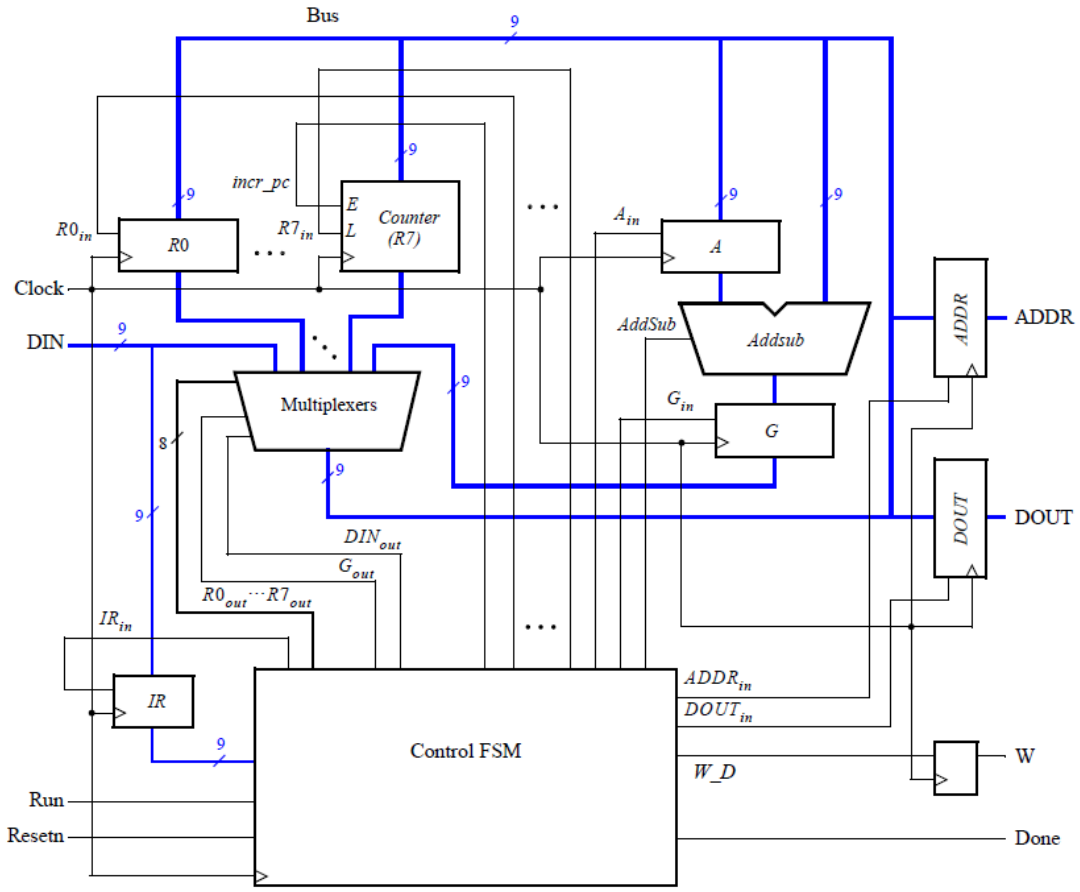
Lab. 11. Ulepszony procesor

Wykonanie

1. Utwórz projekt processor_2 ulepszonej wersji procesora, którego strukturę pokazano na rys. 1.



Rys. 1. Ulepszona wersja procesora.

Zestaw komend nowego procesora jest rozszerzony o następujące komendy:

Tabela. 1. Nowe instrukcje wykonywane w procesorze

Operation	Function performed
ld Rx,$[Ry]$	$Rx \leftarrow [[Ry]]$
st Rx,$[Ry]$	$[Ry] \leftarrow [Rx]$
mvnz Rx, Ry	if G != 0, $Rx \leftarrow [Ry]$

gdzie **ld** (**load**) – zapisuje dane do rejestru **Rx** z komórki pamięci, której adres jest wskazany w rejestrze **Ry**; **st** (**store**) – zawartość rejestru **Rx** zapisuje w komórce pamięci, której adres jest określony w rejestrze **Ry**; **mvnz** (**move if not zero**) zapisuje do rejestru **Rx** zawartość rejestru **Ry**, jeżeli zawartość rejestru **G** nie wynosi zero (to znaczy, że jest wykonywane **mv**, jeśli **G** ≠ **0**).

