

# Technika mikroprocesorowa

Sprawozdanie z laboratorium 1 - Simple Computer System

Radosław Benedykciński, Maciej Wiśniewski

## 1. Cel ćwiczenia:

- a. Poznanie ogólnej zasady funkcjonowania mikroprocesorów,
  - b. Poznanie przykładowego sposobu implementacji mikroprocesora o minimalnej funkcjonalności w językach Python i Verilog,
  - c. Sprawdzenie jego działania na rzeczywistej platformie sprzętowej.

## 2. Przebieg laboratorium:

- a. Zapoznanie się z dokumentacją, skonfigurowanie i uruchomienie wstępnej implementacji projektu w języku Python.
  - b. Zrealizowanie zadań dla dostarczonego procesora.
  - c. Rozszerzenie procesora o nowe funkcjonalności.
  - d. Utworzenie nowego projektu w programie Vivado służącego do zaprogramowania układu FPGA - ZYBO produkcji Digilent.
  - e. Skonfigurowanie projektu przy pomocy dostarczonych plików źródłowych.
  - f. Nawiązanie połączenia z urządzeniem przy pomocy Putty.
  - g. Realizacja zadań w Verilogu.
  - h. Utworzenie nowej instrukcji dla mikroprocesora.

### 3. Obserwacje działań procesora

Działanie naszego pythonowego procesora mogliśmy zaobserwować na konsoli terminala:

c=3 r0=0 r1=0 pc=0 A=8 R=00 W=00 M: 80 90 EE 86 DD 30 40 EE 00 00 00 00 00 00 00

Kolumny konsoli zostały następująco zdefiniowane:

Symbol	Znaczenie	Opis
c	cykl mikrooperacji (0–3)	numer mikrocyklu wewnętrz jednej instrukcji; np. pobranie, dekodowanie, wykonanie, zapis
r0, r1	rejestry ogólnego przeznaczenia	aktualne wartości rejestrów R0 i R1
pc	program counter	adres aktualnej instrukcji (licznik rozkazów)
A	adres	adres pamięci, z którym aktualnie CPU pracuje
R	odczyt z pamięci (Read)	bajt właśnie pobrany z pamięci
W	zapis do pamięci (Write)	bajt, który CPU chce zapisać (jeśli 00, to nic nie zapisuje)
M:	pamięć programu	fragment pamięci RAM/ROM, pokazujący zawartość (tu: kod 86 96 40 30 00 00 )

#### 4. Realizacja zadań w Pythonie

Podstawowa wersja zadania została zrealizowana poprzez poniższą konfigurację instrukcji:

```
program_load_store_program = [
    0x80,
    0x90,
    0xEE,
    0x86,
    0xDD,
    0x20,
    0xEE,
    0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x05, 0x00, 0x00
]
```

Następnie dodaliśmy dodatkowe funkcje - dodawanie z zapisem w R1 i mnożenie, do 0x4 przepisaliśmy dodawanie z zapisem do R1, a do 0x3 mnożenie R1 x R0. Oczywiście najpierw sprawdziliśmy w dokumentacji dostępność wybranych konfiguracji. Zapis wyniku mnożenia zaimplementowaliśmy następujaco: R1 to 4 najstarsze bity, a R0 to 4 najmłodsze

```
elif self._opcode_h == 0x3:
    result = self.r1 * self.r0
    self.r0 = result & 0xF
    self.r1 = (result >> 4) & 0xF
elif self._opcode_h == 0x4:
    self.r1 = self.r0 + self.r1
```

Aby zaobserwować działania naszych ulepszeń zmieniliśmy konfigurację instrukcji w następujący sposób:

```
test_program = [
    0x80, # R0 = 0
    0x90, # R1 = 0
    0x86, # R0 = 6
    0x96, # R1 = 6
    0x40, # R1 = R1 + R0
    0x30, # MUL R1 * R0
    0x00
]
```

