Sprawozdanie

Prosty mikroprocesor i system komputerowy w logice programowalnej

Autorzy: Radosław Niżnik, Adrian Żerebiec

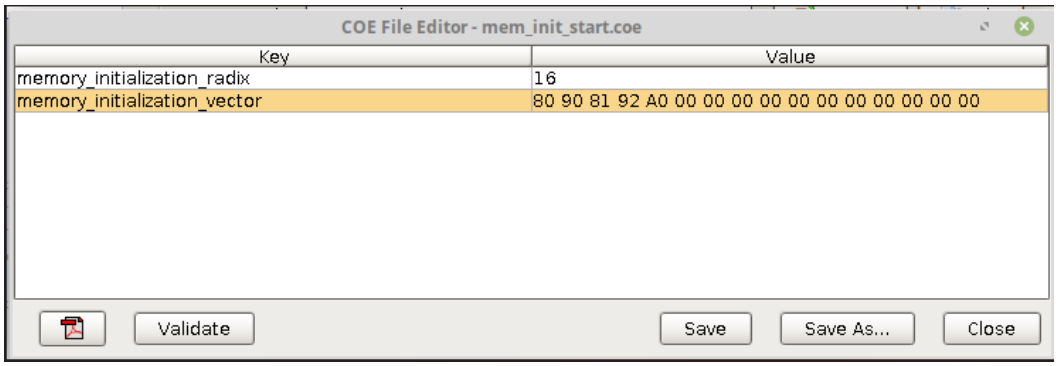
1. Przebieg laboratorium:

a. Wczytanie projektu, pobranego z systemu UPEL, w programie Vivado służącego do zaprogramowania układu FPGA-ZYBO
b. Sprawdzenie połączenia płytki z komputerem
c. Nawiązanie połączenia z płytką przez program PuTTy
d. Wgranie programu testowego, a następnie obserwacja jego działania
e. Napisanie własnego programu zgodnie z instrukcją do laboratorium.
f. Utworzenie nowych instrukcji do mikroprocesora (dodawanie r1 = r0 + r1 oraz mnożenia)

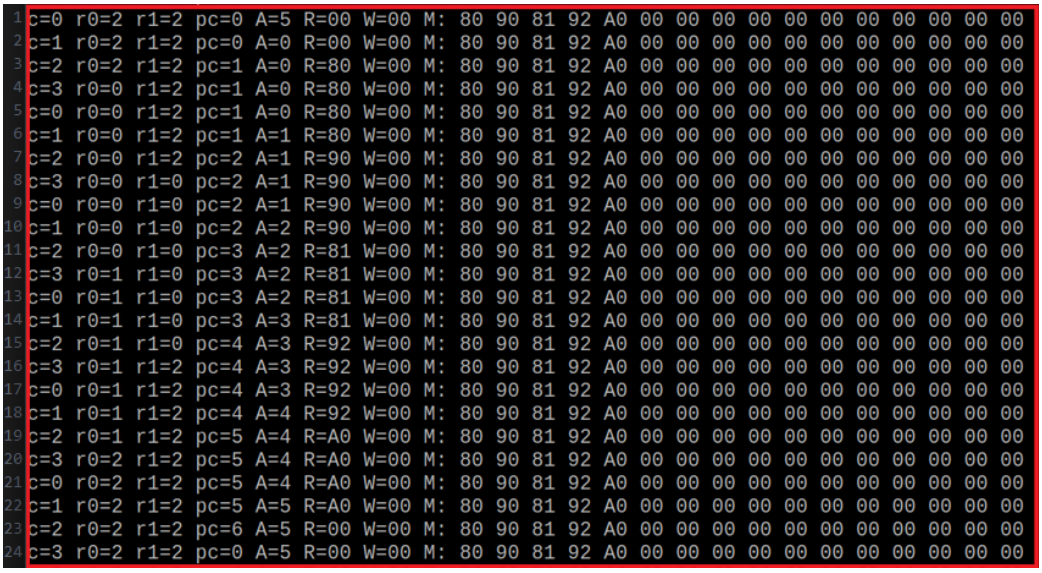
2. Wgranie programu testowego, a następnie obserwacja jego działania

Początkowo wczytaliśmy projekt, pobrany z systemu UPEL, w programie Vivado. Dodatkowo nawiązaliśmy połączenie z płytką poprzez program PuTTy. Dzięki temu mogliśmy analizować program wgrany do pamięci.

