WT/N/07:30

Oddano:

Lenkiewicz Marcin
Pastuszek Mikołaj

Projekt i implementacja procesora 8051

Architektura komputerów 2 - projekt

Prowadzący:

Dr hab. inż. Tadeusz Tomczak

W niniejszej pracy prezentujemy uproszczony projekt procesora 8051 wykonany w programie Logisim. Przedstawiamy implementacje podstawowych bloków procesora oraz pokazujemy jak elementy wchodzące w jego skład oddziałują między sobą. Dzięki wykorzystaniu listy rozkazów mikrokontrolerów rodziny MCS-51, z powodzeniem udało nam się stworzyć symulację procesora, która jest kompatybilna z urządzeniami wchodzącymi w skład tej rodziny.

Spis treści

Spis ilustracji

Spis tabel

1. Wprowadzenie

Tematem wybranego przez nas projektu było zaprojektowanie i symulacja procesora 8051. Z uwagi na dydaktyczny charakter naszej pracy, za cel projektu obraliśmy zrozumienie i zaimplementowanie podstawowych elementów i koncepcji związanych z działaniem mikroprocesorów należących do rodziny MCS-51, a nie skupialiśmy się na idealnym odwzorowaniu całej funkcjonalności procesora 8051.

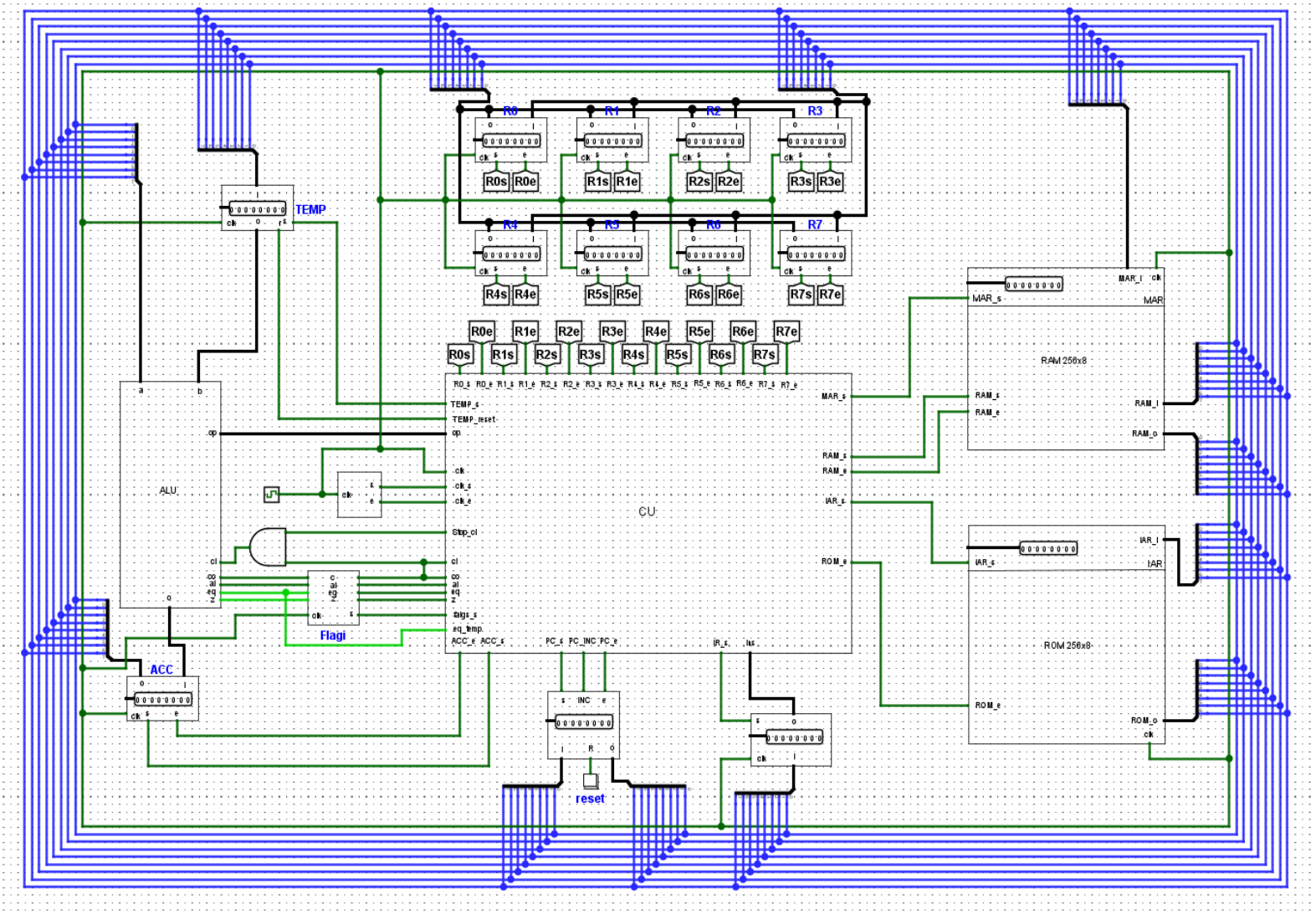
Z takim podejściem wiążą się jednak pewne uproszczenia, które musieliśmy przyjąć. Do najważniejszych z nich należą między innymi:

- Symulacja procesora odbywa się jedynie na poziomie samego wnętrza procesora, co oznacza, że nie bierzemy pod uwagę urządzeń peryferyjnych, które z zewnątrz sterują lub wpływają na pracę procesora.
- Pamięć programu została zredukowana do 256 bajtów z oryginalnych 4 tysięcy bajtów. Ułatwiło to proces dostępu do pamięci ROM, korzystając z 8-bitowej magistrali.
- Symulacja procesora przetwarza jedynie jednocyklowe instrukcje.
- Ze względu na wykorzystanie symulatora do stworzenia projektu nie są brane pod uwagę problemy, które mogą się pojawić przy rzeczywistej pracy procesora, czyli np. opóźnienia występujące z powodu długości przewodów lub nadmierne nagrzewanie się niektórych elementów.
- Ze względu na brak rozkazu HALT w zestawie instrukcji procesora 8051, po wykonaniu ostatniej instrukcji w pamięci programu, wywołany zostanie rozkaz bezwarunkowego skoku pod adres, w którym ta instrukcja się znajduje, tworząc nieskończoną pętlę.

2. Budowa procesora

a. Magistrala

Zespół linii służących do przesyłania sygnałów między połączonymi urządzeniami w mikroprocesorze. W naszym przypadku istnieje wspólna magistrala dla danych, adresów i jest ona 8-bitowa.



Ilustracja 1. Schemat procesora + magistrala (zaznaczona na niebiesko)

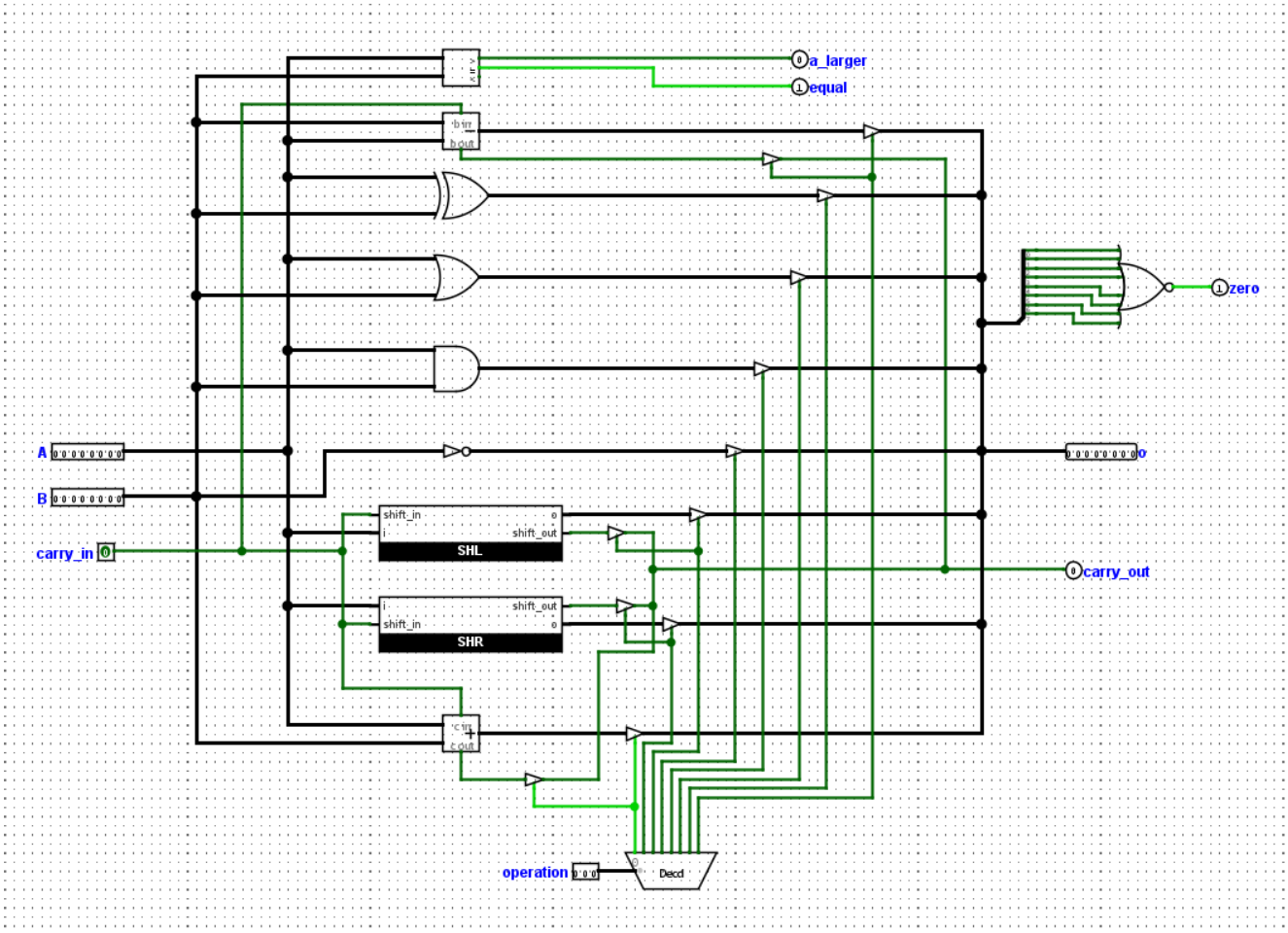
Schemat procesora jest dodatkowo dostępny w większym rozmiarze na stronie 17.

b. ALU

Jednostka arytmetyczno-logiczna. W naszym przypadku wykonuje ona następujące operacje:

- Dodawanie
- Odejmowanie
- Przesunięcie bitowe w lewo
- Przesunięcie bitowe w prawo
- Logiczny NOT
- Logiczny AND
- Logiczny OR
- Logiczny XOR

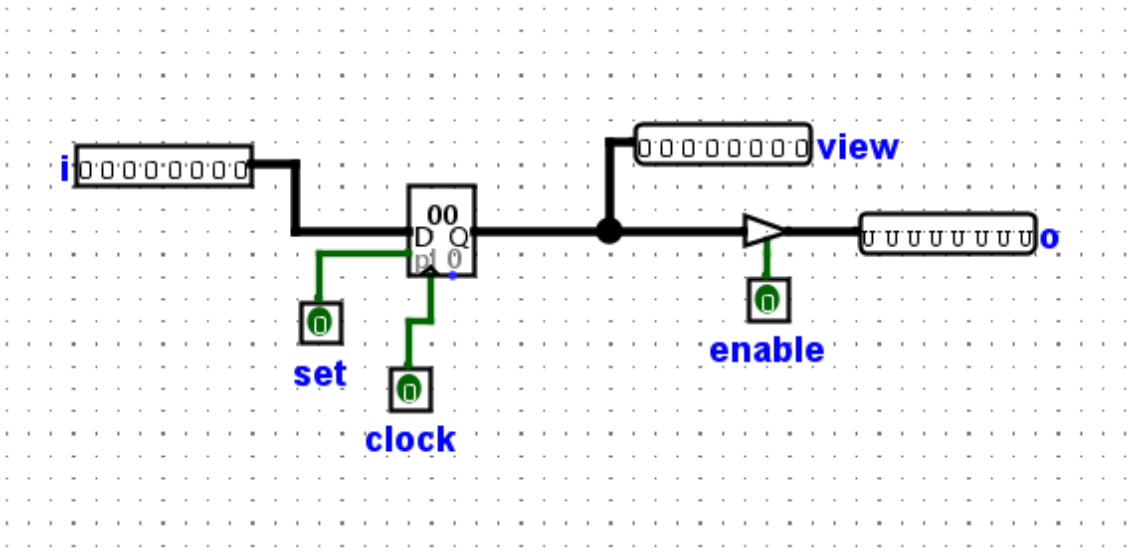
Wszystkie operacje są wykonywane na 8-bitowych argumentach. ALU generuje również flagi.



