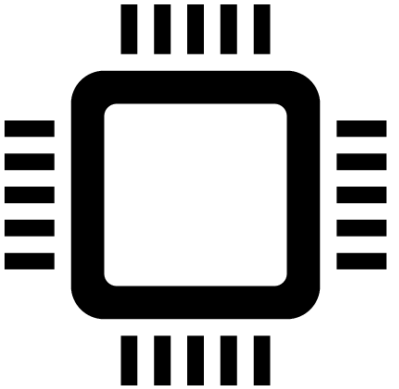
Programowanie struktur cyfrowych



Podstawowe zagadnienia Układy kombinacyjne

dr Aleksander Lamża

Uniwersytet J. Kochanowskiego w Kielcach
Uniwersytet Śląski w Katowicach

aleksander.lamza@us.edu.pl

Układy kombinacyjne

Zapoznawanie się z bardziej złożonymi strukturami logicznymi rozpoczniemy od **układów kombinacyjnych**.
Tak naprawdę w pewnym sensie mieliśmy już z takim układem do czynienia – nasze urządzenie z przyciskami i światłami było układem kombinacyjnym.

Cechą wyróżniającą układy kombinacyjne jest to,
że **stan ich wyjść zależy jedynie od stanu wejść**.

To odróżnia je od układów sekwencyjnych, o których pomówimy później.

Układy kombinacyjne

Jak łatwo zauważyć, istnieje nieskończona liczba możliwych układów kombinacyjnych, jednak w praktyce używa się zwykle pewnych stałych, „wzorcowych” struktur.

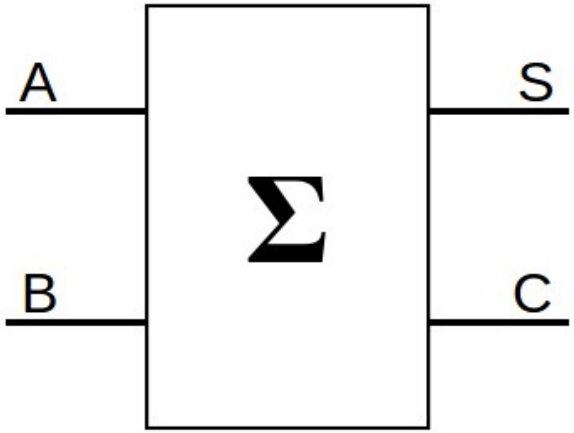
Można je podzielić na trzy główne grupy:

- **układy arytmetyczne**,
- **układy łączące**,
- **układy kodujące**.

Układy kombinacyjne

Zaczynamy od **układów arytmetycznych**.
Na rysunku przedstawiającym budowę procesora widoczny był blok ALU. Jest to jednostka arytmetyczno-logiczna, której zadaniem jest wykonywania działań. Jej podstawowym elementem jest **sumator**. Przyjrzymy mu się bliżej – przeanalizujemy budowę sumatora jednobitowego.

Układ ma dwa wejścia (składniki A i B) oraz dwa wyjścia (S – suma, C – przeniesienie):



Wyjaśnienia wymaga tylko wyjście C. Jest to **przeniesienie**, do którego dochodzi, gdy sumy dwóch składników A i B nie da się przedstawić za pomocą jednego bitu (jest to sytuacja analogiczna do przeniesienia przy dodawaniu liczb dziesiętnych, np. 5+7 = 12, przy czym „1” oznaczające 10 jest właśnie przeniesieniem).

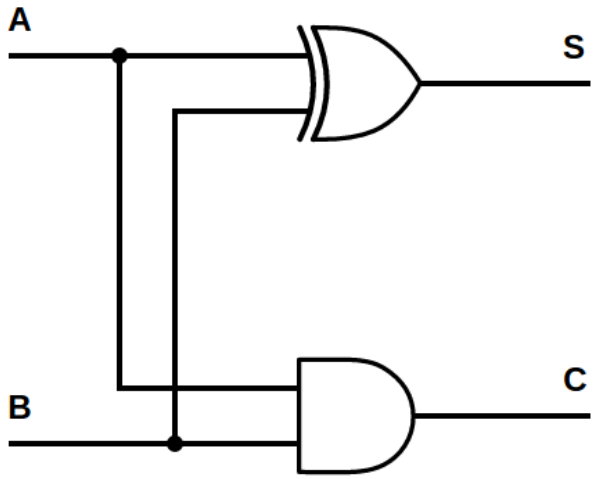
W związku z tym tabela prawdy dla takiego sumatora wygląda następująco:

A	B	S	C	
0	0	**0**	**0**	0 + 0 = 0
0	1	**1**	**0**	0 + 1 = 1
1	0	**1**	**0**	1 + 0 = 1
1	1	**0**	**1**	1 + 1 = 10 (2)

Bez problemu powinniście podać funkcje, które odpowiadają za wyjścia S i C. Co to będzie?

Wyjście sumy realizuje funkcję $A \oplus B$, czyli alternatywę, natomiast przeniesienie to iloczyn $A \cdot B$.

Schemat sumatora zrealizowanego na bramkach wygląda więc tak:



Układ, który zbudowaliśmy, jest tak naprawdę **półsumatorem**, ponieważ **nie ma wejścia przeniesienia**, które jest niezbędne, gdybyśmy chcieli zbudować wielobitowy sumator.

