Projekt prostego procesora - Wersja 3

Student [redacted] 160128
[redacted]160116
Kierunek Inforamtyka
Wydział Wydział Informatyki i Teleinforamtyki

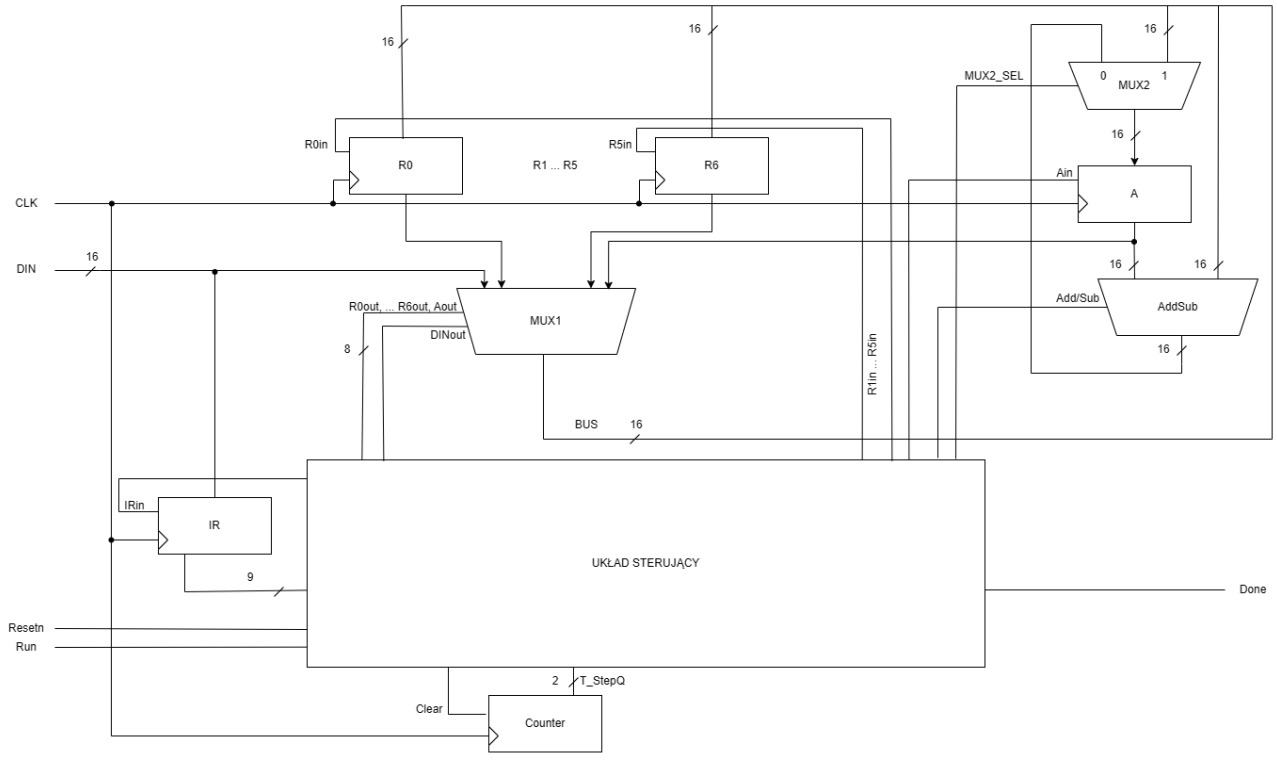
Wstęp

Celem projektu było opracowanie i implementacja cyfrowego układu prostego procesora. Procesor został zaprojektowany jako jednostka zawierająca podstawowe elementy, takie jak 16-bitowe rejestry (R0–R6), akumulator, multiplekser, sumator, licznik oraz układ sterujący. System operuje na 16-bitowej magistrali danych wejściowych (DIN), umożliwiając wykonywanie podstawowych operacji arytmetycznych i przesłań danych między rejestrami.

