Projekt procesora w języku VHDL

Sprawozdanie z zadania projektowego z przedmiotu
Architektura Systemów Komputerowych

Wydział	Wydział Inżynierii Elektrycznej i Komputerowej Politechniki Krakowskiej
Kierunek	Informatyka
Opracowali	Kamil Bałaban Albert Kanak
Grupa	środa 19:45 21i
Data	17.05.2020

Spis treści

Polecenie dla grupy projektowej

Oprócz podstawowych założeń projektowych przedstawionych w instrukcji, dostępnej na stronie laboratorium ASK, należy uwzględnić następujące dodatkowe założenia:

Szyna danych i szyna adresowa są 16-bitowe.

1. Lista rozkazów

MOV arg1, arg2 - przesunięcie arg2 do arg1 (arg1 - adres, arg2 - rejestr).
ADD arg1, arg 2 - dodanie arg2 do arg1, wynik w arg1 (arg1 - adres, arg2 - rejestr).
SUB arg1, arg 2 - odjęcie arg2 od arg1, wynik w arg1 (arg1 - adres, arg2 - rejestr).
CMP arg1, arg2, wynik (porównanie arg1 i arg2, arg1 – rejestr, arg2 – rejestr lub adres, jeśli arg1=arg2 wynik =1, w przeciwnym przypadku wynik=0, ustawiana jest też flaga Z)
PUSH arg1 (odłożenie wartości na stos, arg1 – rejestr, inkrementuje SP)
POP arg1 (pobranie wartości ze stosu, arg1 – rejestr, inkrementuje SP)
XNOR arg1, arg2 (arg1- rejestr, arg2 – rejestr, stała 5 bit, wynik w arg1)
RPL8 arg1 (zamiana 4 starszych bitów z 4 młodszymi bitami wartości 8-bitowej, arg1 – stała lub zawartość pamięci, wynik w arg1)
RPL4 arg1 (zamiana 2 starszych bitów z 2 młodszymi bitami wartości 4-bitowej, arg1 – stała lub zawartość pamięci, wynik w arg1)

W projekcie należy uwzględnić tylko rozkazy podane wyżej. W każdej instrukcji należy, jeśli jest taka potrzeba, ustawić znaczniki C, Z, S, P.

2. Rejestry

- **SP** – wskaźnik stosu (adres 16-bitowy, przemieszczenie w segmencie stosu)
- **ES** – rejestr segmentu (16-bitowy rejestr przechowujący adres segmentu)
- **AP1, AP2** – rejestry adresowe 8-bitowe do przechowywania adresu przemieszczenia

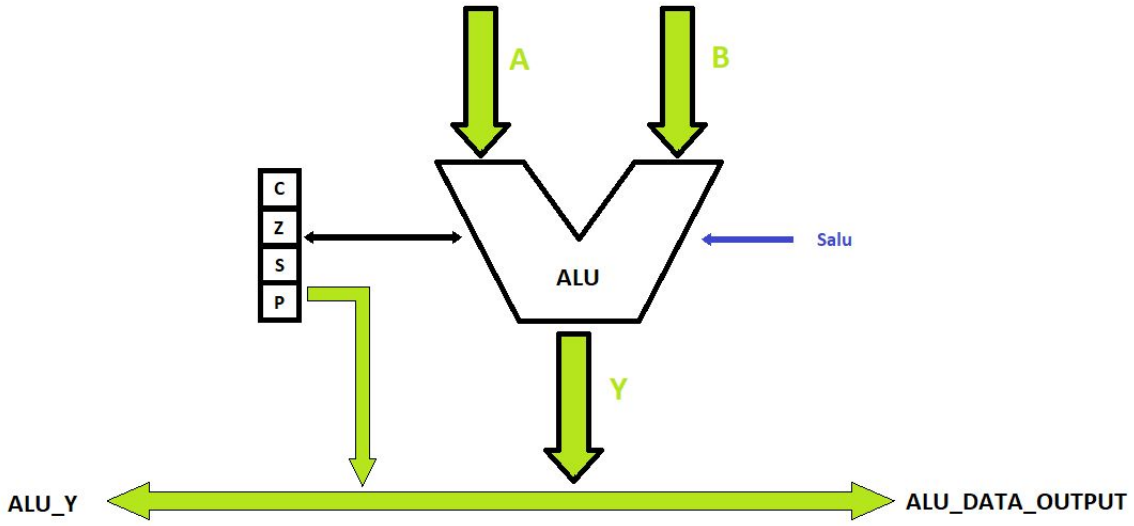
ALU

ALU - (jednostka arytmetyczno-logiczna) jest układem cyfrowym, służącym do wykonywania operacji arytmetycznych (takich jak dodawanie, odejmowanie itp.), operacji logicznych pomiędzy dwiema liczbami (np. XOR, Cmp) oraz operacji jednoargumentowych, takich jak przesunięcie bitów, negacja. ALU jest podstawowym blokiem centralnej jednostki obliczeniowej komputera.

Operacje:

Salu		Operacja
0	00000	Przepisanie wejścia A na wyjście Y
1	00001	Przepisanie wejścia B na wyjście Y
2	00010	dodawanie
3	00011	odejmowanie
4	00100	operacja OR
5	00101	operacja AND
6	00110	operacja XOR
7	00111	operacja XNOR
8	01000	negacja B
9	01001	negacja A
10	01010	odwrócenie znaku wartości podanej na B
11	01011	odwrócenie znaku wartości podanej na A
12	01100	wyzerowanie wyjścia Y
13	01101	suma wejść z przeniesieniem
14	01110	różnica wejść z przeniesieniem
15	01111	inkrementacja wartości wejścia A
16	10000	inkrementacja wartości wejścia B
17	10001	dekrementacja wartości wejścia A
18	10010	inkrementacja wartości wejścia B
19	10011	przesunięcie bitowe w prawo
20	10100	przesunięcie bitowe w lewo
21	10101	porównanie dwóch wejść ze sobą
22	10110	RPL8 (zamiana 4 starszych bitów z 4 młodszymi bitami wartości 8-bitowej)
23	10111	RPL4 (zamiana 2 starszych bitów z 2 młodszymi bitami wartości 4-bitowej)

Schemat



Flagi:

C - flaga przeniesienia (1 gdy wynik przekroczył dostępny zakres)
Z - flaga zera (1 dla wyniku równego zero)
S - flaga znaku (1 dla liczby ujemnej)
P - flaga parzystości (1 dla nieparzystej liczby jedynek)

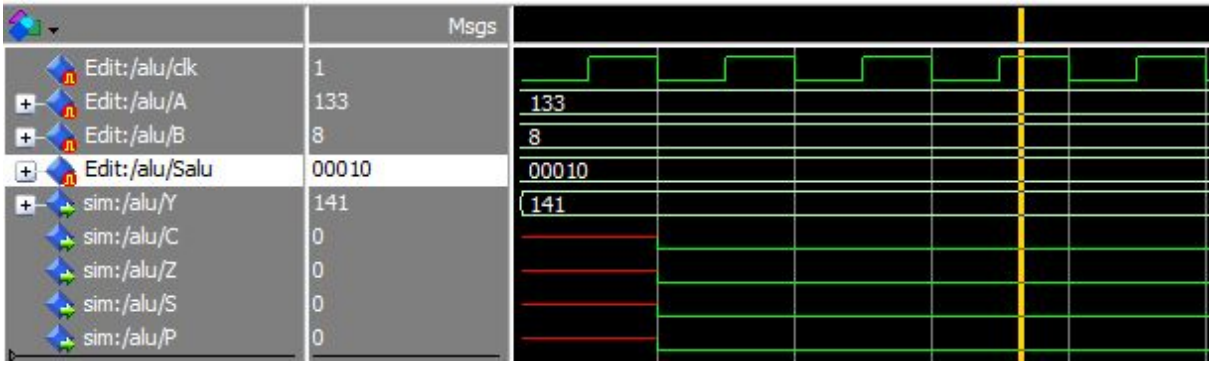
Sygnały

clk - sygnał zegarowy (do celów synchronizacji)
A - wejście pierwszego argumentu
B - wejście drugiego argumentu
Salu - sygnał sterujący pracą ALU
Y - wyjście na które przekazywany jest wynik działania ALU
C, Z, S, P - flagi

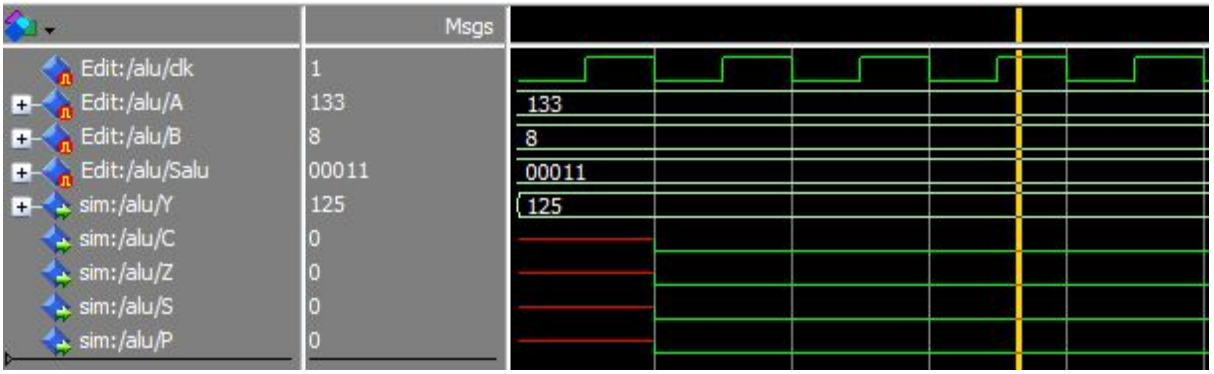
Testowanie

1. Przepisanie wejścia A na wyjście Y

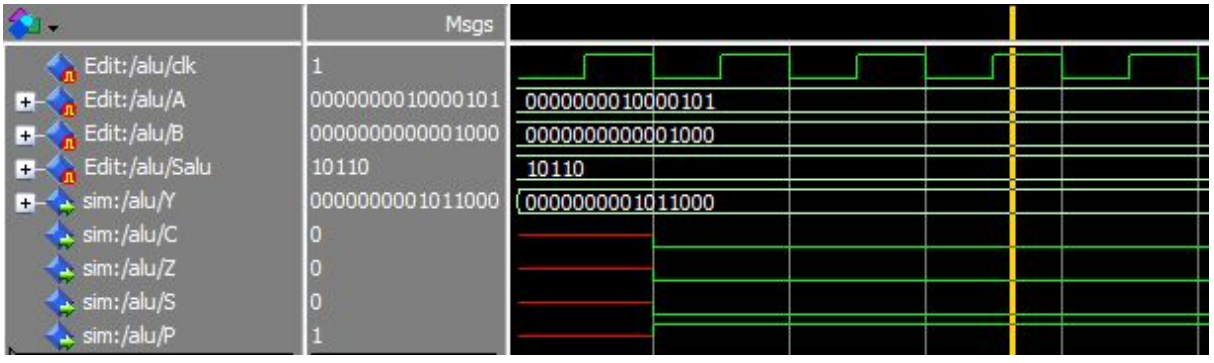
2. Dodawanie



3. Odejmowanie



4. Zamiana 4 starszych bitów z 4 młodszymi



Przetestowane zostały wybrane operacje arytmetyczne i logiczne z puli dostępnych operacji ALU. Wyniki operacji są widoczne jako sygnał Y na powyższych zrzutach ekranu. Wszystkie operacje generują oczekiwane wyniki.

Rejestry

Rejestry procesora – komórki pamięci o niewielkich rozmiarach umieszczone wewnątrz procesora i służące do przechowywania tymczasowych wyników obliczeń, adresów komórek w pamięci RAM itd. Większość procesorów przeprowadza działania wyłącznie korzystając z wewnętrznych rejestrów, kopiując do nich dane z pamięci i po zakończeniu obliczeń odsyłając wynik do pamięci.Rejestry procesora stanowią najwyższy szczebel w hierarchii pamięci, będąc najszybszym z rodzajów pamięci komputera, zarazem najdroższą w produkcji, a co za tym idzie – o najmniejszej pojemności. Realizowane zazwyczaj za pomocą przerzutników dwustanowych, z reguły jako tablica rejestrów.

