POLITECHNIKA WROCŁAWSKA

Wydział Elektroniki

ORGANIZACJA I ARCHITEKTURA KOMUPTERÓW

Projekt i implementacja procesora liczącego na liczbach zespolonych

Sprawozdanie z realizacji projektu

Prowadzący:
Dr inż. Tadeusz Tomczak

Filip Gajewski, 236597

Andrzej Gierlak, 236411

Termin: CZ/TN/13:15

1. Cel projektu

Celem projektu było stworzenie procesora, którego główną cechą miało być przystosowanie do wykonywania obliczeń na liczbach zespolonych. Nie mieliśmy od górnie narzuconego rodzaju reprezentacji liczb zespolonych ani konkretnie sprecyzowanych działań jakie miał wykonywać procesor. Następnie naszym zadaniem była implementacja procesora w dowolnym programie umożliwiającym projektowanie i symulacje działania układów elektronicznych.

2. Założenia

Zdecydowaliśmy się zaprojektować nasz procesor w stylu architektury von Neumanna czyli dane oraz program są przechowywane razem. Procesor pracuje na słowach, danych i adresach składających się z 11 bitów. Początkowo projektowaliśmy procesor ze słowem o szerokości 8 bitów niestety przy tej konfiguracji mogliśmy pozwolić tylko na 4 rejestry swobodnego dostępu co uznaliśmy za zbyt małą liczbę. Procesor wykonuje obliczenia na całkowitych reprezentacjach liczb, wyniki są zaokrąglane przez obcięcie części ułamkowej. Oprócz zdolności do wykonywania operacji na liczbach zespolonych, procesor ma możliwość wykonywania operacji tylko na liczbach całkowitych.

3. Narzędzia i realizacja

Do implementacji naszego procesora użyliśmy darmowego programu logisim-evolution, który jest fanowskim rozwinięciem programu logisim, usprawnionego o dodatkowe opcje m.in. chronogram. Program był uruchamiany na Ubuntu 18.04.

Zrezygnowaliśmy z używania programu Tkgate, od którego zaczynaliśmy nasz projekt ponieważ działał on bardzo nie stabilnie, często się wieszał i wysypywał. Ponadto Tkgate przestał już być oficjalnie wspierany przez jego twórców.

4. Implementacja

4.1.Ogólny zarys

Nasz procesora składa się z dwóch głównych części: jednostki arytmetyczno-logicznej(ALU) i jednostki sterującej(Control section). Oprócz tego w skład budowy procesora wchodzą rejestry:

- 8 rejestrów swobodnego dostępu o adresach od 000 do 111
- Rejestr przechowujący adres kolejnej instrukcji IAR
- Rejestr przechowujący aktualną instrukcje IR
- Rejestr flag
- 2 rejestry akumulujące wyniki ALU, jeden dla części rzeczywistej wyniku ACC_RE a drugi dla części urojonej ACC_IM
- 2 rejestry akumulujące nadmiary z operacji mnożenia ALU. Jeden dla części rzeczywistej Re_OV_MUL i drugi dla części urojonej Im_OV_MUL
- 3 rejestry tymczasowe TempB, TempC, TempD do których są ładowane argumenty ALU
- rejestr IAR_INC_TEMP służący do inkrementacji rejestru IAR

Do stworzenia całego układu użyliśmy:

- standardowych bramek logicznych: AND, OR, XOR, XNOR itp.
- Modułu pamięci ROM oferowanego przez program
- Multiplekserów
- Dekoderów
- Liczników
- Rejestrów
- sumatorów, subtraktorów, układów mnożących i dzielących dostępnych w programie

W pliku projektowym oprócz tego znajduje się też pamięć RAM składająca się z 2048 11 bitowych komórek, służąca do przechowywania danych, adresów i programu, nie jest ona jednak częścią samego procesora. Do stworzenia pamięci RAM wykorzystaliśmy wbudowany w program moduł tej pamięci.

4.2.Słowo procesora

Znaczenie poszczególnych bitów w słowie(bity są numerowane od 10 najstarszego do 0 najmłodszego):

- bit 10 - decyduje czy rozkaz jest wykonywany przez ALU(1) czy jest operacją na pamięci(0)
- bity 9-6 - kod operacji ALU lub na pamięci w zależności od bitu 10
- bity 5-3 - adres rejestru tymczasowego, który jest pierwszym argumentem danej instrukcji procesora. Dalej będzie nazywany jako rejestr A
- bity 2-0 - adres rejestru tymczasowego, który jest drugim argumentem danej instrukcji procesora. Dalej będzie nazywany jako rejestr B

Z takiego podziału bitów słowa możemy łatwo wyliczyć, że mamy dostępnych 8 rejestrów swobodnego dostępu. Maksymalnie procesor może mieć przypisanych 16 operacji ALU oraz 16 operacji na pamięci.

4.3.Opis architektury

Program, dane oraz adresy do innych miejsc w pamięci są przechowywane w pamięci RAM. Oprócz niej użytkownik układu ma dostęp do ośmiu rejestrów swobodnego dostępu. Wszystkie rejestry procesora po uruchomieniu posiadają wartość zero.

Po uruchomieniu procesora rozpoczyna się cykl pracy procesora który jest podzielony na dwie fazy: fazę pobrania i fazę wykonania.

W fazie pobrania procesor pobiera z pamięci RAM instrukcje dla procesora znajdującą się w komórce pamięci pod adresem przechowywanym w rejestrze IAR i zapisuje ją do rejestru IR. Następnie w zależności od treści zapisanej w IR układ wykonuje daną operacje na ALU lub na pamięci modyfikując zawartości rejestrów. Wszystkim tym zarządza sekcja kontrolna

procesora, której działanie koordynuje zawarty w niej stepper, który jest zrealizowany w formie dekodera wchodzących do niego sygnałów licznika. Pojedynczy krok trwa przez dwa cykle zegara, dlatego zakres pracy licznika wynosi od 0 do 11 a sygnał steppera jest uwzględniany z co drugiej pozycji. Cykl pracy procesora składa się z sześciu kroków. Trzy pierwsze kroki obsługują fazę pobrania a trzy ostatnie fazę wykonania. Każdy pojedynczy krok(z wyjątkiem trzeciego na który przypada dodatkowy sygnał obsługujący inkrementację IAR) to kombinacja dwóch sygnałów wyjściowych sekcji kontrolnej: sygnału ustawiającego wartość odpowiedniego rejestru i sygnału udostępniającego wartość odpowiedniego rejestru. Wszystkie używane kombinacje takich dwóch sygnałów są zapisane w pamięci ROM znajdującej się w sekcji kontrolnej.

Po wykonaniu pełnego cyklu pracy układ przystępuje do wykonania następnego. Praca procesora jest zatrzymywana po przez zresetowanie całego układu.

4.4.Wykaz sygnałów

Na wykresach czasowych znajdują się oznaczenia sygnałów oraz wartości rejestrów zgodne z poniższą tabelą.

