

Politechnika Krakowska im. Tadeusza Kościuszki

Wydział Inżynierii Elektrycznej i Komputerowej



**Politechnika Krakowska**  
im. Tadeusza Kościuszki



**Wydział Inżynierii**  
**Elektrycznej i Komputerowej**

Projekt i implementacja 16-bitowego CPU na układzie  
FPGA Altera MAX 10

Piotr Sarna

Informatyka w Inżynierii Komputerowej

Prowadzący:

dr inż. Dariusz Dorota

## Spis treści

1. Założenia projektowe	3
2. Jednostka Arytmetyczno-Logiczna ALU	4
3. Plik Rejestrów	6
4. Układ Współpracy z Pamięcią MMU	9
5. Jednostka Sterująca	11
6. RAM	14
7. Integracja struktury procesora	15
8. Prezentacja działania	16
9. Podsumowanie	19

# 1. Założenia projektowe

## Architektura

- Szyna danych 16-bitowa
- Szyna adresowa 32-bitowa
- Zegar procesora clk na Push-button

## Jednostka Arytmetyczno-Logiczna ALU

- Jednostka Arytmetyczno-Logiczna ALU 16-bitowa
- Realizacja 20 operacji ALU

## Plik rejestrów

- 16 rejestrów roboczych R1 – R16

## Wyświetlanie wartości procesora

- Sa – 7 segment
- Sid – 7 segment
- Sbb – 7 segment
- Sbc – 7 Segment
- Sba – 7 Segment
- Yalu – 7 segment
- Salu – diody LED

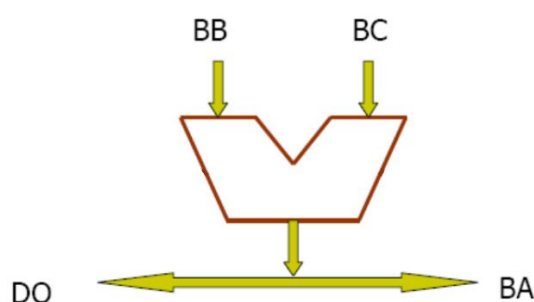
## Kod projektu

<https://github.com/sarnapiotr/CPU16bit-FPGA-Altera-MAX10>

## 2. Jednostka Arytmetyczno-Logiczna ALU

ALU posiada 2 wejścia 16-bitowych argumentów A, B i jedno 16-bitowe wyjście wyniku Y. Na wejście A trafiają dane z szyny danych BB, a na wejście B trafiają dane z szyny danych BC. Wynik Y trafia na szyny danych BA i DO.

Wyniki operacji ustawiają sygnały C (przeniesienie), Z (zero), S (znak).

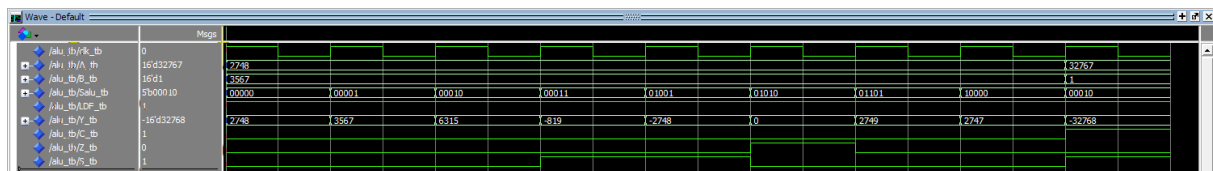


ALU realizuje następujące 20 operacji, w zależności od sygnału sterującego Salu:

- 0  $Y = BB$
- 1  $Y = BC$
- 2  $Y = BB + BC$
- 3  $Y = BB - BC$
- 4  $Y = BB \text{ or } BC$
- 5  $Y = BB \text{ and } BC$
- 6  $Y = BB \text{ xor } BC$
- 7  $Y = BB \text{ xnor } BC$
- 8  $Y = \text{not } BB$
- 9  $Y = -BB$
- 10  $Y = 0$
- 11  $Y = BB + BC + C$
- 12  $Y = BB - BC - C$
- 13  $Y = BB + 1$
- 14  $Y = BB \text{ shl } 1$
- 15  $Y = BB \text{ shr } 1$
- 16  $Y = BB - 1$
- 17  $Y = \min(BB, BC)$
- 18  $Y = BB \text{ nor } BC$
- 19  $Y = BB \text{ nand } BC$

Sygnały C, Z, S wygenerowane przez ALU wracają do Jednostki Sterującej. Jest to kluczowe dla skoków warunkowych. Gdy maszyna stanów napotka rozkaz JZ (Jump if Zero), sprawdza flagę Z. Gdy wynosi ona 1, FSM ładuje nowy adres do licznika programu PC, aby wykonać skok.

