PROCESOR Z80

Projekt i implementacja

Szymon Kluska (263974)
Patryk Hłond (259172)

Spis treści

Lista rysunków

Lista wykresów

Lista tabel

1.Streszczenie projektu

Projekt polegający na odwzorowaniu działania procesora Zilog Z80. Zbudowany układ symuluje działanie procesora i możliwość wykonania instrukcji z różnych grup rozkazów z wykorzystaniem różnych trybów adresowania zgodnych z dokumentacją Z80 [1]. Osiągnięto automatyzację procesora operującego na jednym wspólnym zegarze. Instrukcje operują na rejestrach, pamięci i wartościach natychmiastowych. Procesor jest zdolny do wykonania prostych programów.

2. Założenia i ograniczenia

2.1 Założenia

Głównym założeniem projektu było odwzorowanie instrukcji z różnych grup rozkazów i stworzenie wykresów czasowych ich działania. Zależało nam również na tym, aby procesor był zautomatyzowany i mógł wykonywać przy wykorzystaniu kodów instrukcji napisany przez użytkownika prosty program.
Do zaprojektowania układów cyfrowych wykorzystaliśmy program Logisim-evolution [2] w wersji 3.8.0 w języku angielskim. Jest na bieżąco rozwijany w przeciwieństwie do Logisim oraz działa poprawnie na różnych systemach operacyjnych. Można dostosowywać parametry komponentów, generować wykresy czasowe i zliczać ilość danych komponentów użytych w układzie.
Do odwzorowywania rozkazów korzystaliśmy z dokumentacji Z80 CPU User Manual UM008011-0816. Wspomagaliśmy się również tabelą z wszystkimi kodami instrukcji [3].
Jako narzędzie do kontroli wersji projektu wybraliśmy system kontroli wersji Git i repozytorium na GitHub.

2.2 Ograniczenia

Założyliśmy, że do zrealizowania celów projektu wystarczą instrukcje 24-bitowe zamiast standardowych dla Z80 32-bitowych. Instrukcje wczytywane są z pamięci ROM. Zaimplementowaliśmy tylko wybrane rejestry niezbędne do zrealizowania założonych celów.