Sygnał	Opis
CPUclk	Sygnał zegara procesora
IAR_E	Odczytanie adresu kolejnej instrukcji
IAR_S	Wczytanie adresu kolejnej instrukcji
IR_S	Wczytanie instrukcji
IarIncE	Odczytanie zawartości rejestru IarIncTmp
IarIncS	Wczytanie wartości do IarIncTmp
MAR_S	Ustawienie nowego adresu pamięci RAM
RAM_E	Odczytanie zawartości komórki pamięci RAM
RAM_S	Zapisanie wartości do komórki pamięci RAM
RX_E	Odczytanie wartości jednego z rejestrów swobodnego dostępu (numery od 0 do 7)
RX_S	Zapisanie wartości do jednego z rejestrów swobodnego dostępu (numery od 0 do 7)
AccReE	Odczytanie rzeczywistej części wyniku
AccReS	Zapisanie rzeczywistej części wyniku
AccImE	Odczytanie urojonej części wyniku
AccImS	Zapisanie urojonej części wyniku
ReOvE	Odczytanie nadmiaru części rzeczywistej
ImOvE	Odczytanie nadmiaru części urojonej
SetFlg	Ustawienie wartości flag
RstFlg	Resetowanie wartości flag
TMPX_S	Wczytanie wartości argumentu ALU to jednego z rejestrów B, C lub D

Tabela 1. Wykaz sygnałów

4.5.Zaimplementowane instrukcje do obsługi pamięci

Jak było wspomniane powyżej instrukcje działające na pamięci zaczynają się od 0 na najstarszym bicie, po którym następuje cztero-bitowy kod danej operacji. Instrukcje są przeprowadzane w fazie wykonania, cyklu pracy procesora czyli w krokach od czwartego do maksymalnie szóstego. Do każdej z poniższych instrukcji załączone zostały wykresy czasowe przebiegu działania prostego programu demonstrującego działanie danej instrukcji. Każdy program będzie się kończył instrukcją zawieszenia działania procesora tzn. instrukcją skoku do miejsca tej instrukcji, co stworzy nieskończoną pętle. Wykres czasowy tej części programu

będzie pomijany przy opisie poszczególnych instrukcji jako nie związany z nimi. Wszystkie programy zapisywane są w kodzie szesnastkowym.
Kod programujący zawieszenie się procesora: 100 XXX, gdzie XXX to adres komórki przechowującej rozkaz skoku.

4.5.1. LOAD

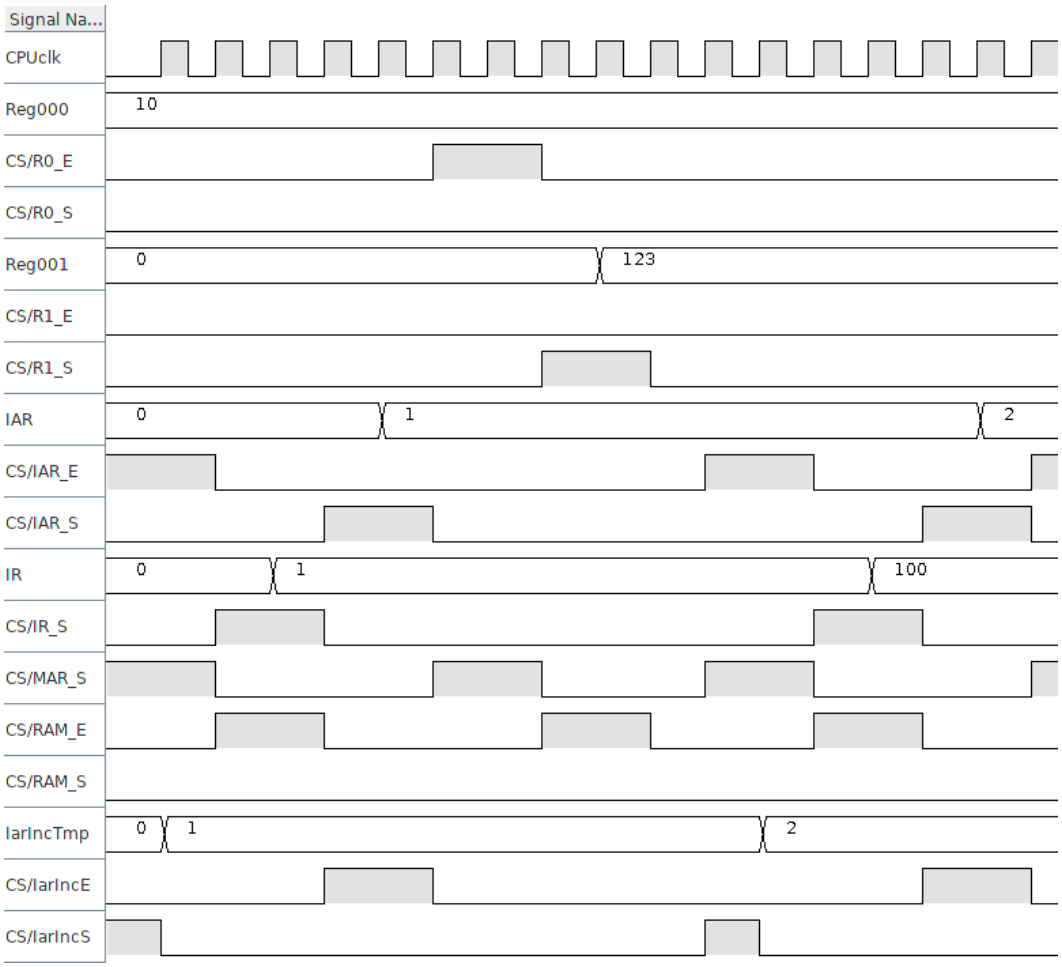
Kod operacji: 0000

Wczytuje do rejestru B zawartości komórki z pamięci RAM o adresie zawartym w rejestrze A.

Wykonanie instrukcji:

- W kroku 4 zawartość rejestru A jest udostępniana i zapisywana jako adres w pamięci RAM.
- W kroku 5 udostępniana jest zawartość aktualnie wskazywanej komórki RAMu i zapisywana do rejestru B

Kod programu demonstracyjnego:
001 100 001
W rejestrze A znajduje się adres 010 pod którym znajduje się wartość 123, która ma zostać wczytana do rejestru B.



Rysunek 1 Wykres czasowy demonstracji instrukcji LOAD

4.5.2. STORE

Kod operacji: 0001

Zapisuje zawartość rejestru B do pamięci RAM pod adresem znajdującym się w rejestrze A.

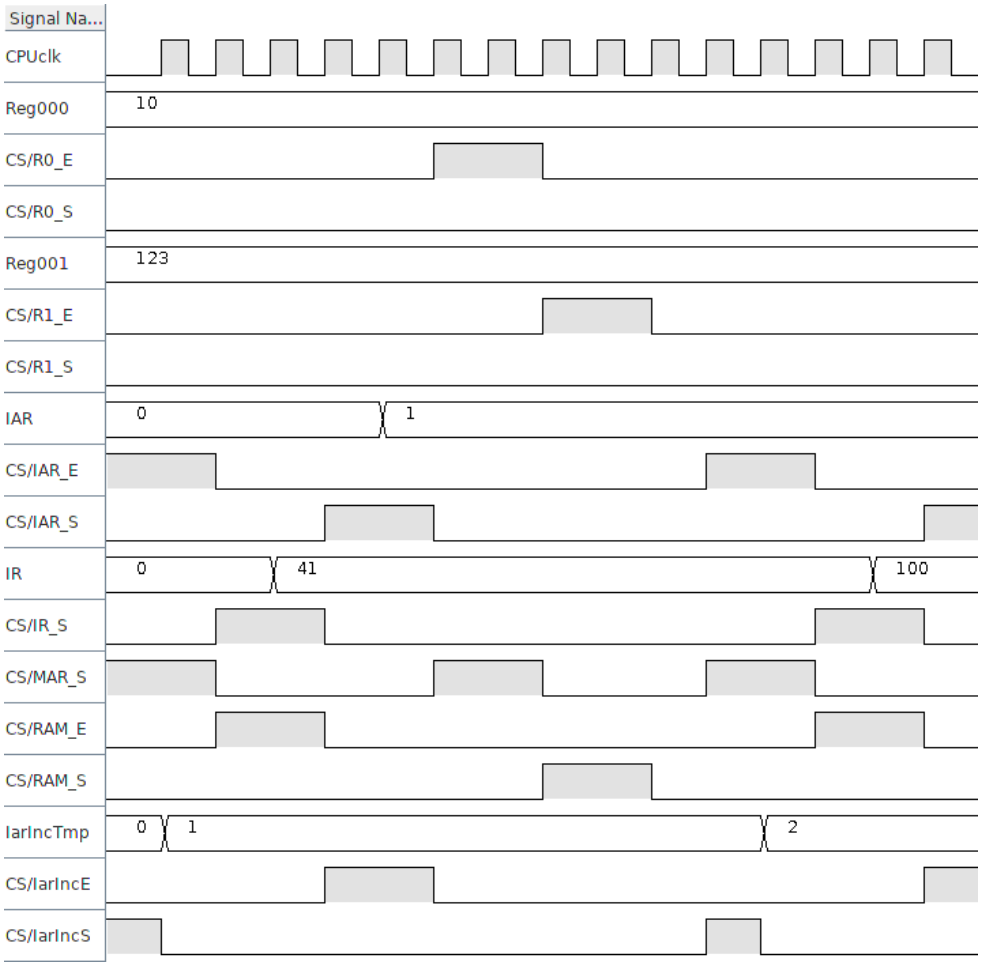
Wykonanie instrukcji:

