Podstawy techniki cyfrowej

Rafał Walkowiak

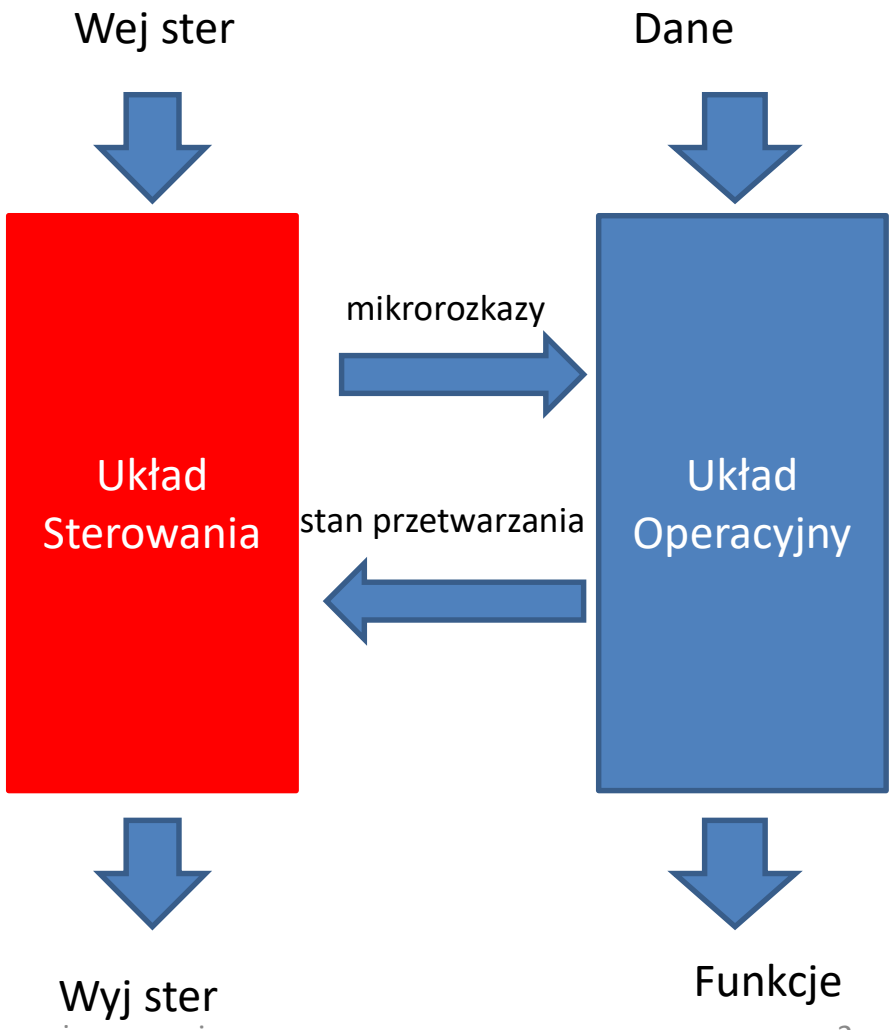
Synteza wyższego poziomu

8.12.2020

Układy cyfrowe

synteza strukturalna wyższego poziomu

- Ogólna struktura logiczna UC:
  - bloki funkcjonalne dla realizacji określonych funkcji przetwarzania danych czyli układ operacyjny (lub ścieżka danych - ang. data path)
  - układ sterowania (automat lub układ mikroprogramowalny)



Synteza wyższego poziomu

- Projektowanie układu cyfrowego (podejście strukturalne)
  - Algorytm dla problemu
  - Synteza układu operacyjnego (układu przetwarzania danych), projekt „ścieżki danych”
  - Diagram ASM
  - Projekt układu sterowania
    - Określenie stanów automatu na podstawie diagramu ASM
    - Wybór struktury fizycznej US (funkcje przełączające + rejestry, pamięć + rejestr, układ mikroprogramowalny,…)
  - Realizacja projektu

Synteza strukturalna układu operacyjnego

Analizując algorytm pracy układu opisany siecią działań dobiera się:

- 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.

Projektowanie układów cyfrowych na poziomie operacji przesłań międzyrejestrowych (RTL ang. register transfer operations level)

Cechy charakterystyczne podejścia:

- zastosowanie koncepcji zmiennych i opis działań za pomocą sekwencji operacji (opisu charakterystycznego dla algorytmów);
- wykorzystanie rejestrów do przechowywania wartości pośrednich – rejestry modelują zmienne z algorytmu;
- zaprojektowana ścieżka danych ma za zadanie realizować opisane w algorytmie operacje na wartościach przechowywanych w rejestrach;
- układ sterujący powinien zapewnić - przy użyciu sygnałów sterujących - kolejność operacji zgodną z opisem z algorytmu.

Metodologia RTL

- Operacja przesłania międzyrejestrowego generuje wartość zapisywaną do rejestru wynikowego wyznaczoną na podstawie argumentów wejściowych pobranych z odpowiednich rejestrów:

  Rrezultat ← funkcja (Rarg_ 1, Rarg_2, …, Rarg_n)

- Funkcja jest realizowana w ramach modułu funkcjonalnego w układzie wykonawczym.
- Operacje przesłania międzyrejestrowego odbywają się w rytm taktów zegara systemowego.

Realizacja operacji przesłań międzyrejestrowych

Operację Rrezultat ← ***f***(Rarg_ 1, Rarg_2, …, Rarg_n) jest realizowana w kolejnych krokach:

1. wraz z narastającym zboczem zegara systemowego rejestry Rarg_ 1, Rarg_2, …, Rarg_n otrzymują nowe wartości,
2. funkcja ***f*** oblicza wartość na podstawie zawartości rejestrów Rarg_ 1, Rarg_2, …, Rarg_n,
3. wynik obliczeń jest przesyłany na wejście danych rejestru Rrez,
4. kolejne narastające zbocze zegara systemowego powoduje:
   - zapamiętanie nowej wartości w rejestrze Rrezultat
   - uaktualnienie zawartości rejestrów Rarg_ 1, Rarg_2, …, Rarg_n
5. Kontynuacja pracy od kroku 2

Operacje przesłań międzyrejestrowych

Przerzutniki i połączone z nimi bramki dowolnego układu sekwencyjnego można nazwać rejestrami.

Operacje jednego rejestru (licznika) to operacje elementarne:

wpis wartości, zliczanie, dodawanie , odejmowanie, przesunięcie.

Mikrooperacje to operacje realizowane na danych przechowywanych rejestrach:

Np. załadowanie informacji z jednego rejestru do innego, dodawanie zawartości dwóch rejestrów i zapis wyniku w innym rejestrze lub rejestrze źródłowym argumentu.

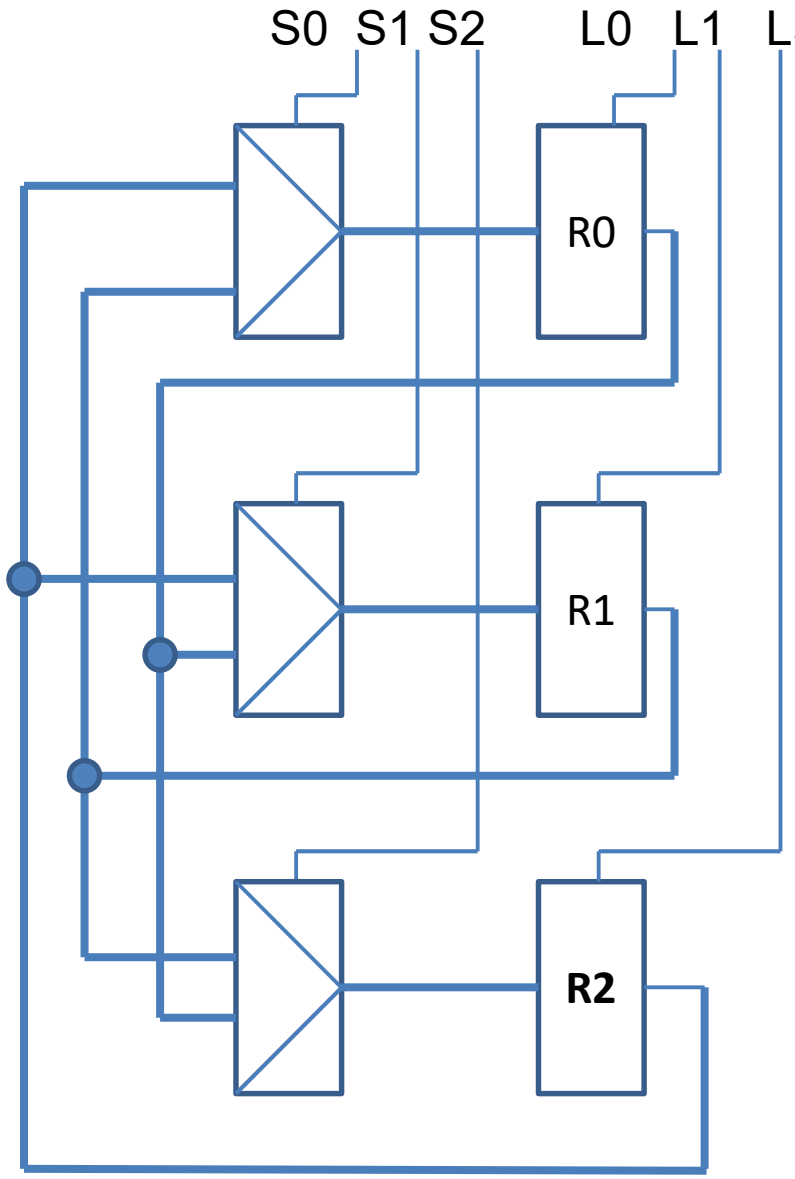
Operacje przesłań międzyrejestrowych przykład

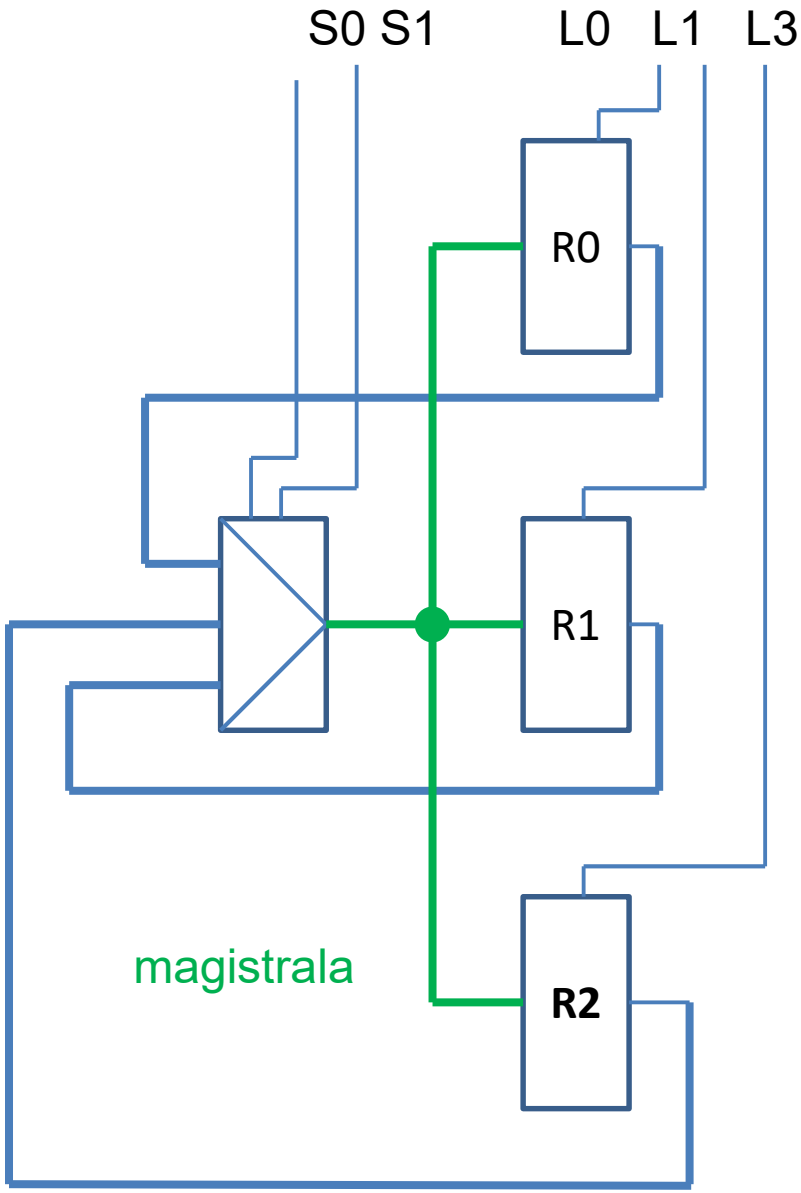
- C1:R2 ← R1
- Zapis oznacza operację przesłania miedzy rejestrami R1 i R2 wykonaną gdy sygnał C1 przyjmie wartość 1. Operacja ta zakłada ścieżkę informacyjną pomiędzy R1 i R2. Sygnał sterujący aktywny zostanie wygenerowany przez układ sterujący na skutek zbocza narastającego zegara, a podany na wejście rejestru spowoduje zapis wartości kolejnym zboczem narastającym zegara.

