Proste układy logiczne

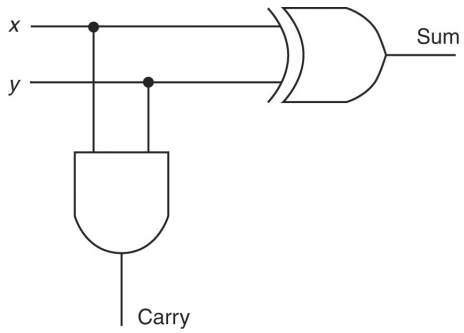
14.X.2009

1 Układ dodający liczby binarne

Spróbujemy się teraz przekonać, że układy wchodzące w skład komputera naprawdę da się zbudować używając tylko prostych bramek logicznych. Na początek zbudujemy układ, który dodaje dwie liczby binarne. Układy takiego typu są podstawowymi składnikami jednostek arytmetyczno-logicznych procesorów.

1.1 Półsumator (ang. *half-adder*) i sumator (and *adder*)

Półsumator dodaje pojedyncze bity, zwraca też przeniesienie (ang. *carry*). Sumator ma trzy wejścia: dwa bity do zsumowania i poprzednie przeniesienie. (Pełny) sumator zbudujemy z dwóch półsumatorów i bramki OR.



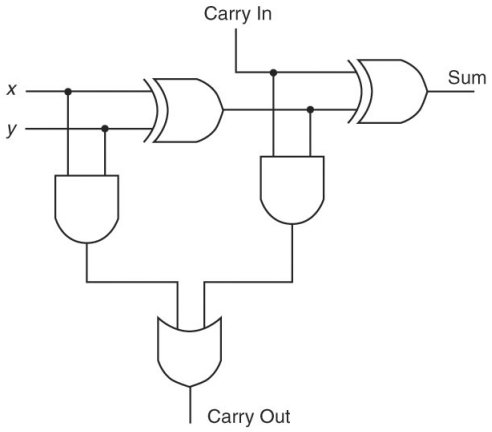
Rysunek 1: Półsumator

1.2 Sumator kaskadowy

Budujemy sumator kaskadowy z n sumatorów (ewentualnie $n-1$ i jednego półsumatora). Uwaga: taki układ w rzeczywistości nie jest używany w praktyce. Stosowane są pewne ulepszenia (carry-look-ahead, carry-select, carry-save). „Urównoleglenie" pewnych operacji i zredukowanie maksymalnej ścieżki przeniesienia pozwala uzyskać prędkości o 40-90 procent lepsze niż sumator kaskadowy. Niedługo omówimy niektóre z takich rozwiązania.

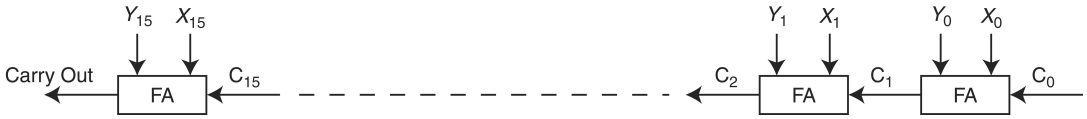
Inputs			Outputs	
x	y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)



(b)

Rysunek 2: Sumator

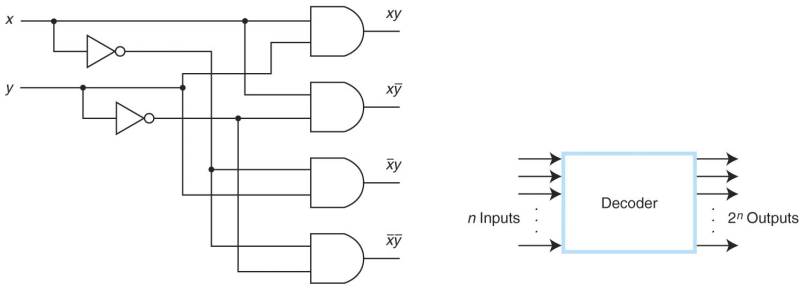


Rysunek 3: 16-bitowy sumator kaskadowy

2 Inne proste układy kombinacyjne

2.1 Dekoder

Układ dostaje na n wejściach zakodowaną (w naturalnym kodzie binarnym) liczbę binarną. Ma 2^n wyjść, jego zadaniem jest uaktywnienie (tzn. ustawienie na nim '1') dokładnie tego o numerze wskazywanym przez liczbę na wejściu.



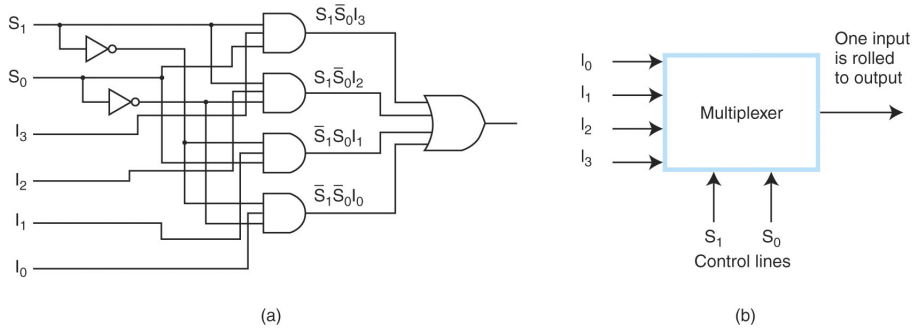
(a)

(b)

Rysunek 4: (a) Dekoder 2-do-4 (b) ogólne oznaczenie

2.2 Multiplekser

Układ na n wejść z danymi (n jest zazwyczaj potęgą dwójki) i pewną liczbę linii sterujących (mniej więcej $log_2 n$). Wejścia sterujące wskazują, które z wejść z danymi ma być skierowane na wyjście. Przykładowe wykorzystanie: sekwencyjne przesyłanie danych na jedno wyjście (stosowane np. do przesyłania rozmów telefonicznych). Z drugiej strony dane są rozdzielane przez demultiplekser.