W strukturze na rys. 1 rejestr R7 jest zastąpiony przez licznik programu (PC), który wskazuje adres polecenia w pamięci. Po zresetowaniu procesora licznik ustawiony jest na adres 0. Na początku każdego polecenia PC jest używany jako adres polecenia w pamięci. Następnie polecenie jest przechowywane w rejestrze instrukcji (rejestr IR), a PC jest automatycznie zwiększany, aby wskazać następne polecenie. Podczas wykonywania polecenia **mvi** rejestr PC wskazuje adres danych, a następnie ponownie zwiększa się kolejno.

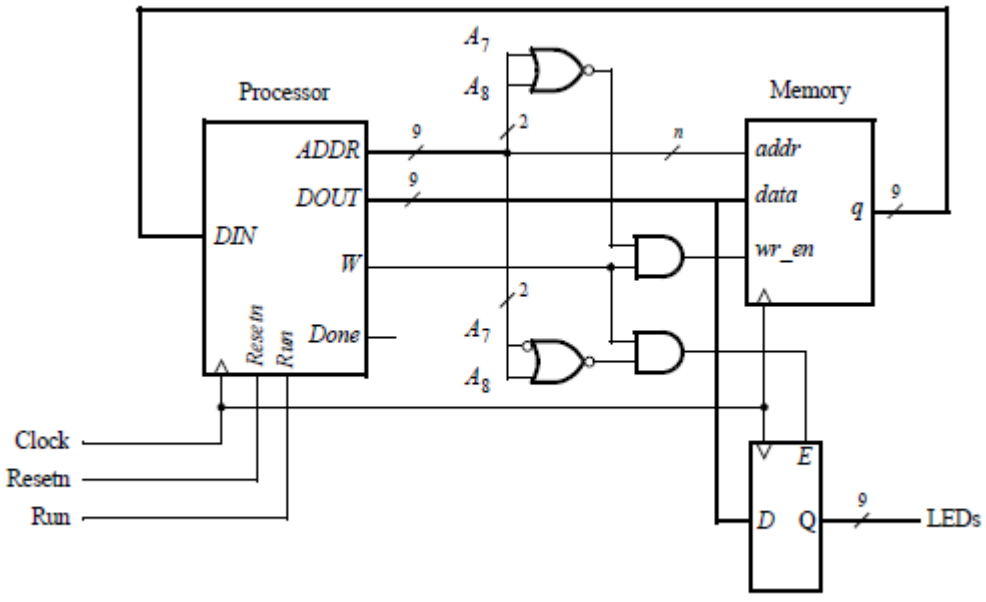
Sygnał sterujący incr_PC jest sygnałem zezwolenia licznika PC. Do PC można również załadować dowolną wartość za pomocą polecenia mv lub mvi. W tym przypadku sygnał sterujący R7in jest wykorzystywany do równoległego załadowania licznika. Bieżąca zawartość PC może zostać skopiowana do innego rejestru za pomocą polecenia mv. Na przykład poniższy fragment kodu implementuje opóźnienie za pomocą pętli, która może być używana w dużym programie do tworzenia opóźnień:

```
mvi   R2,#1
mvi   R4,#10000000      % wartość opóźnienia binarnie
mv    R5,R7             % zapisz adres następnej instrukcji
sub   R4,R2             % dekrementacja wartości opóźnienia
mvnz  R7,R5             % kontynuacja odejmowania,
                        % dopóki wartość opóźnienia nie osiągnie 0
```

Odczytywanie danych z pamięci zewnętrznej i zapisywanie w pamięci odbywa się za pomocą rejestrów ADDR i DOUT, a także sygnałów kontrolnych ADDRin, DOUTin i W_D. Aby odczytać dane z pamięci wystarczy umieścić adres czytelnej komórki w rejestrze ADDR. Aby odczytać (wybrać) polecenie z pamięci, wartość licznika poleceń PC umieszczona jest w rejestrze ADDR. Dane lub polecenia odczytane z pamięci są wysyłane do procesora za pośrednictwem szyny DIN. Aby zapisać do pamięci, adres komórki znajduje się w rejestrze ADDR, a dane, które mają być zapisane w rejestrze DOUT, ustawiony jest sygnał W_D.

