Dydaktyczny Mikroprocesor

Dokumentacja techniczna rdzenia mikroprocesora systemu SCS

ver. 20211001.1 (PL)

ada.brzoza@agh.edu.pl

1. Wprowadzenie

W niniejszej instrukcji omówiono rdzeń mikroprocesora opisany w języku Verilog. Jego kod jest w pełni otwarty i darmowy. Charakteryzuje się on następującymi cechami architektury:

- Wykonuje tylko operacje na rejestrach, a wymiana danych z pamięcią odbywa się poprzez instrukcje load/store,
- Posiada jedną przestrzeń adresową, w której mogą być przechowywane zarówno instrukcje jak i dane, na których są prowadzone obliczenia,
- Ma organizację 4 bitową i jego przestrzeń adresowa ma 16 lokacji (mimo to, szerokość instrukcji wynosi 8 bitów dla wygody implementacji),
- Czas wykonania instrukcji jest stały i wynosi 4 okresy sygnału zegarowego.

2. Interfejs elektryczny

Mikroprocesor $\overline{\text{Foo}}$ posiada bardzo niewielką liczbę wyprowadzeń zewnętrznych. Można tutaj wyróżnić następujące bloki interfejsów:

- wejścia sterujące:
  - `reset_n` – reset, aktywowany stanem logicznym niskim,
  - `clk` – wejście sygnału zegarowego
- interfejs pamięci:
  - `mem_address[3:0]` – 4-bitowy adres wystawiany dla modułu pamięci
  - `mem_data_r[7:0]` – 8-bitowa szyna do odczytu danych z pamięci
  - `mem_data_w[7:0]` – 8-bitowa szyna do zapisu danych do pamięci
  - `mem_we` – zezwolenie na zapis; stan aktywny „1” wraz z narastającym zboczem sygnału zegarowego generuje zapis danych z magistrali `mem_data_w` pod adres ustawiony na `mem_address`.
- interfejs do debugowania
  - `dbg_state[1:0]` – wskazuje aktualny takt danego cyklu
  - `dbg_r0[3:0]` – udostępnia aktualną zawartość rejestru r0,
  - `dbg_r1[3:0]` – udostępnia aktualną zawartość rejestru r1,
  - `dbg_pc[3:0]` – udostępnia aktualny stan rejestru pc.

3. Model programowy

Model programowy rdzenia Foo jest uproszczony do minimum. Składa się on z dwóch rejestrów ogólnego przeznaczenia **R0** i **R1** oraz rejestru licznika programu **PC**.

3.1. Rejestry ogólnego przeznaczenia

Foo udostępnia programiście plik dwóch rejestrów ogólnego przeznaczenia nazwanych R1 i R0. Rejestry te mogą służyć do przechowywania danych, na których prowadzone są obliczenia. Przy pomocy instrukcji przesłań można do nich wpisywać dane z pamięci operacyjnej, a także przepisywać zawarte w nich liczby do pamięci. Oba rejestry mają szerokość 4 bitów.