## Symulacja operacji arytmetycznych

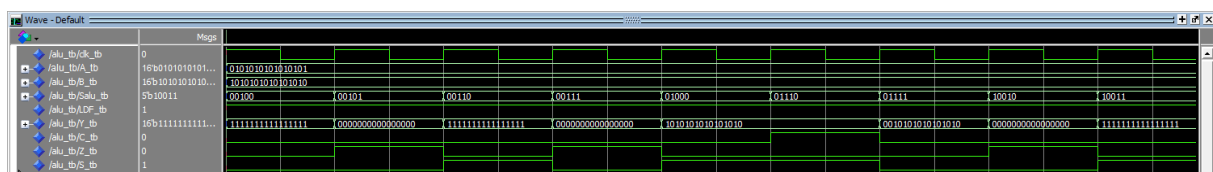


Przy argumentach  $A = 2748$ ,  $B = 3567$  wykonano następujące operacje:

- $Y = BB$ ,  $Y = 2748$
- $Y = BC$ ,  $Y = 3567$
- $Y = BB + BC$ ,  $Y = 6315$
- $Y = BB - BC$ ,  $Y = -819$
- $Y = -BB$ ,  $Y = -2748$
- $Y = 0$ ,  $Y = 0$
- $Y = BB + 1$ ,  $Y = 2749$
- $Y = BB - 1$ ,  $Y = 2747$

Można zauważyć, że sygnały C, Z, S są odpowiednio ustawiane w zależności od otrzymanych wyników.

## Symulacja operacji logicznych



Przy argumentach  $A = 0101010101010101$ ,  $B = 1010101010101010$  wykonano następujące operacje:

- $Y = A \text{ or } B$ ,  $Y = 1111111111111111$
- $Y = A \text{ and } B$ ,  $Y = 0000000000000000$
- $Y = A \text{ xor } B$ ,  $Y = 1111111111111111$
- $Y = A \text{ xnor } B$ ,  $Y = 0000000000000000$
- $Y = \text{not } A$ ,  $Y = 1010101010101010$
- $Y = A \text{ shl } 1$ ,  $Y = 1010101010101010$
- $Y = A \text{ shr } 1$ ,  $Y = 0010101010101010$
- $Y = A \text{ nor } B$ ,  $Y = 0000000000000000$
- $Y = A \text{ nand } B$ ,  $Y = 1111111111111111$

### 3. Plik Rejestrów

Plik rejestrów pełni rolę wewnętrznej pamięci procesora. W mojej realizacji rejestry dzielą się na kilka grup:

- Rejestry robocze (R1 – R16). Przechowują tymczasowe wyniki operacji ALU.
- Rejestry specjalne 16-bitowe:
  - IR (Instruction Register) przechowuje aktualnie wykonywany kod programu.
  - TMP – Tymczasowy rejestr pomocniczy, często używany jako prawy argument operacji ALU.
- Rejestry adresowe 32-bitowe:
  - AD – Rejestr adresowy
  - PC – Licznik programu
  - SP – Wskaźnik stosu
  - ATMP – Rejestr pomocniczy adresowy

Rejestry 32-bitowe są podzielone na dwie 16-bitowe części – wysoką (H) i niską (L), co pozwoli obsłużyć je na 16-bitowych szynach danych.

Układ rejestrów wykorzystuje 3 szyny danych i jedną szynę adresową:

- Szyna BA (16-bitowa). Szyna wejściowa służąca do zapisania wyniku ALU do wybranego rejestru. Wyborem steruje 5-bitowy sygnał Sba.
- Szyny BB i BC (16-bitowe). Szyny wyjściowe, dostarczają operandy do ALU. 5-bitowe sygnały Sbb i Sbc decydują, który rejestr wstawi swoją zawartość na daną szynę.
- Szyna A (32-bitowa). Szyna wyjściowa dla adresów. Prowadzi do rejestru MAR w Układzie Współpracy z Pamięcią MMU. Sygnał Sa wybiera, który z rejestrów 32-bitowych (AD, PC, S.C., ATMP) ma wskazać adres w pamięci.

Rejestr IR jest bezpośrednio połączony do wejścia Jednostki Sterującej. Gdy tylko instrukcja zostanie zapisana w IR, jej zawartość staje się natychmiast widoczna dla Jednostki Sterującej.

Sygnal sterowania Sid pozwala na następujące operacje inkrementacji/dekrementacji 32-bitowych rejestrów adresowych (AD, PC, SP, ATMP):

Sid	Operacja
0	Brak operacji
1	PC++
2	SP++
3	SP--
4	AD++
5	AD--

Sterowanie za pomocą sygnałów Sa, Sbb, Sbc, Sba:

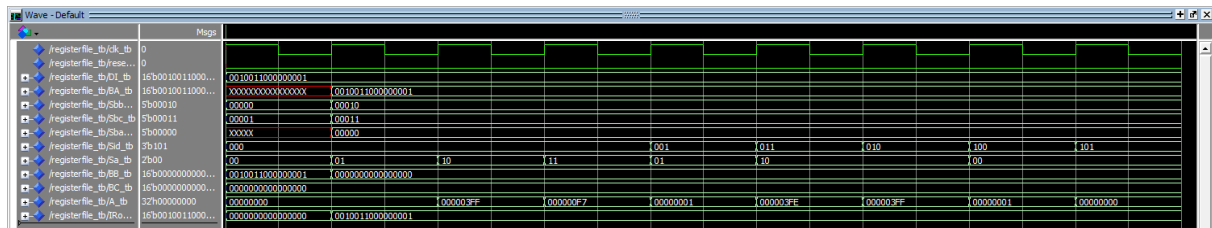
Sa	Operacja
0	A = AD
1	A = PC
2	A = SP
3	A = ATMP

Sbb	Operacja
0	BB = DI
1	BB = TMP
2	BB = R1
3	BB = R2
4	BB = R3
5	BB = R4
6	BB = R5
7	BB = R6
8	BB = R7
9	BB = R8
10	BB = R9
11	BB = R10
12	BB = R11
13	BB = R12
14	BB = R13
15	BB = R14
16	BB = R15
17	BB = R16
18	BB = ADh
19	BB = ADl
20	BB = PCh
21	BB = PCl
22	BB = SPh
23	BB = SPi
24	BB = ATMPPh
25	BB = ATMPI