Ilustracja 2. Schemat ALU

c. Akumulator

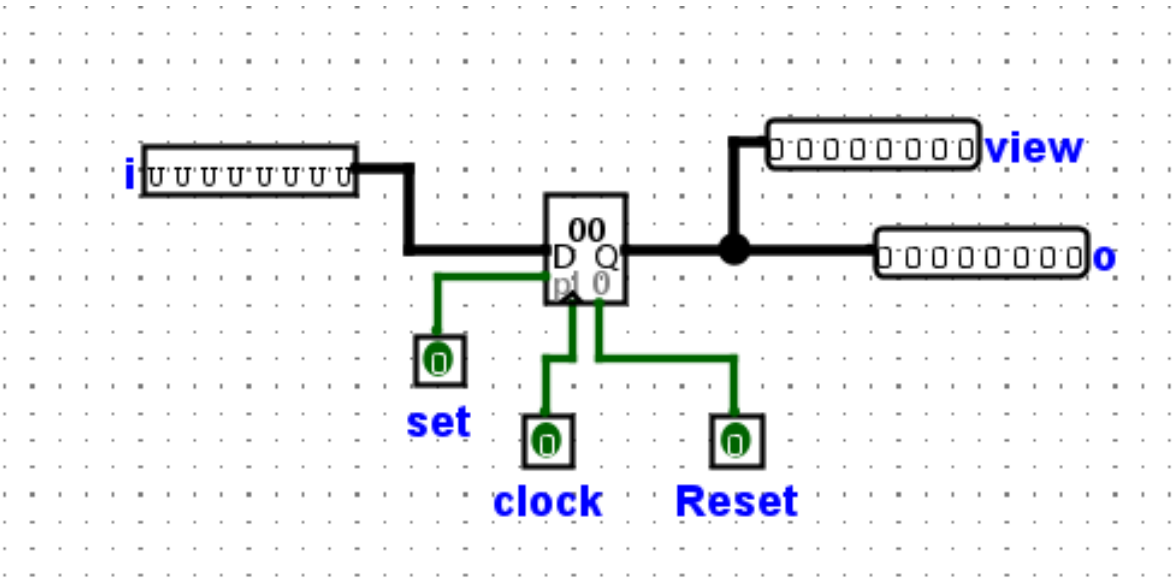
Akumulator (ACC) to specjalizowany 8-bitowy rejestr procesora, w którym umieszczane są wyniki operacji ALU. W działaniach arytmetycznych i logicznych, jeden z argumentów musi być przechowywany w akumulatorze.



Ilustracja 3. Schemat akumulatora

d. Rejestr TEMP

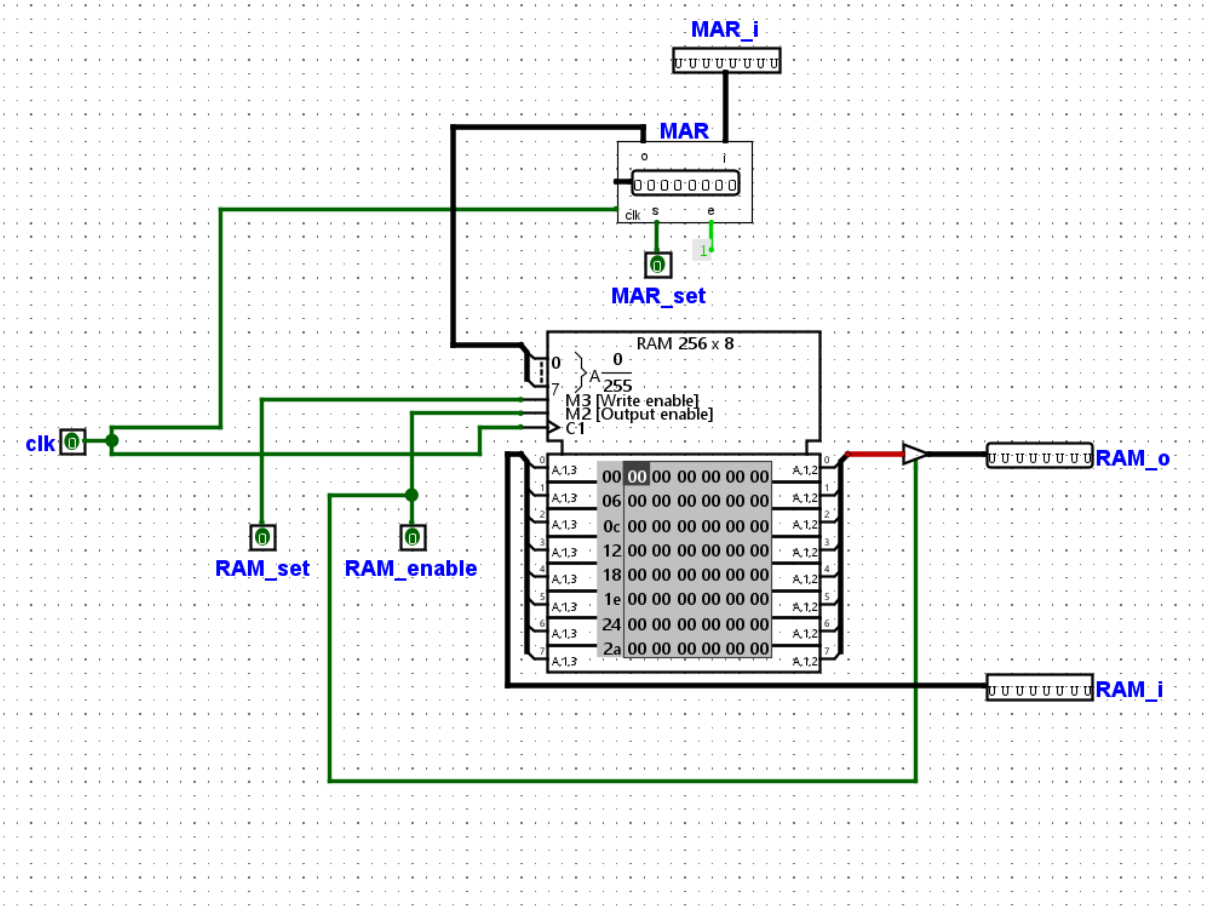
8-bitowy rejestr tymczasowy (TEMP) do którego przekazywana jest zawartość akumulatora przed wykonaniem operacji na ALU.



Ilustracja 4. Schemat rejestru tymczasowego

e. RAM I MAR

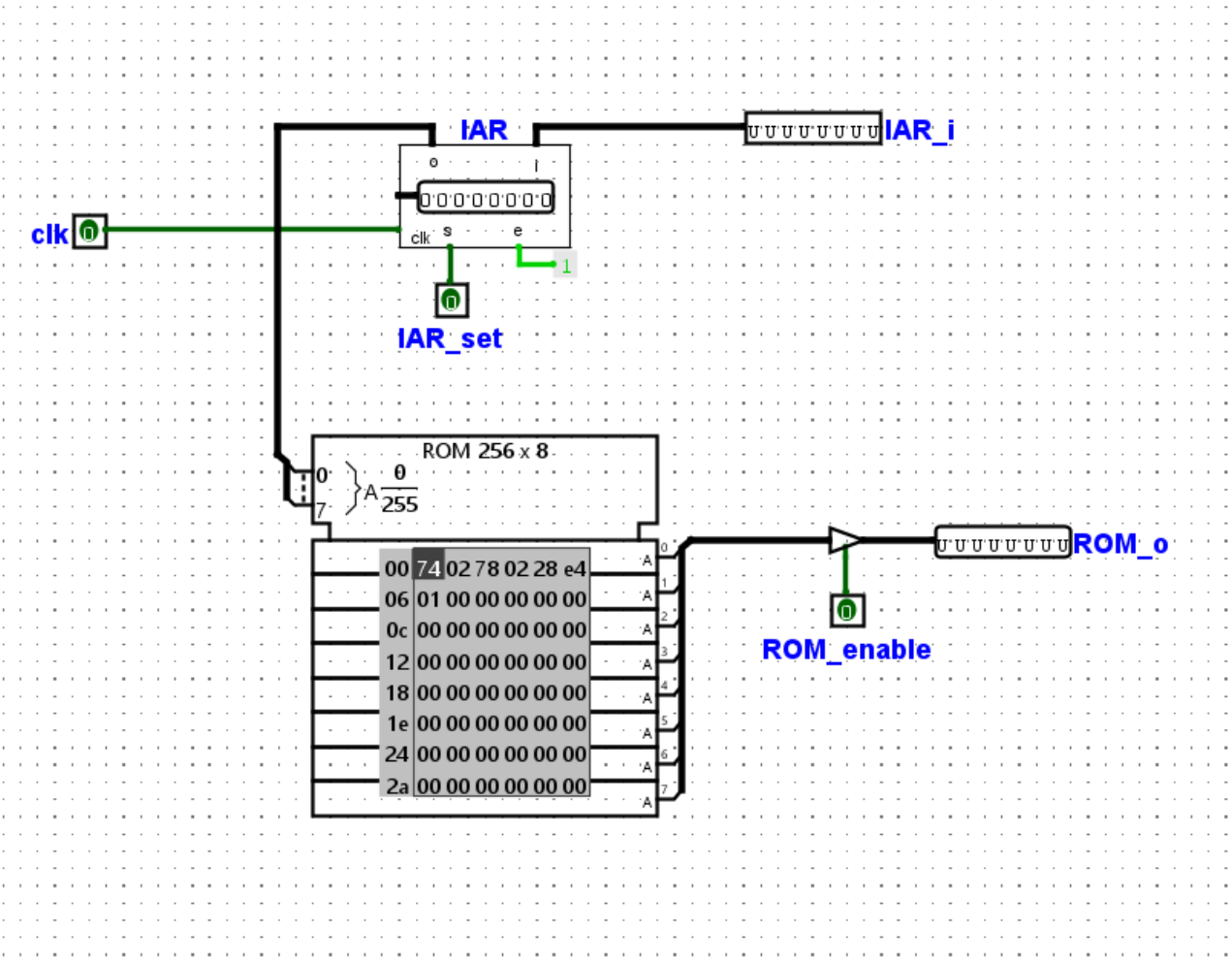
MAR (*Memory Address Register*) to 8-bitowy rejestr adresowy przechowujący adres, spod którego ma być pobrany bądź zapisany bajt w pamięci RAM. RAM to pamięć do odczytu i zapisu o bezpośrednim dostępie do dowolnej komórki pamięci. W naszym przypadku jest ona 256 bajtowa.



Ilustracja 5. Schemat bloku RAM (RAM i MAR)

f. ROM I IAR

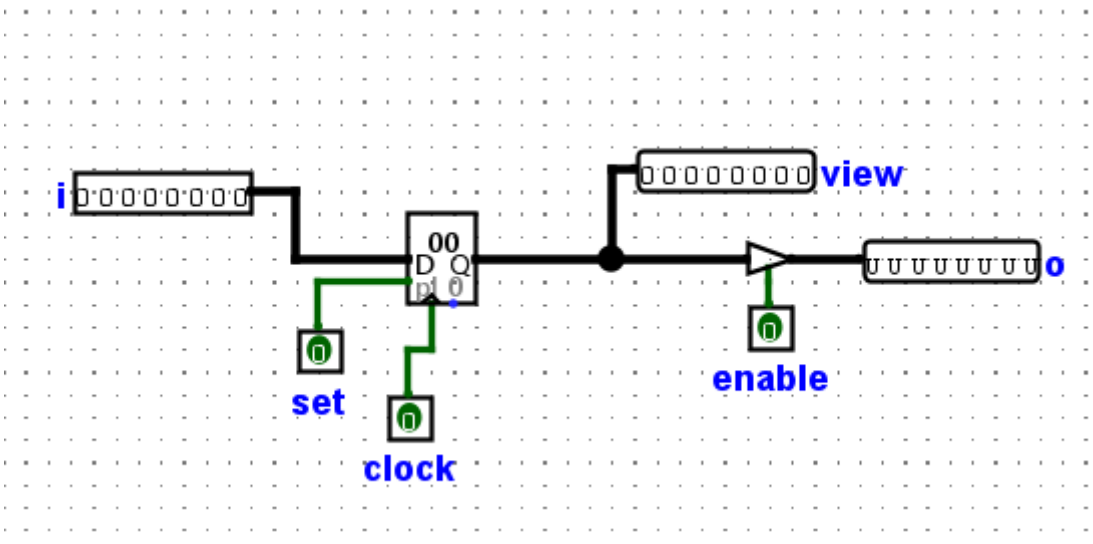
IAR (*Instruction Address Register*) to 8-bitowy rejestr adresowy przechowujący adres, spod którego ma być pobrana komórka z pamięci ROM. ROM to pamięć służąca jedynie do odczytu. Przechowuje ona wykonywujący się program.



