Organizacja i Architektura Komputerów

Sprawozdanie z projektu
Projekt i implementacja procesora 8051

Autorzy:
Adam Szcześniak 241293
Marcin Czepiela 241305

Prowadzący:
dr inż. Tadeusz Tomczak

Adam Szcześniak 241293
Marcin Czepiela 241305

Spis treści

Adam Szcześniak 241293
Marcin Czepiela 241305

1. Wstęp: cele i założenia

Celem projektu było zaprojektowanie własnej wersji procesora 8051 firmy Intel oraz implementacja go w wybranym przez nas programie symulującym działanie układów scalonych.

Przed przystąpieniem do implementacji uzgodniliśmy z prowadzącym, że nie będzie to dokładna kopia procesora Intel 8051. (Dokładne różnice pomiędzy projektami wypisane są w dalszej części sprawozdania) Chcieliśmy aby nasza wersja tego procesora byłą w stanie wykonywać przynajmniej po jednej instrukcji każdego typu, ich szczegółowa lista znajduje się w „*Tabela 1*”.

2. Co udało się zrobić

Udało się stworzyć projekt procesora który implementuje wszystkie założone funkcjonalności ustalone w trakcie zajęć z projektu. Ustaliliśmy że stworzymy 16 instrukcji, które będą przykładami instrukcji z każdego typu instrukcji dostępnych w oryginalnym Intelu 8051. Aby wszystkie instrukcje mogły działać prawidłowo, musieliśmy stworzyć wszystkie wymagane moduły procesora które kontrolują cały proces jakim jest działanie procesora np. rejestry, szyny danych, jednostka arytmetyczno logiczna.

Aby przybliżyć złożoność całego układy zamieszczamy listę wszystkich komponentów procesora łącznie mniejszymi układami wchodzącymi w ich skład. Do łączenia ich skorzystaliśmy z szyn danych.

SZYNY DANYCH:

Do kontrolowania pracy poszczególnych układów wykorzystujemy szynę o nazwie **CONTROL(20 bitowa).** Ścieżki o numerach od 0 do 5 odpowiadają za aktywację zapisu w rejestrze lub układzie którego adres jest podawany na szynę. Ścieżki od 6 do 11 odpowiadają za aktywowanie odczytu z rejestrów lub urządzeń o adresie odpowiadającym temu podanemu na szynie. Ścieżki 17 i 18 służą do przekazywania informacji o flagach Overflow i Carry (jest to odstępstwo od oryginalnego projektu 8051 wymuszone optymalizacją) Ścieżka 20 jest podłączona bezpośrednio do zegara. Pozostałe ścieżki przewidziane są dla dalszego rozwoju projektu.

Do przesyłania danych pomiędzy rejestrami i układami wykorzystujemy **DATA (8 bitowa)**, jej stan nie musi zmieniać się przy każdej mikroinstrukcji ponieważ nie wszystkie moduły są z nią bezpośrednio połączone.

Szyna **ORDER_BUS(8 bitowa)** odpowiada za przesyłanie numeru aktualnie wykonywanej instrukcji.

Kolejne szyny danych to **8 bitowe T1ALU** oraz **T2ALU**. Służą do natychmiastowego przesyłania danych z buforów jednostki arytmetyczno-logicznej (TEMP1 oraz TEMP2) do tejże jednostki.

Szyna **TEMP_IN_1 (8 bitów)** służy do bezpośredniego połączenia akumulatora (ACC) z buforem jednostki arytmetyczno logicznej w celu ominięcia szyny DATA i lepszej optymalizacji pracy procesora. (W obecnej wersji projektu nie jest używana poprawnie z powodu znacznego utrudnienia implementacji).

W projekcie występuje szyna **MEM_CTRL (8 bitów)**, która nie jest obecnie używana.

Szyna **STC_ADDR (8 bitów)** pozwala na uniknięcie kolizji i kolejki (które miały by miejsce gdyby użyta została szyna DATA) przy przesyłaniu adresu do pamięci RAM z rejestru wskaźnika stosu STCK_PTR co znacznie przyśpiesza operacje na stosie.

Wyszczególnić można jeszcze szyny **MEM_ADDR (8 bitów)** oraz **RAM_ADDR(8 bitów)** pozwalają one na bezkolizyjne porozumiewanie się sterowników odpowiednio pamięci ROM oraz RAM z tymi pamięciami.

Ostatnią szyną łączącą oddzielne układy wewnętrzne procesora jest szyna **PSW(11 bitów)** łącząca ALU z rejestrem statusu procesora PSW. 8 bitów (bity 0-7) jest używanych na przesyłanie stanów flag 8 bit służy jako aktywator zapisu w rejestrze PSW a 2 ostatnie bity- 9, 10 są obecnie nie używane.

Lista komponentów procesora:

- ALU:
  - SUMATOR
  - DECREMENTER
  - COMPARATOR
  - OR
  - ALU_CTRL
- A_REG (ACC)
- B_REG
- TEMP1
- TEMP2
- PSW
- ROM
- ROM_ADDR_REG
  - RAM_ADDR_REG
  - RAM_ADDR_CTRL
    - INCREMENTER
    - DATA_ADDER
    - SAVE_DATA
- OPERAND
- PROGCOUNTER
- PDTR
  - DPTR.H
  - DPTR.L
- CONTROL_REG
  - ORDER_REG
  - INST_SET
- PORT2_DRIVER
- RAM
- STACK_PTR
  - PUSH
  - REG
- RAM
- RAM_ADDR_REG
  - ROM_ADDR_REG
  - ROM_ADDR_CTRL
    - INCREMENTER
    - DATA_ADDER
  - SAVE_DATA
  - POP

3. Sposób działania z parametrami

Tryby adresowania:

- Rejestrowy (bezpośredni)
  Argumenty i wyniki są w rejestrach procesora, np. SWP A B
- Bezpośredni
  Adres po rozkazie informuje gdzie jest argument, np. LJMP
- Rejestrowy (pośredni)
  Zawartość rejestru adresuje pamięć, np. STACK (rejestr)
- Dane do rozkazu mogą być przekazywane bezpośrednio w kodzie rozkazu.
  np. AJMP XXX0 0001.

4. Instrukcje i mikroinstrukcje

W procesorze zaimplementowaliśmy następujące instrukcje:

NUMBER	(HEX)	NAME	DEST	SRC	Uwagi
0000 0000	00	NOP			
XXX0 0001	01	AJMP	@IN_DIRECT		PRZEKAZANIE ARG W ROZKAZIE
0000 0010	02	LJMP	@DIRECT		PRZEKAZANIE ARGUMENTU Z PAMIĘCI
0000 0011	03	MOV	B	A	
0000 0100	04	ADD	A	B	
0000 0101	05	ADC	A	B	
0000 0110	06	SET_C	CARRY		
0000 0111	07	CLR_C	CARRY		
0000 1000	08	MOV	A	@DIRECT	
0000 1001	09	PUSH	A		
0000 1010	0A	POP	A		
0000 1011	0B	JO		@DIRECT	
0000 1100	0C	ORL	A	B	
0000 1101	0D	DEC	A		ARG W REJESTRZE
0000 1110	0E	SWP	A	B	
0000 1111	0F	INIT	PROCESSOR		Instrukcja służy do ustalenia położenia stosu i kontrolnym uruchomieniu portów procesora

Tabela 1 Lista instrukcji zaimplementowanych w procesorze

Tabela 1 przedstawia kolejno w kolumnie „NUMBER” – wybranych przez nas numer rozkazu wybranej instrukcji, w kolumnie „(HEX)” – reprezentacja numeru w systemie 16, w kolumnie „NAME” – nazwa instrukcji, w kolumnie „DEST” – Źródłowo-wynikowy argument instrukcji , w kolumnie „SCR” Źródłowy argument instrukcji , instrukcji oraz w kolumnie „Uwagi” – dodatkowe informacje związane z konkretną instrukcją.