Operacje przesłań międzyrejestrowych

- Mikrooperacje przesłania:
  - przesłanie z jednego rejestru do drugiego
  - Realizacje:
    - połączenie bezpośrednie
    - połączenie przez multiplekser dedykowany do rejestru - możliwość przesłań równoległych różnych informacji do rejestrów lub
    - połączenie przez multiplekser i magistralę – możliwość równoległych przesłań jednakowych informacji do rejestrów.

Mikrooperacje przesłania





Operacje przesłań międzyrejestrowych - notacja

Symbol	opis	przykłady
Litery i liczby	Oznaczenie rejestru	DR, AR, D1, IR
Nawiasy	Oznaczenie części rejestru	R2(1), R2(7:0)
Strzałka	Kierunek przesłania danych	R1 ← R2
Przecinek	Specyfikacja przesłań równoczesnych	R1 ← R2 , R3 ← R2
Nawiasy klamrowe	Oznaczenie adresu w pamięci	DR ← M[AR]

Operacje przesłań międzyrejestrowych

- Mikrooperacje arytmetyczne:
  - R0 ← R1+R2 - dodawanie
  - R2 ← R2’ – negacja (uzupełnienie do jeden)
  - R2 ← R2’ + 1 – uzupełnienie do 2
  - R0 ← R1+R2’+1 – odejmowanie (kod uzupełnieniowy)
  - 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, R1 ← sl R2 – operacje przesunięcia zgodne z ich definicją uwzględniającą wartości bitu wchodzącego wpisywanego na pozycji najmłodszej (sl - przesunięcie w lewo) lub najstarszej (sr – przesunięcie w prawo)
  - Przesuniecie zawartości rejestru – dzielenie, mnożenie; rejestr przesuwny

Synteza układu operacyjnego

Na podstawie sieci działań algorytmu można określić schemat układu operacyjnego uwzględniając:

- dane wejściowe, pośrednie i wyjściowe algorytmu – przechowywane w rejestrach,
- operacje wykonywane na danych i realizujące te operacje układy
- przesyłanie danych pomiędzy elementami struktury.

W wyniku analizy sieci działań i schematu układu operacyjnego powstaną:

- struktura powiązań elementów cyfrowych realizujących operacje przesłań międzyrejestrowych
- lista sygnałów sterujących realizacją operacji i
- lista sygnałów określających wyniki operacji.

Przykład: instancja i algorytm dla problemu konwersji liczby binarnej na BCD BIN→BCD

Lk	ld	b			ld	a		
	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	1	0
4	0	0	0	0	0	1	0	0
3	0	0	0	0	1	0	0	0
						+	1	1
	0	0	0	0	1	0	1	1
2	0	0	0	1	0	1	1	0
						+	1	1
	0	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1	0
0	0	1	1	0	0	1	0	0
			6			4		

			lb				
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

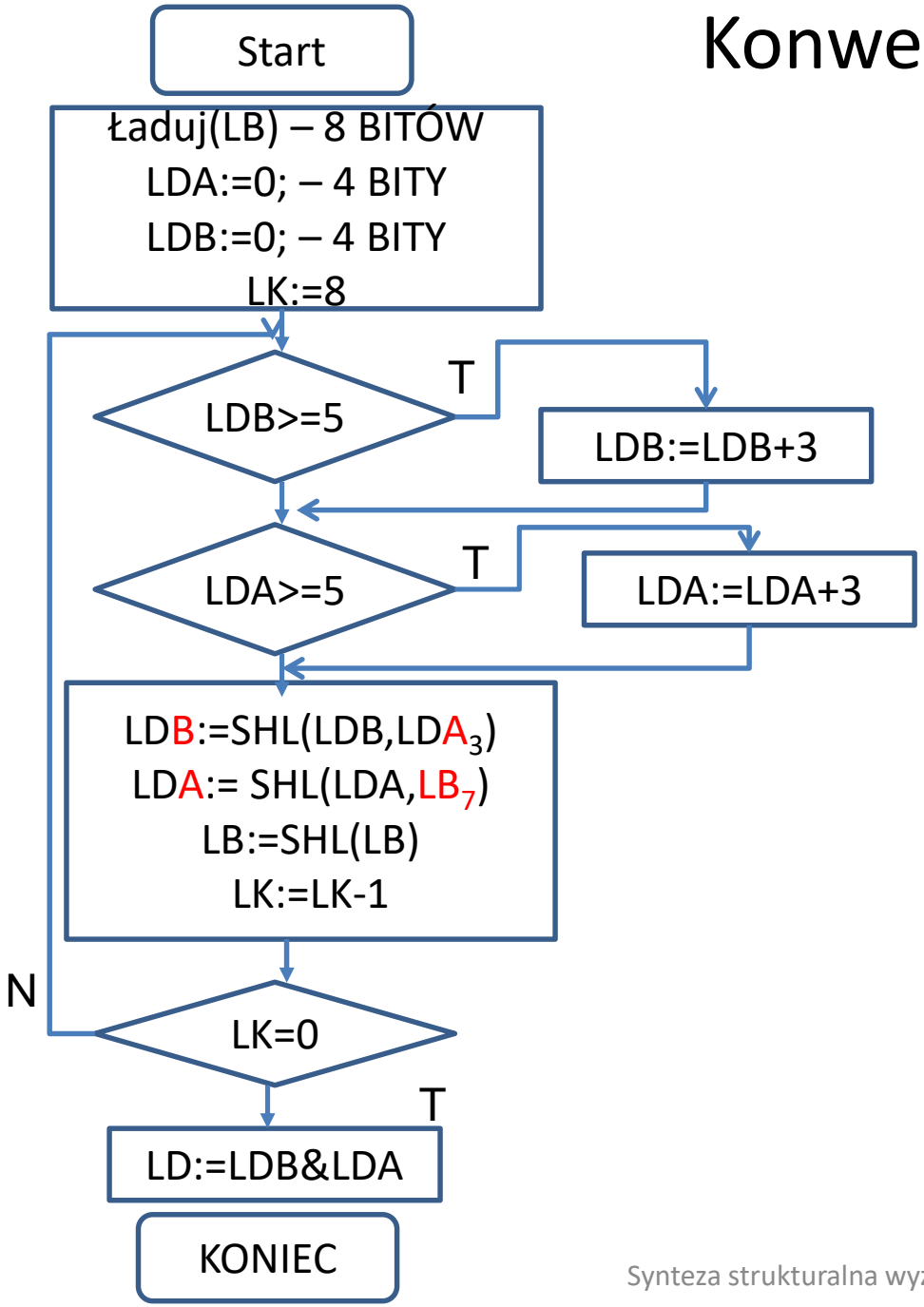
Wejście, wynik tymczasowy w 2 częściach, przesunięcie z testem
Konwersja liczby $64_D = 01000000_{BIN}$
Na liczbę 01100100_{BCD}

BIN→BCD przykład

Lk	ld	b			ld	a		
	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	1	0
4	0	0	0	0	0	1	0	0
3	0	0	0	0	1	0	0	0
						+	1	1
	0	0	0	0	1	0	1	1
2	0	0	0	1	0	1	1	0
						+	1	1
	0	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1	0
0	0	1	1	0	0	1	0	0
			6			4		

			lb				
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Konwersja liczby $64_D = 01000000_{BIN}$
Na liczbę 01100100_{BCD}



Konwersja liczby binarnej na BCD BIN→BCD

ALGORYTM

Ograniczamy rozmiar instancji do liczb < 99 wprowadzając 2 rejestry (algorytm w postaci ogólnej)

LB – konwertowana liczba binarna

LDA –młodsza, LDB starsza część wyniku częściowego

LD – wynik BCD

LK – licznik kroków- liczba bitów konwertowanej liczby

L B_7 -oznacza aktualny najstarszy bit liczby konwertowanej

Jak zrealizować algorytm w układzie cyfrowym?

Jakie elementy są potrzebne do przetwarzania danych ?

Rejestry: zerowanie, przesuw zawartości, wejscie równoległe, wejście szeregowe.