Po wglądzie do pamięci widzimy następujące okno:

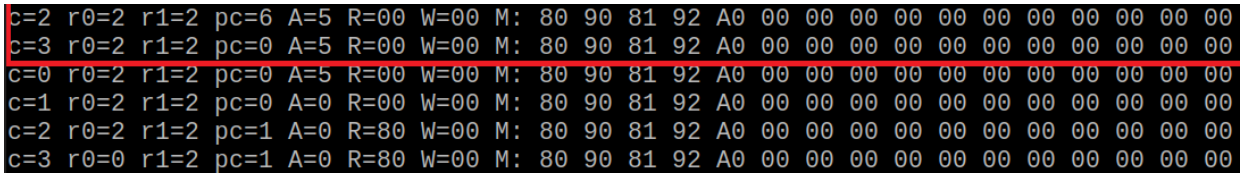


Po przeanalizowaniu jego treści (podświetlona część) widzimy, że program testowy został poprawnie wgrany. Wynik jego działania możemy zobaczyć na poniższym obrazku:



Możemy zobaczyć, że od linii 1 do 4, wykonuje się jeden cykl instrukcji, w naszym mikroprocesorze składają się na niego 4 cykle zegarowe.
W tabeli poniżej znajdziemy przeanalizowane kolejne linie kodu wraz z instrukcjami, które się wykonują:

Adres	Instrukcja	Kod bin.	Kod hex.	Linie kodu odpowiedzialne za instrukcje
00	mov r0, #0	1000 0000	80	1-4 wartość r0 zmieniła się na 0
01	mov r1, #0	1001 0000	90	5-8 wartość r1 zmieniła się na 0
02	mov r0, #1	1000 0001	81	9-12 wartość r0 zmieniła się na 1
03	mov r1, #2	1001 0010	92	13-16 wartość r1 zmieniła się na 2
04	mov r0,r1	1010 0000	A0	17-20 wartość r0 zmienia się na 2, czyli wartość, która była w r1
05	jmp #0	0000 0000	00	Komentarz poniżej



Widzimy, że po wykonaniu instrukcji o adresie 05 program zapętla się i znowu zaczyna wykonywać instrukcje o adresie 00.

