Podstawy Techniki Cyfrowej

Kompendium do ASMR o układach cyfrowych, czyli pierwszych zajęć prowadzony w technice binauralnej

"Co słychać?"

Szept wybrzmiewa w uszach
szzz... pfff... nikły mlask
to kolejny mikrofonu trzask
umysł zanosi się w wszach

Szept wybrzmiewa w uszach
brr... prrr... głośny kwik
zbudujemy przerzutnik
nie myślmy o głuszach

Szept wybrzmiewa w uszach...

Opracowanie powstało na podstawie zajęć i materiałów udostępnionych przez dr Rafała Walkowiaka, dr Krzysztofa Bucholca oraz innych prowadzących z Politechniki Poznańskiej. Autorzy opracowania nie ponoszą odpowiedzialności za ewentualne błędy.

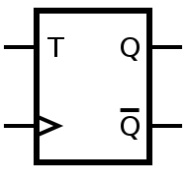
1 AUTOMATY SYNCHRONICZNE

1.1 TYPY PRZERZUTNIKÓW

1.1.1 Przerzutnik T

Inaczej *toggle flip-flop.* Jego działanie polega na zamianie stanu przerzutnika na przeciwny, jeśli wejście T jest w stanie wysokim oraz pojawi się zbocze zegara. Jeśli T ma stan niski to stan przerzutnika się nie zmienia.

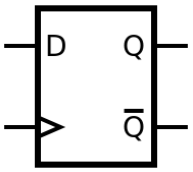
T	Q(t)	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0



1.1.2 Przerzutnik D

Inaczej *data* lub *delay flip-flop*. Nazywany jest często przerzutnikiem opóźniającym, ponieważ nie robi nic bardziej ambitnego niż przepisywanie stanu wejścia D, gdy pojawi się zbocze zegara.

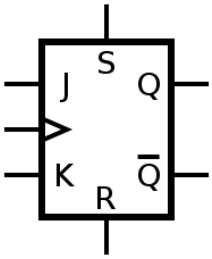
D	Q(t)	Q(t+1)
0	0	0
0	1	0
1	0	1
1	1	1

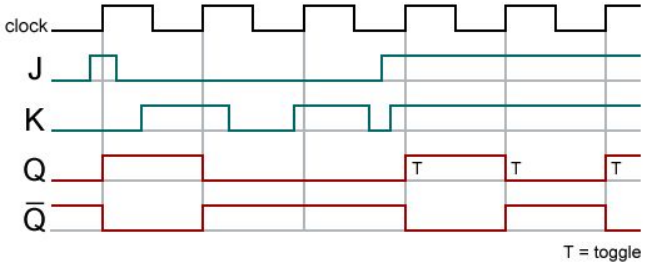


1.1.3 Przerzutnik JK

Najbardziej wkurzający z przerzutników, bo ma dwa wejścia: J – ustawiające, K – kasujące. Jeśli J będzie w stanie wysokim to przerzutnik też się w taki ustawi. Jeśli K będzie w stanie wysokim to przerzutnik ustawi się na stan niski. Jeśli oba będą w stanie wysokim to następuje zmiana stanu przerzutnika na przeciwny. Jeśli oba będą w stanie niskim to przerzutnik utrzymuje swój stan.

J	K	Q(t)	Q(t+1)	Opis
0	0	0	0	Podtrzymanie
0	0	1	1	Podtrzymanie
0	1	0	0	Ustawienie 0
0	1	1	0	Ustawienie 0
1	0	0	1	Ustawienie 1
1	0	1	1	Ustawienie 1
1	1	0	1	Zamiana
1	1	1	0	Zamiana





Na obrazku burżujska wersja tego drania z wejściem Set i Reset. Obok diagram przejść.

1.2 Zamiana przerzutników na inne przerzutniki

Ogólna metoda zamiany przerzutników na inne polega na porównaniu ich tablicy prawdy. Dla stanów $Q(t)$ i $Q(t+1)$ docelowej bramki dobiera się odpowiedne stany wejścia bramki bazowej. Następnie całość można poddać minimalizacji i wybrać sobie odpowiednie bramki, które zrealizują ten układ.

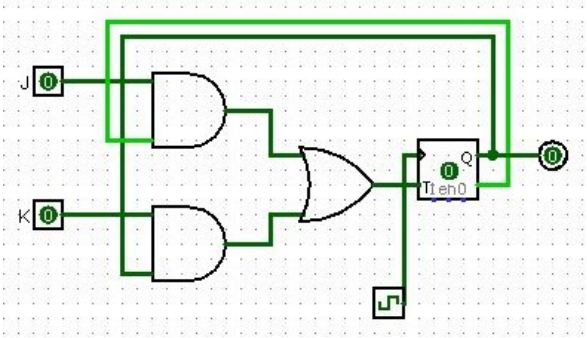
1.2.1 Zamiana przerzutnika T na JK

Bramka docelowa					Bramka bazowa
J	K	Q(t)	Q(t+1)		T
0	0	0	0	→	0
0	0	1	1	→	0

0	1	0	0	→	0
0	1	1	0	→	1
1	0	0	1	→	1
1	0	1	1	→	0
1	1	0	1	→	1
1	1	1	0	→	1

Funkcja po minimalizacji:

$$T = KQ + JQ'$$

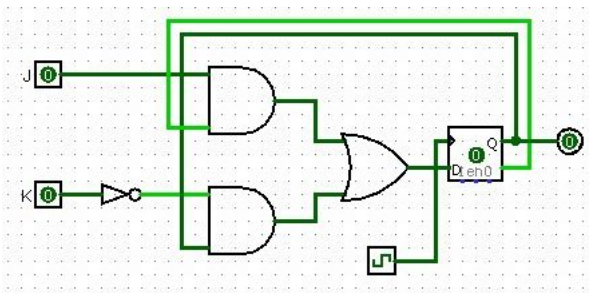


1.2.2 Zamiana przerzutnika D na JK

J	K	Q(t)	Q(t+1)		D
0	0	0	0	→	0
0	0	1	1	→	1
0	1	0	0	→	0
0	1	1	0	→	0
1	0	0	1	→	1
1	0	1	1	→	1
1	1	0	1	→	1
1	1	1	0	→	0

Funkcja po minimalizacji:

$$D = K'Q + JQ'$$

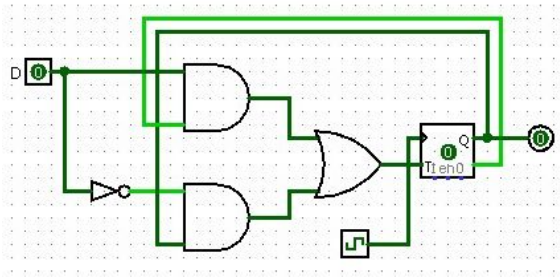


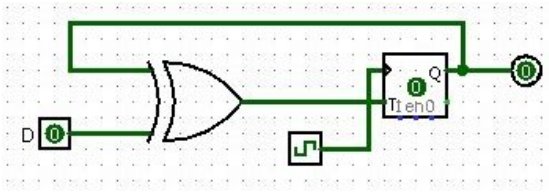
1.2.3 Zamiana przerzutnika T na D

D	Q(t)	Q(t+1)		T
0	0	0	→	0
0	1	0	→	1
1	0	1	→	1
1	1	1	→	0

Funkcja po minimalizacji:

$$T = D'Q + DQ'$$





Wykorzystanie bramki XOR zamiast dwóch bramek AND i jednej OR.

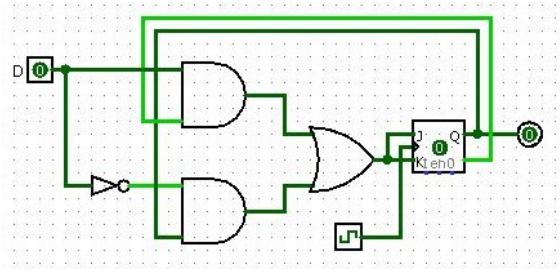
1.2.4 Zamiana przerzutnika JK na D

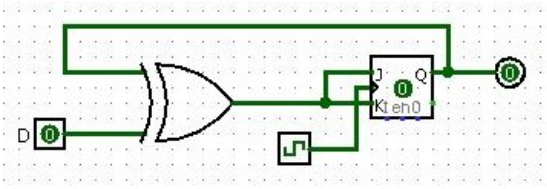
D	Q(t)	Q(t+1)		J	K
0	0	0	→	0	0
0	1	0	→	1	1
1	0	1	→	1	1
1	1	1	→	0	0

Funkcje po minimalizacji:

$$J = D'Q + DQ'$$

$$K = D'Q + DQ'$$



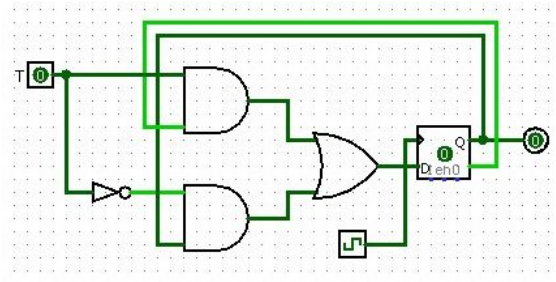


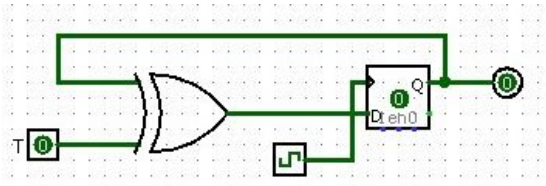
1.2.5 Zamiana przerzutnika D na T

T	Q(t)	Q(t+1)		D
0	0	0	→	0
0	1	1	→	1
1	0	1	→	1
1	1	0	→	0

Funkcja po minimalizacji:

$D = T'Q + TQ'$





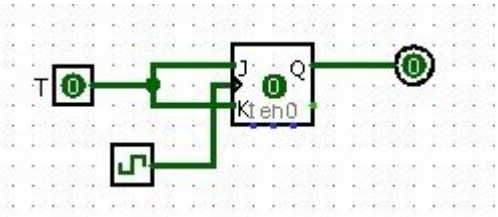
1.2.6 Zamiana przerzutnika JK na T

T	Q(t)	Q(t+1)		J	K
0	0	0	→	0	0
0	1	1	→	0	0
1	0	1	→	1	1
1	1	0	→	1	1

Funkcje po minimalizacji:

$J = T$

$K = T$



1.3 Automaty

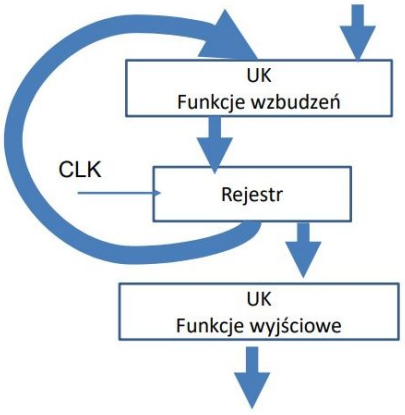
Generalnie rzecz biorąc to automaty przetwarzają sygnał wejściowy na wyjściowy ¯_(ツ)_/¯ i stosują do tego układy kombinacyjne oraz rejestry (pamięć, przerzutniki).