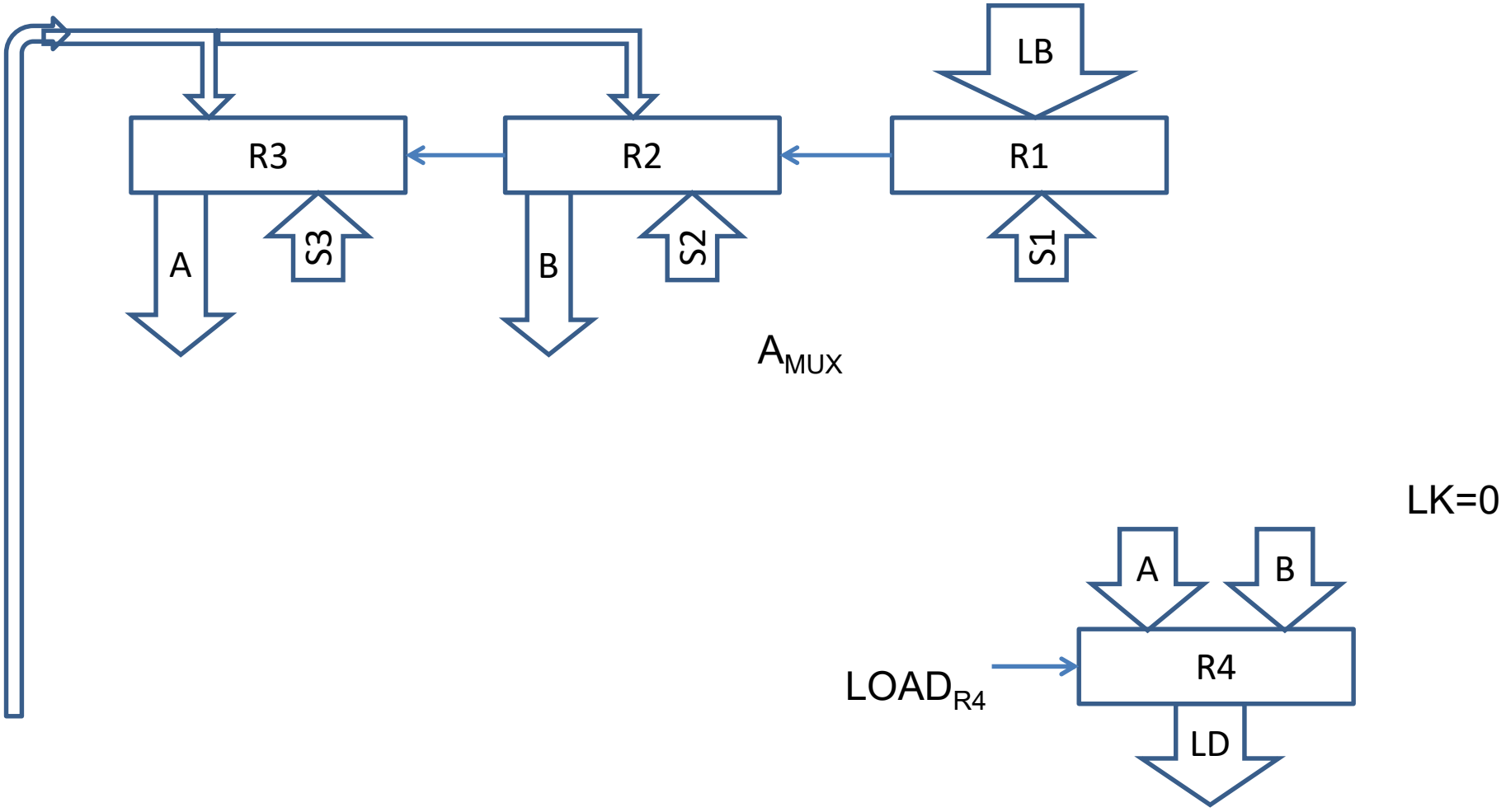
Sumator

Komparator

Licznik

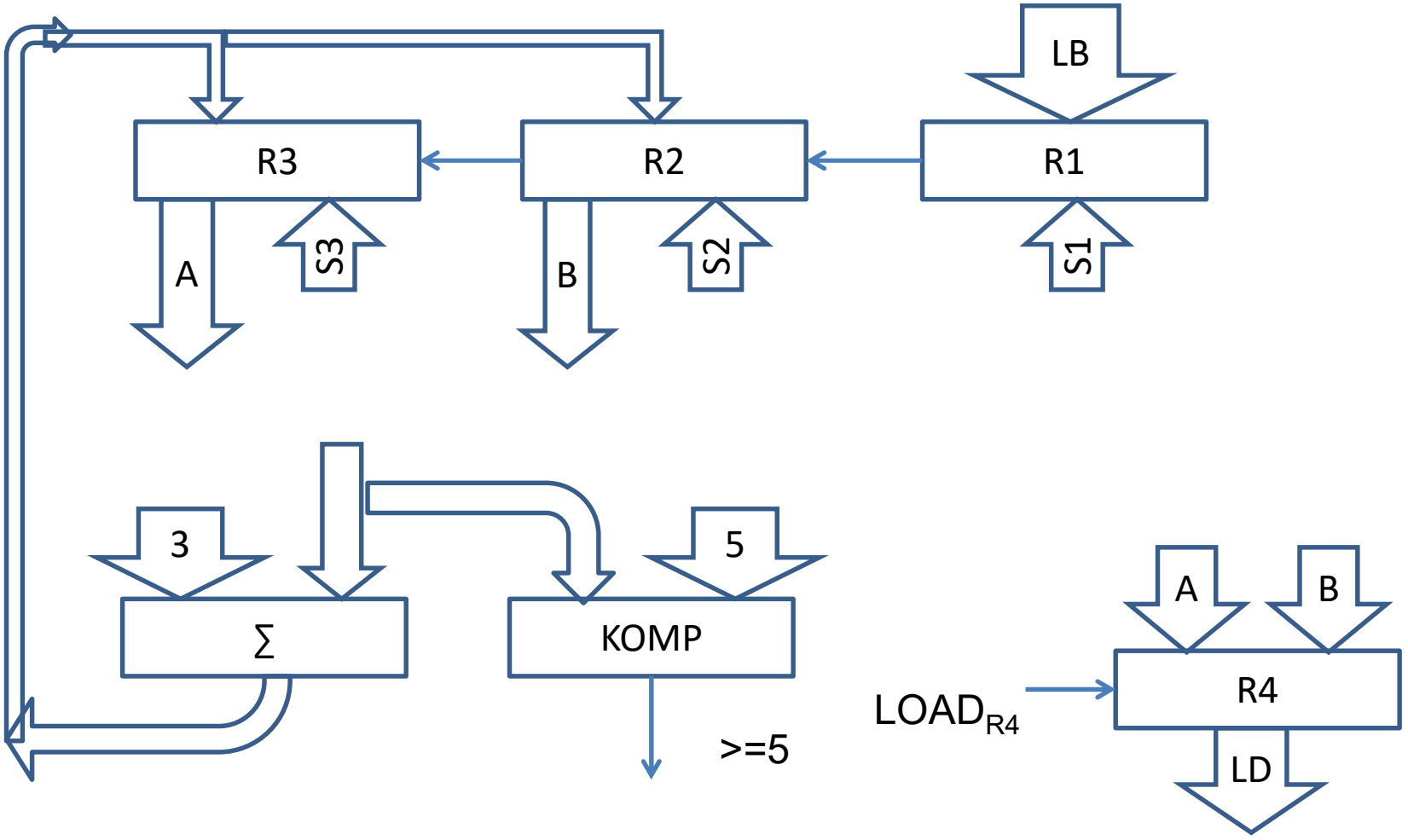
Multiplekser

Schemat układu operacyjnego BIN→BCD



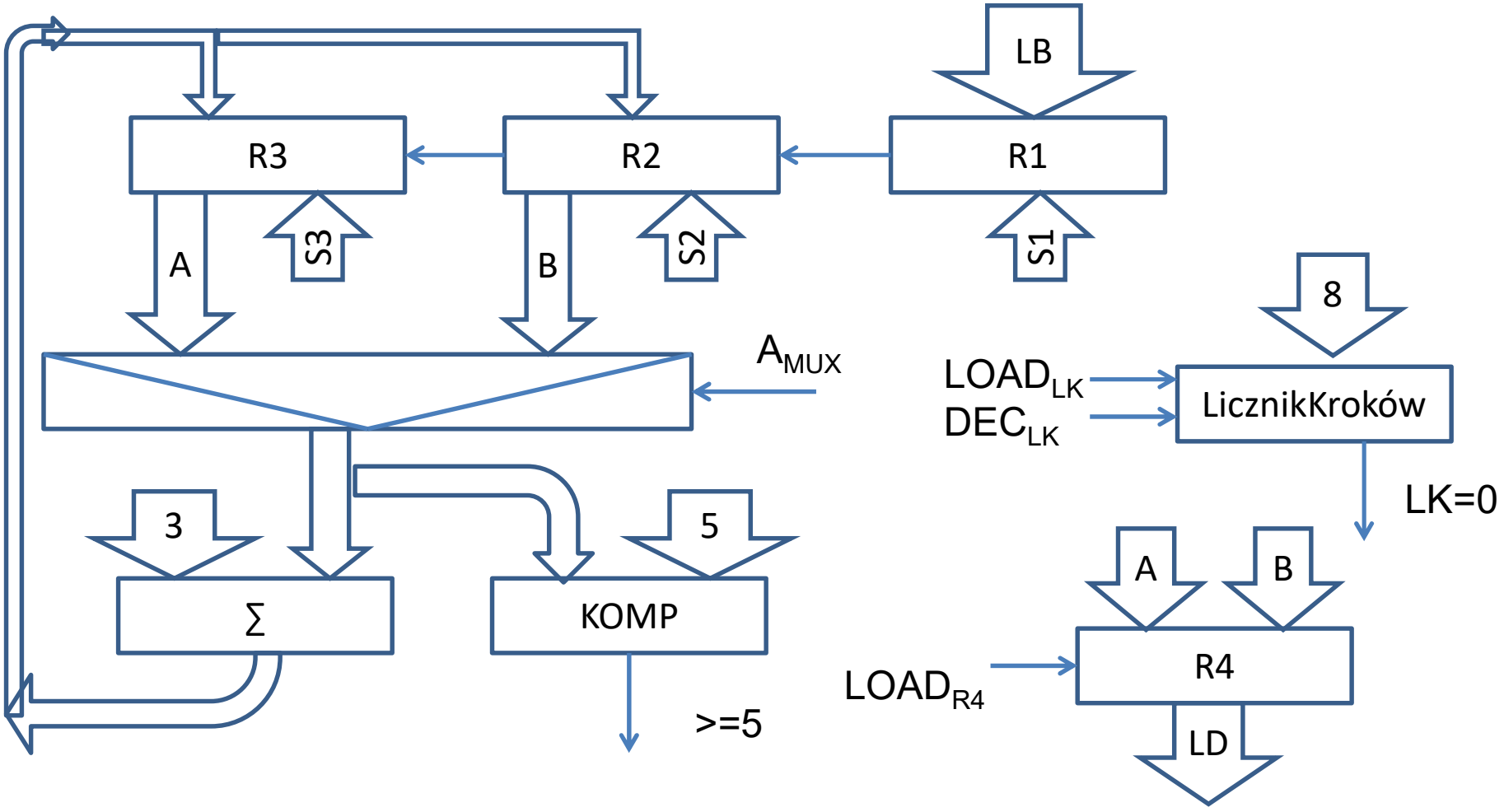
Układy sekwencyjne posiadają niezaznaczone wejścia zegarowe

Schemat układu operacyjnego BIN→BCD



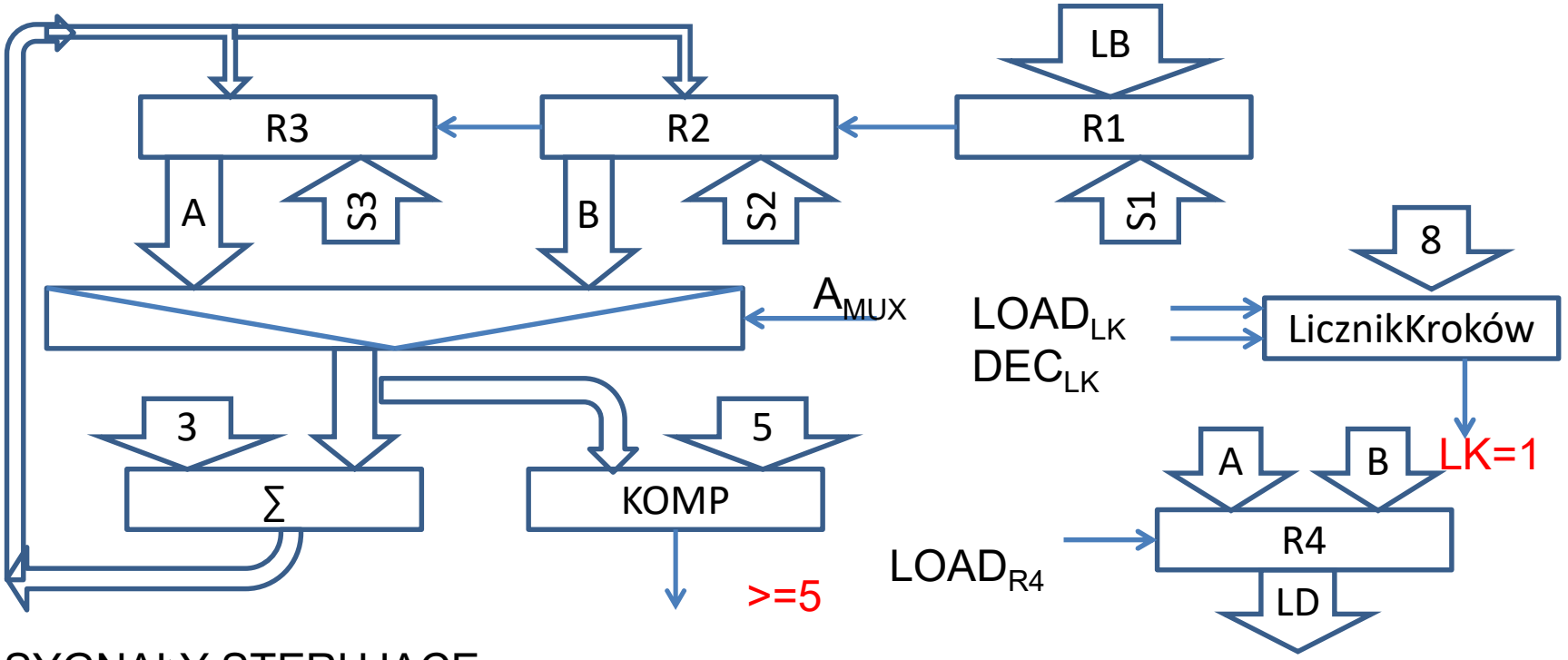
Układy sekwencyjne posiadają niezaznaczone wejścia zegarowe

Schemat układu operacyjnego BIN→BCD



Układy sekwencyjne posiadają niezaznaczone wejścia zegarowe

Schemat układu operacyjnego BIN→BCD



SYGNAŁY STERUJĄCE:

- Inicjalizacja: zerowanie rejestrów R2,R3, ładowanie rejestru R1, ładowanie licznika kroków
- Przesuw zawartości rejestrów R1 do R3 w lewo
- sterowanie multiplekserem
- zgoda na zliczanie licznika – DEC_{LK}
- indywidualne sygnały ładowania rejestrów R2,R3,R4

SYGNAŁY STANU (CZERWONE)

Układ sterujący

- Nomenklatura: układ sekwencyjny, automat, maszyna stanów, automat skończony (FSM).
- Metoda algorytmicznego układu sekwencyjnego to metoda ASM – użycie diagram ASM (mniej sformalizowany graf stanów).