Nazwa	*ODCZYT*	*ZAPIS*
A_REG	000010	000001
B_REG	000100	000011
TEMP1	--------	000101
TEMP2	--------	000111
STACK	001010	001001
PSW	001100	-------
ALU_SUM	001101	--------
ALU_DECREMENT	001110	--------
ALU_CMP	001111	--------
ALU_OR	010000	--------
ADDR_REG_FROM_DATA_ROM	--------	010001
PROGRAM_COUNTER	010100	010011
ROM	010101	--------
ORDER_REG	--------	010110
INCR_ADDR_REG_ROM	--------	010111
ADD_DATA_ADDR_REG_ROM		011000
RAM	011001	011010
SET_CARRY	--------	011011
CLR_CARRY	--------	011100
ADD_CARRY	011101	--------
OPERAND	011110	011111
RAM_ADDR_FROM DATA	---------	100000
INCR_RAM_ADDR	---------	100001
ADD_DATA_TO ROM_ADDR	---------	100010
STACK_PUSH	---------	100011
STACK_POP	---------	100100
RAM_STACK	100101	100110
DPTR.H	100111	101000
DPTR.L	101001	101010
PORT2_SEND_ADDR_H	--------	101011
PORT2_SEND_ADDR_L	--------	101100
PORT2_SEND_DATA	--------	101101
PORT2_GET_DATA	101110	--------

Tabela 2 Lista adresów funkcji zapisu i odczytu w zaimplementowanych układach

W *Tabela 2* widoczna jest lista wszystkich zaimplementowanych układów wraz z kodami które występują na szynie **CONTROL_BUS** w trakcie odczytu lub zapisu z/do układu. Część związana z odczytem wstępuje na bitach od 6 do 11. Część związana z zapisem występuje na bitach od 0 do 5.

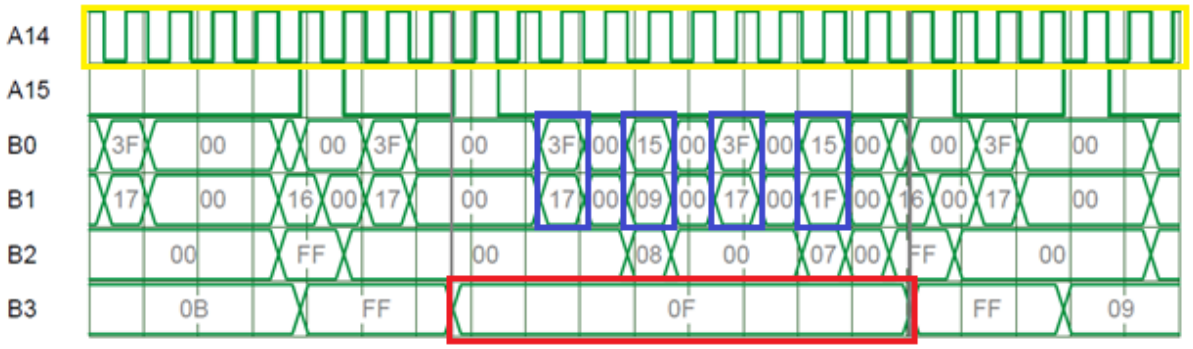
Adam Szcześniak 241293
Marcin Czepiela 241305

5.Technika pomiarów czasu wykonywania instrukcji

W trakcie badania własności naszego procesora posługiwaliśmy się wykresami czasowymi generowanymi przez program Proteus 8 w trakcie symulacji.

Legenda do wykresów czasowych:

A14 – zegar
A15 – sygnał zerujący licznik mikrorozkazów
B0 – CONTROL_BUS (odczyt)
B1 – CONTROL_BUS (zapis)
B2 – DATA_BUS
B3 – ORDER_BUS



Rysunek 1 Przykładowy wykres czasowy

Na „*Rysunek 1*” znajduje się przykładowy wykresu czasowego widzimy wykres czasowy, w tym przypadku jest to instrukcja **INIT**. Kolorem żółtym zaznaczony jest wiersz na którym widoczne są wszystkie cykle zegara. Na niebiesko zaznaczone są informacje o stanie szyn: kontroli, danych, numeru rozkazu kolejno CONTROL_BUS[6..11] oraz CONTROL_BUS[0..5]. Kolorem czerwonym oznaczona jest aktualnie wykonywana instrukcja. Na podstawie wykresu możemy ustalić że wykonywane są 3 mikroinstrukcje, których łączny czas wykonywania zajmuje trzy cykle zegara.

Aby rozpocząć symulację należy wejść do układu **CONTROL_REG** i włączyć symulację. W trakcie korzystania z programu w celu wygenerowania wykresów czasowych nie wolno zamykać okien z pomiarami ponieważ później nie ma możliwości ich ponownego uruchomienia. Zamknął się one automatycznie po zatrzymaniu symulacji.

Adam Szcześniak 241293
Marcin Czepiela 241305

6.Wykresy czasowe zaimplementowanych rozkazów

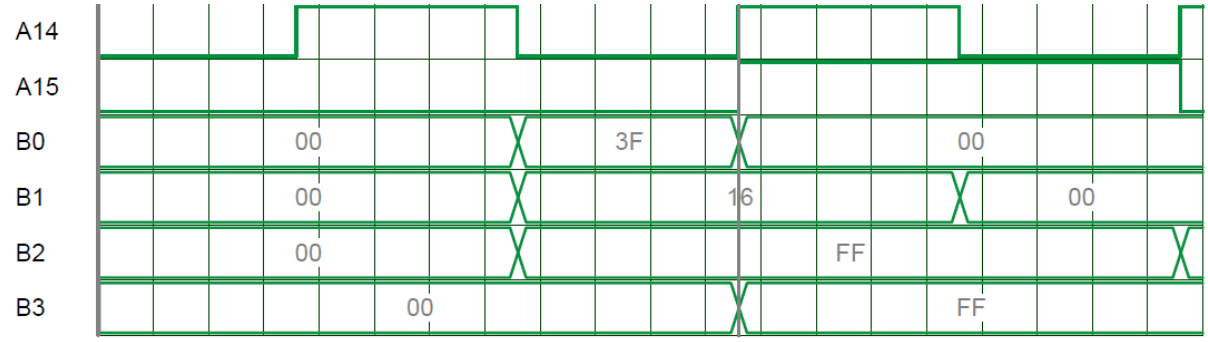
Wykresy czasowe wraz z krótkim opisem:

Legenda do wykresów czasowych:

A14 – zegar	**A15** – sygnał zerujący licznik mikrorozkazów
B0 – CONTROL_BUS (odczyt)	**B1** – CONTROL_BUS (zapis)
B2 – DATA_BUS	**B3** – ORDER_BUS

1) Instrukcja NOP (0000 00002 = 0016):

Instrukcja **NOP** odczekuje **1** cykl procesora.

