Temat	Procesor wersja II
Autorzy	Krzysztof Gągało 160152 Jan Czyżewski 160377
Data oddania	27.01.2025

1. Wstęp

Celem zadania jest zaprojektowanie procesora potrafiącego ubsłużyc 4 podstawowe operacje. W układzie za pomocą magistrali DIN wprowadzamy 16bitów których dokładniejszy rozpis wyjaśniony mam poniżej. Instrukcje realizowane są przez układ sterujący. Organizuje i synchronizuje on pracę całego systemu między innymi poprzez wybór wejścia adresującego multipleksera oraz synchronizuje kroki operacji.
Działalność programu została sprawdzona na płytce FPGA, zaś sam kod został napisany w języku VHDL. Poniżej opisuję szczegóły dotyczące procesora.

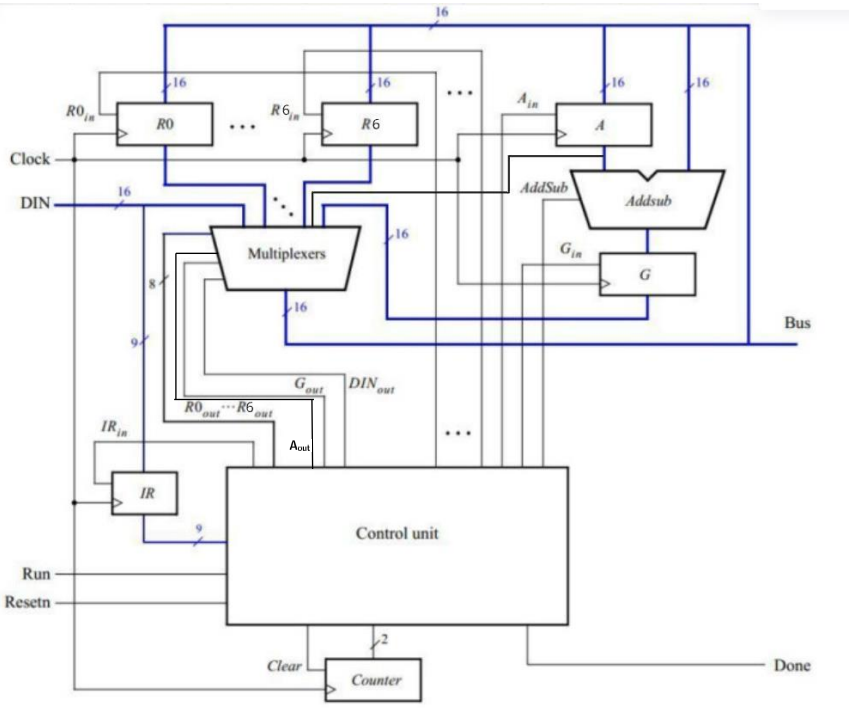
<u>Skład procesora:</u>

- 7 rejestrów
- Akumulator
- Rejestr G
- Sumator
- Multiplekser
- Licznik
- Układ sterujący
- Rejestr IR

Wejściem procesora jest 16-bitowa magistrala DIN.
Rozkaz składa się z 9 bitów YYYXXXIII:

- 3 bity instrukcji (III)
- 3 bity rejestru docelowego (XXX)
- 3 bity rejestru źródłowego (YYY)
- Pozostałe 7 bitów nie są używane w rozkazie