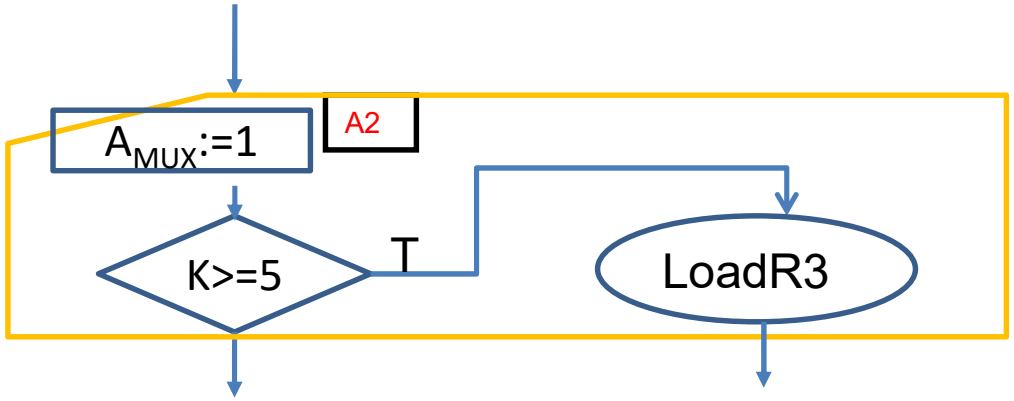
Układ sterujący ma za zadanie wymusić odpowiednią kolejność operacji przesłań międzyrejestrowych.

- US realizowany jest realizowany jako automat tworzony na podstawie sieci działań algorytmu, który ma zostać zrealizowany.
- Kroki dla realizacji układu sterowania:
  - konwersja sieci działań algorytmu na diagram ASM przy uwzględnieniu sygnałów sterujących i sygnałów stanu układu wykonawczego
  - określenie stanów automatu na podstawie diagramu ASM – wyodrębnienie działań równoczesnych i kolejnych
  - realizacja układu sterowania na podstawie diagramu ASM.

Diagramy ASM

- Diagramy ASM stanowią alternatywną (obok grafu stanów) metodę opisu automatów.
- Pozwalają one reprezentować cyfrowe układy sekwencyjne w postaci sieci działań.
- Diagram ASM składa się z bloków ASM. Blok zawiera:
  - klatkę operacyjną (klatkę stanu),
  - klatki decyzyjne i
  - warunkowe klatki wyjść.
- 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.

DIAGRAM ASM



Przykład pełnego zestawu klatek diagramu ASM wchodzących w skład opisu działań w stanie automatu A2:
Klatka operacyjna specyfikuje sygnały aktywowane bezwarunkowo przy wejściu do stanu np A_{MUX}:=1, A_{MUX} wymuszenie stanu 1, A_{MUX}' 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).

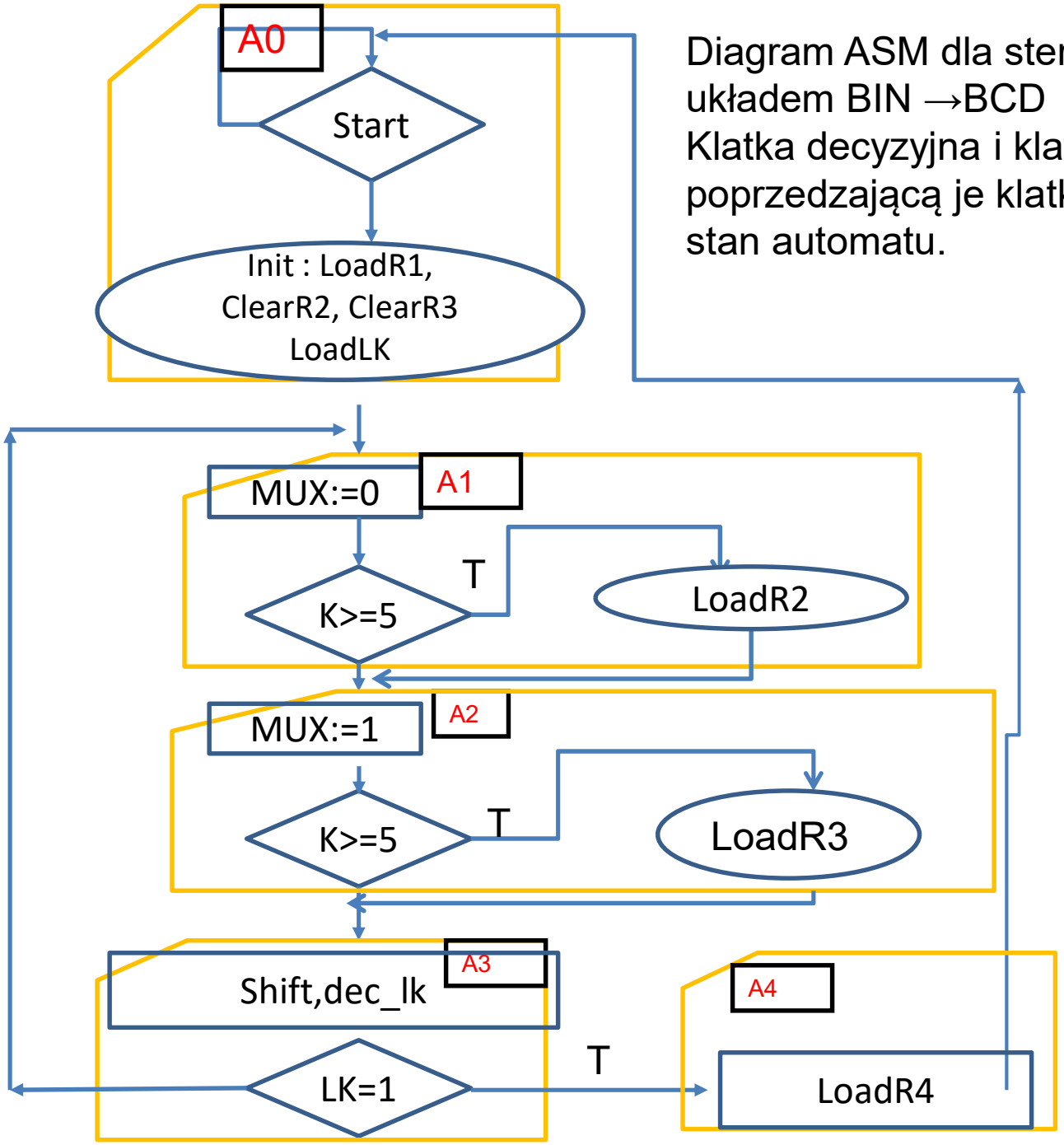


Diagram ASM dla sterowania układem BIN →BCD
Klatka decyzyjna i klatki operacyjne wraz z poprzedzającą je klatką operacyjną tworzą stan automatu.

W klatkach operacyjnych podano listę aktywnych sygnałów wyjściowych
shl - przesuw w lewo,
dec - zmniejsz,
w klatkach decyzyjnych testujemy sygnały wejściowe.

Dodatkowy stan A4 niezbędny ze względu na konieczność wcześniejszego przesunięcia wyniku.

TABLICA PRZEJŚĆ I WYJŚĆ AUTOMATU STERUJĄCEGO BIN →BCD

stan	Start	K>=5	LK=1	stan+	INIT	SHIFT	LO_R2	LO_R3	LO_R4	MUX
000(A0)	0	X	X	000	0	0	0	0	0	X
000	1	X	X	001	1	0	0	0	0	X
001(A1)	X	0	X	011	0	0	0	0	0	0
001	X	1	X	011	0	0	1	0	0	0
011(A2)	X	0	X	010	0	0	0	0	0	1
011	X	1	X	010	0	0	0	1	0	1
010(A3)	X	X	1	110	0	1	0	0	0	X
010	X	X	0	001	0	1	0	0	0	X
110(A4)	X	X	X	000	0	0	0	0	1	X

Powyżej kodowanie kodem Greya
Wektor wejść (zawiera 3 sygnały) : START, K>=5, LK=1
SHIFT=DEC_{LK}

KODOWANIE STANÓW AUTOMATU

stan	Start	K>=5	LK=1	stan+	INIT	SHIFT	LO_R2	LO_R3	LO_R4	MUX
000(A0)	0	X	X	000	0	0	0	0	0	X
000	1	X	X	001	1	0	0	0	0	X
001(A1)	X	0	X	011	0	0	0	0	0	0
001	X	1	X	011	0	0	1	0	0	0
011(A2)	X	0	X	010	0	0	0	0	0	1
011	X	1	X	010	0	0	0	1	0	1
010(A3)	X	X	1	110	0	1	0	0	0	X
010	X	X	0	001	0	1	0	0	0	X
110(A4)	X	X	X	000	0	0	0	0	1	X

WYZANCZENIE FUNKCJI WZBUDZEŃ:

Użycie dekodera 1 z 5 (kodowanie jeden z N) pozwala na uzyskanie sygnałów określających stan automatu A0,A1,A2,A3,A4

Funkcje wzbudzeń przerzutników D można określić bazując na sygnałach określających stan automatów:

D0= A0*Start+A1+A3 *(LK=1)' – 4 czerwone jedynki

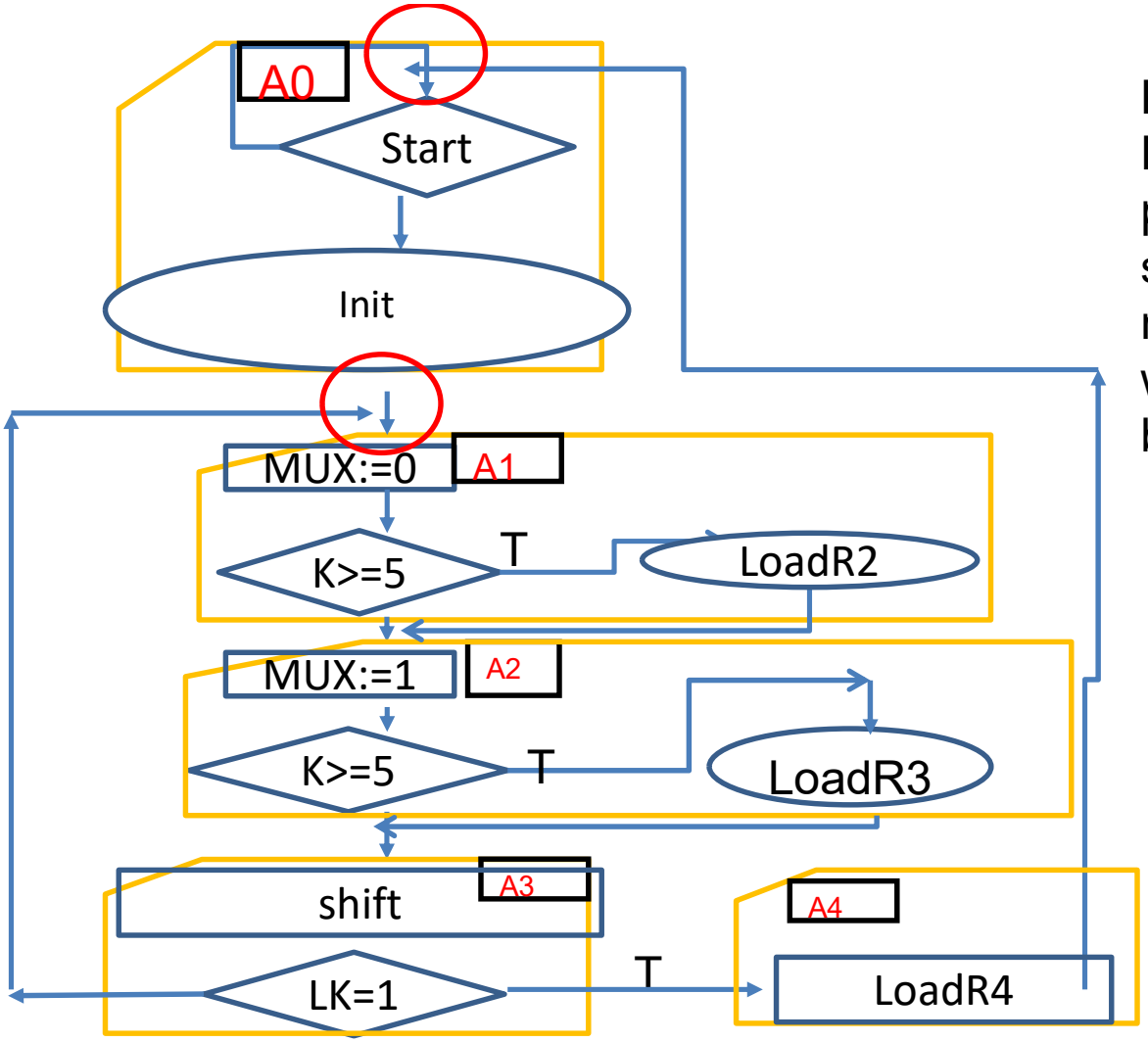
D1=A1+A2+A3*(LK=1) -

D2=A3*(LK=1) – jedna niebieska jedynka

WYZNACZENIE FUNKCJI WYJŚĆ

stan	Start	K>=5	LK=1	stan+	INIT	SHIFT	LO_R2	LO_R3	LO_R4	MUX
000(A0)	0	X	X	000	0	0	0	0	0	X
000	1	X	X	001	1	0	0	0	0	X
001(A1)	X	0	X	011	0	0	0	0	0	0
001	X	1	X	011	0	0	1	0	0	0
011(A2)	X	0	X	010	0	0	0	0	0	1
011	X	1	X	010	0	0	0	1	0	1
010(A3)	X	X	1	110	0	1	0	0	0	X
010	X	X	0	001	0	1	0	0	0	X
110(A4)	X	X	X	000	0	0	0	0	1	X

Funkcje wyjść określamy bazując na sygnałach określających stan A0,A1,A2,A3 :
INIT=A0*Start -- syganał przygotowania rejestrów i licznika
SHIFT=A3
LO_R2=A1*(K>=5); LO_R3=A2* (K>=5)
LO_R4=A4; MUX=A2

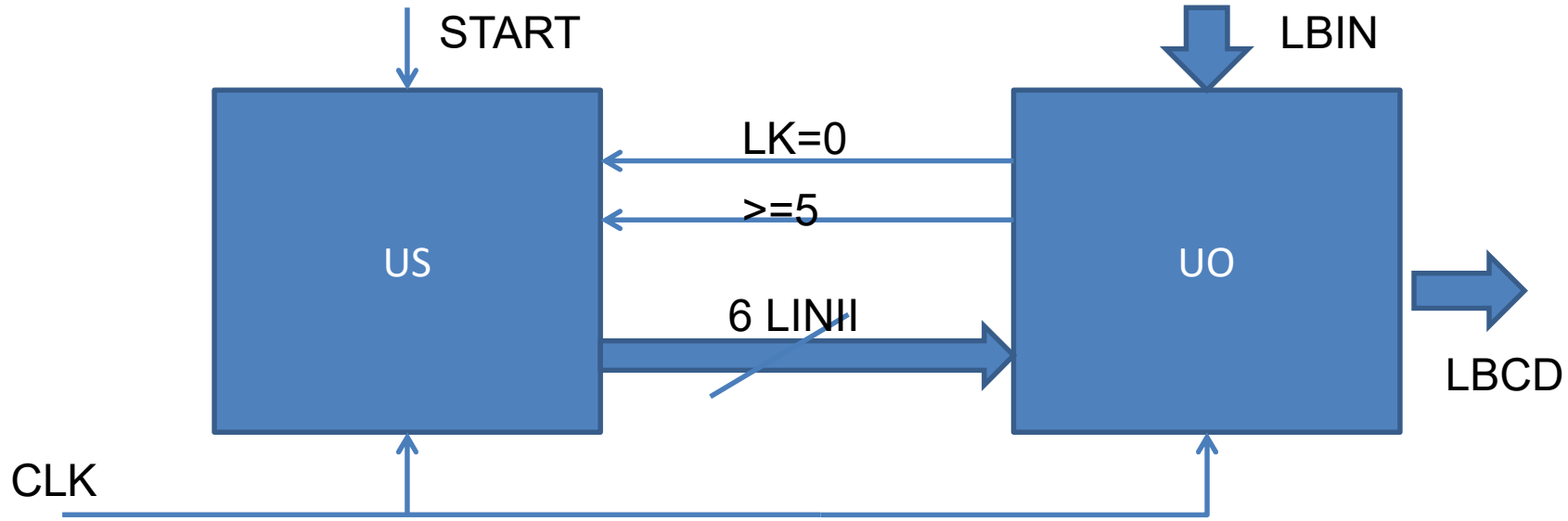


Kodowanie stanów kodem 1 z N i użycie dedykowanych przerzutników (generują sygnały A0,A1,A2 ..) pozwala na wyprowadzenie równań wzbudzeń i funkcji wyjść bezpośrednio z diagramu ASM.

DA0=A0*START' + A4
DA1=A0*START + A3*(LK=1)'
DA2=A1
DA3=A2
DA4=A3*(LK=1)
INIT=A0*Start;
MUX=A2;
LO_R2=A1*(K>=1)
LO_R3=A2*(K>=1)
SHIFT=A3;
LO_R4=A4;

Efektem połączenia linii wyjściowych z bloku stanu w diagramie ASM (czerwone koła) jest suma w wyrażeniu funkcji wzbudzenia przerzutnika. Bazując na powyższych równaniach można narysować schemat układu strowania. (DA0 to wejście przerzutnika dedykowanego do stanu A0)

STRUKTURA I SYGNAŁY WSPÓŁPRACY MODUŁÓW UKŁADU KONWERSJI BIN-BCD



Cykliczne działania w systemie cyfrowym:

1. GENERACJA NOWEGO STEROWANIA DLA BIEŻĄCEGO STANU I STANU WEJŚĆ
2. REALIZACJA PRZETWARZANIA ZGODNIE ZE STEROWANIEM BIEŻĄCEGO STANU,
3. PRZYGOTOWANIE WYNIKU PRZETWARZANIA
4. USTALENIE NOWEGO STAN AUTOMATU
5. ZBOCZE ZEGAROWE ZAPISUJE WYNIK PRZETWARZANIA I GENERUJE NOWY STAN
6. POWRÓT DO KROTU 1

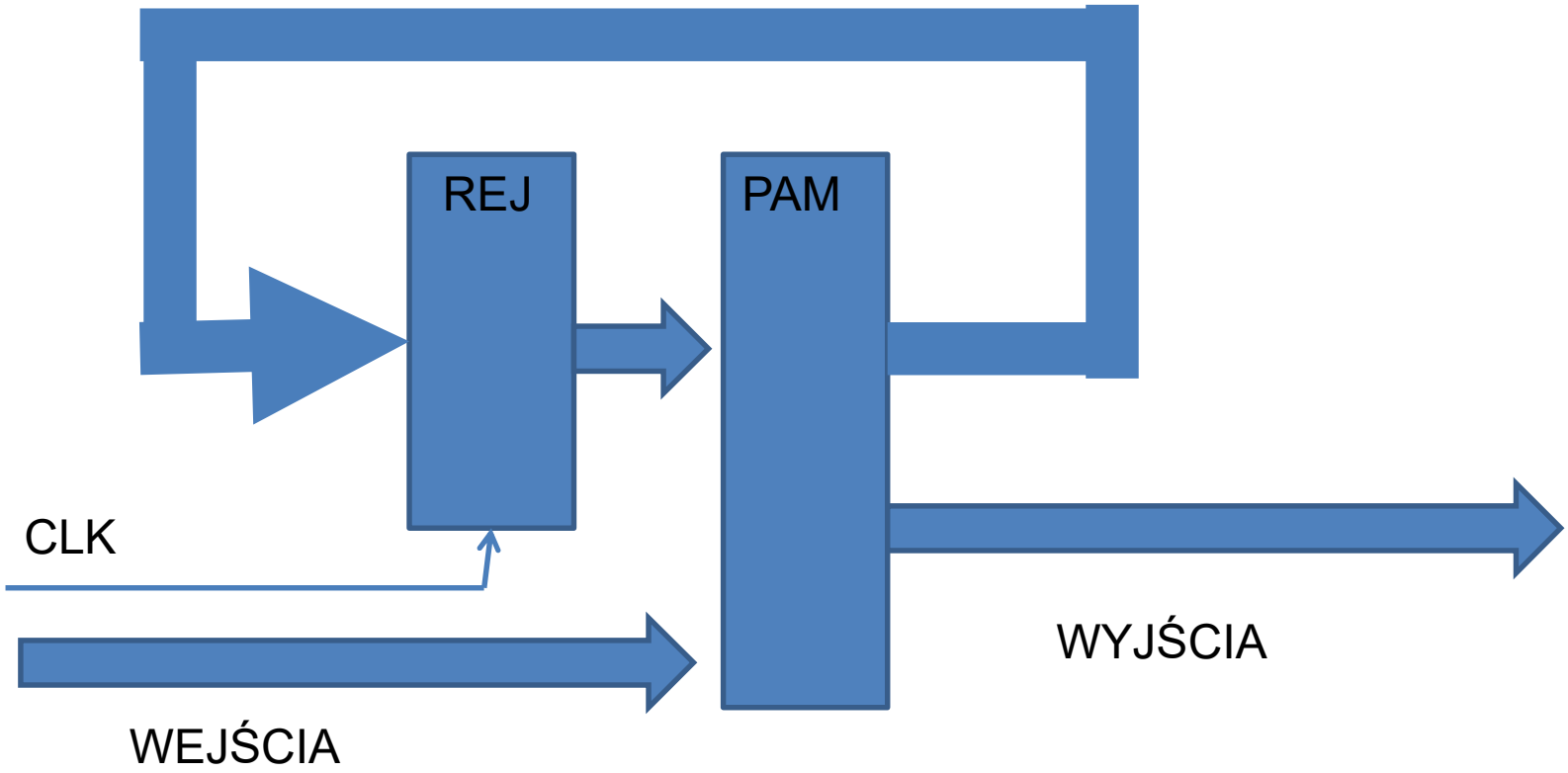
Zastosowanie rejestru i pamięci w strukturze US

stan	Start	K>=5	LK=1	stan+	INIT	SHIFT	LO_R2	LO_R3	LO_R4	MUX
000(A0)	0	X	X	000	0	0	0	0	0	X
000	1	X	X	001	1	0	0	0	0	X
001(A1)	X	0	X	011	0	0	0	0	0	0
001	X	1	X	011	0	0	1	0	0	0
011(A2)	X	0	X	010	0	0	0	0	0	1
011	X	1	X	010	0	0	0	1	0	1
010(A3)	X	X	1	110	0	1	0	0	0	X
010	X	X	0	001	0	1	0	0	0	X
110(A4)	X	X	X	000	0	0	0	0	1	X

Wartości argumentów funkcji zaznaczono na szaro, argumenty mogą być podane na wejścia adresowe pamięci. Pamięć zawiera dane (wartości funkcji) określone w tabeli. Zawartość pamięci wynika bezpośrednio z tabeli przejść. Przydzielając bity adresowe i bity danych zgodnie z zasadą że mniej znaczący bit jest po prawej stronie przykładowa zawartość pamięci pod adresami: D(0..3)=0 ; D(4..7)=060H. Schemat US znajduje się na kolejnej stronie.

Układ sterowania

pamięć i rejestr



Rejestr wymaga asynchronicznego przygotowania do pracy –wpisanie stanu początkowego układu sterowania.

Pamięć stała (ROM) powinna być wysterowana w sposób pozwalający na czytanie.

Przykład drugi: Algorytm NWD

- Największym wspólnym dzielnikiem (NWD) dwóch liczb naturalnych dodatnich nazywamy największą liczbę naturalną, która jest jednocześnie dzielnikiem każdej z liczb .