Rysunek 5: (a) Multiplekser o 4 wejściach danych (b) ogólne oznaczenie

2.3 Koder, demultiplekser

2.4 Prosta dwubitowa jednostka ALU

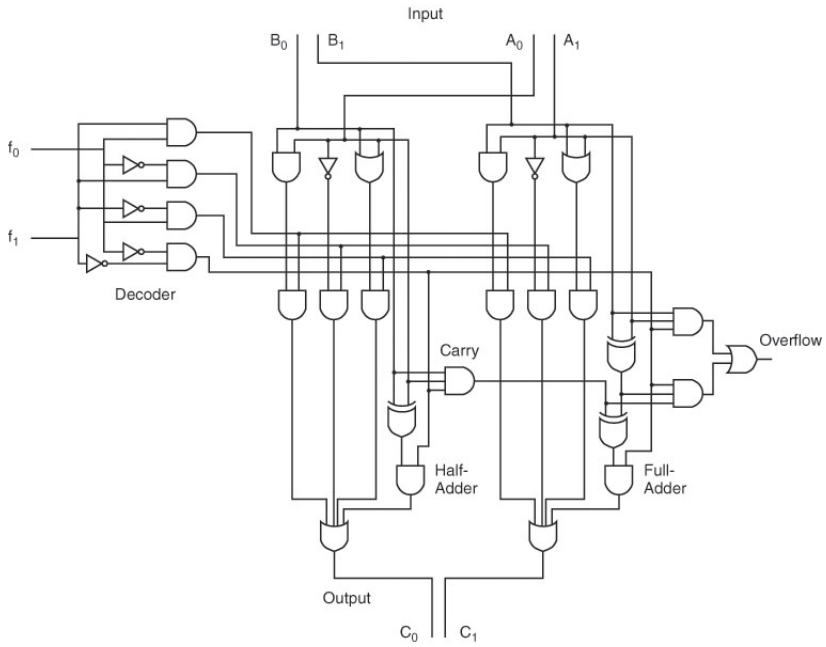
Na wykładzie zaprezentowałem bardziej „modularne" podejście do jej tworzenia (z wykorzystaniem wcześniej wprowadzonych układów). Różne kombinacje wejść sterujących f_0, f_1 oznaczają różne działnia na wejściach A, B: 00 to suma, 01 - negacja wejścia A, 10 - bitowa alternatywa, 11- bitowa koniunkcja.

3 Reprezentacja uzupełnienień do 2. Układ dodający i odejmujący.

Będziemy chcieli teraz zmodyfikować nieco układ, którego używaliśmy do dodawania liczb binarnych, tak aby potrafił on też wykonywać odejmowanie. Ustalmy dla uproszczenia, że używamy arytmetyki 4-bitowej (w praktyce 8,16,32,64). Potrafimy zatem reprezentować liczby z przedziału $[0, 15]$. Oczywiście przy dodawaniu liczb może wystąpić błąd przepełnienia.

Pytanie: jak reprezentować liczby ujemne? Narzucające się rozwiązanie: pierwszy bit oznacza znak liczby. Wady: liczba 0 ma dwie reprezentacje (rozrzutne i niewygodne lub nawet niebezpieczne w obliczeniach), nasz sumator kaskadowy nie potrafi poprawnie wykonać operacji $a + (-a)$. Chcemy mieć taką reprezentację, aby nasz układ bez żadnych modyfikacji poprawnie (z wyjątkiem sytuacji, gdy występuje przepełnienie) dodawał wszystkie reprezentowane liczby. Reprezentacją taką jest reprezentacja uzupełnień do 2.

Co się dzieje, gdy do 0001 dodamy 1111? Dostajemy 0000. Zatem jeśli 0000 reprezentuje liczbę 0, a 0001 liczbę 1, to 1111 powinno być reprezentacją -1. Ile trzeba dodać do 0101, aby otrzymać 0000? Odpowiedź: 1011. Jak zatem otrzymać liczbę przeciwną do zadanej: znajdujemy ostatnią jedynkę, pozostawiamy ją razem z kolejnymi bitami, a bity wcześniejsze odwra-



Rysunek 6: Prosta dwubitowa jednostka arytmetyczno-logiczna

camy. Inaczej: odwracamy wszystkie bity (dopełnienie do 1) po czym do wyniku dodajemy 1 (dopełnienie do 2).

Pełna tabelka:

repr.	bez znaku	ze znakiem
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Zauważmy:

- liczby 0-7 mają w obu reprezentacjach takie same postacie

- pierwszy bit odróżnia liczby ujemne od nieujemnych

Jak zrealizować odejmowanie? Korzystamy z faktu, że $a - b = a + (-b)$. Czyli jeśli układ dostaje sygnał odejmowania wystarczy odwrócić bity drugiej liczby (XOR), dodać do niej 1 (jednynkę możemy podać na wejście c_{in} pierwszego sumatora) i dodać do pierwszej. Zauważmy, że nasz układ potrafi poprawnie odejmować także liczby bez znaku (mniejszą od większej). Zwróć uwagę, że tak naprawdę aby wykonać operację $a - b$ nasz układ wykonuje: $a + (1111 - b) + 1$. Szczegóły zostały omówione na wykładzie.