2. Schemat ideowy



Rysunek 1 Schemat procesora

3. Operacje

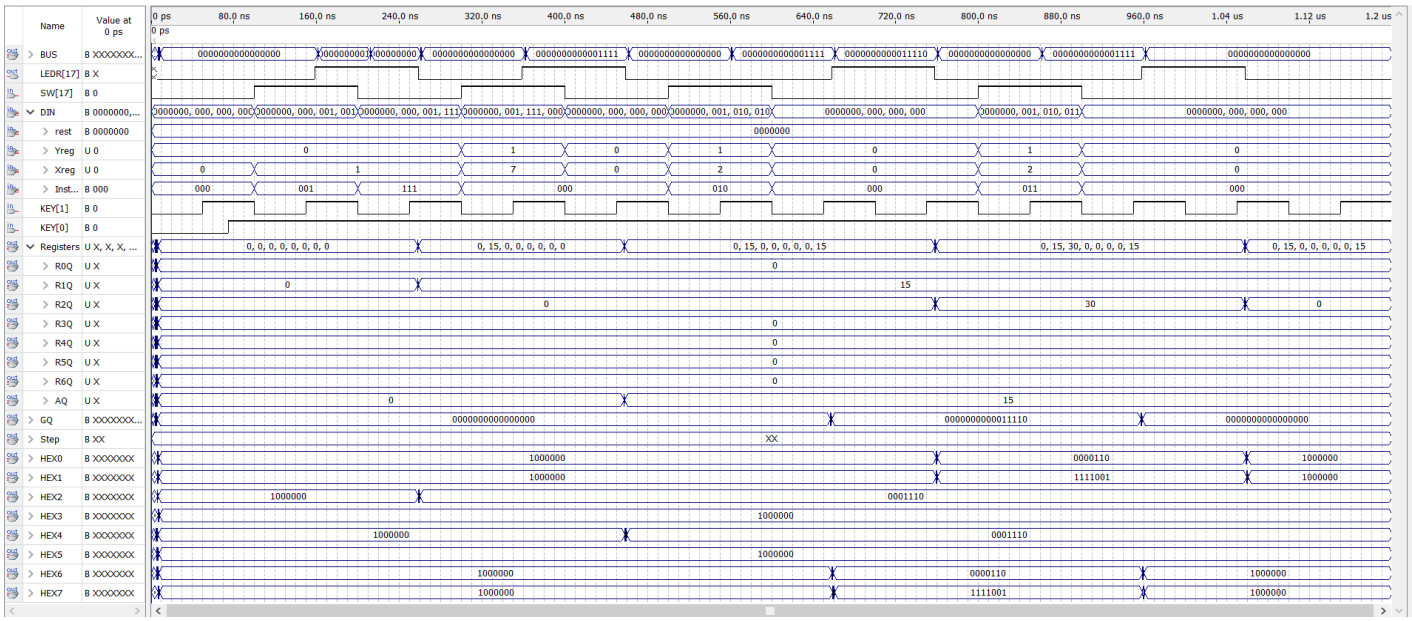
Operacja	Kod w NKB	Funkcja
Mv RX RY	000	Rx <- Ry
Mvi RX D	001	Rx <- D
Add RX RY	010	Rx <- A + Ry
Sub RX RY	011	Rx <- A - Ry

Tabela 1 zakodowane operacje i ich funkcje

	mv	mvi	add	sub
T1	Ry_{out} Rx_{in} Done	Din_{out} Rx_{in} Done	Ry_{out} G_{in}	AddSub RY_{out} G_{in}
T2	-	-	G_{out} RX_{in} Done	Addsub G_{out} Rx_{in} Done
T3	-	-	-	-

Tabela 2 Sygnały aktywne w poszczególnych krokach realizacji rozkazów. W kroku T0 jedynym aktywnym sygnałem jest IRin

4. Przebieg czasowy



Rysunek 2 Przebieg czasowy procesora

Legenda:

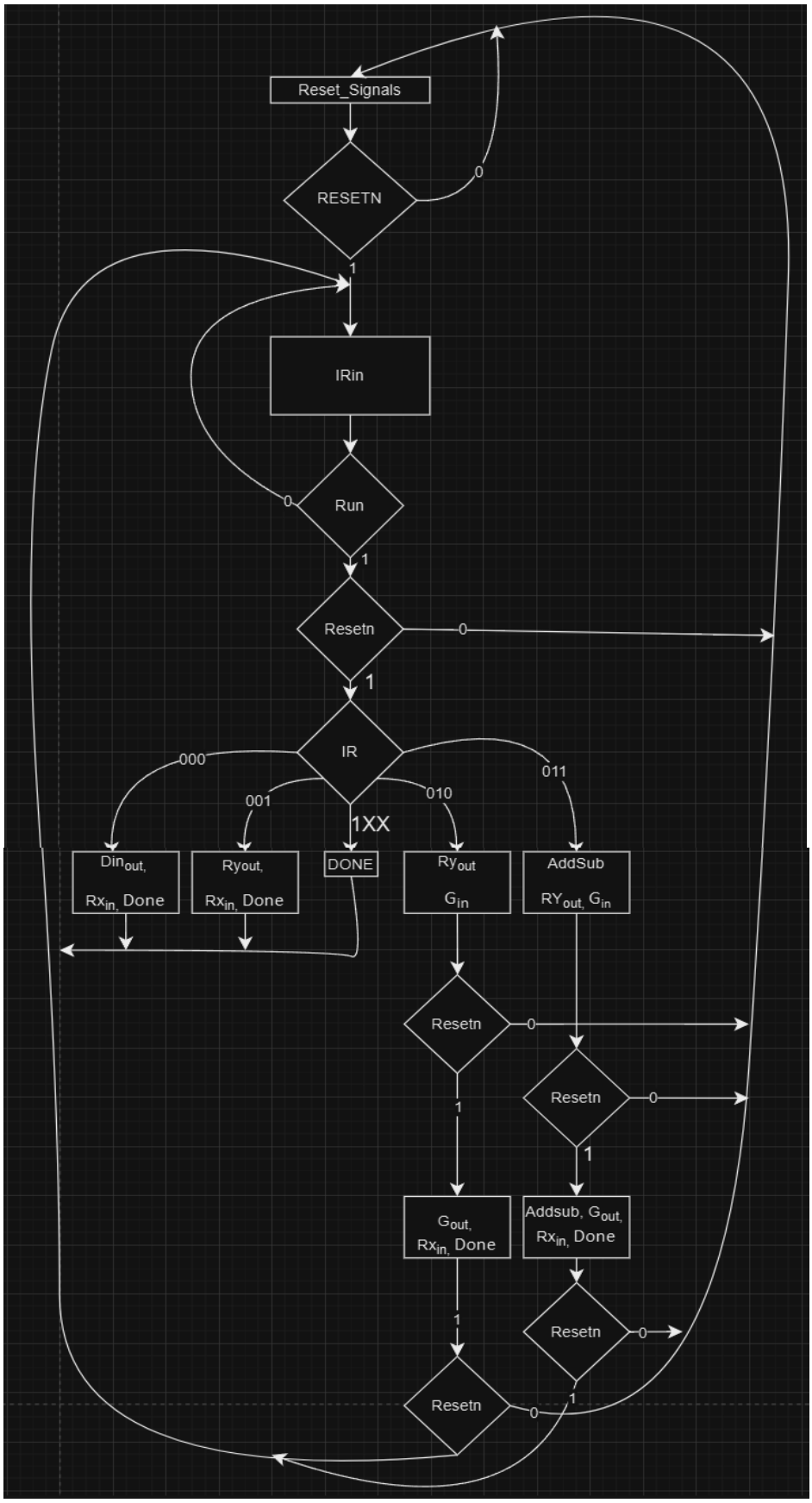
- SW[15-0] – DIN -> sygnał przetrzymujący instrukcie/wejście
- SW[17] – Run -> Rozpoczęcie wykonywania instrukcji z DIN przy następnym tiku zegara
- LEDR[17] – Done -> koniec operacji
- KEY[1] – CLK -> zegar
- KEY[0] – Reset -> sygnał resetujący
- R[0-6]Q – Register -> Wartości przetrzymywane na rejestrach
- AQ – Register -> Wartość przetrzymywana na akumulatorze
- GQ – Register -> Wartość przetrzymywana na wyjściu z sumatora
- Inst – instrukcja -> 3 bity instrukcji
- Xreg - 3 bity rejestru docelowego (XXX)
- Yreg - 3 bity rejestru źródłowego (YYY)
- Rest – pozostałe bity magistrali DIN, które są bez znaczenia w kontekście wykonywanej operacji
- Step – sygnał pomocniczy, wskazujący na obecny krok pracy układu sterującego
- HEX[0-7] – Sygnały wyjściowe dla wyświetlacza 7-segmentowych w celu wyświetlania zawartości rejestru

5. Opis symulacji
    - 0 – 100ns
        - Podawany jest sygnał resetujący (KEY[0]) podczas pierwszego zbocza zegara CLK.
        - Układ jest cały resetowany: resetowane są wszystkie rejestry i sygnały sterujące.
    - 100 – 300ns
        - W chwili 150 ns podawana jest instrukcja na wejście DIN.
        - Instrukcja: 0000000 000 001 001:
            - Rest -> 0000000
            - Yreg -> 000
            - Xreg -> 001
            - Inst -> 001
        - Instrukcja 001 jest kodem dla zapisu liczby w rejestrze – mv RX D.

