Politechnika Wrocławska

Wydział Elektroniki

Architektura komputerów 2
Projekt

Prosty procesor

Dokumentacja techniczna

Autor:
Tymoteusz Bobrowski
209769

Prowadzący:
dr inż. Bartosz Wojciechowski

ROZDZIAŁ 1 - Architektura

ROZDZIAŁ 2 - Implementacja

ROZDZIAŁ I

Architektura

1. Używane konwencje zapisu oraz skróty
   1.1. Bajt – 8 bitów, gdzie najstarszy bit – zapisywany najbardziej po lewej stronie, o najwyższym wykładniku potęgi.
   1.2. Zapis instrukcji – instrukcje są zapisywane w konwencji IA-32

   ```
   etykieta: mnemonik arg1, arg2
   ```

   Etykieta jest zapisem stosowanym w kodzie programu w celu ułatwienia zmiany kolejności wykonywania instrukcji (np. w instrukcji skoku).

   Mnemonik – zarezerwowana nazwa dla grupy instrukcji, które mają tę samą, bądź podobną funkcję.

   Argumenty (arg1, arg2) – arg2 jest opcjonalny. Rozkaz zawiera co najmniej jeden argument i co najwyżej dwa, zależnie od typu rozkazu. Argumenty mogą być nazwami rejestrów, adresami pamięci lub wartościami. Jeżeli rozkaz posiada dwa argumenty, lewy argument jest argumentem docelowym, prawy argument – źródłowym.

   1.3. Format zapisu liczb

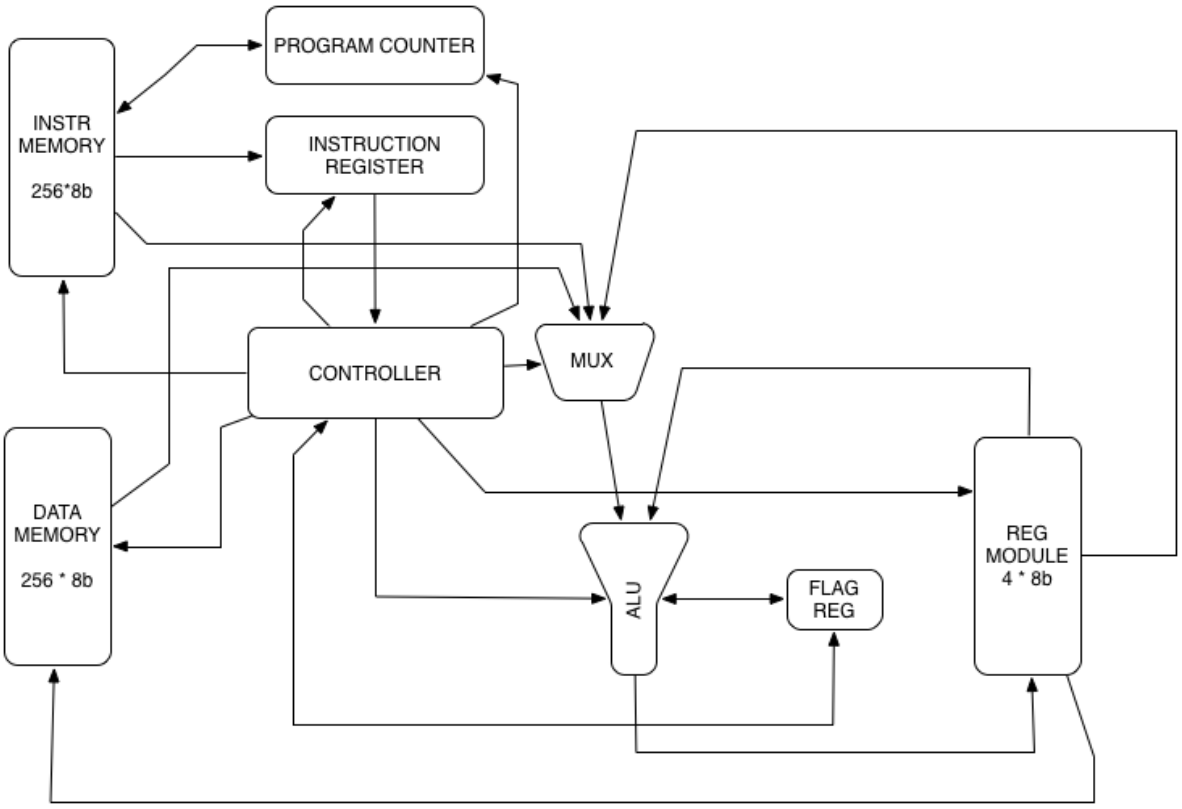
   W dokumentacji, system liczbowy jest podawany na końcu liczby, w indeksie dolnym, w nawiasach okrągłych, np. $0101_{(U2)}$

   W kodzie rozkazu liczby są zapisywane w systemie szesnastkowym (0-9, A-F) i poprzedzone są przedrostkiem 0x, np. 0xA1.