Procesor wykorzystuje zestaw rozkazów kodowanych w 9-bitowym formacie YYYXXXIII, gdzie bity III definiują typ instrukcji, XXX wskazuje rejestr docelowy, a YYY określa rejestr źródłowy. Instrukcje są realizowane przez układ sterujący, który zarządza przepływem danych wewnątrz systemu, aktywuje odpowiednie sygnały sterujące oraz synchronizuje kolejne kroki wykonywania operacji. Projekt został zrealizowany w języku VHDL, a jego poprawność została zweryfikowana za pomocą symulacji czasowysch oraz testów przeprowadzonych na układzie FPGA.

W ramach sprawozdania przedstawiono szczegółowy opis projektu, przebiegi sygnałów potwierdzające poprawność działania procesora, a także dokumentację kodu przygotowanego do działania na układzie FPGA i graficzne odwzorowanie schematu układu.

Schemat ideowy



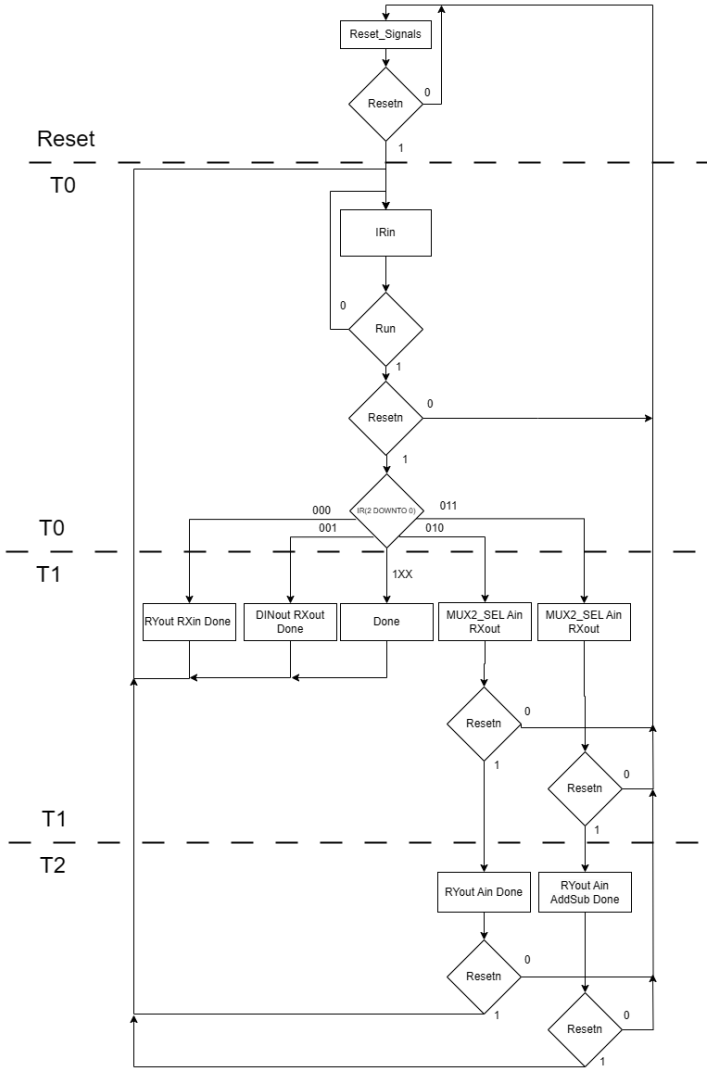
Rysunek 1: Enter Caption

Operacje Procesora

kod(NKB)	Operacja	Wykonywana funkcja	T1	T2	T3
000	mv RX RY	$RX \leftarrow RY$	RYout, RXin, Done		
001	mvi RX #D	$RX \leftarrow D$	DINout, RXin, Done		
010	add RX, RY	$A \leftarrow RX + RY$	RXout, Ain, MUX2_SEL	Ain, Done, RYout	
011	sub Rx, RY	$A \leftarrow RX - RY$	RXout, Ain, MUX2_SEL.	Ain, Done, RYout, AddSub	