3. Napisanie własnego programu.

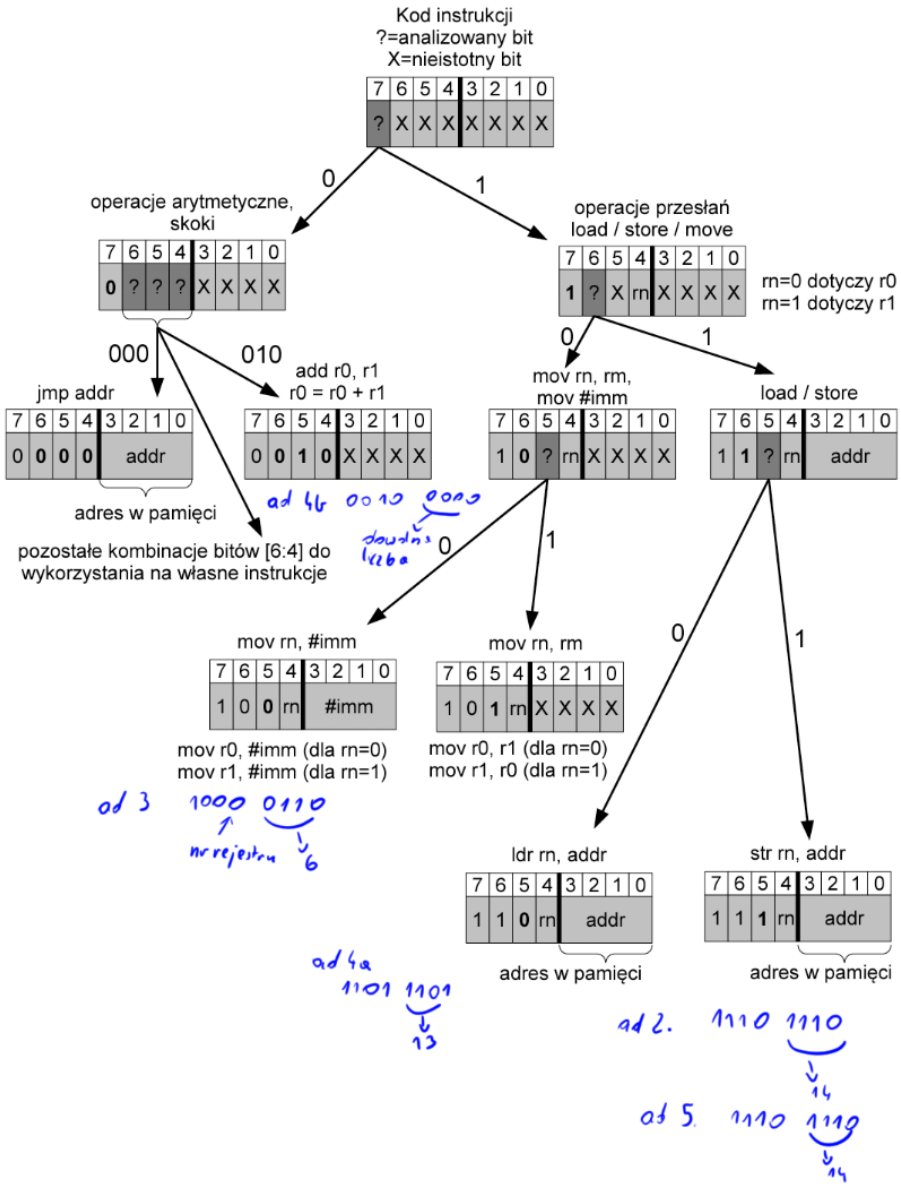
Kolejnym poleceniem było napisanie własnego programu o następujących zadaniach:

0. Wpisanie liczby 0x05 do komórki pamięci o adresie 13,
1. Wyczyścić zawartość rejestrów R0 i R1 wpisując do nich wartość 0,
2. Wyczyścić zawartość komórki pamięci o adresie 14 wpisując do niej wartość 0,
3. Do rejestru R0 wpisać liczbę 0x6,
4. Dodać do rejestru R0 wartość przechowywaną w pamięci RAM pod adresem 13,
5. Umieścić zawartość rejestru R0 w pamięci RAM w komórce o adresie 14,
6. Zapętlić wykonywanie programu.

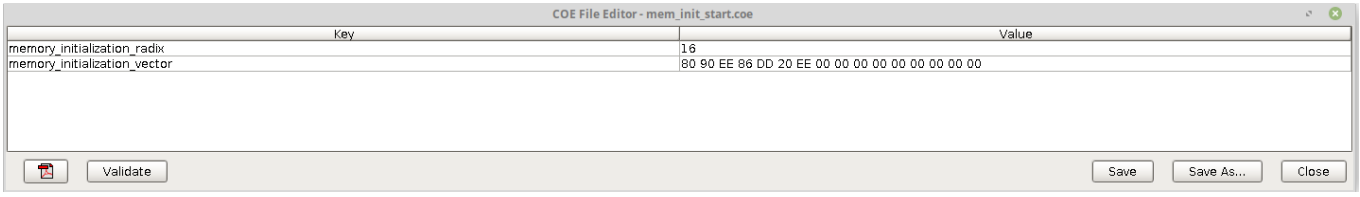
W tabelce zamieszczono informacje o nowo napisanym programie. Możemy zobaczyć instrukcje, ich kody w dwóch systemach: binarnym oraz szesnastkowym, oraz co dokładnie robi dany kod.