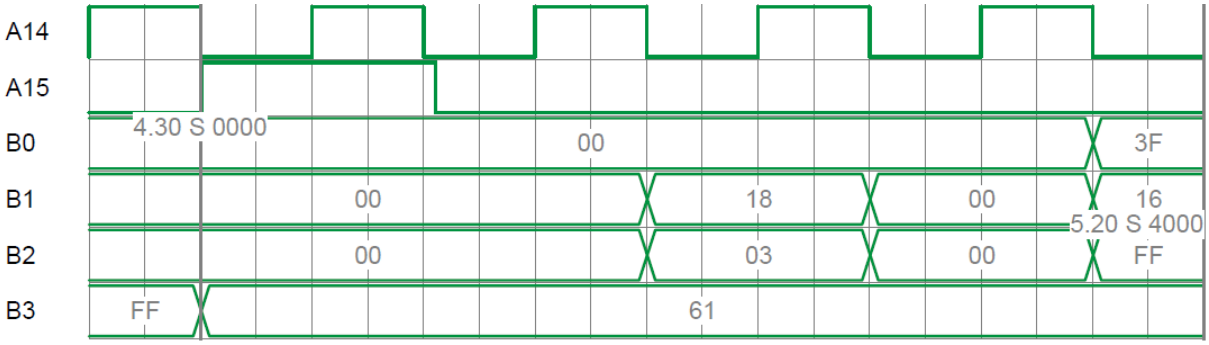


Rysunek 2 Wykres czasowy przebiegu instrukcji NOP

2) Instrukcja AJMP @IN_DIRECT (XXX0 00012 = 0116):

Instrukcja **AJMP** składa się z **2** mikroinstrukcji. Należy zaznaczyć że nie przekazuje ona dane, mianowicie XXX w kodzie to też dane.

- Ustawia 3 najmłodsze bity **DATA_BUS** na kod XXX i dodaje **DATA_BUS** do **ROM_ADDR_REG.**
  CONTROL_BUS[6..11] = ------, CONTROL_BUS[0..5] =011000.

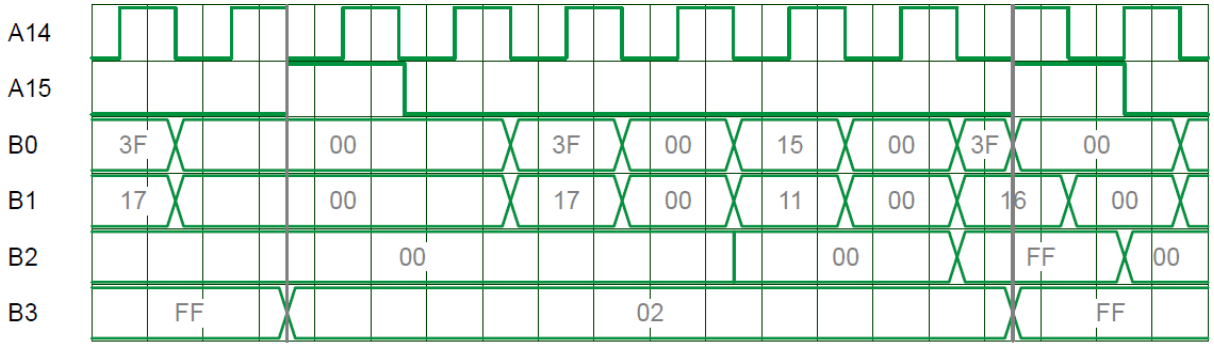


Rysunek 3 Wykres czasowy przebiegu instrukcji AJMP

3) Instrukcja LJMP @Direct (0000 00102 = 0216):

Instrukcja **LJMP** składa się z **2** mikroinstrukcji.

- Wykonuje mikroinstrukcję **INCR_ADDR_REG_ROM**.
  CONTROL_BUS[6..11] = 111111, CONTROL_BUS[0..5] =010111.
- Kopiuje dane z pamięci **ROM** do rejestru **ADDR_REG_FROM_DATA_ROM.**
  CONTROL_BUS[6..11] = 010101, CONTROL_BUS[0..5] =010001.

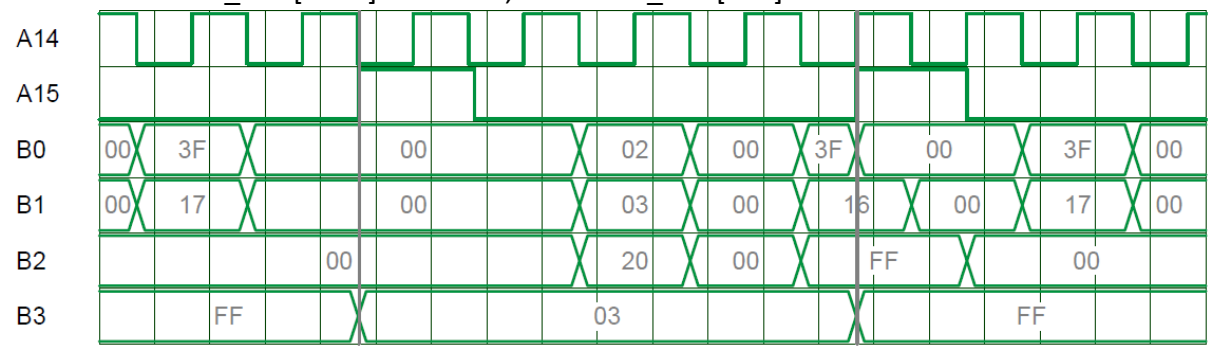


Rysunek 4 Wykres czasowy przebiegu instrukcji LJMP

4) Instrukcja MOV A B (0000 00112 = 032):

Instrukcja **MOV A B** składa się z **1** mikroinstrukcji.

- Kopiuje dane z rejestru **REG_A** do rejestru **REG_B.**
  CONTROL_BUS[6..11] = 000010, CONTROL_BUS[0..5] = 000011.

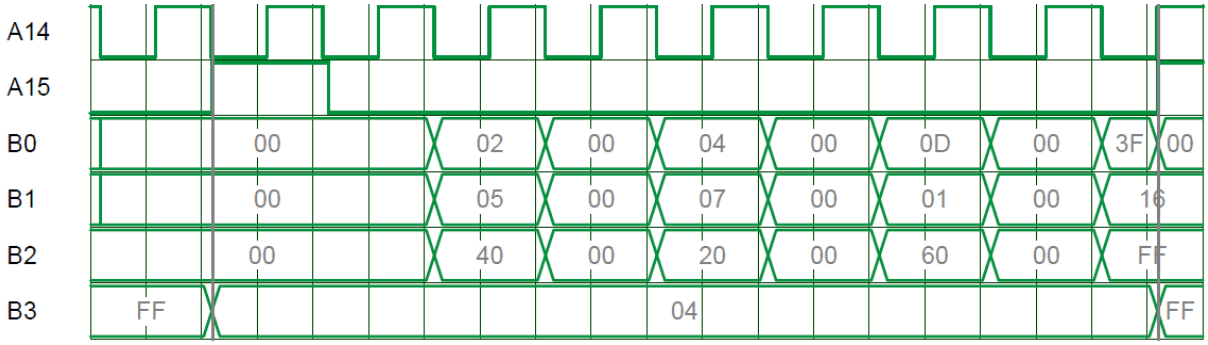


Rysunek 5 Wykres czasowy przebiegu instrukcji MOV A B

5) Instrukcja ADD A B (0000 01002 = 042):

Instrukcja **ADD** składa się z **3** mikroinstrukcji.

- Kopiuje zawartość rejestru **A_REG** do rejestru **Temp1.**
  CONTROL_BUS[6..11] = 000010, CONTROL_BUS[0..5] = 000101.
- Kopiuje zawartość rejestru **B_REG** do rejestru **Temp2.**
  CONTROL_BUS[6..11] = 000100, CONTROL_BUS[0..5] = 000111.
- Odczytuje wynik dodawania z **ALU** i umieszcza go w **A_REG.**
  CONTROL_BUS[6..11] = 001101, CONTROL_BUS[0..5] = 000001.