Tabela 1: Sygnały aktywne w poszczególnych krokach(T1, T2, T3) realizacji rozkazów.(W kroku T0 jedynym aktywnym jest sygnał IRin)

Diagram ASM automatu



Rysunek 2: Diagram prezentujący przejścia między stanami dla układu sterujacego

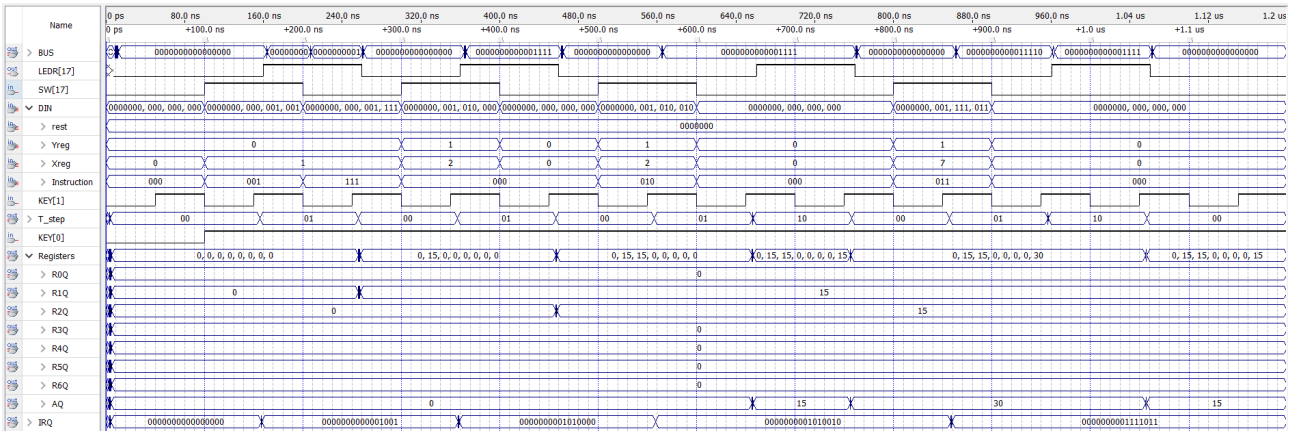
Uwagi odnośnie diagramu ASM w kontekście schematu ideowego

W przypadku opracowywanego diagramu ASM, licznik nie jest brany pod uwagę jako sygnał wejściowy ani sterujący, ponieważ pełni rolę elementu pomocniczego w ramach układu sterującego. Przewidywany przebieg sygnałów w systemie jest jednoznaczny i rosnący: T0 -> T1 -> T2 -> T3, z ewentualnym powrotem do stanu T0. Z tego powodu nie zachodzi sytuacja, w której sygnały przechodzą w sposób nieliniowy, np. z T0 do T2, a następnie T1, ponieważ taka sekwencja nie jest możliwa w analizowanym układzie.

Z uwagi na to, licznik traktujemy jako element pomocniczy, który jest wewnętrzną częścią układu sterującego, pełniącą funkcję ścisłej synchronizacji stanów, ale nie wpływającą bezpośrednio na logikę sterowania. Dodatkowo, sygnały związane z licznikiem, takie jak sygnał czyszczenia (Clear) oraz wyjścia licznika, również są traktowane jako pomocnicze i nie są uwzględniane w diagramie ASM. Zgodnie z wymaganiami specyfikacji zadania (w tym przykładów kodów), sygnały te nie zostały uwzględnione w strukturze sterowania.

Podobnie jak dekodery, które zostały uznane za pomocnicze elementy w układzie, licznik oraz jego sygnały pomocnicze nie mają wpływu na ogólną logikę sterowania systemem, dlatego też nie zostały uwzględnione w analizowanym diagramie ASM.