Adres	Instrukcja	Kod bin.	Kod hex.	Co robi dany kod
00	mov r0, #0	1000 0000	80	wartość r0 zmieniła się na 0
01	mov r1, #0	1001 0000	90	wartość r1 zmieniła się na 0
02	str R0, 14	1110 1110	EE	przeniesienie wartości z R0 do komórki 14 (w naszym wypadku 0)
03	mov r0, #6	1000 0110	86	wartość r0 zmienia się na 6
04	ldr r1, 13	1101 1101	DD	musimy pierw wczytać zawartość komórki 13 do rejestru r1,
05	add r0, r1	0010 0000	20	Dodanie do rejestru r0, rejestr r1
06	str r1, 14	1110 1110	EE	Umieszczenie zawartości rejestru r0 do komórki o adresie 14
07	jmp 0#	0000 0000	00	Skok do komórki o adresie 00
13			05	Wpisanie na sztywno wartości

Poniżej przedstawiono rozumowanie, które towarzyszyło nam podczas pisania własnego programu.



Finalny program został dodany do pamięci oraz uruchomiony w celu sprawdzenia jego poprawności. Jego działanie śledziliśmy z pomocą programu PuTTy. Poniżej znajduje się zawartość pamięci, czyli nasz program:



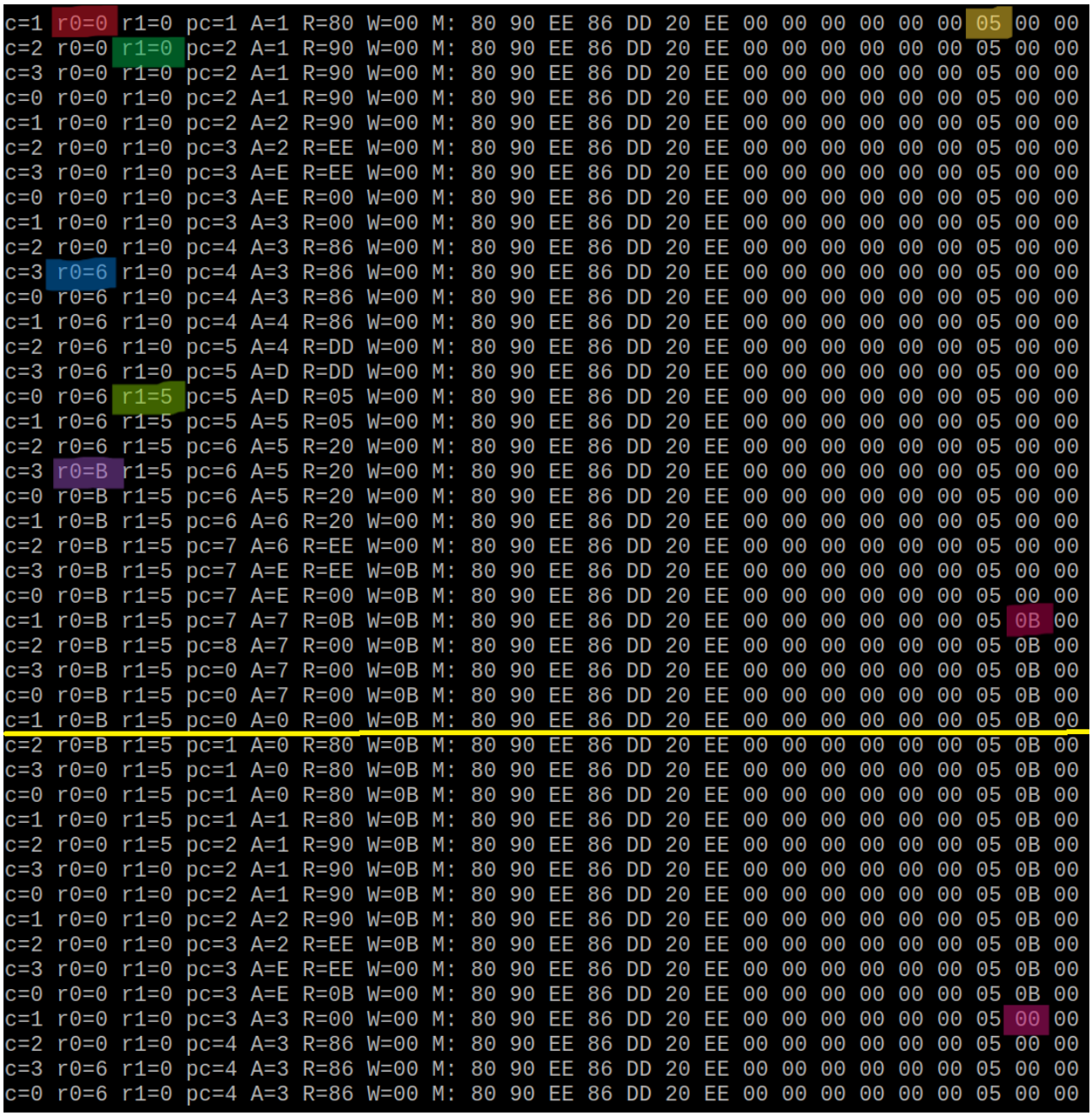
Jak widzimy na poniższym zrzucie ekranu, program działa poprawnie. W komórce 14 pojawia się wynik naszego dodawania. Dodatkowo śledząc zawartości rejestrów, widzimy, że wszystkie kroki zostały wykonane w dobrej kolejności, a całość spełnia polecenie.

```
c=1 r0=0 r1=0 pc=1 A=1 R=80 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=2 r0=0 r1=0 pc=2 A=1 R=90 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=3 r0=0 r1=0 pc=2 A=1 R=90 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=0 r0=0 r1=0 pc=2 A=1 R=90 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=1 r0=0 r1=0 pc=2 A=2 R=90 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=2 r0=0 r1=0 pc=3 A=2 R=EE W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=3 r0=0 r1=0 pc=3 A=E R=EE W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=0 r0=0 r1=0 pc=3 A=E R=00 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=1 r0=0 r1=0 pc=3 A=3 R=00 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=2 r0=0 r1=0 pc=4 A=3 R=86 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=3 r0=6 r1=0 pc=4 A=3 R=86 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=0 r0=6 r1=0 pc=4 A=3 R=86 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=1 r0=6 r1=0 