \quad (\text{równanie dla wejścia } K \text{ przerzutnika } B).$$

Analizowany układ ma również wyjście y , będące funkcją zmiennej wejściowej x i stanu przerzutnika A . Wyjście to można przedstawić za pomocą wyrażenia

$$y = Ax$$

Zachowanie się układu sekwencyjnego jest określone tablicą przedstawiającą stany aktualne przerzutników, wejście ($x(t)$), stany następne przerzutników i wyjście ($y(t)$). Tablica stanów analizowanego układu jest pokazana na rysunku 4.15.

Stan aktualny		Wejście $x(t)$	Stan następny		Wyjście $y(t)$
$A(t)$	$B(t)$		$A(t+1)$	$B(t+1)$	
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	0	0
1	1	1	1	0	1

Rys. 4.15. Tablica stanów układu z rysunku 4.14

W tym przypadku stany aktualne przerzutników A i B oraz wejście x mogą przyjmować jedną z ośmiu możliwych wartości dwójkowych, wyszczególnionych w kolumnach odpowiadających stanowi aktualnemu i wejściu. W kolumnach odpowiadających stanowi następnemu są przedstawione wartości dwójkowe stanów przerzutników po pojawieniu się impulsu zegara. Stany następne są wyznaczone na podstawie równań wejść przerzutników i tablicy wzbudzeń przerzutnika JK (rys. 4.12). Dla stanu aktualnego $A = 0, B = 0$ i wejścia $x = 0$ (pierwszy wiersz tablicy stanów z rysunku 4.15) otrzymujemy następujące wartości równań wejść przerzutników:

$$J_A = Bx = 0, \quad K_A = \bar{x} = 1$$

$$J_B = \bar{A}x = 0, \quad K_B = \bar{x} + A = 1$$

dla których $Q(t+1) = 0$ dla przerzutnika A i przerzutnika B (pierwszy wiersz tablicy wzbudzeń przerzutnika JK z rysunku 4.12).

Dla stanu aktualnego $A = 0, B = 0$ i wejścia $x = 1$ (drugi wiersz tablicy stanów z rysunku 4.15) otrzymujemy następujące wartości równań wejść:

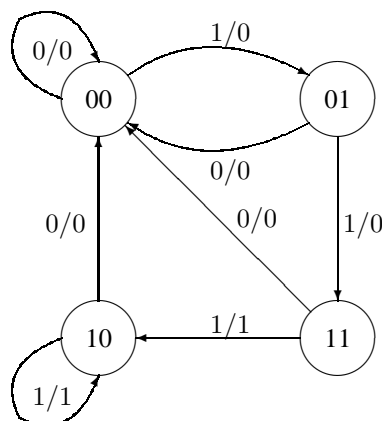
$$J_A = Bx = 0, \quad K_A = \bar{x} = 0$$

$$J_B = \bar{A}x = 1, \quad K_B = A + \bar{x} = 0$$

dla których $Q(t+1) = 0$ dla przerzutnika A (pierwszy wiersz tablicy wzbudzeń przerzutnika JK z rysunku 4.12) i $Q(t+1) = 1$ przerzutnika B (drugi wiersz tablicy wzbudzeń przerzutnika JK z rysunku 4.12). W podobny sposób można wyznaczyć stany następne dla

kolejnych wierszy w tablicy stanów. Wartość wyjścia y w tablicy stanów z rysunku 4.15 jest funkcją zmiennej wejściowej x i stanu aktualnego przerzutnika A ($y = Ax$).

Informację zawartą w tablicy stanów z rysunku 4.15 można przedstawić za pomocą grafu stanów pokazanego na rysunku 4.16.



Rys. 4.16. Graf stanów układu sekwencyjnego z rysunku 4.14

W grafie tym stany są reprezentowane przez kółka, a przejścia pomiędzy stanami przez łuki skierowane. Liczba dwójkowa wewnątrz każdego kółka identyfikuje stany przerzutników. Łuki skierowane są opisane przez parę liczb dwójkowych oddzielonych kreską ukośną. Pierwsza z tych liczb jest wartością wejścia (x) w stanie aktualnym. Druga liczba jest wartością wyjścia (y) w stanie aktualnym dla danej wartości wejścia (x).

4.4 Projektowanie synchronicznych układów sekwencyjnych

Projektowanie synchronicznych układów sekwencyjnych omówimy na przykładach.

Przykład 4.1. Zaprojektować układ sekwencyjny, którego graf stanów jest pokazany na rysunku 4.16, wykorzystując przerzutniki JK .

Do realizacji układu potrzebne są dwa przerzutniki. Zmienną wejściową oznaczmy przez x , zmienną wyjściową przez y . Przerzutniki oznaczmy przez A i B . Stany następne przerzutników A i B , będące funkcjami ich stanów aktualnych i wejścia x , wpisujemy do tablicy stanów z rysunku 4.17. Tablicę stanów wyznaczamy na podstawie grafu stanów projektowanego układu, a odpowiadającą jej tablicę wejść przerzutników JK (rys. 4.17) na podstawie tablicy wzbudzeń przerzutnika JK . Tablica wejść przerzutników JK zawiera wartości wejść przerzutników powodujących wymagane zmiany stanów.

$Q(t)$		Wejście	$Q(t+1)$		Wyjście	Wejścia przerzutników			
A	B	x	A	B	y	J_A	K_A	J_B	K_B
0	0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	0	x	1	x
0	1	0	0	0	0	0	x	x	1
0	1	1	1	1	0	1	x	x	0
1	0	0	0	0	0	x	1	0	x
1	0	1	1	0	1	x	0	0	x
1	1	0	0	0	0	x	1	x	1
1	1	1	1	0	1	x	0	x	1

Rys. 4.17. Tablica stanów projektowanego układu sekwencyjnego i odpowiadająca jej tablica wejść przerzutników (przykład 4.1)

Na podstawie tablicy wejść przerzutników otrzymujemy następujące funkcje logiczne zmiennych A, B (stan aktualny), x :

$$J_A(A, B, x) = \sum(3) + \sum n(4, 5, 6, 7)$$

$$K_A(A, B, x) = \sum(4, 6) + \sum n(0, 1, 2, 3)$$

$$J_B(A, B, x) = \sum(1) + \sum n(2, 3, 6, 7)$$

$$K_B(A, B, x) = \sum(2, 6, 7) + \sum n(0, 1, 4, 5)$$

Na podstawie tablicy stanów otrzymujemy równanie wyjścia

$$y(A, B, x) = \sum(5, 7)$$

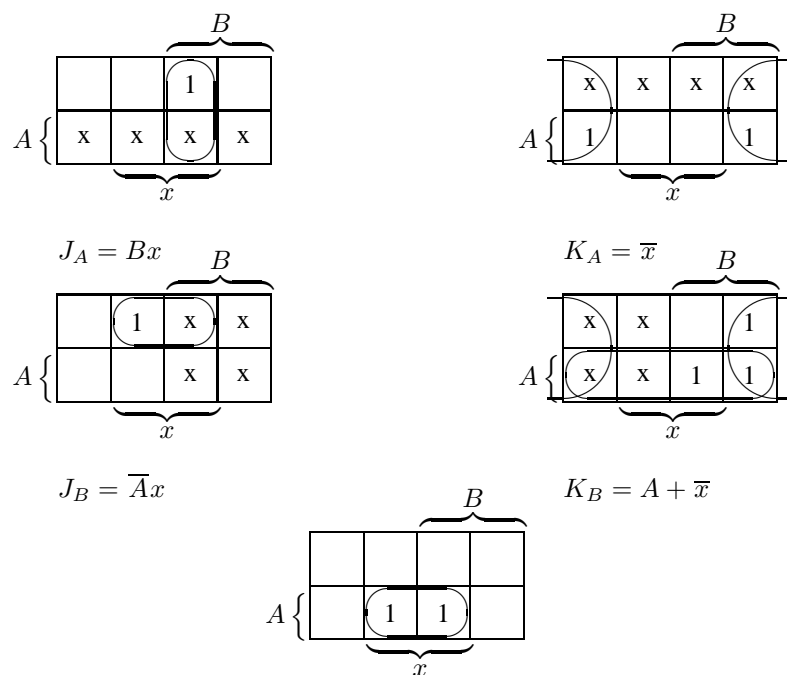
Po minimalizacji (rys. 4.18) otrzymujemy następujące funkcje:

$$J_A(A, B, x) = Bx \quad K_A(A, B, x) = \bar{x}$$

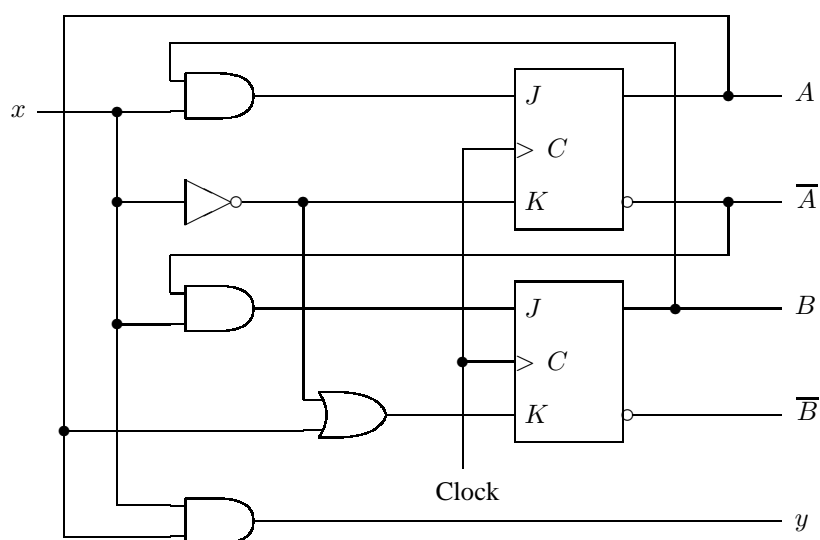
$$J_B(A, B, x) = \bar{A}x \quad K_B(A, B, x) = A + \bar{x}$$

$$y(A, B, x) = Ax$$

Na rysunku 4.19 jest pokazany schemat logiczny projektowanego układu sekwencyjnego.



Rys. 4.18. Minimalizacja funkcji logicznych (przykład 4.1)



Rys. 4.19. Schemat logiczny projektowanego układu sekwencyjnego (przykład 4.1)

Przykład 4.2. Zaprojektować układ sekwencyjny, którego graf stanów jest pokazany na rysunku 4.16, wykorzystując przerzutniki D .

Tablica stanów projektowanego układu i odpowiadająca jej tablica wejść przerzutników D są pokazane na rysunku 4.20.

$Q(t)$		Wejście x	$Q(t+1)$		Wyjście y	Wejścia przerzutników	
A	B		A	B		D_A	D_B
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1
0	1	0	0	0	0	0	0
0	1	1	1	1	0	1	1
1	0	0	0	0	0	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	1	1	0	1	1	0

Rys. 4.20. Tablica stanów projektowanego układu sekwencyjnego i odpowiadająca jej tablica wejść przerzutników (przykład 4.2)

Na podstawie tablicy wejść przerzutników otrzymujemy funkcje logiczne zmiennych A, B (stan aktualny), x :

$$D_A(A, B, x) = \sum(3, 5, 7) \quad D_B(A, B, x) = \sum(1, 3)$$

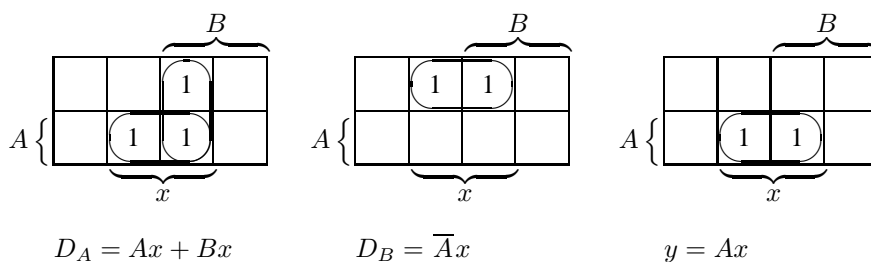
Na podstawie tablicy stanów otrzymujemy równanie wyjścia

$$y(A, B, x) = \sum(5, 7)$$

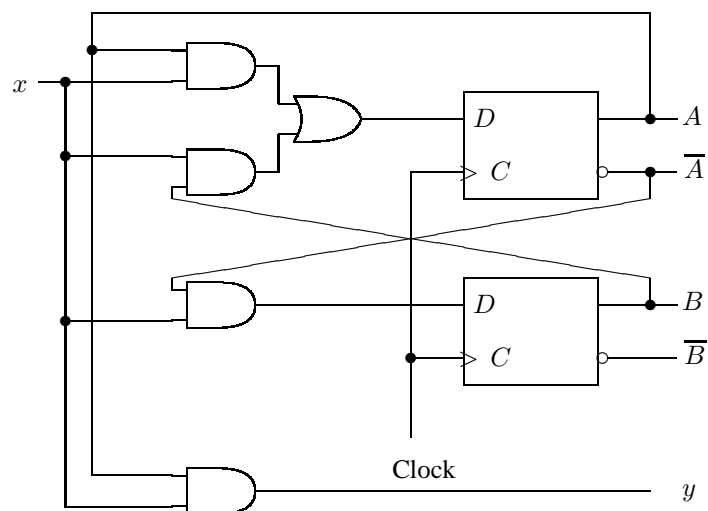
Po minimalizacji (rys. 4.21) otrzymujemy następujące funkcje:

$$D_A(A, B, x) = Ax + Bx \quad D_B(A, B, x) = \bar{A}x \quad y(A, B, x) = Ax$$

Na rysunku 4.22 jest pokazany schemat logiczny projektowanego układu.



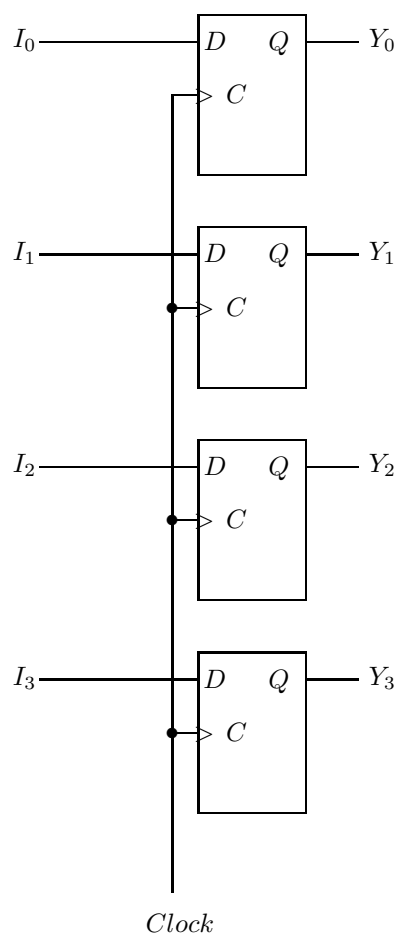
Rys. 4.21. Minimalizacja funkcji logicznych (przykład 4.2)



Rys. 4.22. Schemat logiczny projektowanego układu sekwencyjnego (przykład 4.2)

4.5 Rejestry

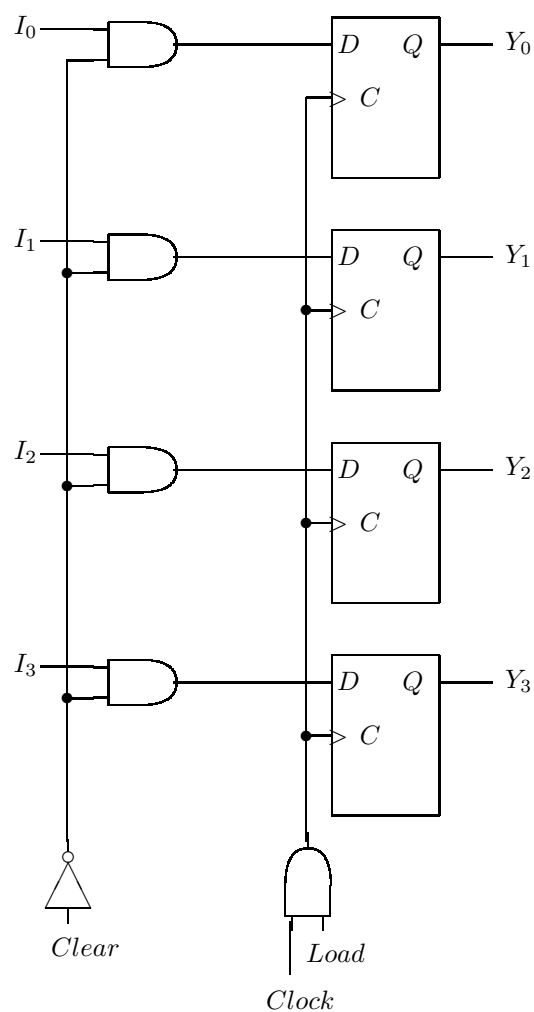
Rejestrem nazywamy układ sekwencyjny do przechowywania informacji dwójkowej. Na rysunku 4.23 jest pokazany 4-bitowy rejestr zbudowany z przerzutników D ([4]).