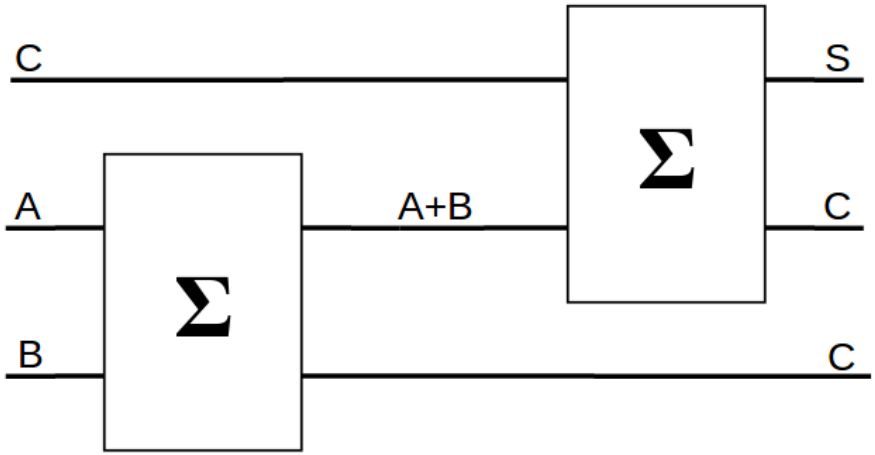
Jak więc będzie wyglądał prawdziwy, **pełny sumator**?

Znów rozpoczynamy od tabeli prawdy:

C_{i-1}	A	B	S	C_i	
0	0	0	**0**	**0**	0 + 0 + 0 = 0
0	0	1	**1**	**0**	0 + 0 + 1 = 1
0	1	0	**1**	**0**	0 + 1 + 0 = 1
0	1	1	**0**	**1**	0 + 1 + 1 = 10 (2)
1	0	0	**1**	**0**	1 + 0 + 0 = 1
1	0	1	**0**	**1**	1 + 0 + 1 = 10
1	1	0	**0**	**1**	1 + 1 + 0 = 10
1	1	1	**1**	**1**	1 + 1 + 1 = 11 (3)

Sprawa się skomplikowała, ponieważ mamy trzy wejścia, więc osiem możliwych stanów. Gdybyśmy chcieli sformułować funkcje z tabeli prawdy (jak do tej pory), powstałyby niezbyt miłe „potworki”.

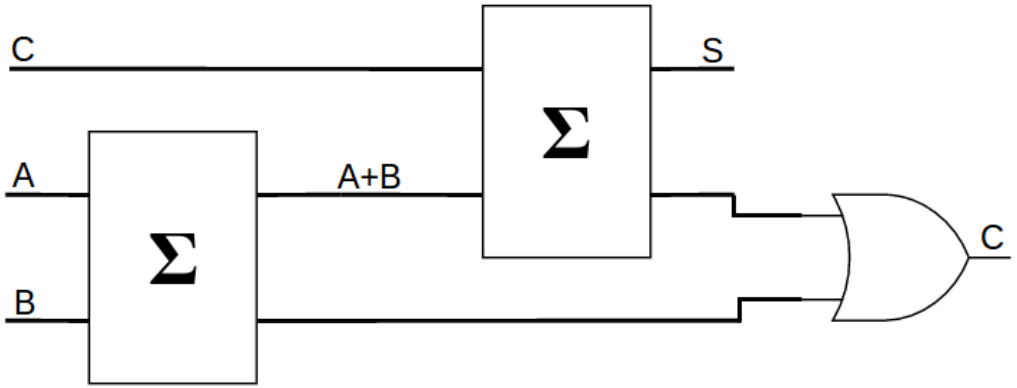
Musimy to rozwiązać inaczej. Odpowiedzmy na pytanie, co należy zrobić z wartością z wejścia przeniesienia.
Oczywiście dodać do sumy A i B.
Rozwiązanie wydaje się więc proste. Wystarczy zastosować dwa półsumatory:



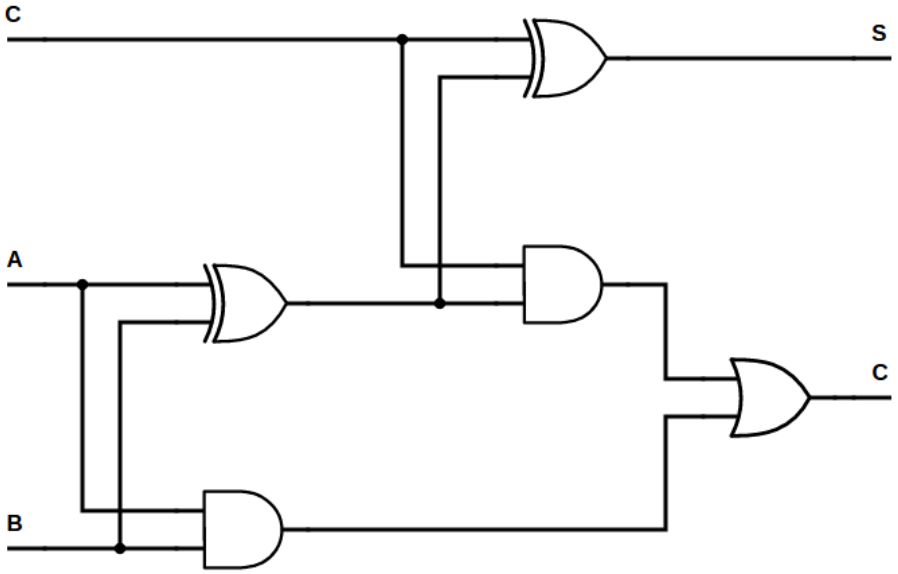
Problem jednak w tym, że mamy aż dwa wyjścia przeniesienia.
Rzut oka na tabelę prawdy wystarczy, by stwierdzić, że przeniesienie ma wystąpić, jeśli dojdzie do niego podczas dodawania A+B **lub** podczas dodawania sumy cząstkowej i przeniesienia.