pc=4 A=4 R=86 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=2 r0=6 r1=0 pc=5 A=4 R=DD W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=3 r0=6 r1=0 pc=5 A=D R=DD W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=0 r0=6 r1=5 pc=5 A=D R=05 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=1 r0=6 r1=5 pc=5 A=5 R=05 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=2 r0=6 r1=5 pc=6 A=5 R=20 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=3 r0=B r1=5 pc=6 A=5 R=20 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=0 r0=B r1=5 pc=6 A=5 R=20 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=1 r0=B r1=5 pc=6 A=6 R=20 W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=2 r0=B r1=5 pc=7 A=6 R=EE W=00 M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=3 r0=B r1=5 pc=7 A=E R=EE W=0B M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=0 r0=B r1=5 pc=7 A=E R=00 W=0B M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 00 00
c=1 r0=B r1=5 pc=7 A=7 R=0B W=0B M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 0B 00
c=2 r0=B r1=5 pc=8 A=7 R=00 W=0B M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 0B 00
c=3 r0=B r1=5 pc=0 A=7 R=00 W=0B M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 0B 00
c=0 r0=B r1=5 pc=0 A=7 R=00 W=0B M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 0B 00
c=1 r0=B r1=5 pc=0 A=0 R=00 W=0B M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 0B 00
c=2 r0=B r1=5 pc=1 A=0 R=80 W=0B M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 0B 00
c=3 r0=0 r1=5 pc=1 A=0 R=80 W=0B M: 80 90 EE 86 DD 20 EE 00 00 00 00 00 00 05 0B 00
```

Poniżej dodatkowo zamieszczamy zrzut ekranu programu PuTTy, w którym różnymi kolorami zaznaczyliśmy kolejne wykonywane operacje (dla uwagi opisy poszczególnych zadań są w kolorze podobnym do zaznaczonych na poniższym zrzucie):

- Żółty - ustawienie zawartości komórki 13 na 5,
- Czerwony - ustawienie r0 na 0,
- Zielony - ustawienie r1 na 0,
- Niebieski - ustawienie r0 na 6,
- Limonkowy - ustawienie r1 na wartość z komórki 13,
- Fioletowy - dodanie r1 do r0,
- Wiśniowy - umieszczenie wartości z r0 w komórce 14,
- Śliwkowy - wyczyszczenie komórki 14 poprzez wpisanie w nią 0 .