   Używane systemy liczbowe: system uzupełnieniowy do dwóch (U2), system szesnastkowy (HEX), system dziesiętny (DEC), system naturalny binarny (NB).

   1.4. Skróty
   - RAM – Random Access Memory - ogólnodostępna pamięć danych
   - ALU – Arithmetic Logic Unit – jednostka arytmetyczno-logiczna
   - CPU – Central Processing Unit - procesor

2. Mikroarchitektura CPU
   - 8 bitowa jednostka arytmetyczno logiczna
   - cztery rejestry ogólnego przeznaczenia R0 – R3 (8 bitowe)
   - rejestr flag (4 bitowy)
   - multiplekser 3 IN/1 OUT
   - 256 słów pamięci rozkazów (16 bitowych)
   - 256 słów pamięci danych (8 bitowych)
   - rejestr instrukcji (16 bitowy)
   - program counter (8 bitowy)
   - kontroler.



Rys 1. Schemat blokowy mikroarchitektury procesora.

3. Organizacja pamięci procesora

3.1. rejestry ogólnego przeznaczenia

Cztery 8-bitowe rejestry ogólnego przeznaczenia R0, R1, R2, R3, umożliwiające przechowywanie operandów operacji arytmetycznych i logicznych. Ładowanie rejestrów odbywa się poprzez multiplekser. Rejestry mogą być ładowane wartością: stałą, z pamięci danych, z innego rejestru.

3.2. rejestr flag

Jeden 4-bitowy rejestr przechowujący flagi, w kolejności od najstarszego bitu: flaga CF, ZF, SF, OVF

3.2.1. flagi statusu

Flagi są ustawiane jako efekt uboczny wykonywanych instrukcji arytmetycznych i logicznych (np. ADD). Funkcje poszczególnych flag:

- CF – Carry flag – flaga przeniesienia/pożyczki. Ustawiana, kiedy obliczony wynik przekroczy zakres reprezentacji, tzn. kiedy prawidłowy wynik nie mieści się na 8 bitach wyniku, które domyślnie zwraca ALU. W innym wypadku czyszczona.
- ZF – Zero flag – flaga zera. Ustawiana, kiedy obliczony wynik jest równy zero. W innym wypadku czyszczona.
- SF – Sign flag – flaga znaku. Ustawiana, kiedy obliczony wynik przyjmuje wartość ujemną. W przeciwnym wypadku czyszczona. Ustawienie flagi SF zależy od najstarszego bitu wyniku zwracanego przez ALU.

- OVF – Overflow flag – flaga nadmiaru. Ustawiana, kiedy obliczony wynik powoduje przepełnienie reprezentacji słowa procesora. Ustawiana, gdy z powodu ograniczonej ilości miejsca do zapisu wyniku, obliczony wynik powstaje nieprawidłowy, np. z dodawania dwóch liczb dodatnich powstaje liczba ujemna.

Flagi nie mogą być modyfikowane bezpośrednio poprzez wywoływanie instrukcji. Ustawianie/czyszczenie flag odbywa się zależnie od argumentów, za pomocą ALU.

3.3. rejestr instrukcji

16-bitowy rejestr instrukcji przechowuje kod aktualnie wykonywanego rozkazu. Jest ładowany po pobraniu instrukcji z pamięci instrukcji. Wykorzystywany jest przez kontroler, który umożliwia opcję zapisu instrukcji do rejestru instrukcji, a następnie korzysta z danych zapisanych w rejestrze instrukcji przy wykonywaniu operacji zawartych w instrukcji, np. pobieraniu adresów rejestrów lub stałych.

3.4. instruction pointer (program counter).

Program counter jest 8 bitowym rejestrem, który przechowuje adres pamięci, pod którym znajduje się kod aktualnie wykonywanego rozkazu. Rejestr jest ładowany przez kontroler kolejnymi wartościami z zakres8 0 – 255. Wartości nie muszą być przypisywane kolejno. Podczas instrukcji skoku, zawartość program counter jest ustawiana bezpośrednio z kontrolera, który w kodzie rozkazu otrzymuje adres kolejnej wykonywanej instrukcji.

4. Jednostka arytmetyczno-logiczna i multiplekser

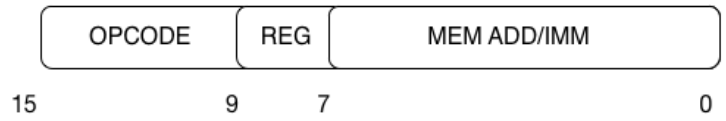
ALU wykonuje operacje arytmetyczne i logiczne na argumentach podawanych jako wejściowe, w zależności od wybranej operacji oraz odczytanych flag z rejestru flag. Zwraca 8 bitowy wektor binarny (w przypadku mnożenia 16 bitowy wektor binarny) oraz ustawia flagi, które następnie zapisywane są do rejestru flag.