Układy kombinacyjne

Jedna bramka OR rozwiązuje sprawę:



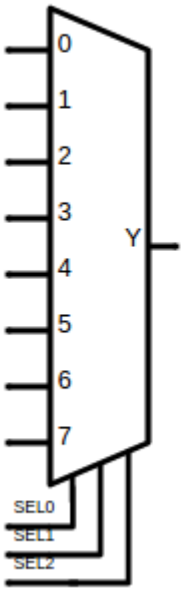
Schemat na bramkach:

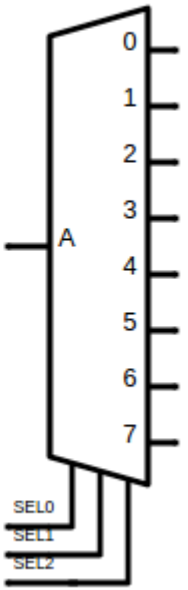


Układy kombinacyjne

Kolejnym typem układów kombinacyjnych są bloki łączące.
Ich zadaniem jest połączenie odpowiednich wejść bądź wyjść z resztą układu. Wyróżniamy:

multipleksery i demultipleksery



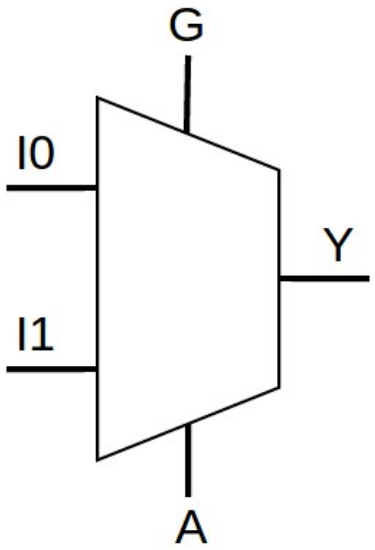


Układy kombinacyjne

Z układami tego typu (ale niekoniecznie cyfrowymi) z pewnością mieliście do czynienia.
Na przykład w sprzęcie grającym mamy zwykle możliwość wyboru sygnału (z CD, tunera, z wejścia AUX itp.), który trafi do wzmacniacza – to jest właśnie multiplekser.

Przyjrzymy się bliżej multiplekserowi. Zobaczymy, jak go zbudować z bramek logicznych.

Na warsztat weźmiemy najprostszy możliwy multiplekser, czyli taki o dwóch wejściach:



Multipleksowane wejścia to I0 i I1. W zależności od stanu linii adresowej A, na wyjście Y trafi sygnał z wejścia I0 (dla A = 0) lub I1 (dla A = 1). A za co odpowiada sygnał G? Jest to wejście uaktywniające – multiplekser działa, jeżeli G = 1. Jeśli G = 0, na wyjściu zawsze jest 0, niezależnie od stanu pozostałych wejść.

Układy kombinacyjne

Jak zwykle zaczynamy od tabeli prawdy:

G	A	I0	I1	Y
0	0	0	0	**0**
0	0	0	1	**0**
0	0	1	0	**0**
0	0	1	1	**0**
0	1	0	0	**0**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**0**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**1**
1	1	1	0	**0**
1	1	1	1	**1**

No ładnie... Uwzględniliśmy wszystkie możliwe stany (dla 4 wejść jest ich aż 16!).

Czy naprawdę jest to konieczne?

Wiemy przecież, że jeśli na wejściu G jest 0, na wyjściu również pojawia się 0.

Dodatkowo wiemy, że jeśli A = 0, na wyjściu pojawi się stan z I0, a stan I1 nie ma znaczenia. Analogicznie sprawa wygląda dla A = 1 i wejścia I1.

W związku z tym tabelę prawdy można uprościć, pomijając „niepotrzebne” pozycje. Wprowadzamy symbol X oznaczający dowolny stan (0 lub 1):

G	A	I0	I1	Y
0	X	X	X	**0**
1	0	0	X	**0**
1	0	1	X	**1**
1	1	X	0	**0**
1	1	X	1	**1**

Uff... Znacznie lepiej.

Zastanówmy się teraz, jak zbudować ten układ z wykorzystaniem bramek. Musimy znaleźć taką bramkę, którą możemy „przepuszczać” lub „blokować” sygnał z jednego wejścia .

Układy kombinacyjne

Spójrzmy na tabelę prawdy czterech bramek: OR, NOR, AND i NAND:

A	B	OR	NOR	AND	NAND
0	0	**0**	**1**	**0**	**1**
0	1	**1**	**0**	**0**	**1**
1	0	**1**	**0**	**0**	**1**
1	1	**1**	**0**	**1**	**0**

Która z nich będzie się nadawała?