Dodatkowo żółta linia oznacza rozpoczęcie programu od nowa (zapętlenie).



Teraz widzimy, że wszystko na pewno dobrze się wykonało. Zatem teraz ze stuprocentową pewnością możemy stwierdzić, iż program jest poprawny.

4. Utworzenie nowej instrukcji mikroprocesora

Następnie należało utworzyć nowe instrukcje dodawania i mnożenia.

A. Dodawanie

Musieliśmy stworzyć nową instrukcję, która pozwoli nam na operacje add r1,r0. W związku z tym w pliku cpu.v należało dodać nowego case'a. Operacja add r0,r1 ma postać 0010xxxx, gdzie x reprezentuje nieistotny bit. Korzystając z tej informacji, tworzymy nową operację add r1,ro, która będzie miała postać 0011xxxx. Kod nowej operacji znajduje się poniżej (zaznaczony czarnymi klamrami):

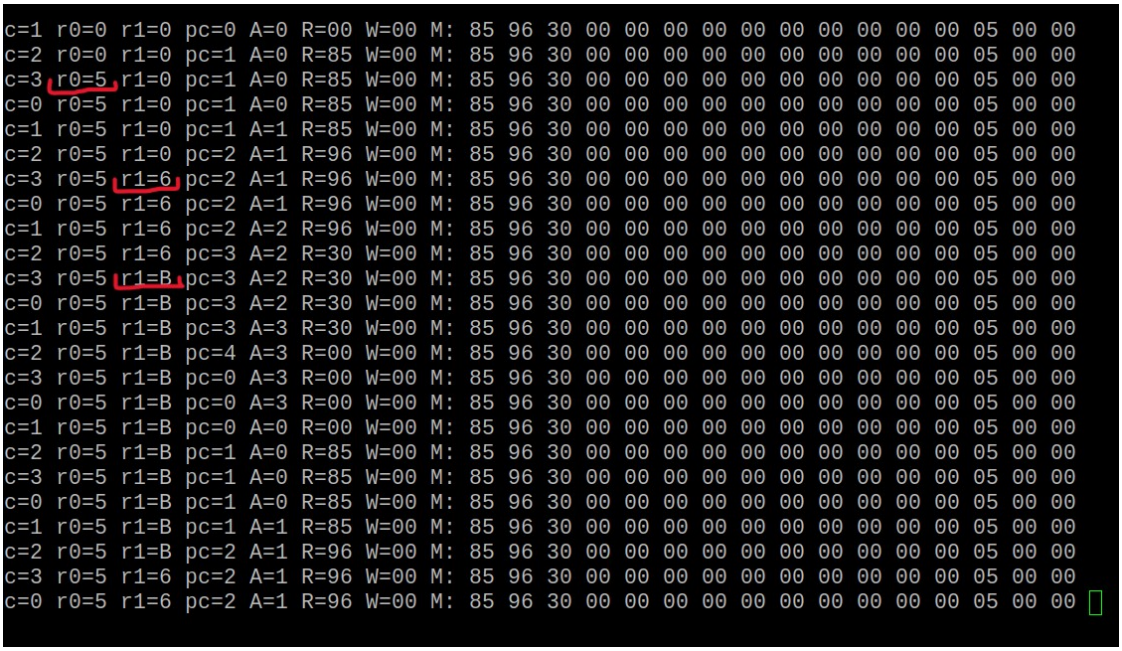
```
2:
begin
    state <= 3;
    if(opcode[7]==0)                //jesli bit 7 ma wartosc 0, to operacja arytmetyczna lub skok
    begin
        case(opcode[6:4])           //analizujemy bity 6, 5 i 4...
            3'b000:                 //jesli wszystkie sa ustawione na 000...
            begin
                                    //wykonujemy skok pod adres dany stala natychmiastowa
                pc <= opcode[3:0];  //zaladuj do PC wartosc znaleziona w bitach 3:0 kodu instrukcji
            end

            3'b010:                 //jesli bity 6:4 maja wartosc 010...
            begin
                r0 <= r0 + r1;      //wykonujemy dodawanie: add r0, r1 (r0 = r0 + r1)
            end

            3'b011:                 //jesli bity 6:4 maja wartosc 011...
            begin
                r1 <= r0 + r1;        //wykonujemy dodawanie: add r1, r0 (r1 = r0 + r1)
            end

        endcase
```