Wskazówka: stwórz moduł counter_9_bits_on_sload 9-bitowego licznika z wejściem aktywującym i możliwością synchronicznego ładowania danych. Użyj instancji tego modułu zamiast rejestru R7. Wykonaj symulację projektu procesor_2.

2. Utwórz project_2_with_RAM, którego strukturę pokazano na rys. 2.



Rys. 2. Poprawione połączenie procesora z pamięcią i rejestrem wyjściowym

Tutaj do procesora podłączone są dwa urządzenia zewnętrzne: pamięć Memory 128x9 i 9-bitowy rejestr. Pamięć służy do przechowywania poleceń i danych, a rejestr pozwala obserwować zawartość rejestru procesora DOUT na diodach LED. Magistrala adresowa (wyjście rejestru ADDR) ma 9 bitów. 7 najmniej znaczących bitów A[6: 0] służy do adresowania pamięci, a bity starsze A8 i A7 są używane do wybierania urządzenia zewnętrznego. Gdy A8A7 = 00, wybierana jest pamięć, a gdy A8A7 = 01, wybierany jest rejestr. Aby zaimplementować takie adresowanie w układzie na rys. 2, stosuje się układ wykorzystujący bramki logiczne.

Aby zaimplementować projekt, utwórz blok pamięci RAM128x9, korzystając z funkcji biblioteki z katalogu IP Catalog. Pobierz program za pomocą pliku MIF. Kod programu przedstawiono na rys. 3.

```
DEPTH = 128;
WIDTH = 9;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN

% Ten kod wyświetla licznik (w rejestrze R2) na czerwonych diodach LED.      %
00 : 001001000; %        mvi    R1,#1 // inicjalizacja R                     %
01 : 000000001;
02 : 001010000; %        mvi    R2,#0 // licznik do wyświetlania na LEDs             %
03 : 000000000;
04 : 001011000; % Loop   mvi    R3,#010000000 // R3 = adres rejestru LEDs    %
05 : 010000000;
06 : 101010011; %        st     R2,R3 // zapis do LEDs                        %
07 : 010010001; %        add    R2,R1 // inkrementacja licznika dla LEDs      %
08 : 001011000; %        mvi    R3,#111111111 // wartość opóźnienia           %
09 : 111111111;
0A: 000101111; %         mv     R5,R7 // zapis adresu następnej instrukcji     %
0B: 001100000; % Outer   mvi    R4,#111111111    // zagnieżdżony cykl opóźnienia   %
0C: 111111111;
0D: 000000111; %         mv     R0,R7 // zapis adresu następnej instrukcji     %
0E : 011100001; % Inner  sub    R4,R1 // dekrementacja zmiennej opóźnienia cyklu  %
0F : 110111000; %        mvnz   R7,R0 // przedłużenie cyklu Inner jeśli R4 !=0    %
10 : 011011001; %        sub    R3,R1 // dekrementacja opóźnienia cyklu Outer    %
11 : 110111101; %        mvnz   R7,R5 // przedłużenie cyklu Outer jeśli R3 !=0    %
12 : 001111000; %        mvi    R7,#Loop // ponowne wykonanie                    %
13 : 000000100;
END;
```

Rys. 3. Przykładowy program w pliku inicjującym pamięć (**MIF**)