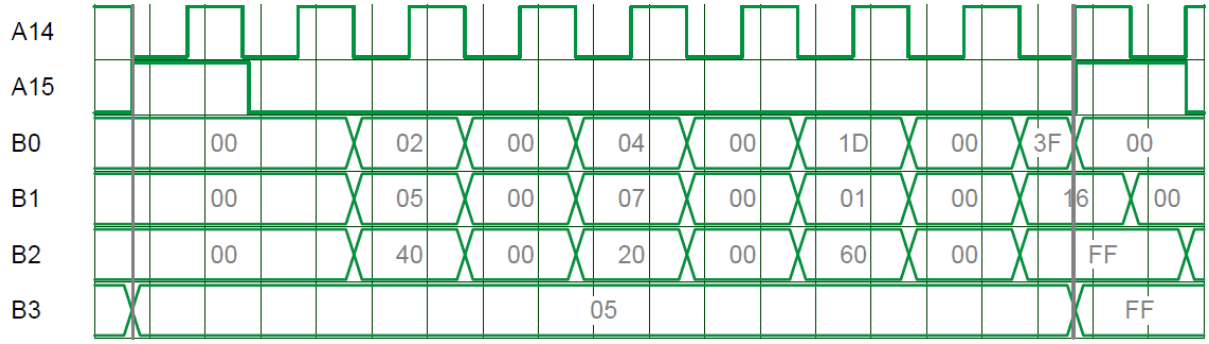


Rysunek 6 Wykres czasowy przebiegu instrukcji ADD A B

6) Instrukcja ADC A B (0000 01012 = 0516):

Instrukcja **ADC** składa się z **3** mikroinstrukcji.

- Kopiuje zawartość rejestru **B_REG** do rejestru **Temp1.**
  CONTROL_BUS[6..11] = 000100, CONTROL_BUS[0..5] = 000101.
- Kopiuje zawartość rejestru **A_REG** do rejestru **Temp2**.
  CONTROL_BUS[6..11] = 000010, CONTROL_BUS[0..5] = 000111.
- Odczyt wynik dodawania z przeniesieniem z **ALU** i zapisuje go w **A_REG**
  CONTROL_BUS[6..11] = 111111, CONTROL_BUS[0..5] = 000001.

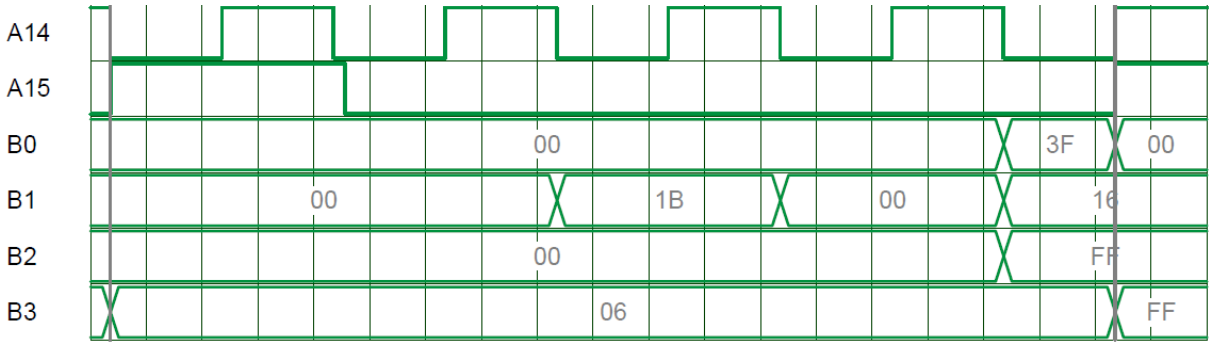


Rysunek 7 Wykres czasowy przebiegu instrukcji ADC A B

7) Instrukcja SET_C (0000 01102 = 0616):

Instrukcja **SET_C** składa się z **1** mikroinstrukcji.

- Wykonuje mikroinstrukcję **SET_CARRY.**
  CONTROL_BUS[6..11] = 000000, CONTROL_BUS[0..5] = 011011.

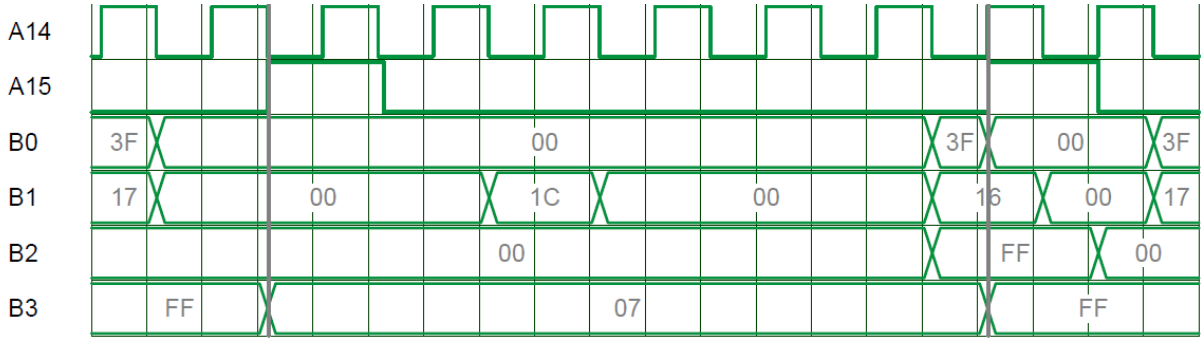


Rysunek 8 Wykres czasowy przebiegu instrukcji SET_C

8) Instrukcja CLR_C (0000 01112 = 0716):

Instrukcja **CLR_C** składa się z **2** mikroinstrukcji.

- Wywołuje mikroinstrukcję **CLR_CARRY.**
  CONTROL_BUS[6..11] = 000000, CONTROL_BUS[0..5] = 011010.
- Wywołuje operację **CLR_OVERFLOW.**
  CONTROL_BUS[6..11] = 000000, CONTROL_BUS[0..5] = 101111.

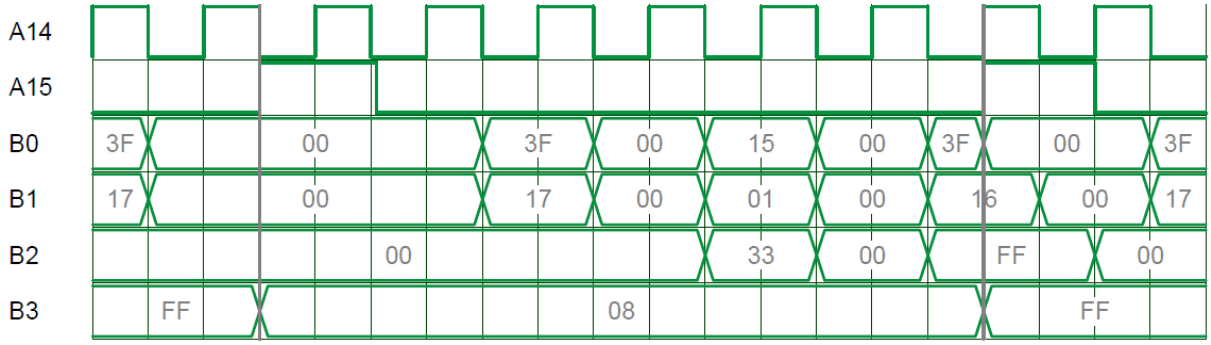


Rysunek 9 Wykres czasowy przebiegu instrukcji CLR_C

9) Instrukcja MOV A @Direct (0000 10002 = 0816):

Instrukcja **MOV** składa się z **2** mikroinstrukcji.

- Inkrementuje rejestr **ROM_ADDR_REG.**
  CONTROL_BUS[6..11] = 111111, CONTROL_BUS[0..5] = 010111.
- Kopiuje dane z **ROM** do rejestru **A_REG**.
  CONTROL_BUS[6..11] = 010101, CONTROL_BUS[0..5] = 000001.