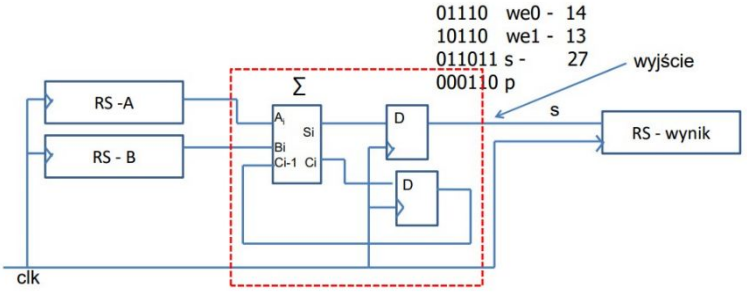
> Jest to układ sekwencyjny, posiadający przynajmniej jedną pętlę sprzężenia zwrotnego.

Wyjście automatu jest funkcją jego stanu wewnętrznego (Moore i Mealy) i sygnałów wejściowych (tylko Mealy).

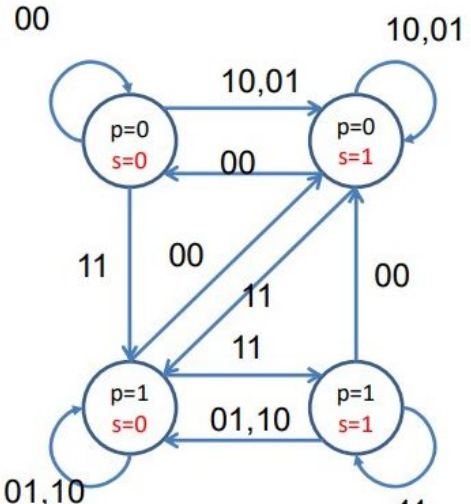
1.3.1 Automat Moore'a

Funkcje wyjściowe czerpią tylko z rejestrów.





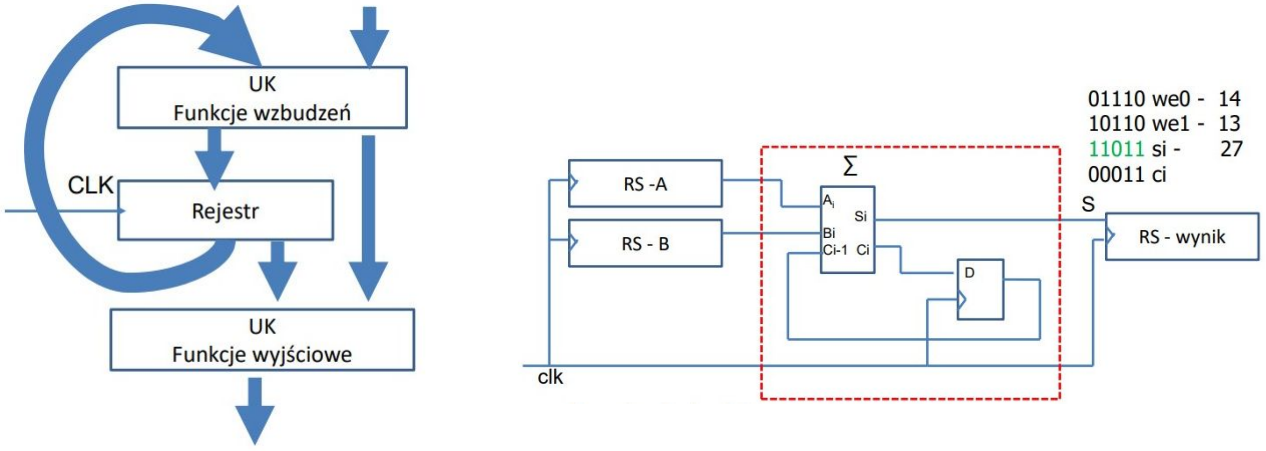
Na obrazku obok jest sumator szeregowy. Tylko część zaznaczona na czerwono jest tak de facto automatem Moore'a. Jest tam wejściowy układ kombinacyjny (sumator), przerzutnik D jako rejestr pamiętający przeniesienie oraz przerzutnik D pamiętający wynik (tu konkretnie sumowania jednego bitu). Wyjście jest pobierane tylko z rejestru (przerzutnika D). RS-A/RS-B to rejestry przechowujące liczby, które mają być zsumowane. RS-wynik przechowuje… wynik.



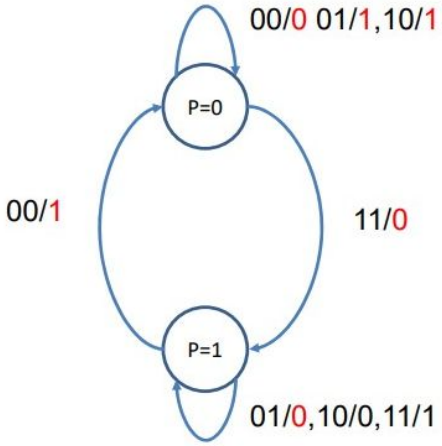
Graf automatu Moore'a w sumatorze szeregowym wygląda następująco. Istnieją 4 stany, w których może być automat. Zależą one od dwóch zmiennych: p – przeniesienie, s – suma. Z każdego stanu wychodzą trzy drogi zależne od bieżących stanów wejścia. „11" oznacza, że na wejściu są dwie jedynki, czyli musimy znaleźć taki stan, który będzie odpowiadał ich sumie oraz uwzględni przeniesienie ze stanu, w którym się znajdowaliśmy (jeśli takie było). Mylące jest to „10,01"/"01,10", bo to tak naprawdę to te same stany wejściowe, czyli jedna '1' i jedno '0'.

1.3.2 Automat Mealy'ego

W porównaniu do automatu Moore'a widać, że ten może już korzystać z funkcji wejściowych.



Jedyna różnica, która nastąpiła w porównaniu do automatu Moore'a to usunięcie przerzutnika D pamiętającego wynik sumowania dwóch bitów. W efekcie wynik zostanie otrzymany o jeden takt zegara szybciej, ale mogą pojawić się niejednoznaczne stany przejściowe w rejestrze wynikowym.

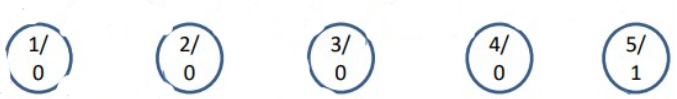


Ponieważ w tym konkretnym automacie Mealy'ego nie pamiętamy wyniku sumowania to musiał on zostać uwzględniony w przejściach (na czerwono), a liczba stanów zmniejszyła się przez to do dwóch.

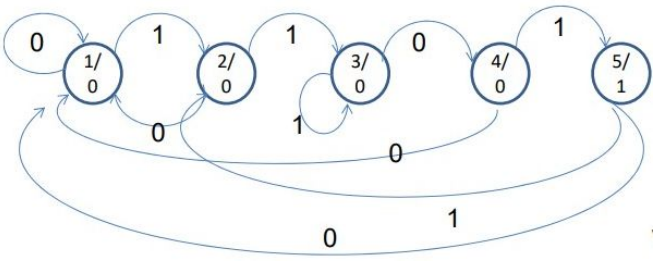
1.3.3 Tworzenie automatu (wykrywającego sekwencję bitów)

Pierwszym krokiem w tworzeniu automatu jest głębokie zastanowienie się, dlaczego to robimy. Można by przecież zamiast tego wypasać owce w Bieszczadach. Jednak, jeśli jesteśmy bardzo

zdeterminowani by zbudować automat, należy **rozpisać wszystkie jego stany**. Dla automatu Moore'a wykrywającego sekwencję 1101 mogą wyglądać następująco:

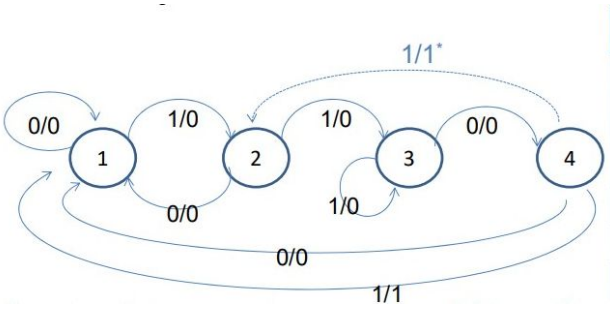


Nad kreską jest numer stanu, a pod kreską jest wartość wyjścia. W tym przypadku jest to bit potwierdzający, że wykryto żądaną sekwencję. Przejścia w tym automacie będą wyglądać następująco:



	Stan	następny
Stan bieżący	We=0	We=1
1	1	2
2	1	3
3	4	3
4	1	5
5	1	2

Na tej podstawie można stworzyć tablicę przejść w zależności od stanu wejścia (po prawej). Automat Mealy'ego wykrywający tę samą sekwencję bitów (1101) będzie wyglądał następująco. Zmiany wynikają z tego, że nie przechowujemy stanu wyjścia bezpośrednio w automacie. Jest ono zależne od przejść. Tym samym mamy o jeden stan mniej.

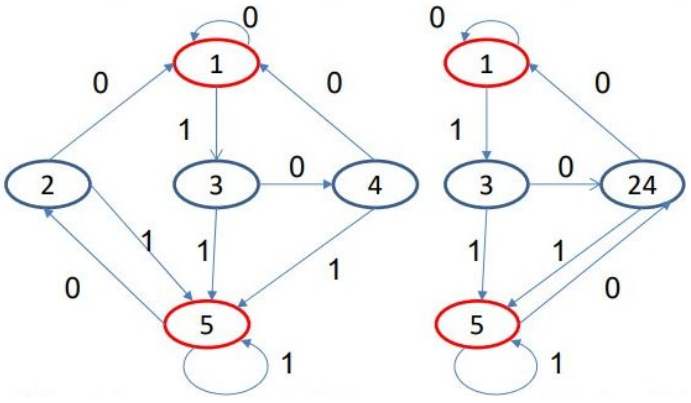


	stan	następny
Stan bieżący	We=0	We=1
1	1/0	2/0
2	1/0	3/0
3	4/0	3/0
4	1/0	1/1

1.3.4 Redukcja stanów automatu

Cześć stanów automatu może być sobie równoważna, więc ich ilość można zredukować. Aby stany można było uznać za równoważne, dla każdej kombinacji stanów wejściowych muszą zostać spełnione następujące warunki:

- Muszą dostarczać (posiadać) taką samą wartość wyjścia
- Muszą przenosić do tego samego stanu lub stanów równoważnych



Na rysunku stany 2 i 4 są sobie równoważne, ponieważ oba są niebieskie (tu kolor jest oznaczeniem stanu wyjścia) oraz w przypadku ‘0’ kierują do stanu 1, a w przypadku ‘1’ kierują do stanu 5.

1.3.5 Wyznaczenie stanów równoważnych dla automatu Moore’a

Wyznaczenie stanów równoważnych można przeprowadzić przy użyciu **tablicy trójkątnej**. Tworzymy sobie takową o rozmiarze $(n-1) \times (n-1)$, gdzie n to liczba stanów. W przypadku wierszy brakuje tego dla stanu 1, a w przypadku kolumn brakuje tej dla stanu n. Następnie postępujemy zgodnie z ogólnym schematem:

1. Jeśli dana para stanów ma różne wyjście to zaznacz w danej komórce X, czyli ‘brak równoważności’
2. Przeszukuj tabelę przejść w poszukiwaniu warunków jakie musiałyby zostać spełnione, by stany mogły być równoważne
3. Sprawdź, które warunki da się zrealizować. Pozostałe można skreślić.
4. Pozycje nieskreślone określają stany równoważne

Równoważność jest przechodnia w przypadku pełnej określoności tablicy przejść (czyli braku przejść do stanów dowolnych). Jeśli są docelowe stany dowolne to dla określenia równoważności - konieczne sprawdzenie warunków pokrycia i zamknięcia dla wyznaczenia klasy stanów równoważnych (rozszerzenie pojęcia pary stanów).