- W kroku 4 zawartość rejestru A jest udostępniana i zapisywana jako adres w pamięci RAM.
- W kroku 5 udostępniana jest zawartość rejestru B i zapisywana jest do komórki pamięci RAM pod wskazanym adresem.

Kod programu demonstracyjnego:

041 100 001

W rejestrze A znajduje się adres 010 natomiast w rejestrze B znajduję się wartość 123, która ma być pod tym adresem zapisana.



Rysunek 2 Wykres czasowy demonstracji instrukcji STORE

4.5.3. DATA

Kod operacji: 0010

Wczytuje do rejestru B zawartość komórki pamięci RAM następującej po tym rozkazie. Rejestr A nie jest używany przez tą instrukcje, ta część rozkazu może zwierać dowolną wartość.

Wykonanie instrukcji:

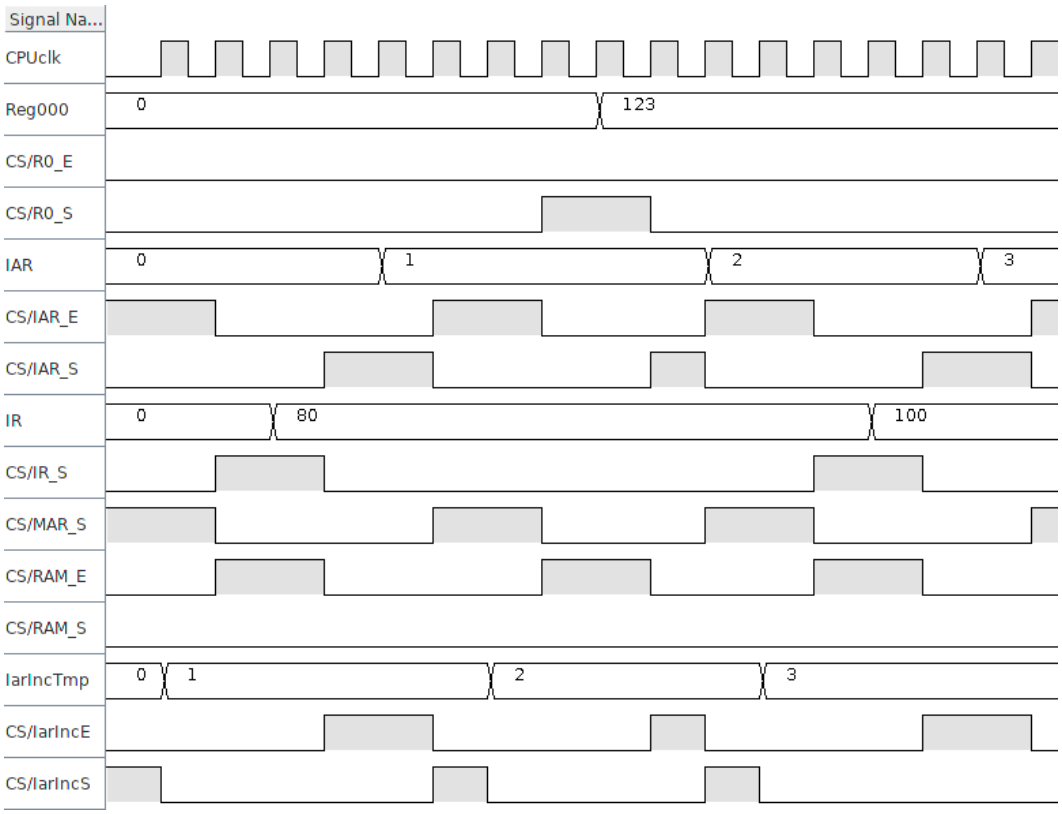
- W kroku 4 następuje pobranie zawartości IAR i zapisanie jej jako adresu w pamięci RAM. Zawartość IAR została już inkrementowana w kroku 1 i teraz wskazuje na zawartość którą chcemy wczytać do rejestru B. Oprócz tego w kroku 4 też

wykonujemy inkrementacje aby po zakończeniu rozkazu DATA pamięć wskazywała na komórkę z kolejny rozkazem do pobrania. Robimy to po przez przekazanie do ALU kodu operacji inkrementacji jej argumentu A oraz po przez ustawienie tej wartości do rejestru IarIncTmp.

- W kroku 5 odczytujemy zawartość z komórki pamięci RAM i zapisujemy ją do rejestru B.
- W kroku 6 odczytujemy zawartość IarIncTmp i zapisujemy ją do IAR, w ten sposób dokonaliśmy inkrementacji adresu pamięci RAM na kolejną instrukcję po rozkazie DATA i zawartości wczytywanej.