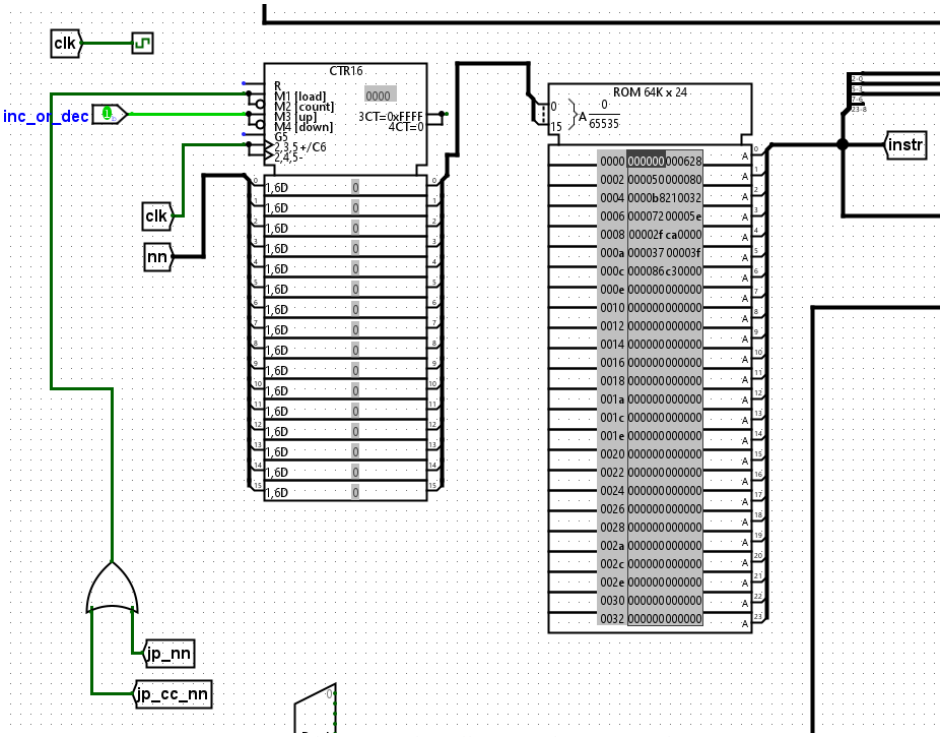
3. Sposób realizacji projektu

3.1 Uruchomienie i korzystanie projektu

1. Należy pobrać i zainstalować skompilowany Logisim-evolution [1] w wersji 3.8.0.
2. Otworzyć plik projektu (z80.circ).
3. W układzie „main” znajduje się całość procesora, a w pozostałych układach są podukłady użyte w układzie „main” jak na przykład „ALU” albo elementy wykonujące konkretne instrukcje użyte wewnątrz „ALU” na przykład „add_8bit”.
4. W głównym układzie „main” jest ROM z przygotowanym przykładowym programem już załadowanym, więc można zaobserwować wykonywanie się instrukcji krok po kroku, uruchamiając symulacje za pomocą Simulate > Manual Tick Full Cycle / Auto Tick Enabled. Zawartość pamięci ROM można zapisać / wczytać klikając prawym przyciskiem myszy na komponent.