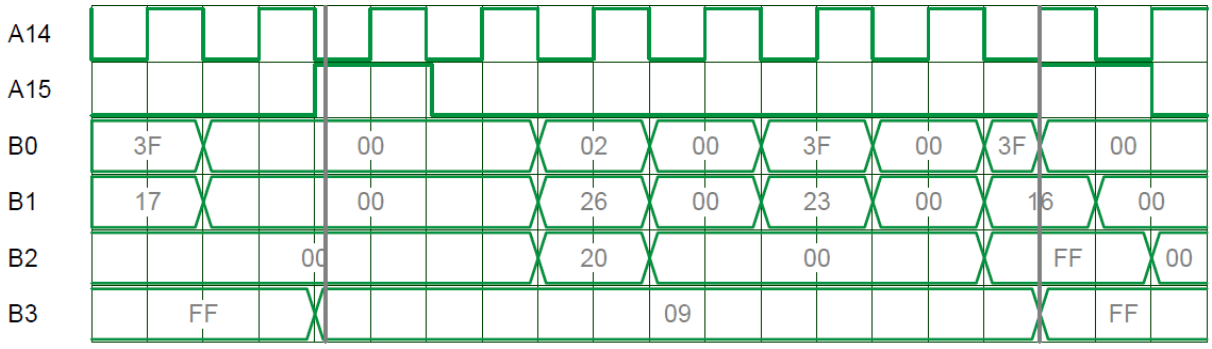


Rysunek 10 Wykres czasowy przebiegu instrukcji MOV A @Direct

10) Instrukcja PUSH A (0000 10012 = 0916):

Instrukcja **PUSH** składa się z **2** mikroinstrukcji.

- Kopiuje zawartość **A_REG** na stos wykonując mikroinstrukcję **RAM_STACK** (zapis)**.**
  CONTROL_BUS[6..11] = 000010, CONTROL_BUS[0..5] = 100110.
- Wykonuje mikrooperację **STACK_PUSH.**
  CONTROL_BUS[6..11] = 111111, CONTROL_BUS[0..5] = 100011.

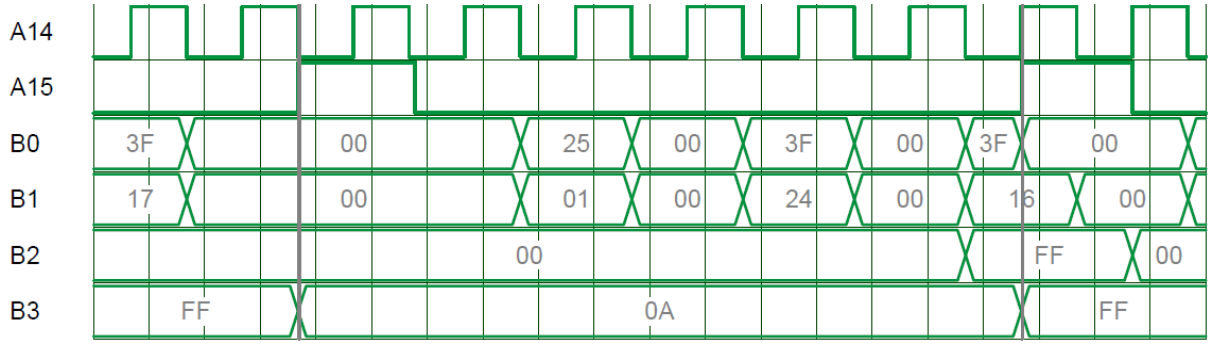


Rysunek 11 Wykres czasowy przebiegu instrukcji PUSH

11) Instrukcja POP A (0000 10102 = 0A16):

Instrukcja **POP** składa się z **2** mikroinstrukcji.

- Odczytuje dane ze stosu wykonujący mikroinstrukcję **RAM_STACK** (odczyt) zapisuje je w rejestrze **A_REG.**
  CONTROL_BUS[6..11] = 100101, CONTROL_BUS[0..5] = 000001.
- Wykonuje mikroinstrukcję **STACK_POP.**
  CONTROL_BUS[6..11] = 111111, CONTROL_BUS[0..5] = 100100.

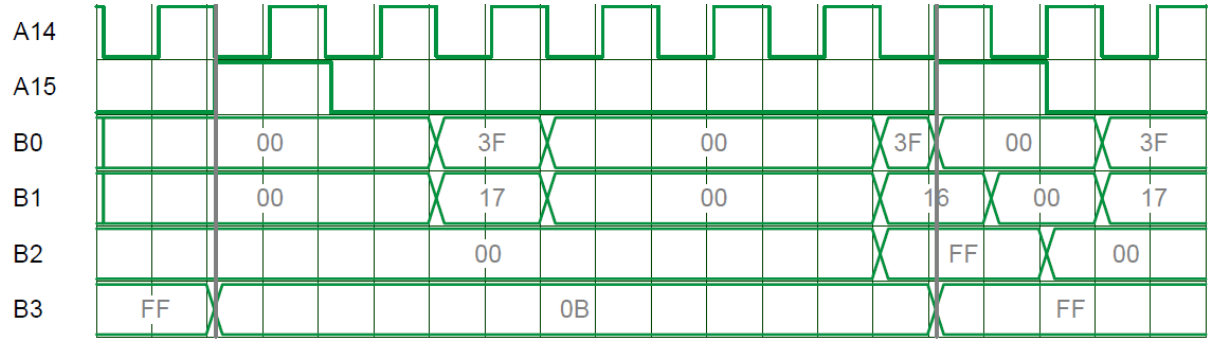


Rysunek 12 Wykres czasowy przebiegu instrukcji POP A

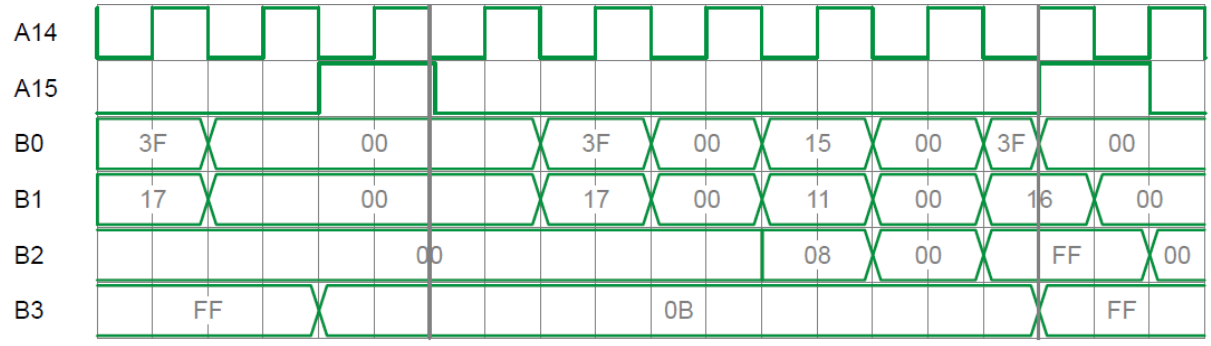
12) Instrukcja JO @Direct (0000 10112 = 0B16):

Instrukcja **JO** składa się z **2** mikroinstrukcji. Ta instrukcja nie występuje w procesorze 8051.

- Wykonuje mikroinstrukcję **INCR_ADDR_REG_ROM.**
  CONTROL_BUS[6..11] = 111111, CONTROL_BUS[0..5] = 010111.
- Jeżeli flaga OVERFLOW jest ustawiona następuje przejście ROM_ADDR_REG na pozycję podaną w argumencie instrukcji
  CONTROL_BUS[6..11] = 111111, CONTROL_BUS[0..5] = 010111.