Sbc	Operacja
0	BC = DI
1	BC = TMP
2	BC = R1
3	BC = R2
4	BC = R3
5	BC = R4
6	BC = R5
7	BC = R6
8	BC = R7
9	BC = R8
10	BC = R9
11	BC = R10
12	BC = R11
13	BC = R12
14	BC = R13
15	BC = R14
16	BC = R15
17	BC = R16
18	BC = ADh
19	BC = ADl
20	BC = PCh
21	BC = PCl
22	BC = SPh
23	BC = SPi
24	BC = ATMPPh
25	BC = ATMPI

Sba	Operacja
0	IR = BA
1	TMP = BA
2	R1 = BA
3	R2 = BA
4	R3 = BA
5	R4 = BA
6	R5 = BA
7	R6 = BA
8	R7 = BA
9	R8 = BA
10	R9 = BA
11	R10 = BA
12	R11 = BA
13	R12 = BA
14	R13 = BA
15	R14 = BA
16	R15 = BA
17	R16 = BA
18	ADh = BA
19	ADl = BA
20	PCh = BA
21	PCl = BA
22	SPh = BA
23	SPi = BA
24	ATMPPh = BA
25	ATMPI = BA

# Symulacja działania Pliku Rejestrów



1. Podając na szynę DI wartość 0010011000000001 i ustawiając Sbb na 00000 oraz Sbc na 00001, na szynie BB otrzymujemy dane z DI a na szynie BC otrzymujemy wartość 0, ponieważ TMP ma wartość 0 po inicjalizacji. Sa ustawione na 00 wyprowadza na szynę A rejestr AD, który również jest wyzerowany.
2. Po ustawieniu Sba na 00000 dane 0010011000000001 z szyny BA zostają zapisane w rejestrze IR, w wyniku czego wyjście IRout prowadzące do Jednostki Sterującej również ma wartość 0010011000000001. Sa wynosi 01, co powoduje wyprowadzenie na szynę A licznik programu PC (obecnie wartość 0).
3. Sa = 10 powoduje wyprowadzenie na A wskaźnik stosu SP, o wartości hex 3FF. Po ustawieniu Sa na 11, na A pojawia się wartość rejestru ATMP o wartości 0x0F7.
4. Sid = 001 powoduje zwiększenie rejestru PC o 1. Możemy zaobserwować nową wartość po wyprowadzeniu jej na szynę A.
5. Sid = 011 powoduje zmniejszenie wskaźnika stosu SP o 1. Wartość SP wyprowadzona na szynę A ma wartość 0x3FE.
6. Sid = 010 powoduje zwiększenie wskaźnika stosu SP o 1. Wartość SP wyprowadzona na szynę A znowu wraca do wartości 0x3FF.
7. Sid = 100 powoduje inkrementację rejestru AD. Zwiększony AD wyprowadzony na A.
8. Sid = 101 powoduje dekrementację rejestru AD. Zmniejszony AD wyprowadzony na A, wartość wraca do 0.

## 4. Układ Współpracy z Pamięcią MMU

Układ współpracy z pamięcią pełni rolę komunikacji pomiędzy procesorem a zewnętrzną pamięcią RAM wykorzystując zatrzaski w rejestrach MAR i MBR. Bez niej procesor nie mógłby współpracować z pamięcią zewnętrzną, bo sygnały z rejestrów wewnętrznych zmieniałyby się zbyt szybko, by RAM mógł na nie zareagować.

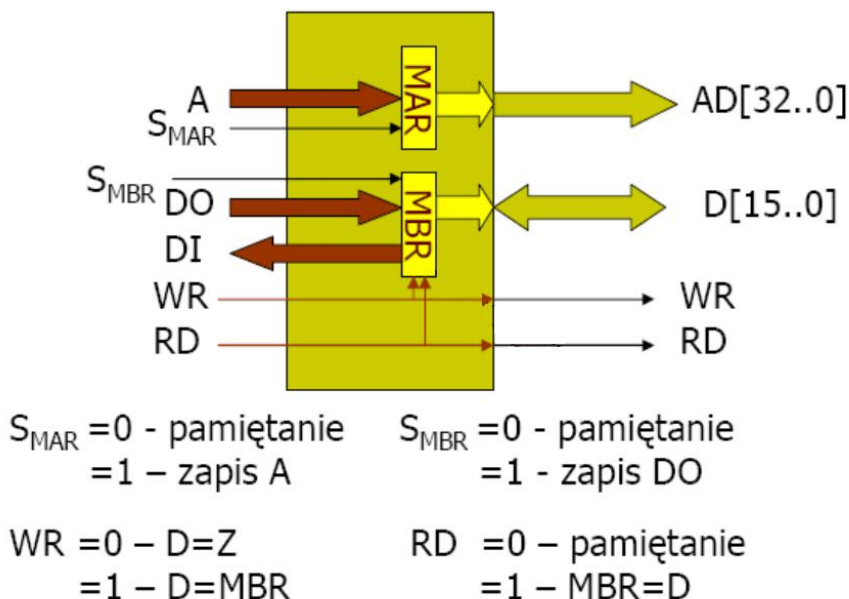
Rejestry MAR i MBR jako zatrzaski:

- MAR (Memory Address Register) – 32 bity. Zatrzaskuje w sobie dane z 32-bitowej szyny A dostarczającej adresy z rejestrów AD, PC, SP i ATMP. Sterowany sygnałem  $S_{MAR}$ . Gdy wynosi on 1, adres z szyny A jest zapisywany w rejestrze. Gdy wynosi 0, rejestr pamięta poprzednią zatrzaśniętą wartość.
- MBR (Memory Buffer Register) – 16 bitów. Bufor dwukierunkowy pośredniczący w wymianie danych:
  - Zapis do RAM z procesora. Gdy sygnał  $S_{mbr}$  wynosi 1, dane z wewnętrznej szyny DO są zatrzaśnięte w rejestrze.
  - Odczyt z RAM do procesora. Przekazuje dane pobierane z RAM na wewnętrzną szynę DI.