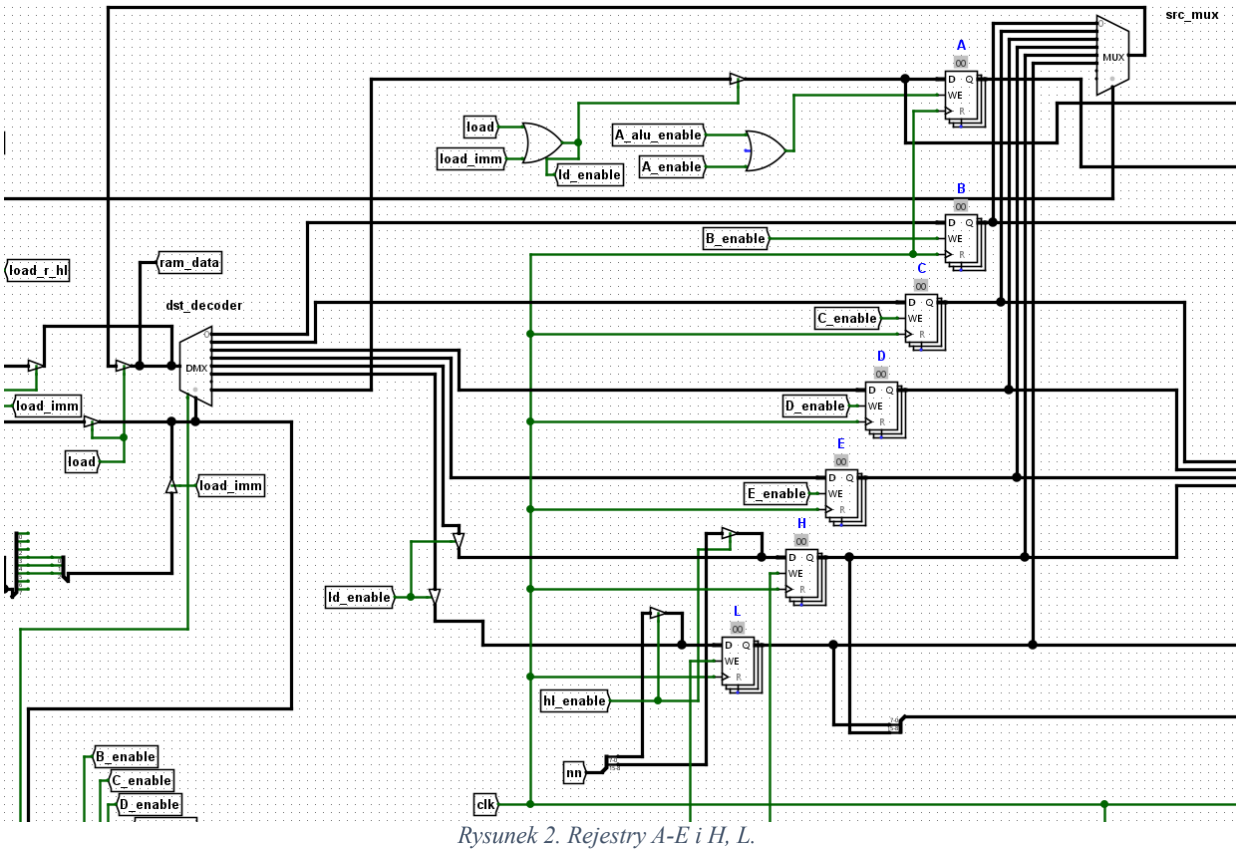
3.2 Struktura układów

Instrukcje programu są przechowywane w ROM o 24-bitowych komórkach pamięci podłączonym do licznika (ang. counter) wskazującego na adres [rysunek 1]. Instrukcja skoku (jp nn / jp cc nn) steruje licznikiem aby zmienić pozycje wskaźnika do danych instrukcji w pamięci ROM. Odwzorowuje to działanie Program Counter.



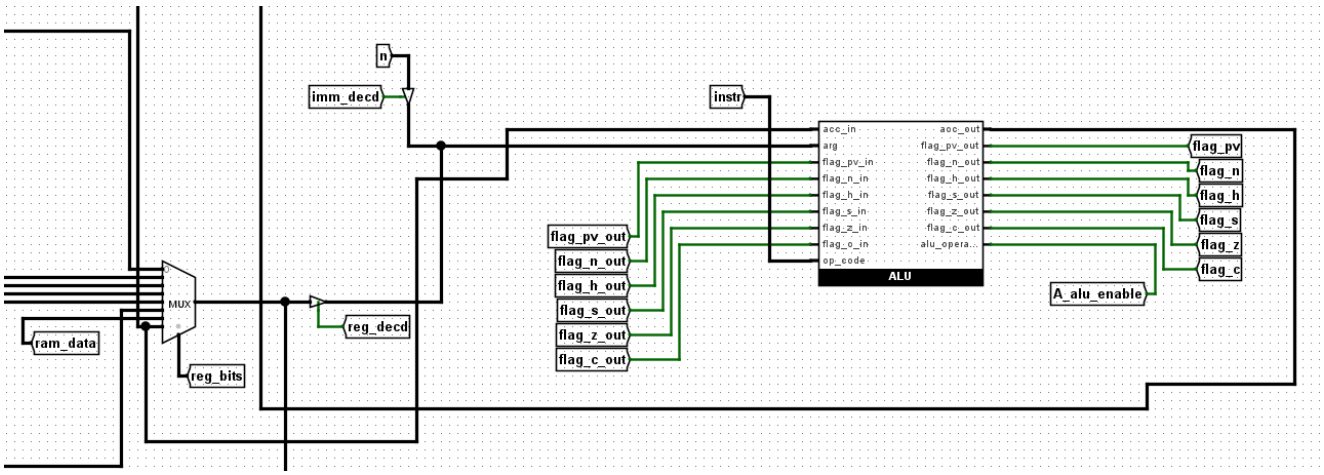
Rysunek 1. Licznik podłączony do ROM 24-bit.

Instrukcje dekodowane są rozdzielaczami (ang. splitters), prostymi bramkami NOR, AND i dekoderami. Jeżeli instrukcja wpisuje dane do któregoś z rejestrów, właściwy rejestr ma podany sygnał na „enable” i na magistrali jest podawany sygnał za pomocą demultipleksera. [rysunek 2].



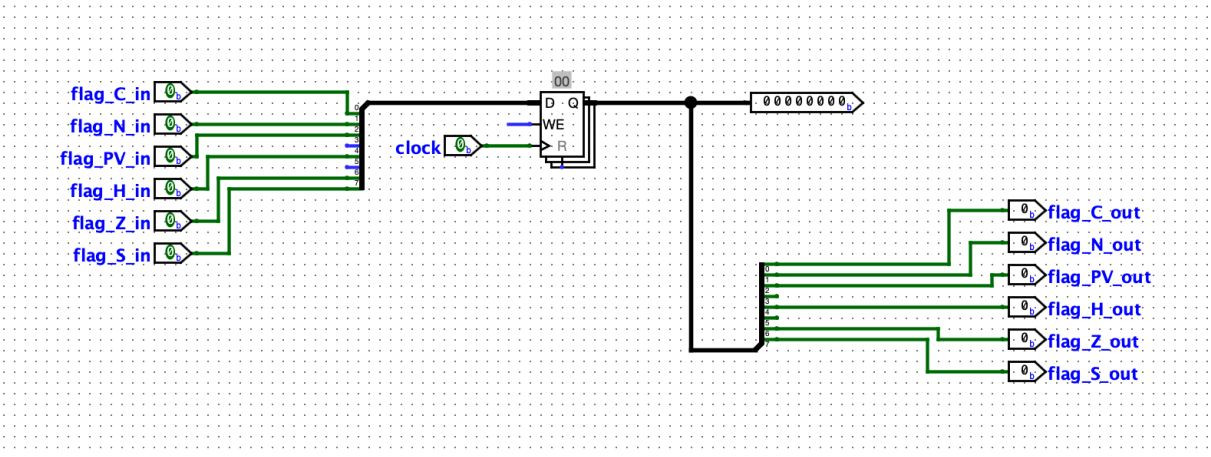
Rysunek 2. Rejestry A-E i H, L.