Ilustracja 6. Schemat bloku ROM (ROM i IAR)

g. R0 – R7

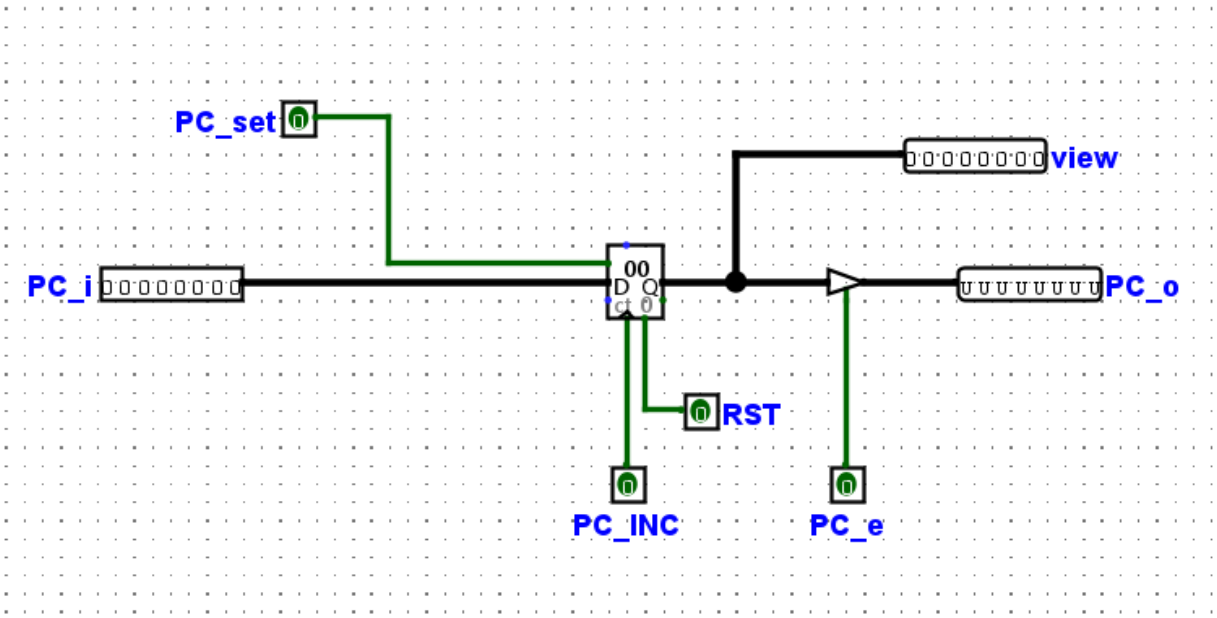
R0-R7 to rejestry robocze.



Ilustracja 7. Schemat rejestru roboczego

h. Program Counter

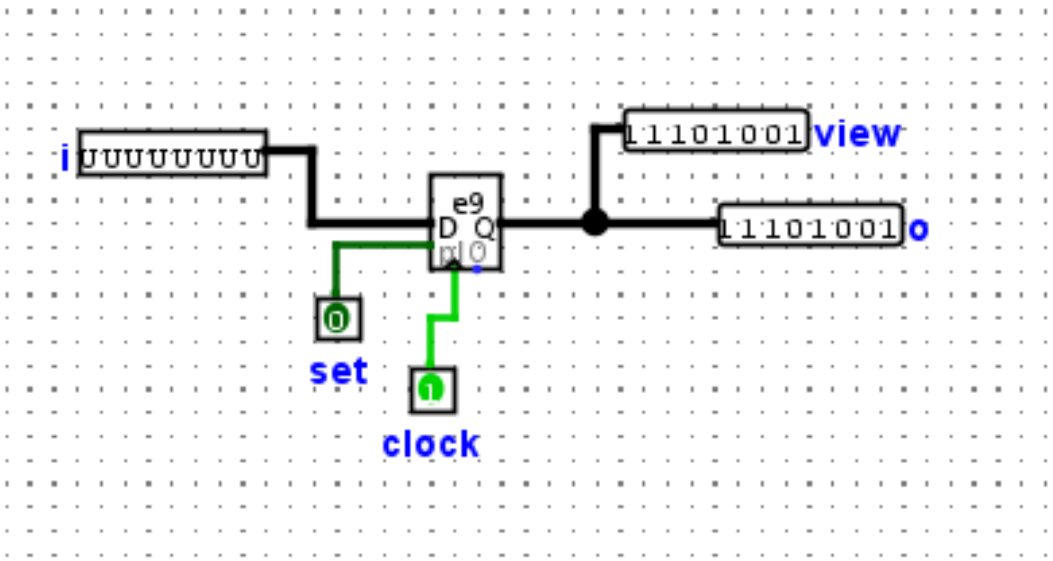
Licznik programu jest rejestrem procesora przechowującym informację, w którym miejscu programu aktualnie jesteśmy.



Ilustracja 8. Schemat licznika instrukcji

i. Instruction Register

Rejestr instrukcji (IR) to 8-bitowy rejestr, w którym przechowywana jest aktualnie wykonywana instrukcja maszynowa. Każda instrukcja, która ma być wykonywana, jest ładowana do rejestru instrukcji.



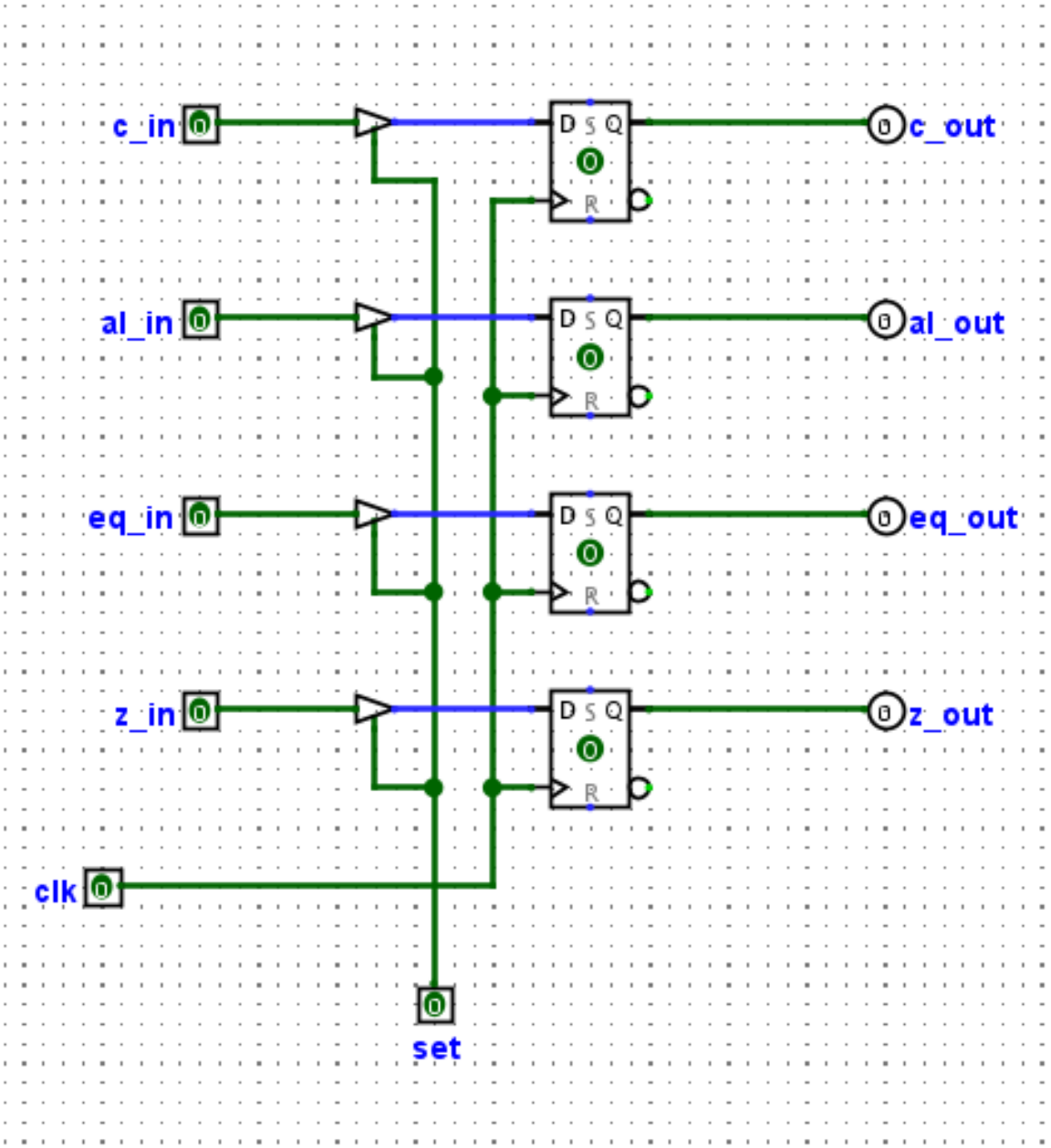
Ilustracja 9. Schemat rejestru instrukcji

j. PSW

Rejestr stanu programu (*PSW*) jest zbiorem flag wskazujących na aktualny stan programu:

- PSW.0=C (Carry) – przeniesienie z najstarszego bitu
- PSW.1 = al (A Larger) – Argument ALU A większy od argumentu B
- PSW.2 = eq (Equals) – Argument ALU A równy argumentowi B
- PSW.3 = z (Zero) – Wynik operacji ALU równy zero

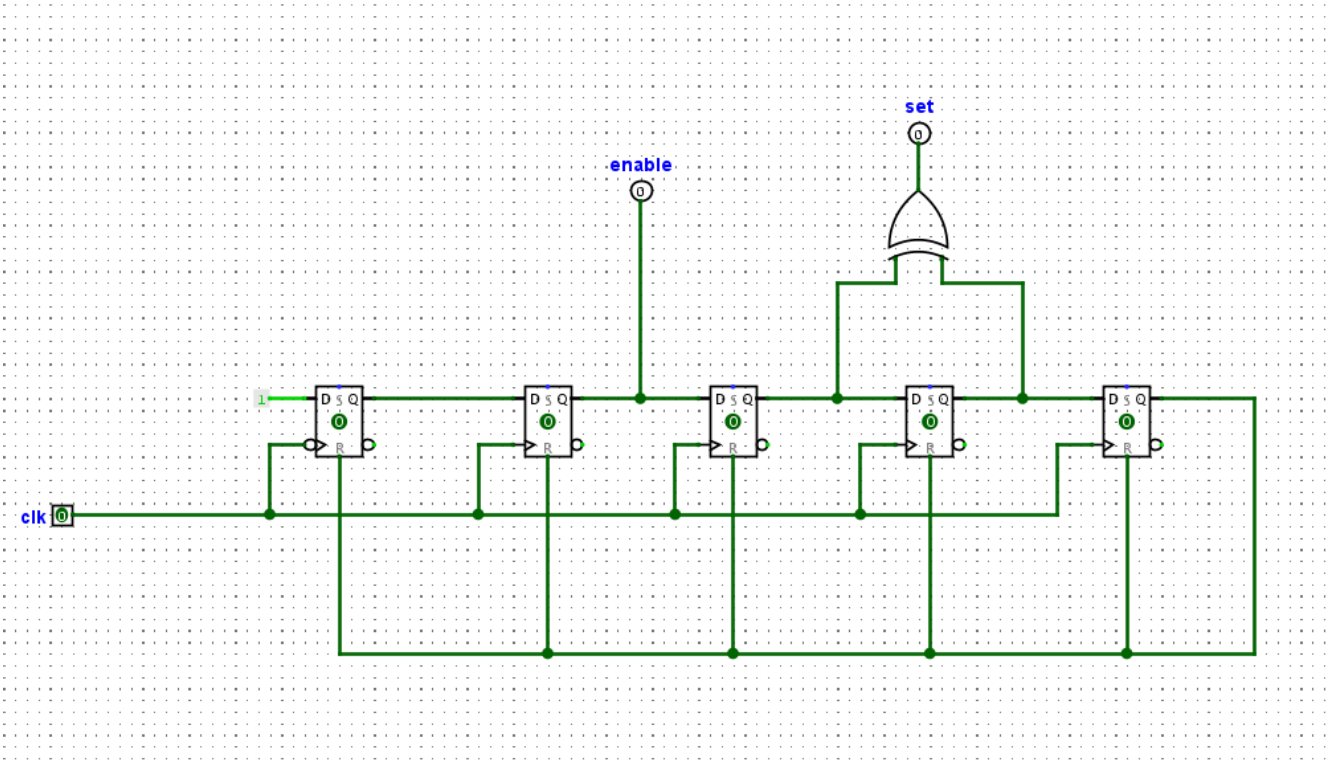
Flagi al, eq i z zostały dodane przez nas i nie występują w oryginalnym procesorze 8051.



Ilustracja 10. Schemat rejestru PSW

k. Generator sygnału SET i ENABLE

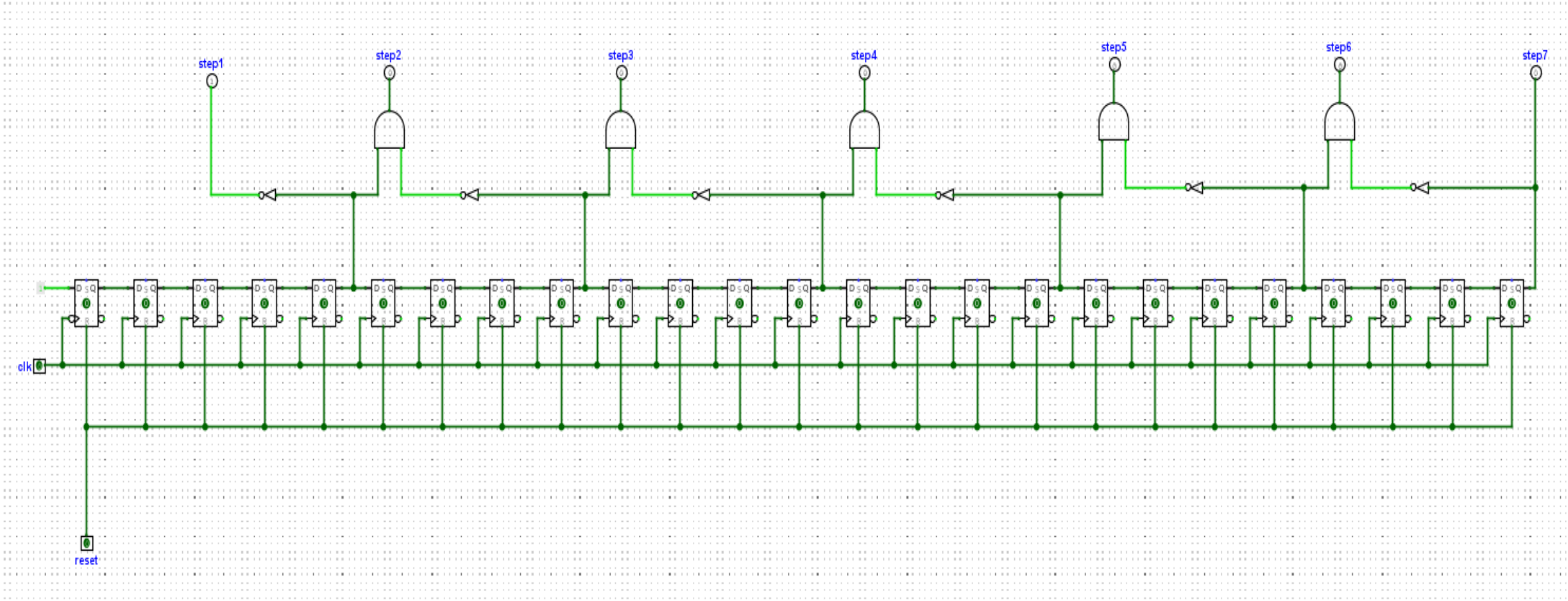
Generator sygnału set i enable został stworzony, aby odczytywać i zapisywać dane w różnych elementach mikroprocesora w odpowiedniej kolejności. Tworzy on cykl składający się z wyłączenia sygnału enable, włączenia sygnału enable, włączenia sygnału set i wyłączenia sygnału set.