Użyte rejestry:

Rejestry użytkownika:

- **A, B, C, D, E, F** - 16-bitowe rejestry ogólnego przeznaczenia

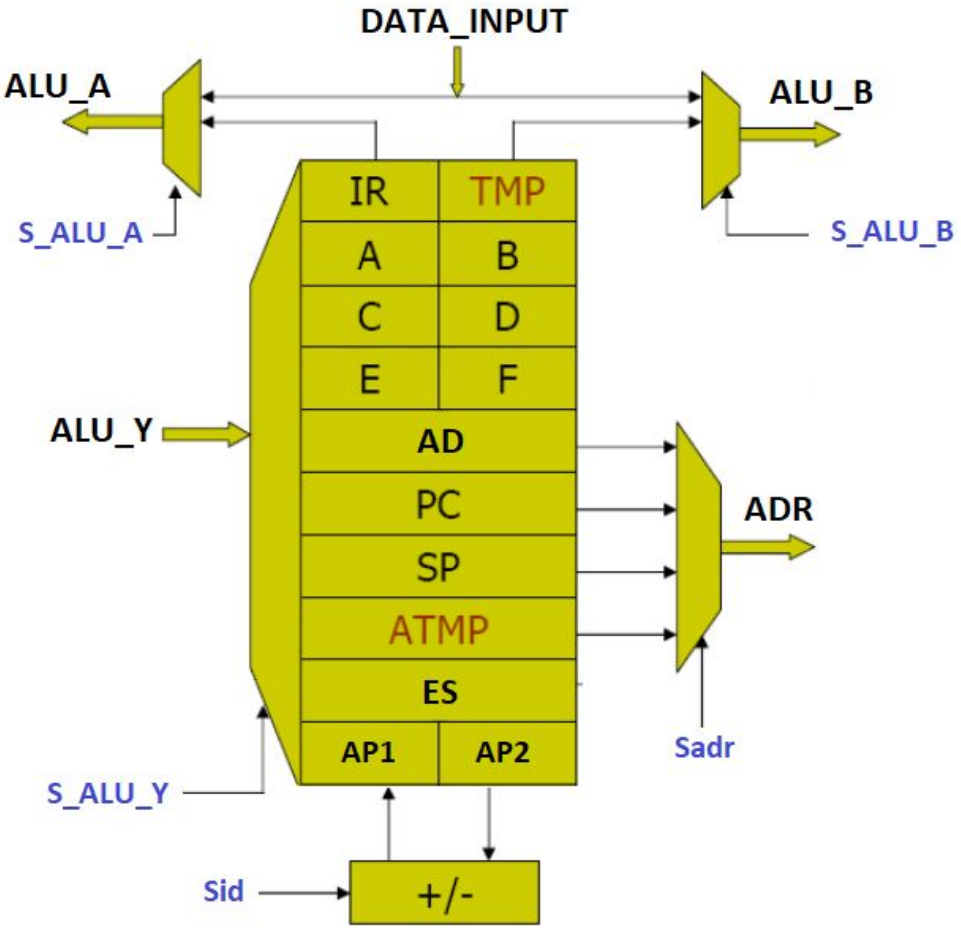
Rejestry procesora:

- **IR -** 16-bitowy rejestr przechowujący pojedynczy rozkaz procesora
- **PC -** 16-bitowy rejestr zawierający adres następnego rozkazu do wykonania (licznik rozkazów)
- **SP -** 16-bitowy rejestr zawierający adres wolnego pierwszego, wolnego miejsca w stosie
- **ES -** 16-bitowy rejestr zawierający adres segmentu
- **AD -** 16-bitowy rejestr przechowujący adres
- **AP1, AP2 -** dwa 8-bitowe rejestry przechowujące adres przemieszczenia
- **MBR -** 16-bitowy rejestr układu współpracy z pamięcią, służący jako bufor dla danych zapisywanych i odczytywanych z pamięci RAM
- **MAR -** 16-bitowy rejestr układu współpracy z pamięcią, służący jako bufor dla adresów pamięci RAM spod których dane mają zostać odczytane / zapisane

Rejestry pomocnicze (niewidoczne):

- **TMP -** 16-bitowy rejestr przechowujący
- **ATMP -** 16-bitowy rejestr przechowujący

Schemat



Sygnały

clk - sygnał zegarowy (do celów synchronizacji)
DATA_INPUT - dane przekazywane z zewnątrz (z modułu współpracy z pamięcią) [DI]
ALU_Y - dane przekazywane z wyjścia Y ALU do modułu rejestrów [BA]
ALU_A - dane przekazywane na wejście A jednostki ALU [BB]
ALU_B - dane przekazywane na wejście B jednostki ALU [BC]
ADR - adres przekazywany do modułu współpracy z pamięcią [A]
reset - sygnał resetu

S_ALU_A - wskazuje rejestr spod którego mają zostać odczytane dane i przekazane do ALU_A [SBB]

S_ALU_A		Funkcja
0	0000	ALU_A = DATA_INPUT
1	0001	ALU_A = TMP
2	0011	ALU_A = A
3	0100	ALU_A = B
4	0101	ALU_A = C
5	0110	ALU_A = D
6	0111	ALU_A = E
7	1000	ALU_A = F
8	1001	ALU_A (15 - 8) = AP1 ALU_A (7 - 0) = AP2
9	1010	ALU_A = ES
10	1011	ALU_A = IR and "0000000000011111"

S_ALU_B - wskazuje rejestr spod którego mają zostać odczytane dane i przekazane do ALU_A [SBC]

S_ALU_B		Funkcja
0	0000	ALU_B = DATA_INPUT
1	0001	ALU_B = TMP
2	0011	ALU_B = A
3	0100	ALU_B = B
4	0101	ALU_B = C
5	0110	ALU_B = D
6	0111	ALU_B = E
7	1000	ALU_B = F
8	1001	ALU_B (15 - 8) = AP1 ALU_B (7 - 0) = AP2
9	1010	ALU_B = ES
10	1011	ALU_B = IR and "0000000000011111"

S_ALU_Y : in signed (3 downto 0); -- wskazuje rejestr do którego mają zostać zapisane dane z ALU_Y [SBA]

S_ALU_Y		Funkcja
0	0000	IR = ALU_Y
1	0001	TMP = ALU_Y
2	0011	A = ALU_Y
3	0100	B = ALU_Y
4	0101	C = ALU_Y
5	0110	D = ALU_Y
6	0111	E = ALU_Y
7	1000	F = ALU_Y
8	1001	ATMP = ALU_Y
9	1010	AP1 = ALU_Y (15 - 8) AP2 = ALU_Y (7 - 0)
10	1011	ES = ALU_Y

Sid - kod operacji inkrementacji lub dekrementacji zawartości wybranych rejestrów [Sid]

Sid		Funkcja
0	000	brak akcji
1	001	PC++
2	011	SP++
3	100	SP--
4	101	AD++
5	110	AD--

Sadr - wskazuje który rejestr ma zostać wyprowadzony na wyjście ADR [SA]

Sadr		Funkcja
0	000	ADR = AD
1	001	ADR = PC
2	011	ADR = SP
3	100	ADR = ATMP
4	101	ADR = IR

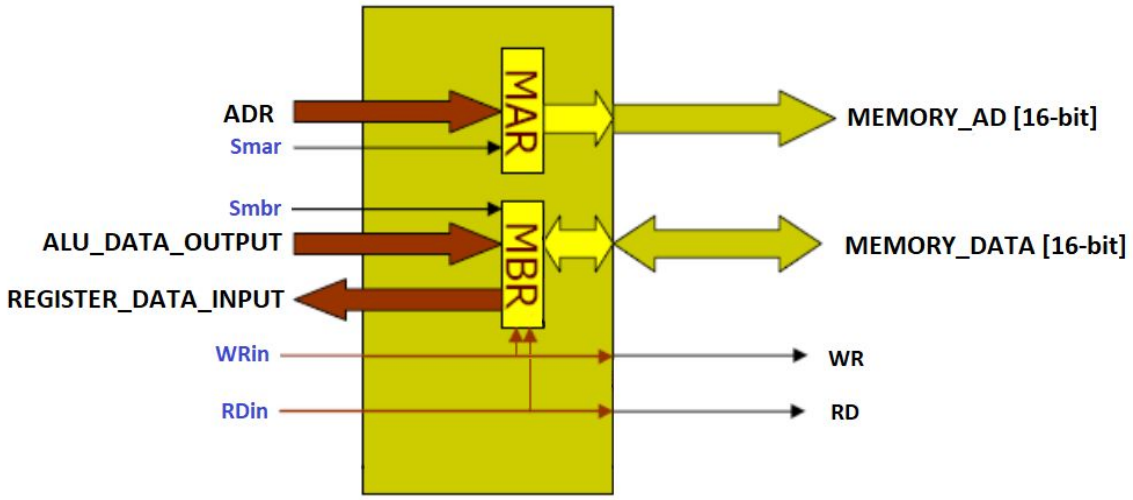
Układ współpracy z pamięcią

Jest to układ pośredniczący w wymianie danych pomiędzy procesorem a pamięcią RAM. Służy do buforowania przesyłanych informacji.

Składa się z dwóch 16-bitowych rejestrów:

- **MAR** - przechowuje adres komórki w pamięci RAM z której dane mają zostać odczytane / zapisane
- **MBR** - przechowuje dane przesyłane do / z pamięci RAM

Schemat



Sygnały

ADR - adres pobierany z rejestrów [A]
ALU_DATA_OUTPUT - dane pobierane z alu [DO]
MEMORY_AD - magistrala adresowa (magistrala do pamięci RAM) [AD]
REGISTER_DATA_INPUT - dane przekazywane do rejestrów [DI]
MEMORY_DATA - magistrala danych we/wy (magistrala do / z pamięci RAM) [D]
WR - sygnał sterujący zapisem do pamięci RAM [WR]
RD - sygnał sterujący odczytem z pamięci RAM [RD]

Smar - sygnał sterujący zapisem do rejestru MAR

Smar	Funkcja
0	pamiętanie
1	zapis ADR -> MAR

Smbr - sygnał sterujący zapisem do rejestru MAR

Smbr	Funkcja
0	pamiętanie
1	zapis ALU_DATA_OUTPUT -> MBR

WRin - sygnał sterujący zapisem do pamięci RAM

WRin	Funkcja
0	pamiętanie
1	zapis MBR -> RAM

RDin - sygnał sterujący odczytem z pamięci RAM i jednoczesnym zapisem do rejestru MBR

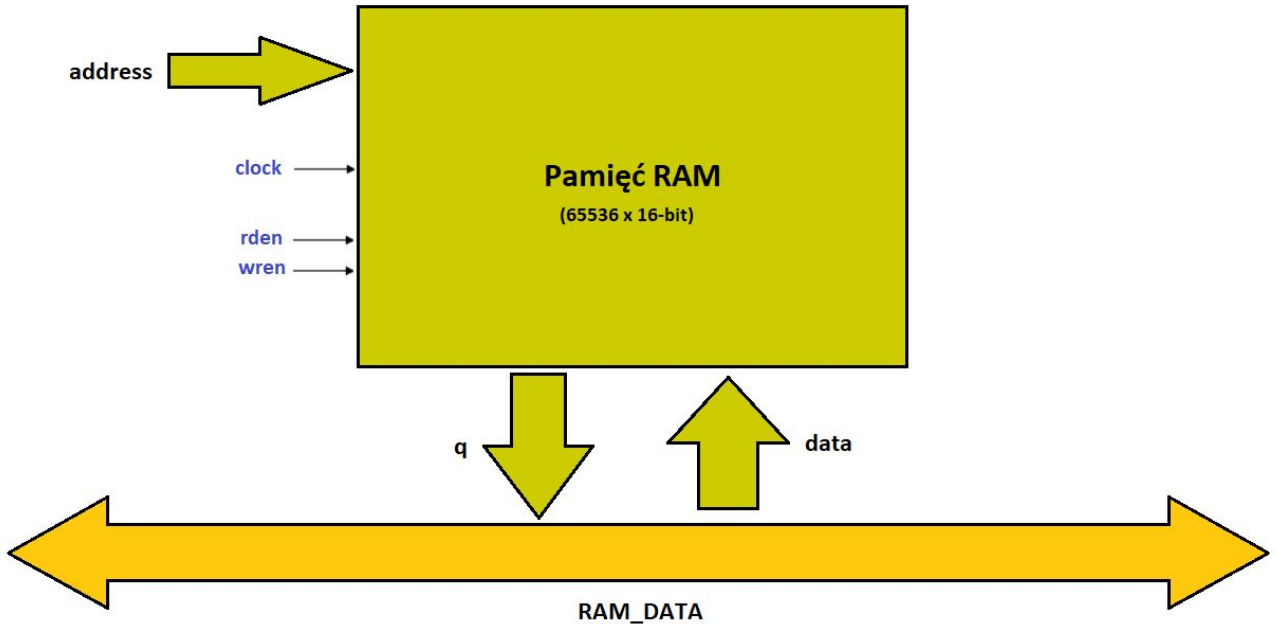
RDin	Funkcja
0	pamiętanie
1	odczyt RAM -> MBR

Pamięć RAM

RAM (od ang. random-access memory), pamięć o dostępie swobodnym – podstawowy rodzaj pamięci cyfrowej. W pamięci RAM przechowywane są aktualnie wykonywane programy i dane dla tych programów oraz wyniki ich pracy. W temperaturze pokojowej zawartość większości pamięci RAM jest tracona w czasie mniejszym niż sekunda po zaniku napięcia zasilania, niektóre typy wymagają także odświeżania, dlatego wyniki pracy programów wymagające trwałego przechowania muszą być zapisane na innym nośniku danych.

W naszym projekcie została utworzona pamięć ram składająca się z **65536 komórek pamięci**, z których każda przechowuje jedno **słowo 16-bitowe**. Dodatkowo w celu uniknięcia kolizji danych na szynie **RAM_DATA** wyjście **q** ustawiane jest w stan wysokiej impedancji (symbolicznie **"Z"**). W tym stanie wyjście zaczyna pracować jak wejście, w związku z czym dostosowuje się do sygnału zewnętrznego. Pozwala to na uniknięcie sytuacji w której dwa wyjścia ustawiają jeden sygnał zewnętrzny w różne wartości, przez co sygnał nie może osiągnąć żadnej z nich. Z taką sytuacją mamy do czynienia przy ustawianiu **rejestru MBR**, ponieważ posiada on dwa sygnały wejściowe (**DATA_INPUT** oraz **RAM_DATA**), które ustawiają jego zawartość w zależności od sygnałów sterujących. W związku z powyższym gdy jeden sygnał wprowadza dane do rejestru, wtedy drugi sygnał musi znajdować się w stanie wysokiej impedancji.

