

Podstawy techniki cyfrowej
zima 2017

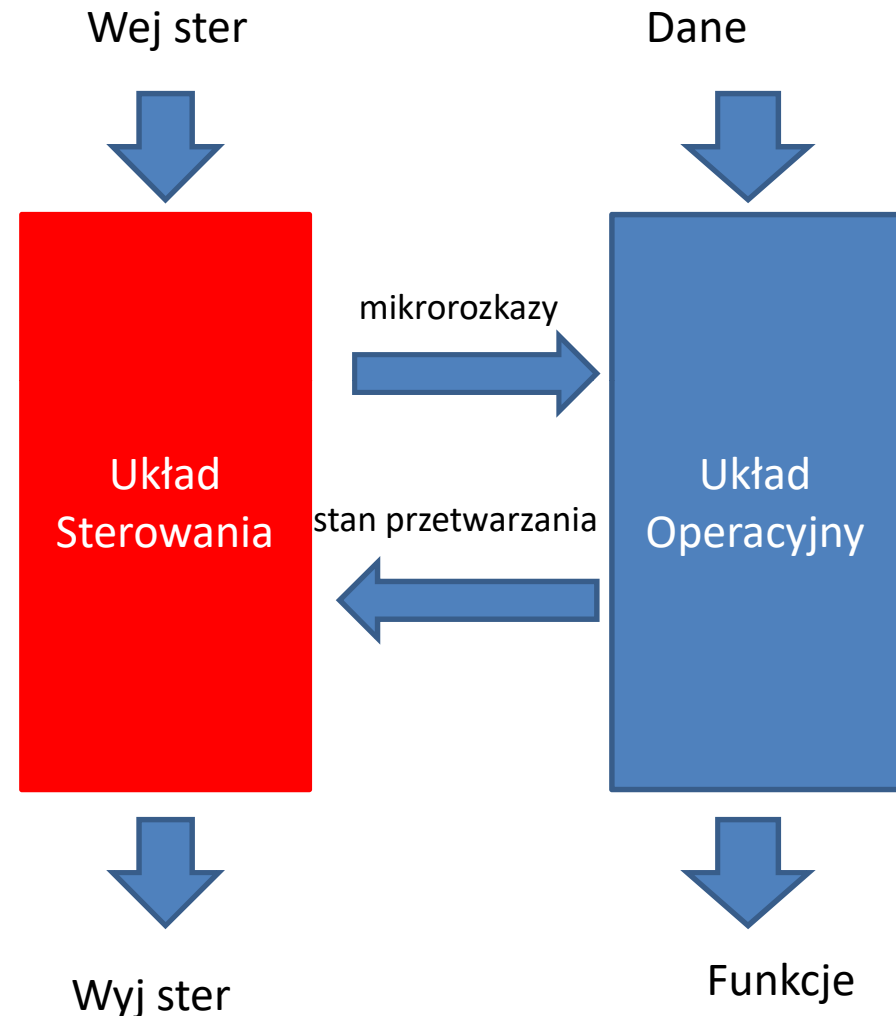
Rafał Walkowiak

Wykład: synteza wyższego poziomu
16.12.2017

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

- Przetwarzaniu w układzie cyfrowym
 - Algorytm dla problemu
 - Synteza układu operacyjnego (przetwarzania 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 cyfrowego

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 przesłań międzyrejestrowych (RTL)

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 kolejność operacji (realizowanych na zawartości rejestrów) 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:

$R_{\text{rezultat}} \leftarrow \text{funkcja}(R_{\text{arg_1}}, R_{\text{arg_2}}, \dots, R_{\text{arg_n}})$

- 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 $R_{\text{rezultat}} \leftarrow f(R_{\text{arg_1}}, R_{\text{arg_2}}, \dots, R_{\text{arg_n}})$ jest realizowana w kolejnych krokach:

1. wraz z narastającym zboczem zegara systemowego rejestry $R_{\text{arg_1}}, R_{\text{arg_2}}, \dots, R_{\text{arg_n}}$ otrzymują nowe wartości,
2. funkcja f oblicza wartość na podstawie zawartości rejestrów $R_{\text{arg_1}}, R_{\text{arg_2}}, \dots, R_{\text{arg_n}}$,
3. wynik obliczeń jest przesyłany na wejście danych rejestru R_{rez} ,
4. kolejne narastające zbocze zegara systemowego powoduje:
 - zapamiętanie nowej wartości w rejestrze R_{rezultat}
 - uaktualnienie zawartości rejestrów $R_{\text{arg_1}}, R_{\text{arg_2}}, \dots, R_{\text{arg_n}}$
5. Kontynuacja pracy od kroku 2

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,
- przesyłanie danych pomiędzy elementami struktury.

W wyniku analizy sieci działań powstaną:

- struktura powiązań elementów cyfrowych realizujących operacje RTL
- 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
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)

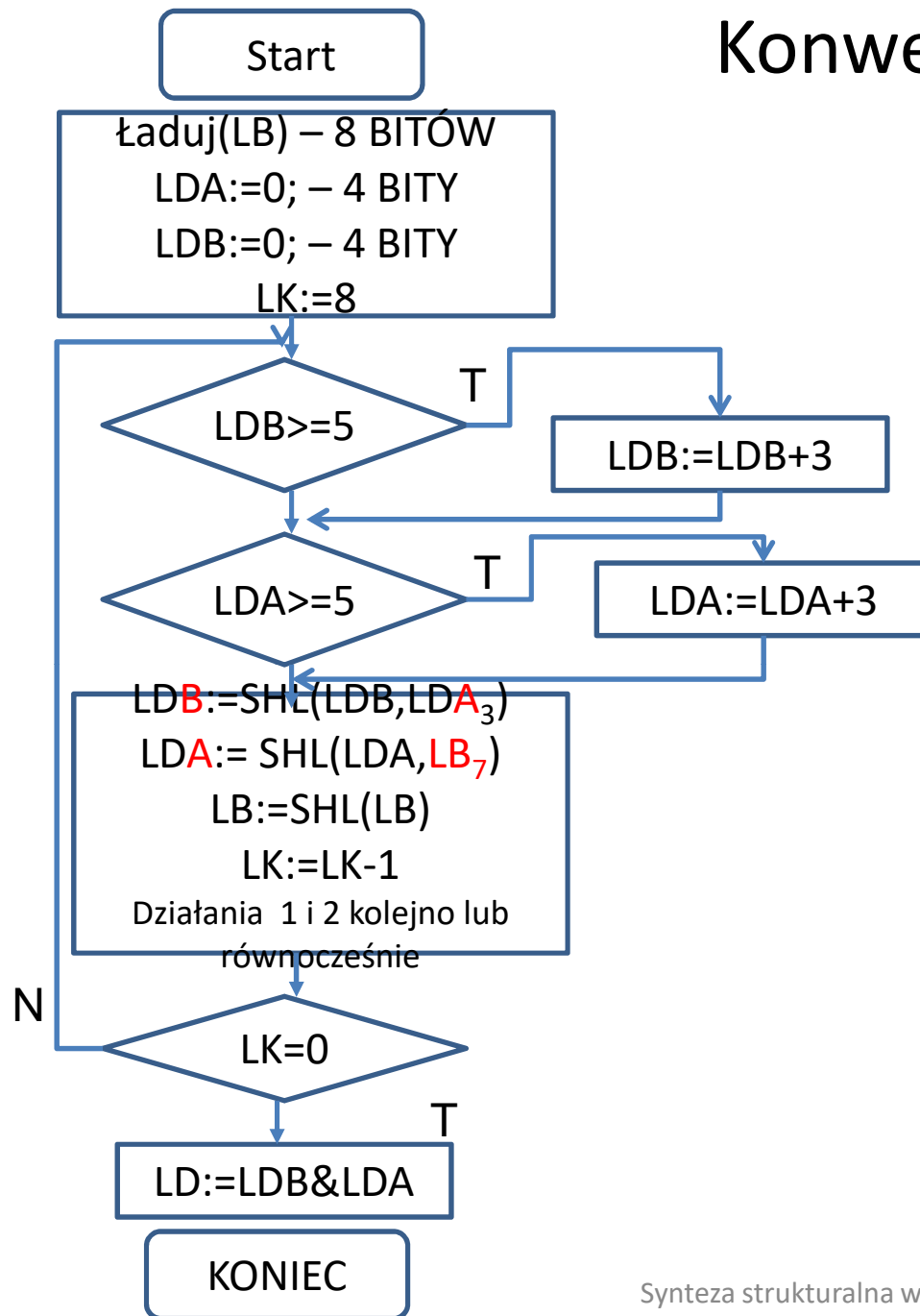
LB – konwertowana liczba
binarna

LDA –młodsza, **LDB** starsza
część wyniku częściowego

LD – wynik BCD

LK – licznik kroków- liczba
bitów konwertowanej liczby

L B₇ -oznacza aktualny najstarszy bit
liczby konwertowanej



Jak zrealizować algorytm w układzie cyfrowym?