Rys. 4.23. Rejestr 4-bitowy

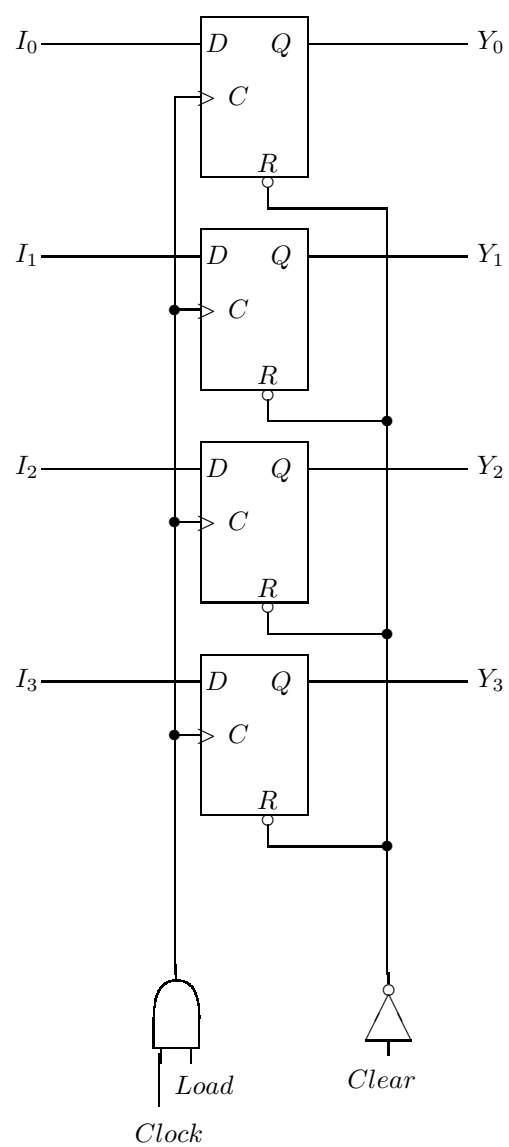
Linie wejściowe: I_0, I_1, I_2, I_3 są połączone z wejściami D przerzutników. Pojawienie się impulsu zegarowego (zbocza dodatniego) powoduje wprowadzenie informacji z linii wejściowych do przerzutników. Wyjścia przerzutników są połączone z liniami wyjściowymi: Y_0, Y_1, Y_2, Y_3 .

Na rysunku 4.24 jest pokazany rejestr z sygnałami sterującymi ładowaniem (*Load*) i zerowaniem (*Clear*) ([4]). Pojawienie się jedynki logicznej na wejściu *Load* i impulsu zegarowego na wejściu (*Clock*) powoduje wprowadzenie informacji z linii wejściowych do przerzutników. Pojawienie się jedynki logicznej na wejściu *Clear* i impulsu zegarowego na wejściu *Clock* powoduje wprowadzenie zera logicznego do przerzutników.



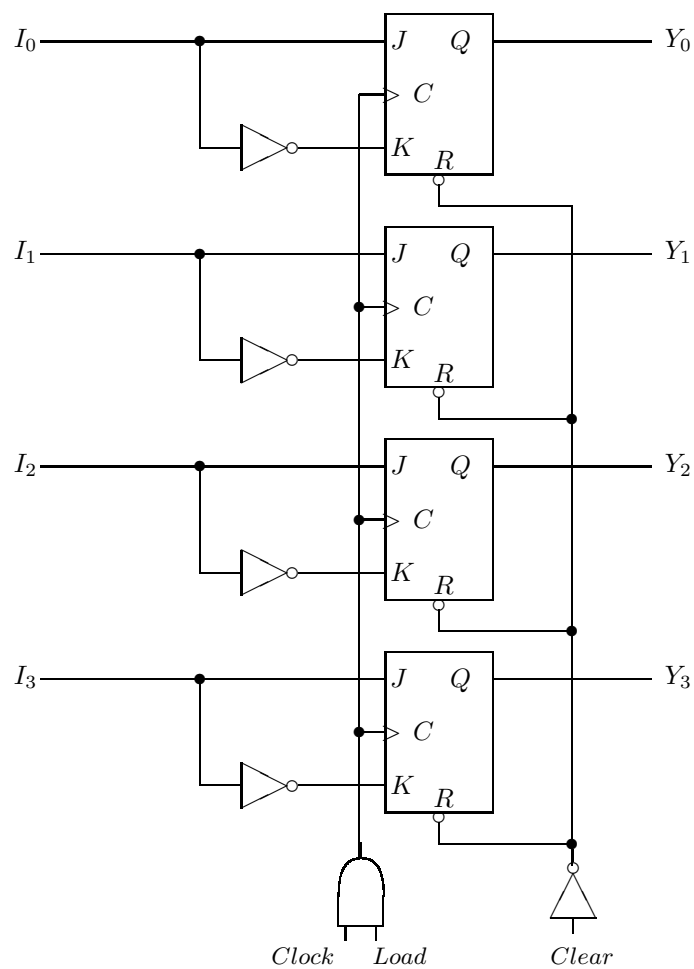
Rys. 4.24. Rejestr 4-bitowy z sygnałami *Load* i *Clear*

Przerzutniki mają dodatkowe wejście zerujące *R* (*reset*, *clear*), działające asynchronicznie (bez udziału impulsów zegarowych). Na rysunku 4.25 jest pokazany rejestr z zerowaniem asynchronicznym ([4]).



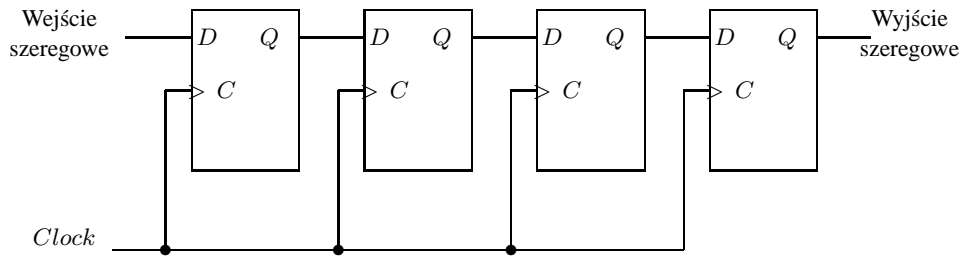
Rys. 4.25. Rejestr 4-bitowy z zerowaniem asynchronicznym

Na rysunku 4.26 jest pokazany rejestr zbudowany z przerzutników JK .



Rys. 4.26. Rejestr 4-bitowy zbudowany z przerzutników JK

Rejestr umożliwiający przesuwanie swojej zawartości jest nazywany rejestrem przesuwającym (*shift register*). Schemat logiczny prostego rejestru przesuwającego w prawo jest pokazany na rysunku 4.27 ([4]).



Rys. 4.27. Schemat logiczny rejestru przesuwającego w prawo

4.6 Liczniki

Licznikiem nazywamy układ logiczny sekwencyjny zliczający impulsy wejściowe (impulsy zegarowe) ([1, 2]). Pojawienie się kolejnego impulsu wejściowego powoduje zmianę stanu licznika. Ogólnie jeśli licznik ma m różnych stanów, przez który przechodzi cyklicznie, to określa się go jako licznik modulo m .

Przykład 4.3. Zaprojektować licznik modulo 16, wykorzystując przerzutniki JK .

Tablica stanów projektowanego licznika i odpowiadająca jej tablica wejść przerzutników są pokazane na rysunku 4.28. Na podstawie tablicy wejść przerzutników otrzymamy równania wejść:

$$J_A(A, B, C, D) = \sum(7) + \sum n(8, 9, 10, 11, 12, 13, 14, 15)$$

$$K_A(A, B, C, D) = \sum(15) + \sum n(0, 1, 2, 3, 4, 5, 6, 7)$$

$$J_B = \sum(3, 11) + \sum n(4, 5, 6, 7, 12, 13, 14, 15)$$

$$K_B = \sum(7, 15) + \sum n(0, 1, 2, 3, 8, 9, 10, 11)$$

$$J_C = \sum(1, 5, 9, 13) + \sum n(2, 3, 6, 7, 10, 11, 14, 15)$$

$$K_C = \sum(3, 7, 11, 15) + \sum n(0, 1, 4, 5, 8, 9, 12, 13)$$

$$J_D = \sum(0, 2, 4, 6, 8, 10, 12, 14) + \sum n(1, 3, 5, 7, 9, 11, 13, 15)$$

$$K_D = \sum(1, 3, 5, 7, 9, 11, 13, 15) + \sum n(0, 2, 4, 6, 8, 10, 12, 14)$$

	$Q(t)$	$Q(t+1)$	Wejścia przerzutników							
	$ABCD$	$ABCD$	J_A	K_A	J_B	K_B	J_C	K_C	J_D	K_D
0	0000	0001	0	x	0	x	0	x	1	x
1	0001	0010	0	x	0	x	1	x	x	1
2	0010	0011	0	x	0	x	x	0	1	x
3	0011	0100	0	x	1	x	x	1	x	1
4	0100	0101	0	x	x	0	0	x	1	x
5	0101	0110	0	x	x	0	1	x	x	1
6	0110	0111	0	x	x	0	x	0	1	x
7	0111	1000	1	x	x	1	x	1	x	1
8	1000	1001	x	0	0	x	0	x	1	x
9	1001	1010	x	0	0	x	1	x	x	1
10	1010	1011	x	0	0	x	x	0	1	x
11	1011	1100	x	0	1	x	x	1	x	1
12	1100	1101	x	0	x	0	0	x	1	x
13	1101	1110	x	0	x	0	1	x	x	1
14	1110	1111	x	0	x	0	x	0	1	x
15	1111	0000	x	1	x	1	x	1	x	1

Rys. 4.28. Tablica stanów projektowanego licznika i odpowiadająca jej tablica wejść przerzutników (przykład 4.3)

Po minimalizacji (rys. 4.29) otrzymujemy następujące równania wejść przerzutników:

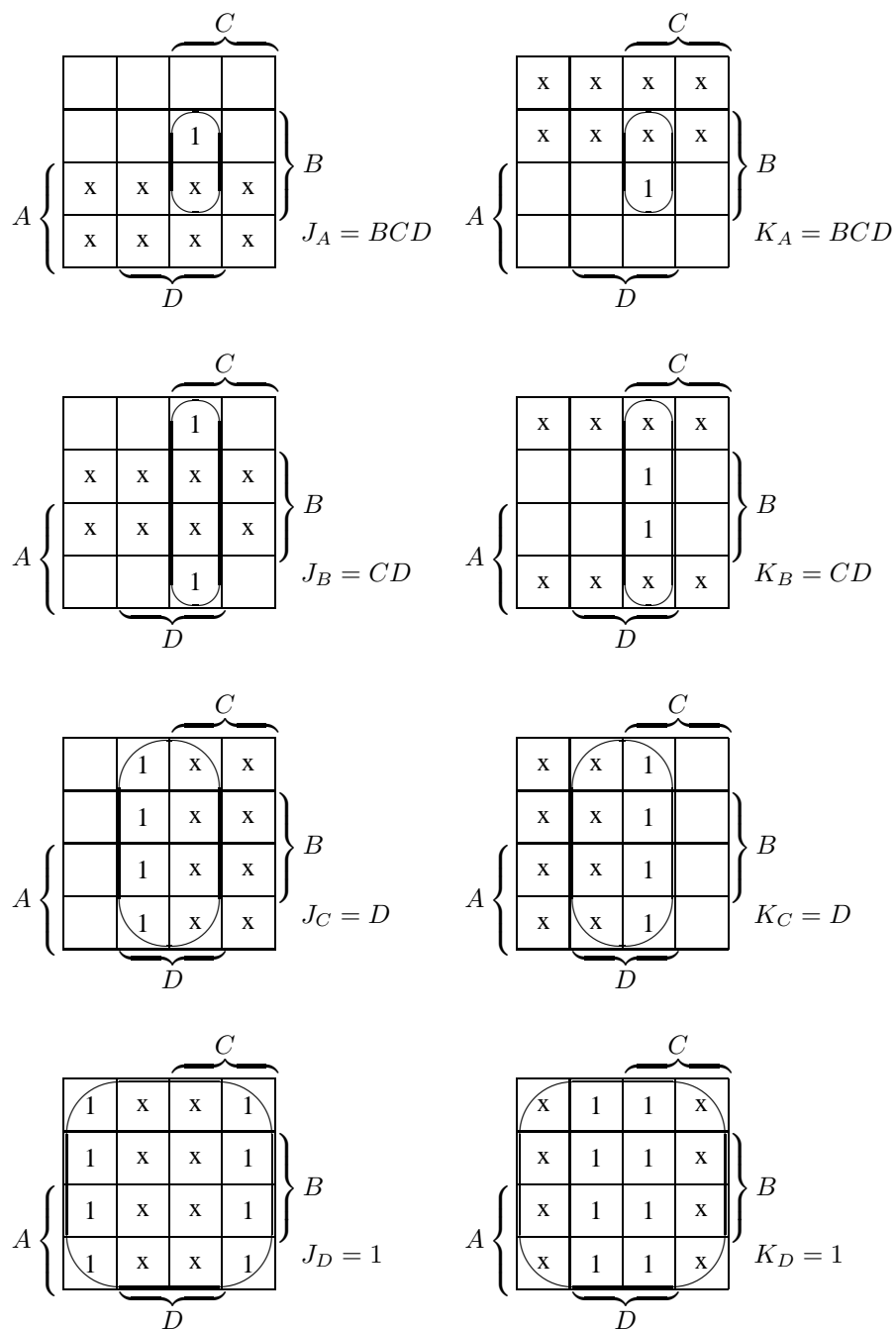
$$J_A = K_A = BCD \quad J_B = K_B = CD$$

$$J_C = K_C = D \quad J_D = K_D = 1$$

Schemat logiczny projektowanego licznika pokazano na rysunku 4.30.

4.7 Podsumowanie

W rozdziale 4 zostały przedstawione elementarne układy sekwencyjne (przerzutniki) realizowane jako układy małego stopnia scalenia ([6]). Omówiona została klasyczna metoda projektowania synchronicznych układów sekwencyjnych wykorzystująca układy małego stopnia scalenia. Metoda ta z odpowiednimi modyfikacjami pozostaje aktualna w przypadku projektowania układów sekwencyjnych z wykorzystaniem układów średniego i wielkiego stopnia scalenia ([2]). Podane zostały podstawowe wiadomości na temat rejestrów i liczników.



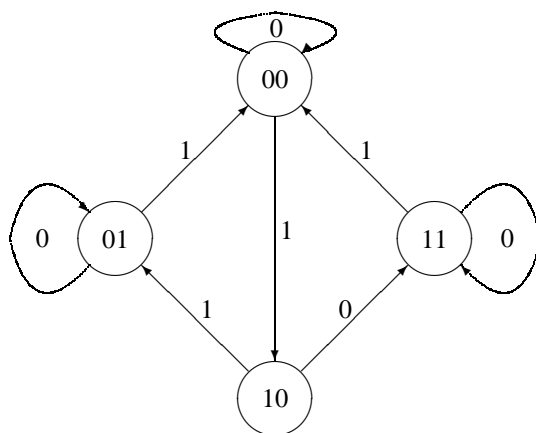
Rys. 4.29. Minimalizacja funkcji logicznych (przykład 4.3)

- [6] Sasal W.: *Układy scalone serii UCY74LS i UCY74S. Parametry i zastosowania*, WKŁ, 1993.

Z a d a n i a

Zadanie 4.1. Wykorzystując przerzutniki JK zaprojektować układ sekwencyjny, którego graf stanów jest pokazany na rysunku 4.31.

Zadanie 4.2. Wykorzystując przerzutniki D zaprojektować układ sekwencyjny, którego graf stanów jest pokazany na rysunku 4.31.



Rys. 4.31. Graf stanów układu sekwencyjnego

Zadanie 4.3. Zaprojektować 4-bitowy sumator szeregowy.

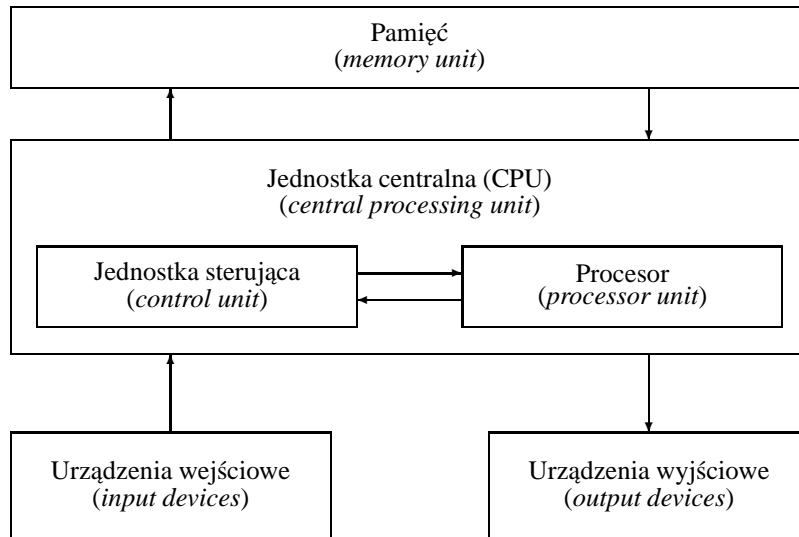
Zadanie 4.4. Wykorzystując przerzutniki D zaprojektować 4-bitowy rejestr przesuwający w prawo i w lewo.

Zadanie 4.5 Wykorzystując przerzutniki JK zaprojektować licznik modulo 8.

5 PROCESOR

5.1 Wstęp

Schemat blokowy komputera jest pokazany na rysunku 5.1 ([2]).



Rys. 5.1. Schemat blokowy komputera

Pamięć przechowuje programy i dane (wejściowe, wyjściowe, pośrednie). Procesor wykonuje operacje wyszczególnione przez program. Jednostka sterująca nadzoruje przepływ informacji w komputerze. Procesor i jednostka sterująca tworzą jednostkę centralną (CPU). Jednostka centralna zrealizowana w postaci układu scalonego jest nazywana mikroprocesorem.

Programy i dane przygotowane przez użytkownika są przesyłane do pamięci za pomocą urządzeń wejściowych. Wyniki obliczeń są wyprowadzane za pomocą urządzeń wyjściowych.

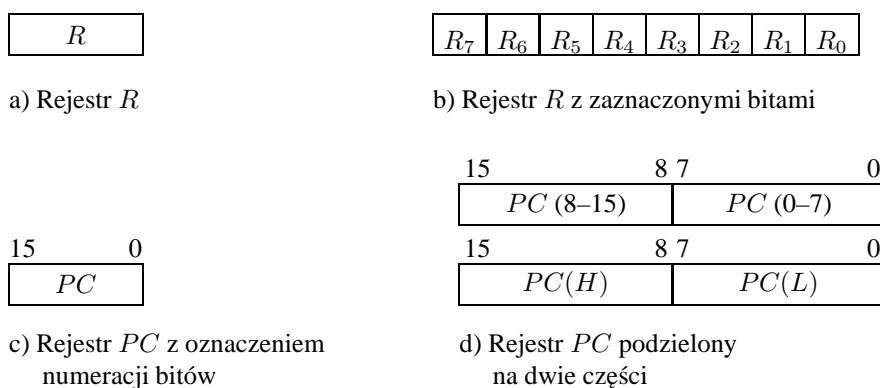
W rozdziale tym zaprojektujemy prosty, przykładowy procesor ([2]).