Sygnały sterujące WR/RD:

Jednostka sterująca zarządza kierunkiem przepływu informacji za pomocą sygnałów WR i RD:

- WR (Write). Gdy sygnał wynosi 1 wstawia wartość MBR na zewnętrzną szynę danych D. Gdy wynosi 0 pozwala pamięci RAM na wstawianie danych do szyny D.
- RD (Read). Gdy sygnał wynosi 1 pobiera dane z szyny D do MBR. Przy sygnale równym 0 MBR zapamiętuje swoją wartość.



# Symulacja działania Układu Współpracy z Pamięcią



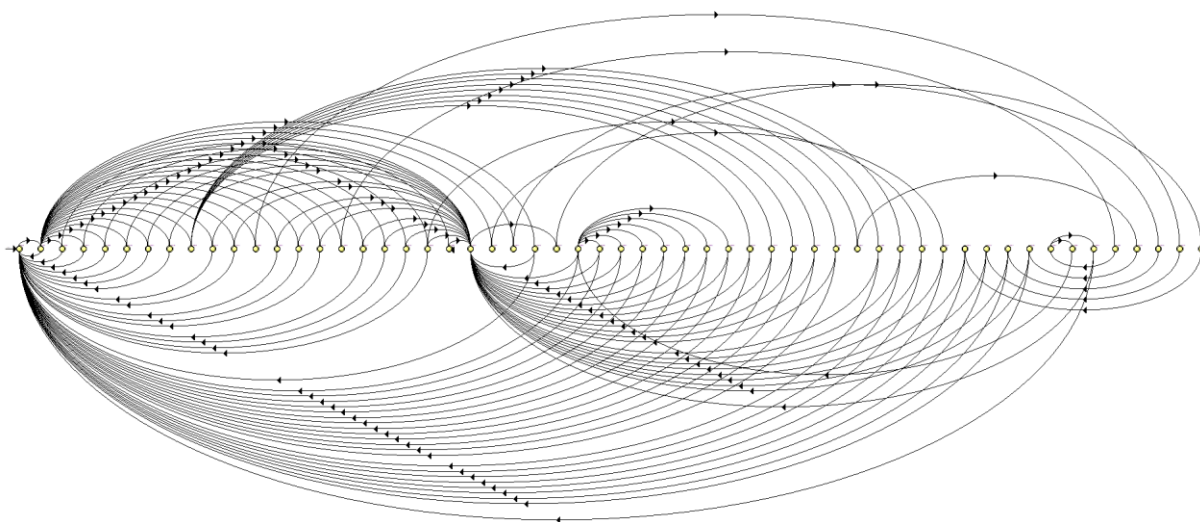
1. Po podaniu adresu 0x01 na szynę A oraz ustawieniu sygnałów Smar i RD na 1, na szynie AD prowadzącej do pamięci RAM otrzymujemy adres 0x01 zatrzaśnięty w rejestrze MAR. Udaje nam się odczytać wartość 0010011000000001, która następnie trafia na szynę DI.
2. Po podaniu adresu 0x3FF na szynę A, wartości 00100110000000010 na szynę DO i ustawieniu sygnałów Smar, Smbr i WR na 1, na szynach AD i D prowadzących do pamięci RAM otrzymano dane uprzednio zatrzaśnięte w rejestrach MAR i MBR.

## 5. Jednostka Sterująca

W mojej implementacji jednostka sterująca to bardzo duża maszyna stanów FSM. Stany od m0 do m92 są ustawiane w zależności od wykonywanych rozkazów. Stany możemy podzielić na następujące grupy:

- m0 i m1 – Fetch, Decode
- m10 – m17 – Grupa 000
- m20 – m36 – Grupa 001
- m40 i m41 – Grupa 010
- m50 – m52 – Grupa 011
- m60 – m70 – Grupa 100
- m80 – m92 – Grupa 101

### Wizualizacja maszyny stanów Jednostki Sterującej



### Analiza rozkazów wykonywanych przez procesor

#### Grupa 000

Rozkazy bezargumentowe – Bity grupy w IR[15..13] = „000”. Kod operacji w IR[12..11]:

- 00 – NOP – Brak operacji, przejście do następnego rozkazu.
- 01 – WAIT – Stan m10, oczekiwanie na przerwanie m11.
- 10 – CALL – Obsługa przerwania, stan m11.
- 11 – RET – Powrót do głównego programu z programu przerwania

## Grupa 001

Rozkazy z argumentem rejestrowym R – Bity grupy w IR[15..13] = „001”. Kod operacji w IR[12..9]. Numer rejestru w IR[4..0].