ALU ma wejście na zawartość akumulatora i wejście na dane z innego rejestru, pamięci albo wartości bezpośredniej. Do ALU wchodzą również aktualne wartości flag z rejestru F. ALU na wyjściu może zmieniać flagi i mieć wartość wychodzącą 8-bitową. Przy wejściu do ALU wykorzystano multiplekser, który przepuszcza dane z rejestrów lub pamięci odpowiednio do kodu instrukcji [rysunek 3].



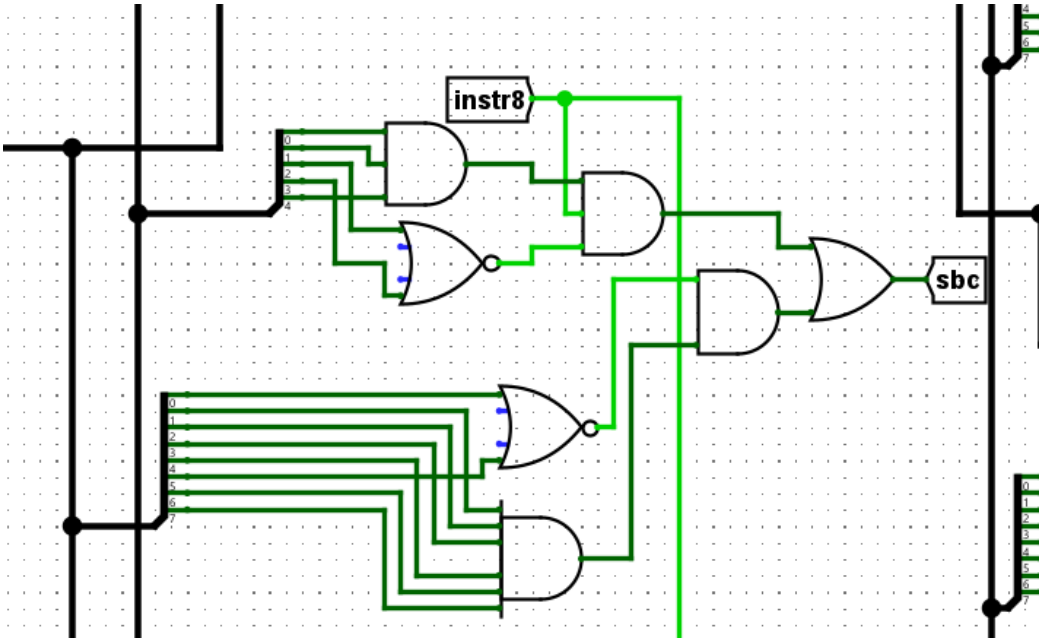
Rysunek 3. ALU z wejściami i wyjściami.

Rejestr F [rysunek 4], odpowiedzialny za przechowywanie wartości flag, ma wejścia podłączone do wyjść ALU i wyjścia podłączone do wejść ALU. Istnieje 6 flag w procesorze Z80, są to carry (C), add/subtract (N), parity/overflow (P/V), Half Carry (H), zero (Z) i sign (S). Flagi są na odpowiednich miejsach w rejestrze zgodnie z dokumentacją.