5.2 Mikrooperacje przesyłania między rejestrami

Mikrooperacja (*microoperation*) jest to elementarna operacja, jaka może być wykonana podczas jednego impulsu zegarowego na danych przechowywanych w rejestrach ([1]). Przykładami mikrooperacji są: ładowanie, zerowanie i przesuwanie w rejestrach.

Rejestry komputera będziemy oznaczać dużymi literami, dobranymi tak, aby jednoznacznie opisywały funkcję rejestru. Po literach będą mogły występować liczby. Na przykład rejestr zawierający adres słowa pamięci jest nazywany rejestrem adresowym pamięci. Będziemy go oznaczać literami *MAR* (*memory address register*). Innymi oznaczeniami rejestrów mogą być na przykład: *PC* (*program counter*), *IR* (*instruction register*), *R1*, *R2*.

Poszczególne bity w n -bitowym rejestrze będziemy numerować kolejno od 0 do $n - 1$, zaczynając od prawej strony. Na rysunku 5.2 są pokazane sposoby rysowania schematów blokowych rejestrów.



Rys. 5.2. Schematy blokowe rejestrów

Przesyłanie między rejestrami realizują mikrooperacje przesań. Przesyłanie może się odbywać równoległe lub szeregowo. Przesyłanie równoległe polega na jednoczesnym przesyłaniu wszystkich bitów z rejestru źródłowego do rejestru docelowego i odbywa się w czasie trwania jednego impulsu zegarowego. Mikrooperację przesyłania równoległego z rejestru $R1$ do rejestru $R2$ będziemy zapisywać w następujący sposób:

$$R2 \leftarrow R1$$

Przesyłanie z jednego rejestru do drugiego powinno odbywać się wtedy, kiedy pojawi się określony sygnał taktujący T_1 , co możemy to zapisać w następujący sposób:

$$if (T_1 = 1) then (R2 \leftarrow R1)$$

lub inaczej

$$T_1 : R2 \leftarrow R1$$

Zapis

$$\text{if } (T_1 = 1) \text{ then } (R3 \leftarrow R1)$$

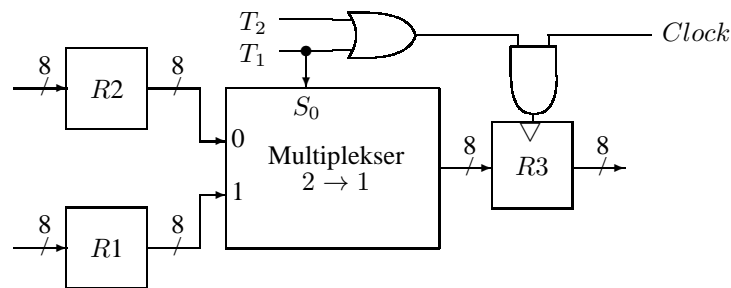
$$\text{else if } (T_2 = 1) \text{ then } (R3 \leftarrow R2)$$

oznacza, że zawartość rejestru $R1$ ma być przesłana do rejestru $R3$ wtedy, kiedy $T_1 = 1$. Kiedy $T_1 = 0$, zawartość rejestru $R2$ ma być przesłana do rejestru $R3$ wtedy, kiedy $T_2 = 1$. Można to zapisać inaczej

$$T_1 : R3 \leftarrow R1$$

$$\overline{T_1}T_2 : R3 \leftarrow R2$$

Na rysunku 5.3 jest pokazany schemat blokowy układu cyfrowego, realizującego dwie powyższe mikrooperacje ([2]).



Rys. 5.3. Schemat blokowy układu cyfrowego realizującego mikrooperacje $T_1 : R3 \leftarrow R1$ i $\overline{T_1}T_2 : R3 \leftarrow R2$

Pojawienie się sygnału taktującego T_1 i impulsu zegarowego $Clock$ spowoduje przesłanie zawartości rejestru $R1$ do rejestru $R3$. Pojawienie się sygnału taktującego T_2 i impulsu zegarowego $Clock$ spowoduje przesłanie zawartości rejestru $R2$ do rejestru $R3$.

5.3 Mikrooperacje arytmetyczne i logiczne

Na rysunku 5.4 są pokazane przykładowe mikrooperacje arytmetyczne.

Mikrooperacja	Opis
$R1 \leftarrow R1 + R2$	Zawartość rejestru $R1$ plus zawartość rejestru $R2$ jest przesyłana do rejestru $R1$
$R1 \leftarrow \overline{R1} + 1$	Uzupełnienie do 2 zawartości rejestru $R1$
$R1 \leftarrow R1 + \overline{R2} + 1$	Zawartość rejestru $R1$ minus zawartość rejestru $R2$ jest przesyłana do rejestru $R1$
$R1 \leftarrow R1 + 1$	Zwiększenie o 1 zawartości rejestru $R1$

Rys. 5.4. Przykładowe mikrooperacje arytmetyczne

Mikrooperacje logiczne są wykonywane na bitach przechowywanych w rejestrach. Każdy bit jest traktowany oddzielnie jako zmienna dwójkowa. Na przykład, mikrooperacja logiczna AND zawartości rejestrów $R1$ i $R2$ zapisana w następujący sposób:

$$T_1 : R1 \leftarrow R1 \wedge R2$$

oznacza mikrooperację wykonywaną na poszczególnych bitach rejestrów. Na rysunku 5.5 są pokazane przykładowe mikrooperacje logiczne.

Mikrooperacja	Opis
$R1 \leftarrow R1 \wedge R2$	AND
$R1 \leftarrow R1 \vee R2$	OR
$R1 \leftarrow R1 \oplus R2$	XOR
$R1 \leftarrow \overline{R1}$	NOT

Rys. 5.5. Przykładowe mikrooperacje logiczne

5.4 Mikrooperacje przesuwania

Mikrooperacje przesuwania są używane do szeregowego przesuwania danych. Zawartość rejestru może być przesuwana w lewo lub w prawo o jedną pozycję. Na rysunku 5.6 są pokazane przykładowe mikrooperacje przesuwania.

Mikrooperacja	Opis
$R \leftarrow shl\ R$ (<i>shift left</i>)	Przesunięcie w lewo zawartości rejestru R (skrajnie prawy przerzutnik jest zerowany)
$R \leftarrow shr\ R$ (<i>shift right</i>)	Przesunięcie w prawo zawartości rejestru R (skrajnie lewy przerzutnik jest zerowany)
$R \leftarrow rol\ R$ (<i>rotate left</i>)	Przesunięcie cykliczne w lewo zawartości rejestru R (zawartość skrajnie prawego przerzutnika staje się równa poprzedniej zawartości skrajnie lewego przerzutnika)
$R \leftarrow ror\ R$ (<i>rotate right</i>)	Przesunięcie cykliczne w prawo zawartości rejestru R (zawartość skrajnie lewego przerzutnika staje się równa poprzedniej zawartości skrajnie prawego przerzutnika)

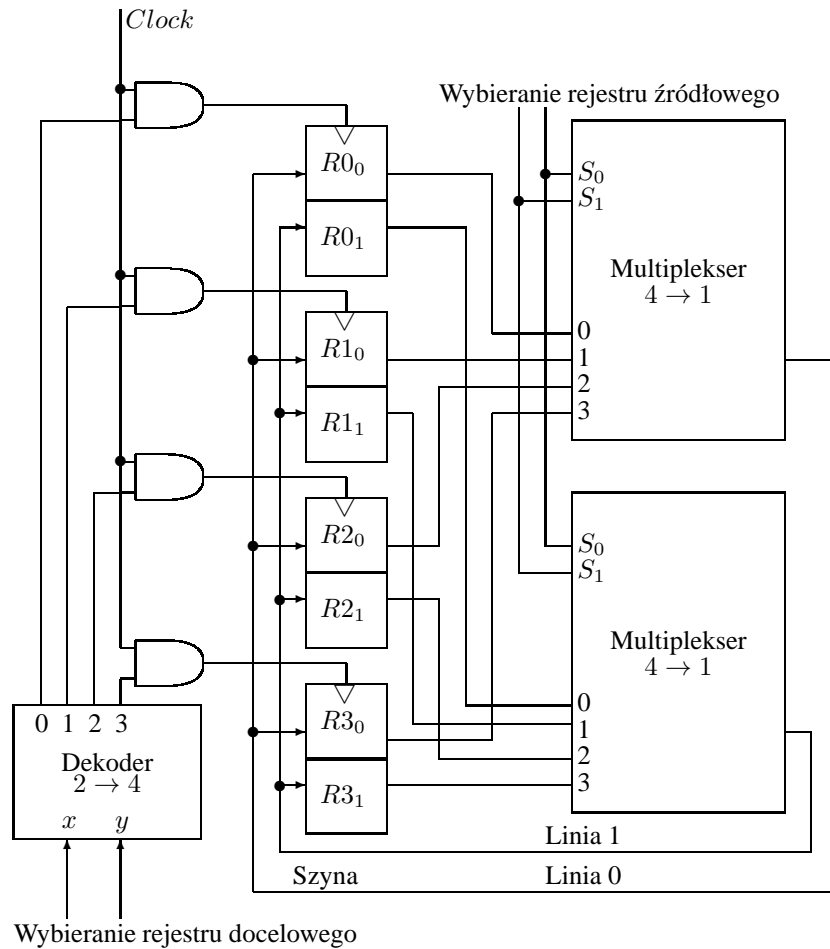
Rys. 5.6. Przykładowe mikrooperacje przesuwania

5.5 Przesyłanie za pomocą szyn

Przesyłanie między rejestrami jest realizowane za pomocą szyny (*bus*), to znaczy połączeń między wyjściami a wejściami rejestrów. Wprowadzanie informacji na szynę może być sterowane za pomocą multiplexerów. Rozwiązanie takie (dla czterech rejestrów 2-bitowych) jest pokazane na rysunku 5.7 ([2]). Rozważmy mikrooperację przesyłania

$$R1 \leftarrow R2$$

Zmienne sterujące multiplexerów muszą wybrać rejestr $R2$ jako rejestr źródłowy ($S_1 = 1, S_0 = 0$), powodując umieszczenie na szynie danych zawartości rejestru $R2$ ($R2_0$ na linii 0, $R2_1$ na linii 1). Zmienne sterujące dekodera muszą wybrać rejestr $R1$ jako rejestr docelowy ($x = 0$ i $y = 1$). Przesyłanie jest realizowane w chwili pojawienia się impulsu zegarowego.



Rys. 5.7. Przesyłanie między rejestrami za pomocą szyny

5.6 Przesyłanie do pamięci i z pamięci

Pamięć będziemy oznaczać przez M . Do adresowania pamięci będziemy używać rejestru MAR . Przesyłanie z pamięci do rejestru buforowego pamięci MBR (*memory buffer register*) jest mikrooperacją odczytu, którą będziemy zapisywać w następujący sposób:

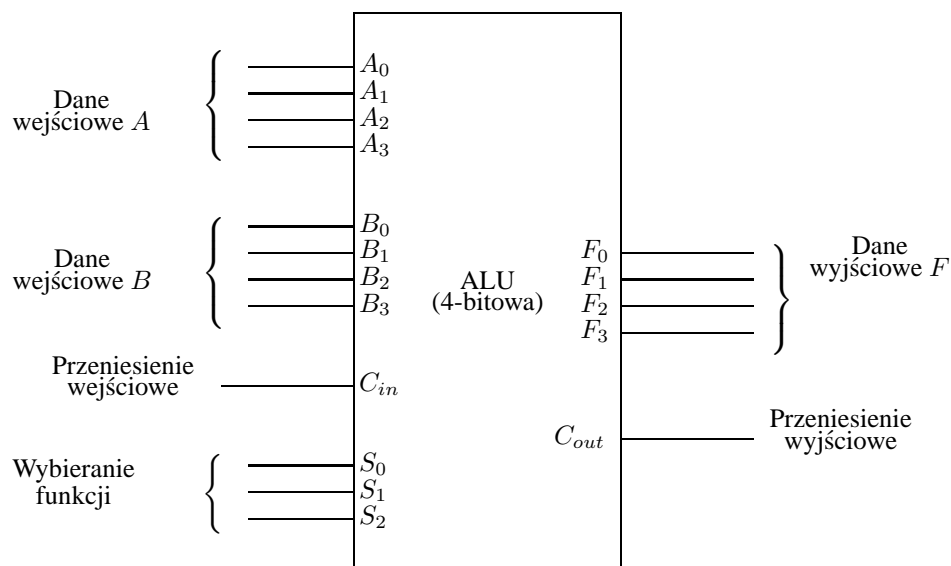
$$Read : MBR \leftarrow M[MAR]$$

Przesyłanie z rejestru buforowego pamięci MBR do pamięci jest mikrooperacją zapisu, którą będziemy zapisywać w następujący sposób:

$$Write : M[MAR] \leftarrow MBR$$

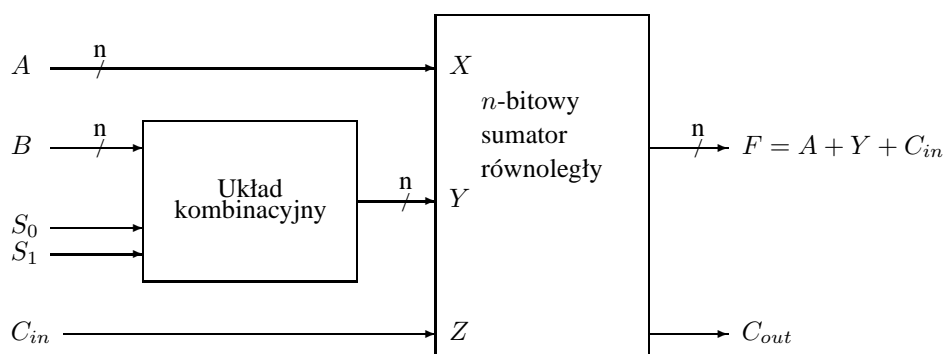
5.7 Projektowanie jednostki arytmetyczno-logicznej

Jednostka arytmetyczno-logiczna ALU (*arithmetic logic unit*) jest układem kombinacyjnym, wykonującym funkcje arytmetyczne i logiczne. Na rysunku 5.8 jest pokazany schemat blokowy przykładowej, 4-bitowej jednostki arytmetyczno-logicznej ([2]).



Rys. 5.8. Schemat blokowy przykładowej, 4-bitowej jednostki arytmetyczno-logicznej

Wewnętrzna konstrukcja jednostki arytmetyczno-logicznej zależy od realizowanych przez nią funkcji. Podstawowym elementem jednostki arytmetycznej jest n -bitowy sumator równoległy. Na rysunku 5.9 jest pokazany schemat logiczny przykładowej jednostki arytmetycznej ([2]).



Rys. 5.9. Schemat blokowy przykładowej jednostki arytmetycznej

Wartość wyjścia F sumatora jest obliczana według wyrażenia

$$F = A + Y + C_{in}$$

gdzie A i B są n -bitowymi liczbami dwójkowymi odpowiednio na wejściach X i Y sumatora. Wybierając za pomocą linii S_1, S_0 układu kombinacyjnego różne wartości na wejściu Y , można otrzymać różne funkcje arytmetyczne na wyjściu F pokazane na rysunku 5.10 ([2]).

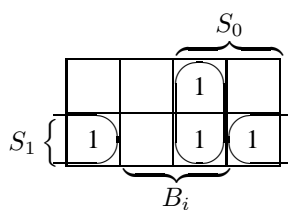
Wejścia		Wejście Y	$F = A + Y + C_{in}$	
S_1	S_0		$C_{in} = 0$	$C_{in} = 1$
0	0	$00 \dots 0$	$F = A$	$F = A + 1$
0	1	B	$F = A + B$	$F = A + B + 1$
1	0	\overline{B}	$F = A + \overline{B}$	$F = A + \overline{B} + 1$
1	1	$11 \dots 1$	$F = A - 1$	$F = A$

Rys. 5.10. Funkcje jednostki arytmetycznej z rysunku 5.9

Otrzymana na podstawie rysunków 5.9 i 5.10 tablica prawdy układu kombinacyjnego jednostki arytmetycznej jest pokazana na rysunku 5.11 ([2]). Na podstawie tablicy z rysunku 5.11 otrzymujemy następującą funkcję logiczną

$$Y_i(S_1, S_0, B_i) = \sum(3, 4, 6, 7)$$

która po minimalizacji



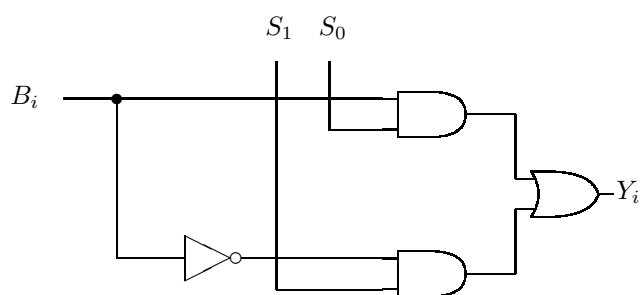
przyjmuje następującą postać:

$$Y_i(S_1, S_0, B_i) = S_0 B_i + S_1 \overline{B_i}$$

Funkcja ta jest realizowana przez układ, którego schemat logiczny jest pokazany na rysunku 5.12 ([2]).