Kod programu demonstracyjnego: 080 123 100 002

Program wczytuje do rejestru B zawartość następnej komórki pamięci czyli 123



Rysunek 3 Wykres czasowy demonstracji instrukcji DATA

4.5.4. JMPR

Kod operacji: 0011

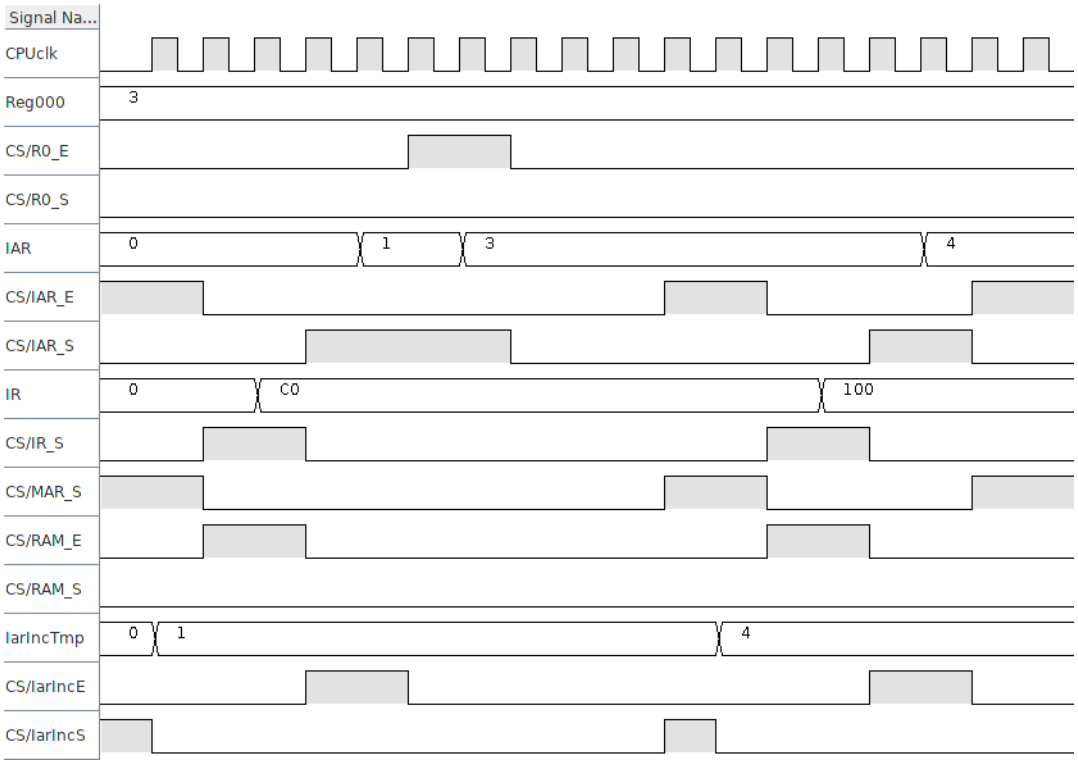
Wykonuje bezwarunkowy skok do miejsca w pamięci RAM znajdującego się pod adresem zapisanym w rejestrze B. Rejestr A nie jest wykorzystywany przez tą operację zatem jego adres może być dowolną wartością.

Wykonanie instrukcji:

- W kroku 4 zawartość z rejestru B jest udostępniana i zapisywana do rejestru IAR.

Kod programu demonstracyjnego: 0c0 000 000 100 003

Program skacze z adresu 000 do adresu 003 zapisanego w rejestrze B



Rysunek 4 Wykres czasowy demonstracji instrukcji JMPR

4.5.5. JUMP

Kod operacji: 0100

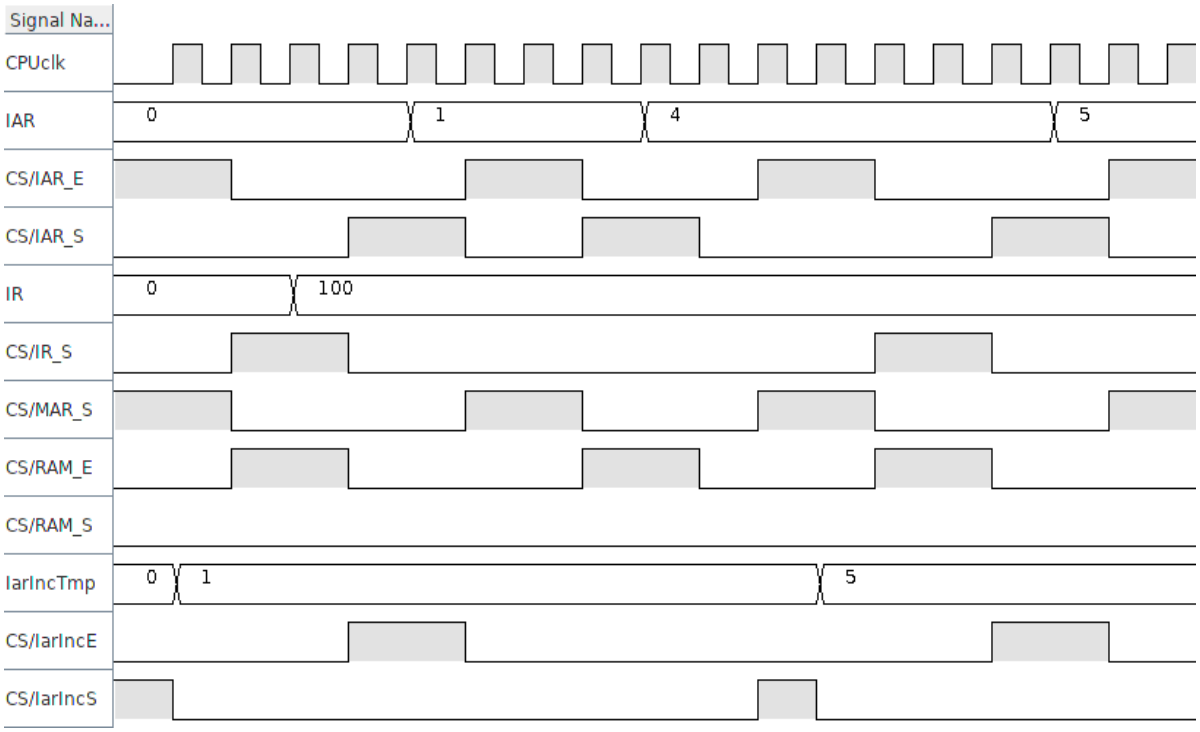
JUMP wykonuje bezwarunkowy skok do komórki pamięci RAM znajdującej się pod adresem zawartym w komórce pamięci następującej po tej instrukcji. JUMP nie wykorzystuje rejestrów A i B zatem ich adresy w instrukcji mogą być dowolne.

Wykonanie instrukcji:

- W kroku 4 udostępniamy zawartość IAR i zapisujemy ją jako adres w pamięci RAM.
- W kroku 5 udostępniamy zawartość tej komórki RAMu i zapisujemy ją do IAR.

Kod programu demonstrującego: 100 004 000 000 100 004

Program wykonuje skok z adresu 000 do adresu 004 zawartego w komórce pamięci następującej po tej instrukcji.



Rysunek 5 Wykres czasowy demonstracji instrukcji JUMP

4.5.6. JMP_IF

Kod operacji: 0101

JMP_IF zachowuje się analogicznie do rozkazu JUMP. Jedyną różnicą jest to że nie wykonuje on skoku bezwarunkowego a skacze wyłącznie wtedy gdy jest ustawiona pożądana przez użytkownika flaga. W naszym procesorze jest zaimplementowanych 5 różnych flag zgłaszanych przez ALU w trakcie wykonywania operacji. W instrukcji JMP_IF bit o numerze 5 nie odpowiada za nic więc jego wartość jest dowolna. Pięć najmłodszych bitów słowa (od 4 do 0) określa w przypadku jakich flag skok ma się wykonać:

- bit 0 - argument A ALU jest równy zero
- bit 1 - liczby zespolone są równe (A,B) = (C,D)
- bit 2 - argument ALU A jest większy od C
- bit 3 - argumenty ALU A i C są równe
- bit 4 - wystąpiło dzielenie przez zero, na liczbach całkowitych lub zespolonych

Jeśli został spełniony co najmniej jeden z wybranych przez nas warunków przebieg tej instrukcji jest identyczny jak JUMP.