Jakie elementy są potrzebne do przetwarzania danych ?

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

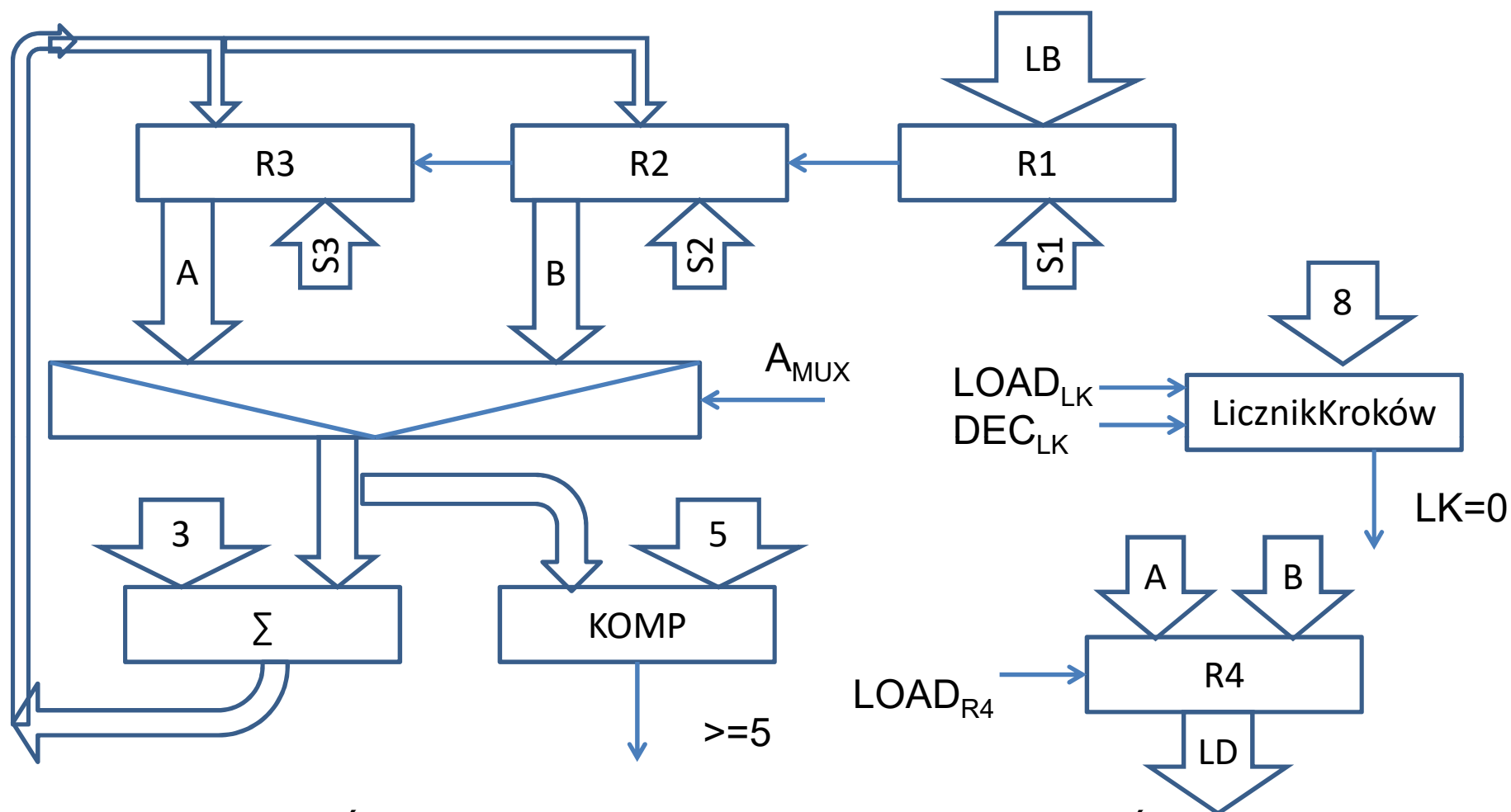
Sumator

Komparator

Licznik

Multiplexer

Schemat układu operacyjnego BIN→BCD



WEKTOR SYGNAŁÓW STERUJĄCYCH ZAWIERA 10 SYGNAŁÓW:
 INDYWIDUALNE STEROWANIE REJESTRAMI R1,R2,R3 (ZEROWANIE, PRZESUW,
 ŁADOWANIE), STEROWANIE LICZNIKIEM, ŁADOWANIE R4, A_{MUX}
 Układy **sekwencyjne** posiadają **niezaznaczone** wejścia zegarowe

Układ sterujący

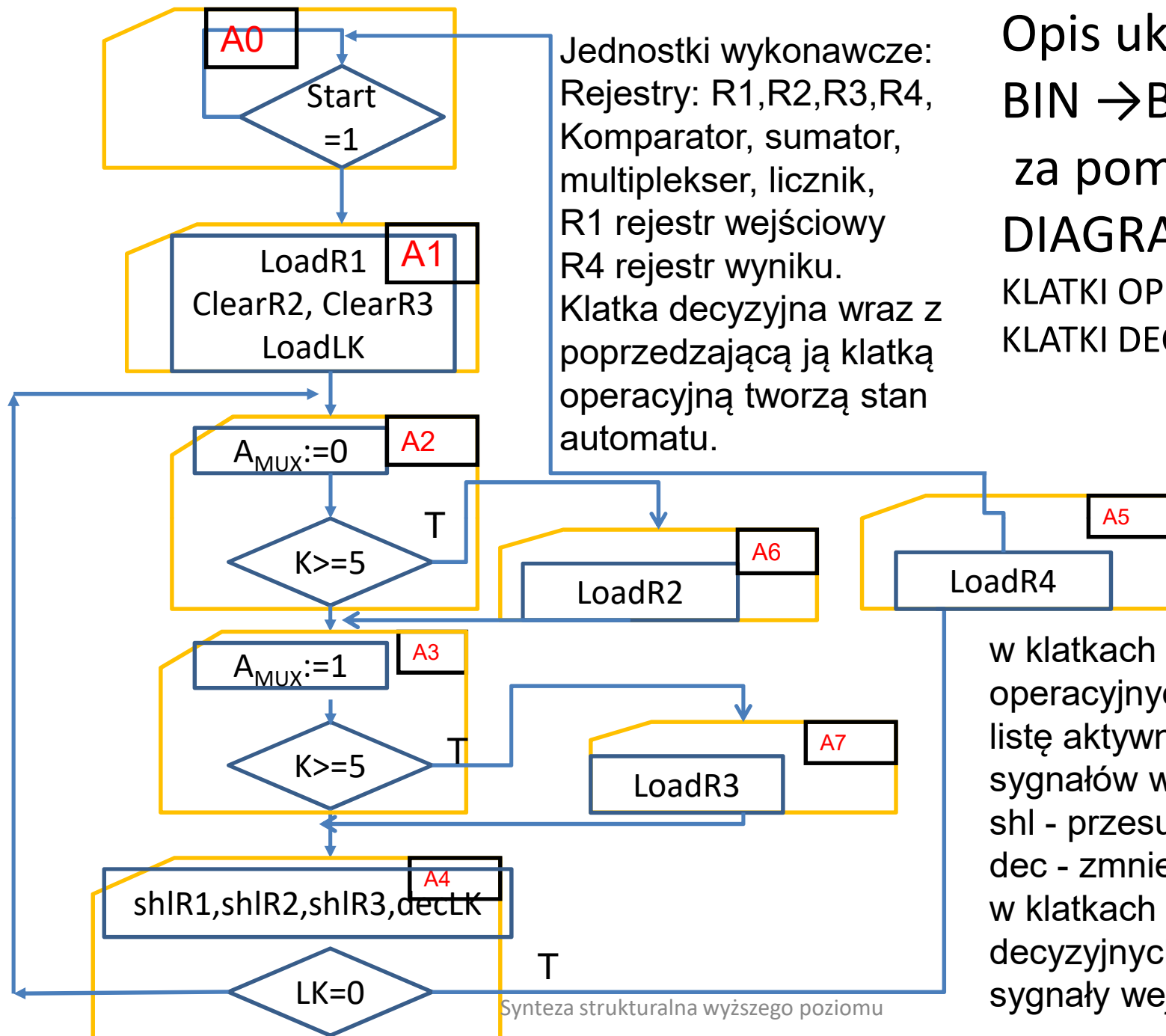
- Nomenklatura: układ sekwencyjny, automat, maszyna stanów, automat skończony (FSM).
- Metoda algorytmicznego układu sekwencyjnego to metoda ASM – użycie diagram ASM (mniej sformalizowany graf stanów).