Po uruchomieniu programu i sprawdzeniu konsoli, proces postąpił zgodnie z naszymi instrukcjami i oczekiwaniami:

```
c=2 r0=0 r1=1 pc=2 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=0 r1=0 pc=2 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=0 r1=0 pc=2 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=0 r1=0 pc=2 A=2 R=EE W=02 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=0 r1=0 pc=3 A=2 R=EE W=02 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=0 r1=0 pc=3 A=E R=EE W=00 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=0 r1=0 pc=3 A=E R=EE W=00 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=0 r1=0 pc=3 A=3 R=86 W=00 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=0 r1=0 pc=4 A=3 R=86 W=00 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=6 r1=0 pc=4 A=3 R=86 W=00 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=6 r1=0 pc=4 A=3 R=86 W=00 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=6 r1=0 pc=4 A=4 R=DD W=00 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=6 r1=0 pc=5 A=4 R=DD W=00 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=6 r1=0 pc=5 A=D R=DD W=00 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=6 r1=5 pc=5 A=D R=DD W=00 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=6 r1=5 pc=5 A=5 R=40 W=00 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=6 r1=5 pc=6 A=5 R=40 W=00 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=6 r1=B pc=6 A=5 R=40 W=00 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=6 r1=B pc=6 A=5 R=40 W=00 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=6 r1=B pc=6 A=6 R=30 W=00 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=6 r1=B pc=7 A=6 R=30 W=00 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=2 r1=4 pc=7 A=6 R=30 W=00 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=2 r1=4 pc=7 A=6 R=30 W=00 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=2 r1=4 pc=7 A=7 R=EE W=00 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=2 r1=4 pc=8 A=7 R=EE W=00 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=2 r1=4 pc=8 A=E R=EE W=02 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=2 r1=4 pc=8 A=E R=EE W=02 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=2 r1=4 pc=8 A=8 R=00 W=02 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=2 r1=4 pc=9 A=8 R=00 W=02 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=2 r1=4 pc=0 A=8 R=00 W=02 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=2 r1=4 pc=0 A=8 R=00 W=02 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=2 r1=4 pc=0 A=0 R=80 W=02 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=2 r1=4 pc=1 A=0 R=80 W=02 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=0 r1=4 pc=1 A=0 R=80 W=02 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=0 r1=4 pc=1 A=0 R=80 W=02 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=0 r1=4 pc=1 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=0 r1=4 pc=2 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=0 r1=0 pc=2 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=0 r1=0 pc=2 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
```

## 5. Realizacja zadań w Verilogu

Po wykonaniu zadania w języku Python, przystąpiliśmy do realizacji drugiej części zadania w języku Verilog na płytce Digilent ZYBO. Logika obsługiwania operacji działań w następujący sposób: każde naciśnięcie przycisku BTN2 skutkowało wypisaniem stanu pamięci i rejestrów po kolejnym takcie cyklu zegara.

W celu realizacji zadania(wpisywania wyniku do rejestru R1 i implementacji mnożenia wartości  $R0 * R1$ ) najpierw zbudowaliśmy projekt, potem sprawdziliśmy działania dla podstawowych instrukcji. Rozwiązywanie rozpoczęliśmy od implementacji dodatkowych zachowań procesora:

Dodaliśmy zmienną result, która będzie odpowiadać za tymczasowe przechowywanie wyniku mnożenia.

```
41
42 reg [1:0] state;      //przechowuje aktualny stan mikroprocesora
43 reg [3:0] pc;         //rejestr PC - licznik programu
44 reg [7:0] opcode;    //rejestr do tymczasowego przechowywania kodu instrukcji pobranego z pamięci
45
46 reg [3:0] r0;        //rejestr r0
47 reg [3:0] r1;        //rejestr r1
48 reg[7:0] result;    // zmienna
49
50
51 //przypisania zapewniające podgląd wewnętrznych sygnałów
52 assign dbg_state = state;
53 assign dbg_r0 = r0;
54 assign dbg_r1 = r1;
55
56 assign dbg_pc = pc;
```

Potem dodaliśmy logikę działania sumowania do rejestrów R1 i mnożenia.

```
100 3'b010:           //jesli bity 6:4 maja wartosc 010...
101 begin
102     r0 <= r0 + r1;      //wykonujemy dodawanie: add r0, r1 (r0 = r0 + r1)
103 end
104
105 3'b011:
106 begin
107     r1 <= r0 + r1;      // dodawanie r1 do r0 zapisem w r1
108 end
109
110 3'b100:
111 begin
112     result = r0 * r1;    // mnożenie r1 * r2
113     r0 <= result[3:0];   // zapis rozbitý na r1 i r0
114     r1 <= result[7:4];
115 end
116 endcase
117 end
```

Na koniec uzupełniliśmy wektor z instrukcjami.