Ilustracja 11. Schemat generatora sygnałów SET i ENABLE

l. Stepper

Stepper to pomocniczy element w CU. Wykonuje cykl 6 kroków. Jeden krok trwa 4 cykle zegara (jedną mikroinstrukcje). 7 krok resetuje STEPER (przechodzi do kroku 1).

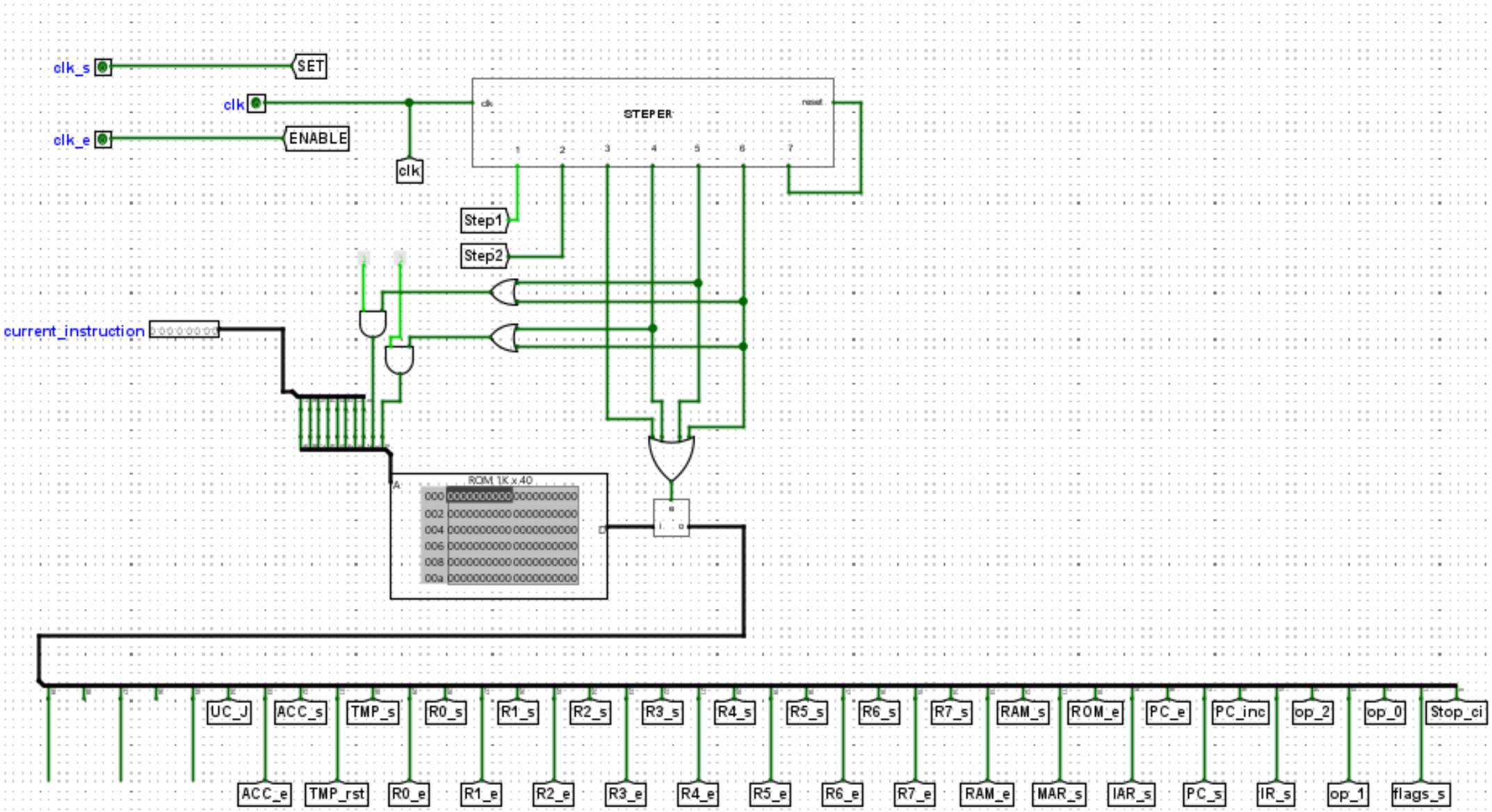


Ilustracja 12. Schemat steppera

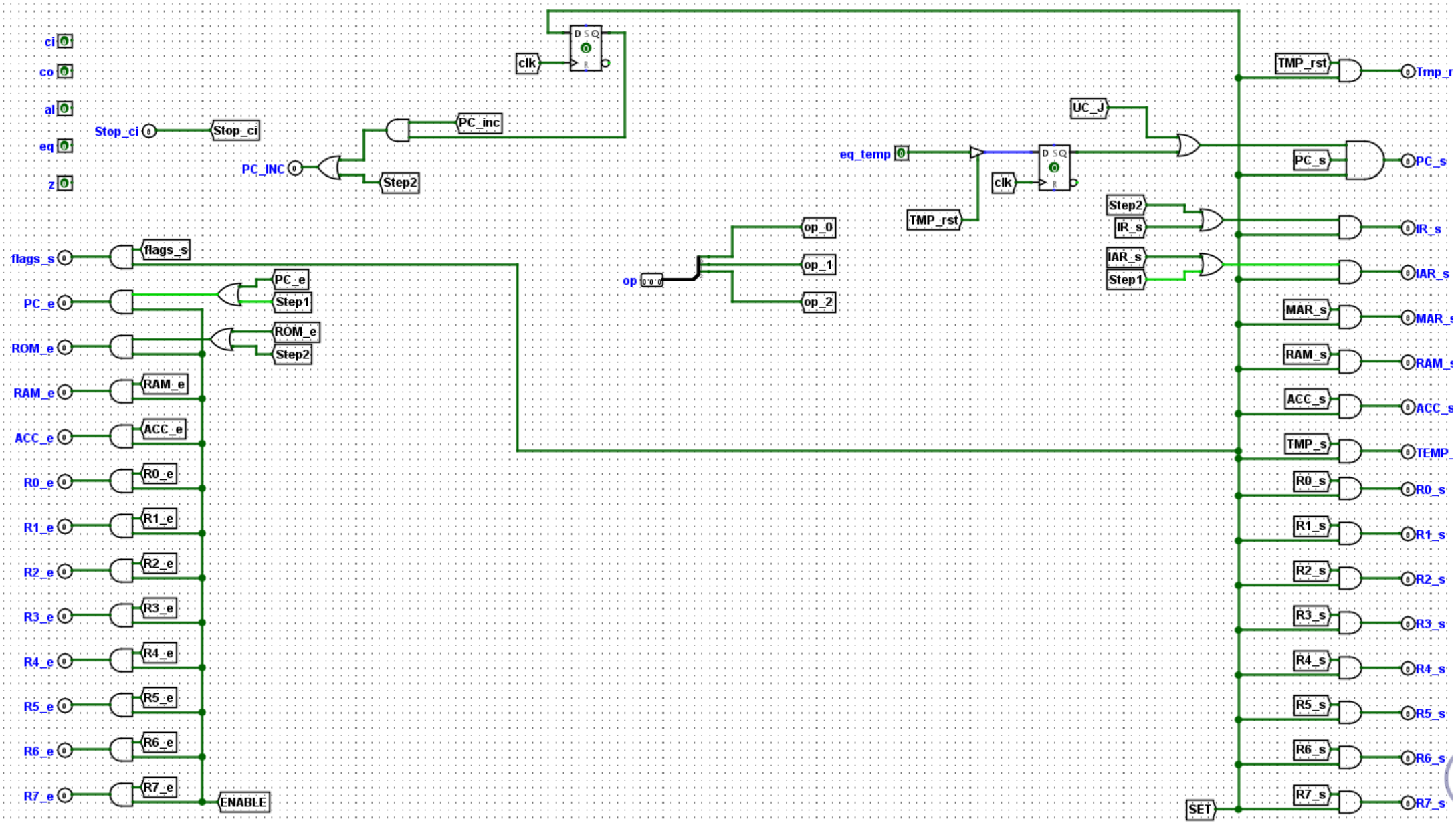
m. Control unit

Jednostka sterująca (*Control Unit*) jest odpowiedzialna za dekodowanie dostarczonych intrukcji i sterowanie pozostałymi elementami procesora. Dla każdej instrukcji dostarczonej do CU jest wykonywane 6 mikroinstrukcji. Dwie pierwsze mikroinstrukcje są takie same dla każdej instrukcji. Pozostałe 4 są zależne od adresu powstałego z połączenia 8 bitów instrukcji i dwóch bitów wytworzonych przez STEPER. Adres ten wskazuje na ciąg 40-bitów zapisany w pamięci ROM. 35 młodszych bitów odpowiada za sygnały przesyłane do bramek AND, które, wraz z sygnałami enable i set, ustawiają sygnały wychodzące do pozostałych bloków procesora. Pięć najstarszych bitów jest nieużywanych i może zostać wykorzystane w przyszłości, gdy procesor będzie rozszerzany o nowe możliwości. Dla przejrzystości i czytelności działania układu sterującego zostały zastosowane tunele.

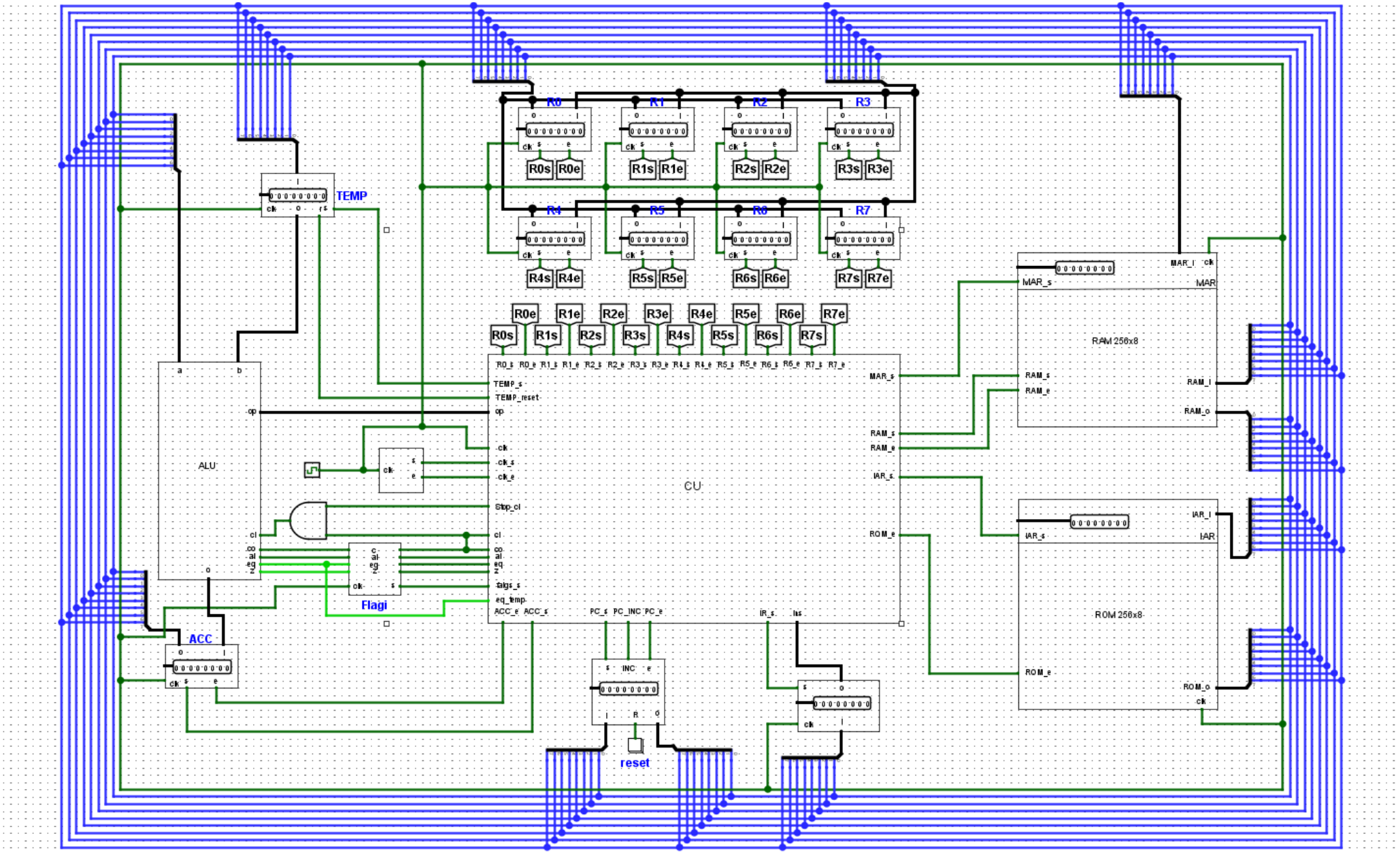
Z powodu rozmiaru układu, schematy zostały dołączone na kolejnych stronach.