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 opisu 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.



Opis układu
 BIN → BCD
 za pomocą
 DIAGRAMU ASM
 KLATKI OPERACYJNE I
 KLATKI DECYZYJNE

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

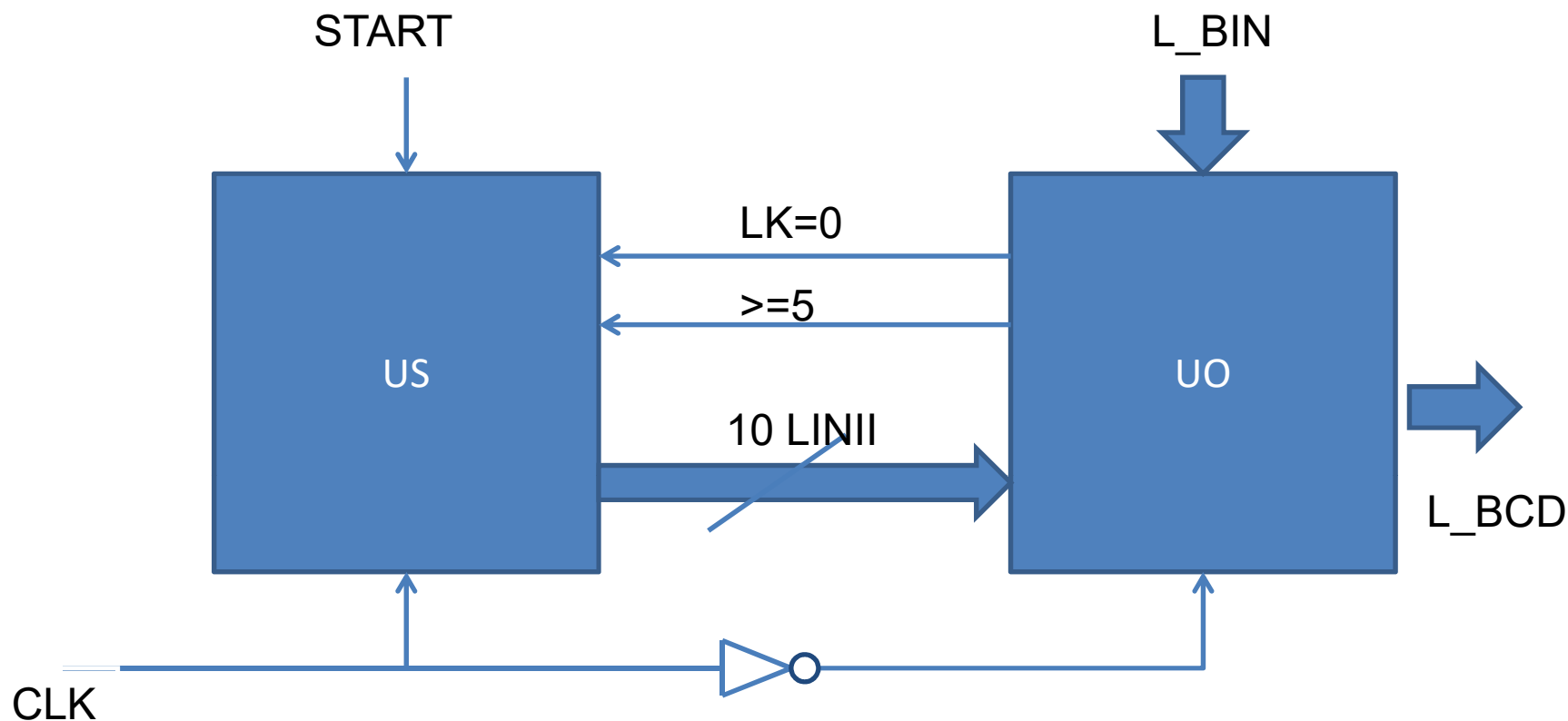
TABLICA PRZEJŚĆ AUTOMATU STERUJĄCEGO BIN → BCD

nazwa	STAN	W	E	J	S	C	I	A	
znaczenie		000	001	011	010	110	111	101	100
START	A0	A0	A0	A0	A0	A1	A1	A1	A1
LOAD	A1	A2	A2	A2	A2	A2	A2	A2	A2
KOMPA	A2	A3	A3	A6	A6	A6	A6	A3	A3
KOMPB	A3	A4	A4	A7	A7	A7	A7	A4	A4
SHL	A4	A2	A5	A5	A2	A2	A5	A5	A2
WYNIK	A5	A0	A0	A0	A0	A0	A0	A0	A0
ADD A	A6	A3	A3	A3	A3	A3	A3	A3	A3
ADD B	A7	A4	A4	A4	A4	A4	A4	A4	A4

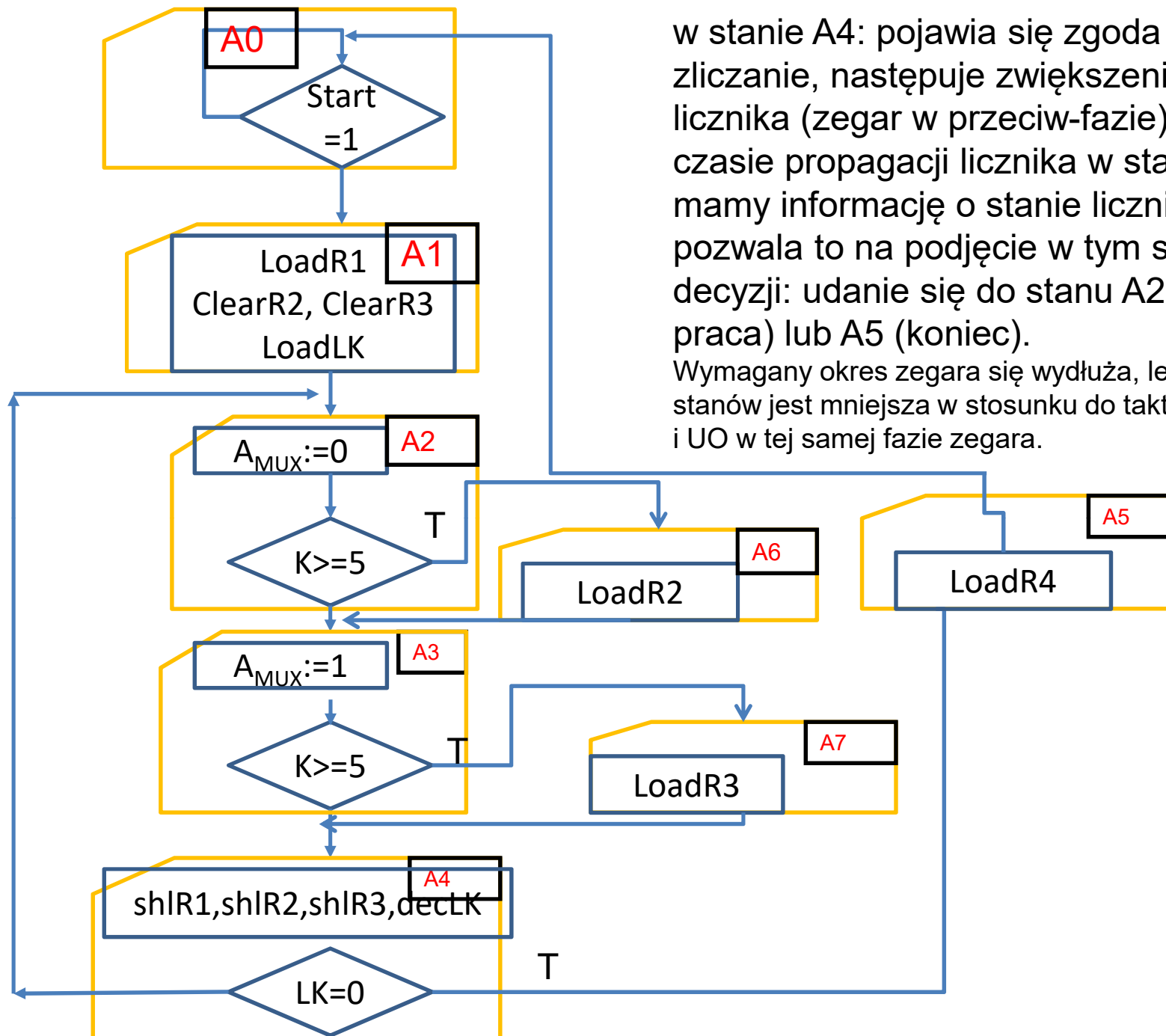
Wektor wejść (zawierstan przetwarzania) a kolejno (od MSB) : **START, K>=5, LK=0**

Wektor wyjść zawiera 10 sygnałów sterujących aktywowanych zgodnie z informacjami zawartymi w diagramie ASM.

