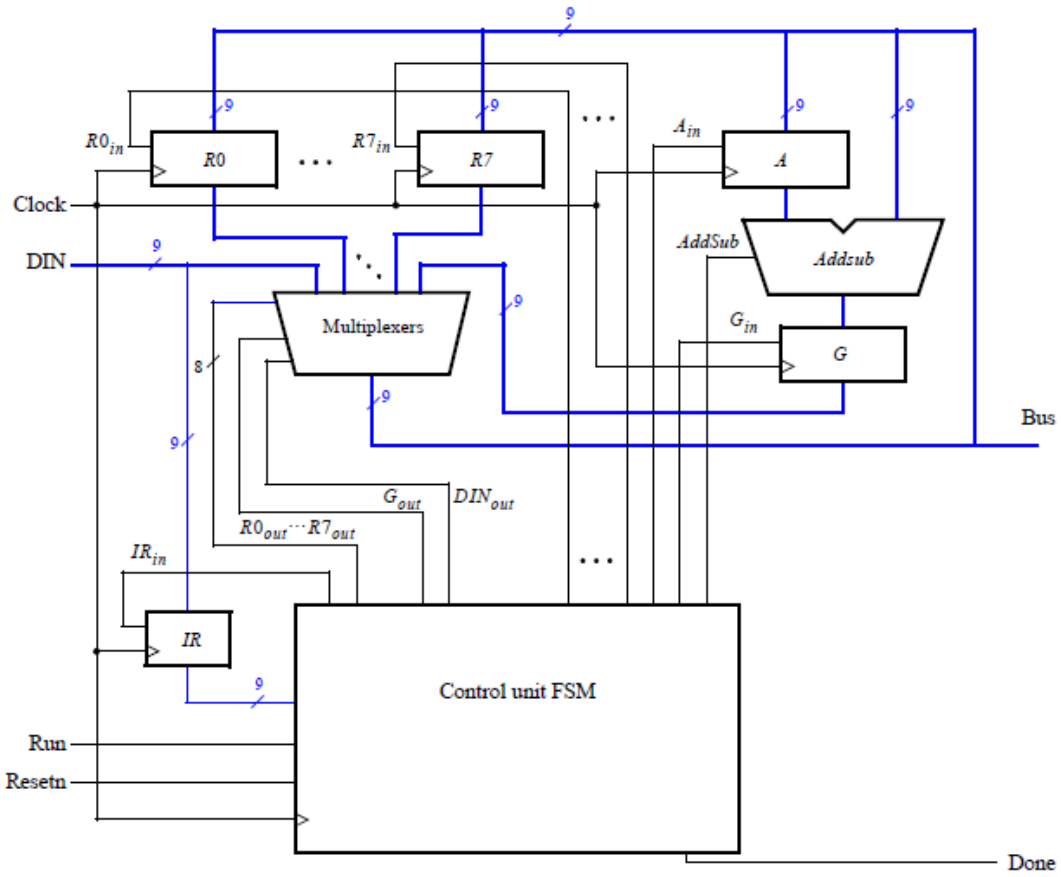
Lab.10. Prosty procesor

Wymagana wiedza

- podstawy projektowania procesorów;
- Prosta struktura procesora;
- zbiór poleceń procesora;
- format prostych poleceń procesora;
- sygnały sterujące procesora;
- wykonywanie poleceń w trzech taktach.

Wykonanie

1. Utwórz projekt proc_1 prostego procesora ze strukturą z rys. 1.



Rys. 1. Struktura prostego procesora

Zestaw instrukcji procesora pokazano w tabeli. 1.

Tabela. 1. Instrukcje w procesorze

Operation	Function performed
mv Rx,Ry	$Rx \leftarrow [Ry]$
mvi Rx,#D	$Rx \leftarrow D$
add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

Tutaj [Rx] oznacza zawartość rejestru **Rx**, $\leftarrow$ – przekazywanie danych.

Komenda mv (move) ładuje zawartość rejestru Ry do rejestru Rx, komenda mvi (move immediate) ładuje stałą D do rejestru Rx, polecenie add dodaje zawartość rejestrów Rx i Ry, wynik jest zapisywany do rejestru Rx; polecenie sub od zawartości rejestru Rx odejmuje zawartość rejestru Ry, wynik jest zapisany w Rx.

Każda instrukcja procesora przedstawiona jest w formacie IIIXXXYYY, gdzie
III - kod polecenia;
XXX to rejestr Rx;
YYY jest kodem rejestru Ry.

Dla naszego procesora akceptowane są następujące kody poleceń:
mv - 000
mvi - 001
add - 010
sub - 011
Dodatkowy bit w kodzie polecenia pozwala rozszerzyć listę poleceń w przyszłości.