- Algorytm Euklidesa (a>=0, b>0)

```
NWD (a, b)
  if a = 0 return b
  while a ≠ 0
     if a <  b
          b := b − a
     else
          a := a − b
  return b
```

a	b
77	35
42	35
7	35
7	28
7	21
7	14
7	7
0	7

Na podstawie algorytmu zaprojektować układ wykonawczy dla NWD

```
NWD (a, b)
    if a = 0 return b
    while a ≠ 0
        if a <  b
            b := b − a
        else
            a := a − b
    return b
```

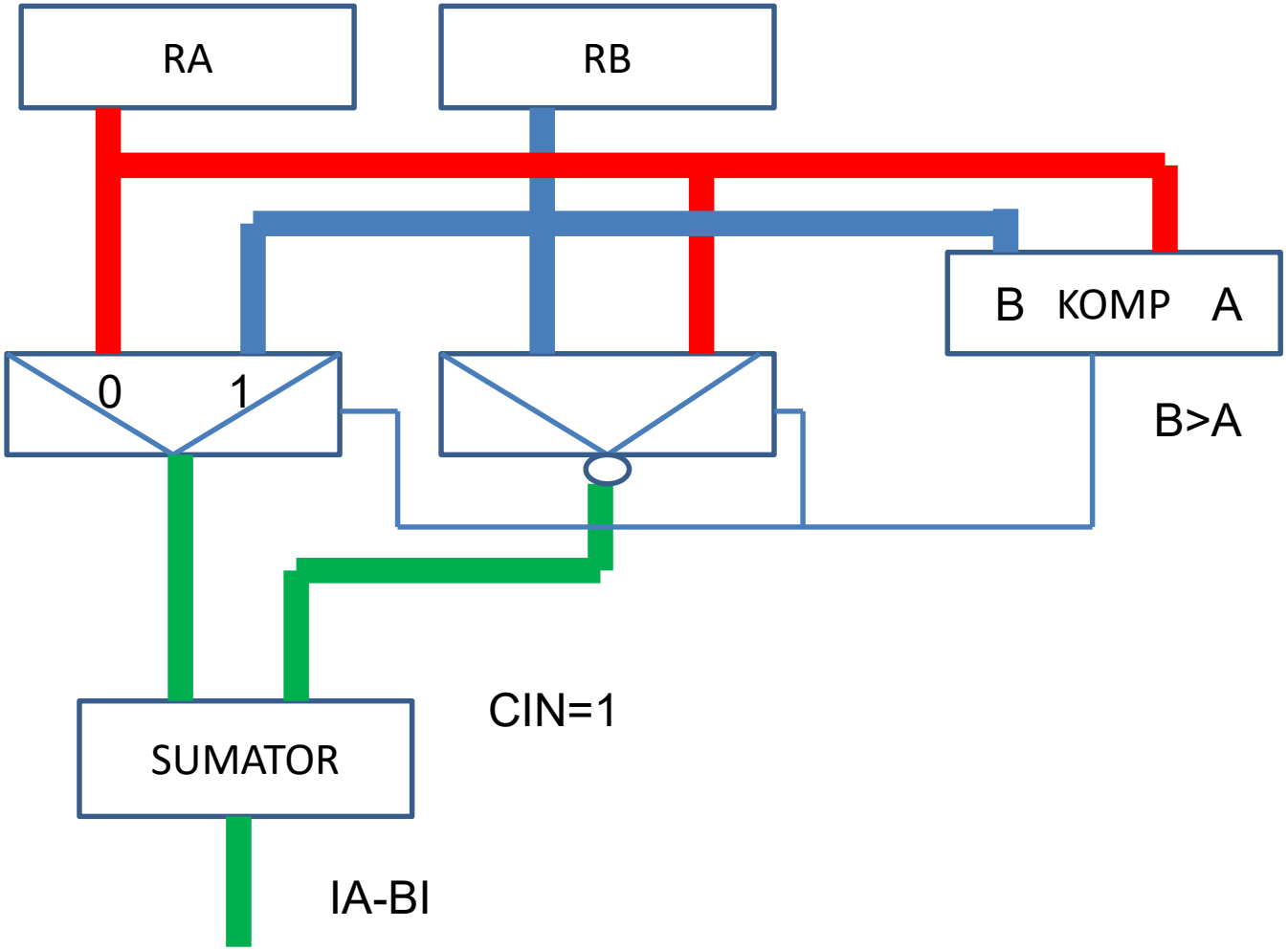
Instrukcje RTL:
R1 ← A
R2 ← B
R1 ← R1- R2
R2 ← R2 - R1
Generacja sygnałów:
- a<b;
- a=0

Elementy składowe układu wykonawczego:

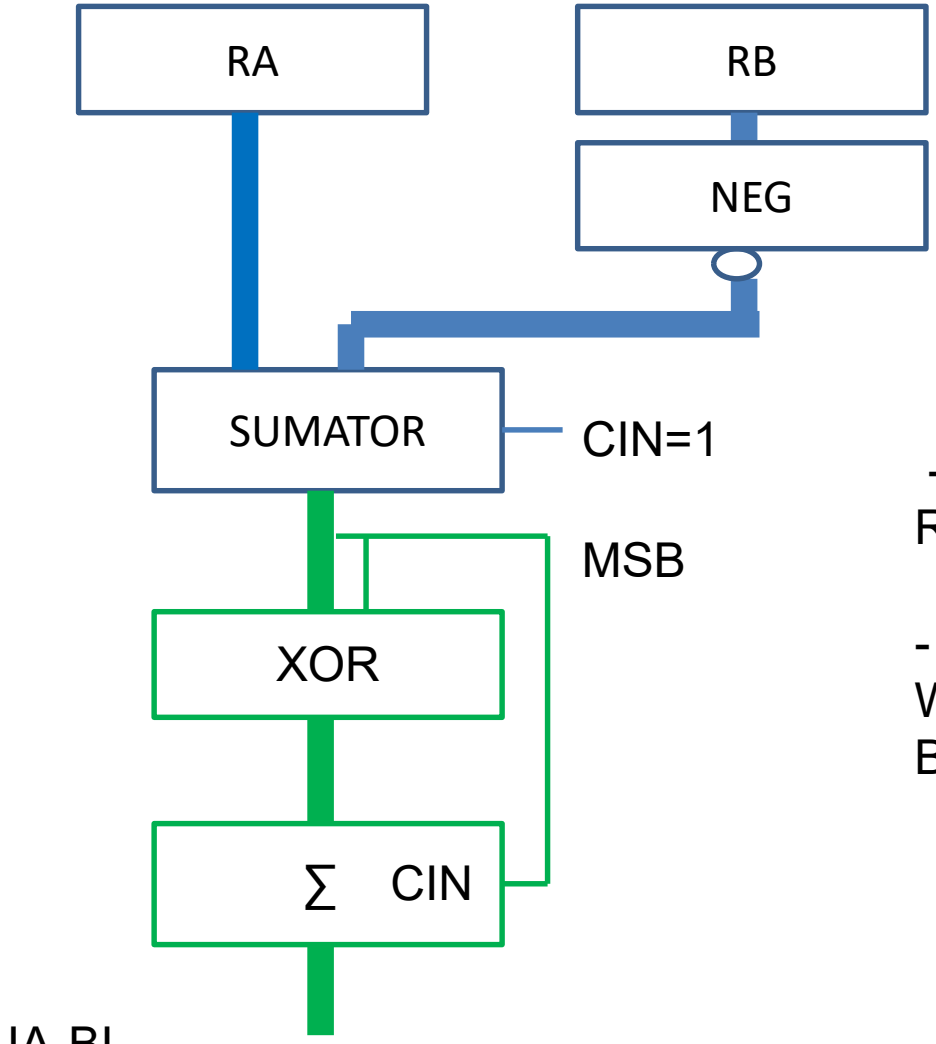
- Rejestry dla a i b
- Komparator a,b
- Test zera dla a
- Sumator (A_{NKB}, B_{NKB}, A_{U2},B_{U2})
- Multipleksery
- NKB -> U2

Dzięki warunkowi **a<b** zawsze **a** wcześniej osiągnie wartość 0

Układ dla IA-BI



Układ dla IA-BI



- WYZNACZANIE RÓZNICY

- WYZNACZANIE WARTOŚCI BEZWZGLĘDNEJ

NWD – synteza funkcjonalna w VHDL

największy wspólny dzielnik

Algorytm Euklidesa

```
NWD (a, b)
  if a = 0 return b
  while b ≠ 0
    if a < b
          b := b – a
    else
          a := a – b
  return a
```

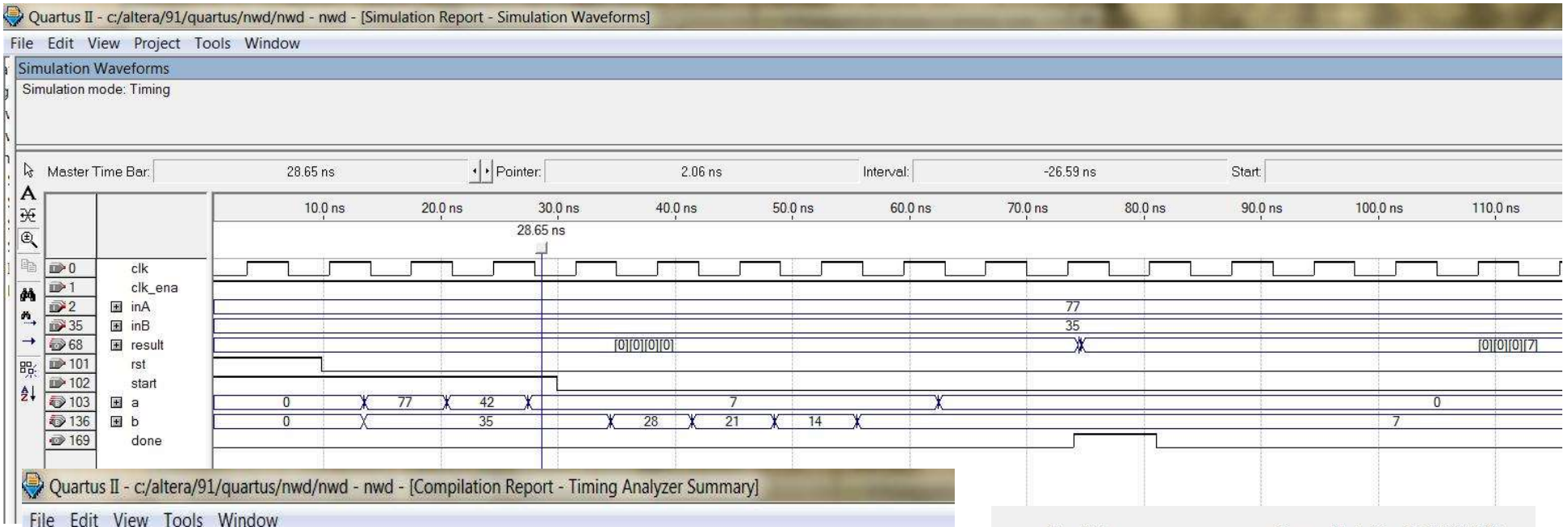
```
if a = 0 then
        result <= std_logic_vector(b);
        working := false; done <= '1';
elsif b = 0 then
        result <= std_logic_vector(a);
        working := false;  done <= '1';
    else
        if a < b then
            b <= b - a;
        else
            a <= a - b;
        end if;
```

NWD – opis funkcjonalny w VHDL
Opis bazujący na algorytmie

```
library ieee;
use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity nwd is
port(
        rst : in std_logic;
        clk : in std_logic;
        start : in std_logic; -- załaduj jeśli skończyłeś pracę
        inA, inB : in std_logic_vector(31 downto 0);
        done : out std_logic; -- wynik gotowy
        result : out std_logic_vector(31 downto 0)
);
end nwd

architecture functional of nwd is
signal a, b : unsigned(31 downto 0);
begin
process(rst, clk)
variable working : boolean;
begin
if rst = '1' then //rst inicjuje przetwarzanie
                        //asynchronicznie
    working := false;
    done  <= '0';
    a <= (others => '0');
    b <= (others => '0');
elsif rising_edge(clk) then
    if not working then // zapobiega przerwaniu
                        //przetwarzania przez start
        done  <= '0';
        if start= '1' then
                working := true;
                a  <= unsigned(inA);
                b  <= unsigned(inB);
        else
                working := false;
        end if;
    else
        if a = 0 then
            result <= std_logic_vector(b);
            working := false; done <= '1';
        elsif b = 0 then
            result <= std_logic_vector(a);
            working := false;  done <= '1';
            else
                if a < b then
                        b <= b - a;
                else
                        a <= a - b;
                end if;
        end if;
    end if;
end process;
end functional;
```