STRUKTURA I SYGNAŁY WSPÓŁPRACY MODUŁÓW UKŁADU KONWERSJI BIN-BCD



CLK
Ważne: Synchronizowanie układów **przeciwnymi fazami** tego samego sygnału zegarowego pozwala na realizację w aktualnym stanie operacji na rejestrach zgodnie z zasadami określonymi za pomocą **sygnałów** sterujących, **ustalonych** w aktualnym stanie automatu. verte

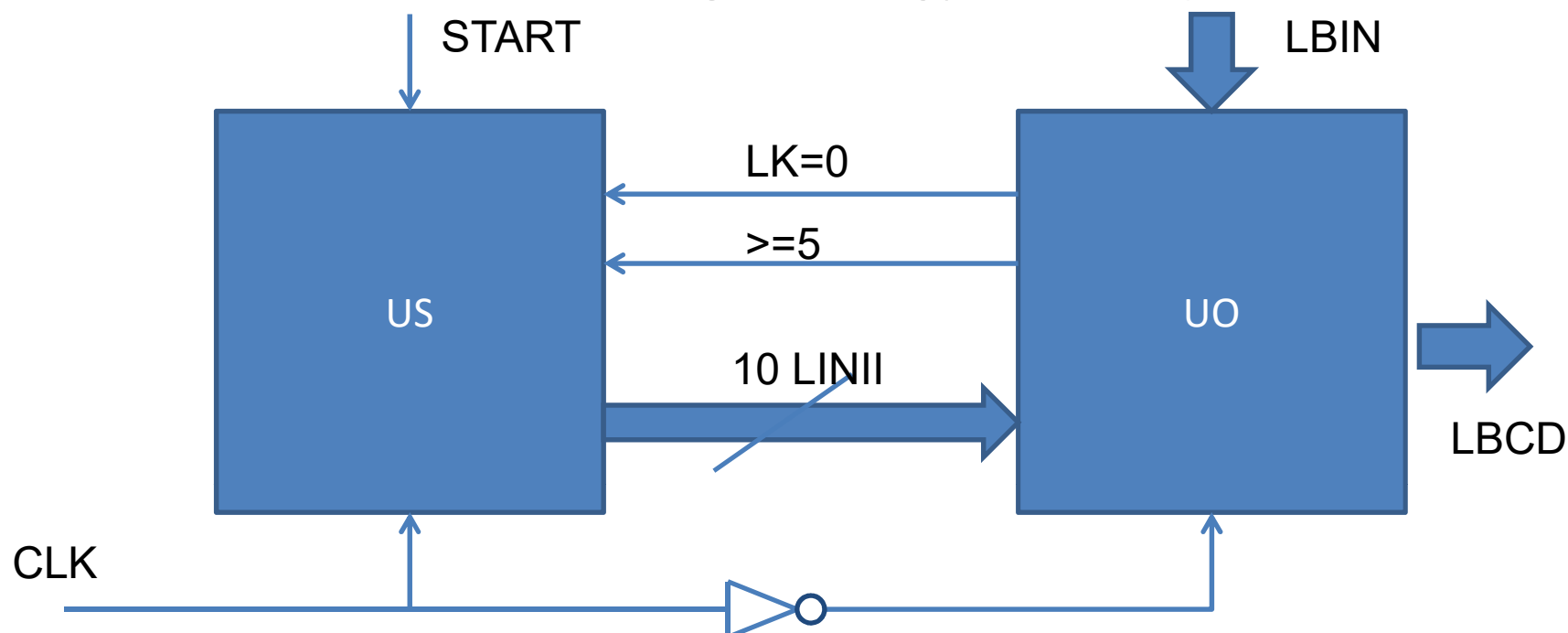


w stanie A4: pojawia się zgoda na zliczanie, następuje zwiększenie licznika (zegar w przeciw-fazie), po czasie propagacji licznika w stanie A4 mamy informację o stanie licznika - pozwala to na podjęcie w tym stanie decyzji: udanie się do stanu A2 (dalsze praca) lub A5 (koniec).

Wymagany okres zegara się wydłuża, lecz liczba stanów jest mniejsza w stosunku do taktowania US i UO w tej samej fazie zegara.

STRUKTURA I SYGNAŁY WSPÓŁPRACY MODUŁÓW

UKŁADU KONWERSJI BIN-BCD



KOLEJNOŚĆ DZIAŁAŃ W SYSTEMIE:

1. ZBOCZE A – NOWY STAN AUTOMATU (na podstawie stanu przetwarzania) - NOWE STEROWANIE DO UO,
2. PROPAGACJA SYGNAŁÓW W UO,
3. ZBOCZE B - ZAPIS WYNIKU PRZETWARZANIA W REJESTRACH, WYKONANIE OPERACJI NA DANYCH,
4. PROPAGACJA STANU PRZETWARZANIA DO US.
5. POWRÓT DO KROKU 1

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 ($b > 0$)

NWD (a, b)