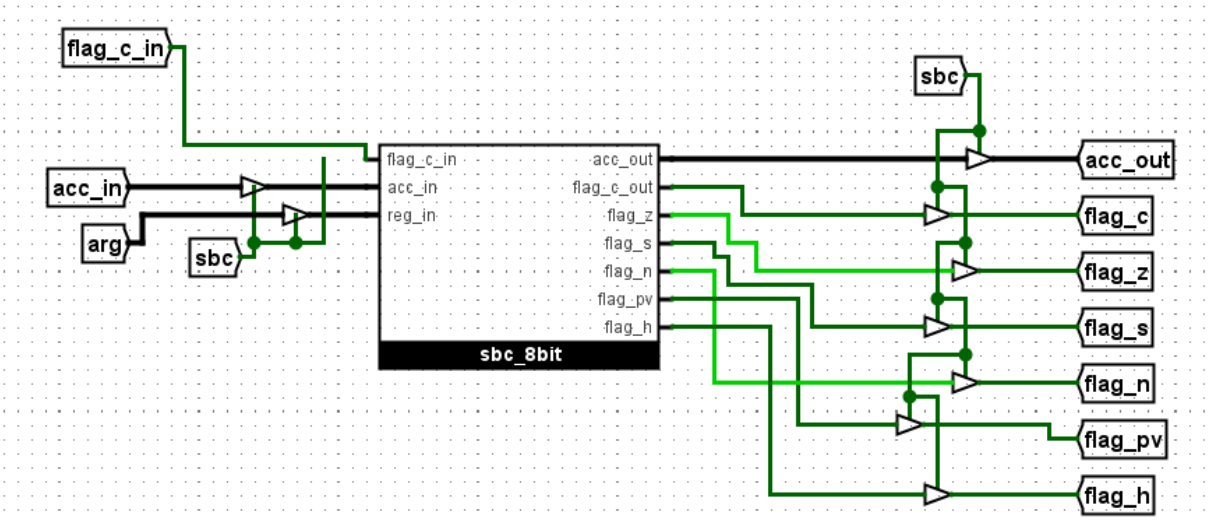
Rysunek 4. Rejestr F (flagi).

Wewnątrz ALU opcode jest dekodowany i może to aktywować sygnał kontrolujący dalsze wykonywanie się danej operacji [rysunek 5]. Oznacza to, że w danym momencie/stanie wykonuje się i podaje na wyjście wynik tylko jedna operacja arytmetyczna.



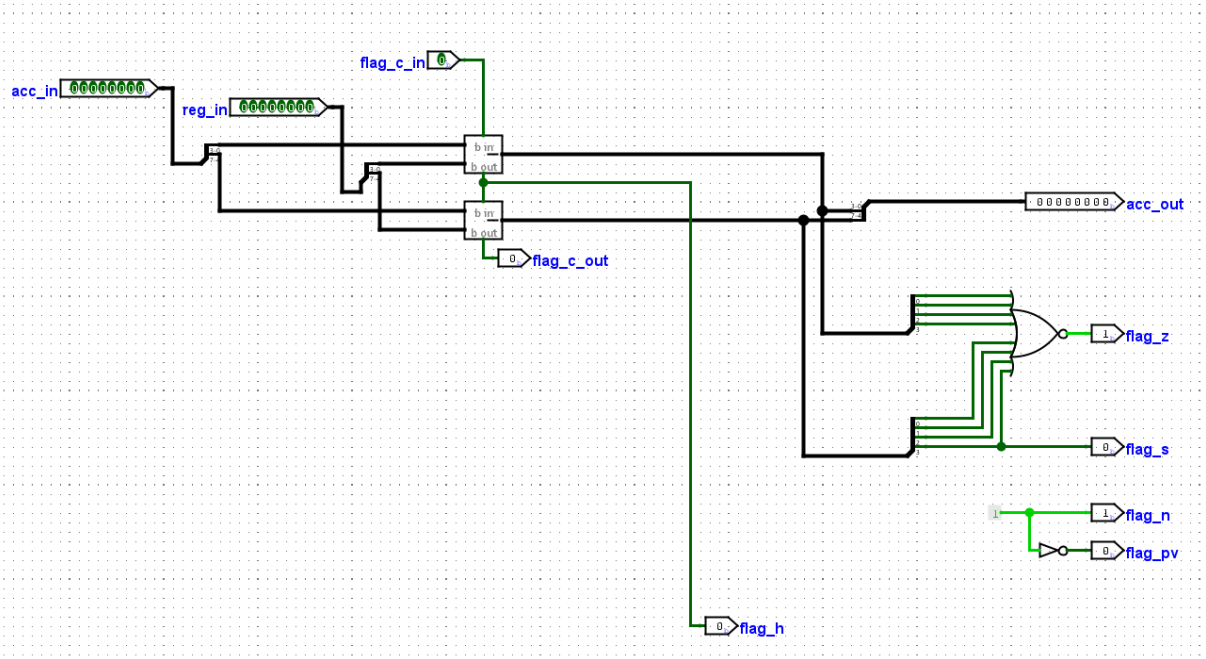
Rysunek 5. Dekodowanie instrukcji wewnątrz ALU by otrzymać sygnał wskazujący instrukcje do wykonania