Rysunek 13 Wykres czasowy przebiegu instrukcji JO @Direct z przykładowego programu

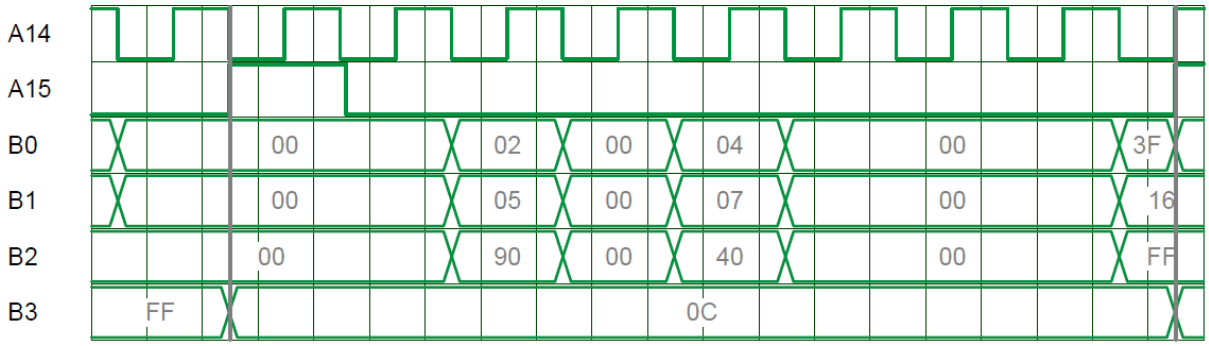


Rysunek 14 Wykres czasowy przebiegu instrukcji JO @Direct dla drugiego programu

13) Instrukcja ORL A B (0000 11002 = 0C16):

Instrukcja **ORL** składa się z **2** mikroinstrukcji.

- Kopiuje zawartość rejestru **A_REG** do rejestru **Temp1.**
  CONTROL_BUS[6..11] = 000010, CONTROL_BUS[0..5] = 000101.
- Kopiuje zawartość rejestru **B_REG** do rejestru **Temp2.**
  CONTROL_BUS[6..11] = 000100, CONTROL_BUS[0..5] = 000111.

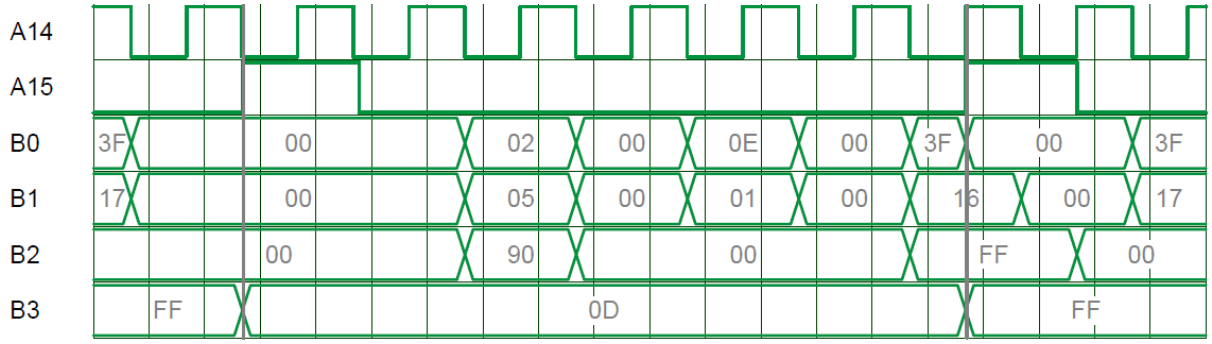


Rysunek 15 Wykres czasowy przebiegu instrukcji ORL A B

14) Instrukcja DEC A (0000 11012 = 0D16):

Instrukcja **DEC** składa się z **2** mikroinstrukcji.

- Kopiuje zawartość rejestru **A_REG** do rejestru **Temp1**.
  CONTROL_BUS[6..11] = 000010, CONTROL_BUS[0..5] = 000101.
- Wykonuje mikroinstrukcję **ALU_DECREMENT**, a jej wynik zapisuje w rejestrze **A_REG.**
  CONTROL_BUS[6..11] = 001110, CONTROL_BUS[0..5] = 000001.

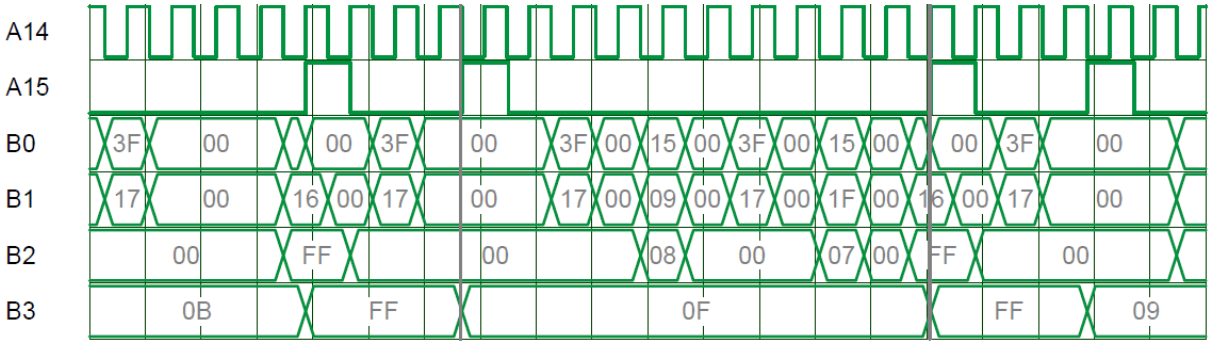


Rysunek 16 Wykres czasowy przebiegu instrukcji DEC A

15) Instrukcja INIT (0000 11112 = 0F16):

Instrukcja **INIT** składa się z **4** mikroinstrukcji.

- Wykonuje mikroinstrukcję **INCR_ADDR_REG_ROM**.
  CONTROL_BUS[6..11] = 111111, CONTROL_BUS[0..5] = 010111.
- Pobiera dane z pamięci **ROM** z umieszcza początek stosu w miejsce przez nie wskazywane.
  CONTROL_BUS[6..11] = 010101, CONTROL_BUS[0..5] = 001001.
- Wykonuje mikroinstrukcję **INCR_ADDR_REG_ROM**.
  CONTROL_BUS[6..11] = 111111, CONTROL_BUS[0..5] = 010111.
- Pobiera dane z pamięci **ROM** i umieszcza je w rejestrze **OPERAND**.
  CONTROL_BUS[6..11] = 010101, CONTROL_BUS[0..5] = 011111.



Rysunek 17 Wykres czasowy przebiegu instrukcji INIT

7. Tworzenie własnych programów

Aby napisać własny program działający na naszym procesorze wystarczy stworzyć plik „**DATA.bin**” i umieścić go w katalogu z projektem. Następnie edytujemy po np. przy pomocy Visual Studio.

Początek programu zawsze rozpoczynamy od instrukcji 00 ponieważ nasz układ nie posiada możliwości resetu. Następnie prowadzamy numer instrukcji zapisany w systemie szesnastkowym, a po nim argumentów które dana instrukcja przyjmuje np.--- 00 08 40 03 08 20 04 --- kopiuje wartość 40 do rejestru A, następnie kopiuje zawartość rejestru A do rejestru B, kopiuje wartość 20 do rejestru A, a następnie dodaje do siebie wartości z rejestrów A oraz B zapisując wynik w rejestrze A

8. Złożoność i budowa poszczególnych elementów

Wszystkie elementy posiadają aktywatory zapisu lub odczytu. Aktywator w sposób ciągły porównuje własny adres z szyną CONTROL/ ORDER_BUS.
Jeżeli wykryje zgodność wszystkich bitów adresu aktywuje linię najczęściej nazwaną ACTIVE która załącza obiekt do linii CONTROL20 (ZEGAR) co powoduje aktywowanie rejestrów.
Aktywator składa się z: 6/8 bramek NXOR oraz 4/5 bramek AND(wersja 1),
6/8 bramek NXOR 8 wejściowej bramki NAND i jednej bramki NOT(wersja 2).
Konstrukcje niektórych aktywatorów zostały zmienione w celu dopasowania ich do wymogów układu w którym pracują.
Aktywatory podpięte do szyny ORDER_BUS(znajdujące się w podukładzie 0000 0000- 0000 1111 układu CONTROL_REG) mają na celu rozpoznanie aktualnie wykonywanej instrukcji i załączenie odpowiedniego dekodera NB na 1 z n w celu poprawnego wykonania wszystkich mikrorozkazów danej instrukcji.