if $a = 0$ return b

while $a \neq 0$

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 $\neq$ 0

if a < b

b := b - a

else

a := a - b

return b

Potrzebne 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 $\rightarrow$ U2

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

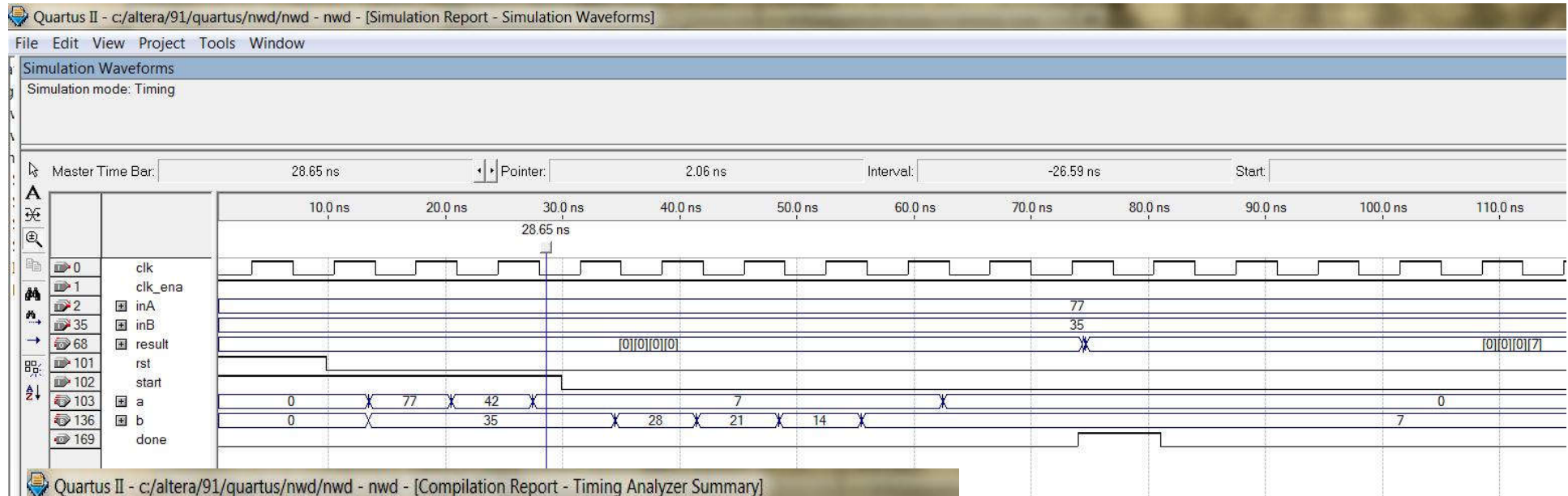
```
library ieee;
use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity nwd is
port(
    rst : in std_logic;
    clk : in std_logic;
    start : in std_logic; -- załaduj jeśli skończyłeś pracę
    inA, inB : in std_logic_vector(31 downto 0);
    done : out std_logic; -- wynik gotowy
    result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

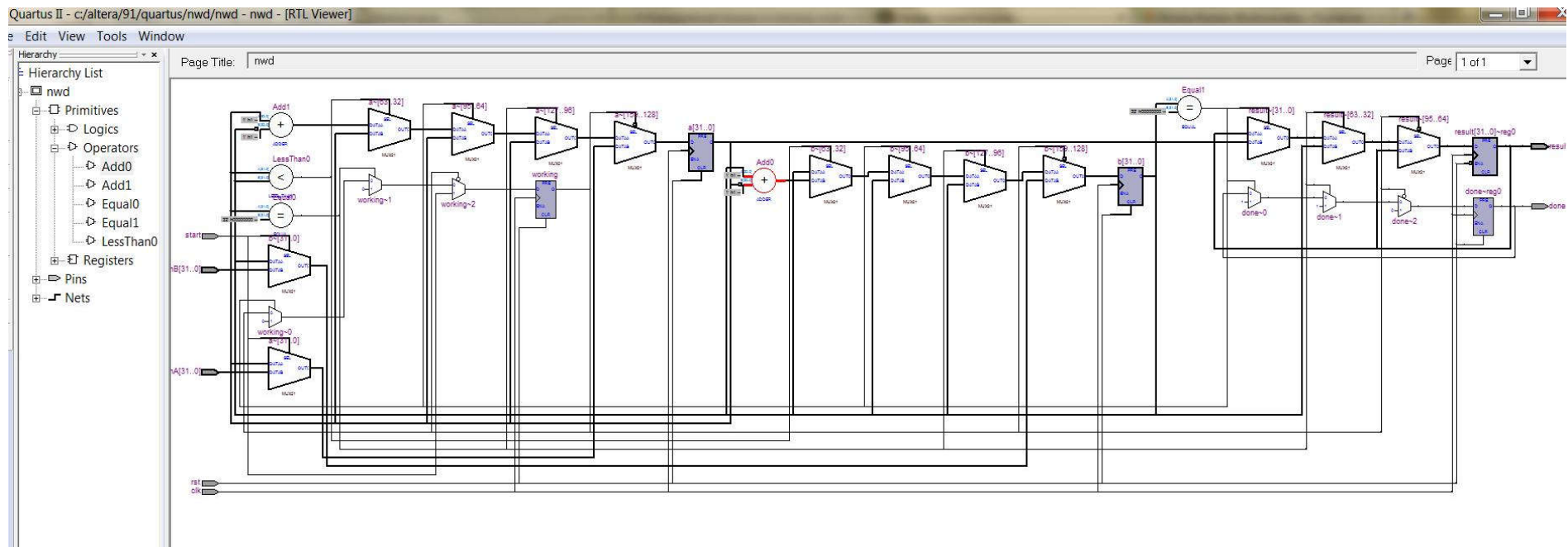
File Edit View Tools Window

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

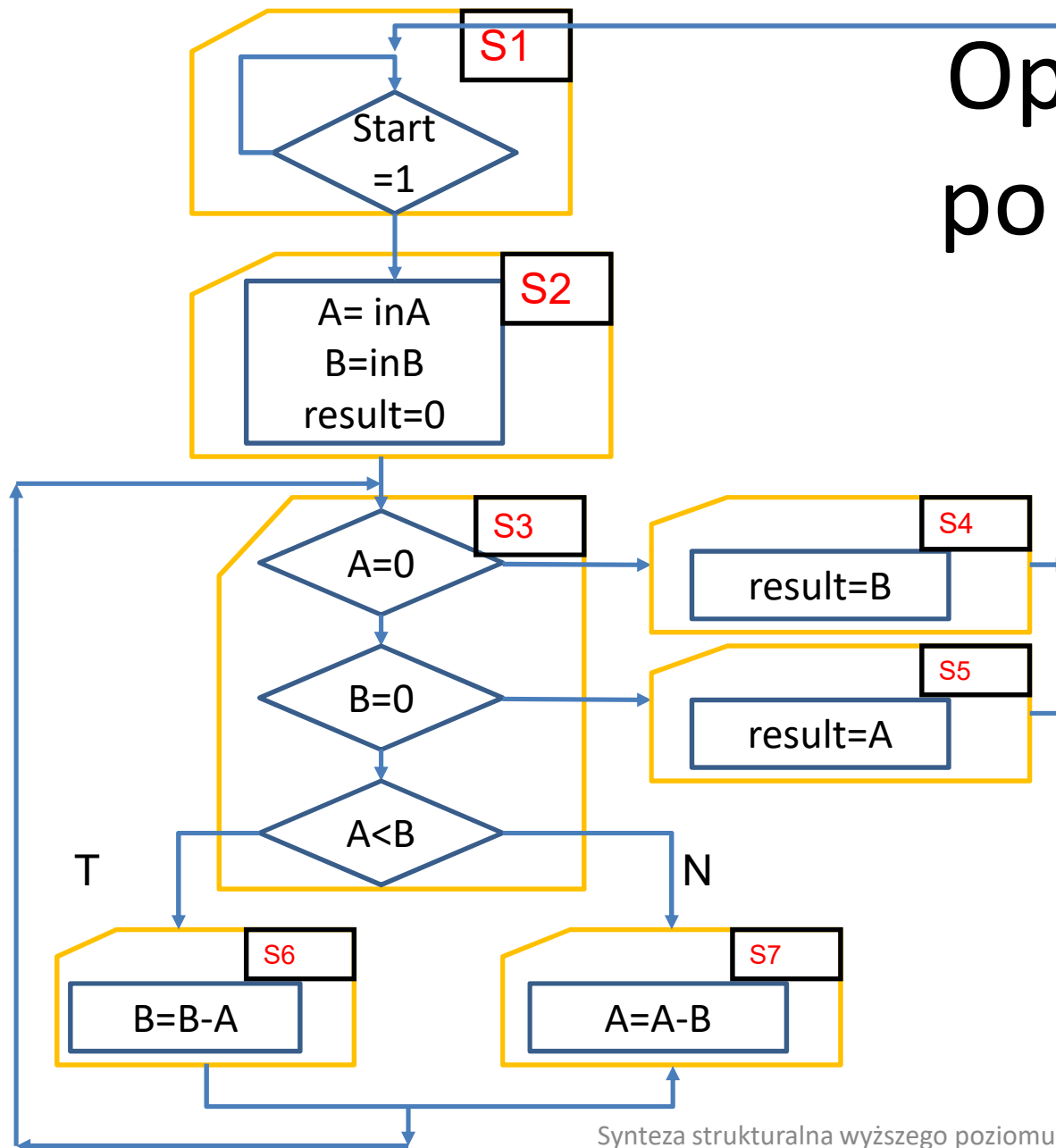


Metoda projektowania UC przy użyciu diagramu ASMD

(diagramu ASM ze zintegrowaną ścieżką danych)

- Brak podziału na część operacyjną i sterującą układu cyfrowego.
- Operacje RT są zintegrowane z opisem automatu sterującego.
- Operacje RT przedstawione są bezpośrednio na diagramie.
- W klatkach decyzyjnych testuje się wartości rejestrów reprezentujących zmienne (a nie wartości **sygnałów** dostarczonych z układu wykonawczego).

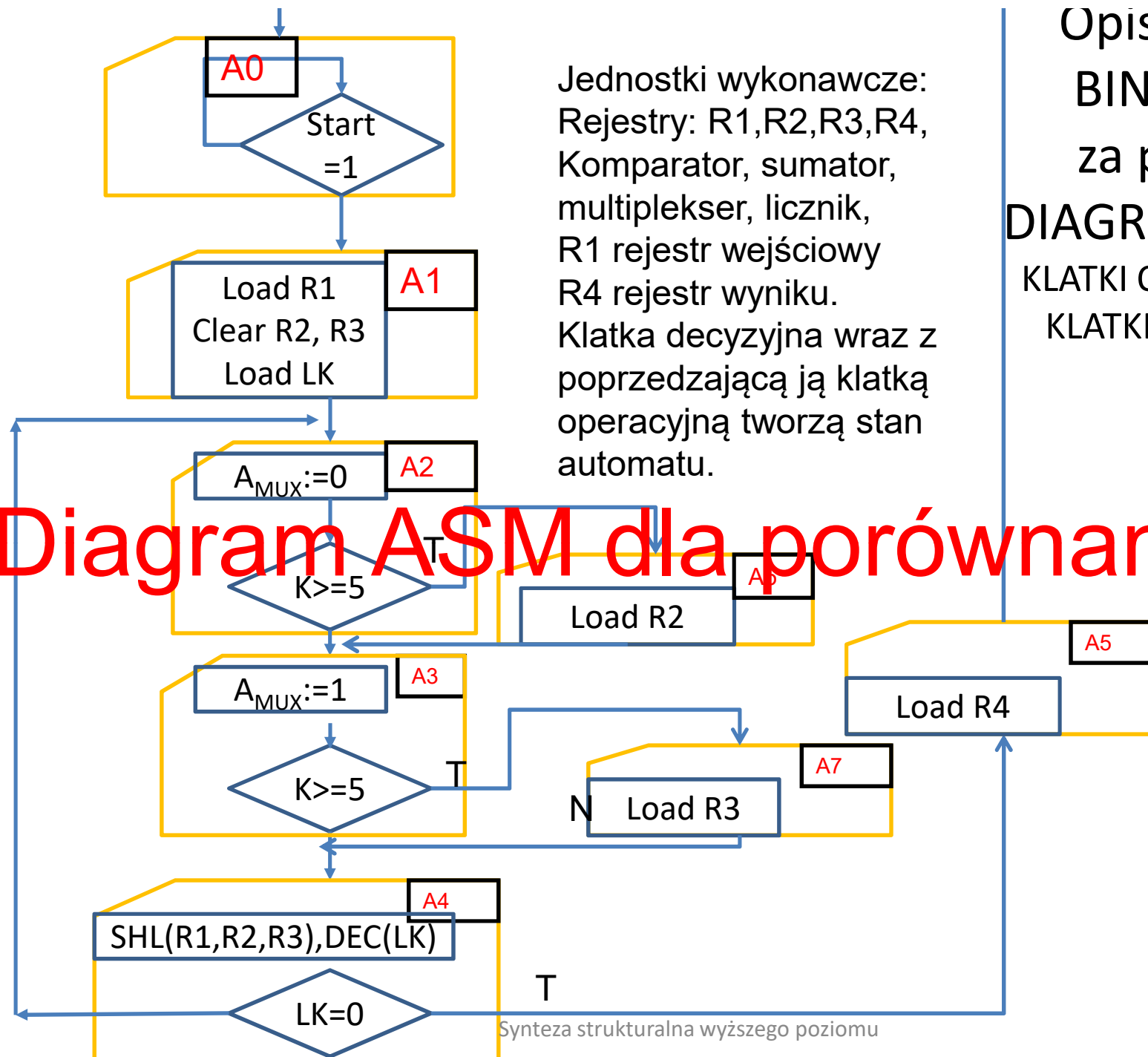
Opis układu za pomocą **ASMD**



Synteza strukturalna wyższego poziomu

Stan	
Brak działania – nop	S1
Ładuj dane do rejestrów - load	S2
Test warunków (wartości rej) - test	S3
Zapisz wynik B (przepisz wart. rej) WRB	S4
Zapisz wynik A (przepisz wart. rej) WRA	S5
Zmniejsz B (wartość rej) DECB	S6
Zmniejsz A (wartość rej) DECA	S7

Diagram ASM dla porównania



Jednostki wykonawcze:
 Rejestry: R1,R2,R3,R4,
 Komparator, sumator,
 multiplekser, licznik,
 R1 rejestr wejściowy
 R4 rejestr wyniku.
 Klatka decyzyjna wraz z
 poprzedzającą ją klatką
 operacyjną tworzą stan
 automatu.

Opis układu
 BIN → BCD
 za pomocą
 DIAGRAMU ASM
 KLATKI OPERACYJNE I
 KLATKI DECYZYJNE

NWD- RTL