Sygnał wskazujący instrukcje wykonania jest podłączony do bufora, który aktywuje przepływ danych na magistrali wejściowej i magistrali wyjściowej [rysunek 6].



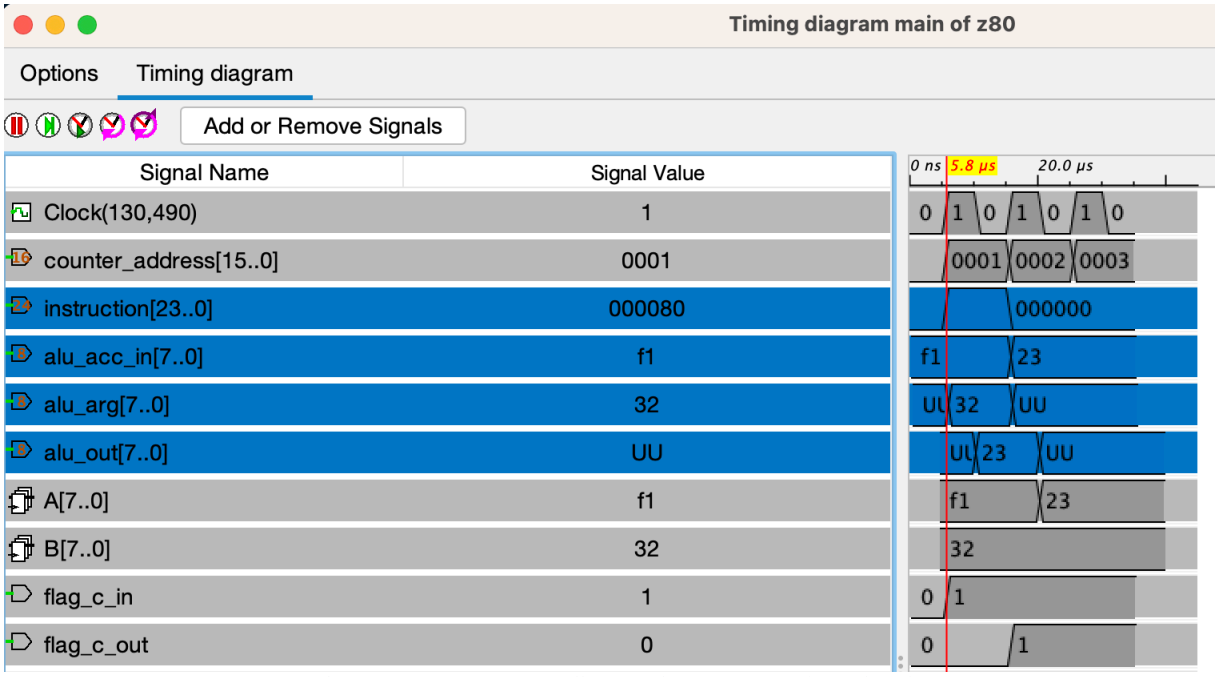
Rysunek 6. Obsługa konkretnej instrukcji wewnątrz ALU

Przykład implementacji instrukcji arytmetycznej [rysunek 7]. Układ wykonuje się po podaniu danych. Dopiero układ ALU za pomocą buforów i dekodowania instrukcji decyduje, czy dane zostaną przekazane do podukładu.