Należy zwrócić uwagę różnice w konstrukcji różnych rejestrów i układów. Pomimo tego że spełniają bardzo podobną funkcję w procesorze to jednak muszą one być bardzo wyspecjalizowane. Zbudowane one są z:
Aktywatorów,
Przerzutników D – blokujących w sobie stan 0 lub 1,
Przełączników – które zwierają i przerywają obwód w zależności od sygnału, który odbierają na bramce sterującej,
Sumatorów (w projekcie używamy sumatorów z szybkim obliczaniem przeniesień)– układów dodających do siebie dwie liczby 4bitowe (pełnią one również rolę dekrementerów),
Podstawowych bramek logicznych.

Rejestr	A_REG	B_REG	TEMP1	TEMP2	PSW	P_COUNTER
			Ilość			
Aktywator	2	2	1	1	17	2
Przerzutnik D	8	8	8	8	8	16
Układ 4016 (Przełącznik)	2	2	8	1	40	49
Bistable	0	0	16	0	0	0
Bramka NOT	0	0	0	0	8	0
Układ 4008 (Sumator)	0	0	0	0	0	2

Tabela 3 Porównanie ilości poszczególnych układów w rejestrach

W trakcie budowy procesora zostały również wykorzystane Bramki logiczne takie jak **OR**, **AND.** Korzystaliśmy również z układów takich jak **Układ 4077**, który zawiera w sobie 4 bramki **EX-NOR**, **Układ 4073**¸który zawiera w sobie 3 trzywejściowe bramki **AND. Układ 4081**, który zawiera w sobie 4 2-wejściowe bramki **AND**. **Układ 27128**, który jest układem scalonym pamięci **EPROM**(pamięć nieulotna kasowana przy pomocy światła ultrafioletowego). **Układ 4068**, który jest 8-wejściową bramką **NAND. Układ 4040**, który jest 12-stopniowym licznikiem dwójkowym. **Układ IS61C1023AL,** gotowy układ pamięci RAM. **Delay** – moduł wprowadzający opóźnienia w procesorze.

Rejestr **A_REG (ACC)** opiera się o przerzutniki D oraz aktywatory, które reagują na odpowiednią sekwencję bitów na szynie **CONTROL_BUS**. Jest on połączony z szybą

DATA_BUS oraz z rejestrem **TEMP2**. Rejestr **B_REG**. Jego budowa opiera się o przerzutniki D oraz aktywatory, które reagują na odpowiednią sekwencję bitów na szynie **CONTROL_BUS**. Jest on połączony z szybą **DATA_BUS**. Rejestr **TEMP1** opiera się o przerzutniki D oraz aktywatory, które reagują na odpowiednią sekwencję bitów na szynie **CONTROL_BUS**. Jest on połączony z szybą **DATA_BUS** oraz **ALU**. Rejestr **TEMP2** opiera się o przerzutniki D oraz aktywatory, które reagują na odpowiednią sekwencję bitów na szynie **CONTROL_BUS**. Jest on połączony z szybą **DATA_BUS** rejestrem **A_REG** oraz **ALU**.

Rejestr **PSW** (Program Status Word) znacząco różni się od reszty rejestrów, są w nim przechowywane flagi potrzebne do poprawnego funkcjonowania całego procesora. Jest on połączony z **ALU** oraz **DATA_BUS.**

Intel 8051 w swoim rejestrze PSW posiada flagi:

- **Carry** -przeniesienia
- **Aux Carry** - pomocniczego przeniesienia
- **General purpose** – flaga 0 ustawiana przez użytkownika
- **Bank SW0** - Register bank select 1.
- **Bank SW1** - Register bank select 0.
- **Overflow** - flaga przepełnienia
- **User** - flaga 1 ustawiana przez użytkownika
- **Party** - wskazanie parzystości

ALU czyli jednostka arytmetyczno logiczna składa się z układu kontrolującego przeprowadzane w nim operacje, oraz moduły wyspecjalizowane w obliczeniach tj. **Decrementer, Comparator, OR** oraz **Sumator. Comparator** jest układem porównującym dwie do siebie dwie liczny. Zwraca on 1 jeżeli dwie liczy są takie same, a 0 jeżeli chociaż 1 bit się różni. W celu magazynowania danych zbudowaliśmy pamięć **ROM**. Przechowuje ona informacje dotyczące programów które uruchamiamy w trakcie symulacji działania procesora. Ponieważ jest to pamięć nieulotna zapisane w niej dane nie znikają po wyłączeniu symulacji. **ROM ADDRESS REGISTER** jest rejestrem, który przetrzymuje adres dla pamięci **ROM**, przesyła go do niej cały czas i umożliwia szybkie operacje na pamięci. **PROGRAM COUNTER** przechowuje rozkaz instrukcji, która wykonywana będzie jako następna, opiera się on na przerzutnikach D, Aktywatorach oraz układach 4016. Jest to rejestr, który przetrzymuje adres dla pamięci ROM. wysyła go cały czas. umożliwia szybkie operacje na własnej pamięci. **DPTR** jest rejestrem, który przechowuje 16bitów dla pamięci zewnętrznej. **PORT2_DRIVER** jest to port przy pomocy którego procesor powinien komunikować się z pamięcią **RAM**. W naszej implementacji posiada on bufor o pojemności 1 bajta. **CONTROL_REG** jest modułem, który przechowuje oraz kontroluje wszystkie instrukcje wykonywane przez procesor. Możemy w nim również sprawdzić wykresy czasowe oraz liczbę cykli zegara od początku pracy i od początku aktualnie wykonywanej instrukcji. **Aktywator** jest układem bardzo podobnym do **Comparatora** ma on tą samą funkcję ale różni się w budowie. **Inkrementer** jest modułem, który dodaje do dane liczby 1.

Podsumowanie wszystkich elementów

NAZWA		ILOŚĆ
Układy 4077	NXOR	127
Układy 4073	3 WEJŚCIOWY AND	60
Układy 4081	UKŁAD 4 BRAMEK AND	58
Układy 4016	SWITCH	240
Układy 27128	PAMIĘĆ ROM	1
Układy 4008	SUMATOR	20
Układy 4068	8 WEJŚCIOWY NAND	17
Układy 4040	LICZNIK	1
Układy 4514	KONWERTER NB NA 1 z N	17
DTFF	PRZERZUTNIK	136
Bistable	PRZERZUTNIK	54
Is64c1024al-12ti	PAMIĘĆ RAM	2
Delay_1	UKŁAD WPROWADZAJĄCY OPÓŹNIENIE	23
Bramki OR		33
Bramka NOT		28
Bramki AND		286
Suma		**1103**

Tabela 4 Lista z ilością wszystkich wykorzystanych układów wraz z ich ilością

9. Problemy

a) **Program:** Na samym początku prac przy projekcie musieliśmy wybrać program w którym implementować będziemy nasz procesor. Chcieliśmy skorzystać z programu **BOOLR**, niestety okazał się on bardzo niestabilny. Symulacje podstawowych elementów naszego układy nie działały poprawnie. System zapisu układów również pozostawiał wiele do życzenia, gdy

projekt stawał się coraz większy części układu znikały bez powodu.
Dlatego zdecydowaliśmy się na korzystanie z nowego programu **Proteus 8**. Jest to program bardzo skomplikowany i na początku mieliśmy wiele trudności w trakcie korzystania z niego np. kopiowanie elementów odbywało się przy pomocy referencji to kompletnie uniemożliwiało korzystanie z tej funkcji i znacząco wydłużało czas pracy czy bardzo podobnych i powtarzających się komponentach. Na szczęście z czasem zaczęliśmy uczyć się jak działają zaawansowane funkcje tego programu co znacząco przyśpieszyło proces implementacji procesora.

