

Technika Mikroprocesorowa

Laboratorium 1

Patryk Wojtyczek i Marcin Wąsowicz

1 Obserwacja działania programu testowego

Ćwiczenie to wymagało odpowiedniego skonfigurowania środowiska przy użyciu przygotowanych plików źródłowych, nawiązaniu połączenia z urządzeniem wykorzystując program Putty, zaprogramowaniu układu FPGA i obserwację działania programu.

Do obserwacji programu należało odpowiednio ustawić przełączniki SW[0] i SW[1]:

- SW[0] - Jeśli ustawimy przełącznik SW[0] w pozycji „w dół”, wówczas ręczny sygnał zegarowy będzie podawany po 4 impulsy za każdym wciśnięciem BTN[2]. W przeciwnym przypadku sygnał ma jeden impuls.
- SW[1] - Jeśli przełącznik SW[1] ustawiony jest w pozycji „w dół”, wówczas sygnał zegarowy podajemy ręcznie, wciskając przycisk BTN[2]. W przeciwnym przypadku sygnał jest generowany automatycznie z częstotliwością 1Hz.



```
mem_init_start.coe buffers
1 memory_initialization_radix=16;
2 memory_initialization_vector=
3 80,
4 90,
5 81,
6 92,
7 A0,
8 00,
9 00,
10 00,
11 00,
12 00,
13 00,
14 00,
15 00,
16 00,
17 00,
18 00;

N... mem_init_start.coe 5% 1/18 1:1
```

Rysunek 1: Kod programu

Program wykonuje następujące kroki:

- zapisz do rejestru r0 liczbę 0
- zapisz do rejestru r1 liczbę 0
- zapisz do rejestru r0 liczbę 1
- zapisz do rejestru r1 liczbę 2
- zapisz do rejestru r0 wartość rejestru r1
- zapętl przez instrukcję skoku.

```
c=0 r0=0 r1=0 pc=0 A=0 R=00 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
00
c=1 r0=0 r1=0 pc=1 A=1 R=80 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=2 r0=0 r1=0 pc=2 A=1 R=90 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=3 r0=0 r1=0 pc=2 A=1 R=90 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=0 r0=0 r1=0 pc=2 A=1 R=90 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=1 r0=0 r1=0 pc=2 A=2 R=90 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=2 r0=0 r1=0 pc=3 A=2 R=81 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=3 r0=1 r1=0 pc=3 A=2 R=81 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=0 r0=1 r1=0 pc=3 A=3 R=81 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=1 r0=1 r1=0 pc=3 A=3 R=81 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=2 r0=1 r1=0 pc=4 A=3 R=92 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=3 r0=1 r1=2 pc=4 A=3 R=92 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=0 r0=1 r1=2 pc=4 A=3 R=92 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=1 r0=1 r1=2 pc=4 A=4 R=92 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=2 r0=1 r1=2 pc=5 A=4 R=A0 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=3 r0=2 r1=2 pc=5 A=4 R=A0 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=0 r0=2 r1=2 pc=5 A=4 R=A0 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=1 r0=2 r1=2 pc=5 A=5 R=A0 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=2 r0=2 r1=2 pc=6 A=5 R=00 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=3 r0=2 r1=2 pc=0 A=5 R=00 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
c=0 r0=2 r1=2 pc=0 A=5 R=00 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00 00
```

Rysunek 2: Działanie programu

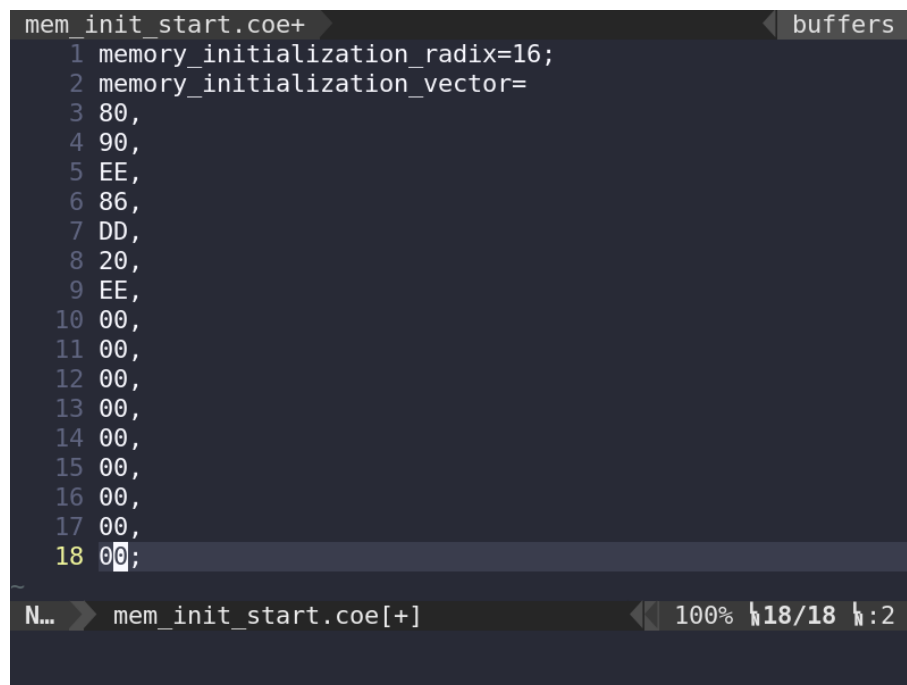
Analizując poszczególne linie widzimy jak instrukcje zmieniają się zgodnie z programem z pliku .coe, jak zmieniają się wartości rejestrów, a także widzimy dump całej pamięci RAM.