NWD - opis funkcjonalny w VHDL – realizacja Cyclone II



Quartus II - c:/altera/91/quartus/nwd/nwd - nwd - [Compilation Report - Timing Analyzer Summary]

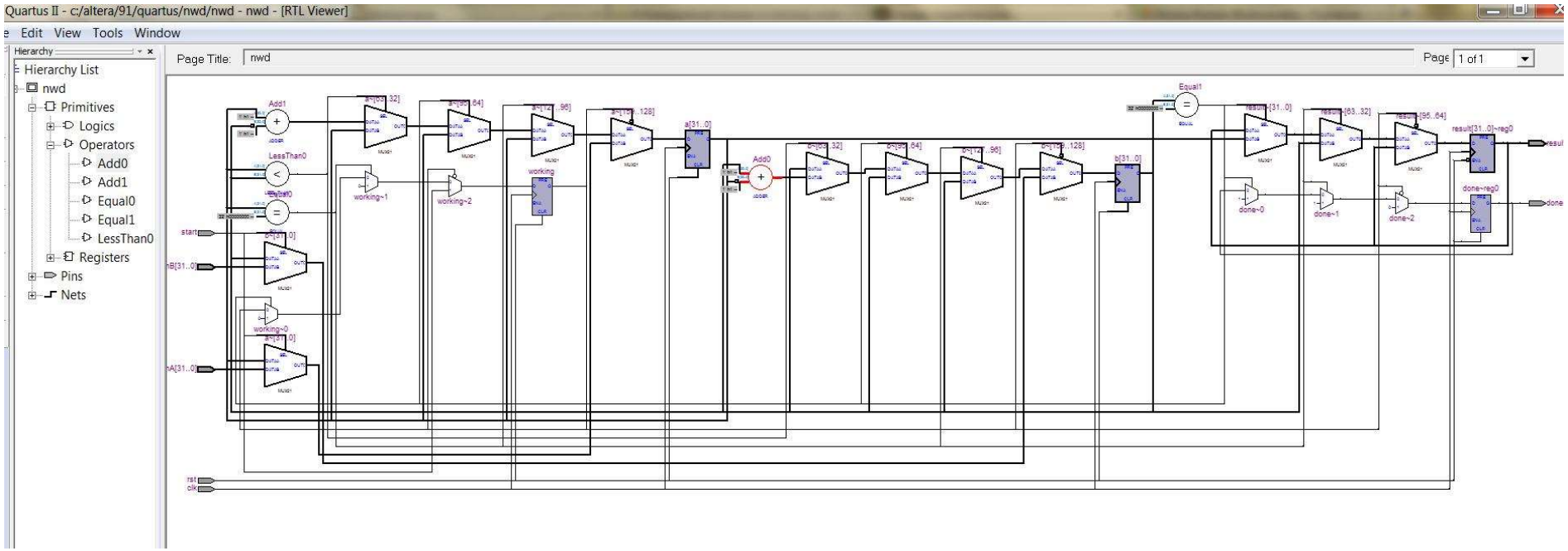
File Edit View Tools Window

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tsu	N/A	None	4.767 ns	inA[28]	a[28]	–	clk	0
2	Worst-case tco	N/A	None	8.147 ns	result[19]~reg0	result[19]	clk	–	0
3	Worst-case th	N/A	None	0.527 ns	inA[0]	a[0]	–	clk	0
4	Clock Setup: 'clk'	N/A	None	151.42 MHz (period = 6.604 ns)	b[0]	b[10]	clk	clk	0
5	Total number of failed paths								0

Flow Status	Successful - Sat Dec 07 13:27:07 2013
Quartus II Version	9.1 Build 222 10/21/2009 SJ Web Edition
Revision Name	nwd
Top-level Entity Name	nwd
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	218 / 33,216 (< 1 %)
Total combinational functions	154 / 33,216 (< 1 %)
Dedicated logic registers	98 / 33,216 (< 1 %)
Total registers	98
Total pins	100 / 475 (21 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

NWD - opis funkcjonalny w VHDL – realizacja w Cyclone II- opis RTL



Układy mikroprogramowane

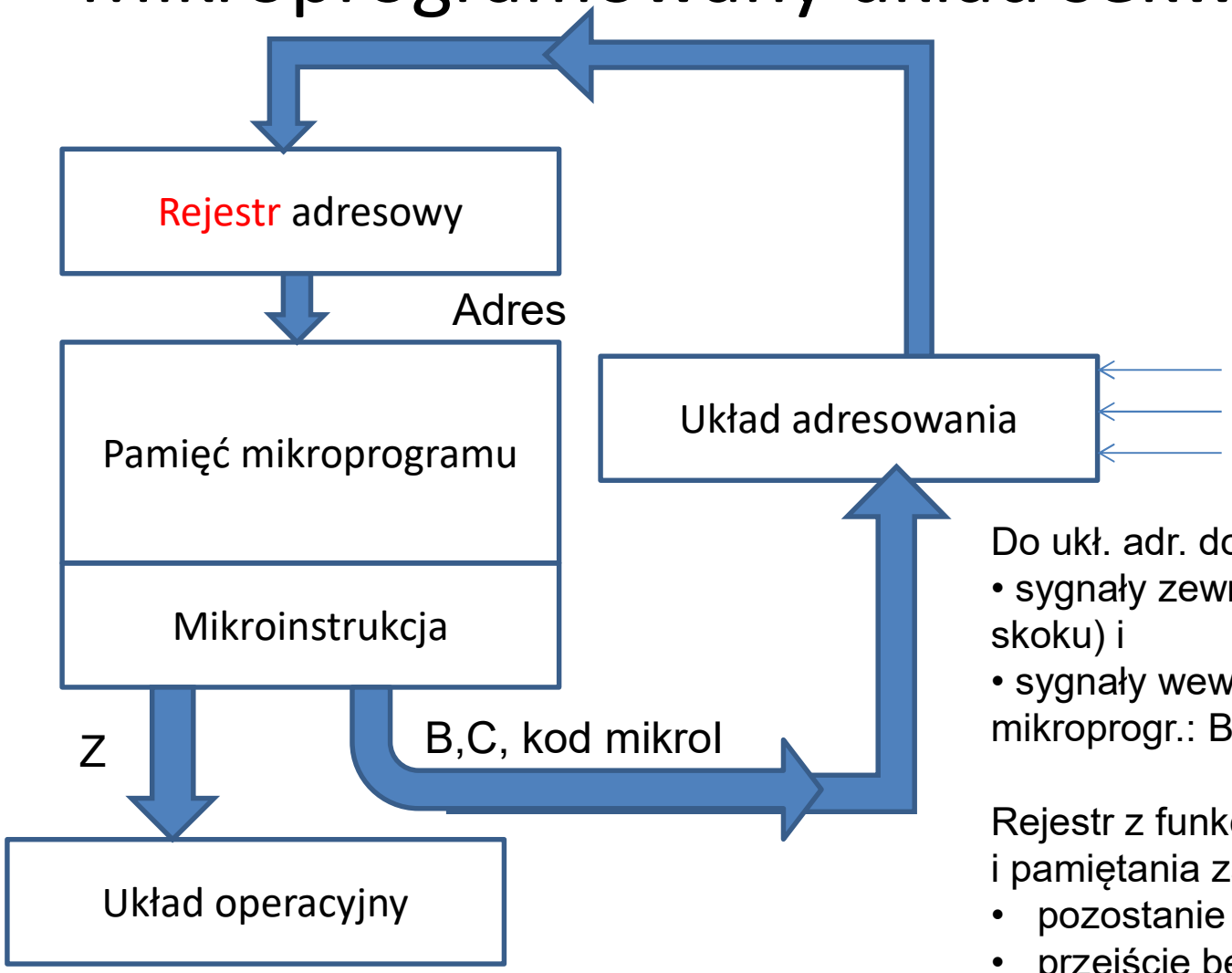
- Technika realizacji układów sterowania polegająca na bezpośredniej transformacji sieci działań (diagramu ASM) na mikroinstrukcje mikroprogramu sterowania.
- Sieć działań automatu umożliwia przyporządkowanie stanów UKŁADU STEROWANIA poszczególnym segmentom sieci działań oraz określenie mikroinstrukcji realizowanych w elementarnym takcie pracy układu sterującego.
- Do podstawowych czynności układu sterującego w stanie A_i należy:
  1. wygenerowanie mikrorozkazu Z_{Ai},
  2. badanie warunku x oraz
  3. określenie stanu **następnego** Ai' (i związanej z nim mikroinstrukcji).

Mikroinstrukcje
koncepcja ogólna

- Moora:
  - A_i: $Z=Z_{Ai}$, if x then $A'=A_j$ else $A'= A_k$
  - Skok warunkowy zależny od wejścia, dwa adresy docelowe (zawartość rozkazu), sterowanie zależne od stanu A_i
- Meale'go
  - A_i: if x then $Z=Z_{A1i}$, $A'=A_j$ else $Z=Z_{A2i}$, $A'= A_k$
  - Skok warunkowy - dwa adresy docelowe (zawartość rozkazu), sterowanie zależne od stanu i wejścia

- Częścią składową mikroprogramowalnego układu sterowania jest pamięć ROM zawierająca mikroprogram pracy całego układu czyli wykaz instrukcji realizowanych w poszczególnych stanach wewnętrznych A_i (stanach określonych adresem mikroinstrukcji A_i)

Mikroprogramowany układ sekwencyjny



Do ukł. adr. doprowadzone są:
- sygnały zewnętrzne (do warunków skoku) i
- sygnały wewnętrzne układu mikroprogr.: B,C,kod instrukcji.

Rejestr z funkcją wpisu równoległego i pamiętania zapewnia:
- pozostanie w stanie,
- przejście bezwarunkowe lub
- przejście warunkowe

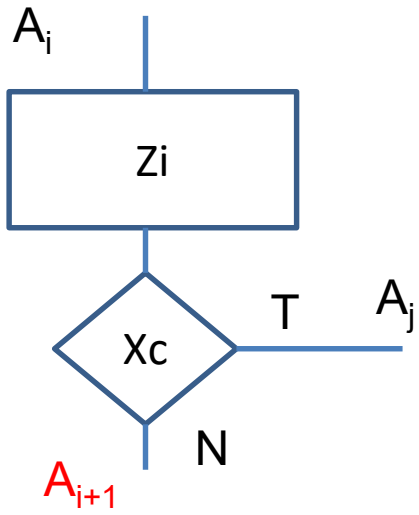
Mikroinstrukcja składa się z:
- pola operacyjnego - Z
- pola adresowego - B i pola warunku – C
- pola kodu mikroinstrukcji S

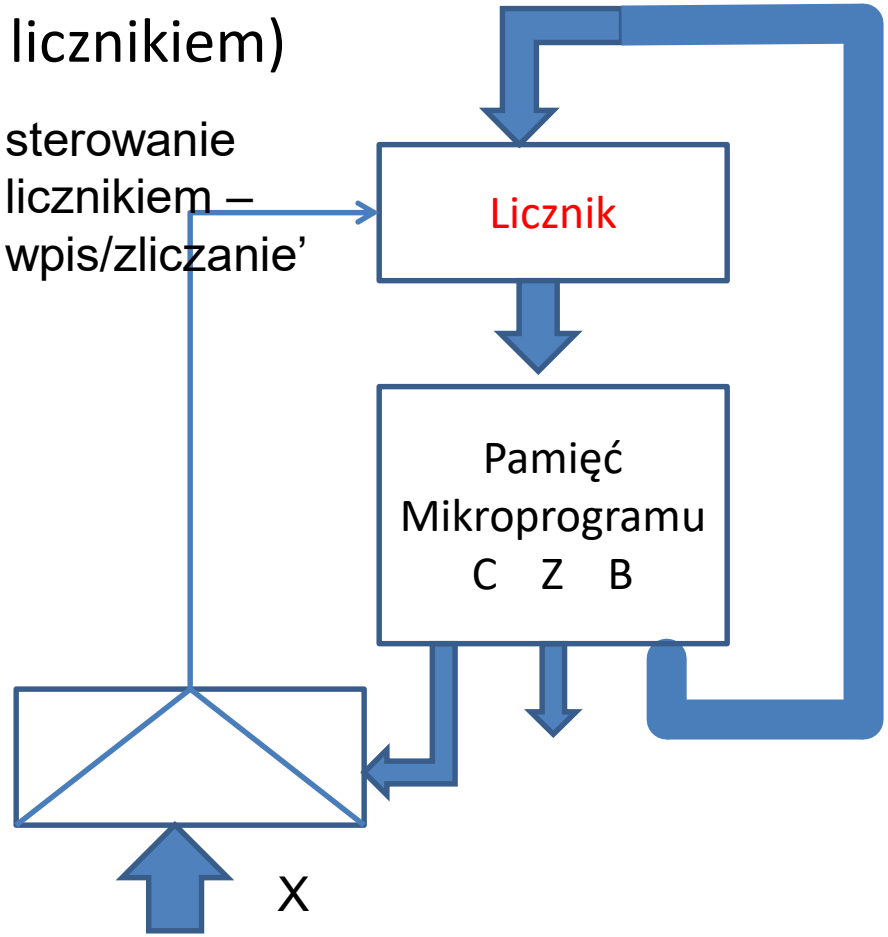
Mikroprogramowany układ sterowania

wer. 2 (z licznikiem)