- 0000 – PUSH      RAM[SP] = R
- 0001 – POP      R = RAM[SP]
- 0010 – NEG      R = -R
- 0011 – INC      R++
- 0100 – DEC      R--
- 0101 – NOT      R = not R
- 0110 – SHR      R = R shr 1
- 0111 – SHL      R = R shl 1
- 1000 – MOV R, M    R = RAM[AD]
- 1001 – MOV M, R    RAM[AD] = R
- 1010 – ADD      R = TMP + R
- 1011 – SUB      R = TMP - R
- 1100 – CMP      CZS(R, TMP)
- 1101 – AND      R = R and TMP
- 1110 – OR      R = R or TMP
- 1111 – XOR      R = R xor TMP

## Grupa 010

Rozkazy z 16-bitową wartością st16 w następnej linijce programu. Bity grupy w IR[15..13] = „010”. Kod operacji w IR[12..11]:

- 00 – JMP – Skacze zawsze. Wartość 16 bit trafia do PCl.
- 01 – JC – Skacze, gdy C = 1
- 10 – JZ – Skacze, gdy Z = 1
- 11 – JS – Skacze, gdy S = 1

## Grupa 011

Rozkaz z 32-bitową wartością st32 w następnych dwóch linijkach programu. Bity grupy w IR[15..13] = „011”. Tylko jeden rozkaz grupy 011:

- JMP – Skacze do nowo podanego 32-bitowego adresu pamięci

## Grupa 100

Rozkazy z argumentem rejestrowym R i 16-bitową wartością st16 w następnej linijce programu. Bity grupy w IR[15..13] = „100”. Kod operacji w IR[12..10]. Numer rejestru w IR[4..0].

- 000 – MOV R, st16            R = st16
- 001 – MOV R, M[st16]       R = RAM[st16]
- 010 – ADD                    R = R + st16
- 011 – SUB                    R = R – st16
- 100 – CMP                   CZS(R, st16)
- 101 – AND                   R = R and st16
- 110 – OR                    R = R or st16
- 111 – XOR                   R = R xor st16

## Grupa 101

Rozkazy z argumentem rejestrowym R i 32-bitowym adresem st32 w następnych dwóch liniijkach programu. Bity grupy w IR[15..13] = „101”. Kod operacji w IR[12..10]. Numer rejestru w IR[4..0].

- 000 – MOV R, M       R = RAM[st32]
- 001 – MOV M, R       RAM[st32] = R
- 010 – ADD            R = R + RAM[st32]
- 011 – SUB            R = R – RAM[st32]
- 100 – CMP            CZS(R, RAM[st32])
- 101 – AND            R = R and RAM[st32]
- 110 – OR             R = R or RAM[st32]
- 111 – XOR            R = R xor RAM[st32]

## 6. RAM

Pamięć RAM w mojej implementacji składa się z 1024 słów, z których każde ma szerokość 16 bitów. Całkowity rozmiar pamięci to zatem 2 KB.

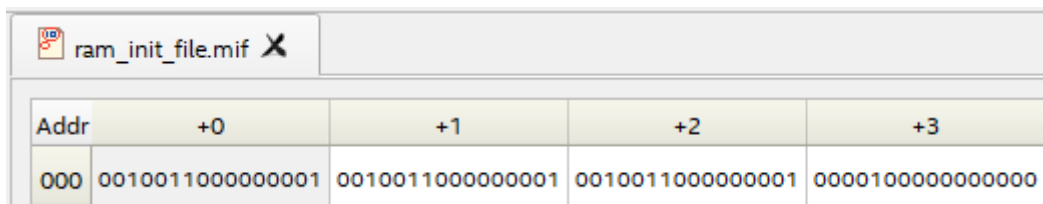
W kodzie VHDL pamięć zadeklarowano jako typ tablicowy. Zawartość pamięci RAM jest inicjalizowana z wykorzystaniem pliku .mif, zawierającym dane dla procesora takie jak np. kod programu.

Dane w RAM umieszczane są w trzech obszarach:

- Sekcja kodu (0x000 – 0x0FF). Przy starcie i resecie procesor zaczyna czytywać rozkazy od adresu 0x000.
- Sekcja danych (0x100 – 0x1FF). Zmienne statyczne adresowane przez rejestr AD.
- Sekcja stosu (0x300 – 0x3FF). Stos rosnący w dół. SP startuje od 0x3FF. Każda operacja PUSH zapisuje dane i dekrementuje SP--.

### Zawartość pliku .mif

Pierwsze cztery komórki pliku inicjalizującego pamięć RAM `ram_init_file.mif` wyglądają następująco:

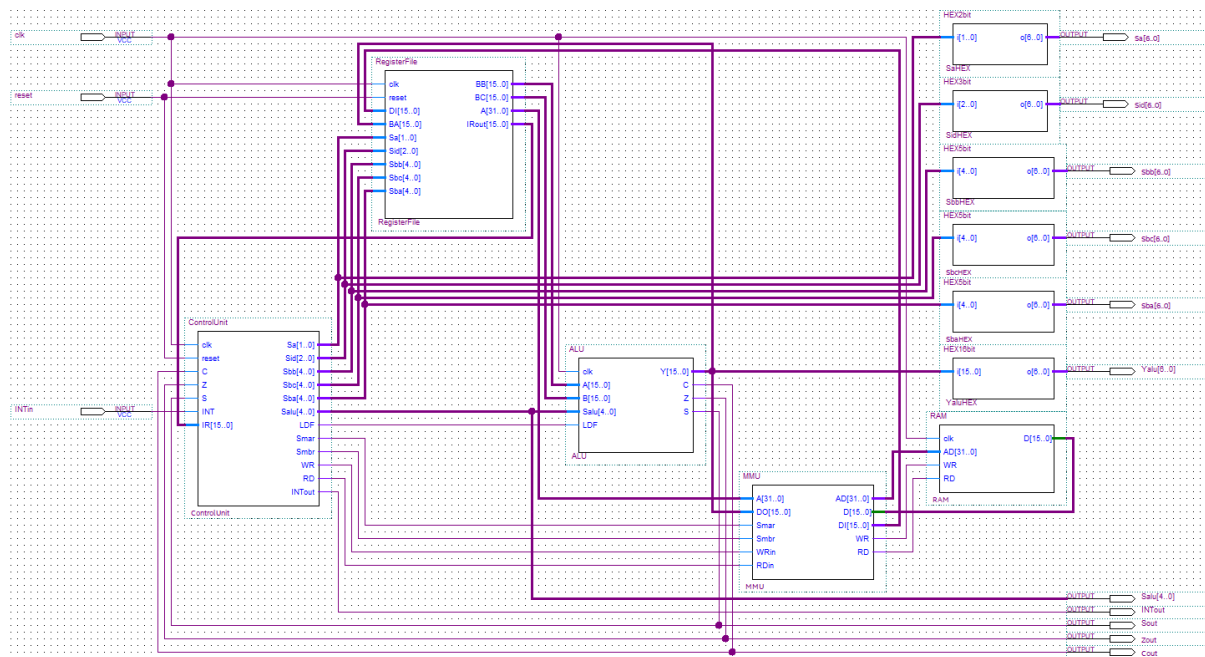


Addr	+0	+1	+2	+3
000	0010011000000001	0010011000000001	0010011000000001	0000100000000000

Zawierają one krótki program testowy, który zostanie przedstawiony w dalszej części.

## 7. Integracja struktury procesora

Wszystkie przedstawione uprzednio komponenty zostały złączone ze sobą z wykorzystaniem pliku .bdf. Utworzony w ten sposób procesor prezentuje się następująco:



Do procesora wchodzi trzy wejścia:

- clk – wejście zegarowe
- reset – wejście resetujące
- INTin – wejście przerwania

Zgodnie z założeniami projektowymi, sygnały sterujące Sa, Sid, Sbb, Sbc, Sba oraz wyjście Y ALU zostały przedstawione na wyświetlaczach 7-segmentowych, dzięki wykorzystaniu odpowiednich dekoderek.

Dodatkowo sygnał sterujący Salu oraz sygnały C, Z, S, INTout zostały również wyprowadzone na wyjścia.