Ilustracja 13. Schemat CU (cz. 1)



Ilustracja 14. Schemat CU (cz. 2)

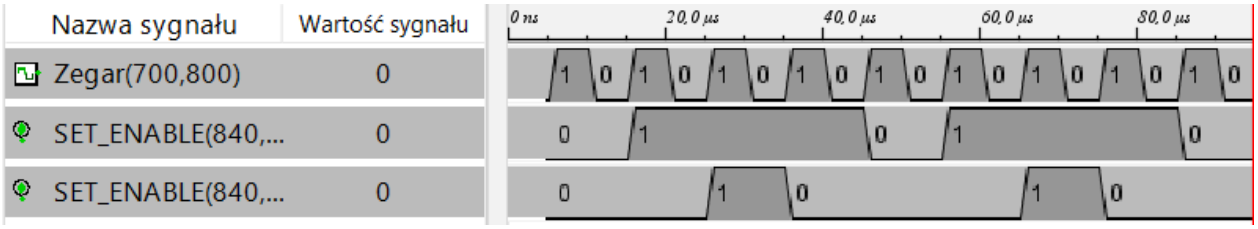


Ilustracja 15. Schemat procesora

3. Opis działania procesora

a. Wykonanie pojedynczej mikroinstrukcji

Pojedyncza mikroinstrukcja (proste działanie jak np. kopiowanie zawartości jednego rejestru do drugiego) wykonuje się w czasie 4 cykli zegara. Jest to najbardziej podstawowa operacja, polegająca na włączeniu i wyłączeniu możliwości zapisu i odczytu z magistrali przez wybrane elementy procesora.



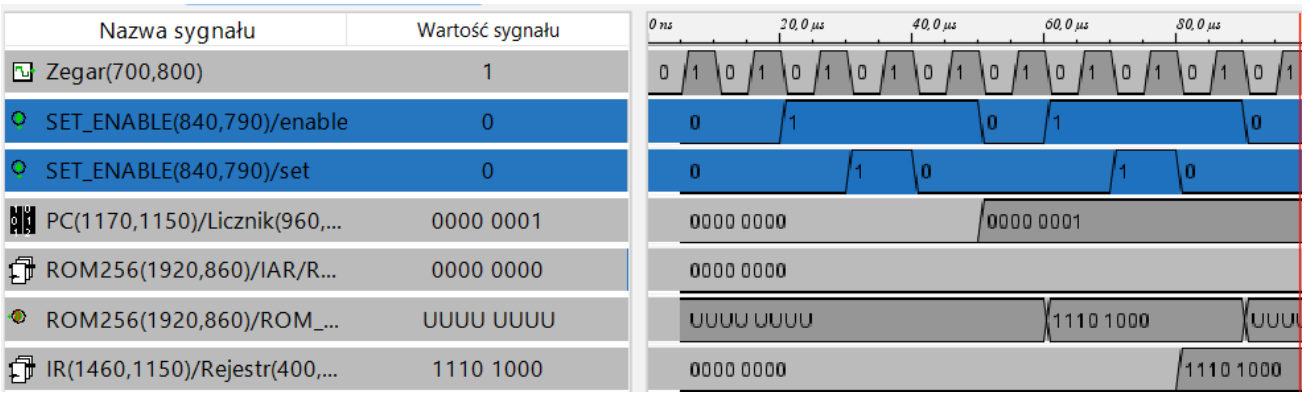
Ilustracja 16. Od góry: sygnał zegarowy, sygnał ENABLE, sygnał SET

b. Cykl maszynowy

W stworzonej przez nas symulacji procesora pojedynczy cykl maszynowy trwa 24 cykle zegarowe. Oznacza to, że pojedyncza instrukcja składa się z maksymalnie sześciu mikroinstrukcji. Dość długi czas trwania cyklu maszynowego wynika z typu architektury CISC (*Complex Instruction Set Computing*) procesora 8051. Architektura ta pozwala na występowanie specjalistycznych instrukcji o szerokiej gamie adresowania.

c. Etap pobieranie instrukcji (*Fetch Cycle*)

Pierwsze dwie mikroinstrukcje w każdym cyklu maszynowym powtarzają się. Ich zadaniem jest odczyt kodu kolejnej instrukcji z pamięci ROM, a następnie skopiowanie danej instrukcji do jednostki centralnej (*Central Unit*) w celu jej zdekodowania.



Ilustracja 17. Skopiowanie do CU kodu instrukcji MOV A, R0 (kod E8).

Od góry: sygnał zegarowy, sygnał ENABLE, sygnał SET, zawartość PC, zawartość IAR, wartość wypisana na magistralę przez ROM, zawartość IR.

Jak widać na powyższej ilustracji algorytm pobrania instrukcji można przedstawić za pomocą następującej listy kroków:

1. Wypisanie na magistrale zawartości PC (*Program Counter*).
2. Wpisanie do IAR (*Instruction Address Register*) zawartości magistrali.
3. Wypisanie na magistrale zawartości pamięci ROM spod adresu zawartego w IAR.
4. Wpisanie do IR (*Instruction Register*) zawartości magistrali (zawartość IR automatycznie przesyłana jest do CU).

d. Etap dekodowania instrukcji

Po dostarczeniu instrukcji do CU, jest ona dekodowana. Dekodowanie polega na pobraniu z wewnętrznej pamięci ROM sygnału sterującego spod wygenerowanego 10 bitowego adresu. Sygnał sterujący to ciąg bitów (40 bitów) decydujący, które elementy procesora będą mogły wpisywać i wczytywać dane z magistrali podczas wykonywania danej mikroinstrukcji. Ustala również dodatkowe opcje dotyczące wykonywania instrukcji (np. jeśli instrukcja jest dwubajtowa – pobiera dodatkowy argument lub uwzględnia flagę przeniesienia podczas operacji dodawania).
Generowanie adresu jest obliczane w następujący sposób:

$$\mathbf{8}\ \boldsymbol{bitow\ instrukcji}\ (\boldsymbol{Opcode}) + \mathbf{2}\ \boldsymbol{bity\ kroku} = \mathbf{10}\ \boldsymbol{bitow\ intrukcji}$$

Taki zapis pozwala, aby każda instrukcja składała się z wielu (czterech, nie biorą pod uwagę dwóch kroków przeznaczonych na pobranie instrukcji) mikroinstrukcji.

Sygnały sterujące są tworzone i zapisywane w wewnętrznej pamięci ROM na etapie projektowanie procesora.

e. Etap wykonywania instrukcji (mikroinstrukcji)

Etap wykonywania instrukcji polega na aktywowaniu wybranych sygnałów ENABLE oraz SET dla danego kroku i ewentualnym wpisywaniu wyników operacji do zadeklarowanego docelowego miejsca w pamięci.

Działanie całego procesora sprowadza się do cyklicznego wykonywania powyższych 3 etapów, do momentu dojścia do ostatniej instrukcji w pamięci programu.

4. Lista zaimplementowanych instrukcji

Legenda do tabel:

Rn – Rejestry robocze R7 - R0

Address – 8-bitowy adres wskazujący na dane w wewnętrznej pamięci RAM bądź ROM

@Ri – 8-bitowa lokalizacja w wewnętrznej pamięci RAM, adresowana pośrednio przez rejestr R1 lub R0

#data – 8-bitowa stała zawarta w instrukcji

Instrukcje odpowiadające za transfer danych

Nazwa instrukcji: MOV

Funkcja: Skopiowanie bajtów

Opis: Bajt wskazany przez drugi operand jest kopiowany do lokalizacji określonej przez pierwszy operand. Operand źródłowy nie jest naruszony. Operacja nie zmienia żadnych flag.

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
MOV A, Rn	MOV A, R0	1	E8	11101000	-
	MOV A, R1	1	E9	11101001	-
	MOV A, R2	1	EA	11101010	-
	MOV A, R3	1	EB	11101011	-
	MOV A, R4	1	EC	11101100	-
	MOV A, R5	1	ED	11101101	-
	MOV A, R6	1	EE	11101110	-
	MOV A, R7	1	EF	11101111	-
MOV A, @Rn	MOV A, @R0	1	E6	11100110	-
	MOV A, @R1	1	E7	11100111	-
MOV A, #data	MOV A, #data	2	74	1110100	#data
MOV A, address	MOV A, address	2	E5	11100101	address
MOV Rn, A	MOV R0, A	1	F8	11111000	-
	MOV R1, A	1	F9	11111001	-
MOV Rn, #data	MOV R0, #data	2	78	01111000	#data
	MOV R1, #data	2	79	01111001	#data
MOV Rn, address	MOV R0, address	2	A8	10101000	address
	MOV R1, address	2	A9	10101001	address
MOV address, A	MOV address, A	2	F5	11110101	address

Tabela 1. Kody rozkazu MOV

Operacje arytmetyczne

Nazwa instrukcji: ADD

Funkcja: Dodaj

Opis: ADD dodaje drugi operand do zawartości akumulatora i wynik operacji zapisuje w akumulatorze. Podczas dodawania dwóch liczb w zależności od wystąpienia przepełnienia na 7 bicie, flaga carry zmienia się na 0 - gdy nie doszło do przepełnia, bądź 1- gdy doszło do przepełnienia.

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
ADD A, Rn	ADD A, R0	1	28	00101000	-
	ADD A, R1	1	29	00101001	-
ADD A, #data	ADD A, #data	2	24	00100100	#data
ADD A, #address	ADD A, address	2	25	00100101	address

Tabela 2. Kody rozkazu ADD

Nazwa instrukcji: ADDC

Funkcja: Dodaj z przeniesieniem

Opis: ADDC dodaje drugi operand do akumulatora i wynik operacji zapisuje w akumulatorze. Gdy flaga carry jest ustawiona na 1 (C = 1) to wynik zwiększy się o 1. Podczas dodawania dwóch liczb w zależności od wystąpienia przepełnienia na 7 bicie, flaga carry zmienia się na 0 - gdy nie doszło do przepełnia, bądź 1- gdy doszło do przepełnienia.

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
ADDC A, Rn	ADDC A, R0	1	38	00101000	-
	ADDC A, R1	1	39	00101001	-
ADDC A, #data	ADDC A, #data	2	34	00100100	#data

Tabela 3. Kody rozkazu ADDC

Nazwa instrukcji: SUBB

Funkcja: Odejmowanie z pożyczką

Opis: SUBB odejmuje od akumulatora drugi operand wraz z flagą carry i wynik zapisuje do akumulatora. Podczas odejmowania w zależności od wystąpienia pożyczki na 7 bicie, flaga carry zmienia się na 0 – gdy pożyczka nie wystąpi, bądź 1 – gdy pożyczka wystąpi.

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
SUBB A, Rn	SUBB A, R0	1	98	10011000	-
	SUBB A, R1	1	99	10011001	-
SUBB A, #data	SUBB A, #data	2	94	10010100	#data
SUBB A, address	SUBB A, address	2	95	10010101	address

Tabela 4. Kody rozkazu SUBB

Operacje logiczne

Nazwa instrukcji: ANL

Funkcja: Logiczny AND dla zmiennych bajtowych

Opis: ANL wykonuje operację bitowo-logiczną AND między akumulatorem a drugim operandem i wynik jest zapisywany do akumulatora. Operacja nie ma wpływu na żadne flagi.

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
ANL A, Rn	ANL A, R0	1	58	01011000	-

Tabela 5. Kody rozkazu ANL

Nazwa instrukcji: ORL

Funkcja: Logiczny OR dla zmiennych bajtowych

Opis: ORL wykonuje operację bitowo-logiczną OR między akumulatorem a drugim operandem i wynik jest zapisywany do akumulatora. Operacja nie ma wpływu na żadne flagi.

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
ORL A, Rn	ORL A, R0	1	48	01001000	-

Tabela 6. Kody rozkazu ORL

Nazwa instrukcji: XRL

Funkcja: Logiczny Exclusive-OR dla zmiennych bajtowych

Opis: XRL wykonuje operację bitowo-logiczną XOR między akumulatorem a drugim operandem i wynik jest zapisywany do akumulatora. Operacja nie ma wpływu na żadne flagi.

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
XRL A, Rn	XRL A, R0	1	68	01101000	-

Tabela 7. Kody rozkazu XRL

Nazwa instrukcji: CLR A

Funkcja: Czyszczenie akumulatora

Opis: Akumulator jest czyszczony (wszystkie bity są ustawione na 0). Operacja nie ma wpływu na żadne flagi.

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
CLR A	CLR A	1	E4	11100100	-

Tabela 8. Kody rozkazu CLR

Nazwa instrukcji: RLC A

Funkcja: Rotacja akumulatora w lewo biorąc pod uwagę flagę carry.