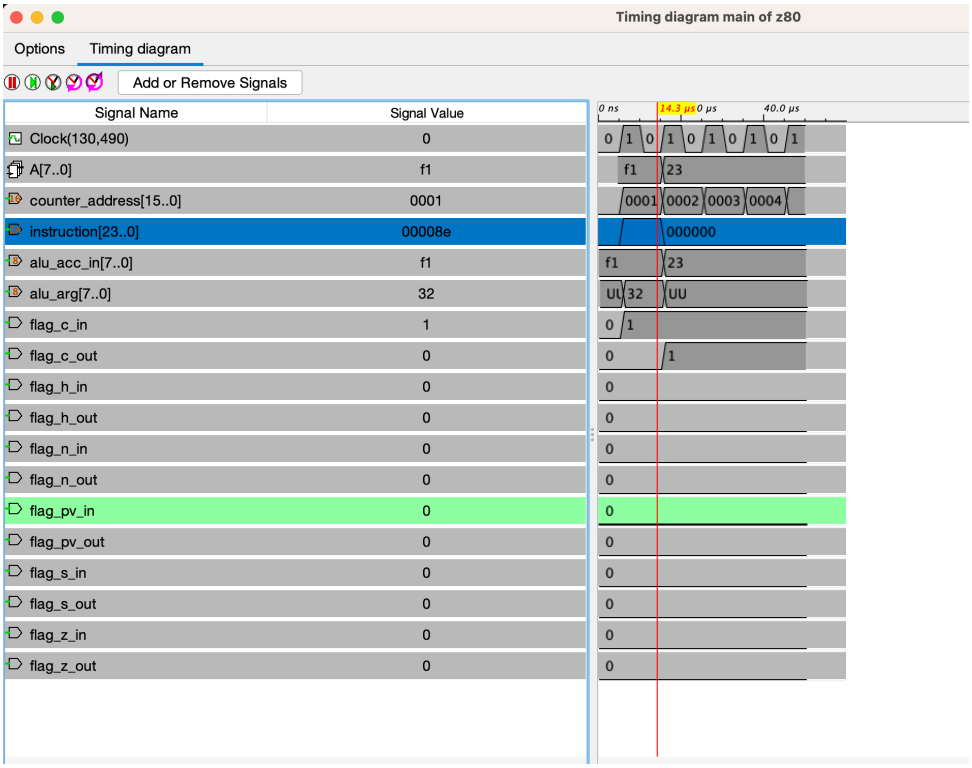
Rysunek 77. Implementacja układu sbc_8bit.

3.3 Wykresy czasowe



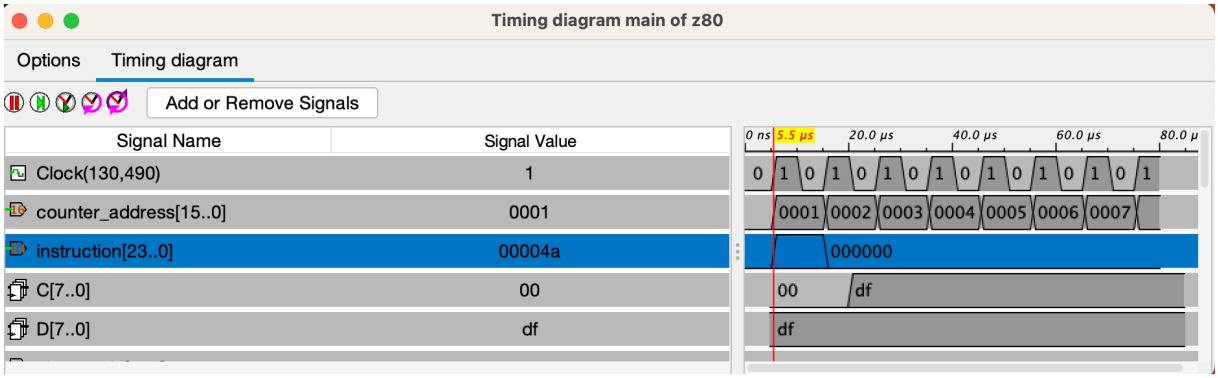
Wykres 1 Diagram czasowy dla instrukcji ADD A, B (opcode 80).

W pierwszym cyklu zegarowym zostaje pobrana instrukcja z pamięci ROM oraz przekazane argumenty do układu arytmetycznego. W ciągu pierwszego cyklu w ALU zostaje obliczona wartość oraz flagi. W drugim cyklu wartość wyjściowa z ALU zostaje wpisana do akumulatora oraz flagi na wyjściu z ALU do rejestru F.