4.5.7. SET_IM

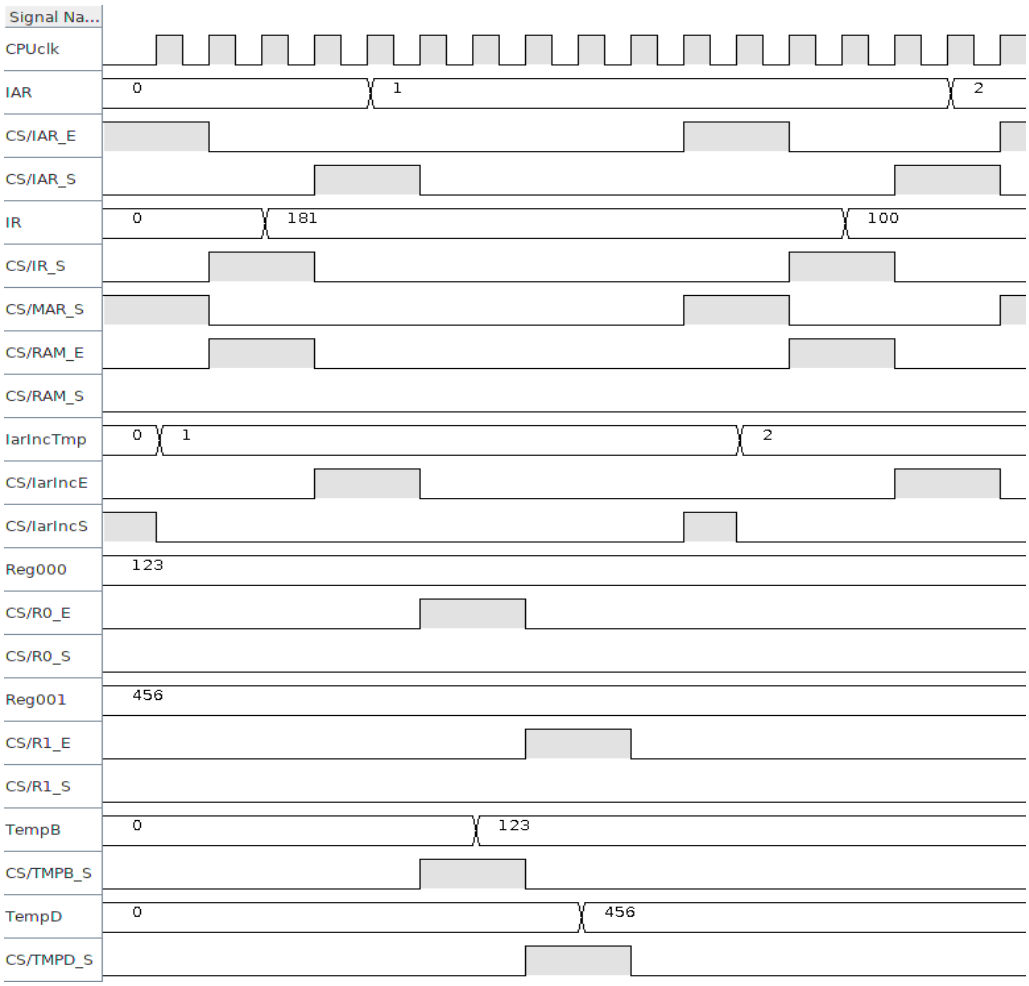
Kod operacji: 0110

Wczytuje zawartości dwóch dowolnie wybranych rejestrów do rejestrów tymczasowych TEMPB i TEMPD, które są częściami urojonymi dwóch liczb na których ALU będzie wykonywać operacje. TEMPB jest częścią urojoną pierwszej liczby operacji ALU a TEMPD częścią urojoną drugiej liczby. Do TEMPB jest wczytywany rejestr A słowa a do TEMPD rejestr B. Rozkaz ten jest niejako przygotowaniem argumentów do operacji ALU.
Wykonanie instrukcji:

- W kroku 4 następuje udostępnienie zawartości rejestru słowa A i zapisanie go do rejestru TEMPB.
- W kroku 5 następuje udostępnienie zawartości rejestru słowa B i zapisanie go do rejestru TEMPD.

Kod programu demonstracyjnego: 181 100 001

Program wczytuje zawartości rejestrów 000 i 001, odpowiednio 123 i 456 jako części zespolone, na których można będzie wykonywać obliczenia.



Rysunek 6 Wykres czasowy demonstracji instrukcji SET_IM

4.5.8. STORE_RESULTS

Kod operacji: 0111

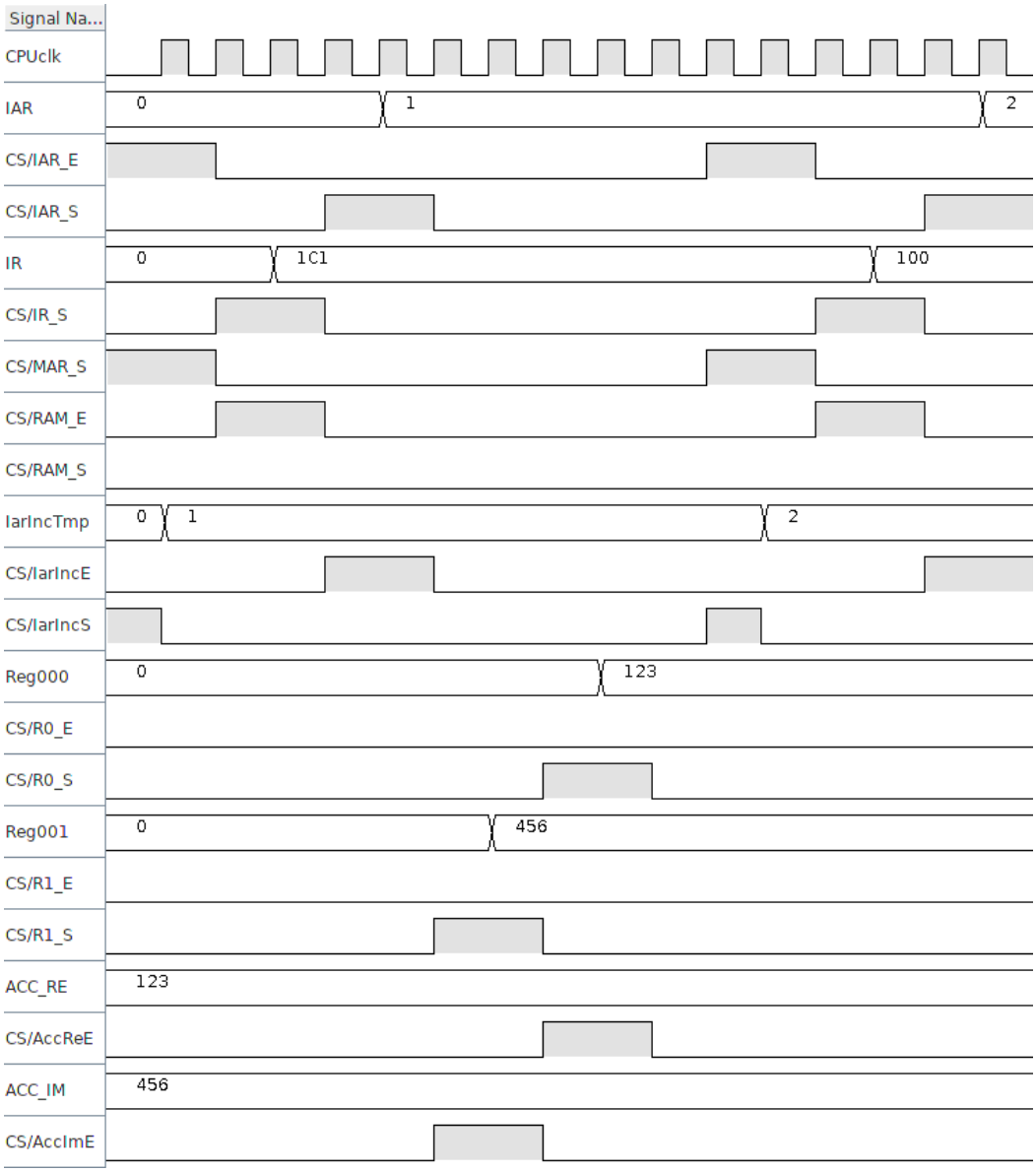
Kopiuje zawartość akumulatorów ACC_IM i ACC_RE odpowiednio do rejestrów B i A, w których znajdują się wyniki ostatniej operacji wykonanej przez ALU. Operacja zapisu jest wykonywana w kolejności, że najpierw jest zapisywana zawartość ACC_IM a dopiero po niej ACC_RE. Jeśli chcemy zapisać tylko zawartość ACC_RE (np. jeśli wykonywaliśmy operacje tylko na liczbach całkowitych) to adres rejestru A i B muszą być takie same. Wartość rejestru ACC_IM zostanie zapisana do rejestru docelowego a następnie nadpisana przez zawartość ACC_RE.