Multiplekser jako swoje sygnały wejściowe przyjmuje stałą (najmłodsze 8 bitów kodu rozkazu), zawartość rejestru lub zawartość komórki pamięci. Zależnie od wyboru, przekazuje jedną z tych wartości na wyjście. Wyjście multipleksera jest jednocześnie wejściem ALU. Drugim wejściem ALU jest zawsze wartość z rejestru.

5. Układ sterujący – steruje pracą całego układu, uruchamia poszczególne bloki procesora, wysyła sygnały odczytu/zapisu, pośredniczy w przekazywaniu danych. Zależnie od kodu operacji wykonuje czynności niezbędne do jej wykonania angażując wymagane podzespoły i wysyłając odpowiednie wartości na wejścia, a także sterując wyjściami, aby wyniki trafiały we właściwe miejsca. Zarządza również dostępem do pamięci danych i rozkazów oraz działaniami program countera i rejestru instrukcji.
6. Rozkaz

6.1. budowa rozkazu

Kod rozkazu składa się z 16 bitów, gdzie najstarsze 6 bitów, to kod operacji, kolejne 2 bity, to kod rejestru, najmłodsze 8 bitów może być adresem w pamięci RAM lub 8 bitową stałą. Rozkazy są jednopoziomowe, tzn. dla kombinacji operandów rozkazu istnieje instancja tego samego rozkazu.



Rys 2. Schemat podziału kodu rozkazu.

6.2. adresowanie operandów

Istnieją 3 typy adresowania operandów rozkazu:

6.2.1. operandy bezpośrednie - w kodzie rozkazu przekazujemy stałą wartość, liczbę. Liczba ta kodowana jest na najmłodszych 8 bitach rozkazu.

6.2.2. operandy rejestrowe – w kodzie rozkazu przekazujemy kod lub kody rejestru. W przypadku instrukcji operującej na jednym rejestrze, kod rejestru jest zapisany na bitach 9-8. W przypadku instrukcji operującej na dwóch rejestrach, kod drugiego rejestru jest zapisany na bitach 1-0.

6.2.3. operandy pamięciowe – w kodzie rozkazu przekazujemy adres pamięci RAM. Ten typ adresowania wykorzystywany jest jedynie do zapisu danych z pamięci RAM do rejestru oraz zapisu danych z rejestru do pamięci. Adres rejestru źródłowego/docelowego jest zapisany na bitach 9-8, natomiast adres źródłowy/docelowy pamięci na bitach 7-0.

7. Podstawowe typy danych

Podstawowym typem danych, na których procesor może operować jest liczba całkowita ze znakiem. Liczba może być zapisana na 8 bitach. Operowanie na danych o długości większej niż 8 bitów nie zostało zautomatyzowane w tym procesorze i wymaga szczególnej uwagi użytkownika. Wartości, na których operuje ALU są kodowane w kodzie U2, zatem największa dodatnia liczba, którą można zapisać na jednym bajcie, to $01111111_{(U2)} = 127_{(DEC)}$, natomiast najmniejsza ujemna liczba, to $10000000_{(U2)} = -128_{(DEC)}$.

8. Asembler – zbiór rozkazów procesora

8.1. instrukcje arytmetyczne

ADD	dodawanie liczb całkowitych
ADC	dodawanie liczb całkowitych z uwzględnieniem flagi CF
SUB	odejmowanie liczb całkowitych
SUBB	odejmowanie liczb całkowitych z uwzględnieniem flagi CF
MUL	mnożenie liczb całkowitych
INC	inkrementacja
DEC	dekrementacja
CMP	porównanie dwóch liczb

8.2. instrukcje logiczne

ANL	operacja logiczna AND
ORL	operacja logiczna OR
XRL	operacja logiczna XOR
CPL	operacja logiczna NOT

8.3. instrukcje transferu danych

MOV	przenosi dane między rejestrami ogólnego przeznaczenia, między rejestrem a pamięcią oraz między pamięcią a rejestrem, umożliwia zapisanie stałej do rejestru

8.4. instrukcje zmiany sterowania

JMP	skok bezwarunkowy
JE	skok warunkowy jeżeli wynik CMP wskazuje równość
JG	skok warunkowy jeżeli wynik CMP wskazuje większość
JL	skok warunkowy jeżeli wynik CMP wskazuje mniejszość

8.5. dostępne argumenty

Rn	numer rejestru
n=[0;3]	rejestry od 0 - 3
0xNN	adres w pamięci
#0xNN	stała
NN=[00;FF]	max wartość zapisana na 8 bitach: FF

8.6. przykłady poleceń

MOV R1, 0x32	załaduj zawartość komórki pamięci o adresie 0x32 do rejestru R1
ADDC R2, #0x11	dodaj $11_{(HEX)}$ do wartości przechowywanej w rejestrze R2 i zapisz wynik w rejestrze R2
INC R3	inkrementuj zawartość rejestru R3 o 1, wynik zapisz w rejestrze R3
MOV 0x12, R2	załaduj zawartość rejestru R2 do pamięci pod adres 0x12

ROZDZIAŁ II

Implementacja

1. Środowisko

   Projekt został zaimplementowany w środowisku Xilinx ISE Design Software, ze względu na zintegrowanie kompilatora, symulatora oraz programu do implementacji kodu w FPGA w jednym pakiecie. Poszczególne moduły były pisane w języku VHDL. Symulacja działania przeprowadzana została przeprowadzona w symulatorze iSim. Większość modułów została zaimplementowana w architekturze Behavioral.