Opis: Osiem bitów w akumulatorze i bit flagi carry są rotowane w lewo, 7 bit akumulatora zostaje przenoszony do flagi carry, a flaga zostaje przeniesiona na 0 bit akumulatora.

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
RLC A	RLC A	1	33	00110011	-

Tabela 9. Kody rozkazu RLC

Nazwa instrukcji: RRC A

Funkcja: Rotacja akumulatora w prawo biorąc pod uwagę flagę carry.

Opis: Osiem bitów w akumulatorze i bit flagi carry są rotowane w prawo, 0 bit akumulatora zostaje przenoszony do flagi carry, a flaga zostaje przeniesiona na 7 bit akumulatora.

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
RRC A	RRC A	1	13	00010011	-

Tabela 10. Kody rozkazu RRC

Nazwa instrukcji: CPL A

Funkcja: Negacja bitów a akumulatora

Opis: Każdy bit akumulatora jest negowany

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
CPL A	CPL A	1	F4	11110100	-

Tabela 11. Kody rozkazu CPL

Rozgałęzienia programu

Nazwa instrukcji: JMP

Funkcja: Skok bezwarunkowy pod podany address

Opis: Przekazuje wykonanie programu pod wskazany adres. Operacja nie ma wpływu na żadne flagi.

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
JMP address	JMP address	1	E1	11100001	address

Tabela 12. Kody rozkazu JMP

Nazwa instrukcji: JZ

Funkcja: Skacz, jeżeli w akumulatorze zero

Opis: Jeżeli zawartość akumulatora jest równa zero, to przekazuje wykonanie programu pod wskazany adres. Operacja nie ma wpływu na żadne flagi.

Rodzaj instrukcji	Instrukcja wraz z operandami	Liczba bajtów	Opcode HEX	Opcode BIN	Drugi Bajt
JZ address	JZ address	1	60	01100000	address

Tabela 13. Kody rozkazu JZ

5. Analiza wybranych instrukcji na podstawie wykresów czasowych

Instrukcja: ADD A, #data

Kod HEX wpisany do pamięci ROM: 24 03

Liczby HEX wpisany do pamięci RAM: brak

Opis: Pierwsze 2 kroki powodują pobranie instrukcji. Szczegółowo zostały opisane w podrozdziale **3c**. W trzecim kroku trzeciego cyklu zegara poprzez wystąpienie sygnałów enable (wiersz 2) i PC enable (wiersz 5) na magistrale została wysłana zwartość PC. W następnych cyklach zegara (3 i 4) został wysłany sygnał set (3 wiersz) oraz IAR set (wiersz 8) po czym zostały wyłączone (4 cykl). Między włączeniem a wyłączeniem sygnałów set, do IAR została wpisana zawartość magistrali. W 4 kroku został inkrementowany PC (wiersz 4) oraz zostały przesłane dane z ACC do TEMP przez magistrale. Cała operacja przesłania danych dzieje się dzięki sterowaniom sygnałów set enable. W 5 kroku zawartość rejestru TEMP i danych z pamięci ROM przesłanych na magistrale zostaje dodane do siebie w ALU i wynik operacji wpisywany jest do ACC (13 wiersz). Krok 6 nie wpływa na wynik działania instrukcji.

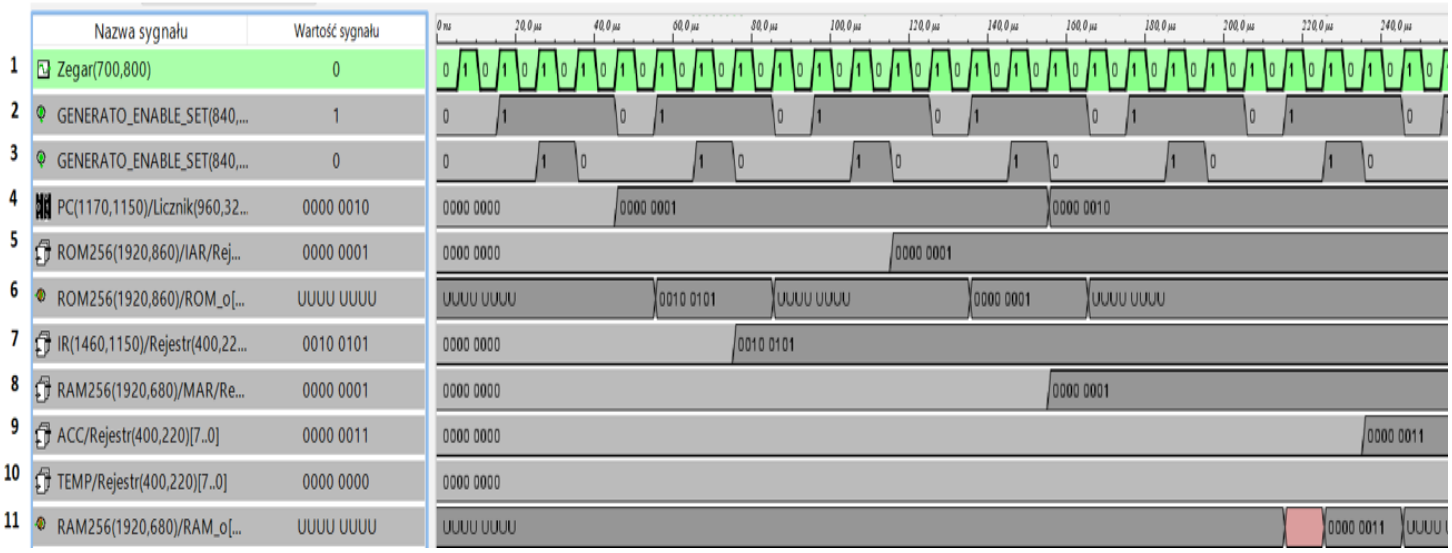
Ilustracja 18. Przebiegi czasowe dla instrukcji MOV A, #data

Instrukcja: ADD A, address

Kod HEX wpisany do pamięci ROM: 25 01

Liczby HEX wpisany do pamięci RAM: 00 03

Opis: Pierwsze 2 kroki (8 cykli zegara) pobierają instrukcje z pamięci ROM i przekazują ją do CU. W 3 kroku zawartość PC zostaje przesłana do IAR (wiersz 5). W 4 kroku PC jest inkrementowany oraz zawartość komórki pamięci ROM, wskazywana przez IAR, zostaje przesłana do MAR. W 5 kroku zawartość ACC zostaje przesłana do TMP. W 6 kroku zawartość komórki pamięci RAM, wskazywana przez MAR, zostaje przesłana do ALU wraz z zawartością TMP. ALU wykonuje operacje dodawania i wynik jest zapisywany do ACC (wiersz 5).



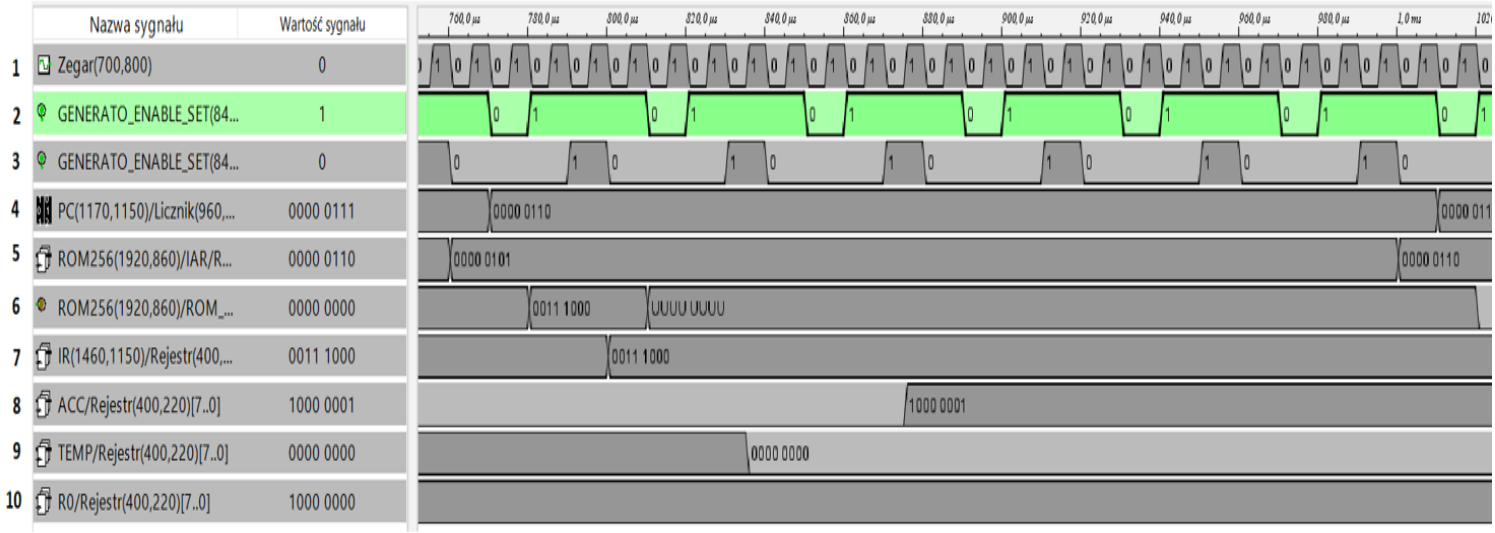
Ilustracja 19. Przebiegi czasowe dla instrukcji ADD A, address

Instrukcja: ADDC A, R0

Kod HEX wpisany do pamięci ROM: 74 80 78 80 28 38

Liczby HEX wpisany do pamięci RAM: brak

Opis: Przed wykonaniem instrukcji ADDC A, R0 zostały wykonane następujące rozkazy: 74 (MOV A, #data) przyjmujący argument 80, 78 (MOV R0, #data) przyjmujący argument 80 oraz 24 (ADD A, address). Dzięki temu flaga C = 1 oraz zawartość rejestru R0 jest równa 80. Pierwsze 2 kroki (8 cykli zegara) pobierają instrukcje z pamięci ROM i przekazują ją do CU. W 3 kroku zawartość ACC zostaje przesłana do TMP. W 4 kroku Zawartość TMP i R0 jest przesyłana do ALU, gdzie jest wykonana operacja dodawania wraz z jednym bitem pochodzącym od flagi C. Wynik operacji jest zapisany do ACC (wiersz 8). Krok 5 i 6 nie wpływa na wynik działania instrukcji.



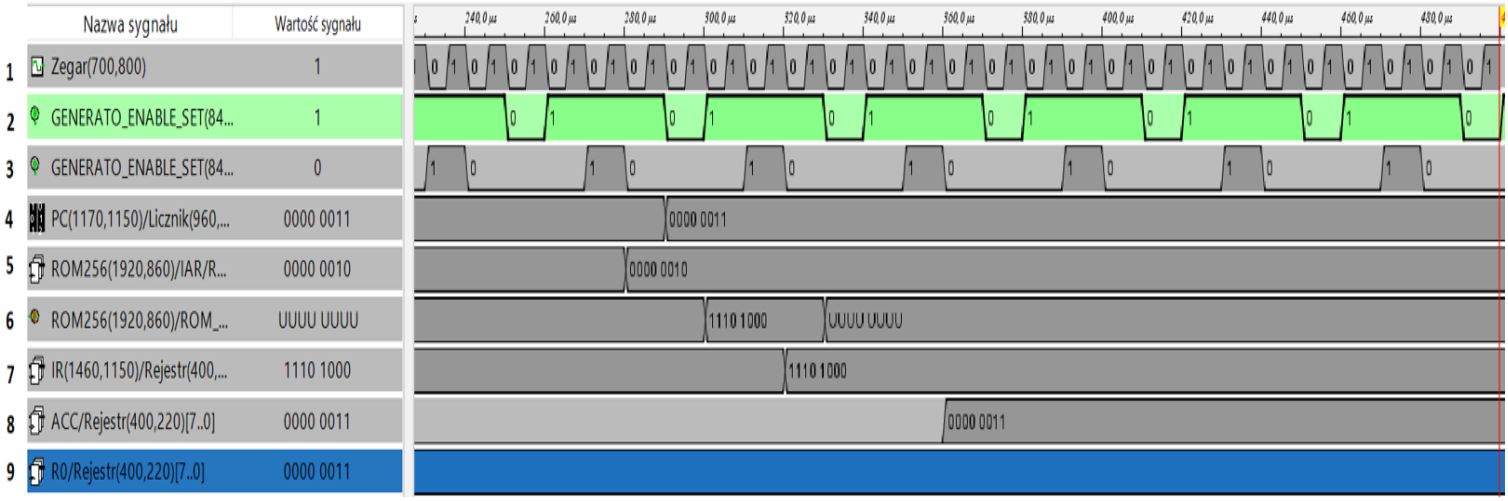
Ilustracja 20. Przebiegi czasowe dla instrukcji ADDC A, R0

Instrukcja: MOV A, R0

Kod HEX wpisany do pamięci ROM: 78 03 E8

Liczby HEX wpisany do pamięci RAM: brak

Opis: Przed wykonaniem instrukcji MOV A, R0 został wykonany rozkaz 78 (MOV R0, #data) przyjmujący jako argument natychmiastowy 03. Dzięki temu zawartość rejestru R0 jest równa 03. Pierwsze 2 kroki (8 cykli zegara) pobierają instrukcje z pamięci ROM i przekazują ją do CU. W 3 kroku zawartość rejestru R0 zostaje przesłana do ACC (wiersz 8). Krok 4,5 i 6 nie wpływa na wynik działania instrukcji.