Przebieg Czasowy



Rysunek 3: Przebieg czasowy

Nazwa na przebiegu	Nazwa Sygnału	Wej/Wyj	Znaczenie
LEDR[17]	Done	Wyj	Zakończenie wykonywanie operacji
SW[17]	Run	Wej	Rozpoczęcie wykonywania wporwadzonej operacji na wejście DIN przy następnym narastającym zboczu zegarowym
SW[15-0]/DIN	DIN	Wej	Sygnał przetrzymujący instrukcję do wykonania bity: -[0-2] kod operacji do wykonania, -[5-3] pierwszy operand operacji Xreg, -[8-6] drugi operand operacji Yreg
KEY[1]	CLK	Wej	Zegar
KEY[0]	Resetn	Wej	Sygnał Resetujący reagujący na stan niski
T_Step	T_StepQ	Wyj	dwubitowy sygnał licznika oznaczający krok pracy
IRQ		Wyj	wyjście rejestru IR (Instruction Register) przetrzymujący obecnie wykonywaną intrukcję
Registers		Wyj	Grupa pokazująca wartości przetrzymywane przez wszstkie rejestry

Tabela 2: Caption

Podczas Symulacji wykonano cztery operacje jakie układ oferuję (patrz Tabela 1.) Cykl zegara został ustawiony na 100[ns], a sama długość symulacji na 1.1[us]. Podczas tej symulacji można po kolei zaobserwować:

- **0–100 ns: Operacja** reset

  Na początku przebiegu, podczas pierwszego zbocza zegara CLK (KEY[1]), podawany jest sygnał resetujący Resetn (KEY[0]). Sygnał ten wprowadza układ procesora w stan początkowy, resetując rejestry, sygnały sterujące oraz liczniki.

- **100–300 ns: Operacja** $R1 \leftarrow 15$

  W chwili $t = 150$ ns, podczas zbocza zegara, na wejście DIN podawana jest instrukcja 0000000 000 001 001.

		Yreg	Xreg	Instruction
DIN	0000000	000	001	001
		R0	R1	$RX \leftarrow \#D$

Tabela 3: Opis instrukcji dla operacji $R1 \leftarrow 15$

  - **Instruction**: 001_{NKB} – kod operacji (zapis liczby w rejestrze).
  - **Xreg**: 001_{NKB} ($Xreg = 1_{10}$) – kod rejestru R1.

  W wyniku tej instrukcji, w kroku pracy T1, zostają wysterowane sygnały: DINout, R1in, Done. Przy następnym zboczu zegara ($t = 250$ ns) liczba podawana na wejściu DIN ($0000000000001111_{NKB} = 15_{10}$) zostaje zapisana w rejestrze R1, co można zaobserwować na wyjściu R1Q.

- **300–500 ns: Operacja** $R2 \leftarrow R1$
  W chwili $t = 350$ ns, podczas zbocza zegara, na wejście DIN podawana jest instrukcja 0000000 001 010 000.

		Yreg	Xreg	Instruction
DIN	0000000	001	010	000
		R1	R2	$RX \leftarrow RY$

Tabela 4: Opis instrukcji dla operacji $R2 \leftarrow R1$

  - **Instruction**: 000_{NKB} – kod operacji (kopiowanie zawartości rejestru).
  - **Xreg**: 010_{NKB} ($Xreg = 2_{10}$) – kod rejestru R2.
  - **Yreg**: 001_{NKB} ($Yreg = 1_{10}$) – kod rejestru R1.

  W kroku pracy T1 wysterowane są sygnały: $RYout = R1out$, $RXin = R2in$, Done. W efekcie, przy następnym zboczu zegara ($t = 450$ ns), wartość 15_{10} z rejestru R1 zostaje zapisana w rejestrze R2, co można zaobserwować na wyjściu R2Q.

- **500–800 ns: Operacja** $A \leftarrow R1 + R2$ W chwili $t = 550$ ns, podczas zbocza zegara, na wejście DIN podawana jest instrukcja 0000000 001 010 010.

		Yreg	Xreg	Instruction
DIN	0000000	001	010	010
		R1	R2	$A \leftarrow RX + RY$

Tabela 5: Opis instrukcji dla operacji $A \leftarrow R1 + R2$

  - **Instruction**: 010_{NKB} – kod operacji (dodawanie).
  - **Xreg**: 010_{NKB} ($Xreg = 2_{10}$) – kod rejestru R2.
  - **Yreg**: 001_{NKB} ($Yreg = 1_{10}$) – kod rejestru R1.