Przykład dla automatu Moore’a:

	stany		następne		
Stan aktualny	00	01	11	10	wyjscie
1	5	3	2	1	0
2	5	3	1	4	0
3	3	4	4	5	1
4	5	3	2	2	0
5	6	7	1	1	0
6	3	3	1	7	0
7	7	1	1	5	1

2	12 14					
3	X	x				
4	12	12 24	x			
5	56 37 12	56 37 14	x	56 37 12		
6	35 12 17	35 47	x	35 12 27	36 37 17	
7	x	x	37 14	x	x	x
	1	2	3	4	5	6

X oznaczono wszystkie pary stanów, które mają różne wyjścia. Dla pozostałych wpisano pary stanów, które musiałyby być równoważne by dana para też była równoważna. Na czerowono oznaczono te komróki, dla których spełenienie warunków nie jest możliwe.

2	12 14					
3	X	x				
4	12	12 24	x			
5	56 37 12	56 37 14	x	56 37 12		
6	35 12 17	35 47	x	35 12 27	36 37 17	
7	x	x	37 14	x	x	x
	1	2	3	4	5	6

Stan akt	00	01	11	10	wyjscie
124	5	37	124	124	0
37	37	124	124	5	1
5	6	37	124	124	0
6	37	37	124	37	0

Kolejność wykreślania:
61,62,64(35), 56(36),51,52,54(56)

Ostateczne zestawienie stanów pokazuje tabelka po prawej.

1.3.6 Wyznaczenie stanów równoważnych dla automatu Mealy'ego

Podobnie jak dla Moore'a. Konieczne jest jednak uwzględnienie zależność stanów wyjściowych od wejścia.

Automat Mealego wykrywania sekwencji (poprzedni slajd) – próba minimalizacji stanów

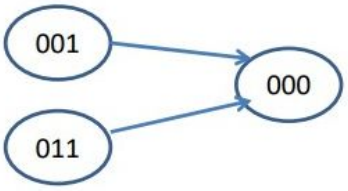
	23		
3	14 23	14	
4	x	x	x
	1	2	3

Stan bieżący	We=0	We=1
1	1/0	2/0
2	1/0	3/0
3	4/0	3/0
4	1/0	1/1

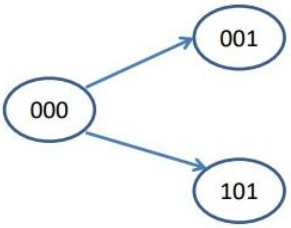
Stany 1 i 4 nie mogą być równoważne gdyż mają różne wartości wyjść dla We=1
W konsekwencji nie równoważna jest są również para stanów 2 i 3.

1.3.7 Kodowanie stanów w automacie

Zasada 1. – *Nigdy nie wspominaj o podziemnym kręgu, a poza tym…* Stanom, którem mają taki sam stan następny należy przyporządkować słowa kodowe różniące się wartością jednego bitu.



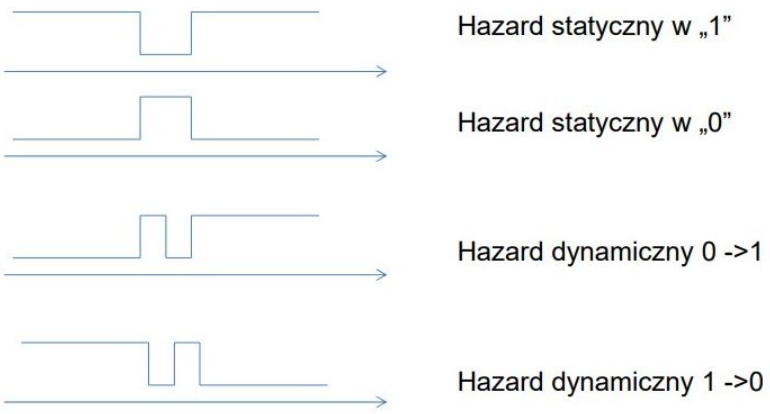
Zasada 2. – Stanom, które mają ten sam stan poprzedzający należy przyporządkować słowa kodowe różniące się wartością tylko jednego bitu.



Po co są te zasady? Umiejętne ich zastosowanie pozwoli na uzyskanie najprostszych funkcji wzbudzeń wymuszających odpowiednie stany przerzutników. W sytuacji, gdy każdy stan ma tylko jeden stan następny warto zastosować dla kolejnych stanów słowa kodowe różniące się na jednej pozycji np. wg kodu Greya.

1.3.8 Hazardy w układach cyfrowych

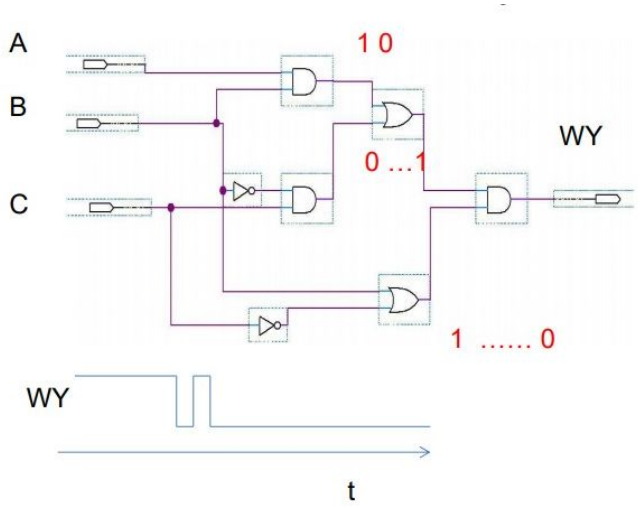
Hazard jak w prawdzimy życiu może mieć różne postacie.



- Hazard statyczny w „1” – chwilowa zmiana stanu wysokiego na niski
- Hazard statyczny w „0” – chwilowa zmiana stanu niskiego na wysoki
- Hazard dynamiczny 0->1 – pojawienie się fluktuacji podczas zmiany stanów
- Hazard dynamiczny 1->0 – pojawienie się fluktuacji podczas zmiany stanów

C\BA	00	01	11	10
0	0	0	1	0
1	1	1	1	0

Dla powyższej tablicy przełączenie ze stanu (ABC) 111 na 101 może spowodować hazard statyczny w „1”. Jeśli początkowo $Q = AB + B'C$ to dodanie AC, by ostatecznie powstało $Q = AB + B'C + AC$ pozawala na uniknięcie hazardu.



Hazard dynamiczny może się pojawić na przykład w takim układzie. Jego wykrycie jest możliwe po analizie, ile bramek musi pokonać zmieniający się sygnał i czy opóźnienie jednego z nich nie wpłynie na końcowy wynik. Tutaj zasugerowano, że dolna bramka OR się opóźni. Różnie może być, ale generalnie im więcej bramek po drodze tym większe opóźnienie między sygnałami.

1.4 Ogólny schemat tworzenia automatu

Krok 1. Na podstawie algorytmu rozrysuj graf przejść

Krok 2. Sprawdź, czy graf jest w postaci minimalnej i w razie potrzeby zminimalizuj go

Krok 3. Zakoduj stany, jeśli nie zostało to zrobione wcześniej (tj. zamień A i inne kwiatki na przykład na kod Gray'a)

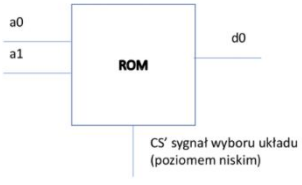
Krok 4. Stwórz tablicę przejść, która posłuży do wyznaczenia funkcji wzbudzeń oraz tablicę wyjść, która posłuży do wyznaczenia funkcji wyjścia

Krok 5. Zminimalizuj obie funkcje i zaprojektuj dla nich układy kombinacyjne

Krok 6. Połącz całość z przerzutnikami i voilà 👩🍳

1.5 Wykorzystanie pamięci ROM dotworzenia automatów

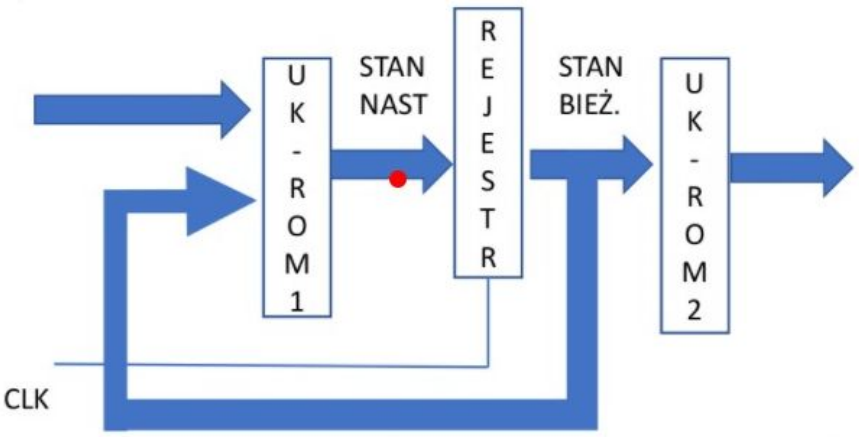
Pamięć ROM (pamięć statyczna) w swojej wspaniałości pozwala na realizację dowolnej funkcji logicznej (funkcji Boolowskiej). Jeśli ilość wejść adresowych to n, to ilość komórek pamięci to 2^n. Ilość wyjść określa długość słów przechowywanych w komórkach.



adres	dane
00	0
01	0
10	0
11	1

Powyżej przykład pamięci ROM zastępującej bramkę AND.

Dzięki niebywałym zdolnościom, bramka ROM może zastępować całe układy kombinacyjne (na przykład takie jak stosowane w automacie Moore'a czy Mealy'ego). Wystarczy odzwierciedlić odpowiednie funkcje w ich rejestrze by uzyskać taki efekt.



Powyżej implementacja układów kombinacyjnych w automacie Moore'a za pomocą dwóch pamięci ROM. Dla następującej tablicy przejść oraz tablicy wyjść:

	0	1
00	00	01
01	01	11
11	01	11

TABLICA PRZEJŚĆ AUTOMATU

stan	wyjście
00	0
01	1
11	0

TABLICA WYJŚĆ

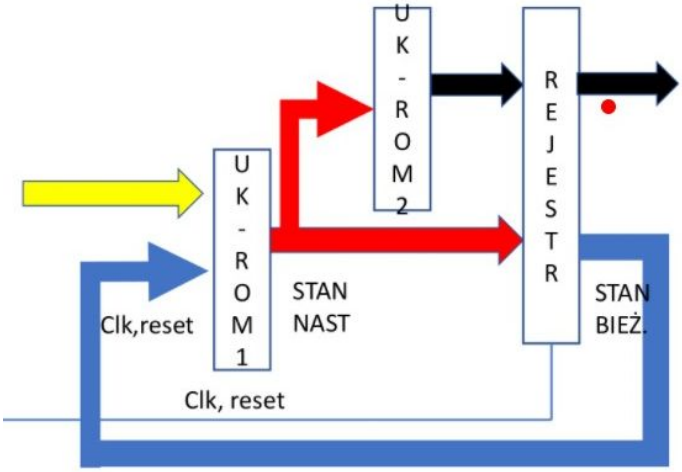
W pamięciach ROM mamy następującą zawartość:

ZAWARTOŚĆ ROM1	
ADRES (STAN,WEJŚCIE)	DANE (STAN*)
00 0	00
00 1	01
01 0	01
01 1	11
10 0	DC
10 1	DC
11 0	01
11 1	11

ZAWARTOŚĆ ROM2	
STAN	WYJŚCIE
00	0
01	1
10	DC
11	0

Sumaryczny rozmiar wykorzystanych pamięci to 20 bitów: 8 słów po 2 bity dla ROM1 i 4 słowa po 1 bit dla ROM2.