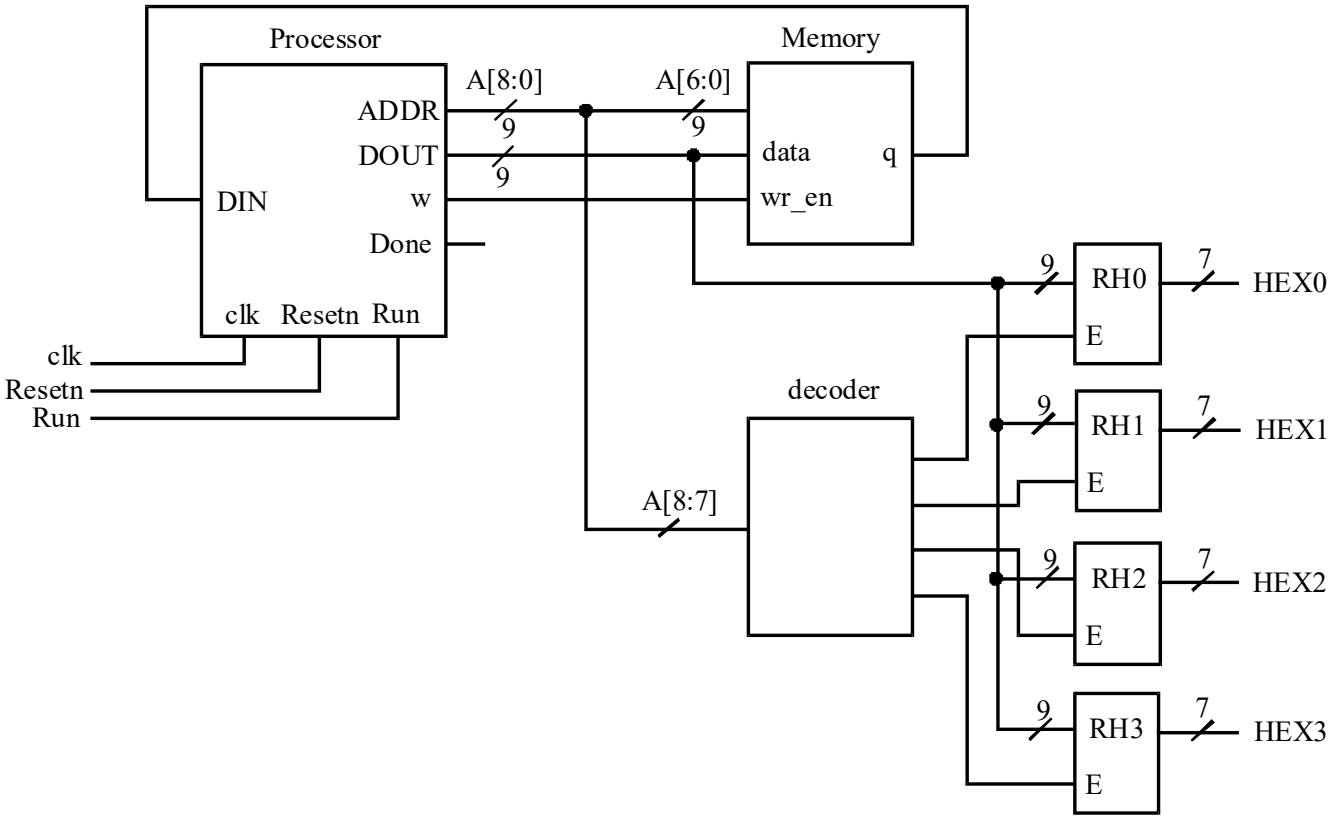
Ten program pokazuje 8-bitową wartość licznika na diodach LED. Program zawiera pętle spowalniające zmiany licznika, aby móc obserwować jego wartości. Zobacz raporty TimeQuest Time Analyzer. Sprawdź, czy schemat procesu działa poprawnie przy częstotliwości 50 MHz. W przeciwnym razie użyj narzędzi Quartus, aby zmienić kod projektu na prędkość 50 MHz.
Użyj przerzutników, aby dopasować asynchroniczne wejście Run do zegara procesora. Użyj następujących przypisań pinów:

Wyprowadzenia	**Piny**

Run	**SW9**
Resetn	**KEY0**
clk	**CLOCK_50**
LEDs	**LEDR8-0**

Wykonaj symulację projektu i przetestuj go na płycie.