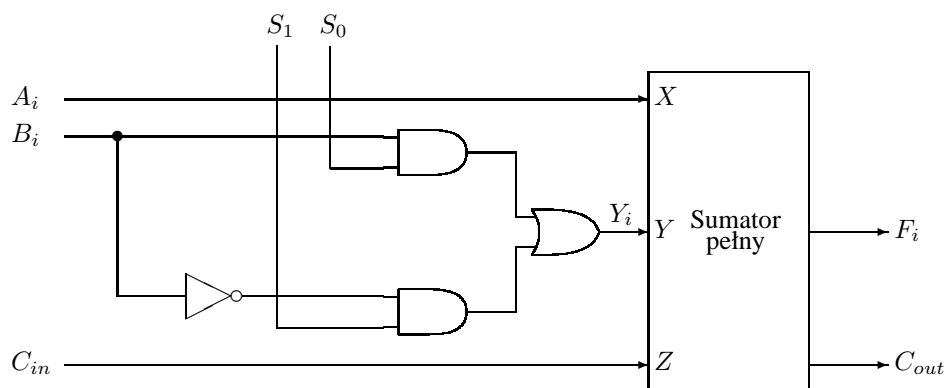
Wejścia			Wyjście	
S_1	S_0	B_i	Y_i	
0	0	0	0	$Y_i = 0$
0	0	1	0	
0	1	0	0	$Y_i = B_i$
0	1	1	1	
1	0	0	1	$Y_i = \overline{B_i}$
1	0	1	0	
1	1	0	1	$Y_i = 1$
1	1	1	1	

Rys. 5.11. Tablica prawdy układu kombinacyjnego projektowanej jednostki arytmetycznej ($i = 0, 1, \dots, n - 1$)



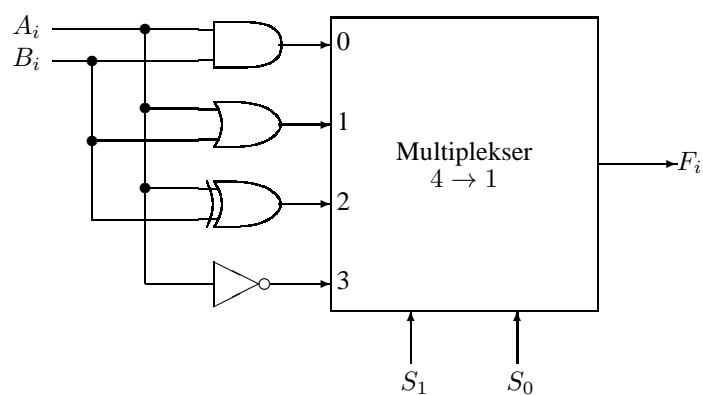
Rys. 5.12. Schemat logiczny układu realizującego funkcję $Y_i(S_1, S_0, B_i) = S_0 B_i + S_1 \overline{B_i}$

Na rysunku 5.13 jest pokazany schemat logiczny 1-bitowej jednostki arytmetycznej, realizującej mikrooperacje z rysunku 5.10 ([2]).



Rys. 5.13. Schemat logiczny 1-bitowej jednostki arytmetycznej realizującej funkcje z rysunku 5.10

Na rysunku 5.14 jest pokazany schemat logiczny 1-bitowej jednostki logicznej ([2]), realizującej funkcje logiczne z rysunku 5.15.



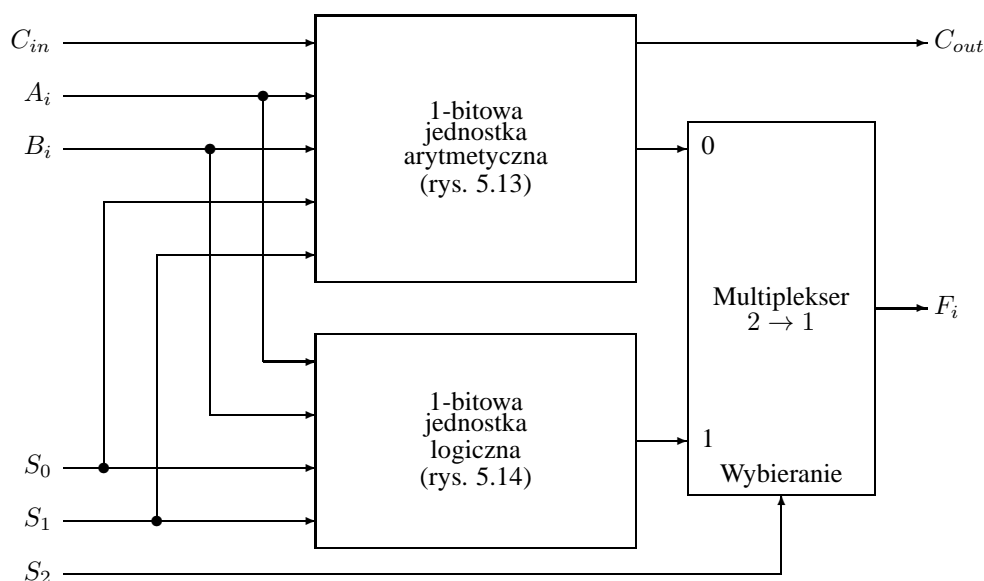
Rys. 5.14 Schemat logiczny 1-bitowej jednostki logicznej

S_1	S_0	F_i	Opis funkcji
0	0	$F_i = A_i \wedge B_i$	AND
0	1	$F_i = A_i \vee B_i$	OR
1	0	$F_i = A_i \oplus B_i$	XOR
1	1	$F_i = \overline{A_i}$	NOT

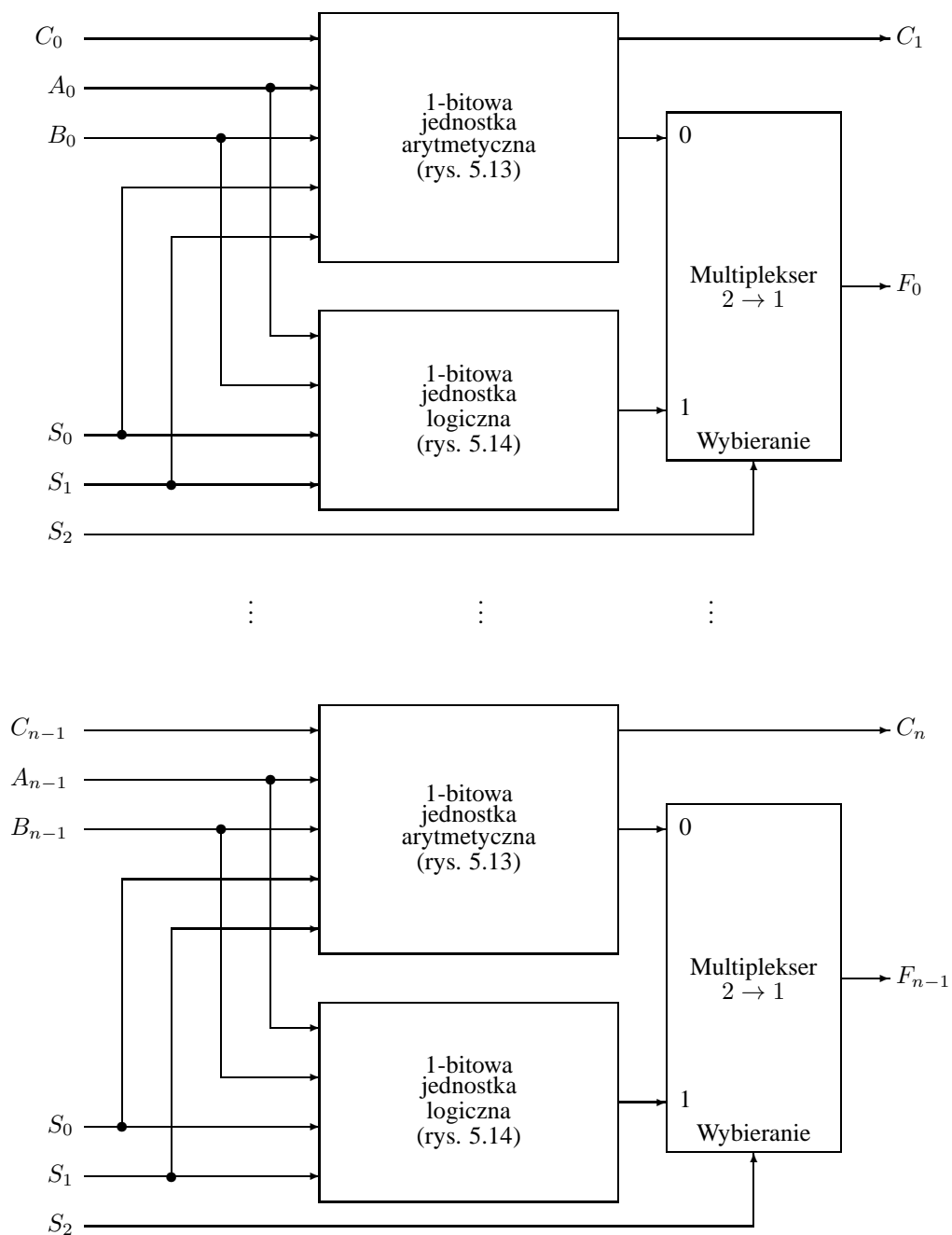
Rys. 5.15 Funkcje logiczne wykonywane przez 1-bitową jednostkę logiczną z rysunku 5.14.

Łącząc 1-bitową jednostkę arytmetyczną (rys. 5.13) z 1-bitową jednostką logiczną (rys. 5.14) otrzymujemy 1-bitową jednostkę arytmetyczno-logiczną, której schemat blokowy jest pokazany na rysunku 5.16 ([2, 3]). Kiedy $S_2 = 0$, wtedy wybierane jest wyjście arytmetyczne. Kiedy $S_2 = 1$, wtedy wybierane jest wyjście logiczne.

Schemat blokowy n -bitowej jednostki arytmetyczno-logicznej jest pokazany na rysunku 5.17.



Rys. 5.16. Schemat blokowy 1-bitowej jednostki arytmetyczno-logicznej



Rys. 5.17. Schemat blokowy n -bitowej jednostki arytmetyczno-logicznej

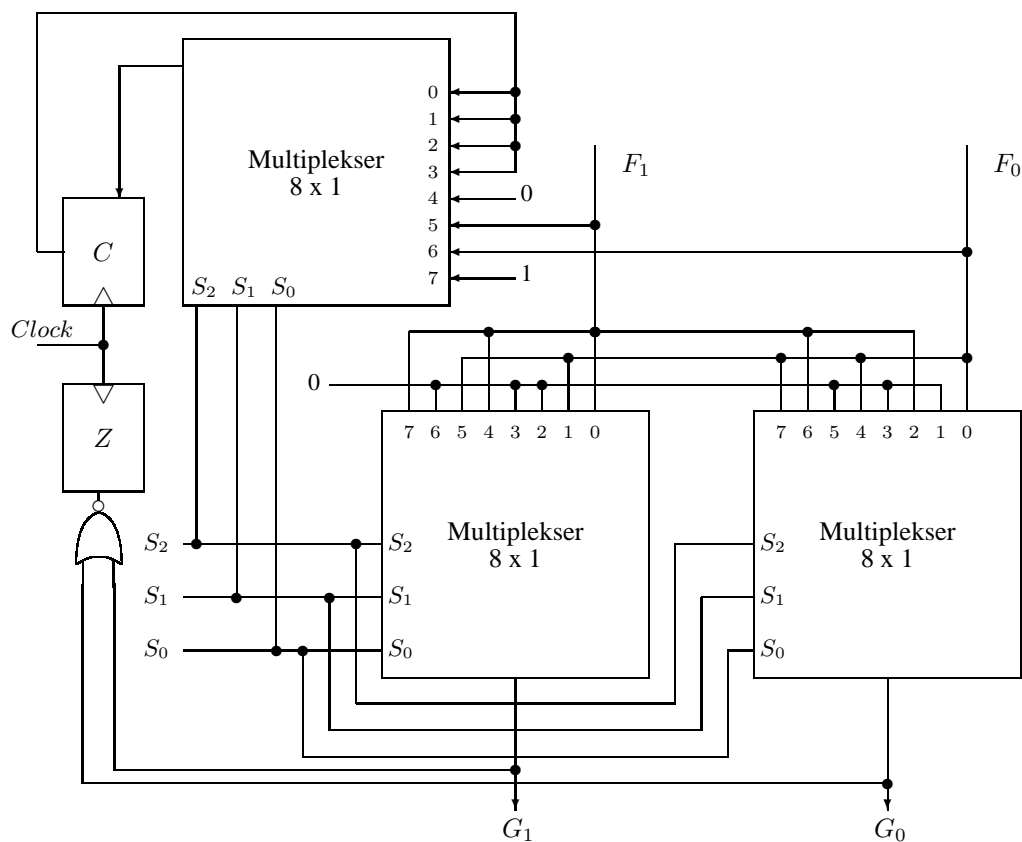
Na rysunku 5.18 są pokazane funkcje arytmetyczne i logiczne, które mogą być wykonywane przez zaprojektowaną n -bitową jednostkę arytmetyczno-logiczną.

S_2	S_1	S_0	C_0	Funkcja	Symboliczny zapis funkcji
0	0	0	0	$F = A$	TSF (<i>transfer A to F</i>)
0	0	0	1	$F = A + 1$	INC (<i>increment A by 1</i>)
0	0	1	0	$F = A + B$	ADD (<i>add A + B</i>)
0	0	1	1	$F = A + B + 1$	ADC (<i>add A + B + 1</i>)
0	1	0	0	$F = A + \overline{B}$	ACB (<i>add A + \overline{B}</i>)
0	1	0	1	$F = A + \overline{B} + 1$	SUB (<i>subtract A - B</i>)
0	1	1	0	$F = A - 1$	DEC (<i>decrement A by 1</i>)
0	1	1	1	$F = A$	TSF (<i>transfer A to F</i>)
1	0	0		$F = A \wedge B$	AND (<i>A AND B</i>)
1	0	1		$F = A \vee B$	OR (<i>A OR B</i>)
1	1	0		$F = A \oplus B$	XOR (<i>A XOR B</i>)
1	1	1		$F = \overline{A}$	NOT (<i>NOT A</i>)

Rys. 5.18. Funkcje arytmetyczne i logiczne wykonywane przez zaprojektowaną n -bitową jednostkę arytmetyczno-logiczną

5.8 Układ przesuwania

Wejściem układu przesuwania (*shift unit*) jest wyjście jednostki arytmetyczno-logicznej. Wyjściem układu przesuwania jest szyna wyjściowa. Układ przesuwania może przesłać wartość jednostki arytmetyczno-logicznej na szynę wyjściową bez zmian lub przesuwając ją w lewo lub w prawo. Na rysunku 5.19 jest pokazany przykładowy 2-bitowy układ przesuwania, realizujący funkcje przedstawione na rysunku 5.20. Przesuwanie jest wykonywane o jedną pozycję. Podczas przesuwania w lewo najmniej znacząca pozycja jest zerowana. Podczas przesuwania w prawo najbardziej znacząca pozycja jest zerowana. Przerzutnik Z jest wskaźnikiem zera. Przerzutnik Z jest ustawiany w stan jedynki logicznej wtedy, kiedy na wyjściu układu przesuwania są same zera, w przeciwnym przypadku jest ustawiany w stan zera logicznego. Przerzutnik C jest wskaźnikiem przeniesienia.



Rys. 5.19. Układ przesuwania realizujący funkcje z rysunku 5.20

S_2	S_1	S_0	Funkcja	Symboliczny zapis funkcji
0	0	0	$G \leftarrow F$	NSH (no shift)
0	0	1	$G \leftarrow shl F$	SHL (shift left F into G)
0	1	0	$G \leftarrow shr F$	SHR (shift right F into G)
0	1	1	$G \leftarrow 0$	ZERO (zero G)
1	0	0	$G \leftarrow F, C \leftarrow 0$	NSHZC (no shift, zero carry)
1	0	1	$G \leftarrow shlc F$	SHLC (shift left F into G with carry)
1	1	0	$G \leftarrow shrc F$	SHRC (shift right F into G with carry)
1	1	1	$G \leftarrow F, C \leftarrow 1$	NSHSC (no shift, set carry)

Rys. 5.20. Tablica funkcji układu przesuwania z rysunku 5.19

5.9 Przykładowy procesor

Na rysunku 5.21 ([2]) jest pokazany schemat blokowy przykładowego procesora, składającego się z jednostki arytmetyczno-logicznej, układu przesuwania, rejestru stanu, czterech rejestrów, dwóch multiplexerów wybierających rejestry źródłowe i dane wejściowe oraz dekodera wybierającego rejestr docelowy ([2]). W procesorze z rysunku 5.21 rejestr stanu zawiera następujące bity stanu wskazujące wyniki poprzedniej operacji:

C – wskaźnik przeniesienia (*carry*),

V – wskaźnik nadmiaru (*overflow*),

Z – wskaźnik zera (*zero*),

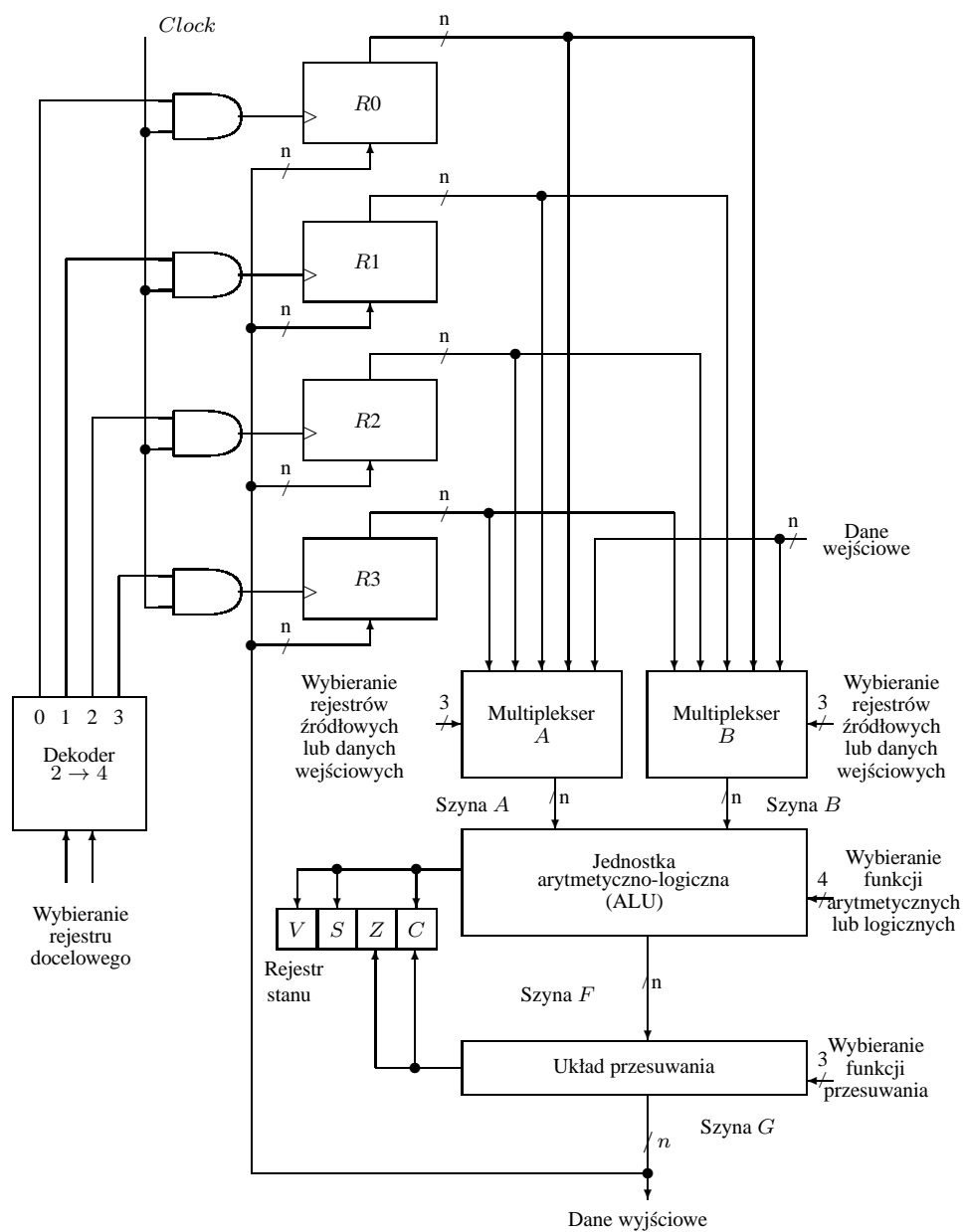
S – wskaźnik znaku (*sign*).