2. Etapy tworzenia poszczególnych modułów procesora.

   2.1. jednostka arytmetyczno-logiczna

   ALU posiada 5 portów:

   - num_A – port wejściowy, pierwszy operand do operacji arytmetycznych i logicznych, port otrzymuje na wejście wyjście z multipleksera
   - num_B – port wejściowy, drugi operand do operacji arytmetycznych i logicznych, port na wejście otrzymuje wyjście z rejestru
   - ALUs – port wejściowy, wybór operacji do wykonania przez jednostkę, port otrzymuje na wejście wartość podawaną z kontrolera
   - ALUout – port wyjściowy, zwraca wynik operacji arytmetycznych i logicznych
   - FLAGS_out - port wyjściowy, zwraca flagi wytworzone podczas wykonywanych operacji arytmetycznych i logicznych
   - FLAGS_in – port wejściowy, pobiera aktualny stan flag z rejestru flag

   Dodatkowo w module zawarte są zmienne lokalne pomocne przy wyliczaniu wyniku końcowego oraz flag. Wszystkie operacje wykonywane są w podobny sposób, z zapisywaniem wyniku najpierw do lokalnej zmiennej, aby zlokalizować wystąpienie poszczególnych flag. Następnie wartość ze zmiennej lokalnej jest przepisywana na port wyjściowy. Podobnie przepisywane są lokalne zmienne flag. Wyniki operacji arytmetycznych zapisywane są w rejestrze docelowym, podawanym jako pierwszy argument rozkazu. Operacją, która nie zwraca wyniku, a jedynie zmienia wartości flag jest CMP. Dla równych argumentów zwraca flagi kolejno CF,ZF,SF,OVF: 1111, dla num_A>num_B: 1000, dla pozostałych 0001. Operacja, która korzysta zarówno z argumentu źródłowego i docelowego, to MUL. Operacja ta po wykonaniu mnożenia zapisuje młodszą część wyniku w rejestrze docelowym, natomiast starszą część wyniku w rejestrze źródłowym.

   Testowanie modułu odbywało się poprzez wykonanie operacji w ALU dla wszystkich możliwych kombinacji danych wejściowych oraz zapisu wyników do pliku. Następnie plik został porównany z plikiem z prawidłowymi wynikami tych operacji.

   2.2. blok rejestrów

   Blok rejestrów posiada następujące porty:

   - clock – port wejściowy, sygnał zegarowy
   - rst – port wejściowy, sygnał reset
   - RFw1e – port wejściowy, bit zapisu pierwszego rejestru
   - RFw2e – port wejściowy, bit zapisu drugiego rejestru

- RFr1e – port wejściowy, bit odczytu pierwszego rejestru
- RFr2e – port wejściowy, bit odczytu drugiego rejestru
- RFw1a – port wejściowy, kod pierwszego rejestru do zapisu
- RFw2a – port wejściowy, kod drugiego rejestru do zapisu
- RFr1a – port wejściowy, kod pierwszego rejestru do odczytu
- RFr2a – port wejściowy, kod drugiego rejestru do odczytu
- RFw1 – port wejściowy, dane do zapisu do pierwszego rejestru
- RFw2 – port wejściowy, dane do zapisu do drugiego rejestru
- RFr1 – port wyjściowy, odczytu danych z pierwszego rejestru
- RFr2 – port wyjściowy, odczytu danych z pierwszego rejestru

Blok rejestrów jest zbudowany jako 4 elementowa tablica 8 bitowych wektorów. Zawiera funkcje write oraz read. Na takt zegara zależnie od ustawionych bitów odczytu lub zapisu do rejestru o zadanym adresie przypisywana jest wartość z portu wejściowego lub na port wyjściowy przekazywana jest wartość z rejestru.

Utworzenie oddzielnych portów dla dwóch rejestrów służy uproszczeniu procesów, w których potrzebujemy odczytać dane z dwóch rejestrów w tym samym czasie, np. w poleceniu ADD R1, R2 lub zapisać dane do dwóch rejestrów jednocześnie, np. po wykonaniu polecenia MUL.