Polecenia mv i mvi są wykonywane w jednym cyklu zegara, polecenia add i sub wykonywane są w trzech cyklach zegara. Sygnały sterujące ustawione na 1 przy każdym takcie zegara dla każdego polecenia są pokazane w tabeli. 2.

Tabela. 2. Sygnały sterujące ustawione w każdej instrukcji / cyklu

	T_1	T_2	T_3
(mv): I_0	RY_{out}, RX_{in}, *Done*		
(mvi): I_1	DIN_{out}, RX_{in}, *Done*		
(add): I_2	RX_{out}, A_{in}	RY_{out}, G_{in}	G_{out}, RX_{in}, *Done*
(sub): I_3	RX_{out}, A_{in}	RY_{out}, G_{in}, *AddSub*	G_{out}, RX_{in}, *Done*

Wskazówka: szkielet kodu procesora pokazano na rys. 2, a, b.

```
module proc (DIN, Resetn, Clock, Run, Done, BusWires);
        input [8:0] DIN;
        input Resetn, Clock, Run;
        output Done;
        output [8:0] BusWires;

        parameter T0 = 2'b00, T1 = 2'b01, T2 = 2'b10, T3 = 2'b11;
        … deklaracja zmiennych

        assign I = IR[1:3];
        dec3to8 decX (IR[4:6], 1'b1, Xreg);
        dec3to8 decY (IR[7:9], 1'b1, Yreg);
```

Rys. 2, a. Szkielet kodu Verilog dla procesora

```
// Zarządzaj tabelą stanów FSM
always @(Tstep_Q, Run, Done)
begin
        case (Tstep_Q)
                T0: // W tej chwili dane są ładowane do IR
                        if (!Run) Tstep_D = T0;
                        else Tstep_D = T1;
                T1….
        endcase
end

// Sterowanie wejściami FSM
always @(Tstep_Q or I or Xreg or Yreg)
begin
        … określenie wartości początkowych
        case (Tstep_Q)
                T0: // zapamiętaj DIN w IR w takcie 0
                begin
                        IRin = 1'b1;
                end
                T1: // określ sygnały w takcie 1
                        case (I)
                        …
                        endcase
                T2: // określ sygnały w takcie 2
                        case (I)
                        …
                        endcase
                T3: // określ sygnały w takcie 3
                        case (I)
                        …
                        endcase
                endcase
end

// sterowanie przerzutnikami FSM
always @(posedge Clock, negedge Resetn)
```

```
        if (!Resetn)
        …

regn reg_0 (BusWires, Rin[0], Clock, R0);
```
… Inicjuj inne rejestry i dodaj moduł sumatora / odejmowania

… określ szynę

endmodule

Рис. **2,b**. Szkielet kodu **Verilog** procesora

Opis zastosowanych modułów pokazano na rys. 2, c.

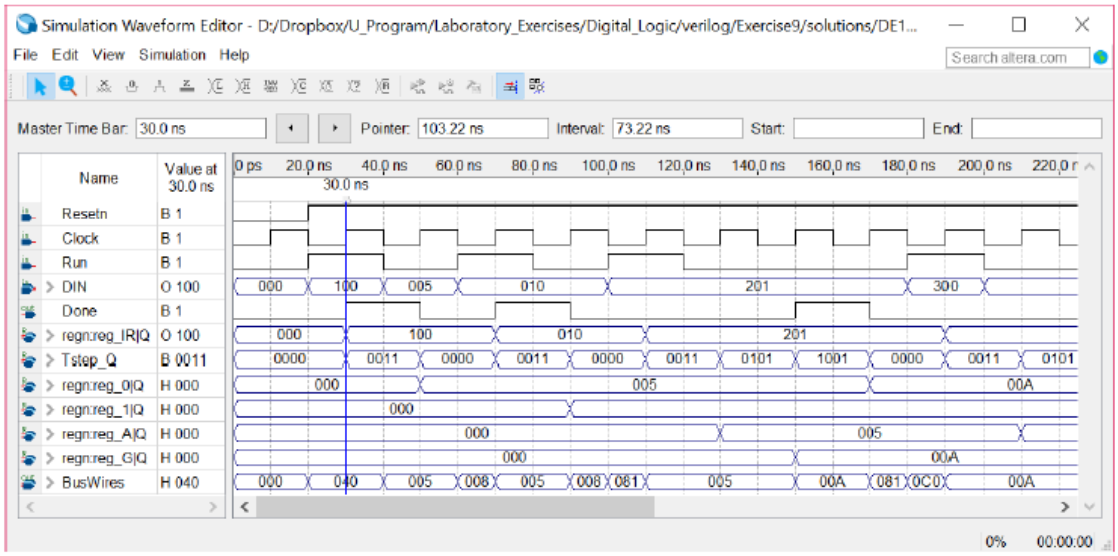
```
module dec3to8(W, En, Y);
        input [2:0] W;
        input En;
        output [0:7] Y;
        reg [0:7] Y;

        always @(W or En)
        begin
                if (En == 1)
                        case (W)
                                3'b000: Y = 8'b10000000;
                                3'b001: Y = 8'b01000000;
                                3'b010: Y = 8'b00100000;
                                3'b011: Y = 8'b00010000;
                                3'b100: Y = 8'b00001000;
                                3'b101: Y = 8'b00000100;
                                3'b110: Y = 8'b00000010;
                                3'b111: Y = 8'b00000001;
                        endcase
                else
                        Y = 8'b00000000;
        end
endmodule

module regn(R, Rin, Clock, Q);
        parameter n = 9;
        input [n-1:0] R;
        input Rin, Clock;
        output [n-1:0] Q;
        reg [n-1:0] Q;

        always @(posedge Clock)
                if (Rin)
                        Q <= R;
endmodule
```