  W pierwszym kroku pracy T1 danej operacji wysterowane są sygnały: $RXout = R2out$, Ain, MUX_SEL_A. W efekcie zawartość rejestru R2 zostaje zapisana w Akumulatorze A ($t = 650$ ns, wyjście AQ).
  W drugim kroku pracy T2 danej operacji wysterowane są sygnały: Ain, Done, $RYout = R1out$. Zawartość rejestru R1 zostaje dodana do zawartości Akumulatora, a wynik 30_{10} zostaje zapisany w Akumulatorze ($t =$ 750 ns, wyjście AQ).

- **800–1100 ns: Operacja** $A \leftarrow R7 - R1$ **(gdzie** $R7 = A$**)** W chwili $t = 850$ ns, podczas zbocza zegara, na wejście DIN podawana jest instrukcja 0000000 001 111 011.

		Yreg	Xreg	Instruction
DIN	0000000	001	111	011
		R1	R7=A	$A \leftarrow RX - RY$

Tabela 6: Opis instrukcji dla operacji $A \leftarrow R7 - R1$

- **Instruction**: 011_{NKB} – kod operacji (odejmowanie).
-**Xreg**: 111_{NKB} ($Xreg = 7_{10}$) – kod rejestru A (R7).
- **Yreg**: 001_{NKB} ($Yreg = 1_{10}$) – kod rejestru R1.

W kroku T1 danej operacji wysterowane są sygnały: $RXout = R7out = Aout$, Ain, MUX_SEL_A. Zawartość rejestru A zostaje ponownie zapisana w Akumulatorze, co nie wprowadza zmian, ale wyświetla jego zawartość na magistrali BUS. W kroku T2 danej operacji wysterowane są sygnały: Ain, Done, $RYout = R1out$, ADD_SUB (ustawiony na odejmowanie). W wyniku tych operacji, od zawartości Akumulatora (30_{10}) zostaje odjęta wartość z rejestru R1 (15_{10}). Wynik 15_{10} zapisuje się w Akumulatorze ($t = 1050$ ns, wyjście AQ).

Kod VHDL

Definicja modułu procesora

Moduł `Processor_Core` zawiera definicje portów wejściowych i wyjściowych, takich jak `SW`, `KEY`, `LEDR`, a także sygnałów wewnętrznych.

```
ENTITY Processor_Core IS
    PORT (
        SW           : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
        KEY          : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        LEDR         : OUT STD_LOGIC_VECTOR(17 DOWNTO 0);
        R0Q, R1Q, R2Q, R3Q, R4Q, R5Q, R6Q, AQ : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        IRQ : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        T_step : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
    );
END Processor_Core;
```

Listing 1: Definicja modułu procesora

Deklaracje komponentów

Sekcja ta definiuje komponenty używane w architekturze procesora, takie jak rejestry, multipleksery, dekodery, sumator i licznik np..

```
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;
... -- deklaracje innych komponentow jak: multipleksery, sumator, dekodery
COMPONENT upcount IS
    PORT (
        Clear, Clock : IN STD_LOGIC;
        Q : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
    );
END COMPONENT;
```

Listing 2: Deklaracje komponentów

Deklaracje sygnałów

Sygnały definiują magistralę danych, zegar, sygnały resetu, sygnały adresowe dla multiplekserów, sygnały zapisu dla rejestrów, wyjściowe sygnały danych rejestrów, sygnał trybu pracy sumatora.

```
SIGNAL BUS_WIRES : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL DIN : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL CLK : STD_LOGIC;
SIGNAL Resetn : STD_LOGIC;
SIGNAL Run : STD_LOGIC;
SIGNAL Done : STD_LOGIC;

SIGNAL R0_OUTPUT : STD_LOGIC_VECTOR(15 DOWNTO 0); -- wyjscie danych rejestru R0
    ... -- pozostale definicje sygnalow wyjsc rejestrow
SIGNAL R6_OUTPUT : STD_LOGIC_VECTOR(15 DOWNTO 0); -- wyjscie danych rejestru R6

    ... --pozostale definicje sygnalow A_OUTPUT, IR_OUTPUT, MUX_SEL, MUX2_SEL itd.
```