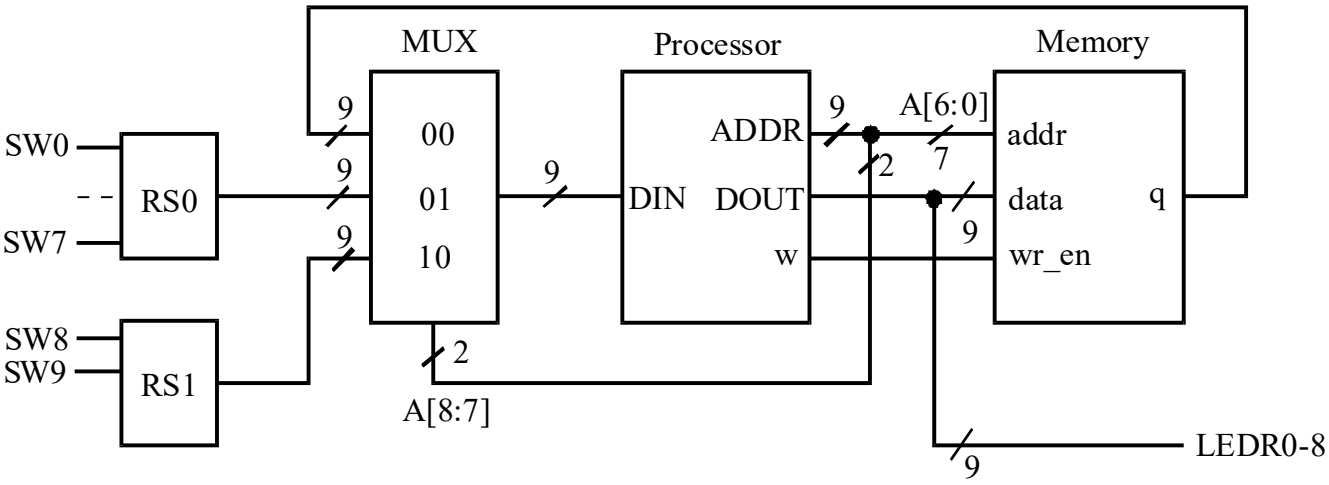
3. Utwórz projekt **processor_with_hex**, którego strukturę pokazano na rys. 4.



Rys. 4. Podłączenie do procesora wyświetlaczy **HEX0-3**

Tutaj linie magistrali adresów A [6: 0] są używane do adresowania lokalizacji pamięci, a A [8: 7] jest używany do adresowania zewnętrznych rejestrów RH0-3. Wyjścia rejestrów bezpośrednio sterują 7-segmentowymi wyświetlaczami HEX0-HEX3 na płycie.
Napisz program dla procesora, który wyświetla słowo na wyświetlaczach, przewija słowa w prawo i w lewo. Symuluj projekt i testuj projekt na płycie.

4. Utwórz projekt **processor_with_SW**, którego strukturę pokazano na rys. 5.



Rys. 5. Podłączanie rejestrów wejściowych do procesora

Ta struktura umożliwia procesorowi odczytanie przełączników SW na płycie. W tym celu do projektu dodane są zewnętrzne wejściowe rejestry 9-bitowe RS0 i RS1, których wejścia są bezpośrednio połączone z przełącznikami SW na płycie (SW0-8 są podłączone do RS0, SW9 do RS1). linie adresowe A8 i A7 są wykorzystywane, aby wybrać źródło szynie DIN Wejście: 00 - Pamięć 01 - RS0, 10 - RS1.

Napisz program dla procesora, który sekwencyjnie skanuje przełączniki SW i wyprowadza ich wartość do diod LED LEDR. Symuluj projekt i testuj projekt na płycie.

Zadania dla chętnych (nieobowiązkowe):

5. Wyszukaj ścieżki krytyczne w obwodzie procesora. Zmień projekt tak, aby obwód działał z maksymalną możliwą częstotliwością.

6. Rozwiń zbiór instrukcji procesora za pomocą logicznych instrukcji AND, OR, poleceń shift i polecenia jump.

7. Napisz polecenie asemblera dla twojego procesora. Powinien automatycznie wygenerować plik MIF z kodu asemblera.