Key	Value
memory_initialization_radix	16
memory_initialization_vector	80 90 EE 86 DD 30 40 EE 00 00 00 00 00 05 00 01

Zaobserwowałem działanie programu w konsoli utworzonej połączeniem z Putty. Wszystkie operacje przebiegły zgodnie z naszymi założeniami i implementacją.

```

c=2 r0=0 r1=1 pc=2 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=0 r1=0 pc=2 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=0 r1=0 pc=2 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=0 r1=0 pc=2 A=2 R=EE W=02 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=0 r1=0 pc=3 A=2 R=EE W=02 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=0 r1=0 pc=3 A=E R=EE W=00 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=0 r1=0 pc=3 A=E R=EE W=00 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=0 r1=0 pc=3 A=3 R=86 W=00 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=0 r1=0 pc=4 A=3 R=86 W=00 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=6 r1=0 pc=4 A=3 R=86 W=00 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=6 r1=0 pc=4 A=3 R=86 W=00 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=6 r1=0 pc=4 A=4 R=DD W=00 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=6 r1=0 pc=5 A=4 R=DD W=00 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=6 r1=0 pc=5 A=D R=DD W=00 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=6 r1=5 pc=5 A=D R=DD W=00 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=6 r1=5 pc=5 A=5 R=40 W=00 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=6 r1=5 pc=6 A=5 R=40 W=00 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=6 r1=B pc=6 A=5 R=40 W=00 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=6 r1=B pc=6 A=5 R=40 W=00 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=6 r1=B pc=6 A=6 R=30 W=00 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=6 r1=B pc=7 A=6 R=30 W=00 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=2 r1=4 pc=7 A=6 R=30 W=00 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=2 r1=4 pc=7 A=6 R=30 W=00 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=2 r1=4 pc=7 A=7 R=EE W=00 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=2 r1=4 pc=8 A=7 R=EE W=00 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=2 r1=4 pc=8 A=E R=EE W=02 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=2 r1=4 pc=8 A=E R=EE W=02 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=2 r1=4 pc=8 A=8 R=00 W=02 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=2 r1=4 pc=9 A=8 R=00 W=02 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=2 r1=4 pc=0 A=8 R=00 W=02 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=2 r1=4 pc=0 A=8 R=00 W=02 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=2 r1=4 pc=0 A=0 R=80 W=02 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=2 r1=4 pc=1 A=0 R=80 W=02 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=0 r1=4 pc=1 A=0 R=80 W=02 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=0 r1=4 pc=1 A=0 R=80 W=02 M: 80 90 EE 86 DD 40 30 EE
c=1 r0=0 r1=4 pc=1 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
c=2 r0=0 r1=4 pc=2 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
c=3 r0=0 r1=0 pc=2 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE
c=0 r0=0 r1=0 pc=2 A=1 R=90 W=02 M: 80 90 EE 86 DD 40 30 EE

```

## 6. Podsumowanie i wnioski

W trakcie ćwiczenia poznaliśmy zasadę działania prostego mikroprocesora oraz sposób jego implementacji w układzie FPGA przy użyciu środowiska Vivado. Stworzony i uruchomiony program działał poprawnie, wykonując wszystkie zadane instrukcje.

Dla lepszego zrozumienia działania procesora początkowo wykonaliśmy implementację i symulację w języku **Python**, odwzorowującą jego logikę. Potwierdziło to poprawność programu i ułatwiło analizę przebiegu instrukcji.

Ćwiczenie pozwoliło praktycznie zrozumieć działanie mikroprocesorów oraz możliwości ich modyfikacji w logice programowej.