Jednostka sterująca (układ sterowania) steruje przepływem danych w procesorze i wybiera funkcje wykonywane przez jednostkę arytmetyczno-logiczną i układ przesuwania. Na przykład w celu wykonania mikrooperacji

$$R3 \leftarrow R1 + R2$$

układ sterowania powinien dostarczyć sygnały sterujące do następujących wejść:

1. Do wejść wybierania multiplexera A sygnał przesłania zawartości rejestru $R1$ na szynę A .
2. Do wejść wybierania multiplexera B sygnał przesłania zawartości rejestru $R2$ na szynę B .
3. Do wejść wybierania funkcji jednostki arytmetyczno-logicznej sygnał wybrania funkcji dodawania $A + B$.
4. Do wejść wybierania funkcji układu przesuwania sygnał wybrania funkcji bezpośredniego przesłania (tzn. bez przesuwania) zawartości jednostki arytmetyczno-logicznej na szynę wyjściową.
5. Do wejść wybierania dekodera sygnał wybrania rejestru docelowego $R3$ i przesłania do niego zawartości szyny wyjściowej.



Rys. 5.21. Schemat blokowy przykładowego procesora

5.10 Podsumowanie

W rozdziale 5 zostało wprowadzone pojęcie mikrooperacji. Podane zostały przykładowe mikrooperacje: przesyłania między rejestrami, arytmetyczne, logiczne i przesuwania. Została zaprojektowana prosta, przykładowa jednostka arytmetyczno-logiczna i jednostka przesuwająca. Został zaprojektowany prosty, przykładowy procesor.

Literatura

- [1] Mano M.M.: *Architektura komputerów*, WNT, 1980.
- [2] Mano M.M.: *Computer engineering: hardware design*, Prentice-Hall, 1988.
- [3] Mano M.M.: *Computer system architecture*, Prentice-Hall, 1993.

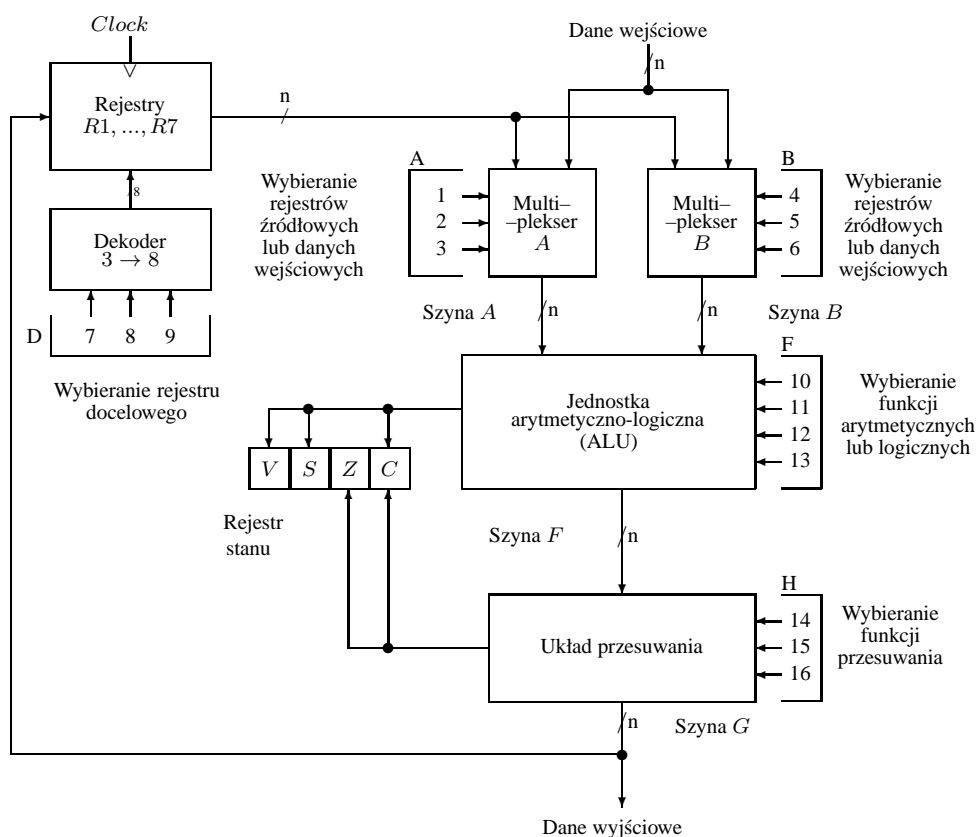
6 JEDNOSTKA STERUJĄCA

6.1 Słowo sterujące

Projektowanie logiczne jednostki centralnej CPU (rys. 5.1) można podzielić na dwie części: projektowanie procesora i projektowania jednostki sterującej ([1]). Istnieją dwa typy organizacji jednostek sterujących: sterowanie sprzętowe (*hardwired control*) i sterowanie mikroprogramowane (*microprogrammed control*). W rozdziale tym przedstawimy podstawowe wiadomości na temat obu typów organizacji sterowania ([1, 2]).

Rozbudujemy procesor z rysunku 5.19 o 3 rejestry i narysujemy jego uproszczony schemat blokowy w sposób pokazany na rysunku 6.1 ([1]). Procesor ten ma 7 rejestrów ($R1, \dots, R7$), których wyjścia są połączone z wejściami dwóch multiplekserów (A, B). Linie wybierania multipleksa A (1, 2, 3) umożliwiają wybieranie jednego z rejestrów lub danych wejściowych na szynie A . Linie wybierania multipleksa B (4, 5, 6) umożliwiają wybieranie jednego z rejestrów lub danych wejściowych na szynie B . Szyny A i B są połączone z wejściami jednostki arytmetyczno-logicznej (ALU). Linie wybierania ALU (10, 11, 12, 13) umożliwiają wybieranie funkcji arytmetycznych lub logicznych. Wyjście ALU (szyna F) jest połączone z wejściem układu przesuwania. Linie wybierania układu przesuwania (14, 15, 16) umożliwiają wybieranie funkcji przesuwania. Wyjście układu przesuwania (szyna G) jest połączone z szyną wyjściową. Linie wybierania dekodera (7, 8, 9) umożliwiają wybieranie rejestru docelowego otrzymującego dane z szyny wyjściowej.

Sygnały sterujące wybierające rejestry źródłowe lub dane wejściowe, funkcje arytmetyczne lub logiczne i rejestry docelowe są zmiennymi dwójkowymi (funkcjami sterującymi) generowanymi przez jednostkę sterującą. Zmienna sterująca jest aktywna, kiedy ma wartość logiczną 1. Sygnały sterujące procesorem z rysunku 6.1 tworzą słowo sterujące (*control word*), pokazane na rysunku 6.2 ([1]). W tym przypadku słowo sterujące składa się z pięciu pól. Bity pola A wybierają rejestr źródłowy lub dane wejściowe na szynie A . Bity pola B wybierają rejestr źródłowy lub dane wejściowe na szynie B . Bity pola D wybierają rejestr docelowy. Bity pola F wybierają funkcję arytmetyczną lub logiczną. Bity pola H wybierają funkcję przesuwania.



Rys. 6.1. Schemat blokowy przykładowego procesora

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A			B			D			F				H		

Rys. 6.2. Słowo sterujące

Bity słowa sterującego (rys. 6.2) podane na wejścia procesora oznaczone przez 1, 2, ..., 16 (rys. 6.1) wyznaczają mikrooperację. Na przykład mikrooperacja

$$R3 \leftarrow R1 + R2$$

określa $R1$ jako rejestr źródłowy na szynie A, $R2$ jako rejestr źródłowy na szynie B i $R3$ jako rejestr docelowy. Określa również wykonanie dodawania $F = A + B$ (rys. 5.18) przez ALU i przesyłanie zawartości ALU na szynę wyjściową bez zmian $G \leftarrow F$ (rys. 5.20) dla układu przesuwania. Przyjmując, że kody rejestrów $R1$, $R2$ i $R3$ są równe odpowiednio

001, 010 i 011, na podstawie rysunków 5.18, 5.20 i 6.2, otrzymujemy słowo sterujące pokazane na rysunku 6.3.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	1	0	1	0	0	1	1	0	0	1	0	0	0	0
A			B			D			F				H		

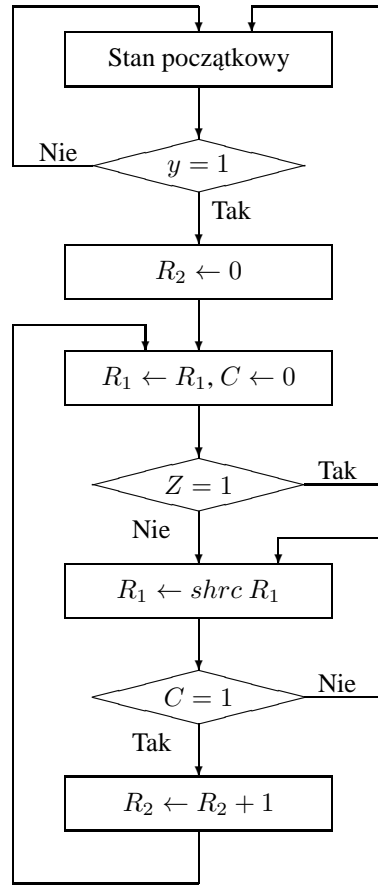
Rys. 6.3. Słowo sterujące dla mikrooperacji $R3 \leftarrow R1 + R2$

6.2 Sterowanie sprzętowe

Projektowanie sprzętowej jednostki sterującej przedstawimy na przykładzie. Projektowana przez nas sprzętowa jednostka sterująca będzie sterować procesorem z rysunku 6.1.

Rozważmy problem zliczania jedynek liczby binarnej przechowywanej w rejestrze procesora z rysunku 6.1 ([1]). Załóżmy, że liczba binarna jest przechowywana w rejestrze $R1$, a wynik w rejestrze $R2$ (na przykład jeżeli $R1 = 01010001$, to wynik w $R2 = 3$). Algorytm zliczania jedynek jest pokazany na rysunku 6.4 ([1]). Proces zliczania jedynek rozpoczyna się wtedy, kiedy zmienna sterująca $y=1$. Początkowo rejestr $R2$ jest zerowany ($R2 \leftarrow 0$). Zawartość rejestru $R1$ jest przesyłana (przez ALU i układ przesuwania procesora z rysunku 6.1) do rejestru $R1$ ($R1 \leftarrow R1$) w celu uaktualnienia bitu stanu Z (zero) i wyzerowania bitu przeniesienia C (rysunki 5.18 i 5.20). Jeżeli $Z = 1$, przechodzimy do stanu początkowego (w tym przypadku $R1 = 0$). Jeżeli $Z = 0$ (w tym przypadku $R1 \neq 0$), zawartość rejestru $R1$ przesuwamy w prawo z przeniesieniem ($R1 \leftarrow shr C R1$) dopóty, dopóki bit przeniesienia $C = 0$. Jeżeli bit przeniesienia $C = 1$, zawartość rejestru $R2$ jest zwiększana o 1 ($R2 \leftarrow R2 + 1$), a następnie zawartość rejestru $R1$ jest przesyłana do rejestru $R1$ ($R1 \leftarrow R1$).

Na podstawie rysunku 6.4 otrzymujemy graf stanów pokazany na rysunku 6.5 ([1]). Dla każdego stanu określamy jakie mikrooperacje muszą być wykonane (rys. 6.5).



Rys. 6.4. Algorytm zliczania jedynek

Do realizacji układu sekwencyjnego, którego graf stanów jest pokazany na rysunku 6.5, potrzebne będą 3 przerzutniki (5 stanów). Wybierzemy przerzutniki D i oznaczmy je przez D_0 , D_1 i D_2 . Tablica stanów projektowanego układu sekwencyjnego jest pokazana na rysunku 6.6. Projektując układ sekwencyjny wykorzystamy dekodery, którego wyjściom T_0 , T_1 , T_2 , T_3 i T_4 przypiszemy odpowiednio stany aktualne 000, 001, 010, 011, 100 (rys. 6.6).

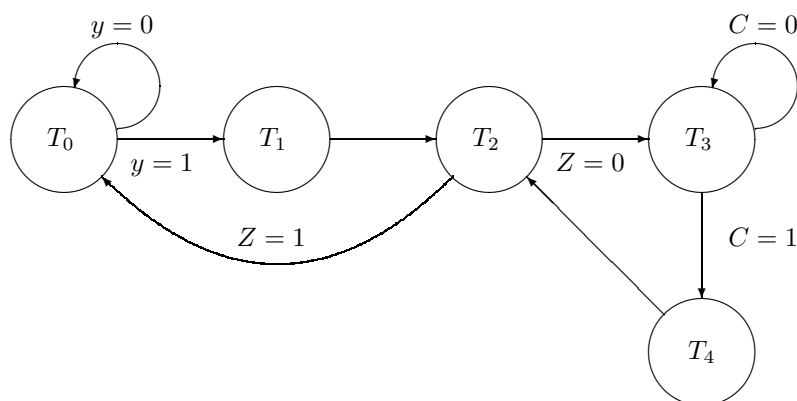
Na podstawie stanu następnego w tablicy stanów z rysunku 6.6 możemy napisać następujące równania wejść dla przerzutników:

$$D_2 = T_3 C$$

$$D_1 = T_1 + T_2 \overline{Z} + T_3 \overline{C} + T_4$$

$$D_0 = T_0 y + T_2 \overline{Z} + T_3 \overline{C}$$

Na podstawie równań wejść przerzutników otrzymujemy schemat logiczny projektowanego układu pokazany na rysunku 6.7.