Wykonanie instrukcji:

- W kroku 4 udostępniana jest zawartość rejestru ACC_IM i zapisywana do rejestru B.
- W kroku 5 udostępniana jest zawartość rejestru ACC_RE i zapisywana do rejestru A.

Kod programu demonstracyjnego: 1c1 100 001

Program wczytuje do rejestru 000 znajdującą się w ACC_RE wartość 123 oraz do rejestru 001 wczytuje znajdującą się w ACC_IM wartość 456.



Rysunek 7 Wykres czasowy demonstracji instrukcji STORE_RESULTS

4.5.9. RESET_FLG

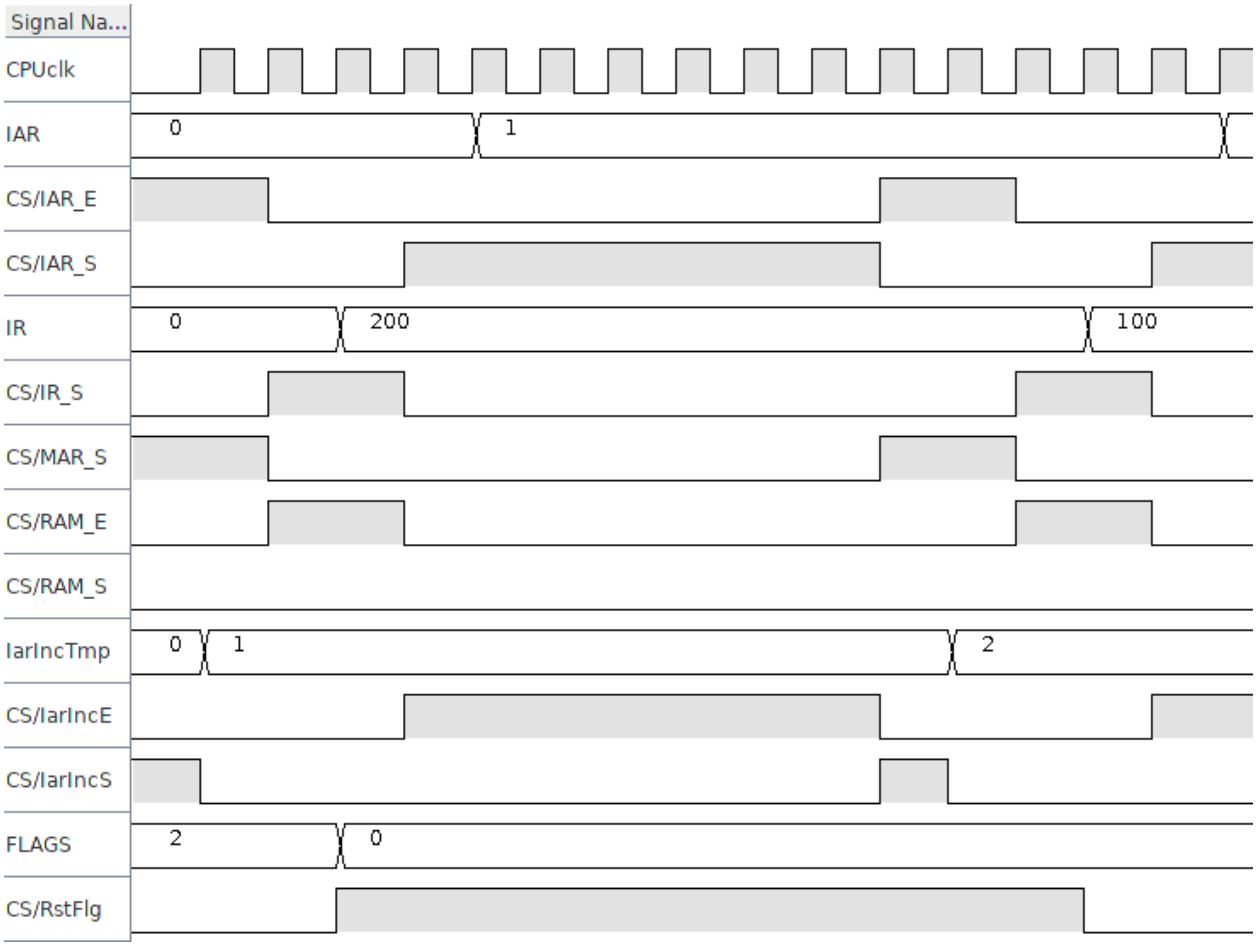
Kod operacji: 1000

Rozkaz resetuje rejestr FLAGS przechowujący informacje o tym, które flagi zostały ustawione. Rejestry A i B nie są wykorzystywane w tej operacji więc wartość ich adresów w rozkazie może być dowolna.

Wykonanie instrukcji:

- W kroku 4 z sekcji kontrolnej wychodzi sygnał ustawiający wartość rejestru FLAGS na zero.