[1]Przerzutniki bistable w projekcie używane są w celu uzyskania konkretnego stanu logicznego ze stanu nieustalonego.
Dla wyjścia Q działają one jak bramka AND dla stanów ustalonych na wejściach.
Jeżeli na wejściu wystąpi stan nieustalony na wyjściu zwracany jest stan niski.
Ten lub inny dedykowany układ powinien być ustawiony przed każdym wejściem każdej pierwszej bramki w układzie której wejście wychodzi poza układ

b) **Rejestry:** Podstawowymi elementami procesora są rejestry które tymczasowo przechowują dane potrzebne mu do prawidłowego funkcjonowania. Na początku stworzyliśmy własną wersję rejestru **ACC**. Następnie zaprojektowane zostały rejestry **B_REG**, **TEMP1**, **TEMP2** oraz **PSW**. Są one do siebie bardzo podobne więc ich stworzenie nie zajęło by dużo czasu gdybyśmy potrafili lepiej korzystać z wybranego przez nas programu.
c) **Zestaw instrukcji:** Kolejnym krokiem w trakcie projektowania naszego układu był wybór zestawu instrukcji które powinien on obsługiwać. Na zajęciach z projektu przedstawiliśmy nasze propozycje i wybraliśmy ten które powinny zostać zaimplementowane. W trakcie prac były one stopniowo dodawane. Ich dokładna lista znajduje się w dalszej części sprawozdania.
d) **Szyny danych:** Ponieważ poszczególne części procesora muszą się ze sobą komunikować, musieliśmy zaimplementować szybą **DATA_BUS**. Okazała się ona niewystarczający w skutek czego dodaliśmy **CONTROL_BUS** która steruje całym procesorem oraz **MEM_ADDR_BUS** przy pomocy której wysyłany jest adres w pamięci do którego chcemy uzyskać dostęp. W naszym procesorze znajdują się również szyny takie jak **T2ALU_BUS** czy też **T1ALU_BUSS** które łączą bezpośredni dwa pomocnicze rejestry z jednostką arytmetyczno logiczną. Ostatnimi szynami które możemy zobaczyć są **MEM_CTRL** która kontroluje moduł **Program Address Register**, **TEMP_IN_1** a także **PSW** która kontroluje działania na flagach.
e) **Pamięć:** Aby procesor mógł w pełni funkcjonować, potrzebuje pamięci z której mógłby korzystać w celu pobierania danych na których ma przeprowadzać operacje oraz zapisywania ich wyników. Początkowo radziliśmy sobie bez niej. Z czasem jednak okazało się że jest ona bardzo potrzebna przy testowaniu bardzie złożonych operacji. Na początku samodzielnie stworzyliśmy pamięć **ROM** przy pomocy podstawowych komponentów oferowanych przez program Proteus 8. Następnie zaprojektowana została pamięć **RAM** opierająca się na przerzutnikach. Ich struktura była bardzo rozbudowana na skutek czego program dział bardzo wolno i praca w nim była bardzo problematyczna. Na zajęciach zdecydowaliśmy że jednak nie będziemy budowali ich samodzielnie i skorzystamy z gotowych układów oferowanych przez nasz program. Modyfikacja ta spowodowała drastyczny wzrost komfortu pracy jednocześnie eliminując możliwość popełnienia przez nas błędów w jej implementacji. Aby pamięć **RAM** mogła działać potrzebuje własnego zegara.

f) **Automatyzacja:** Na początku wszystkie kody ustawaliśmy ręcznie. Dzięki stworzeniu CONTROL_REGISTER w którego skład wchodzą: deszyfratory rozkazów, licznik mikroinstrukcji pozwalający na rozeznanie się którą mikroinstrukcję należy wykonać jako następną, układ pozwalający automatycznie wczytać nowy rozkaz z pamięci do procesora.
g) **Licznik mikroinstrukcji:** Bez niego wszystkie mikroinstrukcje wykonywał się w niezdefiniowanych momentach. na postawie clock zlicza każdą zmianę mikroinstrukcji.
h) **Rejestr PSW:** Ponieważ potrzebowaliśmy rejestru przechowującego flagi musieliśmy dodać rejestr PSW. Jednak różni się on znacząco od pozostałych rejestrów więc wymagał on projektowania od samego początku ponieważ nie mogliśmy skorzystać z już istniejących modłów.
i) **Porty:** Stworzenie portu było kolejny problemem w trakcie prac. Ze względu na ich początkowy brak, pamięć **RAM** znajdowała się w środku procesora. W końcowych fazach rozwijania projektu porty zostały zaprojektowane od zera i dzięki temu **RAM** mógł być przeniesiony poza układ samego procesora.
j) **Stos:** Na początku założyliśmy że w naszym procesorze będzie można używać instrukcji PUSH oraz POP, gdybyśmy nie stworzyli modułu stosu byłoby to niemożliwe. Prostym rozwiązanie było stworzenie tego brakującego elementu naszego procesora oraz przydzielenie odpowiednich kodów dodawanym instrukcjom.
k) **Słaba optymalizacja**: Analizując wykresy czasowe zauważyliśmy że w trakcie wykonywania instrukcje występują cykle w trakcie których nie wykonywane są żadne mikroinstrukcje. Rozwiązanie tego problemu wymagało przebudowy całego układu, dzięki czemu znacząco przyspieszyliśmy działanie całej jednostki.
l) Każda zmiana w programie powoduje zmianę położenia wszystkich operacji i danych następujących po niej, co powoduje konieczność ponownego obliczenia adresów niezbędnych dla instrukcji **JMP** i **JO**.

10. Podsumowanie

Tworzonych przez nas układ był bardzo skomplikowany i wymagał dużej wiedzy związanej z działaniem procesora. W trakcie jego projektowania zrozumieliśmy wiele aspektów związanych z działaniem tego typu układu scalonego.

Podobieństwa i różnice w stosunku do oryginalnego procesora 8051:

1. W naszym procesorze zaimplementowaliśmy tylko część instrukcji z pośród wszystkich oferowanych w CPU na którym się wzorowaliśmy.
2. Korzystamy tylko z 2 flag oferowanych przez PSW pierwowzoru.
3. Ponieważ nasz procesor to symulacja idealna, opóźnienia związane z wykonywaniem operacji zostały prowadzone sztucznie.
4. Z powodu korzystania tylko z pamięci ROM do przechowywania numeru następnego rozkazu w ROM_ADDR_REG (Counter Register jest tylko przykładem 16bitowego rejestru).
5. Kody operacji w większości nie pokrywają się z kodami Intel 8051.
6. Port 2 nie współgra z pamięcią, potrafi on tylko wyświetlać dane na wyjściu.
7. W naszym procesorze występują trzy tryby przesyłania argumentu do instrukcji.

11. Bibliografia

- MCS® 51 MICROCONTROLLER FAMILY USER'S MANUAL
- Wykłady z przedmiotu Podstawy Techniki Mikroprocesorowej; Dr inż. Jacek Mazurkiewicz
- Wykłady z przedmiotu Organizacja i Architektura Komputerów; dr inż. Piotr Patronik

12. Spis tabel

13. Spis rysunków