2.3. rejestr flag

Porty rejestru flag:

- clock – port wejściowy, sygnał zegarowy
- rst – port wejściowy, sygnał reset
- FLwe – port wejściowy, bit możliwości zapisu flag do rejestru
- FLre – port wejściowy, bit możliwości odczytu flag z rejestru
- FLwd – port wejściowy, wektor bitowy, który otrzymuje flagi z ALU
- FLrd – port wyjściowy, wektor bitowy, który zwraca flagi przechowywane aktualnie w rejestrze flag

Działanie rejestru flag jest zbliżone do działania rejestrów ogólnego przeznaczenia. Zależnie czy ustawiony jest bit odczytu, czy zapisu pobierana jest wartość z tymczasowego sygnału w rejestrze i przekazywana na port wyjściowy lub wartość z portu wejściowego jest przekazywana do sygnału w rejestrze.

2.4. pamięć rozkazów i pamięć RAM

Obie pamięci mają niemal identyczną budowę. Porty pamięci RAM (dla pamięci instrukcji dodany przedrostek ‘X’):

- clock – port wejściowy, sygnał zegarowy
- rst – port wejściowy, sygnał reset
- Mre – port wejściowy, bit możliwości odczytu danych z pamięci
- Mwe – port wejściowy, bit możliwości zapisu danych do pamięci
- adress – port wejściowy, wektor bitowy, który otrzymuje adres w pamięci, z którego będziemy odczytywać dane lub pod który chcemy dane zapisać
- data_in – port wejściowy, dane do zapisu

- data_out – port wyjściowy, dane odczytane z pamięci

Pamięć RAM i pamięć rozkazów są zrealizowane podobnie jak pozostałe elementy pamięciowe. Jest to 256 elementowa tablica 8 bitowych wektorów (w przypadku pamięci kodu 16 bitowych). Zależnie od wartości bitów odczytu i zapisu dane są pobierane spod adresu znajdującego się na porcie address i przekazywane na port data_out lub pobierane z portu data_in i zapisywane w tablicy przechowującej komórki pamięci.

2.5. multiplekser

Porty:

- imm_in – port wejściowy, stała bezpośrednio z kodu rozkazu
- mem_in – port wejściowy, dane odczytane z pamięci RAM
- reg_in – port wejściowy, dane odczytane z rejestru
- mux_s – port wejściowy, wybór wejścia z kontrolera
- mux_ou – port wyjściowy, dane zwracane przez multiplekser

Multiplekser jest prostym układem, który zależnie od wartości portu mux_s przepisuje jeden z portów wejściowych na port wyjściowy.

2.6. rejestr instrukcji

Porty:

- IRin – port wejściowy, 15 bitowy wektor, zawierający kod rozkazu pobrany z pamięci rozkazów
- IRld – port wejściowy, bit możliwości zapisu do rejestru instrukcji
- IRout – port wyjściowy, zwraca zawartość rejestru instrukcji

Rejestr instrukcji swoje działanie opiera na bicie IRld. Kiedy bit jest ustawiony do rejestru ładowana jest wartość z portu IRin, w przeciwnym wypadku na port wyjściowy przekazywana jest wartość IRout.

2.7. program counter

Porty:

- PCld – port wejściowy, bit określający ładowanie program countera określoną wartością (przy instrukcji jump)
- PCinc – port wejściowy, bit określający czy należy inkrementować program counter
- PCclr – port wejściowy, bit określający reset stanu program countera
- PCin – port wejściowy, adres instrukcji po wykonaniu skoku
- PCout – port wyjściowy, zwraca adres aktualnie wykonywanej instrukcji

Podczas klasycznego działania program counter w czasie kolejnych taktów zegara, po wykonaniu niezbędnych obliczeń przez resztę komponentów ma ustawiany bit PCinc, aby przejść do następnego rozkazu. Jeżeli wystąpi instrukcja jump, ustawiany jest bit PCld. W takim wypadku z portu PCin pobierany jest adres docelowy skoku i taki adres jest wpisywany do program countera. Dalej program wykonuje się w sposób klasyczny, poczynając od nowego adresu.

2.8. kontroler

Porty:

- clock – port wejściowy, sygnał zegarowy
- rst – port wejściowy, sygnał reset
- IR_word – port wejściowy, aktualny kod instrukcji pobrany z IR
- RFs_ctrl - port wyjściowy, wybór dla multipleksera
- RFw1a_ctrl - port wyjściowy, ustawienie adresu zapisu do rejestru 1