Aby sprawdzić poprawność nowej instrukcji, musieliśmy stworzyć nowy program testowy. Wyglądał on w sposób następujący:

```
M: 85 96 30 00 00 00 00 00 00 00 00 00 00 05 00 00
```

Zatem chcieliśmy, aby program początkowo ustawił wartości: r0 = 5, r1 = 6, a następnie dodał r0 do r1. Oczekujemy, więc że na koniec wartość, którą zobaczymy przypisaną do r1, będzie wynosić 11, zatem w systemie szesnastkowym w tym miejscu musi pojawić się wartość B.

Wynik działania naszego programu testującego wygląda następująco:



Program początkowo ustawia r0 = 5, następnie zmienia wartość r1 na 6. Po tym dodaje nam wartość r0 do r1, dzięki czemu jego nowa wartość to B (wykonanie odpowiednich operacji oznaczone na kolor czerwony na rysunku powyżej). Zatem program działa poprawnie, co oznacza, że nowa operacja dodawania jest prawidłowa.

B. Mnożenie

Drugą instrukcję, którą należało dodać, było mnożenie: mul r0,r1. Wynik mnożenia należało umieścić w rejestrach r0 oraz r1. W związku z tym musieliśmy rozdzielić wynik mnożenia po 4 bity, czyli 4 najstarsze bity wpiszemy do rejestru r0 w systemie szesnastkowym, a cztery najmłodsze do r1 również w systemie szesnastkowym. Musimy do tego wykorzystać operacje dzielenia oraz dzielenia modulo, aby uzyskać potrzebne nam bity. Kod nowej instrukcji o postaci 0011xxxx znajduje się na zrzucie ekranu poniżej (zaznaczona czarną klamrą):

```
2:
begin
    state <= 3;
    if(opcode[7]==0)                    //jesli bit 7 ma wartosc 0, to operacja arytmetyczna lub skok
    begin
        case(opcode[6:4])               //analizujemy bity 6, 5 i 4...
            3'b000:                     //jesli wszystkie sa ustawione na 000...
            begin
                                        //wykonujemy skok pod adres dany stala natychmiastowa
                pc <= opcode[3:0];      //zaladuj do PC wartosc znaleziona w bitach 3:0 kodu instrukcji
            end

            3'b010:                     //jesli bity 6:4 maja wartosc 010...
            begin
                r0 <= r0 + r1;          //wykonujemy dodawanie: add r0, r1 (r0 = r0 + r1)
            end
            3'b100:
            begin
               r0 <= (r0*r1)/16;
               r1 <= (r0*r1)%16;
            end
        endcase
```

Zatem dzięki operacjom dzielenia przez 16 oraz modulo 16 uzyskujemy odpowiednie bity. W celu sprawdzenia poprawności napisaliśmy dwa programy.