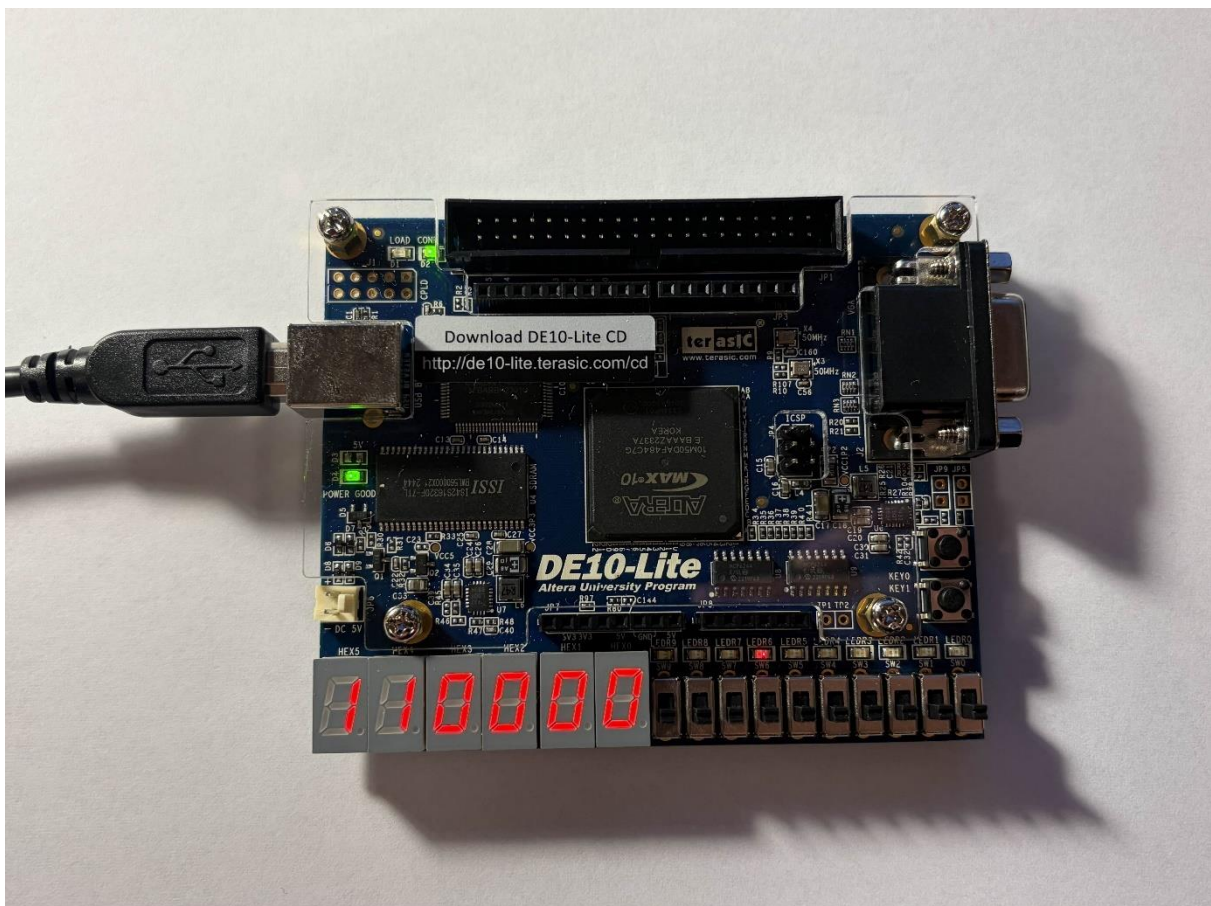
## 8. Prezentacja działania

Zgodnie z przedstawioną uprzednio zawartością pliku inicjalizującego pamięć, przykładowy zamieszczony w nim program zawiera następujące rozkazy:

- 0010011000000001 – INC TMP
- 0010011000000001 – INC TMP
- 0010011000000001 – INC TMP
- 0000100000000000 – WAIT

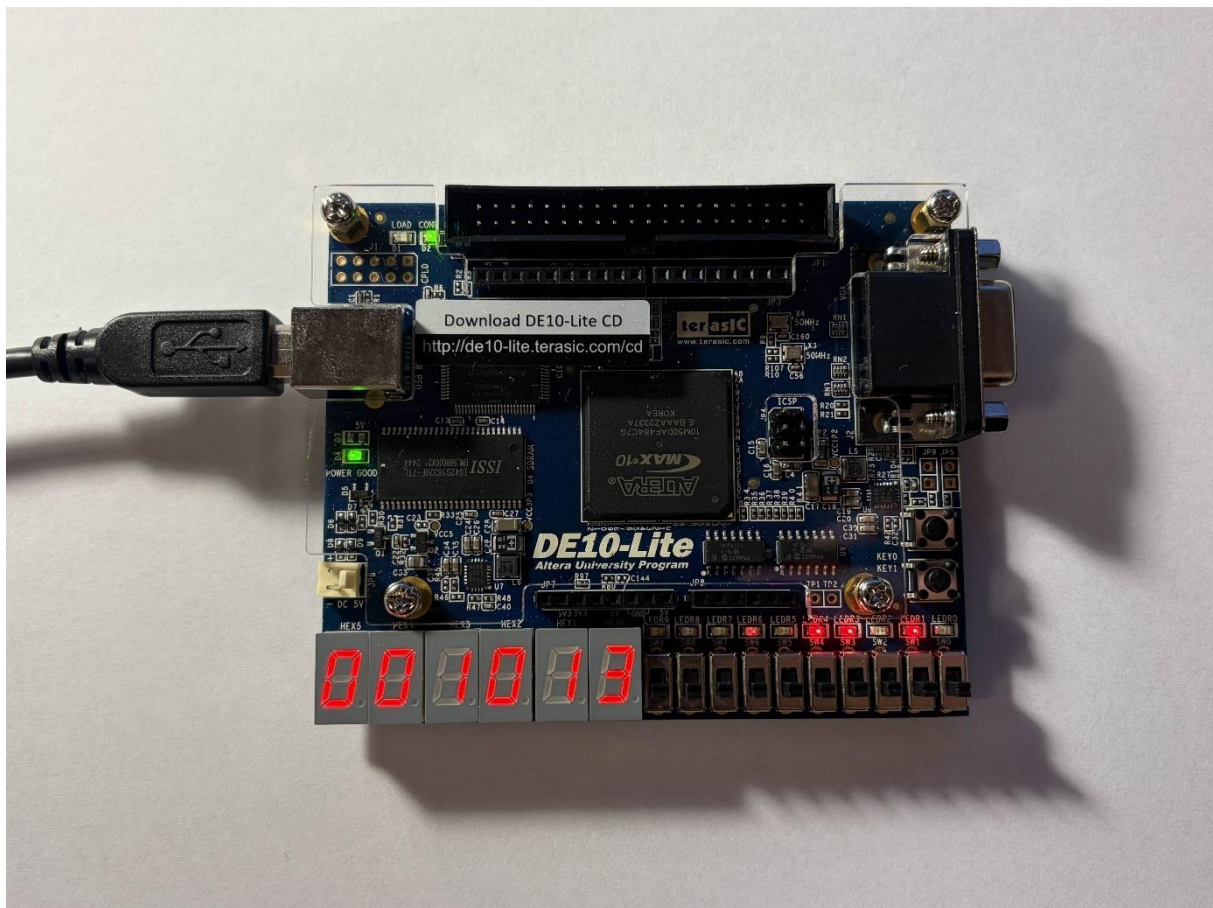
### Stan początkowy

- $S_a = 1$  – Na szynę A trafia adres PC, w celu pobrania nowego rozkazu programu
- $S_{id} = 1$  – Adres PC zostaje inkrementowany, przygotowanie przed następnym cyklem pobrania rozkazu