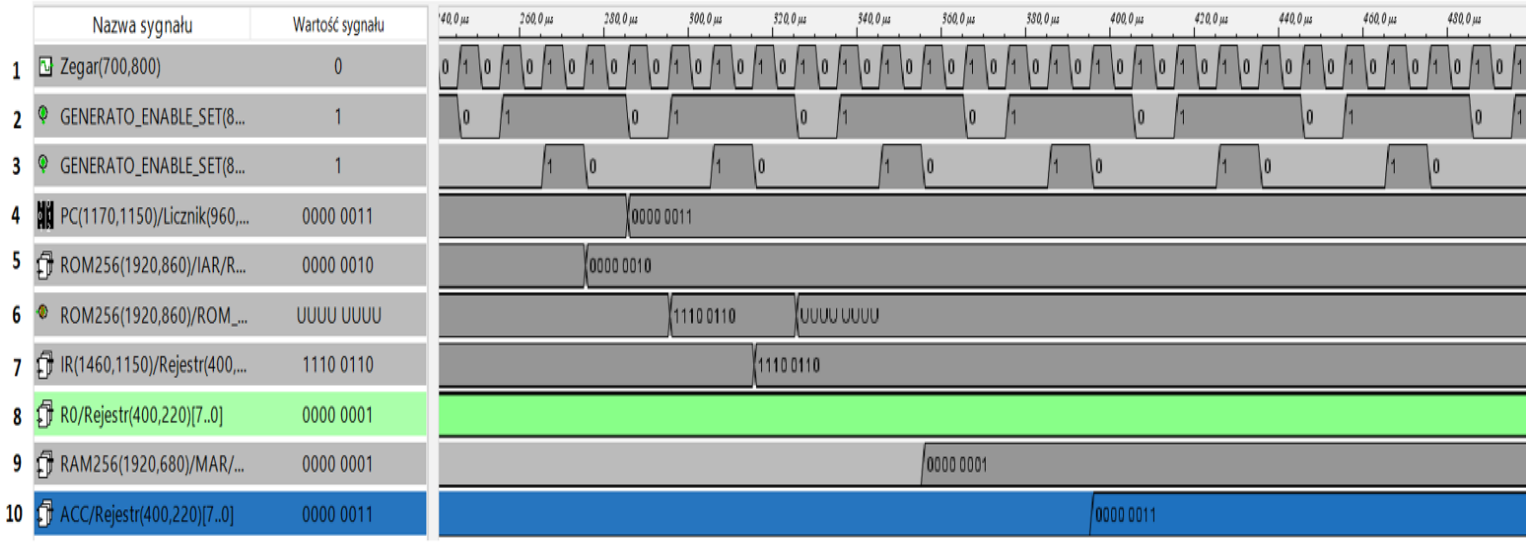
Ilustracja 21. Przebiegi czasowe dla instrukcji MOV A, R0

Instrukcja: MOV A, @R0

Kod HEX wpisany do pamięci ROM: 78 01 E6

Liczby HEX wpisany do pamięci RAM: 00 03

Opis: Przed wykonaniem instrukcji MOV A, R0 zostały wykonane rozkaz 78 (MOV R0, #data) przyjmujący argument 01. Dzięki temu zawartość rejestru R0 jest równa 01. Pierwsze 2 kroki (8 cykli zegara) pobierają instrukcje z pamięci ROM i przekazują ją do CU. W 3 kroku zawartość rejestru R0 zostaje przesłana do MAR. Zawartość komórki pamięci RAM wskazywana przez MAR jest przesyłana do ACC. Krok 5 i 6 nie wpływa na wynik działania instrukcji.



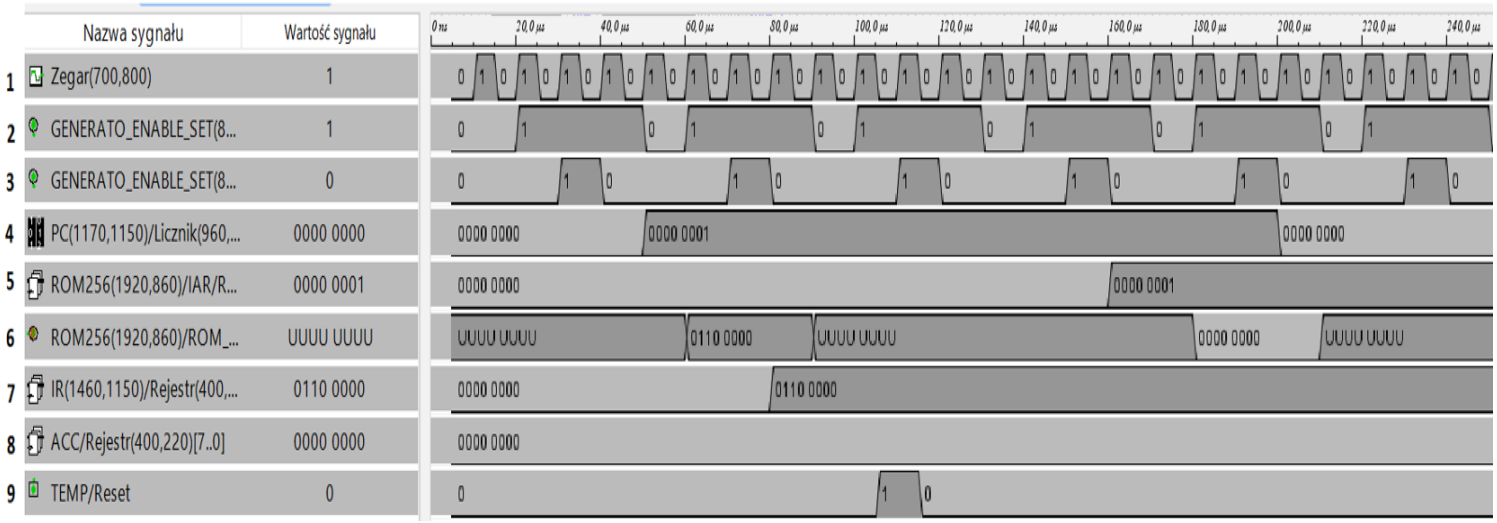
Ilustracja 22. Przebiegi czasowe dla instrukcji MOV A, @R0

Instrukcja: JZ address

Kod HEX wpisany do pamięci ROM: 60 00

Liczby HEX wpisany do pamięci RAM: brak

Opis: Pierwsze 2 kroki (8 cykli zegara) pobierają instrukcje z pamięci ROM i przekazują ją do CU. W 3 kroku TMP został zresetowany, a zawartość ACC została wypisana na magistrale. ALU pobiera zawartość ACC i porównuje zawartość TMP (0) z ACC. Porównanie nie zmienia rejestru flag, lecz wynik flagi eq jest zapisany w specjalnym przerzutniku w CU. W 4 kroku zawartość PC jest przesyłana do IAR. W 5 kroku w zależności od zawartości przerzutnika, przesyła zawartość komórki pamięci wskazywaną przez IAR do PC, jeśli przerzutnik posiada wartość 0 bądź nie przesyła, gdy zawartość przerzutnika ma wartość 1. Krok 6 nie wpływa na wynik działania instrukcji.



Ilustracja 23. Przebiegi czasowe dla instrukcji JZ address

6. Napotkane problemy i rozwiązania

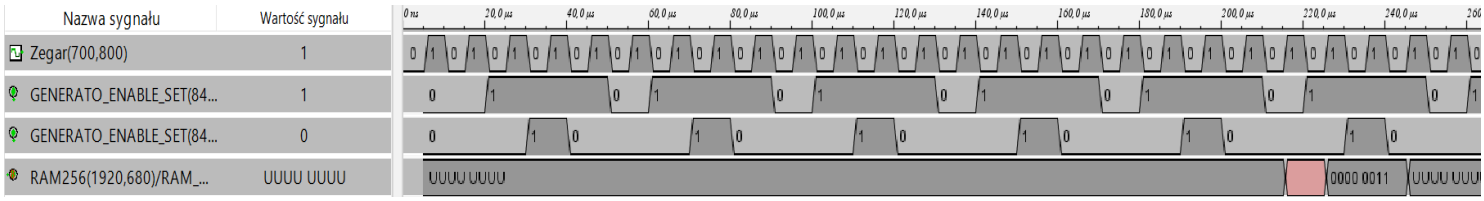
Podczas realizacji projektu napotkaliśmy wiele problemów. Większość z nich wynikało z małych błędów, często spowodowanych nieuwagą i udawało nam się je szybko rozwiązać. Zmierzyliśmy się jednak również z problemami, których rozwiązania nie były od razu oczywiste.

a. Wejścia zegarowe podłączone do innych sygnałów niż sygnały zegarowe

Błąd wynikał głównie z braku doświadczenia w projektowaniu bardziej złożonych układów cyfrowych. Rozwiązanie problemu polegało na dodaniu dodatkowych przerzutników, które odwzorowywały pracę układu bez występowaniu błędu.

b. Czerwona linia wychodząca z pamięci RAM

Problem polega na wypisywaniu na magistralę błędnego sygnału (czerwonego) podczas pierwszego korzystania z pamięci RAM. Błędna wartość zmienia się na właściwą po jednym cyklu zegarowym. Błąd ten wynika z wewnętrznej budowy gotowego bloku pamięci RAM przygotowanego przez Logisim, który potrzebuje dodatkowego cyklu zegarowego do ustalenia poprawnej wartości spod adresu podanego w MAR (*Memory Address Register*). Po przeanalizowaniu wykresów czasowych udało nam się ustalić, że wartość ta nigdy nie zostanie przyjęta przez żadną komórkę pamięci co, po konsultacji z prowadzącym, pozwoliło nam pozostawić to zjawisko w końcowej wersji projektu.



Ilustracja 24. Zjawisko pojawiania się błędnej wartości na magistrali przy pierwszym korzystaniu z pamięci RAM

c. Sposób dekodowanie rozkazów

Problem ten pojawił się na etapie projektowania jednostki sterującej. Początkowy pomysł zakładał zaprojektowanie dekodera rozkazów w taki sposób, aby bezpośrednio zamieniał dostarczony kod instrukcji na wyjściowy sygnał sterujący za pomocą serii bramek i dekoderów. Problem z takim rozwiązaniem polegał na tym, że zakłada on występowanie jakiegoś typu zależności w kodzie instrukcji między instrukcjami wykonywującymi podobne działania (np.

wszystkie instrukcje korzystające z ALU mają 7 bit ustawiony na „1”). Nie udało nam się takich zależności znaleźć w liście rozkazów mikrokontrolerów MCS-51. Zamiast tego zastosowaliśmy rozwiązanie wprowadzające wewnętrzną pamięć ROM, która zawiera długie słowa kontrolne pod adresami generowanymi na podstawie Opcodu i wykonywanego kroku.

d. Problem ze skokami

Problem z instrukcjami skoków pojawił się w jednej z końcowych faz projektu. Polegał on na złej kolejności wysyłaniu sygnałów do Program Countera, przez co nie wykonywał się zamierzony skok. PC przyjmował w tym samym momencie adres, pod który miał wykonać skok oraz sygnał, który informował o zmianie licznika. Aby naprawić ten błąd wprowadziliśmy dodatkowy przerzutnik, który zapamiętywał poprzedni stan sygnału SET i używał go, aby wywołać zmianę licznika PC, po przyjęciu nowego adresu.

7. Podsumowanie i wnioski

Zrealizowany projekt pozwolił nam pogłębić wiedzę na temat budowy i działania procesora 8051. Zgodnie z założeniem projektu, udało nam się stworzyć symulację procesora, który jest w stanie wykonać podstawowe operacje, takie jak transfer danych, działania arytmetyczne i logiczne oraz rozgałęzienie programu. Używając zaimplementowanych instrukcji, mimo konieczności użycia większej ilości rozkazów, jesteśmy w stanie stworzyć proste programy, jak np. generowanie n-tej liczby Fibonacciego dla małych n.

Wykonana symulacja pozwoliła nam na dogłębne przeanalizowanie przebiegu cyklu maszynowego. Mogliśmy zaobserwować wszystkie etapy pojedynczego cyklu – etap pobrania instrukcji, etap jej dekodowania oraz etap wykonania i ewentualne zapisanie wyników. Podczas planowania projektu nie uwzględniliśmy instrukcji dwucyklowych, przez co nie mogliśmy zaprezentować w pełni charakterystykę CISC procesora 8051.

Implementacja podstawowych elementów wchodzących w skład procesora, dzięki dostępnym gotowym blokom w aplikacji Logisim, nie sprawiła większych trudności. Zdecydowanie najtrudniejszym etapem była budowa jednostki centralnej. Wymagała ona zrozumienia podstawowych koncepcji związanych z działaniem procesora (włączanie i wyłączanie sygnałów SET i ENABLE w odpowiedniej kolejności) oraz wymyślenie sposobu automatyzacji pracy CU. Udało nam się to osiągnąć przez stworzenie specjalnego generatora sygnałów SET i ENABLE oraz stworzenie dekodera, opierającego się na odczytywaniu

sygnałów sterujących z wewnętrznej pamięci ROM spod wygenerowanych adresów opartych o aktualnie wykonywane instrukcje.

Projekt zrealizowaliśmy dzięki skoordynowanej, wspólnej pracy. Marcin Lenkiewicz zajmował się głównie szukaniem informacji na temat procesora oraz projektowaniem układów potrzebnych do jego działania. Mikołaj Pastuszek zajmował się implementacją projektów w aplikacji Logisim oraz odpowiadał za ewentualną naprawę błędów. Każdy etap projektu był wspólnie omawiany, w przypadku pojawienia się problemów, były one rozwiązywane wspólnie.