Podpowiedź: szukajcie takiej, która dla zera na jednym wejściu, niezależnie od stanu drugiego, zawsze daje zero na wyjściu.

Układy kombinacyjne

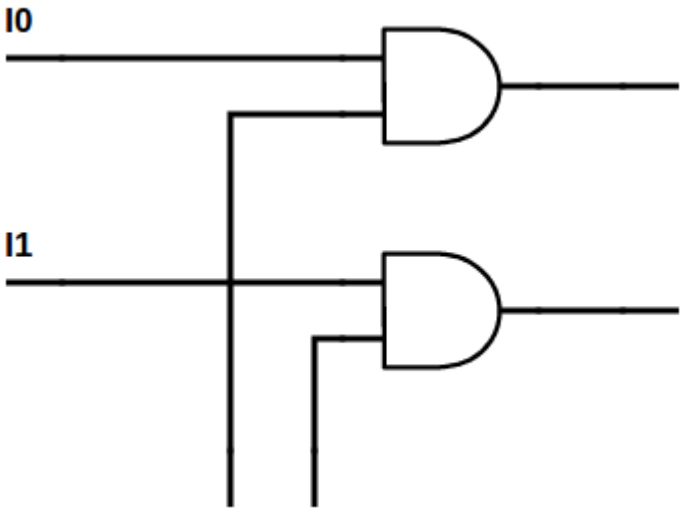
Prawidłowa odpowiedź: **AND**

A	I	Y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Dla A = 0 wyjście zawsze jest 0.

Dla A = 1 wyjście jest równe I.

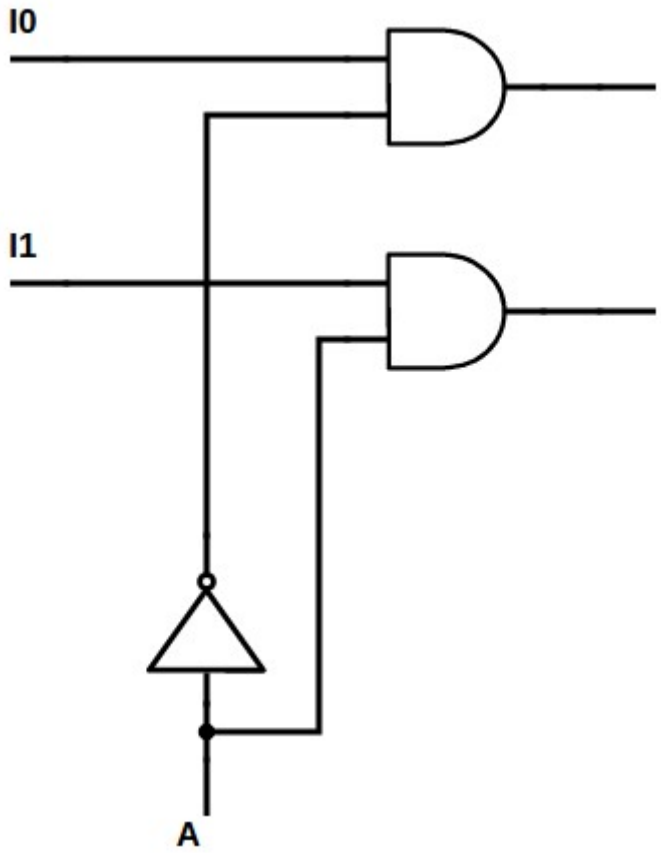
Weźmy teraz dwie bramki AND:



„Przewodzić” będzie ta bramka, na którą podamy 1.
Pamiętajmy, że w multiplekserze zawsze ma „przewodzić” tylko jedna bramka. Analizowany przez nas układ ma tylko dwa wejścia, więc jeżeli 1 podamy na pierwszą bramkę, na drugą musimy podać 0.

Stąd prosty wniosek, że do linii wybierających (adresowych) musimy użyć negacji.

W tej chwili jeśli A = 0 „przewodzi” pierwsza bramka, a jeśli A = 1 „przewodzi” druga. Świetnie!



Czy zauważyliście na tym schemacie coś niepokojącego?