Kod programu demonstracyjnego: 200 100 001



Rysunek 8 Wykres czasowy demonstracji instrukcji RESET_FLAGS

4.5.10. STORE OVs

Kod operacji: 1001

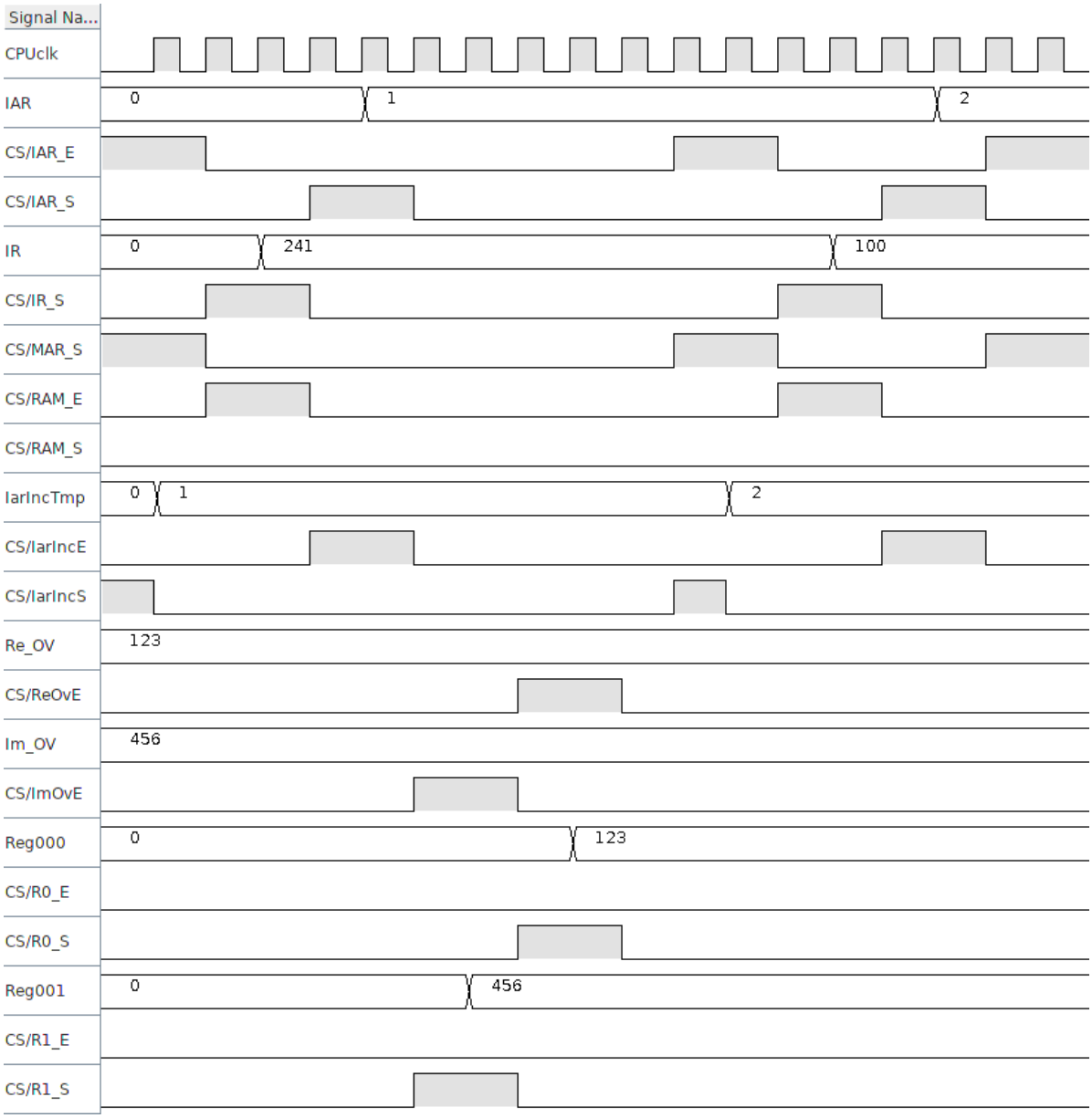
Rozkaz służy do zapisania zawartości rejestrów Re_OV i Im_OV, które przechowują nadmiary części rzeczywistej i urojonej wyników po wykonaniu operacji przez ALU, do rejestrów A i B. Podobnie jak w przypadku instrukcji STORE_RESULTS najpierw jest zapisywany nadmiar części urojonej a dopiero potem rzeczywistej. Jeśli chcemy więc zapisać tylko nadmiar części rzeczywistej adresy rejestrów A i B muszą być jednakowe. Wtedy w docelowym rejestrze najpierw zostanie zapisany nadmiar części zespolonej a następnie zostanie nadpisany nadmiarem części rzeczywistej.

Wykonanie instrukcji:

- W kroku 4 udostępniamy zawartość rejestru Im_OV i zapisujemy ją do rejestru B.
- W kroku 5 udostępniamy zawartość rejestru Re_OV i zapisujemy ją do rejestru A.

Kod programu demonstracyjnego: 241 100 001

Program wczytuje wartość rejestru Re_OV, 123 oraz rejestru Im_OV, 456 odpowiednio do rejestrów 000 i 001.



Rysunek 9 Wykres czasowy demonstracji instrukcji STORE_OVs

4.6.Implementacja ALU

Jednostka arytmetyczno logiczna została zaimplementowana tak aby mogła wykonywać operacje zarówno na liczbach zespolonych oraz całkowitych.

Posiada cztery wejścia na argumenty oznaczone jako A, B, C i D. A i C to wejścia na części rzeczywiste a B i D na części urojone. Wejście na czterobitowy kod operacji determinujący wynik jaki pojawi się na wyjściach. Dwa wyjścia wyniku, jedno odpowiada za część

rzeczywistą a drugie za urojoną. Dwa wyjścia nadmiaru wyniku, jedno na nadmiar części rzeczywistej, drugie na nadmiar części urojonej. Wyniki oraz ich ewentualne nadmiary po wykonaniu operacji są zapisywane do odpowiednich rejestrów tymczasowych. Dopiero z tych rejestrów za sprawą rozkazów STORE_RESULTS oraz STORE_OVs mogą być zapisane do dowolnie wybranych rejestrów swobodnego dostępu.
Ostatnim wyjściem jednostki jest wyjście flag sygnalizujących stan w jakim znajduje się procesor po ostatniej operacji ALU.

Dostępne flagi to:

- argument A ALU jest równy zero
- liczby zespolone są równe (A,B) = (C,D)
- argument ALU A jest większy od C
- argumenty ALU A i C są równe
- wystąpiło dzielenie przez zero, na liczbach całkowitych lub zespolonych

Rejestr przechowujący flagi może być ustawiony na wartość zero przez rozkaz RESET_FLAGS.
ALU jest układem kombinacyjnym. Nie posiada wejścia zegarowego a stan jego wyjść zależy wyłącznie od stanu jego wejść w danej chwili. Układ wykonuje wszystkie dostępne rodzaje operacji w tym samym momencie i kieruje wyniki na multiplekser, który filtruje pożądany wynik w zależności od kodu operacji podanego do ALU.

4.6.1. Zaimplementowane operacje wykonywane przez ALU

Kod	Opis
0000	Dodanie dwóch liczb całkowitych
0001	Dodawanie zespolone
0010	Odejmowanie zespolone
0011	Mnożenie zespolone
0100	Dzielenie zespolone
0101	Moduł z (A,B) do kwadratu
0110	Porównanie liczby A i C
0111	Pomniejszenie wartości A o 1
1000	Powiększenie wartości A o 1
1001	Iloczyn A i C
1010	Iloraz A i C

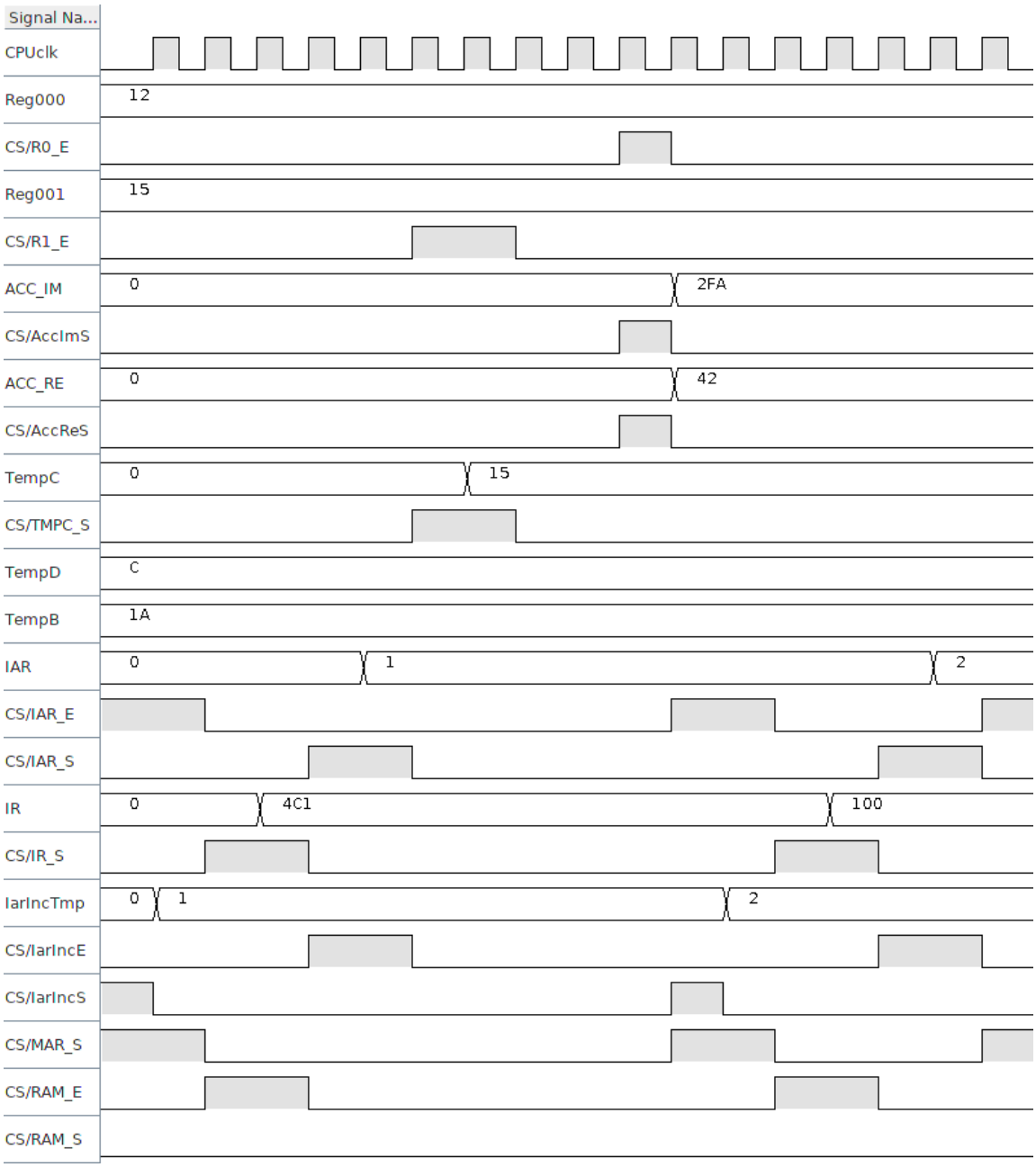
Tabela 2 wykaz operacji wykonywanych przez ALU