Zadania układu sterowania:

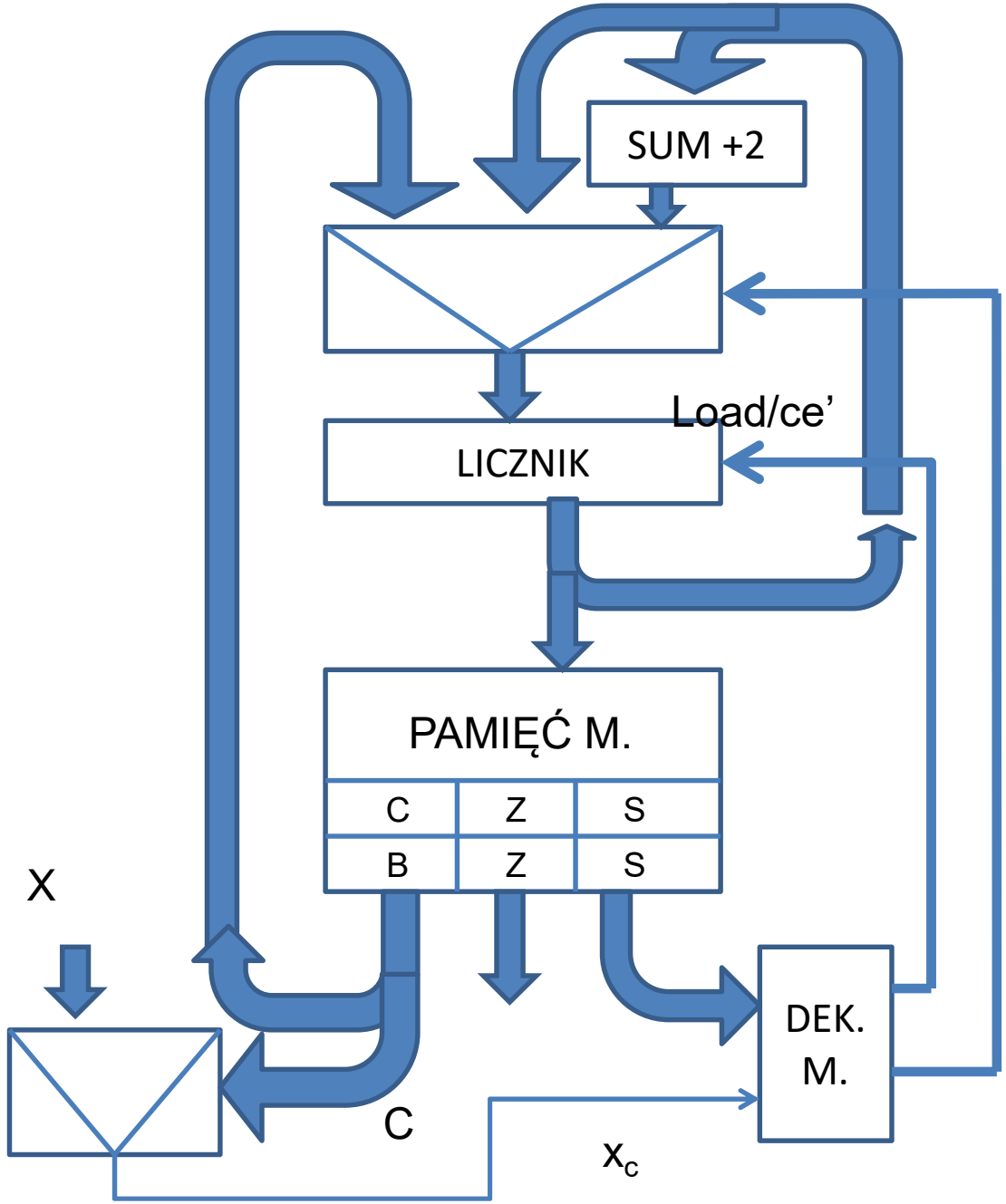
- generacja sterowań,
- wybór warunku i rozejście w mikroprogramie,
- przejście do dowolnego adresu.





W zależności od wartości warunku (pole C rozkazu) następuje przejście do kolejnego stanu (kolejny adres - zgoda na zliczanie licznika) lub skok (zgoda na zapis równoległy licznika) zgodnie z zwartością pola adresowego

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



UKŁAD STERUJĄCY ver 3
Implementowane instrukcje:

Ai :Z, if xc then A'=Ai else A'=Ai+1
---Instrukcja warunkowego przejścia - użyte pola: <C,Z,S>

$A_{i:}$:Z, if x_c then A'=A_{i+1} else A'=A_{i+2}
-- - Instrukcja warunkowego pominięcia kolejnej instrukcji użyte pola <C,Z,S>

$A_{i:}$:Z, A'=A_j --skok bezwarunkowy
użyte pola <A,Z,S>

Pola rozkazowe:
- B pole adresu
- C pole warunku (sterowania)
- Z pole operacyjne
- S pole typu instrukcji

Dekoder mikroinstrukcji w zależności od zakodowanego typu instrukcji (pole S) generuje sygnały sterowania licznikiem (ładowanie, zgoda na zliczanie) i sygnały adresowe multipleksera adresowego.

Inne mikroinstrukcje

Mikroinstrukcje obsługi podprogramu:

Implementacja stosu za pomocą rejestru śladu (RS) do zapisania adresu powrotu z podprogramu

A_i : $RS=A_{i+1}$, $A'=A_j$

-- skok do (wywołanie) podprogramu od adresu Aj z zapisaniem kolejnego (i+1) adresu jako adresu powrotu

A_k : if x_c then $A'=RS$ else $A'=A_{k+1}$

-- warunkowy powrót z podprogramu (skorzystanie z zawartości rejestru śladu)

Mikroinstrukcje realizacji pętli:

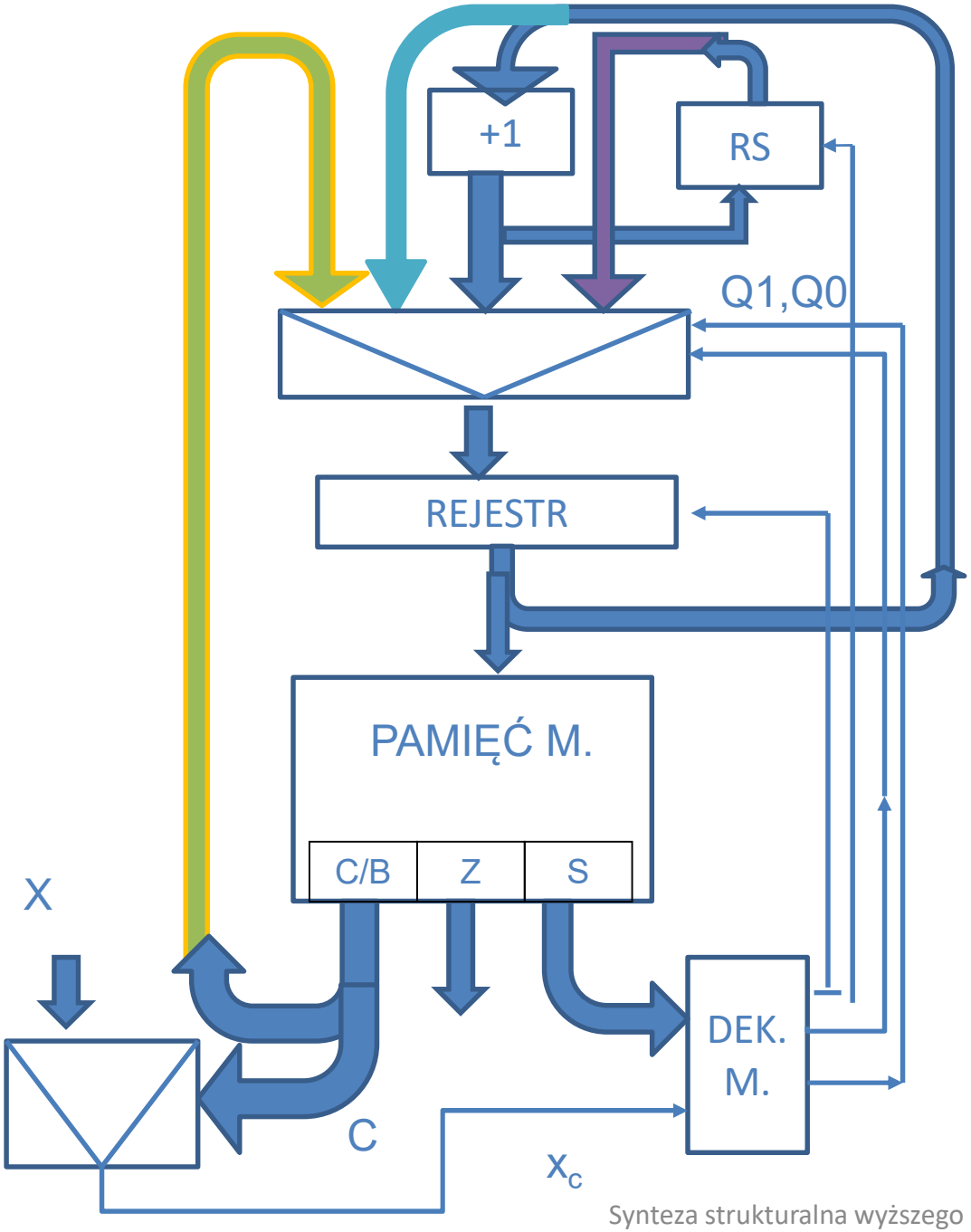
zapamiętanie początku pętli w rejestrze śladu.

A_i : $RS = A_i$, $A'=A_{i+1}$

-- Początek pętli: zapamiętanie początku pętli i przejście do pierwszego rozkazu pętli

A_k : if x_c then $A'= RS$ else $A'=A_{k+1}$

-- warunkowo: powrót na początek pętli lub wykonanie kolejengo rozkazu



UKŁAD STERUJĄCY nr 4

IMPLEMENTOWANE TYPY INSTRUKCJI:

A_i : Z, $A'=A_{i+1}$ --Przejście bezwarunkowe

A_i : Z, $A'=A_j$ – skok bezwarunkowy

A_i : Z, if x_c then $A'=A_{i+1}$, else $A'=A_i$

-- oczekiwanie na spełnienie warunku

A_i : if xc then $A'=A_j$, $RS=A_{i+1}$ else $A'=A_{i+1}$

-- warunkowy skok do podprogramu

A_i : if xc then A'=RS else $A'=A_{i+1}$

-- warunkowy powrót z podprogramu

Multiplekser ma 4 wejścia:

- skoku skoku
- oczekiwania
- przejścia +1(rejestr zamiast licznika)
- powrotu wg rejestru śladu

-Dekoder mikroinstrukcji określa na podstawie pola typu instrukcji i warunku (jeśli ma być uwzględniony) sterowanie bieżącej mikroinstrukcji (zapis RS) i generację kolejnej mikroinstrukcji.