3.2. Rejestr licznika programu

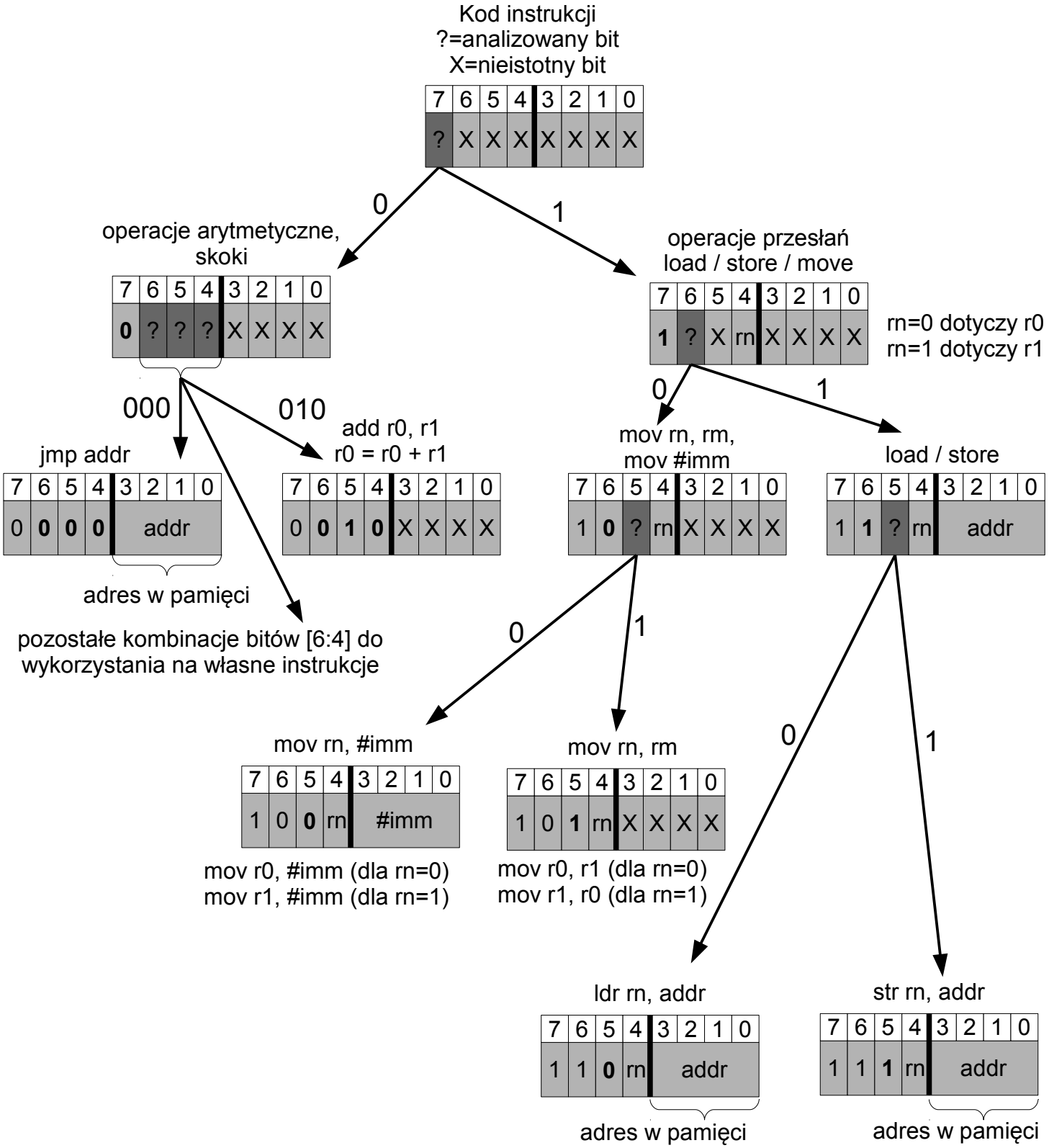
Licznik programu PC przez większość cyklu wskazuje następną instrukcję po aktualnie wykonywanej (wskazuje instrukcję do pobrania w następnym cyklu). Po wykonaniu zerowania (aktywność sygnału `reset_n`) rejestr PC ustawiany jest na 0 i spod tego adresu rozpoczyna pobieranie instrukcji.

3.3. Przestrzeń adresowa i organizacja pamięci

Przestrzeń adresowa ma rozmiar 16 lokacji i mogą się w niej znajdować zarówno instrukcje programu jak i dane. W każdej z 16 lokacji znajduje się jeden bajt pamięci. 4 najbardziej znaczące bity każdego bajta pamięci przeznaczone są do przechowywania kodu instrukcji, a 4 najmniej znaczące do przechowywania stałych natychmiastowych lub danych.

4. Zestaw instrukcji

Poniżej przedstawiono schemat działania dekodera adresowego, natomiast w dalszej części rozdziału omówiono poszczególne instrukcje realizowane przez mikroprocesor.



4.1. Instrukcje skoku

4.1.1. Skok bezwarunkowy do adresu bezwzględnego

Kodowanie:

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	addr			

Zapis: `JMP addr`

Zastosowanie: W wyniku wykonania tej instrukcji mikroprocesor wykona skok do zadanego adresu *addr*. Następną wykonaną instrukcją będzie ta, która znajduje się pod adresem *addr*.

4.2. Instrukcje arytmetyczne

4.2.1. Dodaj

Kodowanie:

7	6	5	4	3	2	1	0
0	**0**	**1**	**0**	X	X	X	X

Zapis: `ADD R0, R1`

Zastosowanie: Wykonuje operację dodawania R0+R1. Wynik wpisywany jest do R0.

4.3. Instrukcje kopiowania i przesłań

4.3.1. Kopiuj zawartość rejestru

Kodowanie:

7	6	5	4	3	2	1	0
1	0	**1**	rn	X	X	X	X

Zapis: `MOV Rn, Rm`

Zastosowanie: kopiuje dane z rejestru Rm do Rn.

4.3.2. Wpisz daną natychmiastową

Kodowanie:

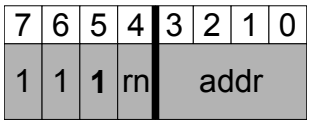
7	6	5	4	3	2	1	0
1	0	**0**	rn	#imm			

Zapis: `MOV Rm,#imm`

Zastosowanie: wpisuje daną natychmiastową *#imm* do rejestru Rm (R0 lub R1).

4.3.3. Zapis do pamięci

Kodowanie:

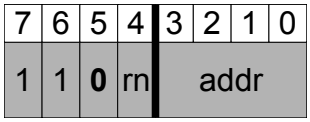


Zapis: `STR Rm, addr`

Zastosowanie: Wpisuje zawartość rejestru Rm do pamięci pod adres wskazany stałą *addr*.

4.3.4. Odczyt z pamięci

Kodowanie:



Zapis: `LDR Rm, addr`

Zastosowanie: Odczytuje daną z pamięci (4 mniej znaczące bity) spod adresu danego stałą *addr* i wpisuje je do rejestru Rm.