```
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;  
    clk_ena   : in std_logic;  
    start    : in std_logic;  
    inA, inB  : in std_logic_vector(31 downto 0);  
    done     : out std_logic;  
    result    : out std_logic_vector(31 downto 0)  
);  
end nwd;
```

architecture RTL of nwd is

```
type STATE_TYPE is (nop, load, test, dec_a, dec_b, WRA,  
                    WRB);  
signal state_reg, state_next : STATE_TYPE;  
  
signal a_reg, a_next      : unsigned(31 downto 0);  
signal b_reg, b_next      : unsigned(31 downto 0);  
signal result_reg, result_next : unsigned(31 downto 0);  
signal done_reg, done_next : std_logic;  
begin
```

```
process(rst, clk) -- Proces modelujący rejestry zmiennych algorytmu  
begin  
    if rst = '1' then  
        a_reg      <= (others => '0');  
        b_reg      <= (others => '0');  
        result_reg  <= (others => '0');  
        done_reg    <= '0';  
    elsif rising_edge(clk) then  
        if clk_ena = '1' then  
            a_reg    <= a_next;  
            b_reg    <= b_next;  
            result_reg <= result_next;  
            done_reg  <= done_next;  
        end if;  
    end if;  
end process;
```

```
process(rst, clk) -- Proces modelujący rejestr stanu automatu ster.  
begin  
    if rst = '1' then  
        state_reg <= idle;  
    elsif rising_edge(clk) then  
        if clk_ena = '1' then  
            state_reg <= state_next;  
        end if;  
    end if;  
end process;
```

```
process(state_reg, start, inA, inB, a_reg, b_reg, result_reg) --
    opis przetwarzania danych w stanach – OPIS
    REALIZOWANYCH W STANACH FUNKCJI
```

```
begin
    done_next <= '0';
    a_next    <= a_reg;
    b_next    <= b_reg;
    result_next <= result_reg;
    case state_reg is
        when nop =>
            if start = '1' then
                state_next <= load;
            else
                state_next <= nop;
            end if;
        when load =>
            a_next    <= unsigned(inA);
            b_next    <= unsigned(inB);
            result_next <= (others => '0') ;
            state_next <= test;
        when test =>
            if a_reg = 0 then
                state_next <= WRB;
            elsif b_reg = 0 then
                state_next <= WRA;
            else
                if a_reg < b_reg then
                    state_next <= decb;
                else
                    state_next <= deca;
                end if;
            end if;
        end if;
    end case;
```

NWD- RTL

```
when deca =>
    a_next    <= a_reg - b_reg;
    state_next <= test;
when decb =>
    b_next    <= b_reg - a_reg;
    state_next <= test;
when WRB =>
    result_next <= unsigned(b_reg);
    done_next  <= '1';
    state_next <= nop;
when WRA =>
    result_next <= unsigned(a_reg);
    done_next  <= '1';
    state_next <= nop;
when others =>
    state_next <= nop;
end case;
end end process;