- Xreg 001 jest kodem dla rejestru R1, czyli RX = R1
- W zboczu zegara 250 ns (krok T1) liczba podawana na wejściu DIN $0000000000001111_{NKB} = 15_{10}$ zapisywana jest w rejestrze R1. Możemy to zaobserwować w symulacji na wyjściu R1Q.

- 300 – 500 ns
  - Na zboczu przy 350 ns wchodzi kolejna instrukcja z magistrali DIN: 0000000001111000
    - Rest -> 0000000
    - Yreg -> 001
    - Xreg -> 111
    - Inst -> 000
  - Instrukcja 000 jest kodem dla kopiowania wartości rejestru do innego rejestru – mvi RX RY.
  - Yreg 001 jest kodem rejestru R1, czyli RY = R1
  - Xreg 111 jest kodem rejestru A (akumulator), czyli RX = A
  - Jako rezultat tej instrukcji w kroku T1 na zboczu 450ns wartość 15 (wartość rejestru w R1) zostanie przekopiowana do akumulatora (A). Można to zaobserwować na wyjściu AQ.
- 550 – 800 ns
  - Na zboczu 550 ns wchodzi następna instrukcja magistrali DIN: 0000000001010010
    - Rest -> 0000000
    - Yreg -> 001
    - Xreg -> 010
    - Inst -> 010
  - Instrukcja 010 jest kodem operacji dodawania – add RX RY
  - Yreg 001 jest kodem rejestru R1, czyli RY = R1
  - Xreg 010 jest kodem rejestru R2, czyli RX = R2
  - Jako rezultat, w kolejnych zboczach zegarowych. możemy zaobserwować, że zawartość rejestru R1 jest wysterowywana na magistralę BUS, suma liczb w akumulatorze i rejestrze R1 zapisywana jest na początku do rejestru G, a następnie przepisywana do rejestru R2.
- 800 – 1100ns
  - Na zobczu 850ns podajemy na magistrali DIN instrukcję: 0000000001010011
    - Rest -> 0000000
    - Yreg -> 001
    - Xreg -> 010
    - Inst -> 010
  - Instrukcja 011 jest kodem operacji odejmowania – sub RX RY
  - Yreg 001 jest kodem rejestru R1, czyli RY = R1
  - Xreg 010 jest kodem rejestru R2, czyli RX = R2
  - Jako rezultat, na kolejnych zboczach zegarowych można zaobserwować, że do rejestru R2 zapisujemy wynik odejmowania A – R1, czyli 15-15. Symulacja potwierdza, że wykonanie operacji przebiega zgodnie z założeniem.

6. ASM



Rysunek 3 ASM

7. **Kod VHDL**

Ta część kodu zawiera definicje portów wejściowych (np. SW) oraz wyjściowych (np. LEDR)

```
ENTITY Processor_Core IS
    PORT (
        SW: IN STD_LOGIC_VECTOR(17 DOWNTO 0);   -- Switches
        KEY: IN STD_LOGIC_VECTOR(3 DOWNTO 0);    -- Keys
        LEDR: OUT STD_LOGIC_VECTOR(17 DOWNTO 0);   -- LEDs



        HEX0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        HEX1 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        HEX2 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        HEX3 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        HEX4 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        HEX5 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        HEX6 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        HEX7 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);


        R0Q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        R1Q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        R2Q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        R3Q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        R4Q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        R5Q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        R6Q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        GQ : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        AQ : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
    );
END Processor_Core;
```