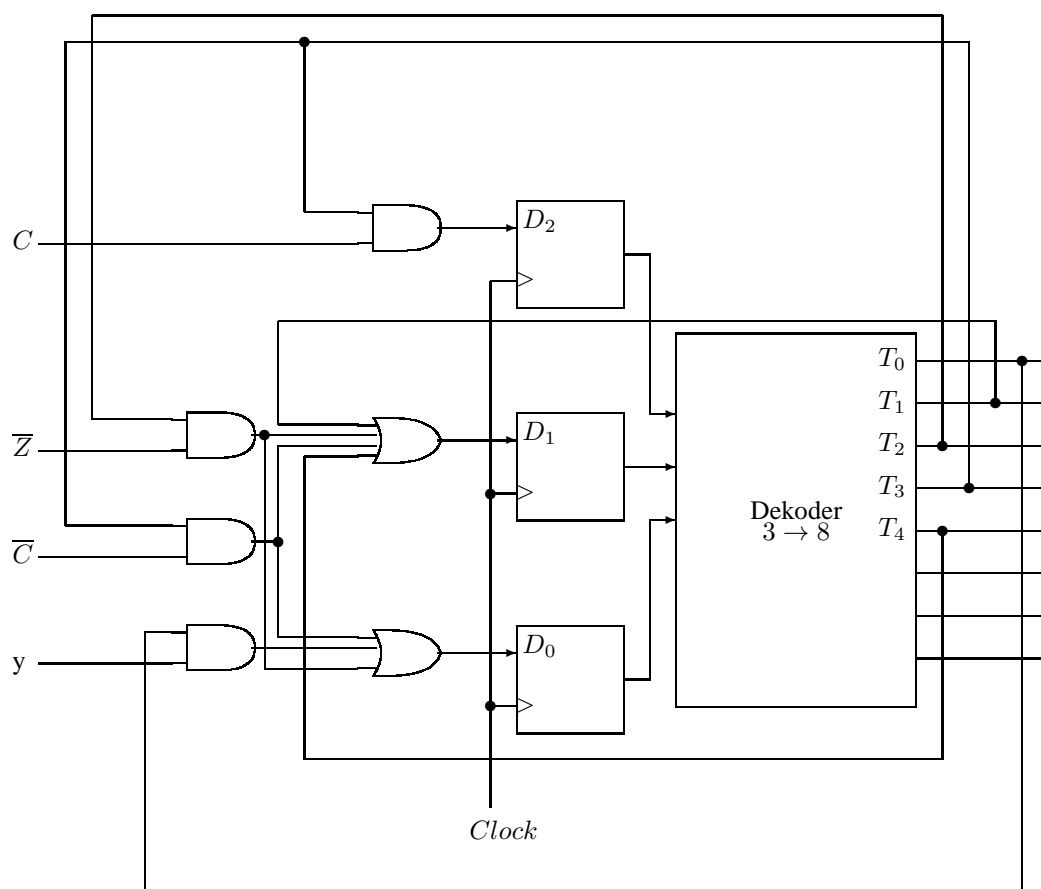


T_0 :
 T_1 : $R_2 \leftarrow 0$
 T_2 : $R_1 \leftarrow R_1, C \leftarrow 0$
 T_3 : $R_1 \leftarrow shr R_1$
 T_4 : $R_2 \leftarrow R_2 + 1$

Rys. 6.5. Graf stanów i lista mikrooperacji

Stan aktualny			Wejścia			Stan następny			Wyjścia dekodera				
D_2	D_1	D_0	y	Z	C	D_2	D_1	D_0	T_0	T_1	T_2	T_3	T_4
0	0	0	0	x	x	0	0	0	1	0	0	0	0
0	0	0	1	x	x	0	0	1	1	0	0	0	0
0	0	1	x	x	x	0	1	0	0	1	0	0	0
0	1	0	x	0	x	0	1	1	0	0	1	0	0
0	1	0	x	1	x	0	0	0	0	0	1	0	0
0	1	1	x	x	0	0	1	1	0	0	0	1	0
0	1	1	x	x	1	1	0	0	0	0	0	1	0
1	0	0	x	x	x	0	1	0	0	0	0	0	1

Rys. 6.6. Tablica stanów grafu stanów z rysunku 6.5



Rys. 6.7. Schemat logiczny zaprojektowanego układu sekwencyjnego

W rozwiązaniu tym wykorzystaliśmy dekodery, którego wyjścia T_0 , T_1 , T_2 , T_3 i T_4 (rys. 6.7) reprezentowały odpowiednio stany T_0 , T_1 , T_2 , T_3 i T_4 grafu stanów z rysunku 6.5. Inne rozwiązanie można uzyskać zakładając, że każdy stan grafu stanów z rysunku 6.5 będzie reprezentowany przez jeden przerzutnik. W tym przypadku konieczne jest użycie pięciu przerzutników (5 stanów grafu stanów z rysunku 6.5). Wybierzmy przerzutniki D i oznaczmy ich wyjścia jako T_0 , T_1 , T_2 , T_3 i T_4 . Na podstawie grafu stanów z rysunku 6.5. otrzymujemy następujące równania wejść przerzutników:

$$D_0 = T_0\bar{y} + T_2Z$$

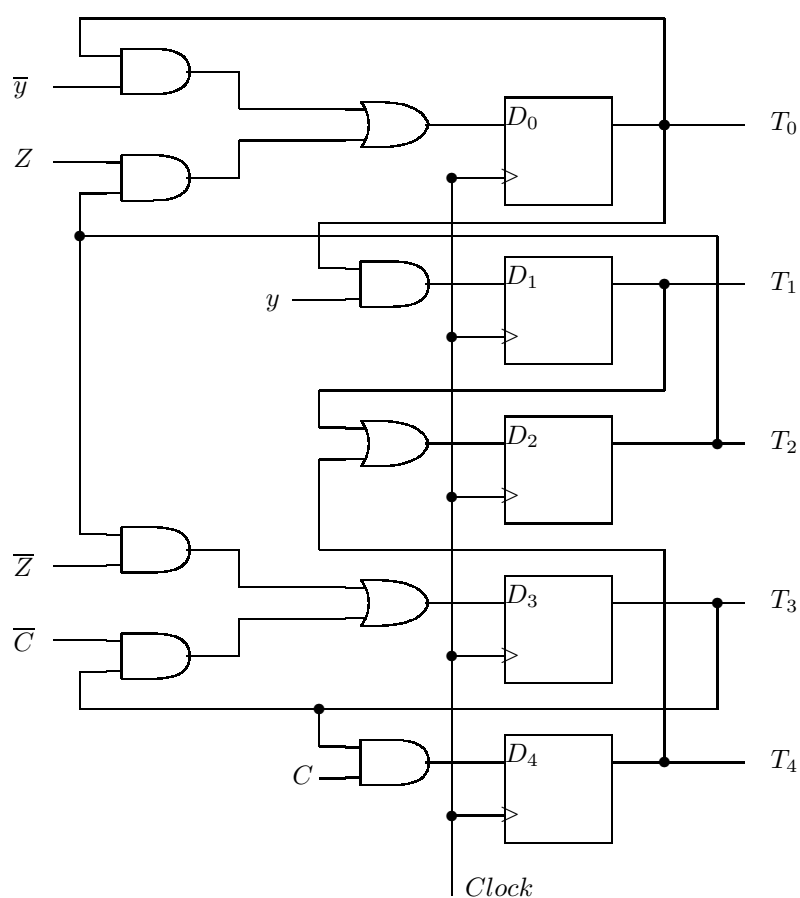
$$D_1 = T_0y$$

$$D_2 = T_1 + T_4$$

$$D_3 = T_2\bar{Z} + T_3\bar{C}$$

$$D_4 = T_3C$$

Na podstawie równań wejść przerzutników otrzymujemy schemat logiczny projektowanego układu sekwencyjnego pokazany na rysunku 6.8.



Rys. 6.8. Schemat logiczny zaprojektowanego układu sekwencyjnego

Założyliśmy, że algorytm zliczania jedynek ma być realizowany za pomocą procesora z rysunku 6.1. Procesor ten jest sterowany słowem sterującym o długości szesnastu bitów (rys. 6.2). Zatem projektowany układ sterujący musi generować 16 sygnałów sterujących.

Dla mikrooperacji $R2 \leftarrow 0$ otrzymujemy (na podstawie rysunków 5.18, 5.20 i 6.2) słowo sterujące pokazane na rysunku 6.9

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
A			B			D			F				H		

Rys. 6.9. Słowo sterujące dla mikrooperacji $R2 \leftarrow 0$

Dla mikrooperacji $R1 \leftarrow R1, C \leftarrow 0$ słowo sterujące jest pokazane na rysunku 6.10.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0
A			B			D			F				H		

Rys. 6.10. Słowo sterujące dla mikrooperacji $R1 \leftarrow R1, C \leftarrow 0$

Dla mikrooperacji $R1 \leftarrow shrc R1$ otrzymujemy słowo sterujące pokazane na rysunku 6.11.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	1	0	0	0	0	0	1	0	0	0	0	1	1	0
A			B			D			F				H		

Rys. 6.11. Słowo sterujące dla mikrooperacji $R1 \leftarrow shrc R1$

Dla mikrooperacji $R2 \leftarrow R2 + 1$ otrzymujemy słowo sterujące pokazane na rysunku 6.12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0
A			B			D			F				H		

Rys. 6.12. Słowo sterujące dla mikrooperacji $R2 \leftarrow R2 + 1$

W dalszym ciągu zaprojektujemy układ kombinacyjny, którego tablica prawdy jest pokazana na rysunku 6.13.

Wejścia (stany)					Wyjścia (słowo sterujące)															
T_0	T_1	T_2	T_3	T_4	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0

Rys. 6.13. Tablica prawdy projektowanego układu kombinacyjnego

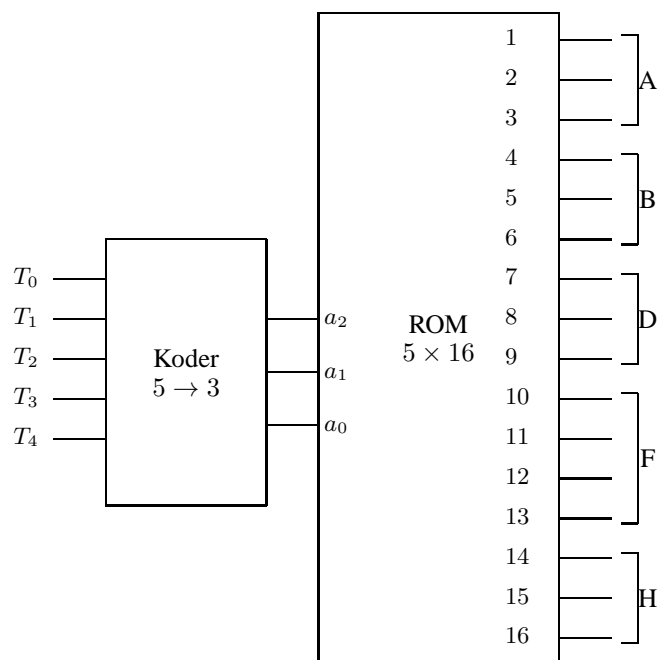
Układ ten umożliwi uzyskanie słowa sterującego (Wyjścia) realizującego mikrooperacje określone dla każdego stanu (Wejścia) grafu stanów z rysunku 6.5. Układ kombinacyjny, którego tablica prawdy jest pokazana na rysunku 6.13, zrealizujemy za pomocą pamięci ROM (5×16) zaprogramowanej w sposób pokazany na rysunku 6.14.

Schemat logiczny zaprojektowanego układu kombinacyjnego jest pokazany na rysunku 6.15.

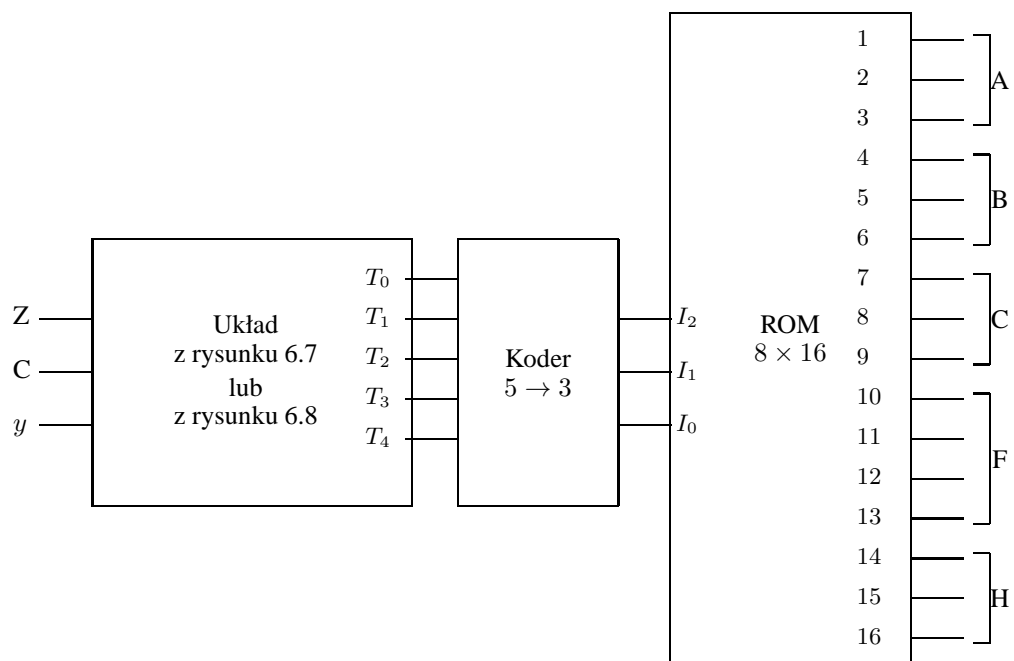
Na rysunku 6.16 jest pokazany schemat logiczny zaprojektowanego układu sterującego procesorem z rysunku 6.1 realizującym algorytm zliczania jedynek.

Adres			Wyjścia (słowo sterujące)																	
a_2	a_1	a_0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1		
0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0		
0	1	1	0	0	1	0	0	0	0	0	1	0	0	0	0	1	1	0		
1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0		

Rys. 6.14. Tablica prawdy pamięci ROM 5×16



Rys. 6.15. Schemat logiczny zaprojektowanego układu kombinacyjnego

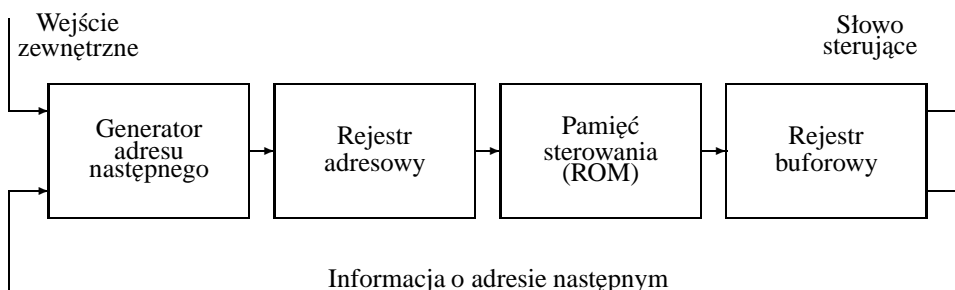


Rys. 6.16. Schemat logiczny zaprojektowanej jednostki sterującej

6.3 Sterowanie mikroprogramowane

Koncepcja sterowania mikroprogramowanego (*microprogrammed control*) polega na zastąpieniu sprzętowej jednostki sterującej przez program zapisany w pamięci stałej ROM, nazywanej pamięcią sterowania (*control memory*). Program zapisany w pamięci sterowania jest nazywany mikroprogramem (*microprogram*). Przygotowanie mikroprogramu jest nazywane mikroprogramowaniem (*microprogramming*). Każde słowo w pamięci sterowania jest nazywane mikrorozkazem (*microinstruction*). Mikrooperacja (*microoperation*) jest to elementarna operacja, jaka może być wykonywana podczas jednego impulsu zegarowego na danych przechowywanych w rejestrach.

Schemat organizacji sterowania mikroprogramowanego jest pokazany na rysunku 6.17 ([1, 2]). Rejestr adresowy podaje adres mikrorozkazu. Rejestr buforowy przechowuje mikrorozkaz pobrany z pamięci sterowania. Część mikrorozkazu zawiera słowo sterujące określające, jaka mikrooperacja ma być wykonana przez procesor (zakładamy, że mikroprogramowana jednostka sterująca steruje procesorem z rysunku 6.1). W czasie wykonywania mikrooperacji przez procesor mikroprogramowana jednostka sterująca musi wyznaczyć adres następnego mikrorozkazu. Adresem tym może być dowolny adres pamięci sterowania. W tym celu konieczne jest użycie części mikrorozkazu do generowania adresu następnego. Adres następnego mikrorozkazu może być także funkcją wejściowych warunków zewnętrznych. Podczas wykonywania mikrooperacji wyznaczany jest adres następnego mikrorozkazu i przesyłany do rejestru adresowego.

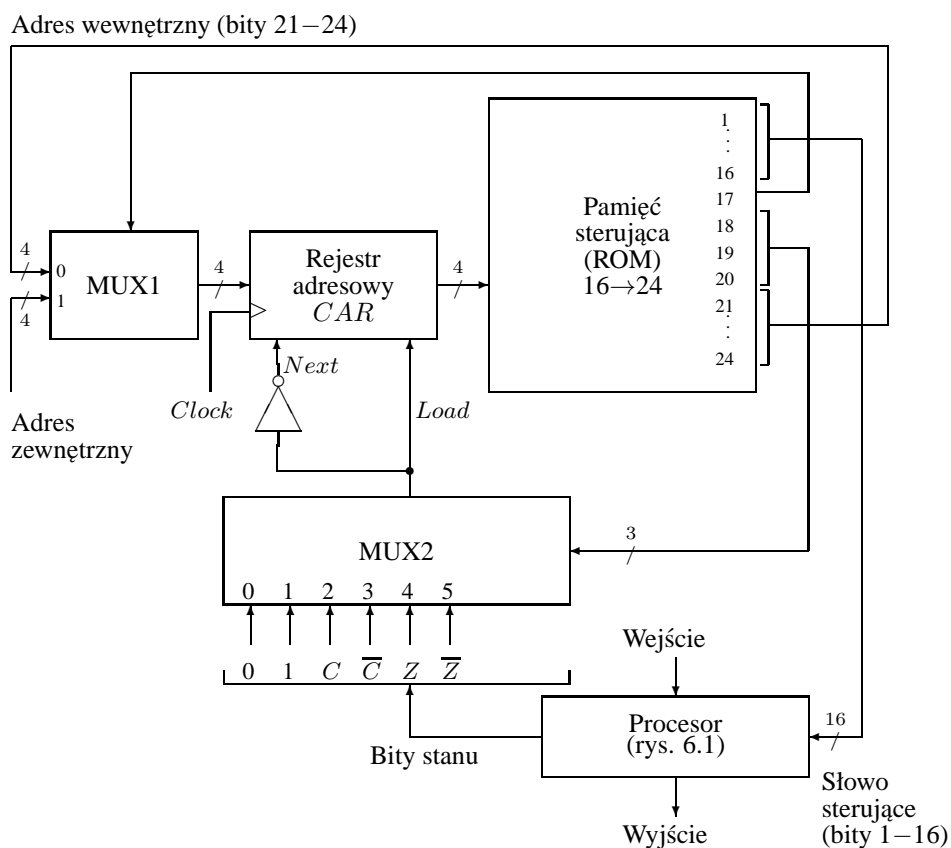


Rys. 6.17. Schemat organizacji sterowania mikroprogramowanego

Rozważmy mikroprogramowaną jednostkę sterującą procesorem pokazaną na rysunku 6.18 ([1]). Jednostka ta składa się z pamięci sterującej ROM, rejestru adresowego pamięci *CAR* i dwóch multiplexerów (*MUX1* i *MUX2*). W układzie pokazanym na rysunku 6.18 mikroinstrukcja składa się z szesnastu bitów słowa sterującego (bity 1–16), jednego bitu sterującego multiplexerem *MUX1* (bit 17), trzech bitów sterujących multiplexerem *MUX2* (bity 18–20) i czterech bitów adresujących pamięć sterującą (bity 21–24).