Operacja na ALU jest wykonywana po przez wczytanie do rejestru tymczasowego TEMPC, który w rozkazie podajemy jako rejestr B, wartości argumentu C ALU (część rzeczywista drugiej liczby). Następnie do głównej magistrali danych jest udostępniana zwartość dowolnego rejestru tymczasowego, którego adres został podany w rozkazie jako rejestr A. Jeżeli operacja ma zostać wykonana na liczbach zespolonych musimy pamiętać aby przed przystąpieniem do wykonywania operacji, za pomocą komendy SET_IM wczytać do rejestrów tymczasowych, TEMPB i TEMPD, części urojone liczb. Po wykonaniu zadanej operacji wyniki są zapisywane do akumulatorów ACC_RE(część rzeczywista wyniku) i ACC_IM(część urojona wyniku). W przypadku jeśli zadana operacja podaje wynik działania tylko dla wejść A i C to wartość akumulatora ACC_IM pozostaje taka sama jak przed wykonaniem operacji.

Przykład wykonania operacji na ALU

Kod programu demonstracyjnego: 4c1 100 001

Program wykonuje mnożenie zespolone na wartościach już wcześniej wczytanych do rejestrów tymczasowych TEMPB: 1A, TEMPD: C oraz do rejestrów swobodnego dostępu o adresach 000: 12 i 001: 15.



Rysunek 10 Wykres czasowy przykładowej operacji na ALU

4.7.Model AT

Niestety program, z którego korzystaliśmy nie oferował automatycznego policzenia obszaru układy czy wyznaczenia jego ścieżki krytycznej. Operacje te musieliśmy wykonać ręcznie. W naszym projekcie wykorzystywaliśmy gotowe układy, wykonujące działania arytmetyczne, logiczne, dekodery, multipleksery oraz układy pamięci. Środowisko również nie dostarczyło

informacji odnośnie tych elementów. Wszystkie wyniki zawarte poniżej zostały przez nas oszacowane na podstawie naszych własnych prototypów układów, które miały zachowywać się tam samo jak rozwiązania zawarte w środowisku. Niestety nie udało nam się zrealizować własnego prototypu układu dzielącego, działającego w ten sam sposób jak ten zawarty w programie. Niedogodność tą możemy ominąć wykonując dzielenie programowe procesorem. Pamięć ROM w naszym projekcie składa się łącznie z 256, 11 bitowych komórek, jednak w tabeli zamieściliśmy obszar zajmowany przez 30, wykorzystanych komórek.

4.7.1. Oszacowany obszar układu

Nazwa elementu	Ilość wystąpień	Oszacowana wartość obszaru jednego elementu
Rejestr 11 bitowy	19	91
Bramka AND	41	1
Bramka OR	26	1
Bramka XNOR	1	1
Dekoder	8	65
Licznik (12 stanów)	1	48
Pamięć ROM	1	2730
Komparator	1	75
Sumator 11 bitowy	11	77
Subtraktor 11 bitowy	7	77
Układ mnożący	13	1070
Multiplekser	7	70
Układ dzielący	3	----------
Suma		4306

Tabela 3 wykaz układów budujących procesor

4.7.2. Ścieżka krytyczna

Ścieżka krytyczna w naszym procesorze występuje podczas wykonywania mnożenia zespolonego. Jej początkiem jest jeden z ośmiu rejestrów swobodnego dostępu. Wartość ta wczytywana jest do rejestru tymczasowego TEMPC a następnie w trakcie wykonywania działania wartość ta przechodzi równolegle przez dwa układy mnożące. Później te dwa iloczyny równolegle przechodzą przez subtraktory, wynik jednego jest nadmiarem części rzeczywistej a drugi właściwym wynikiem. Wynik właściwy przechodzi przez nastawiony na odpowiednie działanie multiplekser i następnie jest zapisywany do rejestru ACC_RE, z

którego może być zapisany do dowolnego rejestru swobodnego dostępu. To kończy ścieżkę krytyczną procesora. Wartość ścieżki krytycznej oszacowaliśmy na 550.

RegXXX → TEMPC → Układ mnożący → Subtraktor → Multiplekser → ACC_RE

5. Wnioski

Projekt i implementacja procesora jest zdaniem bardzo złożonym, wymagającym przeanalizowania wielu zagadnień, dokładnego planowania i ciągłem testowania układu po dodaniu nowych elementów. Wiele czasu straciliśmy na poszukiwaniu odpowiedniego środowiska to realizacji projektu. Zaawansowane środowiska przeznaczone do projektowania takich układów jak np. ISE Xilinx przerastały nasze umiejętności. Natomiast środowiska do tworzenia prostszych układów nie spełniały naszych potrzeb.

Nie udało nam się stworzyć własnego prototypu układu dzielącego dwie liczby, takiego jaki mieliśmy dostępnego w programie logisim-evolution. Nasz procesor jest w stanie zrealizować operacje dzielenia programowo po przez odejmowanie dzielnika od dzielnej i zliczanie operacji odejmowania. Jednak w większości przypadków zajmie to znacznie więcej niż jeden cykl pracy procesora.

Nasz układ przy 11 bitowy słowie, przeznacza 4 bity na kody rozkazu dla ALU lub do zakodowania poszczególnych operacji na pamięci. Maksymalnie mogliśmy zakodować po 16 rozkazów. Niestety nie wykorzystaliśmy wszystkich możliwych kombinacji.

W trakcie projektowania procesora mieliśmy też pomysł na usprawnienie przesyłania części zespolonych liczb do rejestrów tymczasowych ALU po przez stworzenie oddzielnej linii przesyłającej te dane. Niestety nie wprowadziliśmy tych planów do naszej finalnej implementacji.

Projekt znajduje się w repozytorium Github: https://github.com/werd0n4/OiAK_Projekt

6. Spis załączników

- Schemat wykonanego układu
- Schemat ALU
- Program logisim-evolusion w formacie JAR
- Plik README
- Plik z pamięciom ROM sekcji sterującej

Spis treści

Spis ilustracji

Spis tabel