2 Własny program

Zadanie polegało na napisaniu programu, który wykonuje instrukcje:

- Wyczyść zawartość rejestru r0, wpisując do niego wartość 0
- Wyczyść zawartość rejestru r1, wpisując do niego wartość 0
- Do rejestru r0 wpisać liczbę 0x6
- Dodać do rejestru R0 wartość przechowywaną w pamięci RAM pod adresem 13
- Umieścić zawartość rejestru R0 w pamięci RAM w komórce o adresie 14
- zapętlić

Do wykonanie tego zadania wystarczyło lekko zmodyfikować plik .coe bazując na dokumentacji mikroprocesora (lub plik .cpu, gdzie również wszystko było opisane)

A screenshot of a text editor window titled 'mem_init_start.coe+'. The editor shows a memory initialization vector with 18 entries, indexed from 1 to 18. The values are: 80, 90, EE, 86, DD, 20, EE, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, and 00. The last entry, '18 00;', is highlighted in yellow. The status bar at the bottom shows 'N... mem_init_start.coe[+] 100% 18/18 2'.

```
mem_init_start.coe+ buffers
1 memory_initialization_radix=16;
2 memory_initialization_vector=
3 80,
4 90,
5 EE,
6 86,
7 DD,
8 20,
9 EE,
10 00,
11 00,
12 00,
13 00,
14 00,
15 00,
16 00,
17 00,
18 00;
N... mem_init_start.coe[+] 100% 18/18 2
```

Rysunek 3: Kod własnego programu

[illegible]