8. Bibliografia

1. Intel, "*MCS – 51 MICROCONTROLLER FAMILY USER'S MANUAL*", Feb 1994
2. Wikipedia, "*Intel 8051*", https://pl.wikipedia.org/wiki/Intel_8051
3. Antoni Sterna, "*Prezentacja 8051*", http://staff.iiar.pwr.wroc.pl/antoni.sterna/
4. M. H. H. Ichsan, W. Kurniawan, "*Design and implementation 8-bit CPU architecture on Logisim for undergraduate learning support*", Nov 2017
5. Ben Eater, "*8-bit CPU control logic: Part 1*", Apr 2017

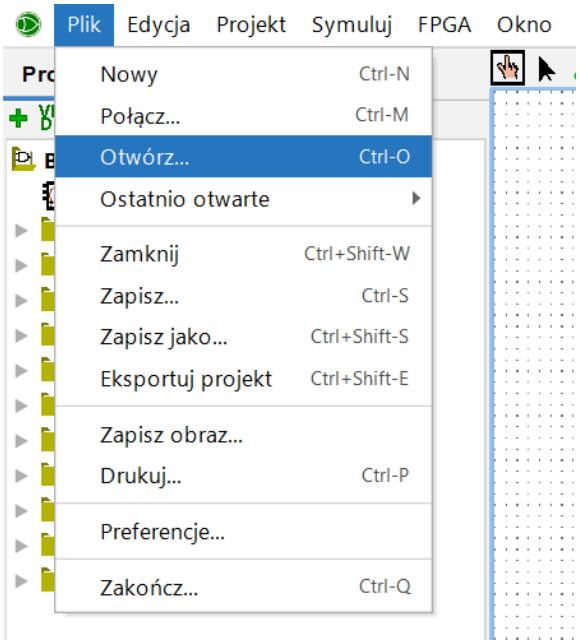
9. Dodatki

a. Instrukcja obsługi symulatora CPU

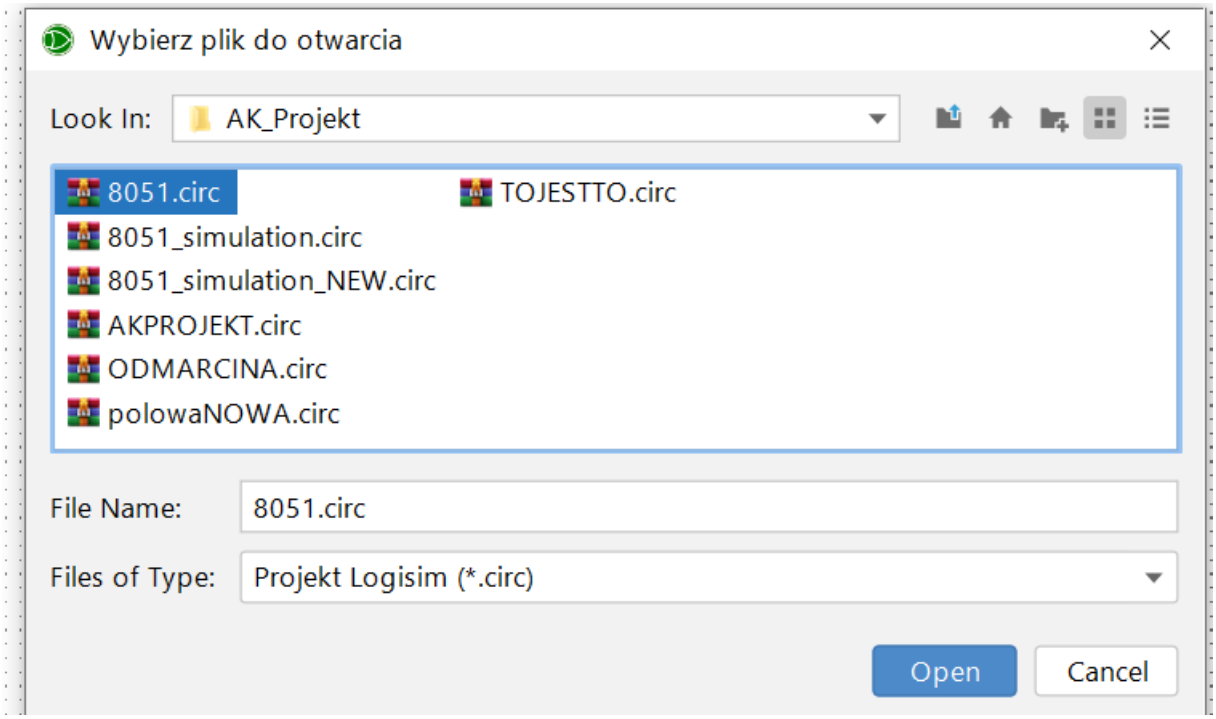
1. Użyć instalator dodanego w pliku zip. Dla systemu Windows: **logisim-evolution-3.7.2.msi** natomiast dla systemu Linux: **logisim-evolution_3.7.2-1_amd64.deb**.

2. Po zainstalowaniu programu, uruchomić go.

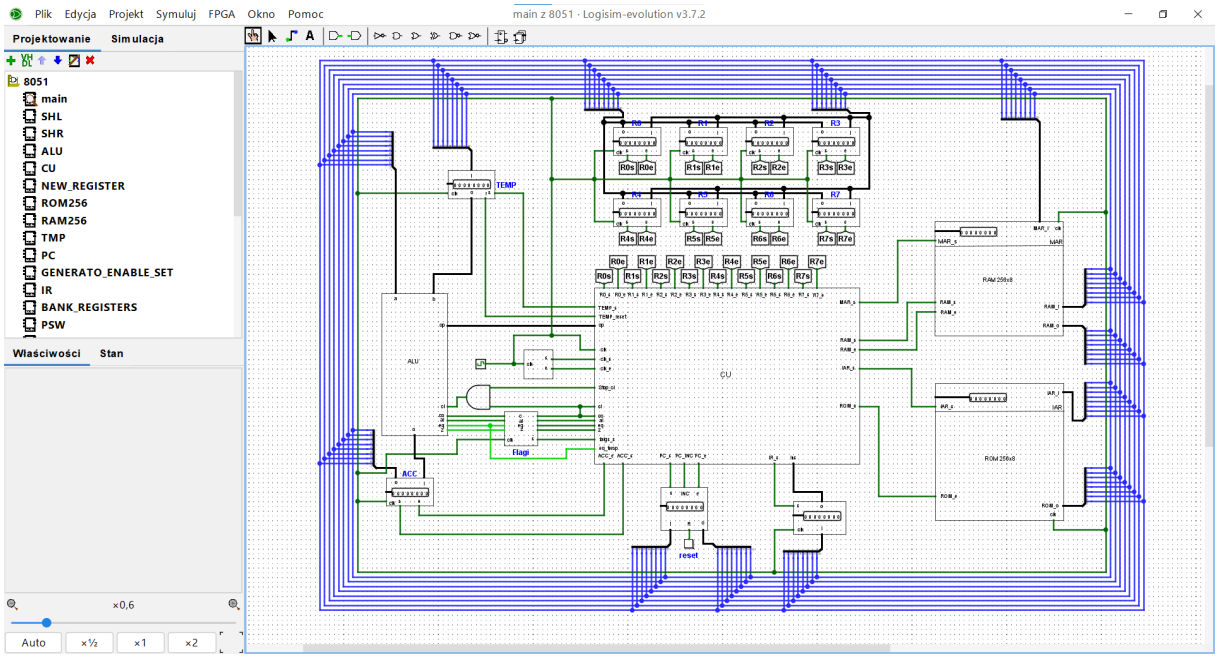
3. Kliknąć w prawym górnym rogu przycisk plik, następnie w rozminięciu otwórz.



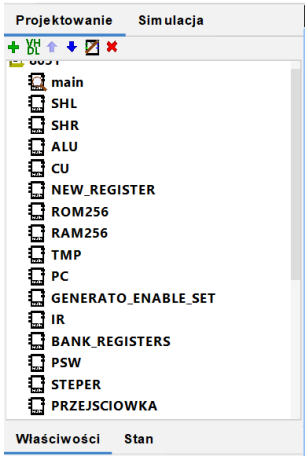
4. Po otworzeniu nowej karty znaleźć plik z projektem o nazwie **8051.circ**, a następnie kliknąć open.



Po otwarciu projektu powinien ukazać się taki obraz.



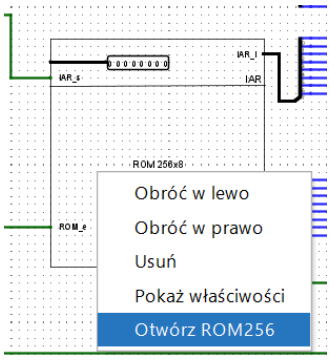
Po lewej stronie znajdują się wszystkie komponenty naszego procesora stworzone przez nas.



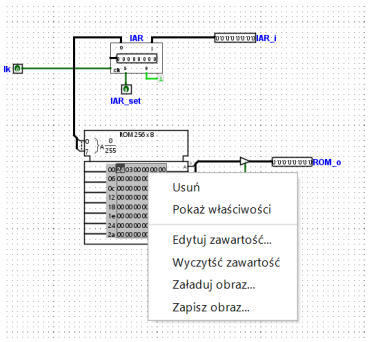
Klikając w każdy element można zobaczyć budowę poszczególne elementu. Main to główny obwód, gdzie jest stworzony cały CPU.

W Logisim-evolution istnieją dwa widoki, Projektowanie bądź Symulacja. W widoku Projektowanie można tworzyć układy. Natomiast w widoku Symulacji możemy w czasie rzeczywistym zobaczyć wszystkie elementy które biorą udział w symulacji.

5. Jeśli chcemy załadować kod programu należy kliknąć prawym przyciskiem na ROM a następnie wybrać opcje Otwórz ROM.



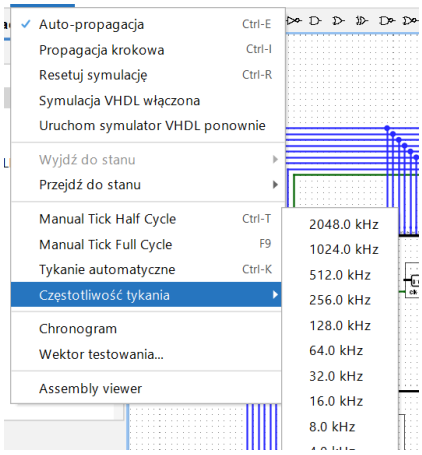
Następnie kliknąć prawym przyciskiem na ROM i wybrać opcje edytuj zawartość



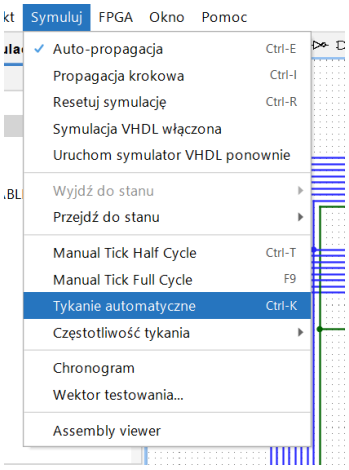
W tym miejscu zapisujemy nasze instrukcje i ich argumenty.

6. Jeśli chcemy zapisać coś do pamięci RAM należy postępować podobnie jak dla ROM tylko że na elementach pamięci RAM.

7. Aby włączyć symulację należy kliknąć na górnym pasku w Symulacja, a następnie wybrać opcje **częstotliwość tykania**. W tym momencie ustawimy częstotliwość naszego zegara.

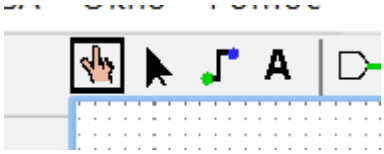


Następnie należy ponownie kliknąć w Symulacja po czym wybrać opcje **tykanie automatyczne**.

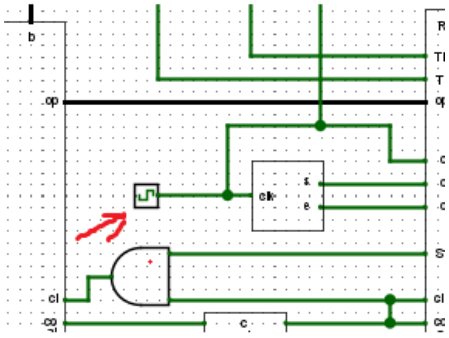


Jeśli chcemy wyłączyć tykanie automatyczne należy ponownie kliknąć w guzik **Tykanie automatyczne**.

Drugą opcją, mniej polecaną, jest obserwacja pracy procesora dla pojedynczych cykli zegara wtedy wybieramy kursor w formie ręki.



I przy wyłączonym tykaniu automatycznym klikamy na zegar. Jeden klik powoduje jedną zmianę stanu.



8. Podczas trwania symulacji możemy na bieżąco obserwować stan pojedynczych elementów procesora w środku. Wystarczy kliknąć prawym przyciskiem myszy na dany element, który nas interesuje i jeśli posiada możliwość podglądu pojawi się opcja **Otwórz *nazwa_elementu***.

b. Zbiór instrukcji wraz z rozpisanymi sygnałami sterującymi

W pliku w formacie zip załączamy również plik w aplikacji Microsoft Excel o nazwie **ZestawInstrukcji.xlsx**. Zawiera on dokładny opis wszystkich instrukcji.