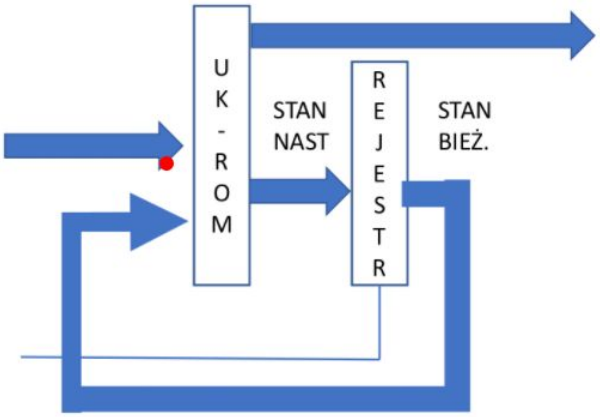
Uwaga! Przedstawiony powyżej przykład jest niestabilny. Pamięć ROM potrzebuje czasu na odpowiedź, a w międzyczasie na wyjściu mogą się pojawić jakieś stany pośrednie. Uniknąć tego



Schemat automatu Moore'a ze stabilnymi sygnałami wyjść
ROM1(WEJ,STAN)=STAN+
ROM2(STAN+)=WYJŚCIE+
Zboczem CLK → WYJŚCIE<=WYJŚCIE+

problemu można przez zaprojektowanie automatu z rejestrem przechowującym wyjście i synchronizującym je. Poniżej przykład:

Podobnie został zaprojektowany automat Mealy'ego. Tu



	0	1
0	0	1
1	1	1

Kodowana tablica przejść

stan	Wyjście (0)	Wyjście (1)
0	0	1
1	1	0

ZAWARTOŚĆ ROM	
ADRES - STAN,WEJŚCIE	DANE WYJŚCIE,STAN*
0 0	0 0
0 1	1 1
1 0	1 1
1 1	0 1

wystarczy rejestr jednobitowy – tylko na stan.

1.6 Przykłady automatów

//Tododododo

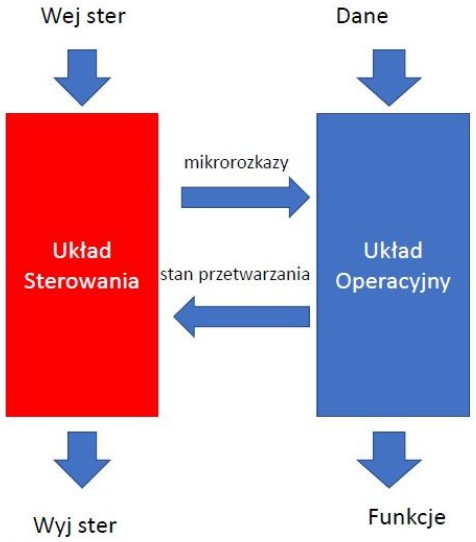
2 Synteza wyższego poziomu

W syntezie wyższego poziomu (RTL) układów cyfrowych (UC) jesteśmy w stanie wyróżnić dwa układy: układ operacyjny, układ sterujący. Oba znakomicie się uzupełniają, ale leniwy informatyk mógłby zapytać „Po co tworzyć i programować robota do sterowania kolejnym robotem?". W istocie tak właśnie można rozumieć te dwie części układu cyfrowego. Układ operacyjny jest

odpowiedzialny za wykonywanie operacji na danych, a układ sterujący odpowiada za sterowanie tym pierwszym.

W skrócie:

- Układ operacyjny (lub ścieżka danych) pobiera dane i je przetwarza. Może na przykład porównywać liczby, coś mnożyć, dodawać, cokolwiek… Istotne jest to, że poszczególne operacje realizuje dopiero po otrzymaniu mikrorozkazu od układu sterującego. Z wdzięczności układ operacyjny wysyła temu sterującemu informacje o stanie przetwarzania, czyli na której operacji obecnie jest.
- Układ sterujący pobiera dane z wejścia sterującego z zewnątrz (może być to chociażby sygnał START) i za pomocą mikrorozkazów nakłania układ operacyjny do wykonania swojej pracy. W zamian otrzymuje informacje o bieżącym stanie operacji. Warto zauważyć, że układ sterujący jest automatem (np. Mealy'ego) lub układem mikroprogramowalnym.



2.1 Podstawowe bloki i operacje [układ operacyjny]

W układzie operacyjnym jest kilka typów bloków strukturalnych:

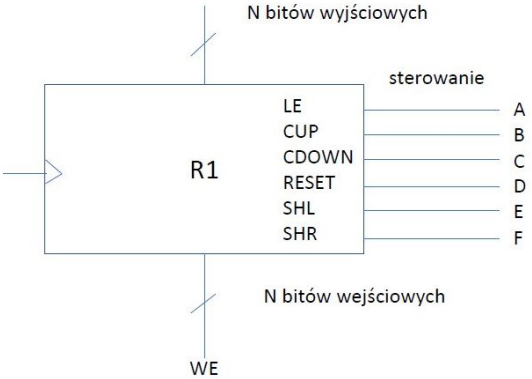
- **bloki funkcjonalne** służące do przechowywania zmiennych
- **bloki operacyjne** służące do wykonywania operacji występujących w algorytmie
- **bloki funkcjonalne** służące do przesyłania danych między rejestrami i blokami operacyjnym

I tak na przykład można wyróżnić rejestr (w tym przypadku licznik w NKB).

Ma on wejście zegarowe (CLK), wejścia na N bitów oraz wyjścia na N bitów. Do tego został wyposażony w wejścia sterujące:

Wpis równoległy:
A: R1 ← WE

Zwiększenie wartości R1:
B: R1 ← R1+1



Zmniejszenie wartości R1:
C: R1 ⟵ R1-1

Zerowanie:
D: R1 ⟵ 0

Mnożenie R1*2:
E: R1 ⟵ 0

DZIELENIE R1/2:
F: R1 ⟵ 0

Operacje wykonywane tylko w obrębie jednego rejestru (takie jak te powyżej) nazywamy **operacjami elementarnymi**. Istnieje jednak coś bardziej skomplikowanego, a mianowicie:

Mikrooperacje arytmetyczne:

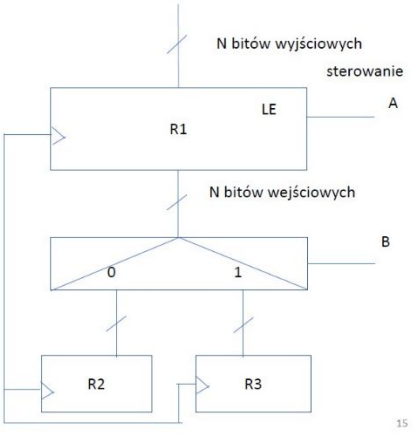
– R0 ⟵ R1+R2 – dodawanie
– R2 ⟵ R2' – negacja (uzupełnienie do jeden)
– R2 ⟵ R2' + 1 – uzupełnienie do 2
– R0 ⟵ R1+R2'+1 – odejmowanie
– R1 ⟵ R1+1 – inkrementacja zawartości R1
– R1 ⟵ R1-1 – dekrementacja zawartości R1

Mikrooperacje logiczne – operacje bitowe

– R1 ⟵ R1 OR 1 – ustawianie zawartości rejestru (wpis 1)
– R1 ⟵ R1 AND 0 – zerowanie zawartości rejestru
– R1 ⟵ R1 Exor R2 – negacja bitów rej R1 sterowana zawartością rejestru R2

Mikrooperacje przesunięcia

– R0 ⟵ sr R0 – przesunięcie w prawo (dzielenie przez 2)
- R1 ⟵ sl R2 – przesunięcie w lewo (mnożenie przez 2)



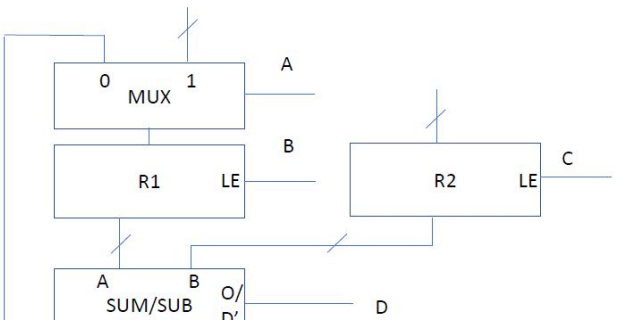
W ten sposób możemy stworzyć różnorodne układy realizujące różnorodne zadania. Przykładowo ten po prawej zapisuje wartość z wejścia do rejestru R1, a później poprzez multiplekser zapisuje je w R2 lub R3. Ważne są sygnały sterujące A (pozwolenie na wczytanie) oraz B (wybór wyjścia).

AB: R1 ⟵ R3
AB': R1 ⟵ R2

Poniżej natomiast układ, który może **wielokrotnie dodawać/odejmować dwie liczby**.

Przygotowanie argumentów (równoczesne):
AB: R1 ⟵ WEJ1, C: R2 ⟵WEJ2

Dodawanie:
A'BD' : R1 ⟵ R1 + R2

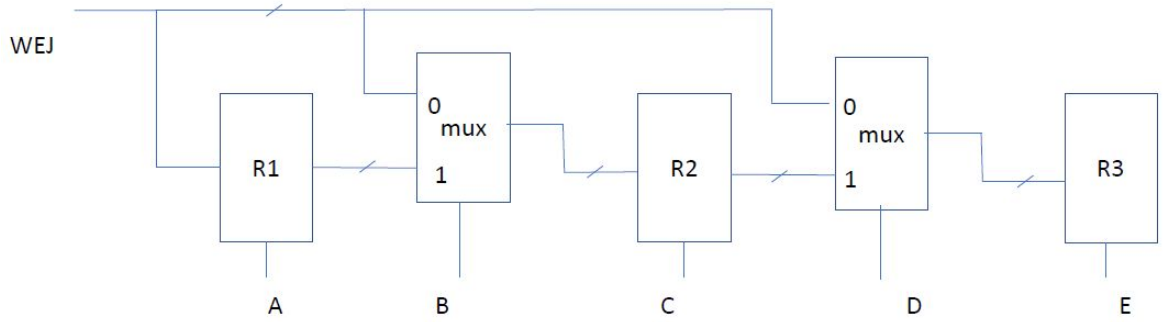


Odejmowanie:
A’BD : R1 ⟵ R1 - R2

Pamiętanie:
B’: R1 ⟵ R1

Sidenote
Zwróć uwagę, że każda operacja ma schemat sygnał_sterujący : wykonywana_operacja. Warto się przyjrzeć tym sygnałom, ponieważ są one łącznikiem między układem operacyjnym (który teraz jest omawiany) a układem sterującym.

Poniżej kolejny przykład, ale tym razem pozwalający na **ładowanie wartości do rejestrów** oraz ich przekazywanie.