Schemat



Sygnały

address - adres pamięci RAM spod którego dane mają być odczytywane / do którego mają być zapisywane
clock - sygnał zegarowy
data - dane które mają być zapisane do pamięci RAM
q - dane odczytane z pamięci RAM

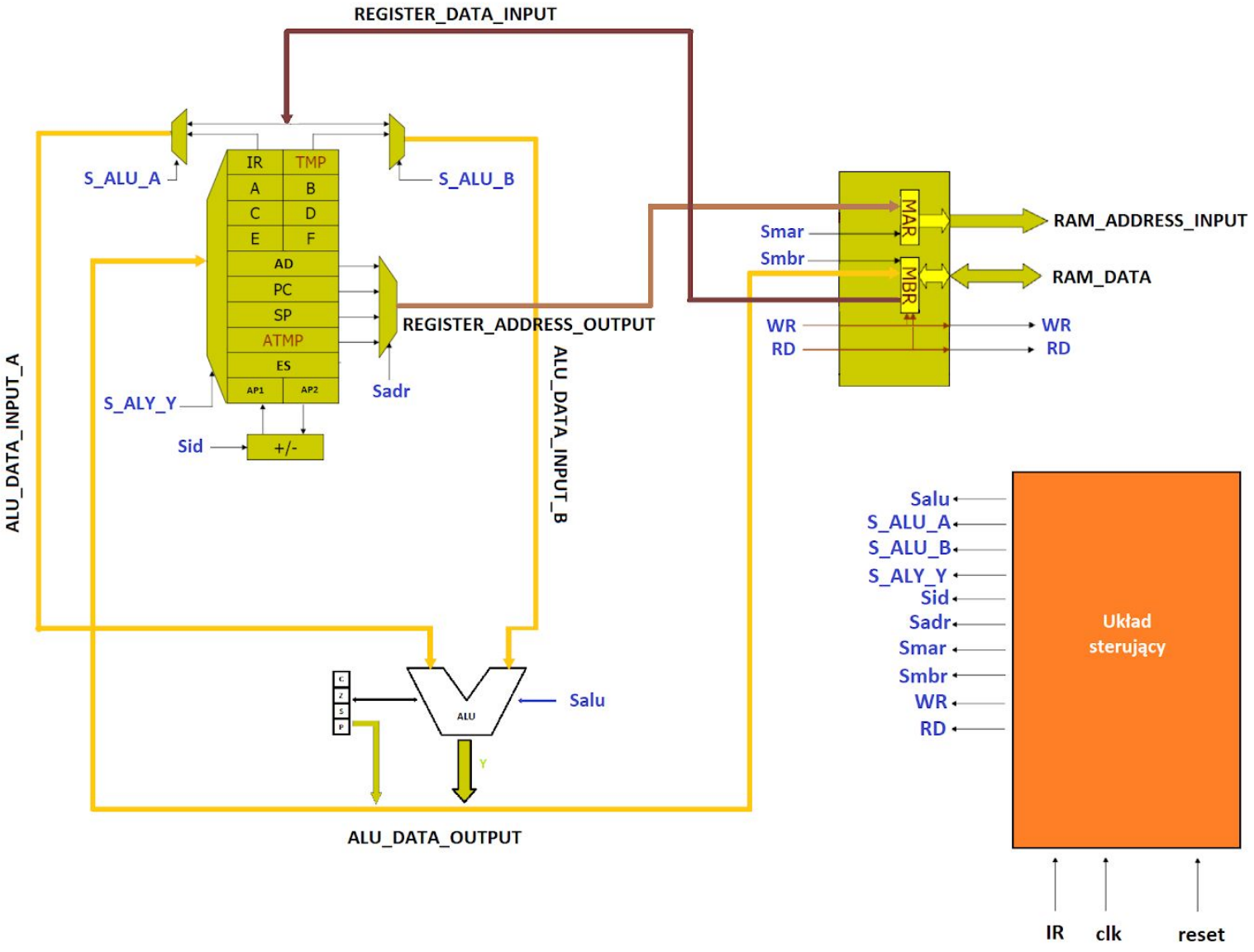
rden - sygnał sterujący odczytem z pamięci RAM

rden	Funkcja
0	nie odczytuj z pamięci RAM
1	odczyt RAM -> MBR

wren - sygnał sterujący zapisem do pamięci RAM

wren	Funkcja
0	nie zapisuj do pamięci RAM
1	zapis MBR -> RAM

Struktura procesora



Jednostka sterująca

Służy do sterowania pracą pozostałych układów. Z jednostki sterującej wyprowadzony jest szereg sygnałów sterujących, połączonych z odpowiednimi modułami procesora (takimi jak ALU, rejestry, układ współpracy z pamięcią). W jednostce sterującej pobierane są i dekodowane poszczególne rozkazy procesora umieszczone w rejestrze rozkazów IR. Na podstawie zdekodowanej informacji procesor przechodzi w odpowiedni stan, w którym ustawiane są sygnały sterujące charakteryzujące dany rozkaz.

Struktura rozkazu

1001110000100011

Kolejno od lewej:

2 bity - ilość argumentów
3 bity - kod operacji
1 bit - typ argumentów
10 bitów - do wykorzystania w zależności od wykonywanej operacji

Dostępne rozkazy (z przykładami)

Kod rozkazu	Opis rozkazu
Rozkazy jednoargumentowe	
0100000000000RRRR	PUSH rej
0100100000000RRRR	POP rej
0101000000000000	RPL8 adr
0101010000000RRRR	RPL8 rej
0101100000000000	RPL4 adr
0101110000000000	RPL4 rej
Rozkazy dwuargumentowe	
1000000000000RRRR AAAAAAAAAAAAAAAA	MOV adr rej
1000100000000RRRR AAAAAAAAAAAAAAAA	ADD adr rej
1001000000000RRRR AAAAAAAAAAAAAAAA	SUB adr rej

1001100000000RRRR AAAAAAAAAAAAAAAA	CMP rej adr
10011100ZZZZRRRR	CMP rej1 rej2 (w rozkazie rej1 - Z, rej2 - R)
10100000CCCCCRRRR	XNOR rej st
10100100ZZZZRRRR	XNOR rej1 rej2 (w rozkazie rej1 - Z, rej2 - R)

LEGENDA:

R - oznacza bity na których przechowywany jest kod rejestru podawanego jako argument
Z - oznacza bity na których przechowywany jest kod rejestru podawanego jako argument
A - oznacza bity na których przechowywany jest adres komórki pamięci RAM (podawany jest w kolejnym rozkazie)
C - oznacza bity których wartość używana jest jako stała wartość używana w operacji

OPIS STANÓW DLA POSZCZEGÓLNYCH ROZKAZÓW:

FETCH	
nazwa stanu	**opis**
fetch	wpisuje zawartość rejestru PC do rejestru MAR
fetch2	odczytuje zawartość komórki pamięci RAM spod adresu wskazywanego przez MAR, a następnie umieszcza ją w IR

DECODE	
nazwa stanu	**opis**
decode	analizuje rozkaz przechowywany w rejestrze IR i ustawia procesor w odpowiedni stan

PUSH	
nazwa stanu	**opis**
push	wkłada wartość umieszczoną w rejestrze na szczyt stosu, a następnie zmniejsza wskaźnik szczytu stosu

POP	
nazwa stanu	**opis**
pop	zwiększa wskaźnik stosu o 1 (SP++)
pop2	ustawia rejestr MAR na wartość spod wskaźnika szczytu stosu
pop3	odczytuje wartość ze szczytu stosu i wprowadza odczytaną wartość do wskazanego rejestru

RPL8 adr	
nazwa stanu	**opis**
rpl8A	wpisuje zawartość rejestru PC do rejestru MAR
rpl8A_2	odczytuje zawartość komórki pamięci RAM spod adresu wskazywanego przez MAR, a następnie umieszcza ją w IR
rpl8A_3	wpisuje zawartość rejestru IR do rejestru MAR
rpl8A_4	odczytuje zawartość komórki pamięci RAM spod adresu wskazywanego przez MAR, a następnie wyprowadza ją na wejście ALU_A oraz oblicza wynik operacji RPL8 i wysterowuje wyjście ALU_Y
rpl8A_5	zapisuje wyjście ALU_Y do komórki pamięci RAM, spod której została uprzednio odczytana wartość

RPL8 rej	
nazwa stanu	**opis**
rpl8R	odczytuje podany w rozkazie rejestr i przesyła jego zawartość do ALU_A, wykonuje operacje RPL8, przesyła wynik operacji do rejestru, z którego wartość została wcześniej pobrana

RPL4 adr	
nazwa stanu	**opis**
rpl4A	wpisuje zawartość rejestru PC do rejestru MAR
rpl4A_2	odczytuje zawartość komórki pamięci RAM spod adresu wskazywanego przez MAR, a następnie umieszcza ją w IR
rpl4A_3	wpisuje zawartość rejestru IR do rejestru MAR
rpl4A_4	odczytuje zawartość komórki pamięci RAM spod adresu wskazywanego przez MAR, a następnie wyprowadza ją na wejście ALU_A oraz oblicza wynik operacji RPL4 i wysterowuje wyjście ALU_Y
rpl4A_5	zapisuje wyjście ALU_Y do komórki pamięci RAM, spod której została uprzednio odczytana wartość

RPL4 rej	
nazwa stanu	**opis**
rpl4R	odczytuje podany w rozkazie rejestr i przesyła jego zawartość do ALU_A, wykonuje operacje RPL4, przesyła wynik operacji do rejestru, z którego wartość została wcześniej pobrana

MOV adr rej	
nazwa stanu	**opis**
mov	zapisuje zawartość wskazanego w rozkazie rejestru do TMP. Następnie wprowadza zawartość rejestru PC do rejestru MAR, pobiera pierwszy argument rozkazu z pamięci RAM oraz zwiększa licznik rozkazów PC o 1
mov2	wprowadza odczytany adres do rejestru rozkazów IR
mov3	zapisuje zawartość TMP do komórki pamięci RAM o adresie podanym w rejestrze IR

ADD adr rej	
nazwa stanu	**opis**
add	zapisuje zawartość wskazanego w rozkazie rejestru do TMP. Następnie wprowadza zawartość rejestru PC do rejestru MAR, pobiera pierwszy argument rozkazu z pamięci RAM oraz zwiększa licznik rozkazów PC o 1
add_2	odczytuje zawartość komórki pamięci RAM spod adresu wskazywanego przez MAR, a następnie umieszcza ją w IR
add_3	wpisuje zawartość rejestru IR do rejestru MAR
add_4	odczytuje zawartość komórki pamięci RAM spod adresu wskazywanego przez MAR, a następnie wyprowadza ją na wejście ALU_A. W tym samym czasie na wejście ALU_B wprowadza zawartość rejestru TMP, po czym wykonuje operację sumy i wysterowuje wyjście ALU_Y
add_5	zapisuje wyjście ALU_Y do komórki pamięci RAM, spod której została uprzednio odczytana wartość

SUB adr rej	
nazwa stanu	**opis**
sub	zapisuje zawartość wskazanego w rozkazie rejestru do TMP. Następnie wprowadza zawartość rejestru PC do rejestru MAR, pobiera pierwszy argument rozkazu z pamięci RAM oraz zwiększa licznik rozkazów PC o 1
sub_2	odczytuje zawartość komórki pamięci RAM spod adresu wskazywanego przez MAR, a następnie umieszcza ją w IR
sub_3	wpisuje zawartość rejestru IR do rejestru MAR
sub_4	odczytuje zawartość komórki pamięci RAM spod adresu wskazywanego przez MAR, a następnie wyprowadza ją na wejście ALU_A. W tym samym czasie na wejście ALU_B wprowadza zawartość rejestru TMP, po czym wykonuje operację różnicy i wysterowuje wyjście ALU_Y
sub_5	zapisuje wyjście ALU_Y do komórki pamięci RAM, spod której została uprzednio odczytana wartość

CMP rej adr	
nazwa stanu	**opis**
cmpA	zapisuje zawartość wskazanego w rozkazie rejestru do TMP. Następnie wprowadza zawartość rejestru PC do rejestru MAR, pobiera pierwszy argument rozkazu z pamięci RAM oraz zwiększa licznik rozkazów PC o 1
cmpA_2	odczytuje zawartość komórki pamięci RAM spod adresu wskazywanego przez MAR, a następnie umieszcza ją w IR
cmpA_3	wpisuje zawartość rejestru IR do rejestru MAR
cmpA_4	odczytuje zawartość komórki pamięci RAM spod adresu wskazywanego przez MAR, a następnie wyprowadza ją na wejście ALU_A. W tym samym czasie na wejście ALU_B wprowadza zawartość rejestru TMP, po czym wykonuje operację porównania i wysterowuje wyjście ALU_Y oraz flagę Z, w której znajduje się wynik (Z = 1 dla różnych wartości, Z=0 dla takich samych wartości)

CMP rej rej	
nazwa stanu	**opis**
cmpR	wprowadza zawartość rejestru będącego pierwszym argumentem na wejście ALU_A oraz zawartość rejestru będącego drugim argumentem na wejście ALU_B, po czym wykonuje operację porównania i wysterowuje wyjście ALU_Y. W tym samym stanie wprowadza wyjście ALU_Y oraz flagę Z, w której znajduje się wynik

XNOR rej st	
nazwa stanu	**opis**
oxnorC	wprowadza zawartość rejestru będącego pierwszym argumentem na wejście ALU_A oraz stałą na wejście ALU_B, po czym wykonuje operację XNOR i wysterowuje wyjście ALU_Y. W tym samym stanie wprowadza wyjście ALU_Y do rejestru, z którego pobrana została wartość