- RFw2a_ctrl - port wyjściowy, ustawienie adresu zapisu do rejestru 2
- RFr1a_ctrl - port wyjściowy, ustawienie adresu odczytu z rejestru 1
- RFr2a_ctrl - port wyjściowy, ustawienie adresu odczytu z rejestru 2
- RFw1e_ctrl – port wyjściowy, ustawienie bitu zapisu do rejestru 1
- RFw2e_ctrl – port wyjściowy, ustawienie bitu zapisu do rejestru 2
- RFr1e_ctrl - port wyjściowy, ustawienie bitu odczytu z rejestru 1
- RFr2e_ctrl - port wyjściowy, ustawienie bitu odczytu z rejestru 2
- ALUs_ctrl - port wyjściowy, przekazuje kod operacji do ALU
- PCclr_ctrl - port wyjściowy, ustawienie bitu resetu program countera
- PCld_ctrl - port wyjściowy, ustawienie bitu ładowania program countera
- PCad_ctrl - port wyjściowy, przekazuje adres skoku do program countera
- IRld_ctrl - port wyjściowy, ustawienie bitu ładowania IR
- Mre_ctrl - port wyjściowy, ustawienie bitu odczytu z RAM
- Mwe_ctrl - port wyjściowy, ustawienie bitu zapisu do RAM
- Mra_ctrl - port wyjściowy, przekazuje adres odczytu/zapisu z pamięci ram
- XMre_ctrl - port wyjściowy, ustawienie bitu odczytu z pamięci rozkazów
- FLwe_ctrl - port wyjściowy, ustawienie bitu zapisu do rejestru flag
- FLre_ctrl - port wyjściowy, ustawienie bitu odczytu z rejestru flag
- FLin_ctrl - port wejściowy, odczytane flagi z rejestru flag (wykorzystywane przy sprawdzaniu warunków do skoków warunkowych)

Kontroler jest kluczowym elementem całego układu. Jest zbudowany jako skończony automat, w którym kolejno wykonywane operacje zależą od danych wejściowych oraz aktualnego stanu maszyny. Kontroler swoją pracę rozpoczyna od pobrania adresu instrukcji z portu `IR_word`. Zależnie od najstarszych 6 bitów na tym porcie, kontroler warunkuje swoją dalszą pracę. Dla każdego z zaimplementowanych poleceń istnieje w kontrolerze osobny stan maszyny. Stan zmienia się zawsze na każdy kolejny takt zegara. Dodatkowo każdy ze stanów głównych posiada określoną liczbę swoich podstanów (zwykle 4). Spowodowane jest to koniecznością zapewnienia ciągłości wykonywanych poleceń i synchronizacji ich z zegarem (tak, aby jedno polecenie nie spowodowało przerwania wykonania drugiego). Każde polecenie otrzymuje tyle taktów zegarowych, które są niezbędne do prawidłowego wykonania polecenia od początku do końca. W każdym stanie i podstanie ustawiane są odpowiednie bity odczytu, zapisu, czy inne wartości bitowe, przekazywane np. jako wybór operacji ALU lub wybór opcji w multiplekserze. Ustawienie poszczególnych wartości zależne jest od aktualnie wykonywanego polecenia. Z polecenia zawartego w `IR_word`, jeżeli istnieje taka potrzeba wyciągane są wartości, które są kodami rejestrów lub adresami pamięci i za pomocą odpowiednich portów przekazywane są w odpowiednie miejsca. Po wykonaniu każdego polecenia stany wykorzystywanych portów są przywracane do stanów początkowych, a stan kontrolera jest ustawiany do stanu pobrania następnej instrukcji.

2.9. oprócz wymienionych wyżej modułów, istnieją również moduły dodatkowe: datapath, CU i CPU. Moduły te służą do mapowania portów pomiędzy poszczególnymi modułami wymienionymi w punktach 2.1 – 2.8, czyli łączą w całość poszczególne elementy procesora. Wykorzystywanie portów oraz ich zależności przedstawione są w załącznikach 1,2,3.

3. Asembler

Asembler został napisany w języku Python. Program pozwala na tłumaczenie mnemoników na kod maszynowy. Wykonywanie programu rozpoczyna się od analizy kolejnych wierszy w pliku z kodem. Zależnie od tego, do którego wzorca pasuje dany wiersz, może on być zdefiniowany jako rozkaz lub etykieta. Następnie analizowane są operandy instrukcji. Również porównywane są z wzorcem. Kolejnym krokiem jest sprawdzenie, czy podany typ instrukcji, wraz z parametrami istnieje we wzorcu. Przy okazji, dla etykiet zapisywany jest kod wiersza, w której etykieta się znajduje. Dla znalezionego typu polecenia wywoływana jest funkcja, która tłumaczy mnemonik wraz z operandami na kod maszynowy. Na koniec ten kod maszynowy zapisywany jest do pliku wyjściowego. Operacja jest powtarzana dla wszystkich rozpoznanych rozkazów.