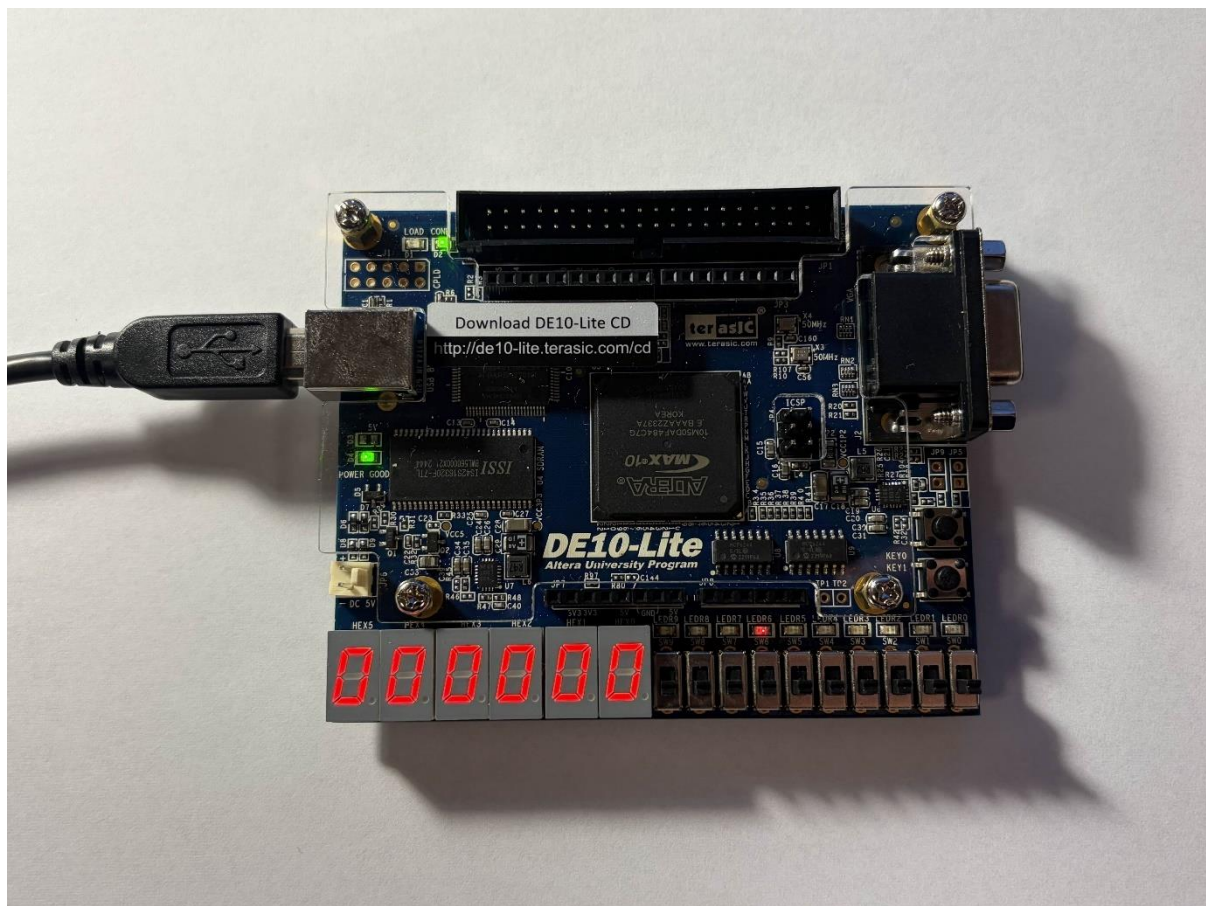


## Stan po wykonaniu trzech rozkazów programu

- $Sa = 0$  – Nie ma potrzeby, by na szynę A trafiał adres PC
- $Sid = 0$  – Brak działania
- $Sbb = 1$  – Wskazanie na rejestr TMP, z którego została pobrana wartość do inkrementacji
- $Sba = 1$  – Wskazanie na rejestr TMP, na który trafia zwiększona wartość po wykonaniu inkrementacji.
- $Yalu = 3$  – Po wykonaniu trzech inkrementacji rejestru TMP, początkowo zawierającego wartość 0, wyjście ALU wynosi 3
- $Salu = 01101$  – Wartość sygnału odpowiadająca operacji inkrementacji



Stan po wejściu w rozkaz WAIT



Wszystkie sygnały sterujące są wyzerowane, procesor oczekuje na sygnał przerwania INTin.

## 9. Podsumowanie

Na podstawie przeprowadzonych symulacji i testu na układzie FPGA można stwierdzić, że procesor poprawnie realizuje założony algorytm. Analiza poszczególnych modułów potwierdza ich poprawne działanie:

- Jednostka Sterująca – Moduł wykazał poprawną implementację maszyny stanów, przechodząc między fazami pobrania (Fetch), dekodowania (Decode) i wykonania instrukcji INC i WAIT. Sygnały sterujące były aktywowane zgodnie z rozkazami programu.
- Pamięć RAM i MMU – Poprawny odczyt z pliku inicjalizacyjnego .mif potwierdza prawidłową współpracę procesora z pamięcią RAM, ponieważ poprawnie zostały rozczytane i wykonane zapisane w niej rozkazy.
- Jednostka Arytmetyczno-Logiczna ALU – Poprawnie obliczyła wynik programu, co potwierdza wyprowadzony na wyświetlacz 7-segmentowy wynik oraz kod operacji Salu.
- Plik Rejestrów – Testy wykazały poprawność zapisu i odczytu danych do rejestru TMP oraz prawidłową pracę inkrementacji adresu PC. Mechanizm sterowania rejestrami przez sygnały Sba, Sbb i Sbc działa zgodnie z założeniami architektury.