done    <= done_reg;
result  <= std_logic_vector(result_reg);
end RTL;
```

Proces pozwala na przygotowanie **kolejnego** (next) stanu układu (bez wejścia do niego) i **nowych** (next) wartości rejestrów na podstawie wykonanych w stanie operacji (odejmowanie, podstawianie itp.) możliwe wielokrotne przypisanie wartości (por. lista czułości) uwzględniające czas propagacji układu wykonawczego (ważne ostatnie przypisanie). Z wartości *_next korzystają pozostałe procesy w momencie synchronizacji (z boczem CLK) – wyznaczenie nowego stanu – nowy stan wyznaczany w oparciu o proponowane wartości (*next) rejestrów, których wartości tym samym zboczem (synchronicznie) są uaktualniane.

DASMD* a VHDL

- Projektowanie z wykorzystaniem ASMD pozwala na bezpośrednią (bez określenia struktury układu wykonawczego i sygnałów sterujących) realizację algorytmu zadanego schematem blokowym jako systemu cyfrowego.
- Proste operacje RT z DASMD mogą być realizowane jako **operatory języka VHDL**.
- Złożone operacje RT mogą być realizowane jako **niezależne moduły sprzętowe** i wykorzystane w postaci komponentów projektowanego systemu cyfrowego.

* Diagram algorytmicznego układu sekwencyjnego ze zintegrowaną ścieżką danych

Układy mikroprogramowane

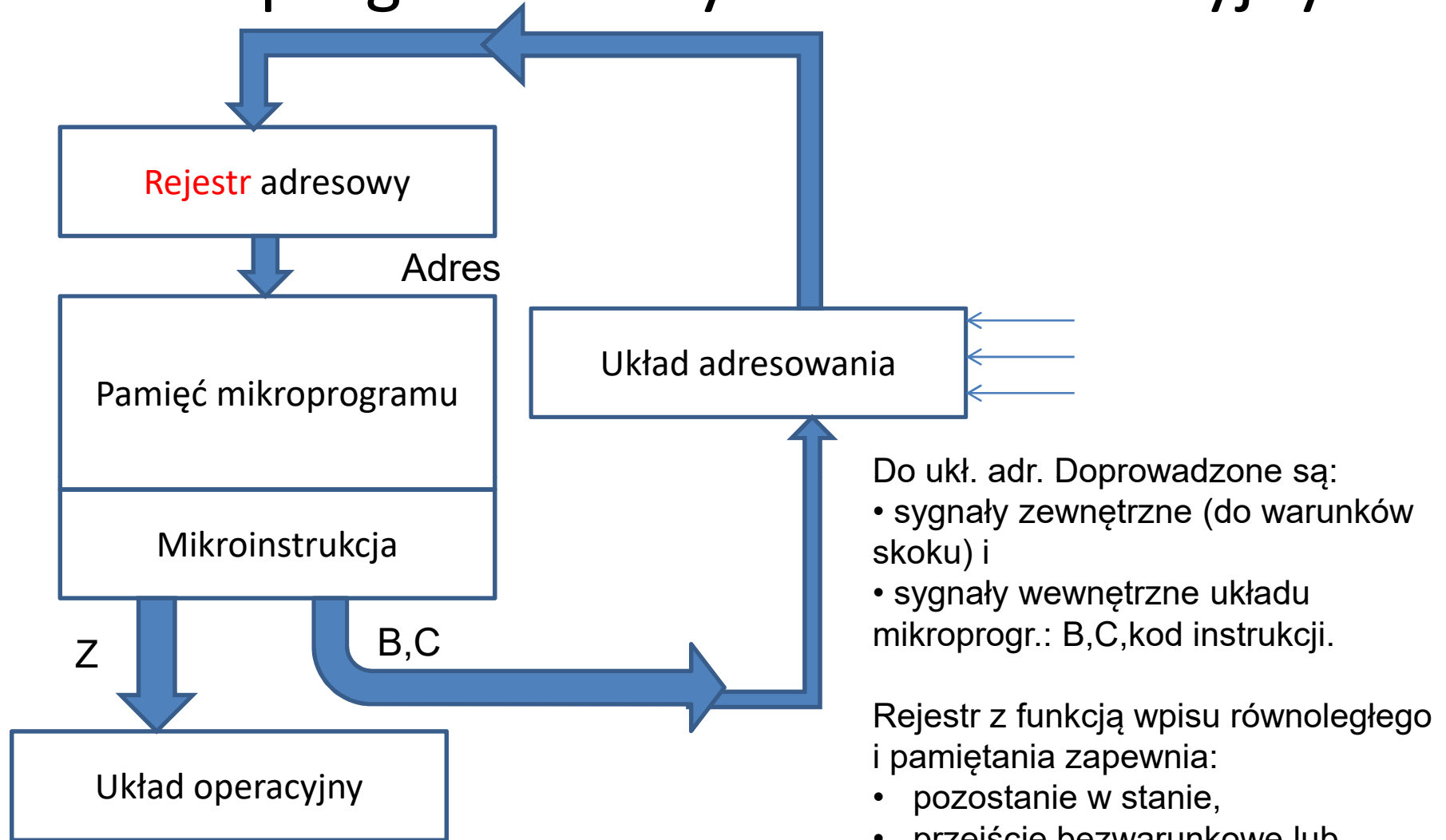
- Technika realizacji układów sterowania polegająca na bezpośredniej transformacji sieci działań (diagramu ASM) na mikroinstrukcje mikroprogramu sterowania.
- Sieć działań automatu umożliwia przyporządkowanie stanów UKŁADU STEROWANIA poszczególnym segmentom sieci działań oraz określenie mikroinstrukcji 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** A_i' (i związanej z nim mikroinstrukcji).

Mikroinstrukcje

koncepcja ogólna

- Moora:
 - $A_i: Z=Z_{A_i}, \text{ if } x \text{ then } A'=A_j \text{ else } A'=A_k$
 - Skok warunkowy zależny od wejścia, dwa adresy docelowe (zawartość rozkazu), sterowanie zależne od stanu A_i