Listing 3: Deklaracje sygnałów

Główne połączenia i instancje elementów układu

W tym fragmencie kodu zrealizowane jest połączenie odpowiednich wyprowadzeń układu FPGA do opisanych sygnałów między innymi `Run`, `CLK`, `DIN`, `Resetn` itd. oraz realizowane są wszystkie połączenia między elementami układu procesora. Zgodność tych połaczenia można porównać z schematem ideowym (Rysunek 1).

```
DIN <= SW(15 DOWNTO 0);
CLK <= KEY(1);
Resetn <= KEY(0);
Run <= SW(17);
LEDR(15 DOWNTO 0) <= BUS_WIRES;
LEDR(17) <= Done;

Clear <=  (((NOT Run OR Done) OR (NOT Resetn) )AND NOT(Tstep_Q(0) OR Tstep_Q(1))) OR
    Done;

Tstep: upcount PORT MAP (Clear, CLK, Tstep_Q);
T_step <= Tstep_Q;
I <= IR_OUTPUT(2 DOWNTO 0);
decX: dec3to8 PORT MAP (IR_OUTPUT(5 DOWNTO 3), High, Xreg);
decY: dec3to8 PORT MAP (IR_OUTPUT(8 DOWNTO 6), High, Yreg);

A: regn PORT MAP(R => MUX_OUTPUT, Rin=>Rin(7), Clock =>CLK, Q=>A_OUTPUT);
R0: regn PORT MAP(R => BUS_WIRES, Rin=>Rin(0), Clock =>CLK, Q=>R0_OUTPUT);
... -- pozostale instancje rejestrow R1, ..., R5
R6: regn PORT MAP(R => BUS_WIRES, Rin=>Rin(6), Clock =>CLK, Q=>R6_OUTPUT);
IR: regn PORT MAP(R => DIN, Rin=>IRin, Clock =>CLK AND RUN, Q(8 DOWNTO 0) =>
    IR_OUTPUT);

mux: MUX2 PORT MAP (IN0 => SUM_SUB_OUTPUT, IN1 => BUS_WIRES, SEL => MUX_SEL_A,
    DATA_OUT => MUX_OUTPUT);

reg_mux : Mux8to1 PORT MAP (IN_0=>R0_OUTPUT, IN_1=>R1_OUTPUT, IN_2=>R2_OUTPUT, IN_3=>
    R3_OUTPUT, IN_4=>R4_OUTPUT, IN_5=>R5_OUTPUT, IN_6=>R6_OUTPUT, IN_7=>A_OUTPUT,
    IN_8=>DIN, SEL=>(DINout & Rout),  MUX_OUT=>BUS_WIRES);

sumator: Adder PORT MAP(A=>A_OUTPUT, B=>BUS_WIRES, SUM=>SUM_SUB_OUTPUT, ADD_SUB=>
    ADD_SUB);
```

Listing 4: Główne połączenia i instancję elementów układu

Sygnał `Clear` (Listing 4. linia 8) steruje resetowaniem licznika w zależności od etapu wykonywania instrukcji.

- Licznik nie powinien być czyszczony (stan niski) w dwóch przypadkach:
  1. Gdy `rozpoczyna się wykonywanie instrukcji`, co sygnalizuje aktywność sygnału `Run`.
  2. Kiedy procesor jest `w trakcie wykonywania instrukcji`, czyli znajduje się w kroku innym niż `T0`. Stan ten jest określany przez funkcję: `NOT(Tstep_Q(0) OR Tstep_Q(1))`
- Licznik powinien być czyszczony (stan wysoki) w następujących przypadkach:
  1. Kiedy `instrukcja została zakończona`, co sygnalizuje sygnał `Done`.
  2. Gdy aktywowany zostaje `sygnał resetowania układu`, czyli sygnał `NOT Resetn`.