Kodowanie poszczególnych rozkazów, wraz z modyfikowanymi flagami po wykonaniu polecenia.
Legenda: R – rejestr, IMM – stała, M – adres w pamięci, LBL - etykieta

Polecenie	Kod	Modyf. flagi	Polecenie	Kod	Modyf. flagi
MOV R,IMM	000000	ZF, SF	DEC R	001110	CF,ZF,SF,OVF
MOV R,R	000001	ZF, SF	ANL R,IMM	010000	ZF,SF
MOV M,R	000011	ZF, SF	ANL R,R	010001	ZF,SF
MOV R,M	000010	ZF, SF	ORL R,IMM	010010	ZF,SF
ADD R,IMM	000100	CF,ZF,SF,OVF	ORL R,R	010011	ZF,SF
ADD R,R	000110	CF,ZF,SF,OVF	XRL R,IMM	010100	ZF,SF
ADC R,IMM	000111	CF,ZF,SF,OVF	XRL R,R	010101	ZF,SF
ADC R,R	001000	CF,ZF,SF,OVF	CMP R,IMM	010110	CF,ZF,SF,OVF
SUB R,IMM	001001	CF,ZF,SF,OVF	CMP R,R	010111	CF,ZF,SF,OVF
SUB R,R	001010	CF,ZF,SF,OVF	CPL	011011	ZF,SF
SUBB R,IMM	001011	CF,ZF,SF,OVF	JMP LBL	000101	-
SUBB R,R	001100	CF,ZF,SF,OVF	JE	011000	-
MUL R,R	001111	ZF,SF	JG	011001	-
INC R	001101	CF,ZF,SF,OVF	JL	011010	-

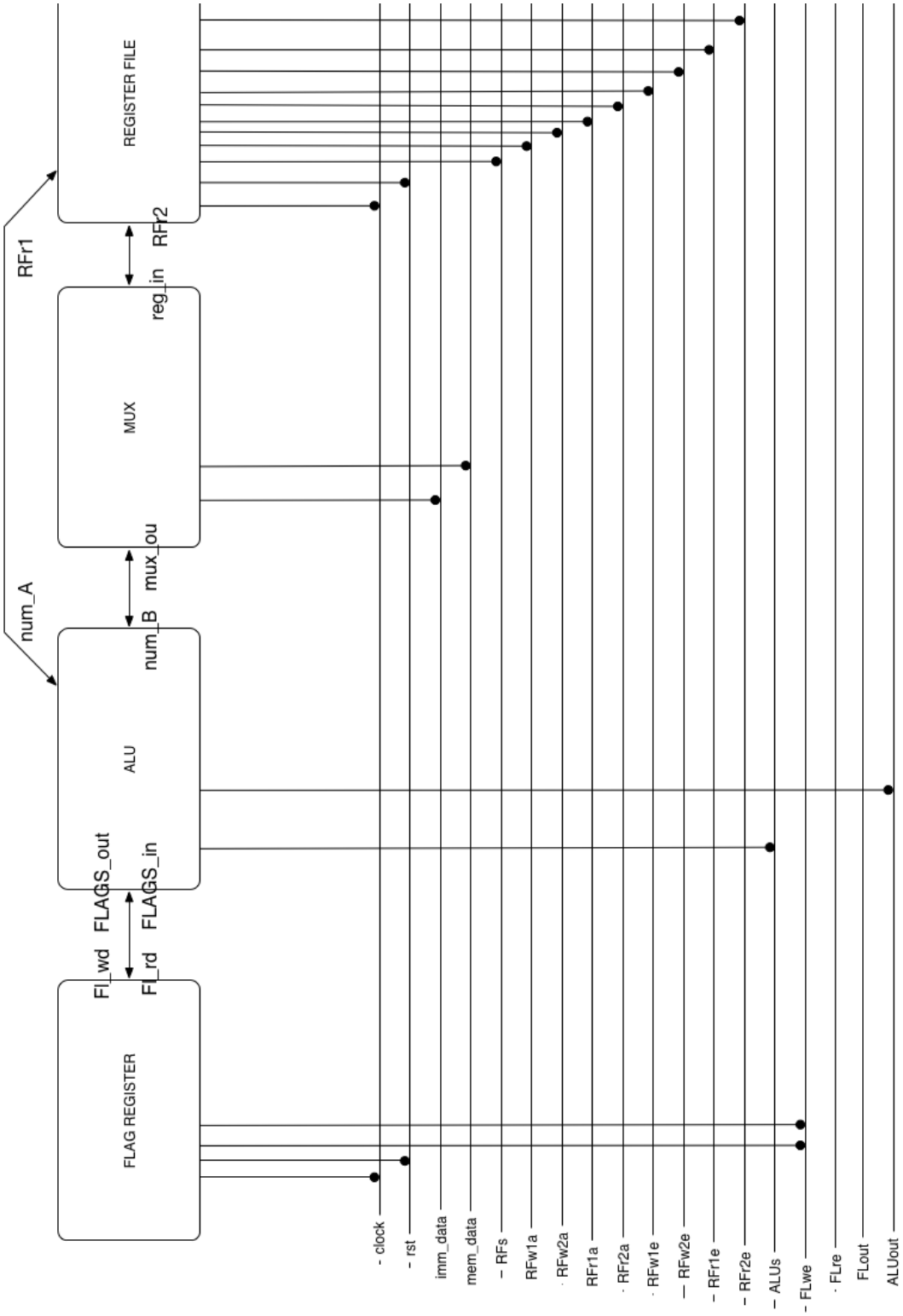
Kodowanie rejestrów :
R0 – 00, R1- 01, R2 – 10, R3 - 11