Definiujemy wszystkie komponenty, których używamy w procesorze

```
    COMPONENT regn IS
        PORT(
              R : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
              Rin, Clock : IN STD_LOGIC;
              Q : BUFFER STD_LOGIC_VECTOR(15 DOWNTO 0)
            );
    END COMPONENT;

    COMPONENT upcount IS
        PORT (
            Clear, Clock : IN STD_LOGIC;
            Q : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT dec3to8 IS
        PORT (
            W  : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
            En : IN STD_LOGIC;
            Y  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
    END COMPONENT;
```

```
65      COMPONENT Mux8to1 IS
66          PORT (
67              IN_0 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
68              IN_1 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
69              IN_2 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
70              IN_3 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
71              IN_4 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
72              IN_5 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
73              IN_6 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
74              IN_7 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
75              IN_8 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
76              IN_9 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
77              SEL  : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
78              MUX_OUT : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
79          );
80      END COMPONENT;
81
82      COMPONENT Adder IS
83          PORT (
84              A : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
85              B : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
86              SUM : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
87              ADD_SUB : IN STD_LOGIC
88          );
89      END COMPONENT;
90
91
92      COMPONENT hex_to_7seg IS
93          PORT(
94                binary_in : in  STD_LOGIC_VECTOR (3 downto 0);
95                seg_out   : out STD_LOGIC_VECTOR (6 downto 0)
96              );
97      END COMPONENT;
98
```

Definiujemy sygnały między innymi dla magistrali, sygnałów adresowych, pozwolenia zapisów, wyjścia rejestrów

```
100     SIGNAL BUS_WIRES : STD_LOGIC_VECTOR(15 DOWNTO 0);
101     SIGNAL DIN : STD_LOGIC_VECTOR(15 DOWNTO 0);
102     SIGNAL CLK : STD_LOGIC;
103     SIGNAL Resetn : STD_LOGIC;
104     SIGNAL Run : STD_LOGIC;
105     SIGNAL Done : STD_LOGIC;
106
107     SIGNAL R0in : STD_LOGIC_VECTOR(7 DOWNTO 0); -- R0in, R1in, R2in, R3in, R4in, R5in, R6in, Ain
108     SIGNAL Gin : STD_LOGIC;
109     SIGNAL IRin : STD_LOGIC;
110
111     SIGNAL Rout : STD_LOGIC_VECTOR(7 DOWNTO 0); --wejście adresowe
112     SIGNAL DINout : STD_LOGIC;
113     SIGNAL Gout : STD_LOGIC;
114
115     SIGNAL ADD_SUB : STD_LOGIC;
116
117     SIGNAL A_OUTPUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
118     SIGNAL G_OUTPUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
119     SIGNAL IR_OUTPUT : STD_LOGIC_VECTOR(8 DOWNTO 0);
120
121     SIGNAL R0_OUTPUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
122     SIGNAL R1_OUTPUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
123     SIGNAL R2_OUTPUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
124     SIGNAL R3_OUTPUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
125     SIGNAL R4_OUTPUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
126     SIGNAL R5_OUTPUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
127     SIGNAL R6_OUTPUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
128
129     SIGNAL SUM_SUB_OUTPUT : STD_LOGIC_VECTOR(15 DOWNTO 0); --wyjście z sumatora
130
131     SIGNAL Xreg : STD_LOGIC_VECTOR(7 DOWNTO 0); --sygnal z dekodera po IR
132     SIGNAL Yreg : STD_LOGIC_VECTOR(7 DOWNTO 0);
133
134     SIGNAL MUX_SEL_REG : STD_LOGIC_VECTOR(9 DOWNTO 0); --
```