Równoczesne załadowanie wejścia:
A: R1⟵ WEJ, B’C R2 ⟵WEJ, D’E R3⟵WEJ

Równoczesne przepisanie między rejestrami:
BC: R2 ⟵R1, DE: R3 ⟵R2

Sekwencyjne przepisanie między rejestrami:
DE: R3 ⟵R2
BC: R2 ⟵R1

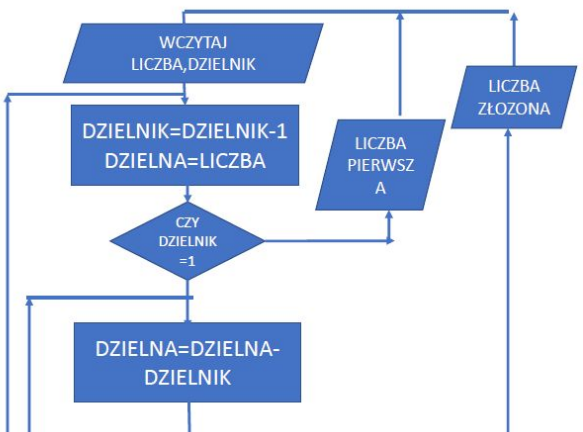
Ten może zostać rozwinięty do **znajdowania trzech największych liczb dodatnich**:

ZERUJ R1,R2,R3
JEŻELI WEJ>R1 TO A: R1 ⟵ WEJ, BC: R2 ⟵R1, DE: R3 ⟵R2
JEŻELI (WEJ>R1)’ * (WEJ>R2) TO B’C: R2 ⟵WEJ, DE: R3 ⟵R2
JEŻELI (WEJ>R1)’ * (WEJ>R2)’ * (WEJ>R1) TO D’E: R3 ⟵WEJ

2.1.1 Jak zaprojektować układ operacyjny?

Zróbmy to na przykładzie. Stwórzmy układ, który będzie sprawdzał, czy liczba jest liczbą pierwszą. Algorytm jest następujący:

1. Pobierz testowaną liczbę i zapisz ją jako dzielną i dzielnik
2. dzielnik = dzielnik – 1
3. Jeśli dzielnik == 1 to koniec
4. Wykonuj dzielenie przez odejmowanie aż:



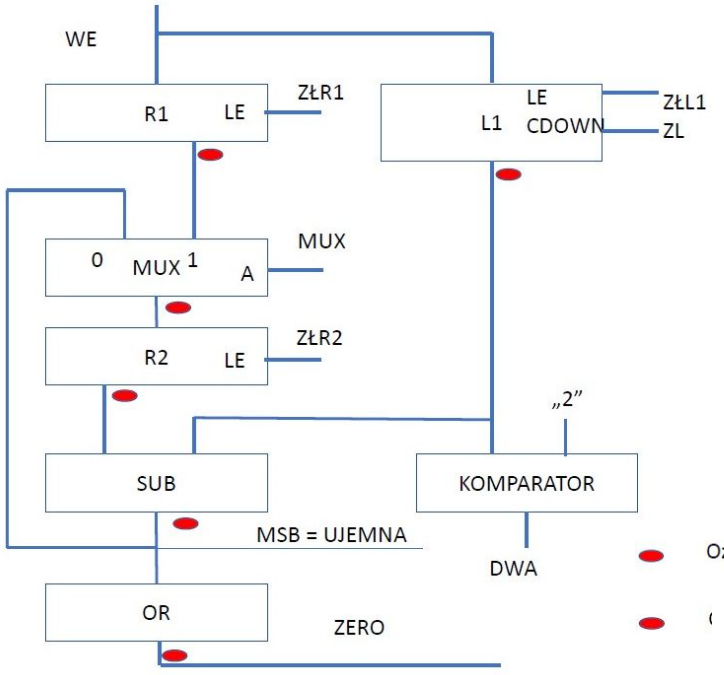
5. Jeśli dzielna == 0 to koniec i liczba złożona
6. Jeśli dzielna < 0 to powróć do 2

Działanie prezentuje następujący schemat blokowy:

Operacje przedstawione na schemacie można zrealizować następującymi komponentami:

ZADANIA	UKŁAD WYKONAWCZY
odejmowanie 2 liczb	sumator
odejmowanie 1	Licznik liczący w dół w NKB
test wektor=0	Bramka OR
Test wektor=STAŁA WARTOŚĆ	Komparator
test wektor(NKB) < 0	Test najstarszego bitu
nadawanie zmiennej wartości z różnych źródeł	multiplekser

Tym samym powstaje nam układ, który można też opisać następującymi operacjami:



ŁADUJ LICZBĘ TESTOWANĄ:
ZŁR1: R1⟵WE

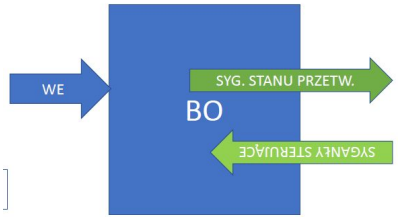
PRZYGOTUJ DZIELNIK:
ZL:L1⟵WE

ODEJMIJ:
L1 ⟵L1-1

ZAŁADUJ DZIELNĄ:
MUX * ZŁR2: R2 ⟵R1

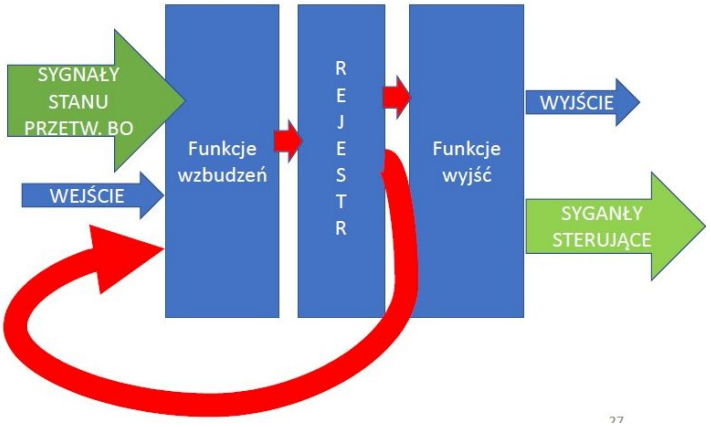
ODEJMIJ:
MUX’*ZŁR2: R2 ⟵R2-L1

Zrealizowany w ten sposób układ jest układem operacyjnym. Posiada wejście na dane, realizuje jakąś funkcję i posiada wejścia na sygnały sterujące oraz wyjścia informacyjne dla układu sterującego.



2.2 Układ sterujący

Tak jak zostało to już pokazane, bloki (układy) współpracują ze sobą. Blok sterujący jest automatem (lub układem mikroprogramowalnym), którego stany odpowiadają poszczególnym etapom pracy układu operacyjnego. Co chyba już oczywiste, układ sterujący ma strukturę automatu, czyli rejestry i układy kombinacyjne:



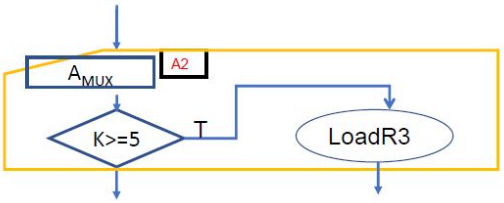
Do zamodelowania automatu można wykorzystać diagram ASM.

2.3 Diagram ASM

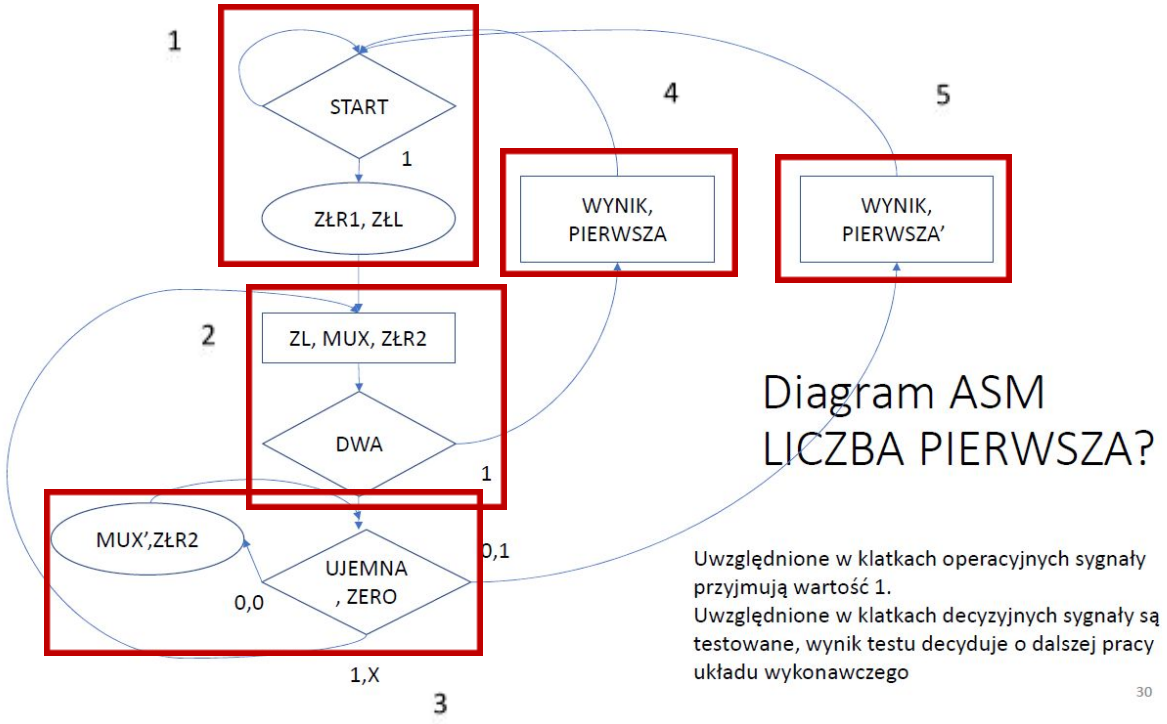
Diagram ASM to kolejny obok grafu i tablicy przejść sposób opisu działania automatu. Składa się on z bloków, a te składają się z klatek:

- **Klatka operacyjna** przedstawiana jest jako prostokąt i reprezentuje stan automatu, nazwę stanu umieszcza się obok prostokąta. Wewnątrz klatki umieszcza się akcje przedstawiające przypisania wartości do sygnałów, jakie powinny zostać wykonane w momencie wejścia automatu do tego stanu. Odpowiadają one wyjściom Moore'a automatu.
- **Klatki decyzyjne** sprawdzają warunki wejściowe (stan sygnału) w celu określenia ścieżki przejścia automatu do następnego stanu. Możliwe jest powiązanie wielu klatek decyzyjnych w jedną dla opisania złożonych warunków przejść automatu.
- **Warunkowe klatki** wyjść także opisują przypisania do sygnałów. Umieszczane są one na ścieżkach wyjściowych ze stanu, dlatego reprezentują wyjścia Mealy'ego.

Przykład:
Poniżej blok pewnego automatu odpowiadający stanowi A2.