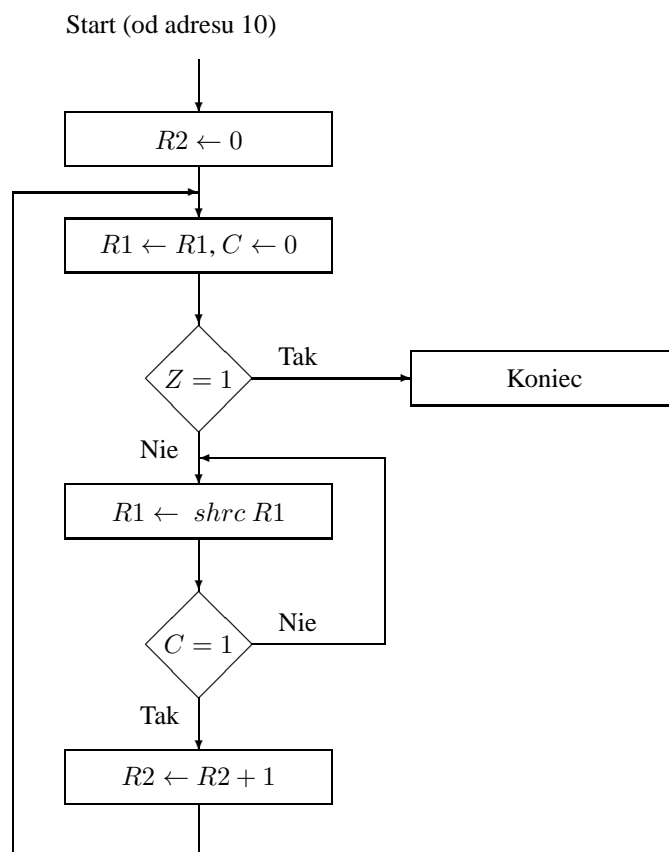
Działanie układu na rysunku 6.18 można opisać w następujący sposób. Po pojawieniu się impulsu zegara (*Clock*) do rejestru *CAR* jest ładowany nowy adres. Z pamięci sterowania jest pobierana mikroinstrukcja o adresie zawartym w *CAR*. Słowo sterujące

(rys. 6.2) mikroinstrukcji wyznacza mikrooperacje, które ma wykonać procesor (rysunki 5.18 i 5.20). Multiplexery MUX1 i MUX2 wyznaczają nowy adres dla *CAR*. Pojawienie się następnego zbocza narastającego impulsu zegarowego kończy wykonywanie aktualnych mikrooperacji przez procesor i powoduje umieszczenie w *CAR* nowego adresu, którym może być albo adres poprzedni zwiększony o 1 ($Next = 1$, $Load = 0$) albo adres otrzymany z multiplexera MUX1 ($Next = 0$, $Load = 1$). Adres otrzymany z multiplexera MUX1 może być albo adresem wewnętrznym zawartym w poprzedniej mikroinstrukcji (bity 21–24) albo adresem zewnętrznym. Kiedy bit sterujący multiplexserem MUX1 (bit 17) aktualnie wykonywanej mikroinstrukcji jest równy 0, wtedy adres następnej mikroinstrukcji będzie adresem wewnętrznym. Kiedy bit sterujący multiplexserem MUX1 aktualnie wykonywanej mikroinstrukcji jest równy 1, wtedy adres następnej mikroinstrukcji będzie adresem zewnętrznym. Kiedy bity sterujące multiplexserem MUX2 (bity 18–20) aktualnie wykonywanej mikroinstrukcji są odpowiednio równe 0, 0, 0, wtedy $Next = 1$ i $Load = 0$. Kiedy bity sterujące multiplexserem MUX2 (bity 18–20) aktualnie wykonywanej mikroinstrukcji są odpowiednio równe 0, 0, 1, wtedy $Next = 0$ i $Load = 1$.



Rys. 6.18. Mikroprogramowana jednostka sterująca procesorem

Napišemy teraz mikroprogram na podstawie pokazanego na rysunku 6.19 ([1]) algorytmu zliczania jedynek liczby binarnej przechowywanej w rejestrze procesora z rysunku 6.1.



Rys. 6.19. Algorytm zliczania jedynek

Liczba binarna jest przechowywana w rejestrze $R1$, a wynik w rejestrze $R2$. Załóżmy, że mikroprogram będzie umieszczony w pamięci sterowania począwszy od adresu 10. Początkowo rejestr $R2$ jest zerowany ($R2 \leftarrow 0$). Zawartość rejestru $R1$ jest przesyłana (przez ALU i układ przesuwania procesora z rysunku 6.1) do rejestru $R1$ ($R1 \leftarrow R1$) w celu uaktualnienia bitu stanu Z (zero) i wyzerowania bitu przeniesienia C . Jeżeli $Z = 1$, kończymy algorytm (w tym przypadku $R1 = 0$). Jeżeli $Z = 0$ (w tym przypadku $R1 \neq 0$), zawartość rejestru $R1$ przesuwamy w prawo z przeniesieniem ($R1 \leftarrow shr R1$) dopóty, dopóki $C = 0$. Jeżeli $C = 1$, zawartość rejestru $R2$ jest zwiększana o 1 ($R2 \leftarrow R2 + 1$), a następnie zawartość rejestru $R1$ jest przesyłana do rejestru $R1$.

Algorytm ten można również zapisać w sposób pokazany na rysunku 6.20 ([1]). Na rysunku 6.20 *EXT* oznacza adres zewnętrzny.

Adres	Mikrooperacje i skoki warunkowe
10	$R2 \leftarrow 0, CAR \leftarrow CAR + 1$
11	$R1 \leftarrow R1, C \leftarrow 0, CAR \leftarrow CAR + 1$
12	$if (Z = 1) then (CAR \leftarrow EXT) else (CAR \leftarrow CAR + 1)$
13	$R1 \leftarrow shrc R1, CAR \leftarrow CAR + 1$
14	$if (C = 1) then (CAR \leftarrow CAR + 1) else (CAR \leftarrow 13)$
15	$R2 \leftarrow R2 + 1, CAR \leftarrow 11$

Rys. 6.20. Algorytm zliczania jedynek

Na podstawie rysunku 6.20, funkcji procesora (rys. 5.18 i 5.20) i słowa sterującego (rys. 6.2) można napisać mikroprogram zliczania jedynek pokazany na rysunku 6.21 ([1]).

Adres pamięci sterującej	Słowo sterujące (bity 1 – 16)					MUX1 (bit 17)	MUX2 (bity 18 – 20)	Pole adresowe mikroinstrukcji (bity 21 – 24)
	A	B	D	F	H			
10	–	–	<i>R2</i>	TSF	ZERO	–	NEXT	–
11	<i>R1</i>	–	<i>R1</i>	TSF	NSHZC	–	NEXT	–
12	–	–	–	TSF	NSH	EXT	LZ	–
13	<i>R1</i>	–	<i>R1</i>	TSF	SHRC	–	NEXT	–
14	–	–	–	TSF	NSH	INT	LNC	13
15	<i>R2</i>	–	<i>R2</i>	INC	NSH	INT	LOAD	11

Rys. 6.21. Symboliczny mikroprogram zliczania jedynek

Na rysunku 6.21 *EXT* oznacza adres zewnętrzny (bit 17 = 1), a *INT* adres wewnętrzny (bit 17 = 0). *NEXT* oznacza taką kombinację bitów 18–20, dla których *Next* = 1 i *Load* = 0 (rys. 6.18). *LOAD* oznacza taką kombinację bitów, dla których *Next* = 0 i *Load* = 1. *LZ* (*Load if zero*) oznacza taką kombinację bitów, dla których *Next* = 0 i *Load* = 1 wtedy, kiedy *Z* = 1. *LNC* (*Load if not carry*) oznacza taką kombinację bitów, dla których *Next* = 0 i *Load* = 1 wtedy, kiedy *C* = 0. Pola oznaczone na rysunku 6.21 za pomocą kreski – nie są istotne. Oznacza to, że pisząc mikroprogram w postaci binarnej możemy pola te zaprogramować w dowolny sposób, na przykład możemy wypełnić je zerami.

Na podstawie rysunków 6.21, 6.18, 6.2, 5.20 i 5.18 możemy napisać mikroprogram zliczania jedynek w postaci binarnej pokazanej na rysunku 6.22 ([1]).

Adres ROM	Mikroprogram							
1010	000	000	010	0000	011	0	000	0000
1011	001	000	001	0000	100	0	000	0000
1100	000	000	000	0000	000	1	100	0000
1101	001	000	001	0000	110	0	000	0000
1110	000	000	000	0000	000	0	011	1101
1111	010	000	010	0001	000	0	001	1011

Rys. 6.22. Mikroprogram zliczania jedynek

6.4 Podsumowanie

W rozdziale 6 zostały podane podstawowe wiadomości na temat sterowania sprzętowego i sterowania mikroprogramowanego. Jednostka sterująca procesorem może być zrealizowana jako sprzętowa lub mikroprogramowana. Układy mikroprogramowane są omówione, na przykład w [3, 4].

Literatura

- [1] Mano M.M.: *Computer engineering: hardware design*, Prentice-Hall, 1988.
- [2] Mano M.M.: *Computer system architecture*, Prentice-Hall, 1993.
- [3] Kalisz J.: *Podstawy elektroniki cyfrowej*, WKŁ, 1993.
- [4] Traczyk W.: *Układy cyfrowe. Podstawy teoretyczne i metody syntezy*, WNT, 1986.

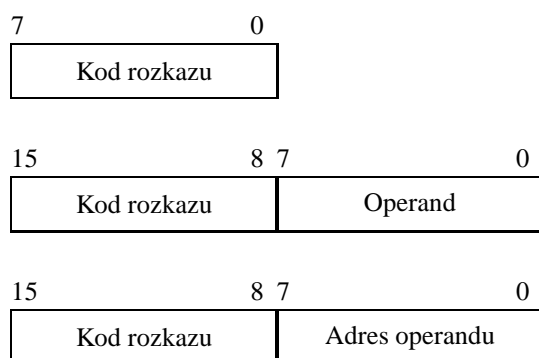
7 PRZYKŁADOWY KOMPUTER

7.1 Wstęp

W rozdziale tym zaprojektujemy prosty, przykładowy komputer ([1]). Projektowanie rozpoczniemy od przyjęcia formatów rozkazów i listy rozkazów przykładowego komputera. Następnie wyszczególnimy wszystkie mikrooperacje projektowanego komputera, na podstawie których zaprojektujemy sprzętową jednostkę sterującą.

7.2 Formaty rozkazów

Przyjmijmy dla projektowanego komputera formaty rozkazów pokazane na rysunku 7.1 ([1]).



Rys. 7.1. Formaty rozkazów przykładowego komputera

7.3 Lista rozkazów

Przyjmijmy dla projektowanego komputera listę rozkazów pokazaną na rysunku 7.2 ([1]). Na rysunku tym zostały przyjęte następujące oznaczenia:

- AC* – akumulator (*accumulator*),
- OPRD* – operand (*operand*),
- ADRS* – adres pamięci (*address*),
- M* – pamięć (*memory*),
- INR* – rejestr wejściowy (*input register*),
- OUR* – rejestr wyjściowy (*output register*).

Akumulator jest jednym z rejestrów procesora.

Kod binarny	Kod symboliczny	Opis
0000 0001	INA (<i>increment AC</i>)	$AC \leftarrow AC + 1$
0000 0010	CMA (<i>complement AC</i>)	$AC \leftarrow \overline{AC}$
0000 0011	LDI OPRD (<i>load immediate operand</i>)	$AC \leftarrow OPRD$
0000 0100	ADI OPRD (<i>add immediate operand</i>)	$AC \leftarrow AC + OPRD$
0000 0101	LDA ADRS (<i>load to AC</i>)	$AC \leftarrow M[ADRS]$
0000 0110	STA ADRS (<i>store from AC</i>)	$M[ADRS] \leftarrow AC$
0000 0111	INP (<i>input</i>)	$AC \leftarrow INR$
0000 1000	OUT (<i>output</i>)	$OUR \leftarrow AC$

Rys. 7.2. Lista rozkazów przykładowego komputera

7.4 Wykonywanie rozkazów

Za pomocą przyjętych rozkazów (rys. 7.2) napiszemy przykładowy program obliczający wartość wyrażenia arytmetycznego $43 - (36 + 7)$. Program ten można napisać w sposób pokazany na rysunku 7.3.

```

LDI 36       $AC \leftarrow 36$ 
ADI 7        $AC \leftarrow AC + 7$ 
CMA          $AC \leftarrow \overline{AC}$ 
INA          $AC \leftarrow AC + 1$ 
ADI 43       $AC \leftarrow AC + 43$ 
STA 100      $M[100] \leftarrow AC$ 

```

Rys. 7.3. Program obliczający wartość wyrażenia $43 - (36 + 7)$

Na rysunku 7.4 jest pokazana zawartość pamięci po wykonaniu programu z rysunku 7.3.

Adres dziesiętny	Zawartość pamięci		
0			
	⋮		
5	0000 0011	Kod operacji=3	
6	0010 0100	Operand=36	$AC=0010\ 0100$
7	0000 0100	Kod operacji=4	
8	0000 0111	Operand=7	$AC=0010\ 1011$
9	0000 0010	Kod operacji=2	$AC=1101\ 0100$
10	0000 0001	Kod operacji=1	$AC=1101\ 0101$
11	0000 0100	Kod operacji=4	
12	0010 1011	Operand=43	$AC=0000\ 0000$
13	0000 0110	Kod operacji=6	
14	0110 0100	Adres=100	
	⋮		
100	0000 0000		$M[100]=0000\ 0000$

Rys. 7.4. Przykładowa zawartość pamięci

Adres rozkazu, który ma być wykonany, jest zawarty w rejestrze PC (*program counter*). W celu wykonania rozkazu kod rozkazu (*operation code*) jest pobierany z pamięci do rejestru DR (*data register*). Następnie kod rozkazu jest przesyłany z rejestru DR do rejestru IR (*instruction register*). Równocześnie licznik rozkazów PC jest zwiększany o 1. Jest to faza pobrania rozkazu (*instruction fetch phase*), którą można zapisać w następujący sposób ([1]):

$$T_0 : DR \leftarrow M[PC]$$

$$T_1 : IR \leftarrow DR, PC \leftarrow PC + 1$$

gdzie T_0 i T_1 są sygnałami taktującymi. Kod rozkazu zawarty w rejestrze IR jest dekodowany przez dekodery rozkazów. W projektowanym komputerze przyjmujemy, że dekodery rozkazów

ma 8 wyjść: D_1, \dots, D_8 , odpowiadających ośmiu rozkazom z rysunku 7.2. Kombinacja na wyjściu dekodera rozkazów:

$$D_1 = 1, D_2 = D_3 = D_4 = D_5 = D_6 = D_7 = D_8 = 0$$

oznacza, że został zdekodowany rozkaz INA (rys. 7.2). Kombinacja na wyjściu dekodera rozkazów:

$$D_1 = 0, D_2 = 1, D_3 = D_4 = D_5 = D_6 = D_7 = D_8 = 0$$

oznacza, że został zdekodowany rozkaz CMA (rys. 7.2). Kombinacja na wyjściu dekodera rozkazów:

$$D_1 = D_2 = D_3 = D_4 = D_5 = D_6 = D_7 = 0, D_8 = 1$$

oznacza, że został zdekodowany rozkaz OUT (rys. 7.2).

Faza pobrania jest identyczna dla wszystkich rozkazów. Po pobraniu rozkazu następuje faza jego wykonania. Jest ona inna dla każdego rozkazu.

Wykonanie rozkazu INA można zapisać w następujący sposób:

$$D_1 T_2 : AC \leftarrow AC + 1, TC \leftarrow 0$$

Jeżeli $D_1 = 1$ i sygnał taktujący $T_2 = 1$, wtedy zawartość rejestru AC jest zwiększona o 1 i zerowany (reset) jest rejestr TC (*timing counter*). Wyzerowanie TC powoduje, że kolejną zmienną sterującą będzie T_0 (a nie T_3).

Wykonanie rozkazu CMA (rys. 7.2) możemy zapisać w następujący sposób:

$$D_2 T_2 : AC \leftarrow \overline{AC}, TC \leftarrow 0$$

Wykonanie rozkazu LDI OPRD (rys. 7.2) możemy zapisać w następujący sposób:

$$D_3 T_2 : DR \leftarrow M[PC]$$

$$D_3 T_3 : AC \leftarrow DR, PC \leftarrow PC + 1, TC \leftarrow 0$$

gdzie T_3 jest sygnałem taktującym.

Wykonanie rozkazu ADI OPRD (rys. 7.2) można zapisać w następujący sposób:

$$D_4 T_2 : DR \leftarrow M[PC]$$

$$D_4 T_3 : AC \leftarrow AC + DR, PC \leftarrow PC + 1, TC \leftarrow 0$$

Wykonanie rozkazu LDA ADRS (rys. 7.2) można zapisać w następujący sposób:

$$D_5T_2 : DR \leftarrow M[PC]$$

$$D_5T_3 : AR \leftarrow DR, PC \leftarrow PC + 1$$

$$D_5T_4 : DR \leftarrow M[AR]$$

$$D_5T_5 : AC \leftarrow DR, TC \leftarrow 0$$

gdzie T_4 i T_5 są sygnałami taktującymi.