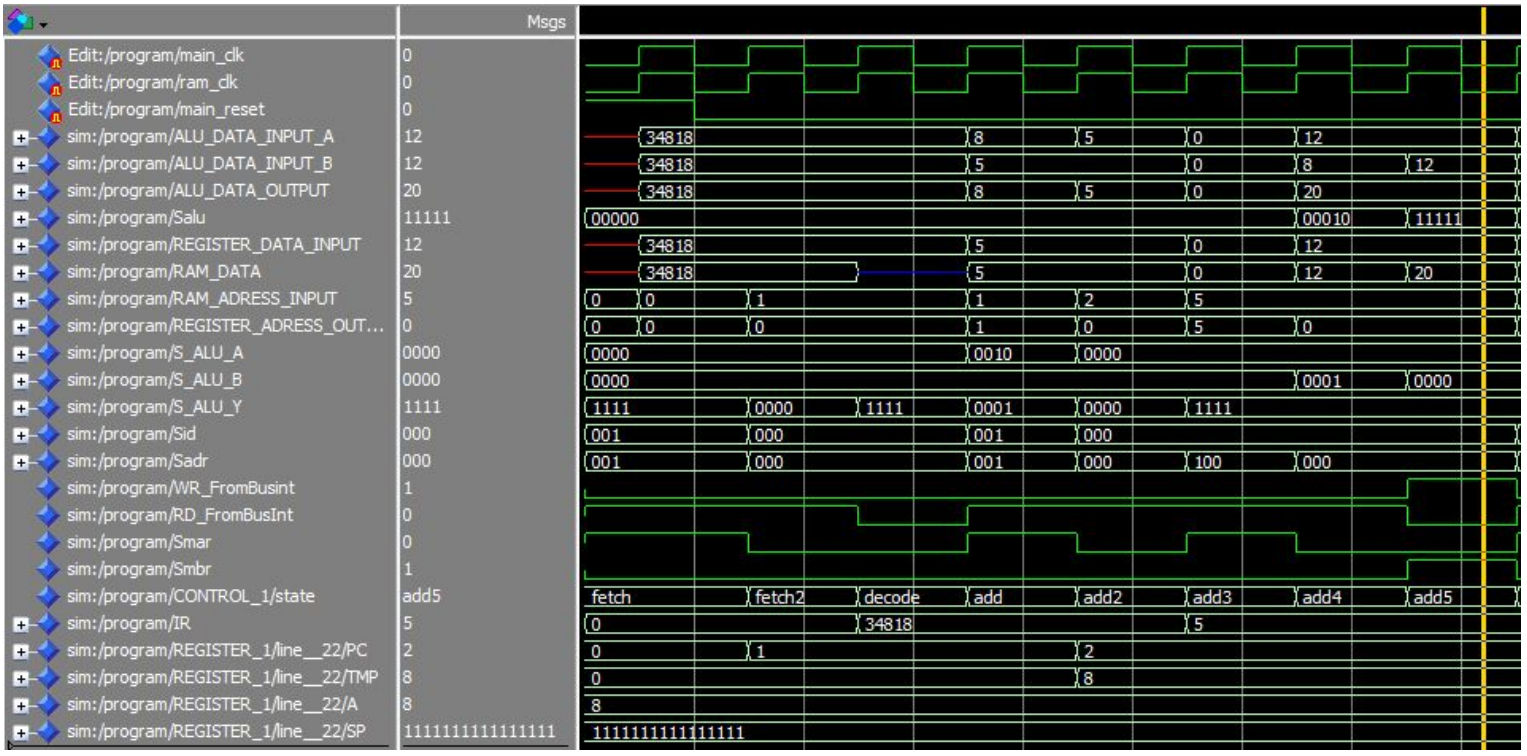
XNOR rej rej	
nazwa stanu	**opis**
oxnorR	wprowadza zawartość rejestru będącego pierwszym argumentem na wejście ALU_A oraz zawartość rejestru będącego drugim argumentem na wejście ALU_B, po czym wykonuje operację XNOR i wysterowuje wyjście ALU_Y. W tym samym stanie wprowadza wyjście ALU_Y do rejestru będącego pierwszym argumentem rozkazu

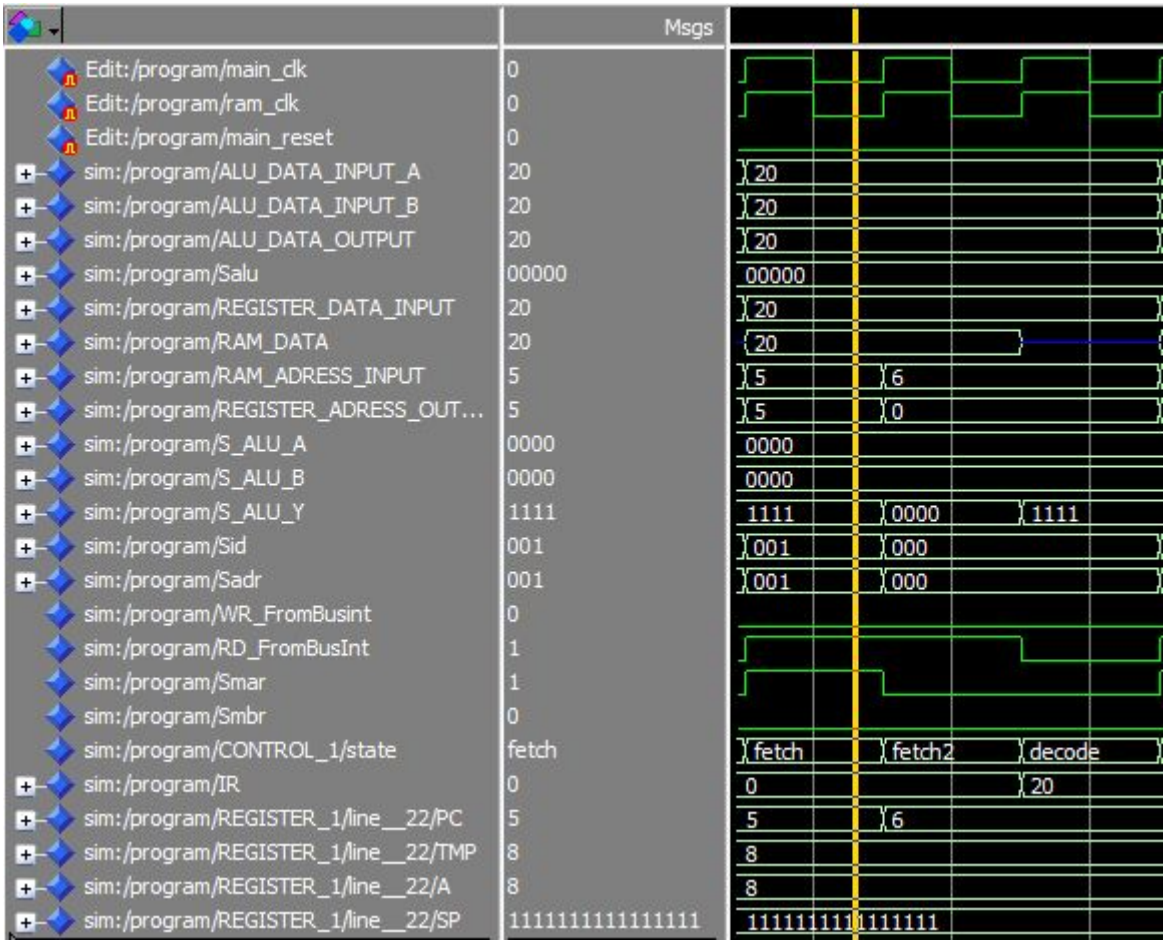
Symulacja działania procesora

W programie ModelSim zostały przetestowane wszystkie dostępne rozkazy procesora na dwa sposoby. Pierwszym z nich było przetestowanie każdego rozkazu z osobna, natomiast drugim sposobem było przetestowanie ciągu kilku rozkazów. Ze względu na konieczność utrzymania czytelności i spójności raportu poniżej umieściliśmy zrzuty ekranu z przebiegu testów każdego rozkazu z osobna.

ADD adr rej

Kod rozkazu: 1000100000000010
Symulacja:





Omówienie:

Do rejestru A została wprowadzona wartość 8.

Do komórki pamięci RAM o adresie 5 została wprowadzona wartość 12.

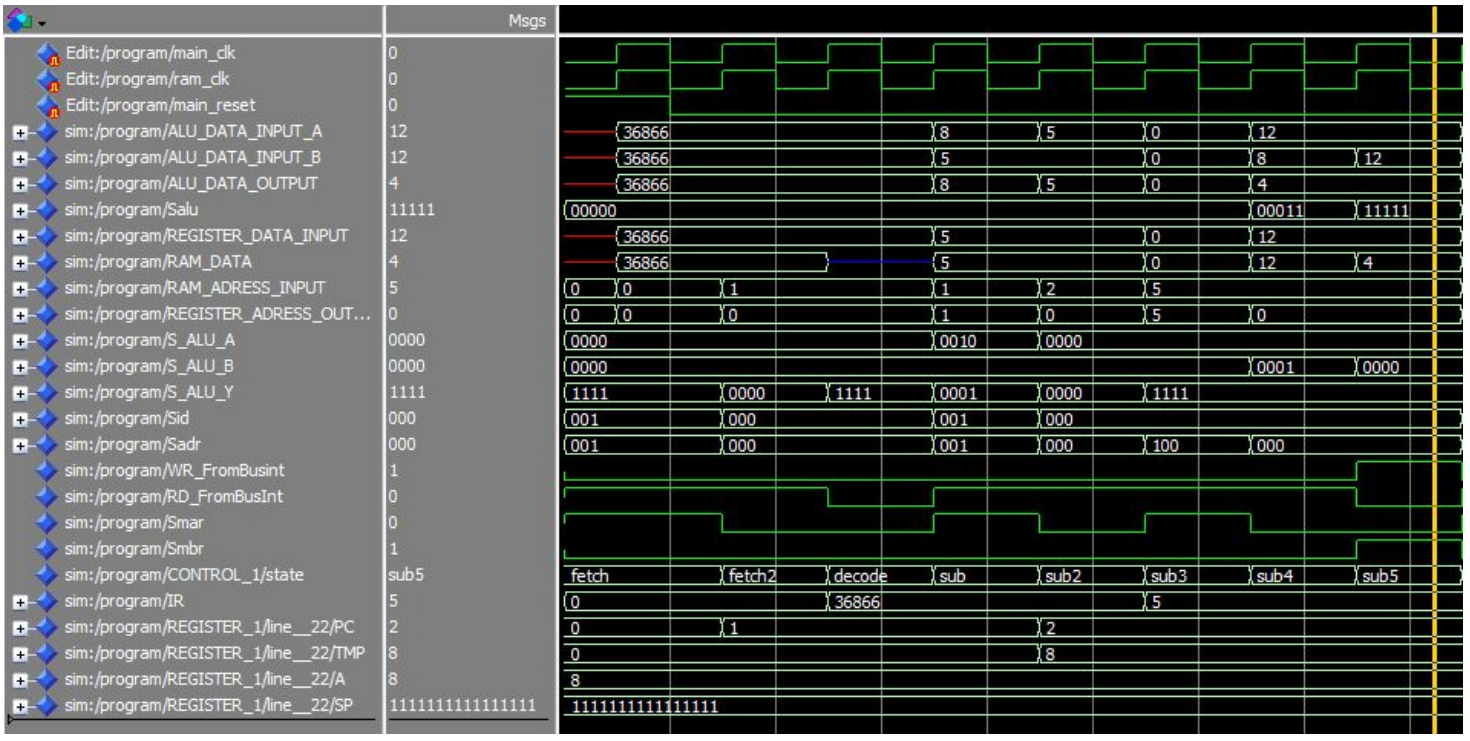
Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. W 4 najmłodszych bitach rozkazu został zawarty kod oznaczający z którego rejestru ma zostać odczytana wartość argumentu 2. Wartość spod odpowiedniego rejestru (w tym przypadku rejestru A) została przesłana do rejestru TMP. Następnie został pobrany kolejny rozkaz programu, będący adresem komórki pamięci RAM w której przechowywana jest wartość argumentu 1(w naszym przypadku jest to komórka pamięci o adresie 5). W kolejnym kroku pobierana jest wartość argumentu 1 spod wskazanej komórki pamięci i wprowadzana na wejście A jednostki arytmetyczno-logicznej. Równolegle, odczytywana jest zawartość rejestru TMP i zostaje ona wprowadzona na wejście B jednostki arytmetyczno-logicznej. Następnie zostaje wykonana operacja dodawania obu wartości i wynik zostaje przesłany do komórki pamięci RAM, z której został pobrany wcześniej argument 1.

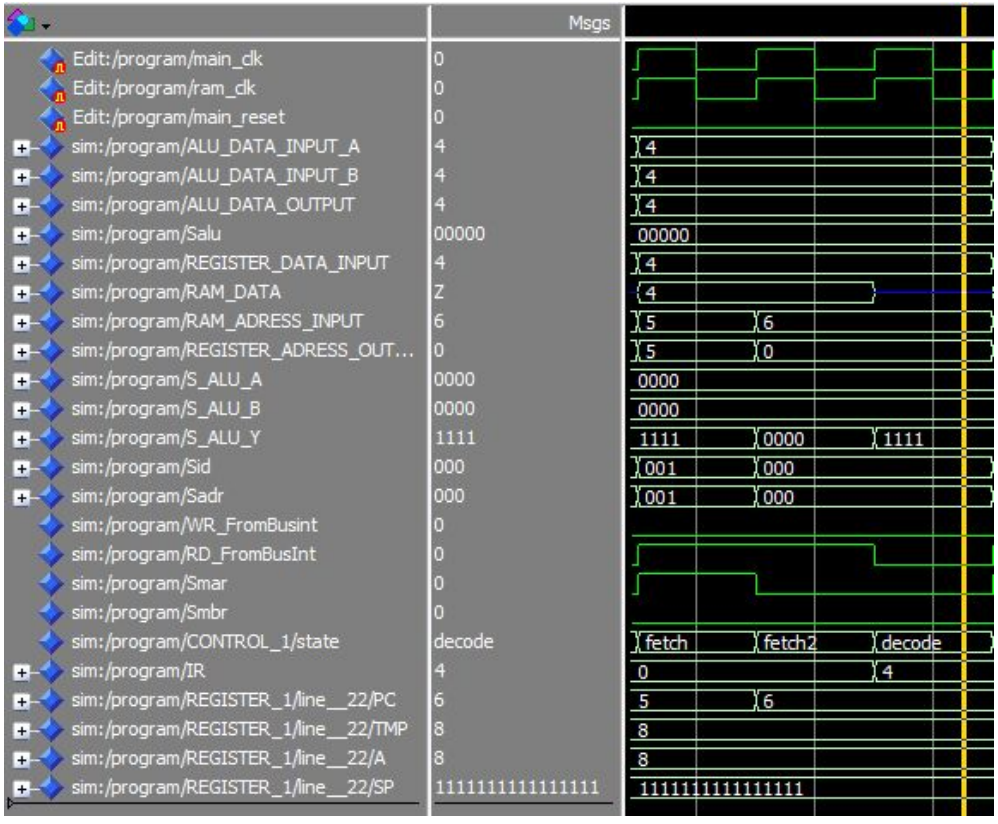
Jak widać na drugim zrzucie ekranu procesor odczytując zawartość komórki pamięci o adresie 5 rozkazem fetch (RAM_ADRESS_INPUT = 5) pobiera wartość 20, co oznacza poprawne wykonanie operacji dodawania (12 + 8 = 20).