4. Testowanie układu

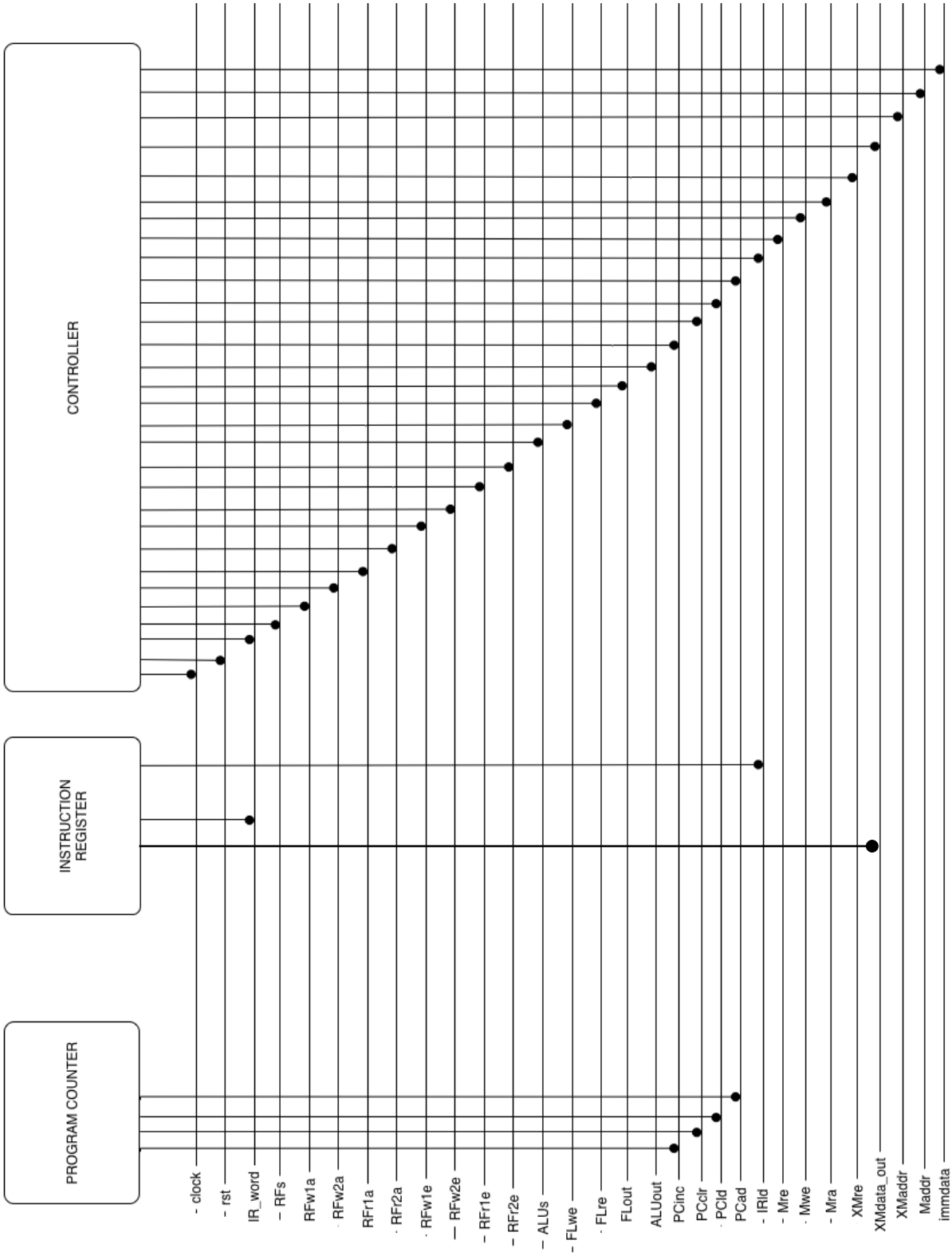
Testowanie układu odbywało kilku etapach.

- testowanie poszczególnych modułów
    - moduły pamięciowe: ustawianie bitów odczytu i zapisu, próba odczytania i zapisania danych z portów wejścia/wyjścia
    - jednostka arytmetyczno – logiczna: pkt 2.1
- testowanie całości układu
    - przetłumaczenie prostego programu napisanego w asemblerze
    - uruchomienie wytworzonego kodu maszynowego w procesorze i zasymulowanie jego działania
    - sprawdzanie poszczególnych stanów w symulacji całego układu.

Załącznik 1
Mapowanie portów procesora – data path.



Załącznik 2
Mapowanie portów procesora – instruction path



Załącznik 3
Mapowanie portów – układy pamięciowe.