- **Klatka operacyjna** specyfikuje sygnały aktywowane bezwarunkowo przy wejściu do stanu np, AMUX wymuszenie stanu 1, AMUX' wymuszenie stanu 0
- **Klatka decyzyjna** dla specyfikacji testowania warunku/warunków przejścia do kolejnego stanu i warunków generacji sterowania warunkowego np. w zależności od wartości sygnału K>=5 generacja sygnału LoadR3 (lub nie) oraz przejście do potencjalnie różnych stanów.
- **Klatka warunkowa** określa sterowania warunkowe w bieżącym stanie generowane jako następstwo spełnienia warunku (wyjścia automatu Mealy'ego).
- Klatka warunkowa jest opcjonalna.
- Para - klatki warunkowa i decyzyjna jest opcjonalne



Powyżej zaprezentowano przykład diagramu ASM dla układu sterującego układem operacyjnym, który sprawdza czy liczba jest pierwsza. W czerwonych ramkach oznaczono klatki odpowiadające stanom automatu. Uproszczona tablica przejść dla tego automatu:

DS1: S1 START' + S4 + S5
DS2: S1 START + S3 UJEMNA
DS3: S2 DWA' + S3 UJEMNA' ZERO'
DS4: S2 DWA
DS5: S3 UJEMNA' ZERO

Dla układu można też wyznaczyć funkcje wyjść (wyjść sterujących):

ZŁR1 =S1 START
ZŁL = S1 START
ZL = S2
MUX = S2
ZŁR2= S2 + S3 ZERO' UJEMNA'
WYNIK = S4 + S5
PIERWSZA = S4

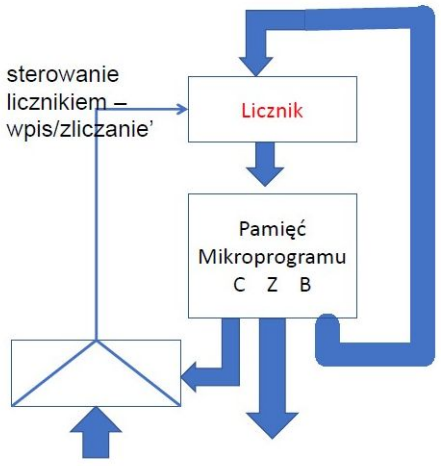
Ponieważ takie sygnały jak ZŁR1 i ZŁL oraz ZL i MUX mają takie same funkcje to można je zredukować. Możemy przyjąć nazwy: SYG1 dla ZŁR1 i ZŁL oraz SYG2 dla ZL i MUX. Tym samym mamy 4 sygnały stanu: START, DWA, UJEMNA, ZERO. Ponadto mamy 5 sygnałów sterujących: SYG1, SYG2, ZŁR2, WYNIK, PIERWSZA.

Tak prezentuje się tablica przejść i sygnałów wyjściowych dla tego automatu.

	Stan następny warunkowy				Wyjścia warunkowe				
Stan aktualny	START	DWA,	UJEMNA,	ZERO	SYG1	SYG2	ZŁR2	WYNIK	PIERWSZA
S1 (INIT)	S2/S1				1/START				
S2 (DZIELNIK)		S4/S3				1	1		
S3 (ODEJMIJ)			S2/S3	S5			1/ UJEMNA'ZERO'		
S4 (PIERWSZA)	S1	S1	S1	S1				1	1
S5 (ZŁOŻONA)	S1	S1	S1	S1				1	0

2.4 Układ mikroprogramowalny

Zamiast automatu jako blok sterujący można wykorzystać układ mikroprogramowalny. Jest to po prostu pamięć, która przechowuje mikrokod oraz licznik, który wskazuje na mikrorozkaz do odczytu z pamięci. Najprostsza wersja wygląda tak:



Struktura udostępnia rozkaz:

- A_i: $Z=Z_{Ai}$, if x_c then $A^+=A_j$ else $A^+= A_{i+1}$

Warianty rozkazu:

Warunek zawsze spełniony (x_c=1):

- A_i: $Z=Z_{Ai}$, if 1 then $A^+=A_j$ else $A^+= A_{i+1}$
  Skok bezwarunkowy
- Sterowanie warunkowe Zc(x_c) możliwe dzięki użyciu 2 rozkazów:

A_0: $Z=Z_{A0}$, if x_c' then $A^+=A_2$ else $A^+= A_1$

A_1: $Z=Z_c$, if 1 then $A^+=A_2$ else $A^+= A_2$

„Z” w tym rozkazie to sygnał sterujący, który ma zostać podany na wyjście. „A” to numer mikrorozkazu. Tak na przykład będzie wyglądała implementacja dla układu „czy liczba pierwsza?”. Mikrokod odpowiada tabelce przejść umieszczonej wcześniej.

A_0(S1): Z=ø, if START' then $A^+=A_0$ else $A^+= A_1$
A_1: Z=SYG1, if 1 then $A^+=A_2$ else $A^+= A_{i+1}$
A_2 (S2): Z=SYG2,ZŁR2, if DWA then $A^+=A_{S4}$ else $A^+= A_3$
A_3 (S3) : Z=ø, if ZERO then $A^+=A_{S5}$ else $A^+= A_4$
A_4: Z=ø, if UJEMNA then $A^+=A_{S2}$ else $A^+= A_5$
A_5: Z=ZŁR2, if 1 then $A^+=A_3$ else $A^+= A_{i+1}$
A_6 (S4): : Z=WYNIK, PIERWSZA, if 1 then $A^+=A_{S1}$ else $A^+= A_{i+1}$
A_7 (S5): : Z=WYNIK, PIERWSZA', if 1 then $A^+=A_{S1}$ else $A^+= A_{i+1}$

Rozkazy, warunki i skoki możemy zakodować.

Warunki skoku:

WARUNEK	KOD WARUNKU
START'	000
1	001
DWA	010
ZERO	011
UJEMNA	100

Mamy 5 różnych warunków, więc można je elegancko zapisać w NKB na 3 bitach. Można by myśleć o kodowaniu (np. przez dekoder), ale tu nie będzie z tego profitu. Podobnie robimy z wektorem sterowania. Mamy 6 różnych wektorów sterowania, więc można by je upchnąć na 3 bitach, ale to by wymagało kodowania. Tutaj te wektory zostaną zapisane na 5 bitach, bo w wektorach korzystamy z 5 sygnałów.

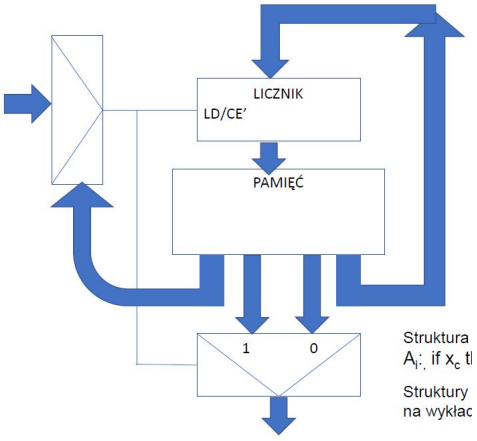
Adres pamięci	Bity słowa rozkazowego			
	Sterowanie	Kod warunku	Adres skoku	Słowo Pamieci (HEX)
000	00000	000	000	000
001	10000	001	010	40A
010	01100	010	110	316
011	00000	011	111	01F
100	00000	100	010	022
101	00100	001	011	10B
110	00011	001	000	0C8
111	00010	001	000	088

Po prawej stronie zawartość gotowa to wprowadzenia do pamięci układu. Tylko słowa w postaci heksadecymalnej zostaną do niej zapisane. Warto jednak wiedzieć, jak są tworzone. Mamy ustawione w kolejności kody sterowania, warunku i adresu skoku. Zbijamy bity w grupy po 4 i tak tworzymy słowo heksadecymalne. Teraz też widać, **ile pamięci potrzebuje ten układ**. Mamy 8 słów po 11 bitów, czyli będzie to 88 bitów => 11 bajtów [*do sprawdzenia*].

Warto zauważyć, że w przedstawionym układzie warunki są realizowane przez co najmniej 2 rozkazy. Takiego problemu (?) nie ma w bardziej zaawansowanym układzie mikroprogramowalnym. Poprzez dodatkową strukturę (tu multiplekser) może on realizować operacje warunkowe w jednym rozkazie:

Struktura udostępnia rozkaz:
A_i:, if x_c then $Z=Z_{xc}$;$A^+=A_j$ else $Z=Z_{xc'}$;$A^+= A_{i+1}$

Różnica jest również w ilości zajmowanej pamięci przez taki rozkaz, ponieważ mamy tutaj do przechowania dodatkowy kod warunku. Poniżej rzeczona struktura.



2.5 Implementacja pamięci w układach cyfrowych

2.6 Przykłady układów cyfrowych

3 Pamięci półprzewodnikowe

Dla pamięci można wyszczególnić następujące parametry:

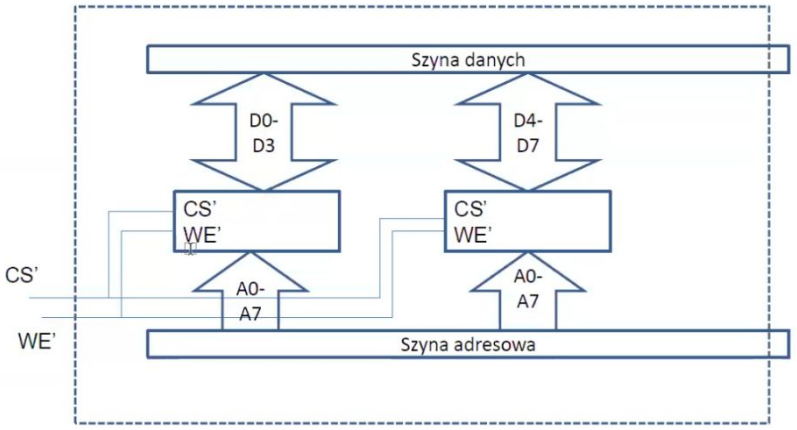
- Liczba bajtów
- Liczba słów (ciąg informacji zakodowanych pod danym adresem)
- Liczba linii adresowych wynikająca z kodowania binarnego adresów słów
- Liczba słów w zależności od liczby linii adresowych (LLA) to 2^{LLA}
- K – kilo (1024), M – mega (1024*1024=1048576)

3.1 Łączenie modułów pamięci w większe struktury, zwiększanie długości słów lub ich liczby

Zwiększanie pojemności pamięci jest możliwe przez łączenie ich w większe struktury.

3.1.1 Dłuższe słowo

Jeśli mamy pamięć, która przechowuje słowa o długości 4 bitów, a chcemy mieć słowa 8 bitowe, możemy dołożyć drugi taki sam moduł. Wtedy jeden będzie przechowywał pierwsze 4 bity słowa, a drugi ostatnie 4 bity słowa. Liczba adresów pozostaje bez zmian. Konieczne jest tylko odpowiednie wpięcie drugiego modułu do tej samej szyny adresowej i tej samej szyny danych co pierwszy. Podobnie z sygnałem CS' i WE'.