Układy kombinacyjne

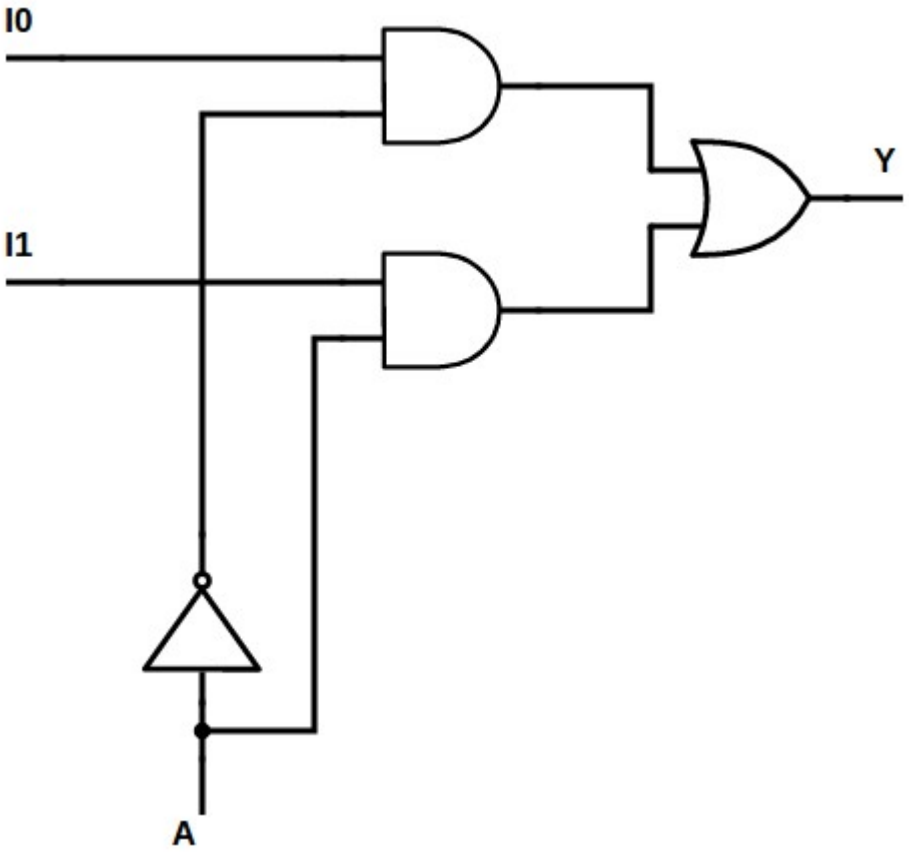
No jasne! Mamy dwa wyjścia, a powinno być jedno. Co z tym zrobić?

Powinniśmy „zebrać” stany z wszystkich bramek przełączających i „przepuścić” jedynkę.

Jaką bramką to zrobić?

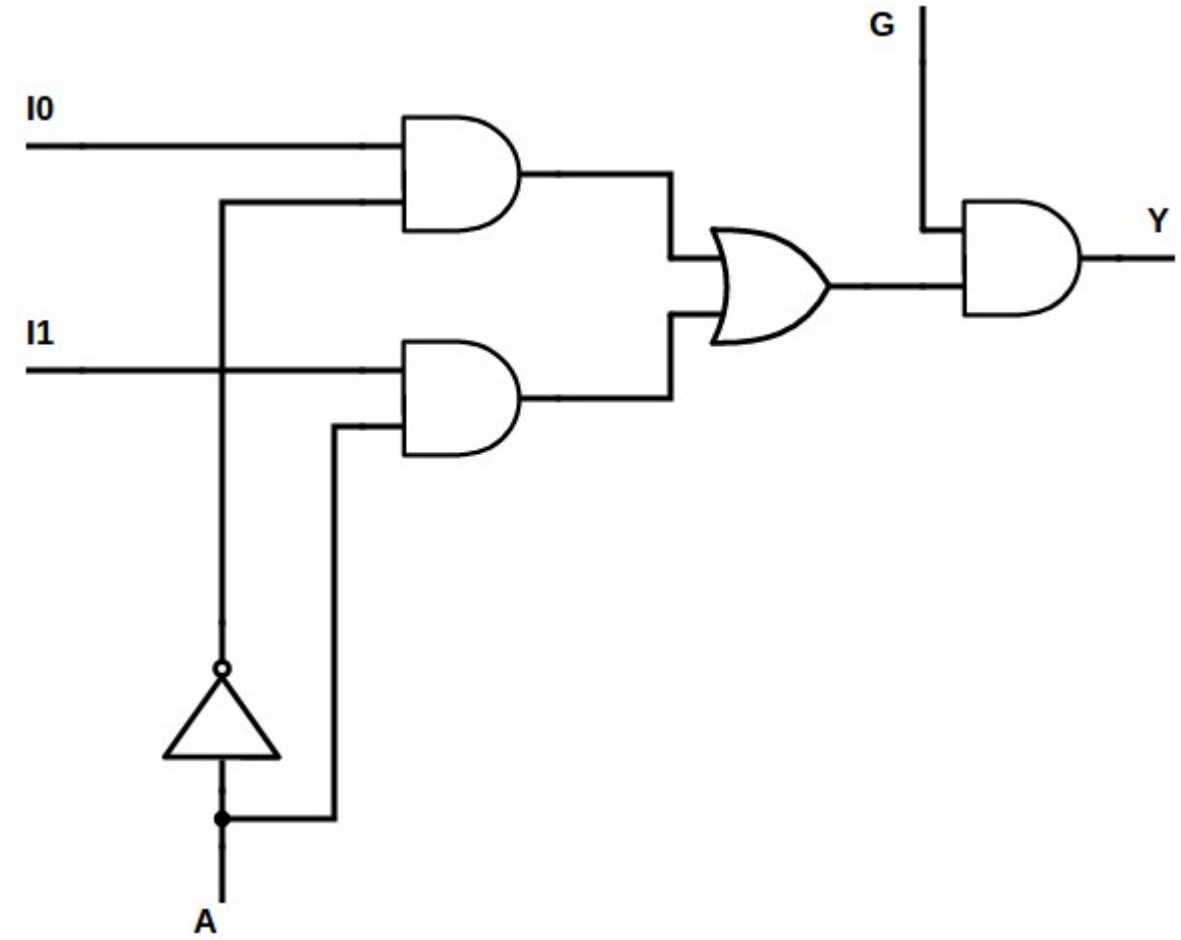
A	B	OR	NOR	AND	NAND
0	0	**0**	**1**	**0**	**1**
0	1	**1**	**0**	**0**	**1**
1	0	**1**	**0**	**0**	**1**
1	1	**1**	**0**	**1**	**0**

Odpowiedź jest oczywista – **OR**.