Definicja sygnałów sterujących – obługa instrukcji, czyli opisujemy jak działają operacje. Wysterowane sygnały są zgodne z tabelą wyżej w sprawozdaniu.

```
161   controlsignals: PROCESS (Tstep_Q, I, Xreg, Yreg, Resetn)
162   BEGIN
163      Rout <= (OTHERS => '0');
164      Rin <= (OTHERS => '0');
165      DINout <= '0';
166      Gout <= '0';
167      GIN <= '0';
168      Done <= '0';
169      ADD_SUB <= '0';
170      IRin <= '0';
171
172
173      IF Resetn = '0' THEN
174        Rout <= (OTHERS => '0');
175        Rin <= (OTHERS => '1');
176        DINout <= '0';
177        Done <= '0';
178        ADD_SUB <= '0';
179
180      ELSE
181         CASE Tstep_Q IS
182             WHEN "00" => -- store DIN in IR as long as Tstep_Q = 0
183                  IRin <= '1';
184             WHEN "01" => -- step T1
185                CASE I IS
186                  WHEN "000" => -- instrukcja mv RX, RY
187                     Rout <= Yreg; --Rxout <= Yreg
188                     Rin <= Xreg;
189                     Done <= High;
190                  WHEN "001" => -- instrukcja mvi RX,#D
191                     DINout <= High;
192                     Rin <= Xreg;
193                     Done <= High;
194                  WHEN "010" => -- instrukcja add RX, RY
195                     Rout <= Yreg; --R[y]
196                     Gin <= High; --zapis do G
197                  WHEN "011" => -- instrukcja sub RX, RY
198                     ADD_SUB <= '1';
199                     Rout <= Yreg; --R[y]
200                     Gin <= High; --zapis do G
201                  WHEN OTHERS =>
202                END CASE;
203             WHEN "10" => -- step T2
204                CASE I IS
205                  WHEN "000" => -- instrukcja mv RX, RY
206                  WHEN "001" => -- instrukcja mvi RX,#D
207                  WHEN "010" => -- instrukcja add RX, RY
208                     Gout <= high;
209                     Rin <= Xreg;
210                     Done <= High;
211                  WHEN "011" => -- instrukcja sub RX, RY
212                     ADD_SUB <= '1';
213                     Gout <= high;
214                     Rin <= Xreg;
215                     Done <= High;
216                  WHEN OTHERS =>
217                END CASE;
218             WHEN "11" => -- step T3
219          END CASE;
220        END IF;
221    END PROCESS;
```

Realizujemy połączenia między elementami układu zgonie ze schematem

```
223     A: regn PORT MAP(R => BUS_WIRES, Rin=>Rin(7), Clock =>CLK, Q=>A_OUTPUT);
224     G: regn PORT MAP(R => SUM_SUB_OUTPUT, Rin=>Gin, Clock =>CLK, Q=>G_OUTPUT);
225
226     R0: regn PORT MAP(R => BUS_WIRES, Rin=>Rin(0), Clock =>CLK, Q=>R0_OUTPUT);
227     R1: regn PORT MAP(R => BUS_WIRES, Rin=>Rin(1), Clock =>CLK, Q=>R1_OUTPUT);
228     R2: regn PORT MAP(R => BUS_WIRES, Rin=>Rin(2), Clock =>CLK, Q=>R2_OUTPUT);
229     R3: regn PORT MAP(R => BUS_WIRES, Rin=>Rin(3), Clock =>CLK, Q=>R3_OUTPUT);
230     R4: regn PORT MAP(R => BUS_WIRES, Rin=>Rin(4), Clock =>CLK, Q=>R4_OUTPUT);
231     R5: regn PORT MAP(R => BUS_WIRES, Rin=>Rin(5), Clock =>CLK, Q=>R5_OUTPUT);
232     R6: regn PORT MAP(R => BUS_WIRES, Rin=>Rin(6), Clock =>CLK, Q=>R6_OUTPUT);
233
234     HEX_DECODER0: hex_to_7seg PORT MAP(binary_in => R2_OUTPUT(3 DOWNTO 0), seg_out => HEX0);
235     HEX_DECODER1: hex_to_7seg PORT MAP(binary_in => R2_OUTPUT(7 DOWNTO 4), seg_out => HEX1);
236
237     HEX_DECODER2: hex_to_7seg PORT MAP(binary_in => R1_OUTPUT(3 DOWNTO 0), seg_out => HEX2);
238     HEX_DECODER3: hex_to_7seg PORT MAP(binary_in => R1_OUTPUT(7 DOWNTO 4), seg_out => HEX3);
239
240     HEX_DECODER4: hex_to_7seg PORT MAP(binary_in => A_OUTPUT(3 DOWNTO 0), seg_out => HEX4);
241     HEX_DECODER5: hex_to_7seg PORT MAP(binary_in => A_OUTPUT(7 DOWNTO 4), seg_out => HEX5);
242
243     HEX_DECODER6: hex_to_7seg PORT MAP(binary_in => G_OUTPUT(3 DOWNTO 0), seg_out => HEX6);
244     HEX_DECODER7: hex_to_7seg PORT MAP(binary_in => G_OUTPUT(7 DOWNTO 4), seg_out => HEX7);
245
246     IR: regn PORT MAP(R => DIN, Rin=>IRin, Clock =>CLK AND RUN, Q(8 DOWNTO 0) => IR_OUTPUT);
```