SUB adr rej

Kod rozkazu: 1001000000000010

Symulacja:





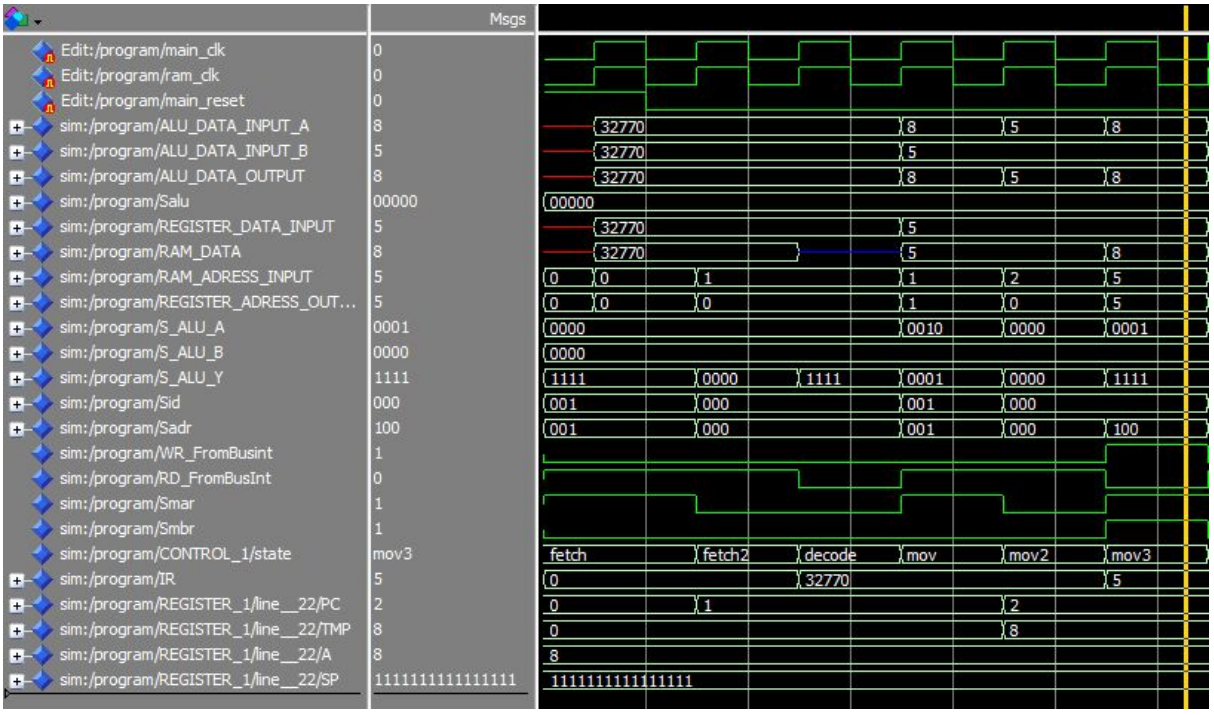
Omówienie:

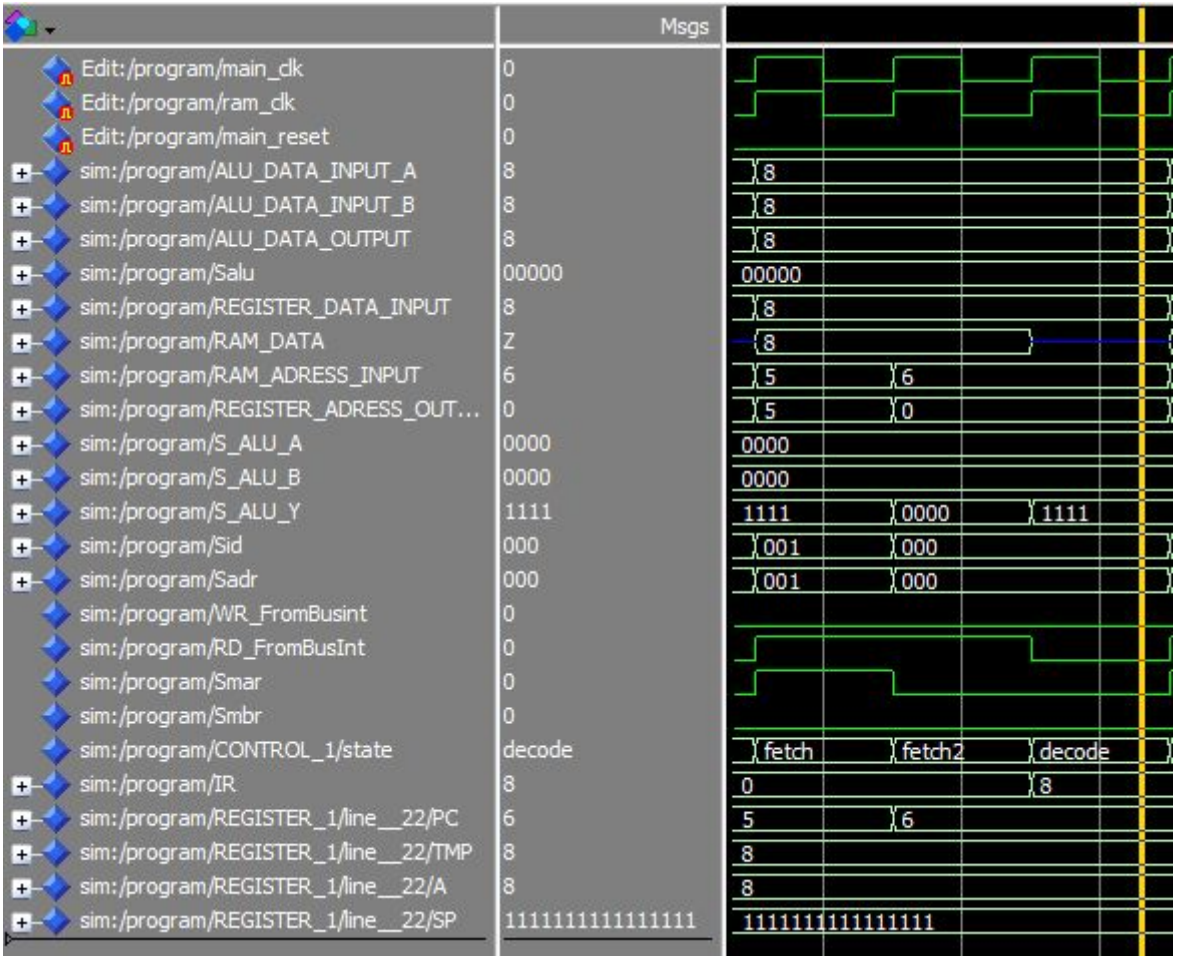
Do rejestru A została wprowadzona wartość 8.
Do komórki pamięci RAM o adresie 5 została wprowadzona wartość 12.
Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. W 4 najmłodszych bitach rozkazu został zawarty kod oznaczający z którego rejestru ma zostać odczytana wartość argumentu 2. Wartość spod odpowiedniego rejestru (w tym przypadku rejestru A) została przesłana do rejestru TMP. Następnie został pobrany kolejny rozkaz programu, będący adresem komórki pamięci RAM w której przechowywana jest wartość argumentu 1(w naszym przypadku jest to komórka pamięci o adresie 5). W kolejnym kroku pobierana jest wartość argumentu 1 spod wskazanej komórki pamięci i wprowadzana na wejście A jednostki arytmetyczno-logicznej. Równolegle, odczytywana jest zawartość rejestru TMP i zostaje ona wprowadzona na wejście B jednostki arytmetyczno-logicznej. Następnie zostaje wykonana operacja odejmowania obu wartości i wynik zostaje przesłany do komórki pamięci RAM, z której został pobrany wcześniej argument 1.
Jak widać na drugim zrzucie ekranu procesor odczytując zawartość komórki pamięci o adresie 5 rozkazem fetch (RAM_ADRESS_INPUT = 5) pobiera wartość 4, co oznacza poprawne wykonanie operacji odejmowania(12 - 8 = 4).

MOV adr rej

Kod rozkazu: 1000000000000010

Symulacja:



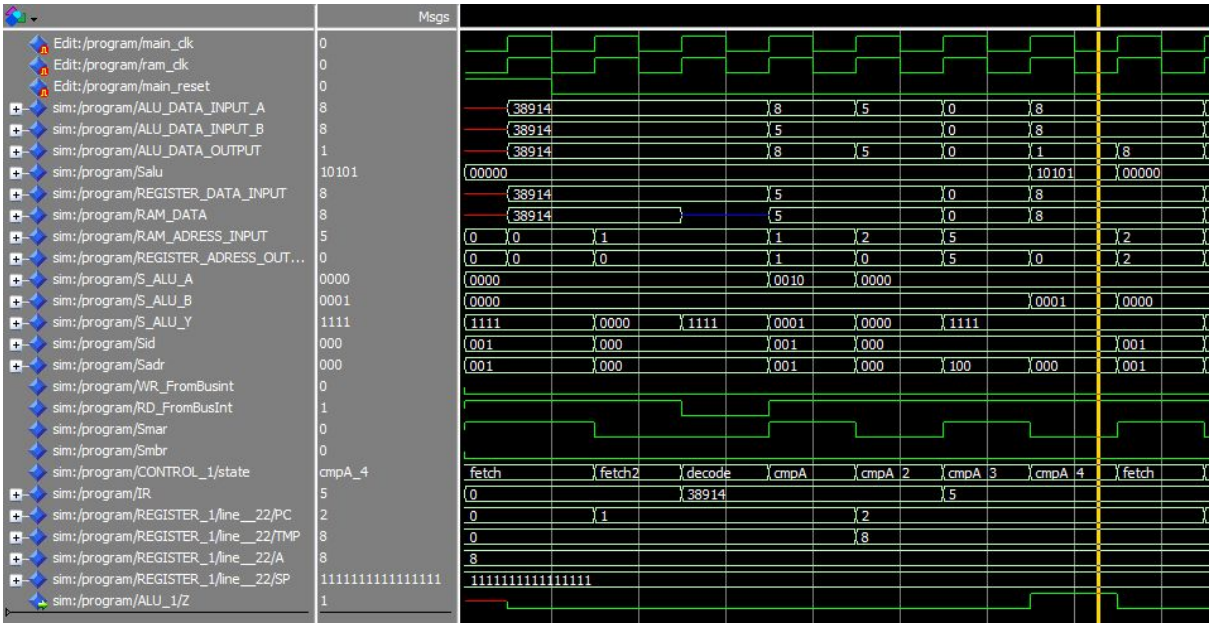


Omówienie:
Do rejestru A została wprowadzona wartość 8.
Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. W 4 najmłodszych bitach rozkazu został zawarty kod oznaczający z którego rejestru ma zostać odczytana wartość argumentu 2. Wartość spod odpowiedniego rejestru (w tym przypadku rejestru A) została przesłana do rejestru TMP. Następnie został pobrany kolejny rozkaz programu, będący adresem komórki pamięci RAM do której wpisana ma zostać wartość argumentu 1 (w naszym przypadku jest to komórka pamięci o adresie 5). W kolejnym kroku zawartość rejestru TMP wprowadzana na wejście A jednostki arytmetyczno-logicznej. Następnie zostaje wykonana operacja przeniesienia wejścia A na wyjście Y jednostki arytmetyczno-logicznej, a wynik zostaje przesłany do komórki pamięci RAM wskazywanej przez adres zawarty w rejestrze IR.
Jak widać na drugim zrzucie ekranu procesor odczytując zawartość komórki pamięci o adresie 5 rozkazem fetch (RAM_ADRESS_INPUT = 5) pobiera wartość 8, co oznacza poprawne wykonanie operacji przeniesienia(rejestr A przechowuje wartość 8).

CMP rej adr

Kod rozkazu: 1001100000000010

Symulacja:



Omówienie:

Do rejestru A została wprowadzona wartość 8.

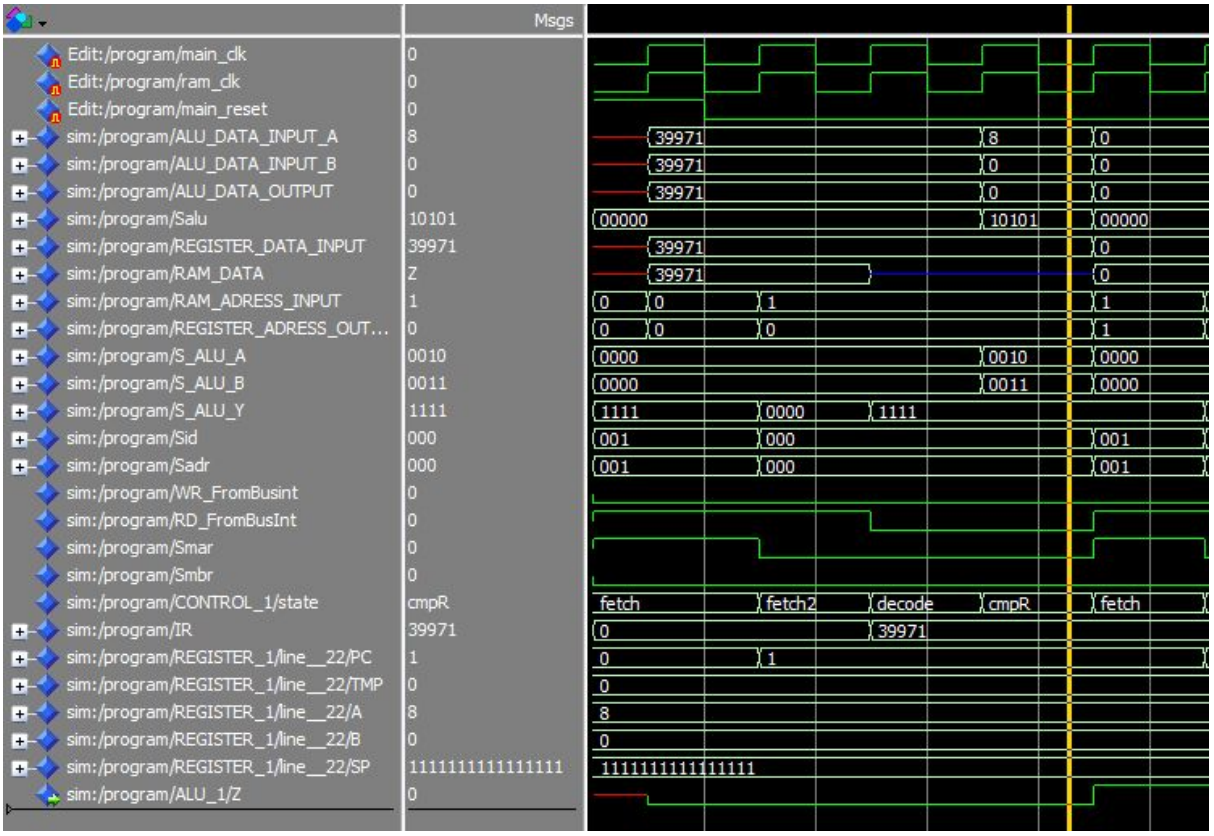
Do komórki pamięci RAM o adresie 5 została wprowadzona wartość 8.

Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. W 4 najmłodszych bitach rozkazu został zawarty kod oznaczający z którego rejestru ma zostać odczytana wartość argumentu 1. Wartość spod odpowiedniego rejestru (w tym przypadku rejestru A) została przesłana do rejestru TMP. Następnie został pobrany kolejny rozkaz programu, będący adresem komórki pamięci RAM w której przechowywana jest wartość argumentu 1(w naszym przypadku jest to komórka pamięci o adresie 5). W kolejnym kroku pobierana jest wartość argumentu 1 spod wskazanej komórki pamięci i wprowadzana na wejście A jednostki arytmetyczno-logicznej. Równolegle, odczytywana jest zawartość rejestru TMP i zostaje ona wprowadzona na wejście B jednostki arytmetyczno-logicznej. Następnie zostaje wykonana operacja porównania obu wartości. Ponieważ obie wartości są sobie równe na wyjście ALU podawana jest wartość 1 (ALU_DATA_OUTPUT = 1). Flaga Z oznaczająca wartość zero wyniku ALU ustawiana jest dopiero po przejściu ostatniego stanu operacji porównania. W naszym przypadku po ostatnim stanie (cmpA_4), flaga Z ustawiana jest w ‘0’, co widać na powyższym zrzucie.

CMP rej rej

Kod rozkazu: 1001110000100010

Symulacja:



Omówienie:

Do rejestru A została wprowadzona wartość 8.

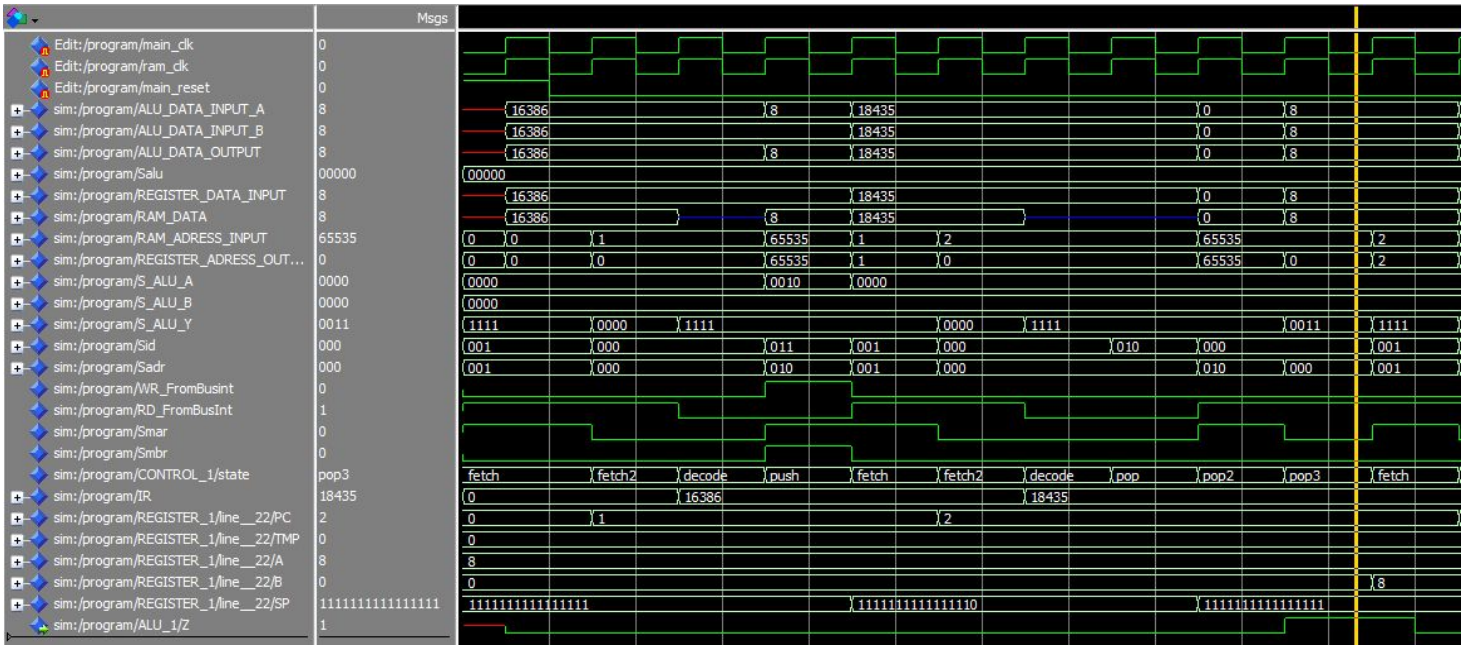
Do rejestru B została wprowadzona wartość 0.

Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. W bitach od 3 do 0 (tj. 4 najmłodsze bity) rozkazu został zawarty kod oznaczający z którego rejestru ma zostać odczytana wartość argumentu 1 (w naszym przypadku jest to rejestr A). W bitach od 7 do 4 rozkazu został zawarty kod oznaczający z którego rejestru ma zostać odczytana wartość argumentu 2 (w naszym przypadku jest to rejestr B). Następnie wartości obu wskazanych rejestrów są równolegle odczytywane i zostają odpowiednio przesłane na wejścia ALU (w naszym przypadku wartość rejestru A zostaje przesłana na wejście A ALU, a wartość rejestru B na wejście B ALU). Później zostaje wykonana operacja porównania obu wartości. Ponieważ obie wartości są różne na wyjście ALU podawana jest wartość 0 (ALU_DATA_OUTPUT = 0). Flaga Z oznaczająca wartość zero wyniku ALU ustawiana jest dopiero po przejściu stanu operacji porównania. W naszym przypadku po stanie (cmpR), flaga Z ustawiana jest w ‘1’, co widać na powyższym zrzucie ekranu.

PUSH rej

Kod rozkazu: 100000000000010

Symulacja:



Omówienie:

Do rejestru A została wprowadzona wartość 8.

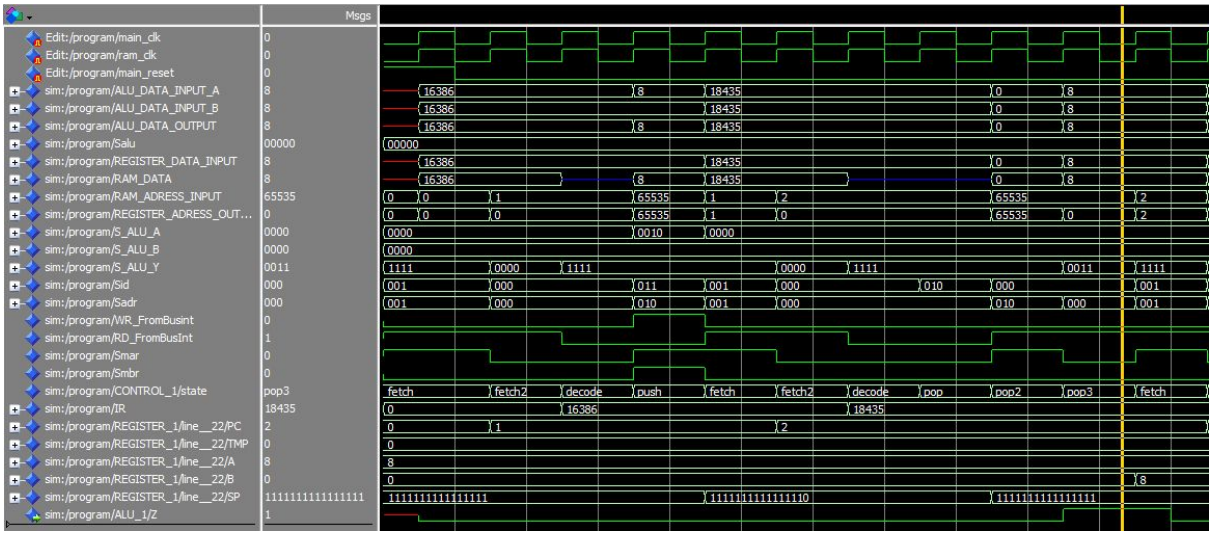
Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. W 4 najmłodszych bitach rozkazu został zawarty kod oznaczający z którego rejestru ma zostać odczytana wartość, która ma zostać włożona na szczyt stosu. Wartość spod odpowiedniego rejestru (w tym przypadku rejestru A) została przesłana na wejście A jednostki arytmetyczno-logicznej. Następnie zostaje wykonana operacja przeniesienia wejścia A na wyjście Y jednostki arytmetyczno-logicznej, a wynik zostaje przesłany do komórki pamięci RAM wskazywanej przez adres zawarty w rejestrze SP (rejestr przechowujący wskaźnik szczytu stosu).

Poprawność operacji możemy wywnioskować poprzez zastosowanie jej w parze z operacją pop. Operacja pop w tym przypadku ma za zadanie zdjąć wartość ze szczytu stosu i wpisać ją do rejestru B. Na powyższym zrzucie ekranu widać, że po zakończeniu wykonywania ostatniego stanu operacji pop rejestr B zmienia swoją zawartość na 8.

POP rej

Kod rozkazu: 100100000000011

Symulacja:



Omówienie:

Do rejestru A została wprowadzona wartość 8.

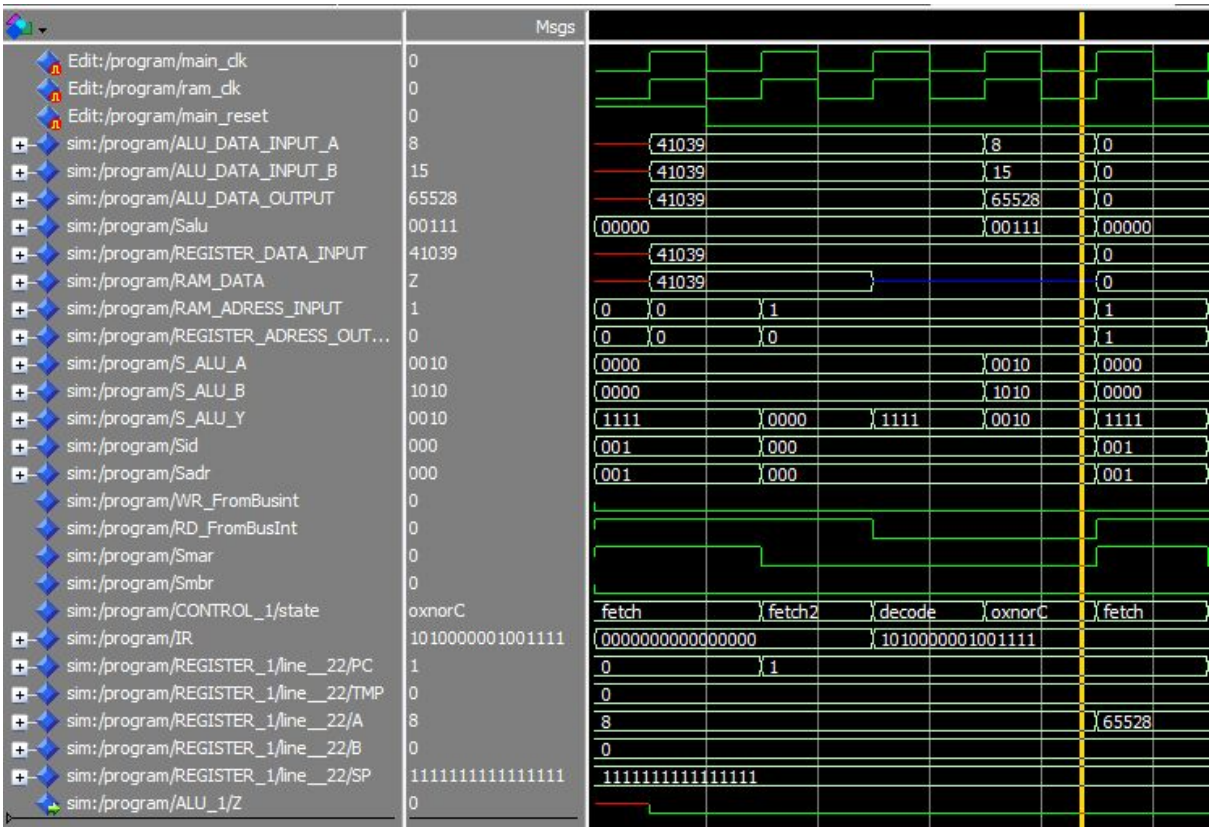
Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. W 4 najmłodszych bitach rozkazu został zawarty kod oznaczający do którego rejestru ma zostać zapisana wartość ze szczytu stosu. Program odczytuje zawartość komórki pamięci RAM wskazywanej przez adres zawarty w rejestrze SP zwiększony o 1. Wartość ta zostaje wpisana do podanego rejestru.

W naszym przypadku w celu sprawdzenia poprawności działania operacji pop, uprzednio została wykonana operacja PUSH wkładająca na szczyt stosu wartość 8. Operacja pop w tym przypadku ma za zadanie zdjąć wartość ze szczytu stosu i wpisać ją do rejestru B. Rejestr B po zakończeniu wykonywania operacji POP zmienił swoją zawartość na 8 (co widać na powyższym zrzucie).