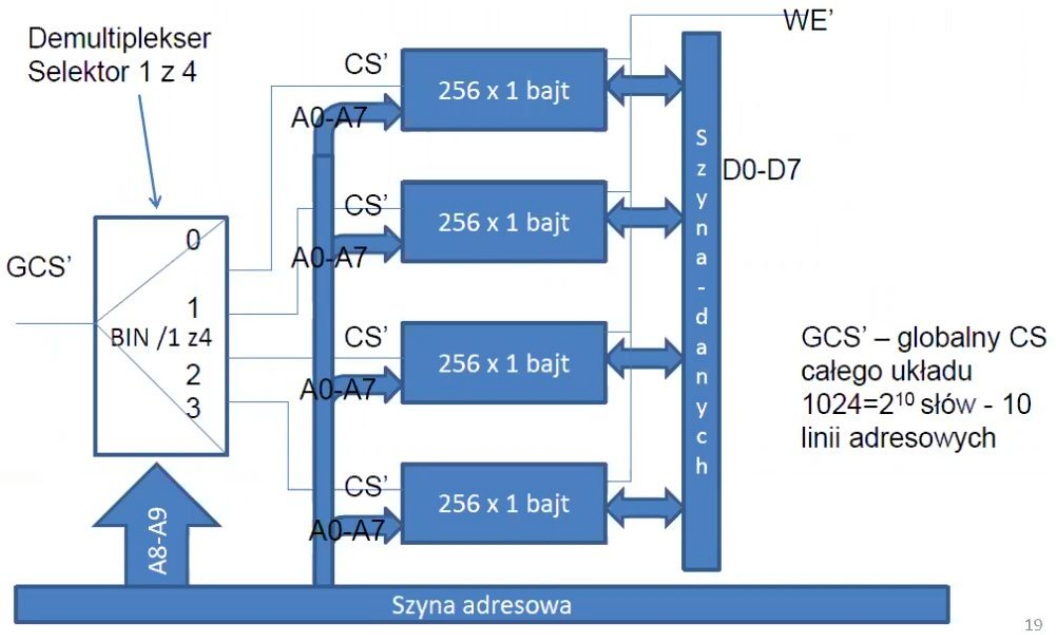


Moduł 2x 256 po 4 bity daje 256 X 1 bajt
256= 2^8 słow adres 8 bitowy

18

3.1.2 Zwiększanie ilości słów w pamięci

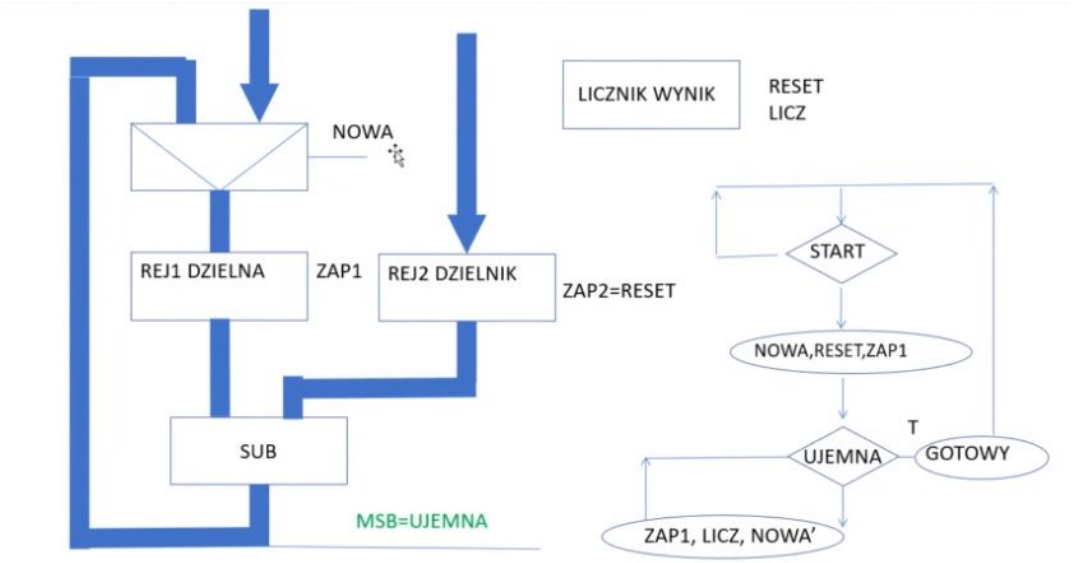
Zwiększenie ilości słów bez zmieniania ich długości również wymaga dołożenia kolejnych modułów. Istotne jest to, że zmianie ulegnie również przestrzeń adresowa, przez co konieczne będzie dodanie nowych linii adresowych. Warto wykorzystać np. demultiplekser do określania, z którego modułu ma zostać wybrane słowo.



19

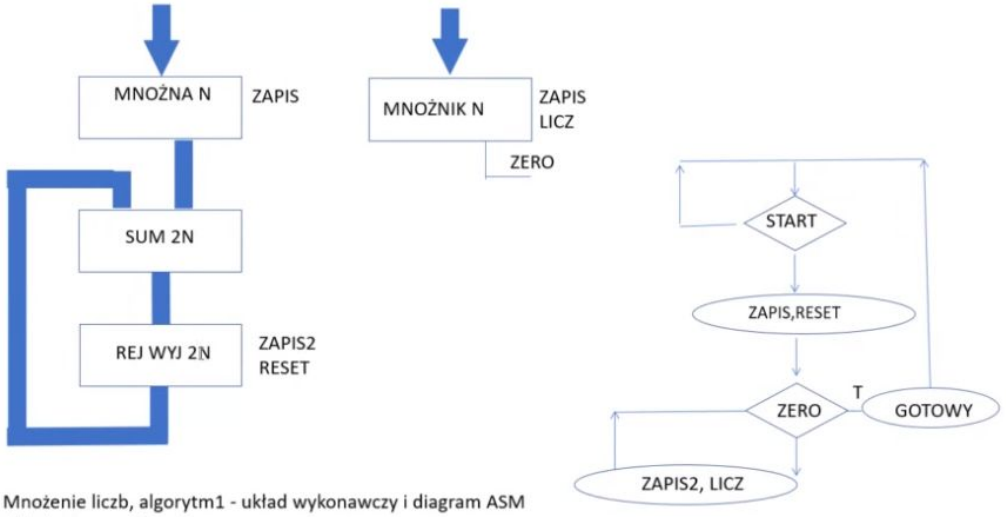
4 Przykłady układów

4.1 Dzielenie dwóch liczb w NKB



Dzielenie liczb - układ wykonawczy i diagram ASM układu sterowania

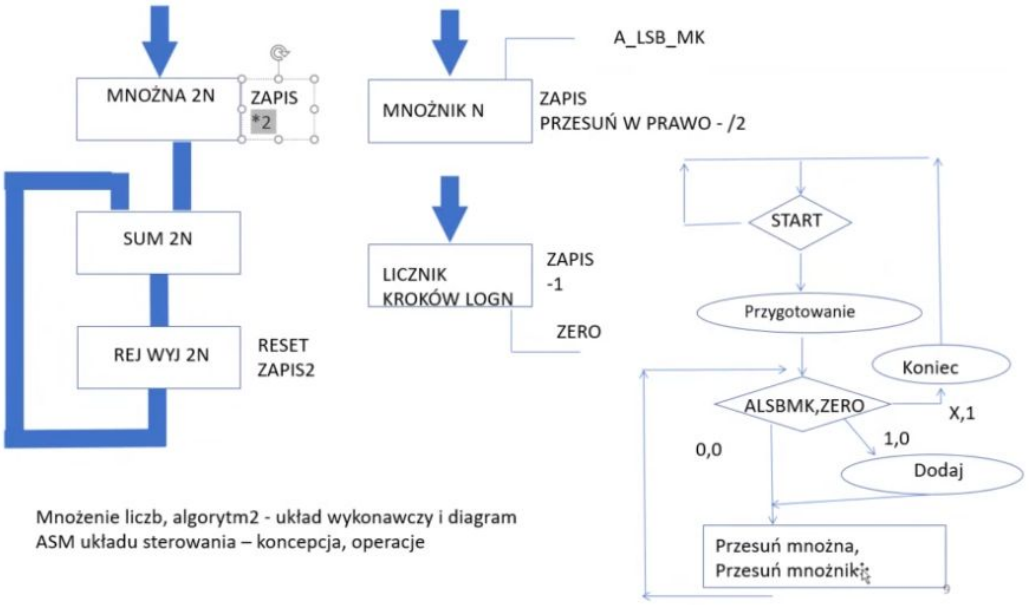
4.2 Mnożenie dwóch liczb w NKB



Mnożenie liczb, algorytm1 - układ wykonawczy i diagram ASM układu sterowania

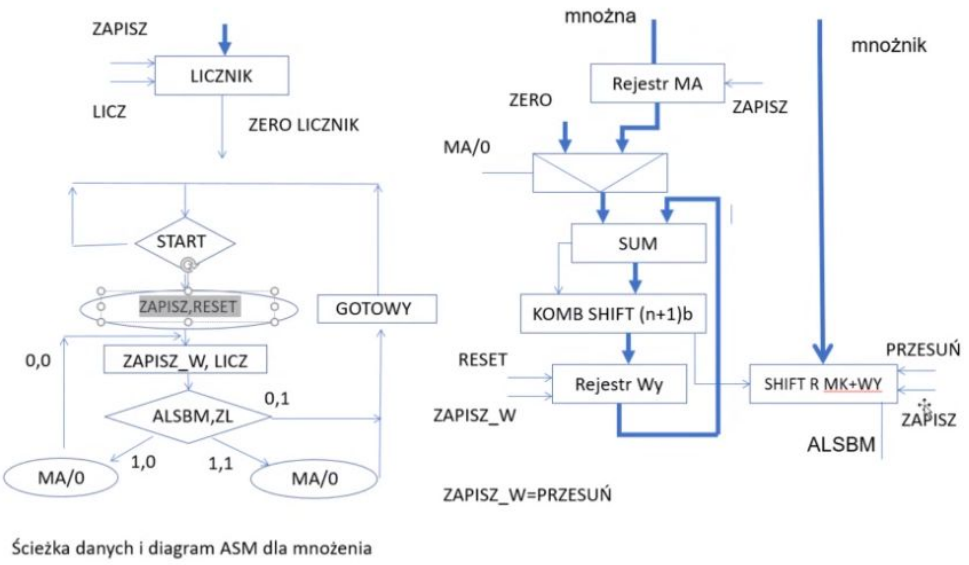
6

Alternatywna wersja mnożenia (dla mnie niezrozumiała).



Mnożenie liczb, algorytm2 - układ wykonawczy i diagram ASM układu sterowania – koncepcja, operacje

Jeszcze ciekawsza metoda, która jest jeszcze bardziej skomplikowana:



Ścieżka danych i diagram ASM dla mnożenia

5 Cześć dotycząca sprawdzianu nr 2 u dr Bucholca

Automaty:

- **Moore'a –** funkcje wyjściowe korzystają wyłącznie z rejestrów
- **Mealy'ego** – funkcje wyjściowe korzystają zarówno z wejść jak i rejestrów

Liczba stanów w automacie:

Lista stanów w automacie Moore'a jest **co najmniej równa** liczbie stanów automatu Mealy'ego.

Wyjścia automatu Mealy'ego **nie są synchroniczne.**

Wyjścia automatu Moore'a **są synchroniczne.**

Zapis liczb dla mikroprocesorów – niech liczba będzie 4F52, wtedy:

- Big-endian – 4F 52
- Small-endian – 52 4F

Obliczanie wartości liczby jako liczby zmiennoprzecinkowej 32-bitowej zgodnej ze standardem IEEE 754.

https://www.h-schmidt.net/FloatConverter/IEEE754.html

Obliczanie wartości liczby w U2:

https://www.exploringbinary.com/twos-complement-converter/

Minimalizacja funkcji (karnough)

http://32x8.com/index.html

Sumator:

http://home.agh.edu.pl/~brzoza/Technika_Cyfrowa/referaty/uk%B3ady%20arytmetyczne.pdf

Mnożenie liczb binarnych – wynikowa liczba będzie miała liczbę bitów równą sumie liczb bitów sumowanych liczb (?) – moim zdaniem tak

O dzień dobry

PicoBlaze – dokumentacja: Xilinx_UG129_PicoBlaze_8-bit_Embeded_Microcontroller_User_Guide (v2.0)

PicoBlaze **nie jest** przykładem architektury Von Neumanna.

Oznaczenie $wartość - oznacza, że wartość jest w postaci heksadecymalnej.