- Meale'go
 - $A_i: \text{ if } x \text{ then } Z=Z_{A_{1i}}, A'=A_j \text{ else } Z=Z_{A_{2i}}, A'=A_k$
 - 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:

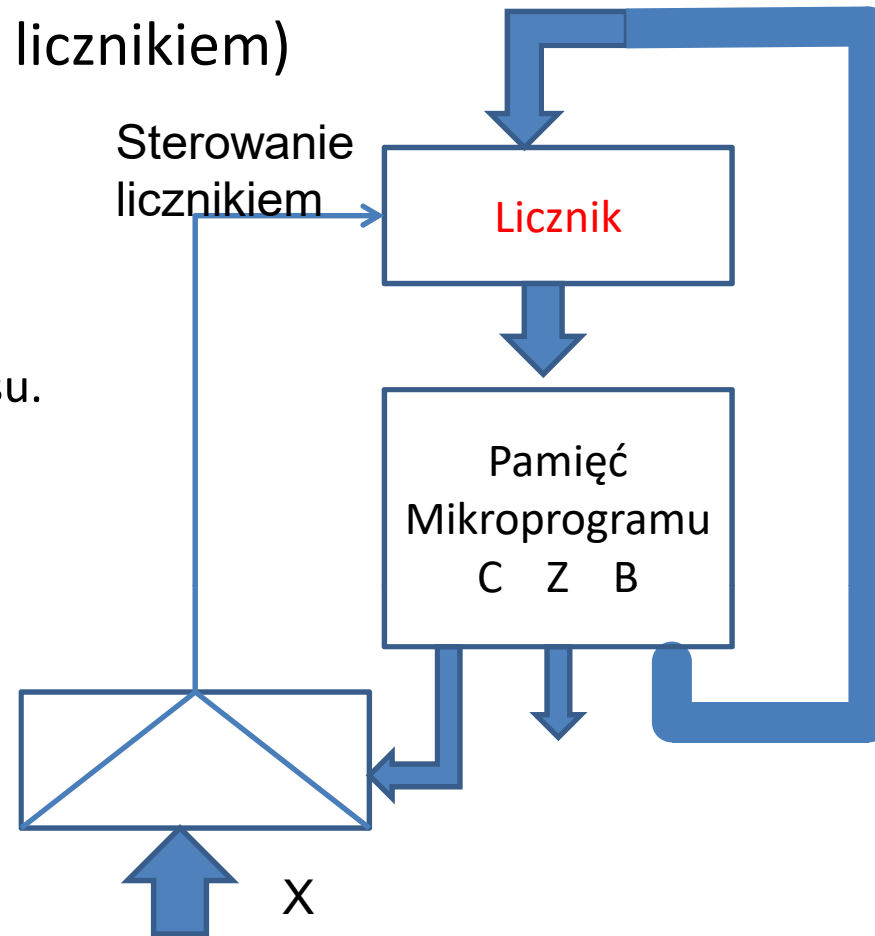
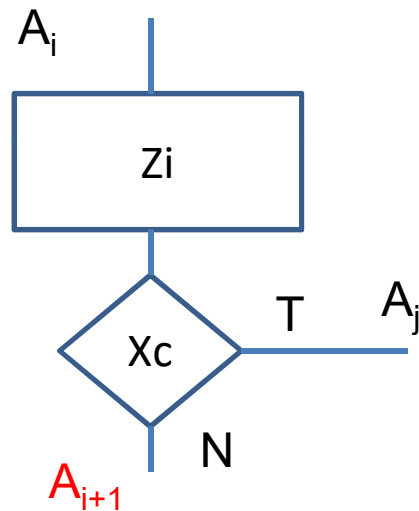
- pola operacyjnego Z
- pola adresowego B i
- pola warunku C

Mikroprogramowany układ sterowania

wer. 2 (z licznikiem)

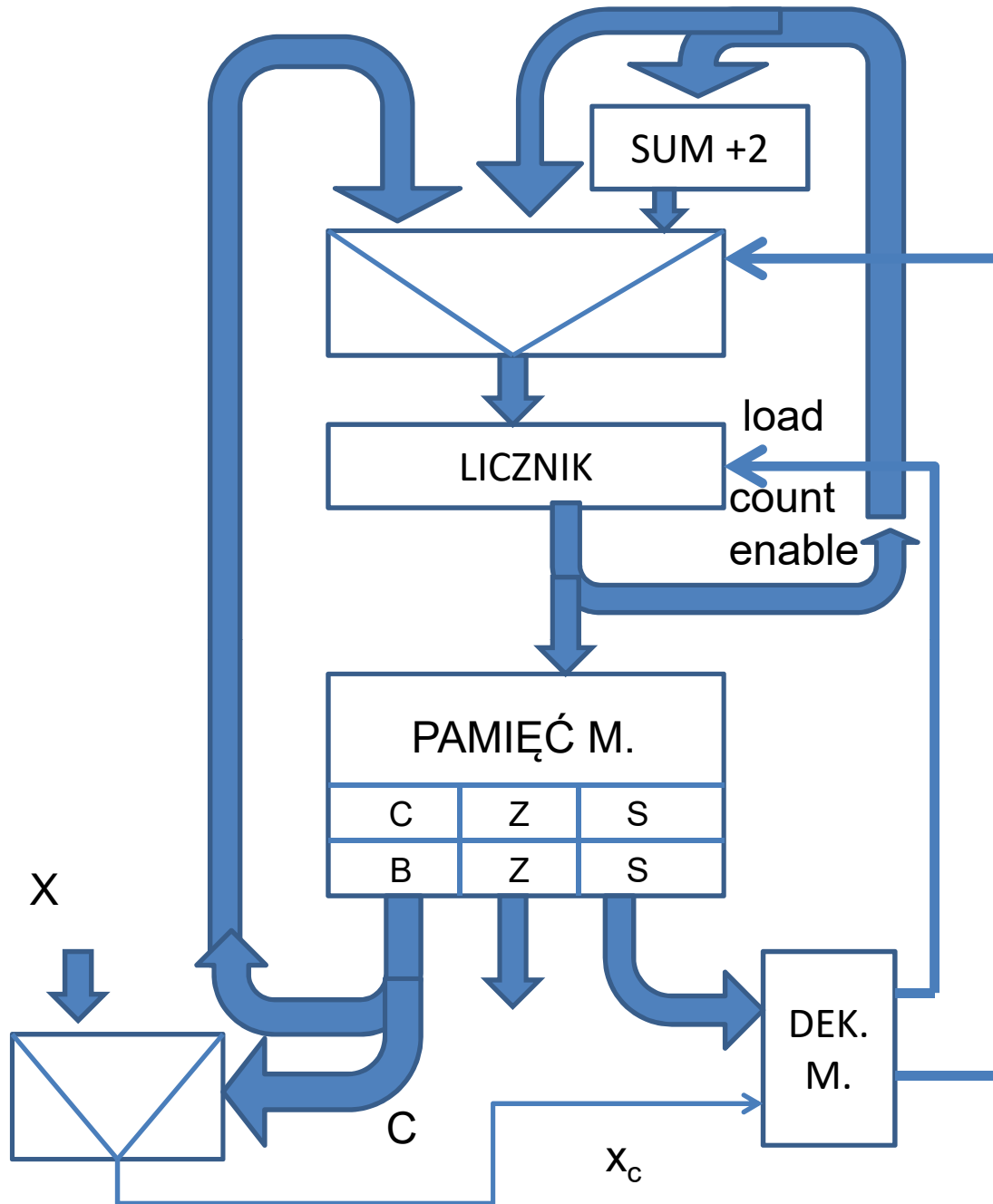
Zadania układu sterowania:

- 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 zawartością pola adresowego

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



UKŁAD STERUJĄCY ver 3

Implementowane instrukcje:

$A_i : Z, \text{ if } x_c \text{ then } A' = A_i \text{ else } A' = A_{i+1}$
 ---Instrukcja skoku warunkowego
 użyte pola: $\langle C, Z, S \rangle$

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

$A_i : Z, A' = A_j$ --skok bezwarunkowy
 użyte pola $\langle A, Z, S \rangle$

Pola rozkazowe:

- B pole adresu
- C pole warunku (strowania)
- 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 A_j z zapisaniem kolejnego (i+1) adresu jako adresu powrotu

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

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

Mikroinstrukcje realizacji pętli:

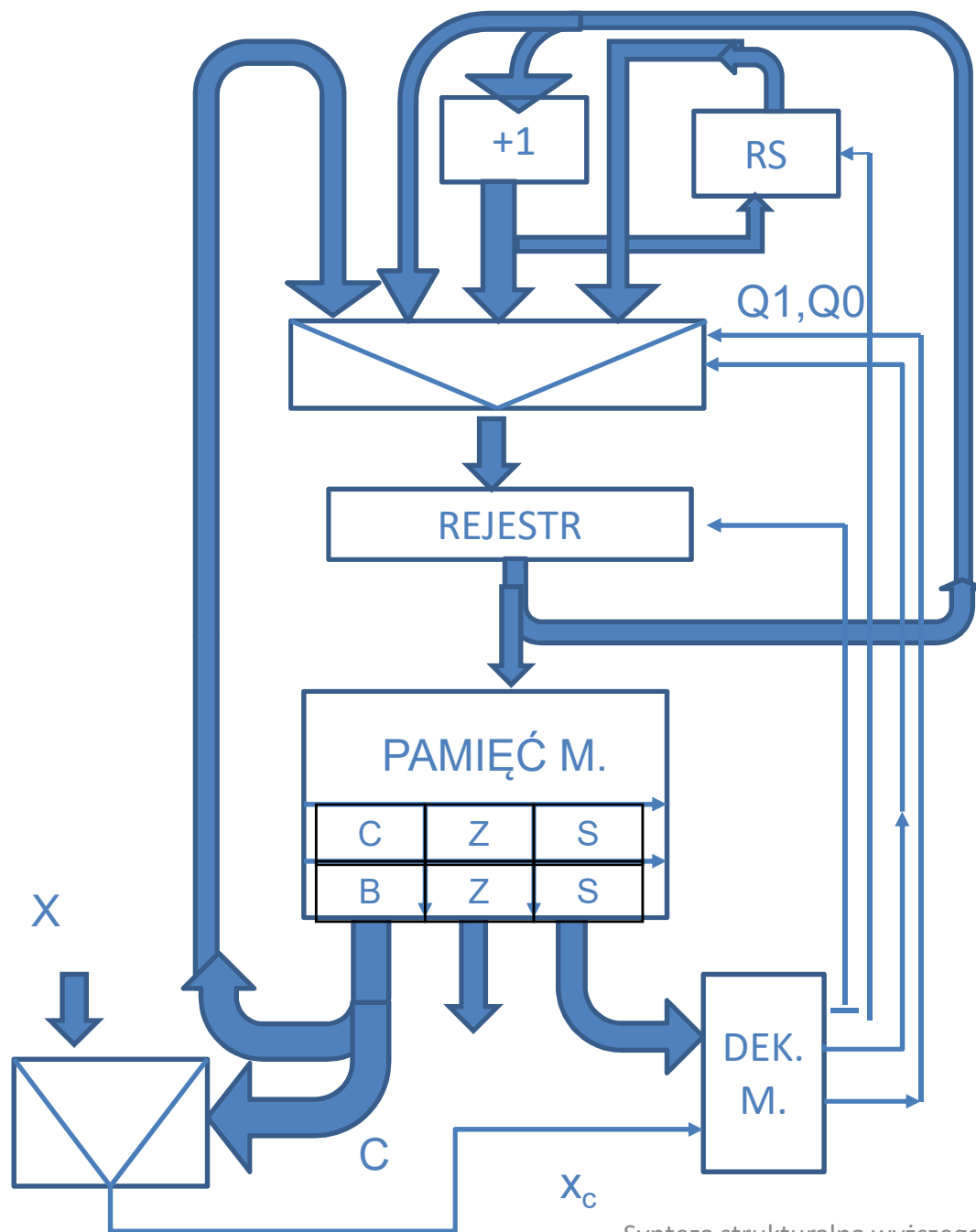
zapamiętanie początku pętli w rejestrze śladu.

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

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

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

-- warunkowo: powrót na początek pętli lub wykonanie kolejnego 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

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

-- oczekiwanie na spełnienie warunku

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

-- warunkowy skok do podprogramu

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

-- 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..