XNOR rej st

Kod rozkazu: 1010000001001111

Symulacja:



Omówienie:

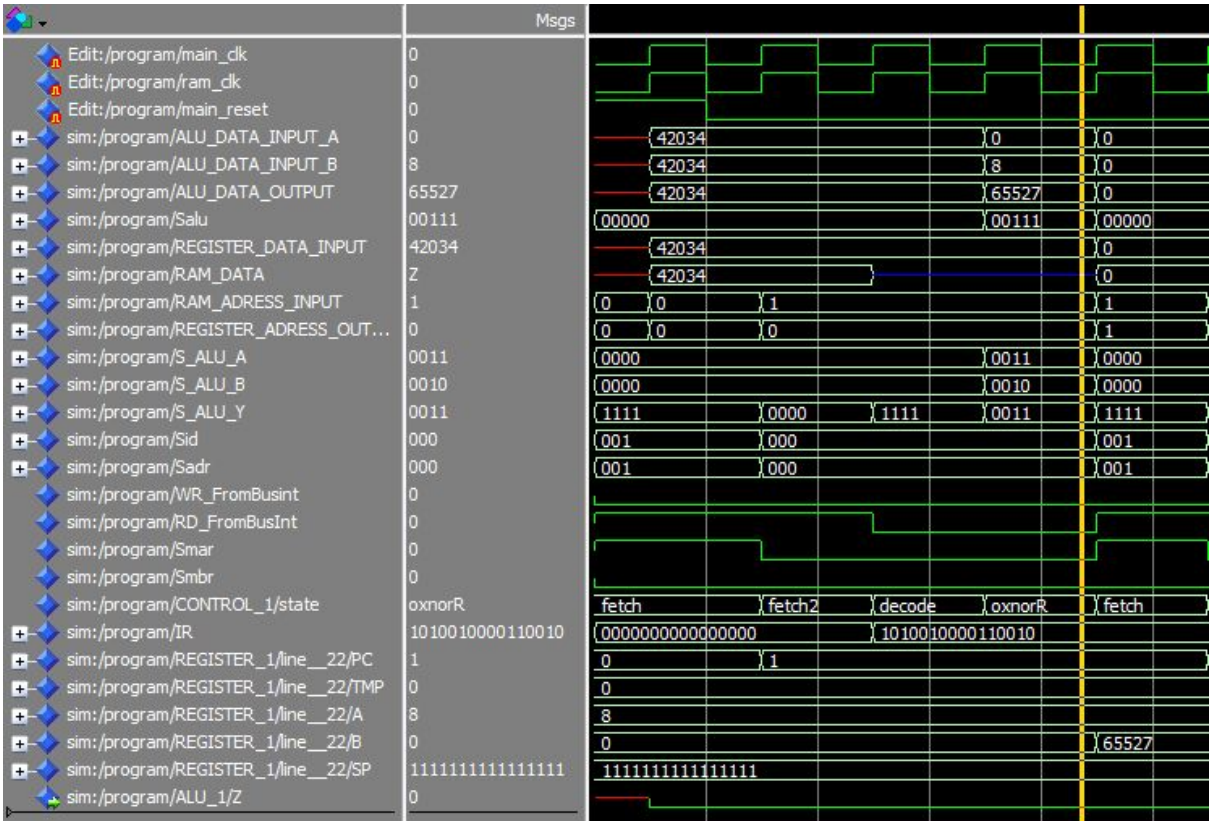
Do rejestru A została wprowadzona wartość 8.

Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. W bitach od 4 do 0 (tj. 5 najmłodszych bitów) rozkazu został zawarta wartość stałej, będącej argumentem 2 (w naszym przypadku jest to wartość 15). W bitach od 8 do 5 rozkazu został zawarty kod oznaczający z którego rejestru ma zostać odczytana wartość argumentu 1 (w naszym przypadku jest to rejestr A). Następnie wartości obu wskazanych rejestrów są równolegle odczytywane i zostają odpowiednio przesłane na wejścia ALU (w naszym przypadku wartość rejestru A zostaje przesłana na wejście A ALU, a wartość stałej na wejście B ALU). Później zostaje wykonana operacja xnor. Na koniec wynik przesyłany jest do rejestru będącego argumentem rozkazu.

XNOR rej1 rej2

Kod rozkazu: 1010010000110010

Symulacja:



Omówienie:

Do rejestru A została wprowadzona wartość 8.

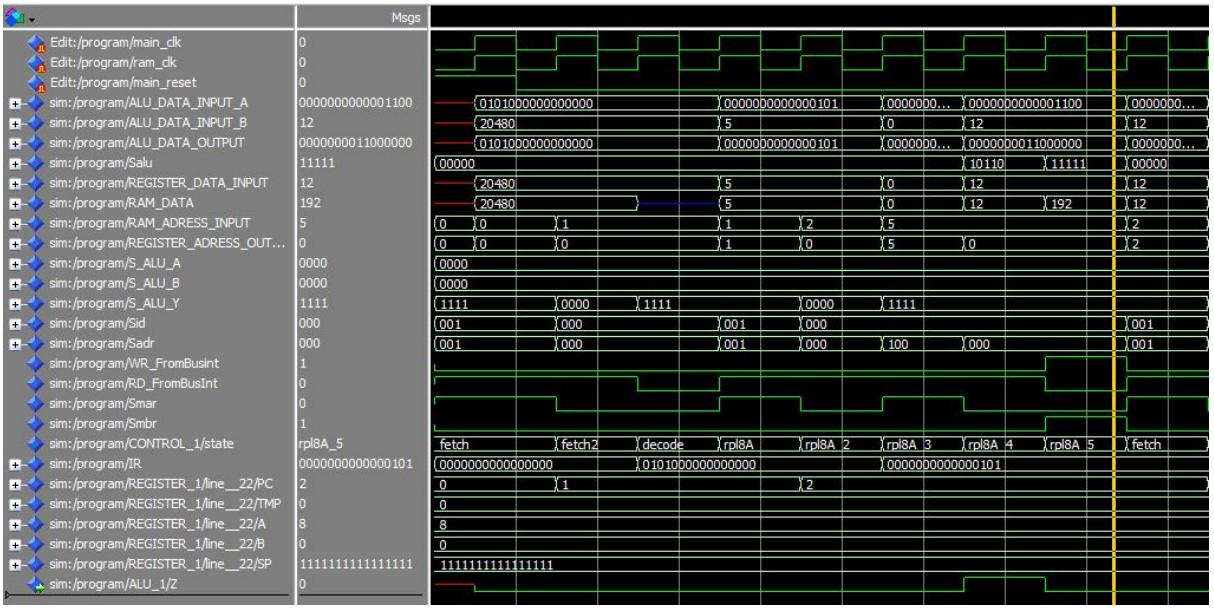
Do rejestru B została wprowadzona wartość 0.

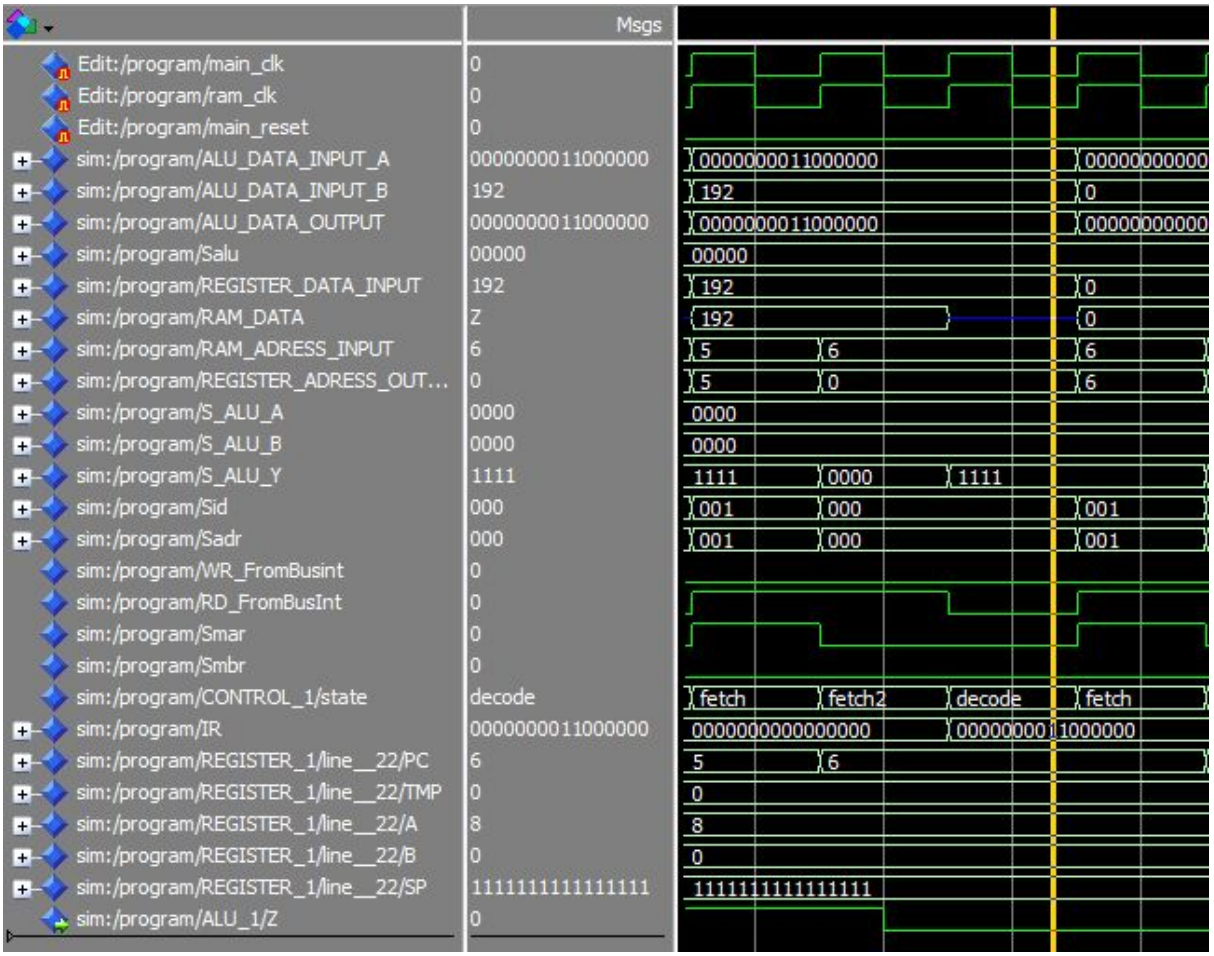
Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. W bitach od 3 do 0 (tj. 4 najmłodsze bity) rozkazu został zawarty kod oznaczający z którego rejestru ma zostać odczytana wartość argumentu 2 (w naszym przypadku jest to rejestr A). W bitach od 7 do 4 rozkazu został zawarty kod oznaczający z którego rejestru ma zostać odczytana wartość argumentu 1 (w naszym przypadku jest to rejestr B). Następnie wartości obu wskazanych rejestrów są równolegle odczytywane i zostają odpowiednio przesłane na wejścia ALU (w naszym przypadku wartość rejestru A zostaje przesłana na wejście A ALU, a wartość rejestru B na wejście B ALU). Później zostaje wykonana operacja xnor. Na koniec wynik przesyłany jest do rejestru będącego argumentem 1 rozkazu.

RPL8 adr

Kod rozkazu: 0101000000000000

Symulacja:





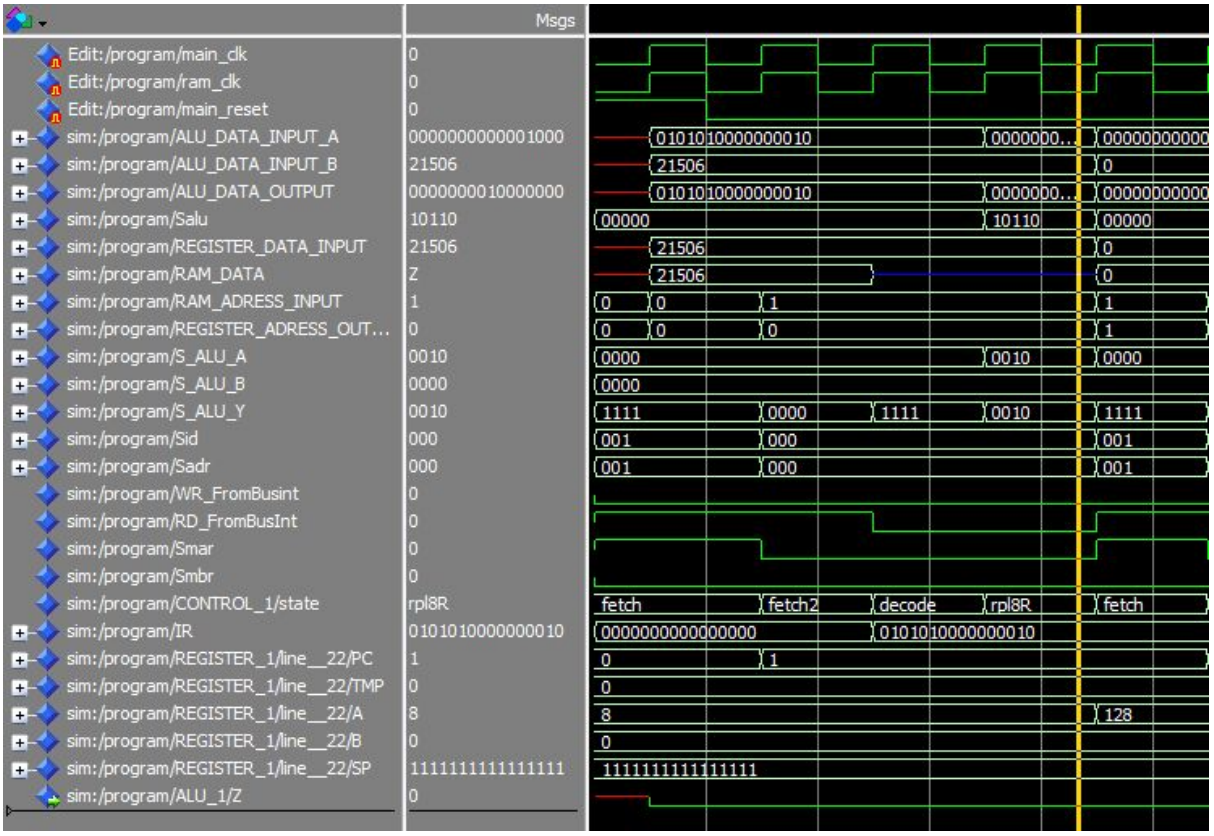
Omówienie:
Do komórki pamięci RAM o adresie 5 została wprowadzona wartość 12.
Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. Następnie został pobrany kolejny rozkaz programu, będący adresem komórki pamięci RAM w której przechowywana jest wartość argumentu (w naszym przypadku jest to komórka pamięci o adresie 5). W kolejnym kroku pobierana jest wartość argumentu 1 spod wskazanej komórki pamięci i wprowadzana na wejście A jednostki arytmetyczno-logicznej. Następnie zostaje wykonana operacja RPL8. Wynik jej działania zostaje przesłany do komórki pamięci RAM, z której został pobrany wcześniej argument.
Jak widać na drugim zrzucie ekranu procesor odczytując zawartość komórki pamięci o adresie 5 rozkazem fetch (RAM_ADRESS_INPUT = 5) pobiera wartość 192, co oznacza poprawne wykonanie operacji RPL8.
12 = 0000000000001100

W ośmiu najmłodszych bitach wartości binarnej liczby 12 bity 7-4 zostają zamienione z bitami 3-0. W wyniku takiej operacji otrzymujemy wartość: 0000000011000000, która jest binarną reprezentacją liczby 192.