Pierwszy z programów polegał na pomnożeniu r0 = 10 oraz r1 = 6. Oczekiwany przez nas wynik wynosi 60, czyli w systemie szesnastkowym dokładnie 3C.

```
c=1 r0=0 r1=0 pc=0 A=0 R=8A W=00 M: 8A 96 40 00 00 00 00 00 00 00 00 00 00 05 00 00
c=2 r0=0 r1=0 pc=1 A=0 R=8A W=00 M: 8A 96 40 00 00 00 00 00 00 00 00 00 00 05 00 00
c=3 r0=A r1=0 pc=1 A=0 R=8A W=00 M: 8A 96 40 00 00 00 00 00 00 00 00 00 00 05 00 00
c=0 r0=A r1=0 pc=1 A=0 R=8A W=00 M: 8A 96 40 00 00 00 00 00 00 00 00 00 00 05 00 00
c=1 r0=A r1=0 pc=1 A=1 R=8A W=00 M: 8A 96 40 00 00 00 00 00 00 00 00 00 00 05 00 00
c=2 r0=A r1=0 pc=2 A=1 R=96 W=00 M: 8A 96 40 00 00 00 00 00 00 00 00 00 00 05 00 00
c=3 r0=A r1=6 pc=2 A=1 R=96 W=00 M: 8A 96 40 00 00 00 00 00 00 00 00 00 00 05 00 00
c=0 r0=A r1=6 pc=2 A=1 R=96 W=00 M: 8A 96 40 00 00 00 00 00 00 00 00 00 00 05 00 00
c=1 r0=A r1=6 pc=2 A=2 R=96 W=00 M: 8A 96 40 00 00 00 00 00 00 00 00 00 00 05 00 00
c=2 r0=A r1=6 pc=3 A=2 R=40 W=00 M: 8A 96 40 00 00 00 00 00 00 00 00 00 00 05 00 00
c=3 r0=3 r1=C pc=3 A=2 R=40 W=00 M: 8A 96 40 00 00 00 00 00 00 00 00 00 00 05 00 00
c=0 r0=3 r1=C pc=3 A=2 R=40 W=00 M: 8A 96 40 00 00 00 00 00 00 00 00 00 00 05 00 00
```

Jak widzimy, uzyskany wynik jest poprawny i zgodny z oczekiwaniami.

W tym miejscu warto wspomnieć również o jednym problemie, który pojawił się nam podczas pisania kodu. Język SystemVerilog dopuszcza dwa rodzaje znaku przypisania, czyli “=” oraz “<=”. Początkowo użyliśmy “=”, co spowodowało, że wynik w rejestrze r0 był poprawny natomiast w r1 już nie. Działo się tak, gdyż “=” przypisał nam od razu do r0 wynik działania. W następnej operacji obliczenia wykonywały się już z uzyskanym nowym r0. Operator “<=” nie przypisuje od razu wyniku do r0, ale dzieje się to dopiero później, więc r1 obliczane jest ze starym r0, co pozwala nam poprawnie napisać instrukcję.

Dodatkowo napisaliśmy drugi program, aby upewnić się, że wszystko działa prawidłowo. Chcieliśmy w nim sprawdzić mnożenie r0 = 5 oraz r1 = 6, czyli wynikiem powinno być 30 - w systemie szesnastkowym 1E.

```
c=1 r0=0 r1=0 pc=0 A=0 R=96 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
c=2 r0=0 r1=0 pc=1 A=0 R=85 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
c=3 r0=5 r1=0 pc=1 A=0 R=85 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
c=0 r0=5 r1=0 pc=1 A=0 R=85 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
c=1 r0=5 r1=0 pc=1 A=1 R=85 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
c=2 r0=5 r1=0 pc=2 A=1 R=96 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
c=3 r0=5 r1=6 pc=2 A=1 R=96 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
c=0 r0=5 r1=6 pc=2 A=1 R=96 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
c=1 r0=5 r1=6 pc=2 A=2 R=96 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
c=2 r0=5 r1=6 pc=3 A=2 R=40 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
c=3 r0=1 r1=E pc=3 A=2 R=40 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
c=0 r0=1 r1=E pc=3 A=2 R=40 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
c=1 r0=1 r1=E pc=3 A=3 R=40 W=00 M: 85 96 40 00 00 00 00 00 00 00 00 00 05 00 00
```

W tym przypadku rezultat działania programu również jest poprawny, zatem instrukcja jest napisana prawidłowo.

5. Wnioski

Płytka ZYBO-FPGA pozwala nam, przy wykorzystaniu programu Vivado stworzyć własne środowisko uruchomieniowe. Możemy dzięki temu pisać własne programy z wykorzystaniem wbudowanych instrukcji. Co więcej, z pomocą języka SystemVerilog, możemy stworzyć nowe instrukcje takie jak mnożenie czy dodawanie. W tym celu należy zmodyfikować w odpowiedni sposób zawartość pliku cpu.v. Po napisaniu nowych instrukcji warto napisać programy testujące ich poprawność. Pozwoli nam to na weryfikację nowych instrukcji. To wszystko daje nam możliwość stworzenia własnego prostego komputera, wykonującego wcześniej napisane przez nas programy.