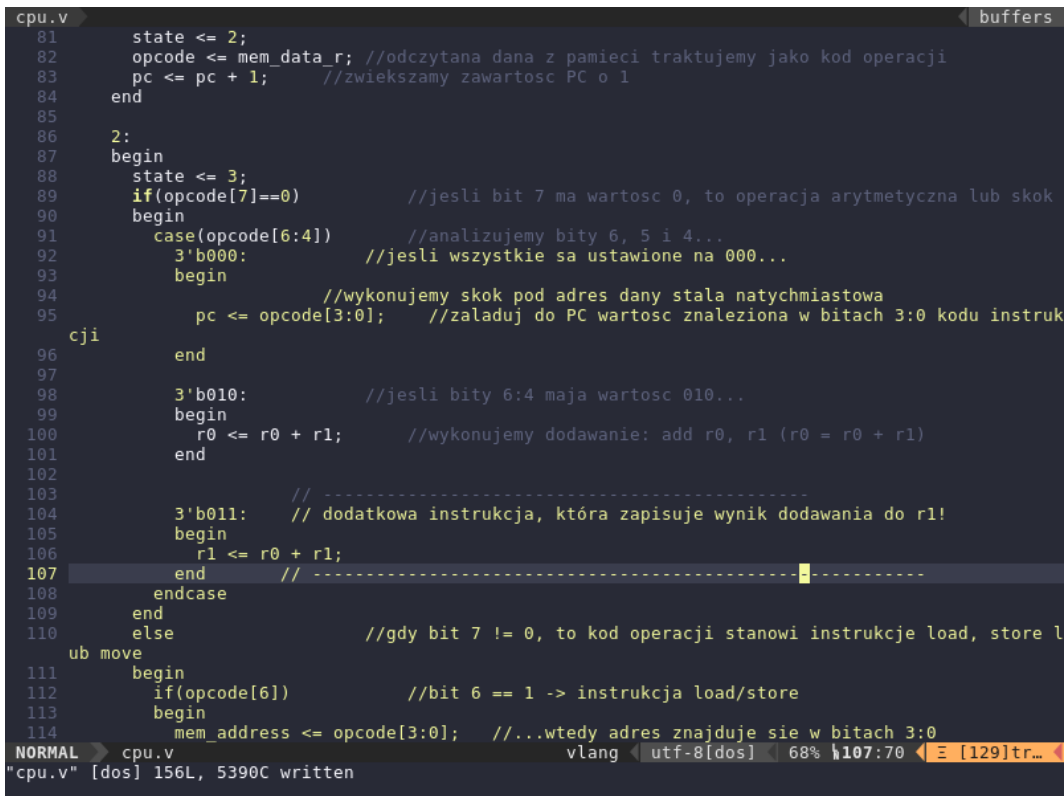
$\frac{1}{\sqrt{2}}$

3 Tworzenie nowej instrukcji procesora

3.1 Dodawanie

Celem tego ćwiczenia było dodanie nowej instrukcji procesora, która za pomocą 4bitu operacji pozwalałaby sprecyzować do którego rejestru mamy zapisać $r0 + r1$.

Instrukcja, która dodaje oba rejestry i zapisuje w $r0$ to 0010. Zauważamy, że instrukcja **0011** jest wolna więc będzie ona odpowiedzialna za zapis sumy rejestrów do rejestru $r1$.



```
cpu.v buffers
81 state <= 2;
82 opcode <= mem_data_r; //odczytana dana z pamieci traktujemy jako kod operacji
83 pc <= pc + 1; //zwiększamy zawartosc PC o 1
84 end
85
86 2:
87 begin
88 state <= 3;
89 if(opcode[7]==0) //jesli bit 7 ma wartosc 0, to operacja arytmetyczna lub skok
90 begin
91 case(opcode[6:4]) //analizujemy bity 6, 5 i 4...
92 3'b000: //jesli wszystkie sa ustawione na 000...
93 begin
94 //wykonujemy skok pod adres dany stala natychmiastowa
95 pc <= opcode[3:0]; //zaladuj do PC wartosc znaleziona w bitach 3:0 kodu instrukcji
96 end
97
98 3'b010: //jesli bity 6:4 maja wartosc 010...
99 begin
100 r0 <= r0 + r1; //wykonujemy dodawanie: add r0, r1 (r0 = r0 + r1)
101 end
102
103 // -----
104 3'b011: // dodatkowa instrukcja, która zapisuje wynik dodawania do r1!
105 begin
106 r1 <= r0 + r1;
107 end // -----
108 endcase
109 end
110 else //gdy bit 7 != 0, to kod operacji stanowi instrukcje load, store lub move
111 begin
112 if(opcode[6]) //bit 6 == 1 -> instrukcja load/store
113 begin
114 mem_address <= opcode[3:0]; //...wtedy adres znajduje sie w bitach 3:0
115
116 vlang utf-8[dos] 68% 107:70 129]tr...
NORMAL cpu.v
"cpu.v" [dos] 156L, 5390C written
```

Rysunek 5: Zmodyfikowany plik `cpu.v` z instrukcją dodawania

3.2 Mnożenie

Następnie należało zmodyfikować `cpu.v` tak, aby umożliwić wykonywanie mnożenia. Problemem, który należy tu rozwiązać jest overflow przy mnożeniu dwóch liczb. Możemy to obejść zapisując wynik do obu rejestrów (sumarycznie mają dość pamięci, żeby pomieścić wynik).

```
cpu.v buffers
90 begin
91     case(opcode[6:4]) //analizujemy bity 6, 5 i 4...
92     3'b000: //jesli wszystkie sa ustawione na 000...
93         begin
94             //wykonujemy skok pod adres dany stala natychmiastowa
95             pc <= opcode[3:0]; //załaduj do PC wartosc znaleziona w bitach 3:0 kodu instrukcji
96         end
97     3'b010: //jesli bity 6:4 maja wartosc 010...
98         begin
99             r0 <= r0 + r1; //wykonujemy dodawanie: add r0, r1 (r0 = r0 + r1)
100        end
101    3'b011: // dodatkowa instrukcja, która zapisuje wynik dodawania do r1!
102        begin
103            r1 <= r0 + r1;
104        end
105    3'b110: // Dodatkowe instrukcje ktore wykonuja mnozenie i zapisują wynik
106        // do obu rejestrów ze wzgledu na overflow
107        begin
108            {r0, r1} <= r0 * r1
109        end
110    3'b111:
111        begin
112            {r0, r1} <= r0 * r1
113        end
114    // -----
115    endcase
116    end
117    else //gdy bit 7 != 0, to kod operacji stanowi instrukcje load, store lub move
118        ub move
119    end
120 NORMAL cpu.v vlang utf-8[dos] 71% 120:83 [142]tr...
```

Rysunek 6: Zmodyfikowany plik `cpu.v` z instrukcją mnożenia