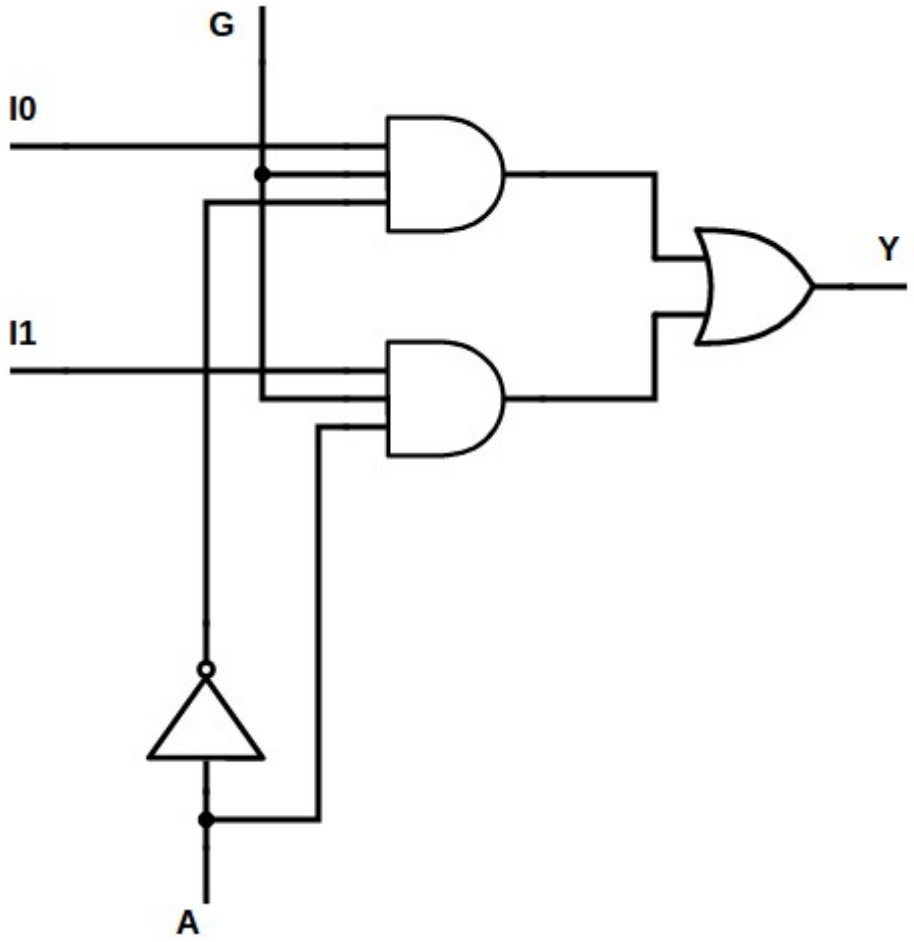


Mamy już działający multiplekser! Linią A wybieramy wejście, którego stan pojawi się na wyjściu.
Brakuje tylko wejścia aktywującego G.

Możemy powielić rozwiązanie z bramką AND, która będzie „przepuszczała” (bądź nie) stan z wyjścia dotychczasowego multipleksera:

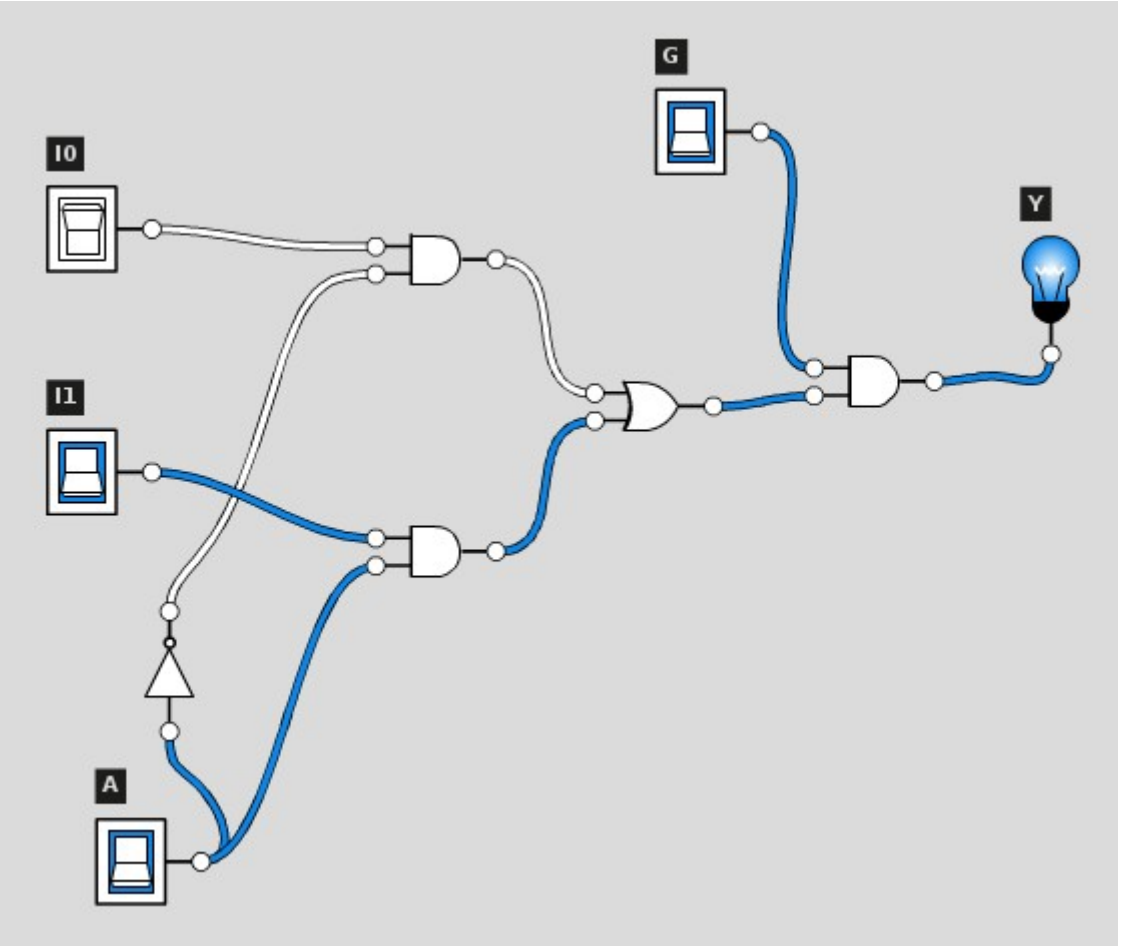


Możemy też podejść do sprawy inaczej i blokować multiplekser na wejściu:



Układy kombinacyjne

Zachęcam, żebyście zbudowali sobie ten układ w logic.ly
i przeanalizowali jego działanie.



Układy kombinacyjne

Ostatnią z omawianych grup układów kombinacyjnych są bloki kodujące (nie mylić z szyfrowaniem). Ich zadaniem jest zamiana jednego kodu na inny.

Klasyczny podział wyróżnia:
- **kodery**, które zamieniają kod „1 z n” na wartość binarną,
- **dekodery** działające odwrotnie do koderów,
- **transkodery**, które zamieniają jeden kod na inny.

Aby zobaczyć, jak są zbudowane ogólnie rozumiane kodery, zaprojektujemy transkoder kodu BCD na 7-segmentowy.

Czym jest **kod BCD**?

Rozwinięcie skrótu BCD to Binary-Coded Decimal. Jest to kod reprezentujący wszystkie cyfry systemu dziesiętnego za pomocą systemu dwójkowego.

cyfra	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

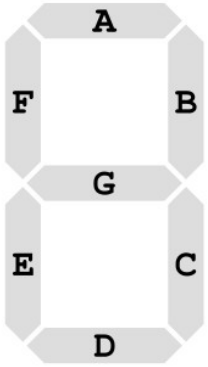
Jest to naturalny kod dwójkowy. Wartości zapisujemy na czterech bitach.
Kod ten jest zazwyczaj stosowany, gdy wartości mają być wyświetlane w postaci dziesiętnej, np. na wyświetlaczu.

Na przykład liczba dziesiętna 62 w BCD wygląda tak:

0110 0010

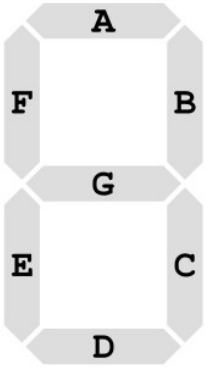
Czym jest **kod 7-segmentowy**?

Jest to kod służący do sterowania wyświetlaczami 7-segmentowymi:



Jest to powszechny sposób wyświetlania cyfr (i innych symboli).
Segmenty oznacza się literami od A do G.

Kod 7-segmentowy dla cyfr od 0 do 9:



cyfra	A	B	C	D	E	F	G
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1

cyfra	A	B	C	D	E	F	G
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Układy kombinacyjne

Ponieważ naszym zadaniem jest zaprojektowanie transkodera BCD na kod 7-segmentowy, musimy zdefiniować funkcje opisujące każdy segment w zależności od wartości BCD.

BCD dcba	**7-segmentowy** ABCDEFG
0000	1111110
0001	0110000
0010	1101101
0011	1111001
0100	0110011
0101	1011011
0110	1011111
0111	1110000
1000	1111111
1001	1111011

Robimy to tak samo, jak ostatnio:

$$A = \overline{a}\cdot\overline{b}\cdot\overline{c}\cdot\overline{d} + \overline{a}\cdot b\cdot\overline{c}\cdot\overline{d} + a\cdot b\cdot\overline{c}\cdot\overline{d} + a\cdot\overline{b}\cdot c\cdot\overline{d} + \overline{a}\cdot b\cdot c\cdot\overline{d} + a\cdot b\cdot c\cdot\overline{d} + \overline{a}\cdot\overline{b}\cdot\overline{c}\cdot d + a\cdot\overline{b}\cdot\overline{c}\cdot d$$

Wyszła nam dosyć rozbudowana funkcja. Można ją odwzorować w układzie w takiej postaci lub ją zminimalizować.