Funkcja ta zapewnia poprawną pracę licznika potrzebną dla poprawnej pracy układu sterujacego US.

Zdefiniowane dekodery `decX`, `decY` służą do zamiany dwóch operandów operacji z *kodu NKB* na *kod 1 z N* by później odpowiednie rejestry były wyprowadzane na magistralę `BUS` lub była zapisywana do nich nowa wartość.

Obsługa instrukcji

Logika procesora jest podzielona na kroki czasowe (`T0`, `T1`, ...), a dla każdej instrukcji realizowane są odpowiednie operacje, np. przeniesienie danych między rejestrami (`mv RX, RY`) czy dodawanie (`add RX, RY`).

```
controlsignals: PROCESS (Tstep_Q, I, Xreg, Yreg)
BEGIN -- wartosci poczatkowe sygnalow sterujacych
    Rout <= (OTHERS => '0');
    Rin <= (OTHERS => '0');
    DINout <= Low;
    MUX_SEL_A <= Low;
    Done <= Low;
    ADD_SUB <= Low;
    IRin <= Low;

    IF Resetn = '0' THEN        -- stan resetowania zawartosci rejestrow poprzez
        Rout <= (OTHERS => '0'); -- zapisanie do ich wszystkich zawartosci magistrali
        Rin <= (OTHERS => '1');  -- BUS ktora w tym stanie jest rowna 0
        DINout <= Low;
        Done <= Low;
        MUX_SEL_A <= High;
        ADD_SUB <= Low;
    ELSE
        CASE Tstep_Q IS -- kolejne kroki pracy wraz z wysterowywaniem odpowiednich
            sygnalow zaleznie od obecnego kroku pracy i wykonywanej instrukcji
            WHEN "00" => -- krok T0
                IRin <= High;     -- IRin
            WHEN "01" => -- krok T1
                CASE I IS
                    WHEN "000" =>           -- T1: mv RX, RY
                        Rout <= Yreg;       -- RYout
                        Rin <= Xreg;        -- RXin
                        Done <= High;       -- Done
                    WHEN "001" =>           -- T1: mvi RX,#D
                        DINout <= High;     -- DINout
                        Rin <= Xreg;        -- RXin
                        Done <= High;       -- Done
                    WHEN "010" =>           -- T1: add RX, RY
                        MUX_SEL_A <= High;  -- MUX2_SEL
                        Rin(7) <= High;     -- Ain
                        Rout <= Xreg;       -- RXout
                    WHEN "011" =>           -- T1: sub RX, RY
                        MUX_SEL_A <= High;  -- MUX2_SEL
                        Rin(7) <= High;     -- Ain
                        Rout <= Xreg;       -- Rxout
                    WHEN OTHERS => Done <= High;
                END CASE;
```

```
42              WHEN "10" => -- krok T2
43                  CASE I IS
44                      WHEN "010" =>           -- T2: add RX, RY
45                          Rout <= Yreg;       -- RYout
46                          Rin(7) <= High;     -- Ain
47                          Done <= High;       -- Done
48                      WHEN "011" =>           -- T2: sub RX, RY
49                          Rout <= Yreg;       -- RYout
50                          Rin(7) <= High;     -- Ain
51                          ADD_SUB <= High;    -- AddSub
52                          Done <= High;       -- Done
53                      WHEN OTHERS => Done <= High;
54                  END CASE;
55              WHEN OTHERS => Done <= High;
56          END CASE;
57      END IF;
58  END PROCESS;
```

Listing 5: Obsługa instrukcji

Wysterowywane sygnały są zgodne z tabelą 1, wktórej zdefiniowane są wysterowywane sygnały dla obecnie realizowanej operacji `I` w obecnym kroku pracy `Tstep_Q`. Jeżli zostaje podana operacja która nie jest zdefiniowana przez układ sterujący wysterowywany jest wyłącznie sygnał `Done`