RPL8 rej

Kod rozkazu: 0101010000000010

Symulacja:



Omówienie:

Do rejestru A została wprowadzona wartość 8.

Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. W 4 najmłodszych bitach rozkazu został zawarty kod oznaczający z którego rejestru ma zostać odczytana wartość argumentu. Następnie zawartość wskazanego rejestru wprowadzana jest na wejście A jednostki arytmetyczno-logicznej. Następnie zostaje wykonana operacja RPL8. Wynik jej działania zostaje przesłany do rejestru, z którego została odczytana wartość.

Jak widać na zrzucie ekranu procesor odczytał zawartość rejestru A, czyli wartość 8 oraz wyprowadził ją na wejście A ALU. Wyjście Y ALU wysterował w wartość 128, co oznacza poprawne wykonanie operacji RPL8.

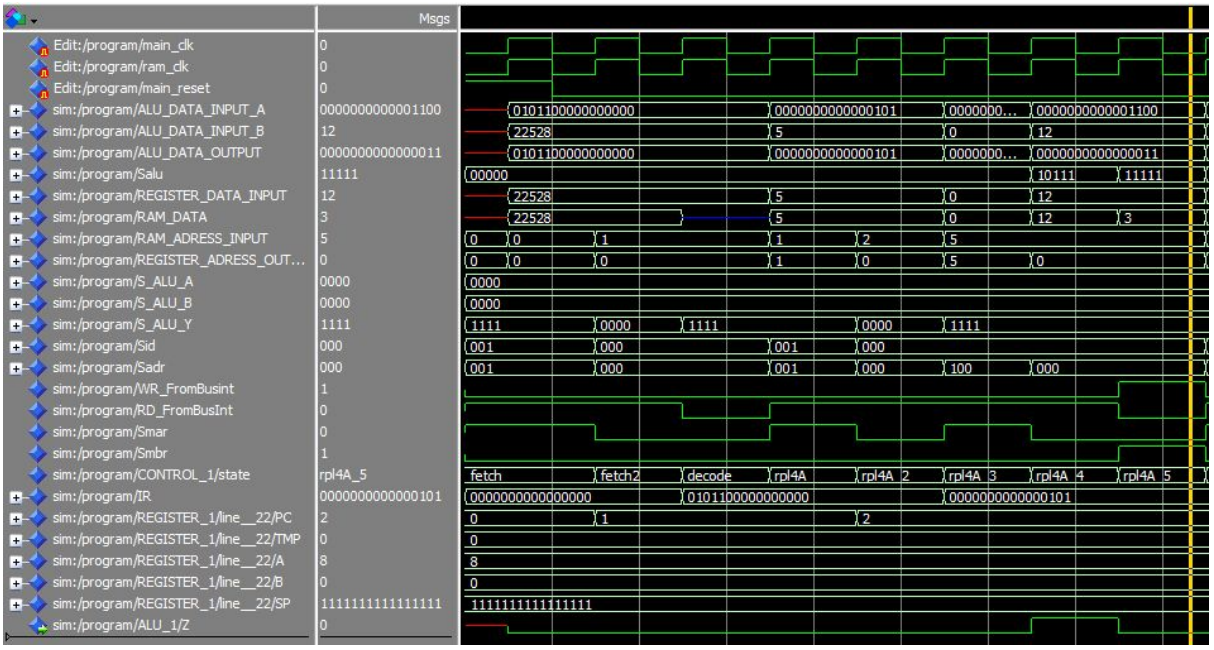
8 = 0000000000001000

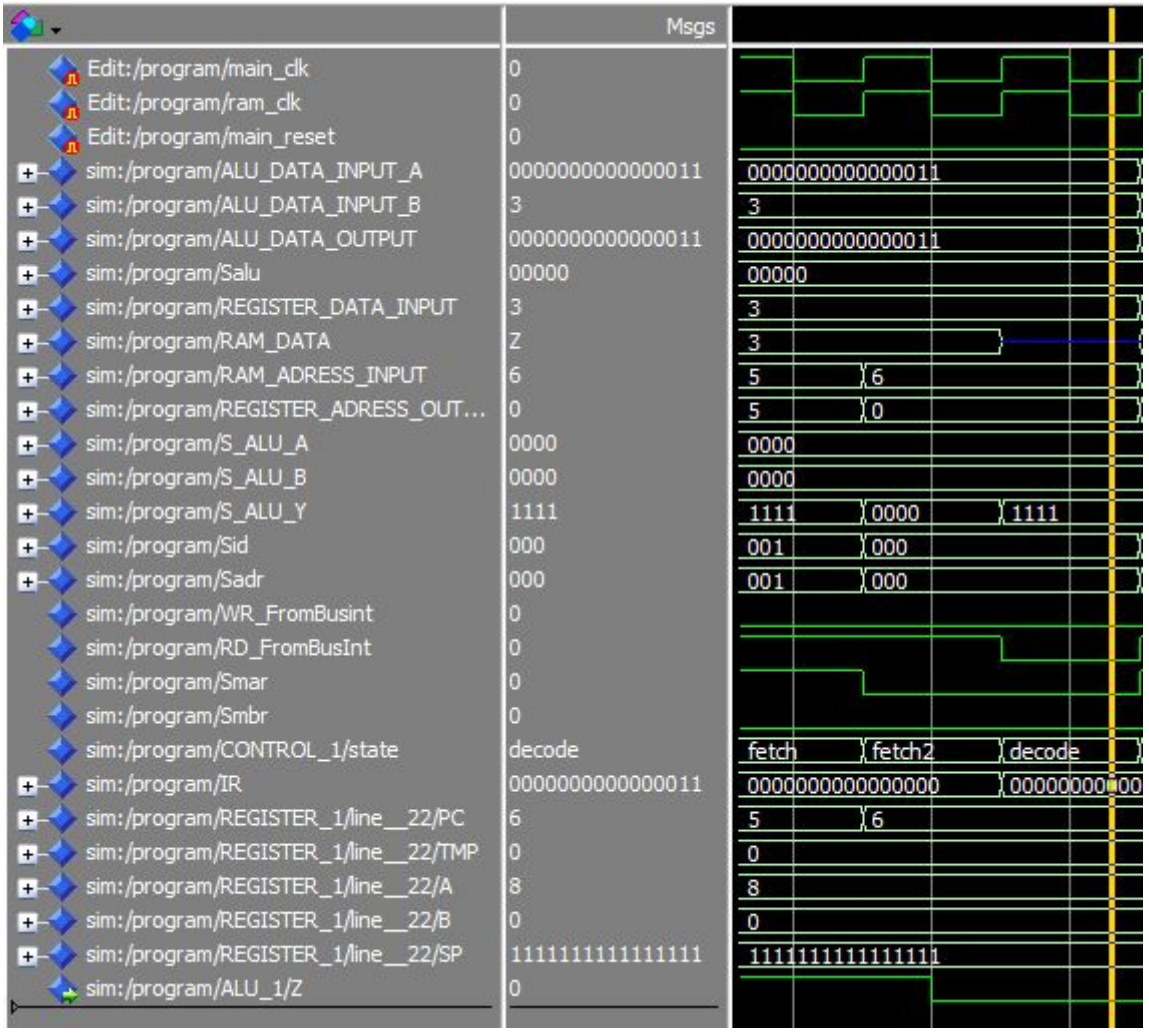
128 = 0000000010000000

RPL4 adr

Kod rozkazu: 0101100000000000

Symulacja:





Omówienie:

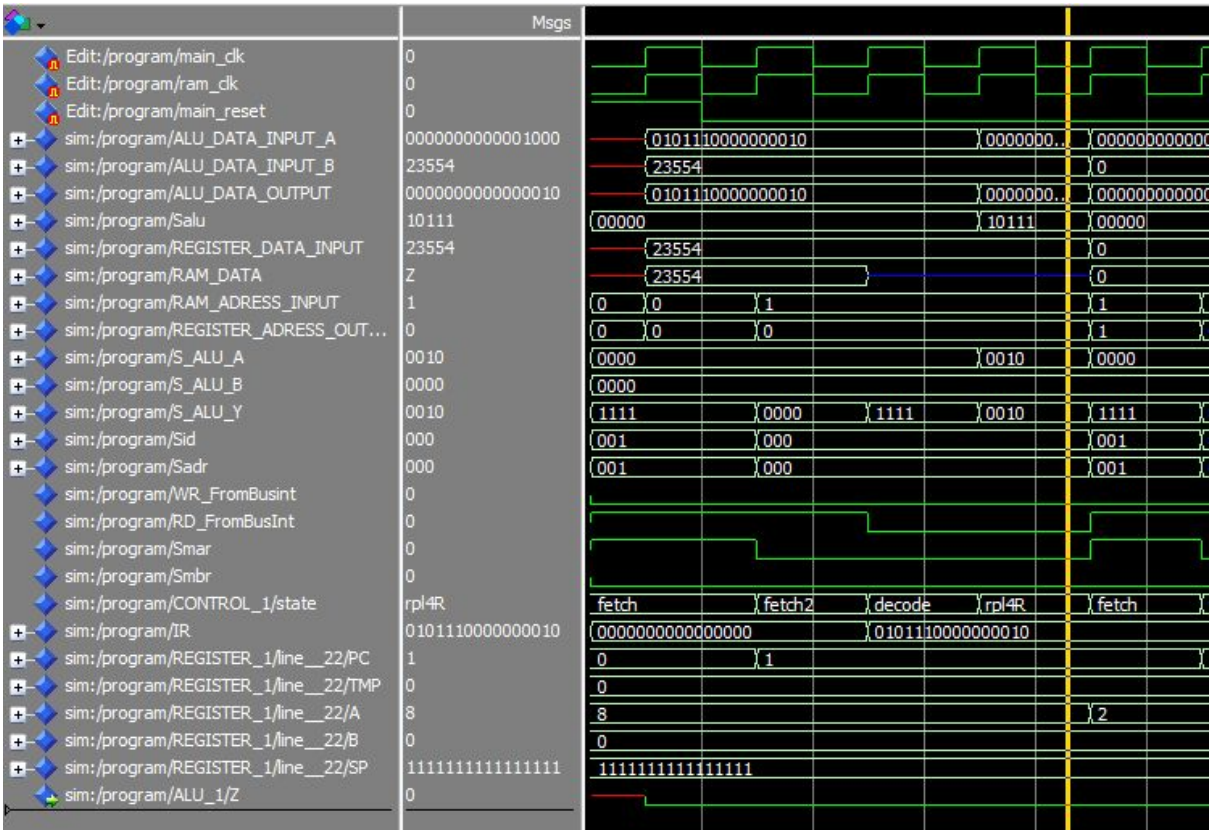
Do komórki pamięci RAM o adresie 5 została wprowadzona wartość 12.
Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. Następnie został pobrany kolejny rozkaz programu, będący adresem komórki pamięci RAM w której przechowywana jest wartość argumentu (w naszym przypadku jest to komórka pamięci o adresie 5). W kolejnym kroku pobierana jest wartość argumentu 1 spod wskazanej komórki pamięci i wprowadzana na wejście A jednostki arytmetyczno-logicznej. Następnie zostaje wykonana operacja RPL4. Wynik jej działania zostaje przesłany do komórki pamięci RAM, z której został pobrany wcześniej argument.
Jak widać na drugim zrzucie ekranu procesor odczytując zawartość komórki pamięci o adresie 5 rozkazem fetch (RAM_ADRESS_INPUT = 5) pobiera wartość 3, co oznacza poprawne wykonanie operacji RPL4.
12 = 0000000000001100

W czterech najmłodszych bitach wartości binarnej liczby 12 bity 3-2 zostają zamienione z bitami 1-0. W wyniku takiej operacji otrzymujemy wartość: 0000000000000011, która jest binarną reprezentacją liczby 3.

RPL4 rej

Kod rozkazu: 0101110000000010

Symulacja:



Omówienie:

Do rejestru A została wprowadzona wartość 8.

Na początku wykonane zostały rozkazy fetch i decode w wyniku których został pobrany do rejestru IR, a następnie zdekodowany rozkaz programu. W 4 najmłodszych bitach rozkazu został zawarty kod oznaczający z którego rejestru ma zostać odczytana wartość argumentu. Następnie zawartość wskazanego rejestru wprowadzana jest na wejście A jednostki arytmetyczno-logicznej. Następnie zostaje wykonana operacja RPL4. Wynik jej działania zostaje przesłany do rejestru, z którego została odczytana wartość.

Jak widać na zrzucie ekranu procesor odczytał zawartość rejestru A, czyli wartość 8 oraz wyprowadził ją na wejście A ALU. Wyjście Y ALU wysterował w wartość 2, co oznacza poprawne wykonanie operacji RPL4.

8 = 0000000000001000

2 = 0000000000000010