Do minimalizowania funkcji logicznych używa się zasad logiki Boole'a.
Bardzo pomocna byłaby też siatka Karnaugha.

Układy kombinacyjne

Do tej pory zawsze zakładaliśmy, że stanem domyślnym jest 0 i określamy sytuacje, w których ma wystąpić stan 1.
A gdyby podejść do sprawy na odwrót? Czyli? Szukamy zer i łączymy je w trochę inny, ale równoważny, sposób:

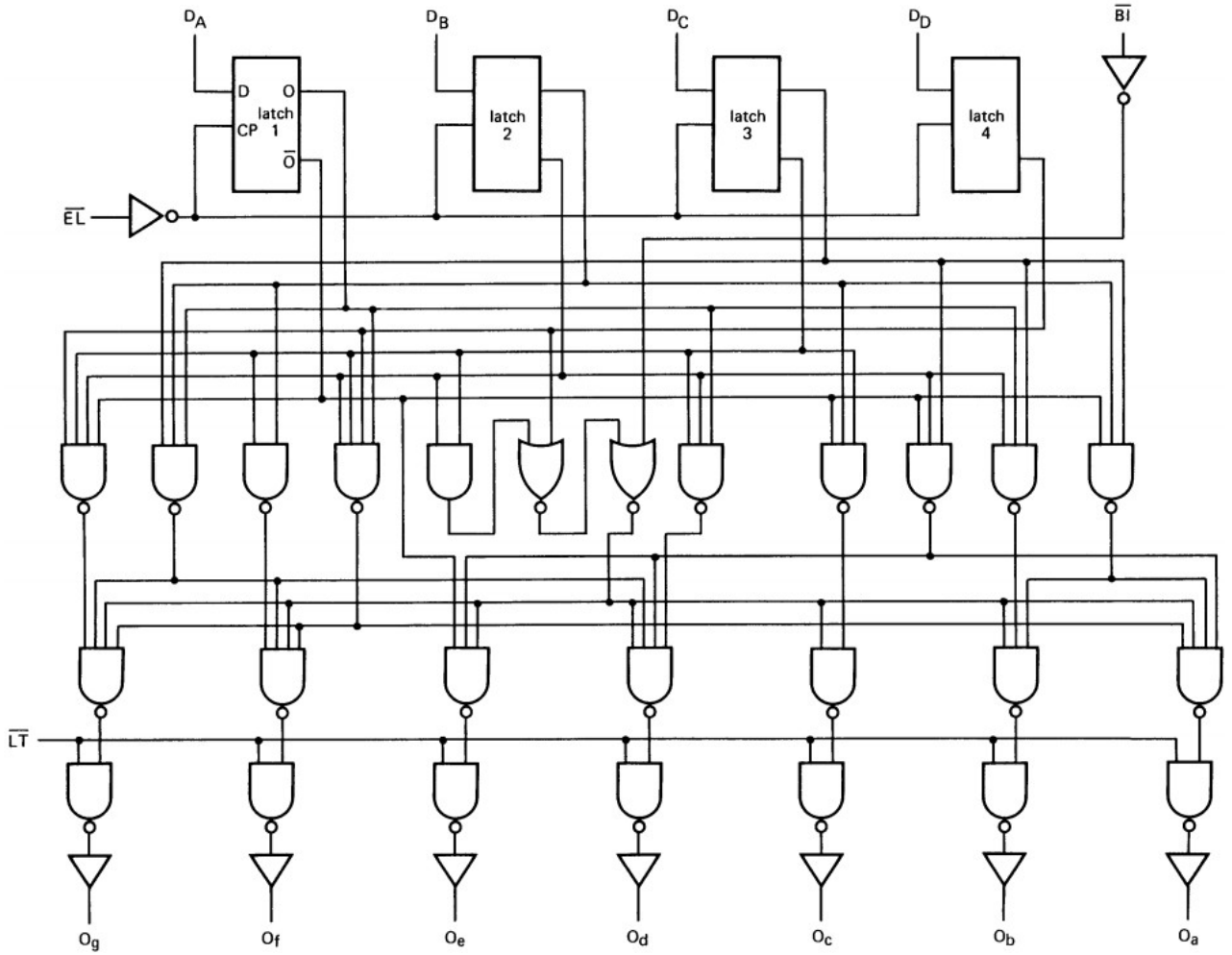
BCD dcba	**7-segmentowy** ABCDEFG
0000	1111110
0001	0110000
0010	1101101
0011	1111001
0100	0110011
0101	1011011
0110	1011111
0111	1110000
1000	1111111
1001	1111011

$$A = \overline{a \cdot \overline{b} \cdot \overline{c} \cdot \overline{d}} \cdot \overline{\overline{a} \cdot \overline{b} \cdot c \cdot \overline{d}}$$

Zwróćcie uwagę na negacje nad iloczynami oraz to, że iloczyny są łączone również iloczynem, a nie sumą.

Układy kombinacyjne

Praktyczna realizacja tego dekodera może wyglądać tak jak na poniższym rysunku.



Jest to schemat rzeczywistego układu dekodera o symbolu 4511.

Układy kombinacyjne

Na tym zakończyliśmy omawianie układów kombinacyjnych.
Teraz pora na **układy sekwencyjne**.