Ryc. 2, c. Moduły podukładów do wykorzystania w procesorze

Wykonaj symulację działania procesora dla kodu:

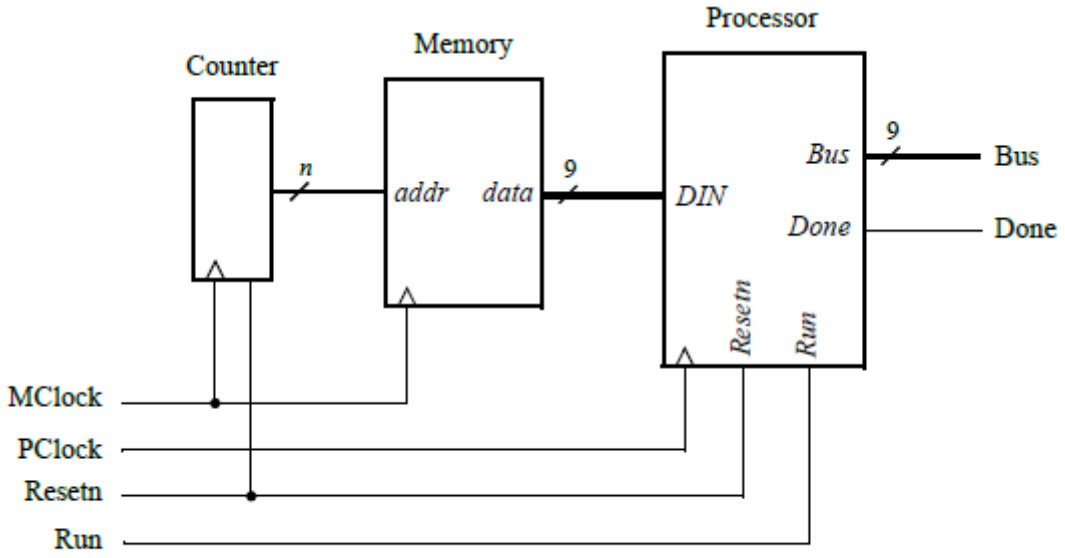
```
mvi  R0, #D     // 30 ns
D = 5           // 50 ns
mv   R1, R0     // 90 ns
add  R0, R1     // 110 ns
sub  R0, R0     // 190 ns
```

Wyniki symulacji powinny pokrywać się z wynikami przedstawionymi na rys. 3.



Rys. 3. Wyniki symulacji procesora

2. Utwórz projektor procesora processor_with_ROM, który jest podłączony do pamięci (rys. 4)



Rys. 4. Podłączenie procesora do pamięci i modułu licznika

Tutaj licznik służy do wyboru adresów komórek pamięci, z których dane wprowadzane są do procesora. Pamięć i procesor są taktowane przez różne sygnały zegarowe: MClock i PClock. Blok pamięci 32 x 9 jest zaimplementowany jako pamięć typu ROM przy użyciu funkcji bibliotecznej z IP Catalog. Przykład pliku inicjującego pamięć pokazano na rys. 5.

```
DEPTH = 32;
WIDTH = 9;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN

00 : 001000000;       % mvi r0,#5   %
01 : 000000101;
02 : 000001000;       % mv r1,r0    %
03 : 010000001;       % add r0, r1    %
04 : 011000000;       % sub r0, r0    %
05 : 000000000;
06 : 000000000;
… (some lines not shown)
1E : 000000000;
1F : 000000000;

END;
```

Rys. 5. Przykład pliku inicjującego pamięć (**MIF**)

Przypisania pinów procesora:

Wyprowadzenia	**Piny**
Run	**SW9**
Reset	**SW0**
MClock	**KEY0**
PClock	**KEY1**
Done	**LED9**
Bus	**LED8-0**

Wykonaj symulację procesora. Upewnij się, że wyniki są takie same jak na Rys. 3.