Wykonanie rozkazu STA ADRS (rys. 7.2) można zapisać w następujący sposób:

$$D_6T_2 : DR \leftarrow M[PC]$$

$$D_6T_3 : AR \leftarrow DR, PC \leftarrow PC + 1$$

$$D_6T_4 : DR \leftarrow AC$$

$$D_6T_5 : M[AR] \leftarrow DR, TC \leftarrow 0$$

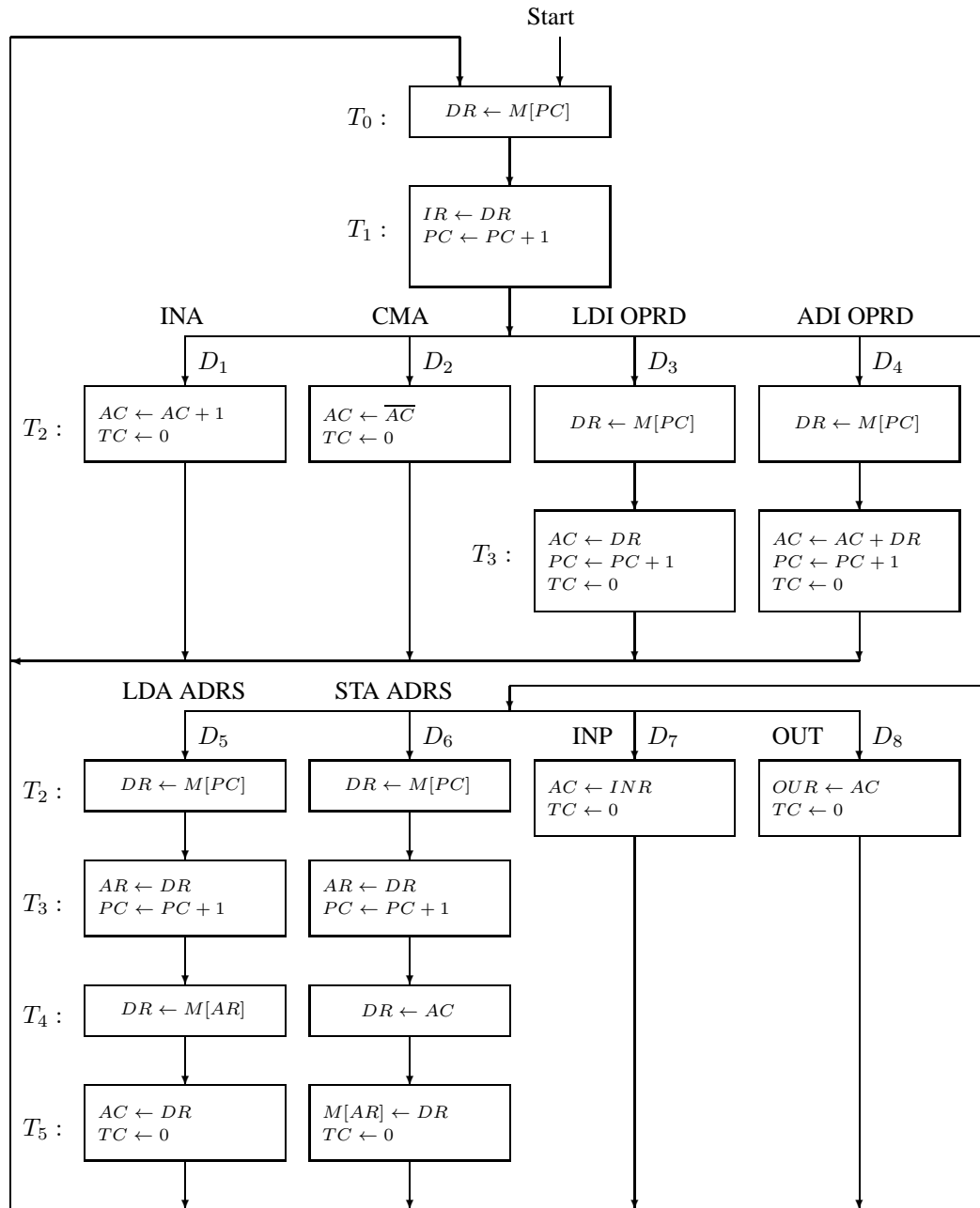
Wykonanie rozkazu INP (rys. 7.2) można zapisać w następujący sposób:

$$D_7T_2 : AC \leftarrow INR, TC \leftarrow 0$$

Wykonanie rozkazu OUT (rys. 7.2) można zapisać w następujący sposób:

$$D_8T_2 : OUR \leftarrow AC, TC \leftarrow 0$$

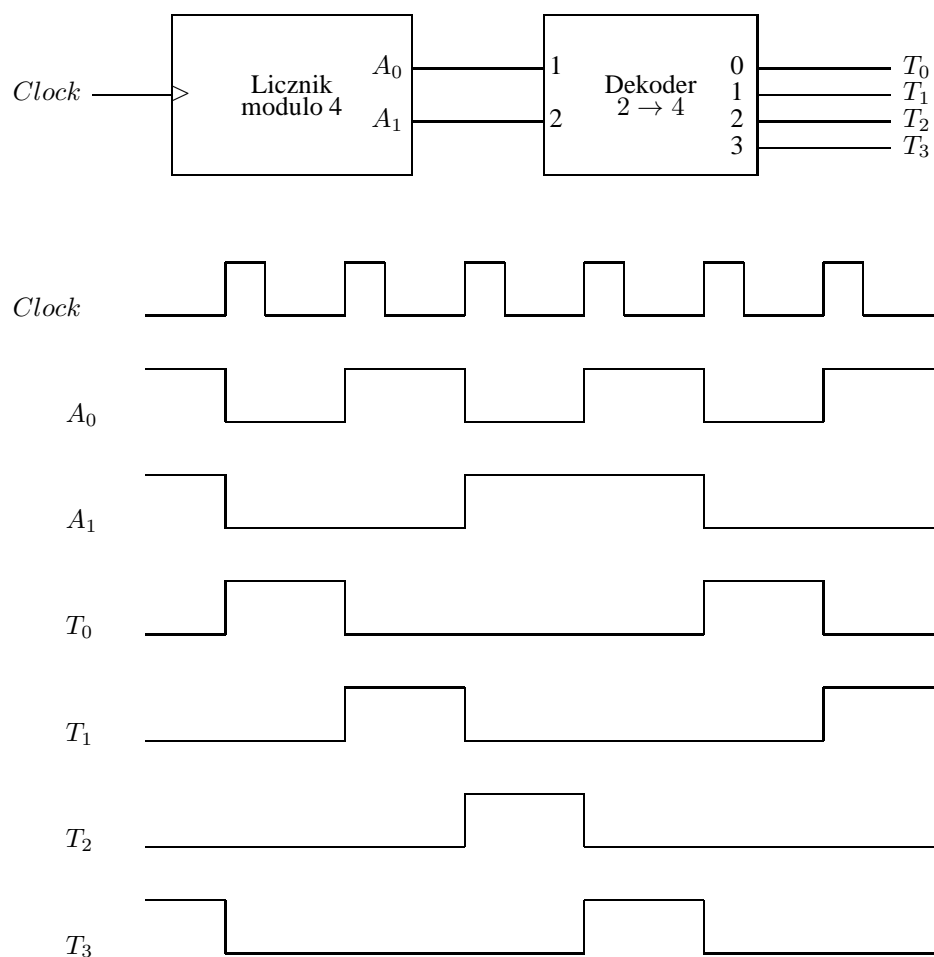
Na rysunku 7.5 ([1]) są wyszczególnione wszystkie mikrooperacje projektowanego komputera. Wykonanie każdego z ośmiu rozkazów (rys. 7.2) wymaga wykonania wybranych mikrooperacji w określonej kolejności.



Rys. 7.5. Mikrooperacje projektowanego komputera

7.5 Generowanie sygnałów taktujących

Na rysunku 7.6 ([1]) jest pokazany schemat blokowy układu generującego 4 kolejne sygnały taktujące T_0 , T_1 , T_2 i T_3 za pomocą licznika modulo 4 i dekodera $2 \rightarrow 4$ ([1]).



Rys. 7.6. Generowanie sygnałów taktujących T_0, T_1, T_2 i T_3

7.6 Projektowanie jednostki sterującej

Mikrooperacja $DR \leftarrow M[PC]$ zostanie wykonana wtedy, kiedy (rys. 7.5)

$$T_0 = 1 \text{ lub } D_3T_2 = 1 \text{ lub } D_4T_2 = 1 \text{ lub } D_5T_2 = 1 \text{ lub } D_6T_2 = 1$$

co można zapisać w następujący sposób:

$$T_0 + (D_3 + D_4 + D_5 + D_6)T_2 : DR \leftarrow M[PC]$$

$$\text{lub } C_1 : DR \leftarrow M[PC]$$

$$\text{gdzie } C_1 = T_0 + (D_3 + D_4 + D_5 + D_6)T_2.$$

Mikrooperacja $PC \leftarrow PC + 1$ zostanie wykonana wtedy, kiedy

$$T_1 = 1 \text{ lub } D_3T_3 = 1 \text{ lub } D_4T_3 = 1 \text{ lub } D_5T_3 = 1 \text{ lub } D_6T_3 = 1$$

co można zapisać w następujący sposób:

$$T_1 + (D_3 + D_4 + D_5 + D_6)T_3 : PC \leftarrow PC + 1$$

$$\text{lub } C_2 : PC \leftarrow PC + 1$$

$$\text{gdzie } C_2 = T_1 + (D_3 + D_4 + D_5 + D_6)T_3.$$

Mikrooperacja $IR \leftarrow DR$ zostanie wykonana wtedy, kiedy $T_1 = 1$, co można zapisać w następujący sposób:

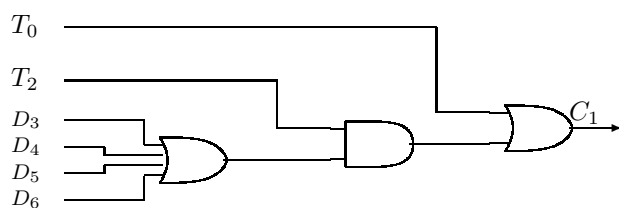
$$T_1 : IR \leftarrow DR \text{ lub } C_3 : IR \leftarrow DR \text{ gdzie } C_3 = T_1$$

Funkcje sterujące C_1, \dots, C_{14} wszystkich mikrooperacji z rysunku 7.5 są wyszczególnione na rysunku 7.7 ([1]).

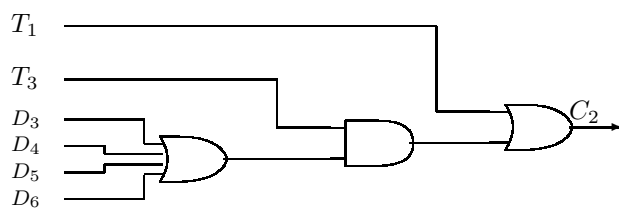
Na podstawie rysunku 7.7 możemy narysować schematy logiczne funkcji sterujących C_1, \dots, C_{14} . Na przykład, na rysunku 7.8 jest pokazany schemat logiczny funkcji C_1 , a na rysunku 7.9 funkcji C_2 .

Funkcja sterująca	Mikrooperacja
$C_1 = T_0 + (D_3 + D_4 + D_5 + D_6)T_2$	$DR \leftarrow M[PC]$
$C_2 = T_1 + (D_3 + D_4 + D_5 + D_6)T_3$	$PC \leftarrow PC + 1$
$C_3 = T_1$	$IR \leftarrow DR$
$C_4 = D_1T_2$	$AC \leftarrow AC + 1$
$C_5 = D_2T_2$	$AC \leftarrow \overline{AC}$
$C_6 = D_3T_3 + D_5T_5$	$AC \leftarrow DR$
$C_7 = D_4T_3$	$AC \leftarrow AC + DR$
$C_8 = (D_5 + D_6)T_3$	$AR \leftarrow DR$
$C_9 = D_6T_4$	$DR \leftarrow AC$
$C_{10} = D_5T_4$	$DR \leftarrow M[AR]$
$C_{11} = D_6T_5$	$M[AR] \leftarrow DR$
$C_{12} = (D_1 + D_2 + D_7 + D_8)T_2 + (D_3 + D_4)T_3 + (D_5 + D_6)T_5$	$TC \leftarrow 0$
$C_{13} = D_7T_2$	$AC \leftarrow INR$
$C_{14} = D_8T_2$	$OUR \leftarrow AC$

Rys. 7.7. Funkcje sterujące mikrooperacji z rysunku 7.5

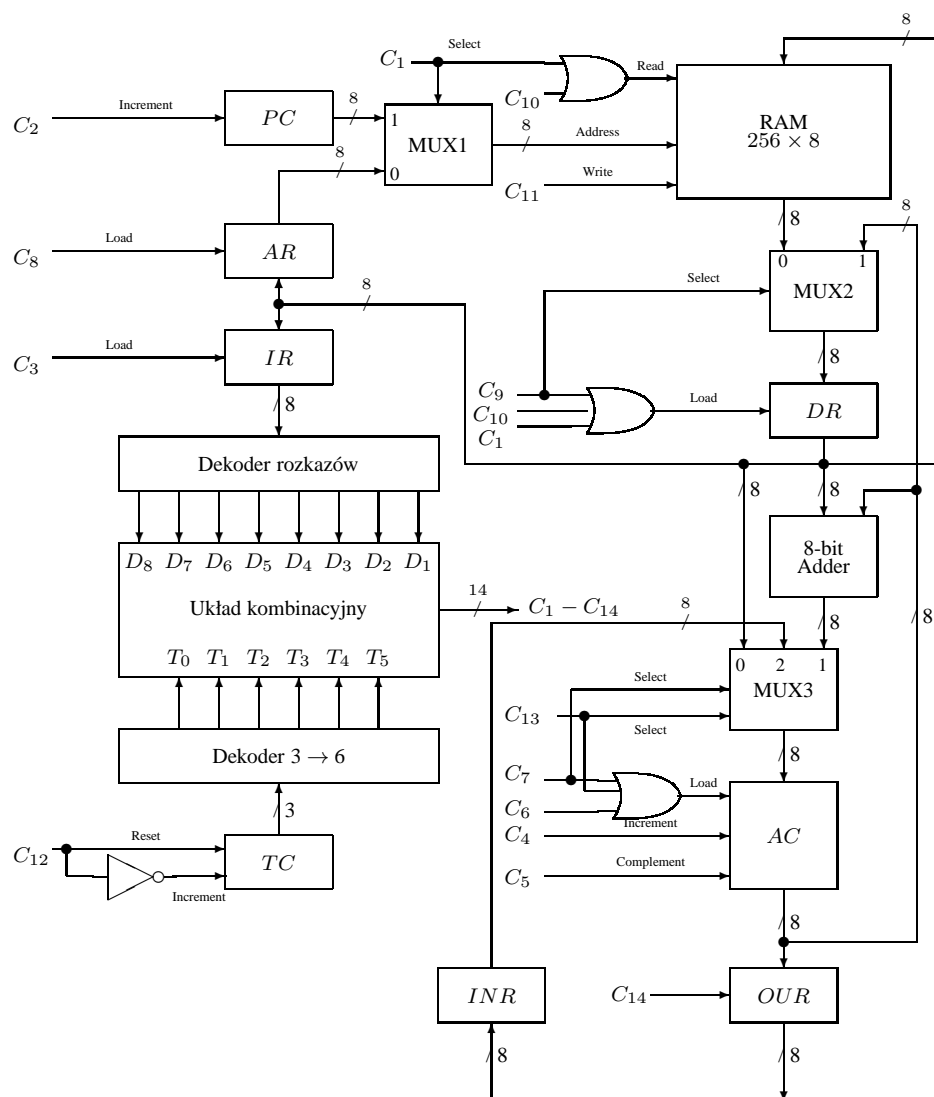


Rys. 7.8. Schemat logiczny funkcji sterującej C_1



Rys. 7.9. Schemat logiczny funkcji sterującej C_2

Schemat logiczny projektowanego komputera jest pokazany na rysunku 7.10 ([1]) (MUX1, MUX2 i MUX3 oznaczają multiplexery).



Rys. 7.10. Schemat logiczny projektowanego komputera

7.7 Podsumowanie

W rozdziale 7 został zaprojektowany prosty, przykładowy komputer. Projekty prostych, przykładowych komputerów są przedstawione, na przykład, w [2, 3, 4].

Literatura

- [1] Mano M.M.: *Computer engineering: hardware design*, Prentice-Hall, 1988.
- [2] Mano M.M.: *Computer system architecture*, Prentice-Hall, 1993.
- [3] Shiva S.G.: *Computer design and architecture*, Little, Brown and Company, 1985.
- [4] Tomek I.: *The foundations of computer architecture and organization*, Computer Science Press, 1990.

Indeks

- algebra Boole'a, 17
- ALU, 85
- bramki logiczne, 34
- CPU, 79
- dekoder, 46
- funkcja logiczna, 17
- iloczyn logiczny (AND), 17
- iloczyn pełny, 18
- jednostka sterująca, 96
- koder, 48
- liczby dwójkowe ze znakiem, 13
- metoda Quine'a–McCluskeya, 28
- metoda tablic Karnaugh'a, 21
- mikrooperacja, 80
- mikroprocesor, 79
- mikroprogram, 106
- mikrorozkaz, 106
- multiplekser, 49
- nadmiar, 14
- NAND (Not AND), 20
- negacja (NOT), 17
- NOR (Not OR), 20
- PAL, 54
- pamięć ROM, 50
- PLA, 53
- PLD, 51
- procesor, 93
- przerzutnik, 57
- przerzutnik *D*, 58
- przerzutnik *JK*, 61
- przerzutnik *SR*, 57
- przerzutnik *T*, 62
- rejestr, 70
- słowo sterujące, 96
- sterowanie sprzętowe, 98
- suma logiczna (OR), 17
- suma pełna, 18
- sumator, 44
- system ósemkowy, 7
- system dwójkowy, 7
- system funkcjonalnie pełny, 20
- system szesnastkowy, 7
- tablica prawdy, 17
- układ kombinacyjny, 34
- układ przesuwania, 91
- układ sekwencyjny, 57
- uzupełnienie do $(p - 1)$, 10
- uzupełnienie do p , 10
- uzupełnienie do 1, 11
- uzupełnienie do 2, 11