Rejestry – od s0 do sF (0-15)

1. **Porty definiujemy dyrektywami –** numery portów są z zakresu 0-255**:**
    - DSIN
    - DSOUT

- DSIO

Np. We DSIN 1 – (1 jest numerem portu z zakresu 0-255)

2. **Przerwania**
    a. Na początku programu należy odblokować przerwania – eint;
    b. Początek programu przerwania – zawsze pod adresem $3FF.
    c. Powrót z programu obsługi przerwania realizuje rozkaz RETURNI ENABLE (**w PicoBlaze IDE RETI ENABLE**) – jeżeli przerwania mają zostać zablokowane, to zamiast enable używamy disable
3. **Programy powinny działać w nieskończonej pętli**
4. **Podstawowe polecenia**
    a. **out [port] [wartość/rejestr] ??**
    b. **Skoki warunkowe:**

JUMP C, aaa	Jeśli flaga CARRY jest ustawiona, to skok programu do adresu aaa	Jeśli CARRY=1, PC←aaa	–	–
JUMP NC, aaa	Jeśli flaga CARRY nie jest ustawiona, to skok programu do adresu aaa	Jeśli CARRY=0, PC←aaa	–	–
JUMP NZ, aaa	Jeśli flaga ZERO nie jest ustawiona, to skok programu do adresu aaa	Jeśli ZERO=0, PC←aaa	–	–
JUMP Z, aaa	Jeśli flaga ZERO jest ustawiona, to skok programu do adresu aaa	Jeśli ZERO=1, PC←aaa	–	–

Tab. 2.

Instrukcja	Opis	Funkcja	ZERO	CARRY
ADD sX, kk	Suma rejestru sX i stałej kk	sX←sX+kk	?	?
ADD sX, sY	Suma rejestru sX i rejestru sY	sX←sX+sY	?	?
ADDCY sX, kk (ADDC)	Suma rejestru sX i stałej kk z bitem przeniesienia CARRY	sX←sX+kk+CARRY	?	?
ADDCY sX, sY (ADDC)	Suma rejestru sX i rejestru sY z bitem przeniesienia CARRY	sX←sX+sY+CARRY	?	?
AND sX, kk	Logiczny iloczyn bitów rejestru sX i stałej kk	sX←sX AND kk	?	0
AND sX, sY	Logiczny iloczyn bitów rejestru sX i rejestru sY	sX←sX AND sY	?	0
CALL aaa	Skok do procedury pod aaa	TOS←PC PC←aaa	–	–
CALL C, aaa	Jeśli flaga CARRY jest ustawiona, skok do procedury pod aaa	Jeśli CARRY=1, {TOS←PC, PC←aaa}	–	–
CALL NC, aaa	Jeśli flaga CARRY nie jest ustawiona, skok do procedury pod aaa	Jeśli CARRY=0, {TOS←PC, PC←aaa}	–	–
CALL NZ, aaa	Jeśli flaga ZERO nie jest ustawiona, skok do procedury pod aaa	Jeśli ZERO=0, {TOS←PC, PC←aaa}	–	–
CALL Z, aaa	Jeśli flaga ZERO jest ustawiona, skok do procedury pod aaa	Jeśli ZERO=1, {TOS←PC, PC←aaa}	–	–
COMPARE sX, kk (COMP)	Porównaj rejestr sX i stałą kk, ustaw odpowiednio flagi CARRY i ZERO	Jeśli sX=kk, ZERO←1 Jeśli sX<kk, CARRY←1	?	?
COMPARE sX, sY (COMP)	Porównaj rejestr sX i rejestr sY, ustaw odpowiednio flagi CARRY i ZERO	Jeśli sX=sY, ZERO←1 Jeśli sX<sY, CARRY←1	?	?
DISABLE INTERRUPT (DINT)	Zablokuj wejście przerwania	INTERRUPT_ENABLE←0	–	–
ENABLE INTERRUPT (EINT)	Odblokuj wejście przerwania	INTERRUPT_ENABLE←1	–	–
FETCH sX, (sY) (FETCH sX, sY)	Odczytaj zawartość pamięci podręcznej RAM, wskazywanej przez rejestr sY, do rejestru sX	sX←RAM[(sY)]	–	–
FETCH sX, ss	Odczytaj zawartość pamięci podręcznej RAM, wskazywanej przez wartość ss, do rejestru sX	SX←RAM[ss]	–	–
INPUT sX, (sY) (IN sX, sY)	Odczytaj zawartość portu wejściowego, wskazywanego przez rejestr sY, do rejestru sX	PORT ID←sY sX←IN_PORT	–	–
INPUT sX, pp (IN sX, pp)	Odczytaj zawartość portu wejściowego, wskazywanego przez wartość pp, do rejestru sX	PORT ID←pp sX←IN_PORT	–	–
JUMP aaa	Bezwarunkowy skok programu do adresu aaa	PC←aaa	–	–
JUMP C, aaa	Jeśli flaga CARRY jest ustawiona, to skok programu do adresu aaa	Jeśli CARRY=1, PC←aaa	–	–
JUMP NC, aaa	Jeśli flaga CARRY nie jest ustawiona, to skok programu do adresu aaa	Jeśli CARRY=0, PC←aaa	–	–
JUMP NZ, aaa	Jeśli flaga ZERO nie jest ustawiona, to skok programu do adresu aaa	Jeśli ZERO=0, PC←aaa	–	–
JUMP Z, aaa	Jeśli flaga ZERO jest ustawiona, to skok programu do adresu aaa	Jeśli ZERO=1, PC←aaa	–	–
LOAD sX, kk	Załaduj rejestr sX wartością kk	sX←kk	–	–
LOAD sX, sY	Załaduj rejestr sX wartością rejestru sY	sX←sY	–	–
OR sX, kk	Logiczna suma bitów rejestru sX i stałej kk	sX←sX OR kk	?	0
OR sX, sY	Logiczna suma bitów rejestru sX i rejestru sY	sX←sX OR sY	?	0
RETURN (RET)	Bezwarunkowy powrót z procedury	PC←TOS+1	–	–
RETURN C (RET C)	Jeśli flaga CARRY jest ustawiona, to powrót z procedury	Jeśli CARRY=1, PC←TOS+1	–	–
RETURN NC (RET NC)	Jeśli flaga CARRY nie jest ustawiona, to powrót z procedury	Jeśli CARRY=0, PC←TOS+1	–	–
RETURN NZ (RET NZ)	Jeśli flaga ZERO nie jest ustawiona, to powrót z procedury	Jeśli ZERO=0, PC←TOS+1	–	–
RETURN Z (RET Z)	Jeśli flaga ZERO jest ustawiona, to powrót z procedury	Jeśli ZERO=1, PC←TOS+1	–	–
RETURN DISABLE (RETI DISABLE)	Powrót z procedury obsługi przerwania, przerwanie pozostaje zablokowane	PC←TOS ZERO←zachowane ZERO CARRY←zachowane CARRY INTERRUPT_ENABLE←0	?	?
RETURN ENABLE (RETI ENABLE)	Powrót z procedury obsługi przerwania, aktywowanie przerwania	PC←TOS ZERO←zachowane ZERO CARRY←zachowane CARRY INTERRUPT_ENABLE←1	?	?
RL sX	Obróć bity rejestr sX w lewo o jedną pozycję	sX←{sX[6:0],sX[7]} CARRY←sX[7]	?	?

Tab. 2. c.d.

Instrukcja	Opis	Funkcja	ZERO	CARRY
RR sX	Obróć bity rejestr sX w prawo o jedną pozycję	sX←{sX[0],sX[7:1]} CARRY←sX[0]	?	?
SL0 sX	Obróć bity rejestru sX w lewo o jedną pozycję, uzupełnij wartością '0'	sX←{sX[6:0],0} CARRY←sX[7]	?	?
SL1 sX	Obróć bity rejestru sX w lewo o jedną pozycję, uzupełnij wartością '1'	sX←{sX[6:0],1} CARRY←sX[7]	?	?
SLA sX	Obróć bity rejestru sX w lewo o jedną pozycję, uzupełnij wartością flagi CARRY	sX←{sX[6:0],CARRY} CARRY←sX[7]	?	?
SLX sX	Obróć bity rejestru sX w lewo o jedną pozycję, nie zmieniaj wartości bitu sX[0]	sX←{sX[6:0],sX[0]} CARRY←sX[7]	?	?
SR0 sX	Obróć bity rejestru sX w prawo o jedną pozycję, uzupełnij wartością '0'	sX←{0,sX[7:1]} CARRY←sX[0]	?	?
SR1 sX	Obróć bity rejestru sX w prawo o jedną pozycję, uzupełnij wartością '1'	sX←{1,sX[7:1]} CARRY←sX[0]	?	?
SRA sX	Obróć bity rejestru sX w prawo o jedną pozycję, uzupełnij wartością flagi CARRY	sX←{CARRY,sX[7:1]} CARRY←sX[0]	?	?
SRX sX	Obróć bity rejestru sX w prawo o jedną pozycję, nie zmieniaj wartości bitu sX[7]	sX←{sX[7] ,sX[7:1]} CARRY←sX[0]	?	?
STORE sX, (sY) (STORE sX, sY)	Zapisz zawartość pamięci podręcznej RAM, wskazywanej przez rejestr sY, wartością rejestru sX	RAM[(sY)]← sX	–	–
STORE sX, ss	Zapisz zawartość pamięci podręcznej RAM, wskazywanej przez wartość ss, wartością rejestru sX	RAM[ss] ←sX	–	–
SUB sX, kk	Różnica rejestru sX i stałej kk	sX←sX–kk	?	?
SUB sX, sY	Różnica rejestru sX i rejestru sY	sX←sX–sY	?	?
SUBCY sX, kk (SUBC)	Różnica rejestru sX i stałej kk z bitem przeniesienia CARRY	sX←sX–kk–CARRY	?	?
SUBCY sX, sY (SUBC)	Różnica rejestru sX i rejestru sY z bitem przeniesienia CARRY	sX←sX–sY–CARRY	?	?
TEST sX, kk	Test bitów rejestru sX z wartością kk	Jeśli (sX AND kk)=0, ZERO←1 CARRY←parzystość bitów wyrażenia (sX AND kk)	?	?
TEST sX, sY	Test bitów rejestru sX z wartością rejestru sY	Jeśli (sX AND sY)=0, ZERO←1 CARRY←parzystość bitów wyrażenia (sX AND sY)	?	?
XOR sX, kk	Logiczna suma XOR bitów rejestru sX i stałej kk	sX←sX XOR kk	?	0
XOR sX, sY	Logiczna suma XOR bitów rejestru sX i rejestru sY	sX←sX XOR sY	?	0

sX – jeden z 16 rejestrów: od s0 do sF
sY – jeden z 16 rejestrów: od s0 do sF
aaa – 10–bitowy adres, możliwe są tylko wartości heksadecymalne od 000 do 3FF (od 0 do 1023)
kk – 8–bitowa stała w postaci heksadecymalnej: od 00 do FF
pp – 8–bitowy adres portu w postaci heksadecymalnej: od 00 do FF
ss – 6–bitowy adres pamięci podręcznej w postaci heksadecymalnej: od 00 do 3F
RAM[n] – zawartość podręcznej pamięci pod adresem n
TOS – wartość na szczycie stosu

Zakres sprawdzianu:

- Automaty
  - Teoria działania
  - Tworzenie
- Przeliczanie do systemu IEEE 754
- Asembler dla procesora 8080
- Konwersja stanu procesora (wykresu) na kod maszynowy
- Procesor PicoBlaze i jego dialekt asemblera
- Budowa „płyty głównej”, czyli połączenia między urządzeniami wejścia/wyjścia, pamięcią
- Wszystkie operacje arytmetyczne z poprzedniego sprawdzianu