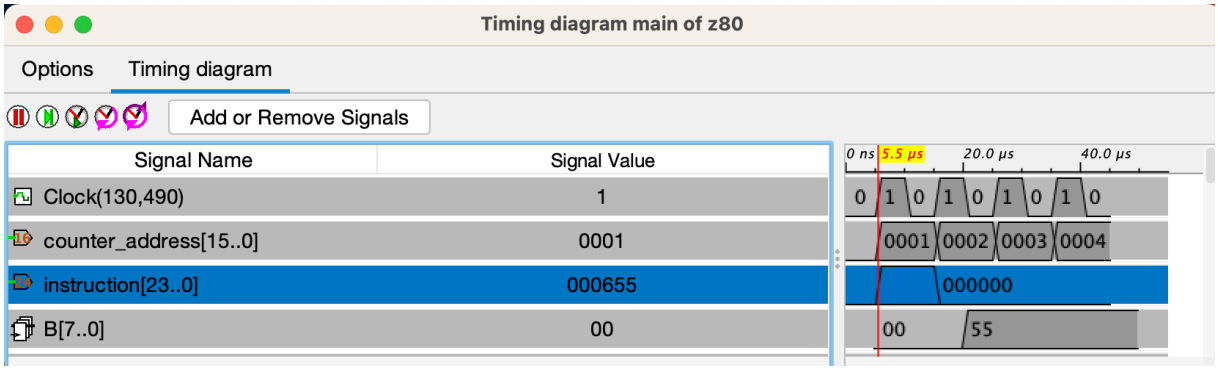
Wykres 2 Diagram czasowy dla instrukcji ADC A, (HL) (opcode 8E).

Dla instrukcji dodawania wartości z pamięci do akumulatora przebieg operacji nie różni się cyklami od dodawania wartości z rejestru do rejestru.



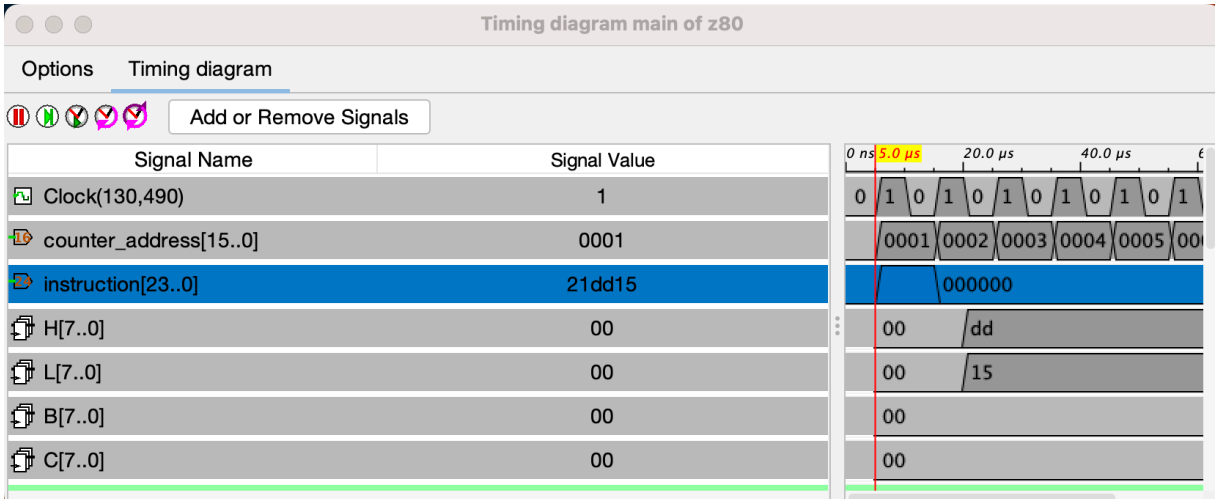
Wykres 3 Diagram czasowy dla instrukcji LD C, D (opcode 4A).

Skopiowanie zawartości rejestru D do C zajmuje dwa cykle zegarowe. W pierwszym cyklu zostaje pobrana instrukcja i przekazana na magistrale, a w drugim z magistrali do rejestru.



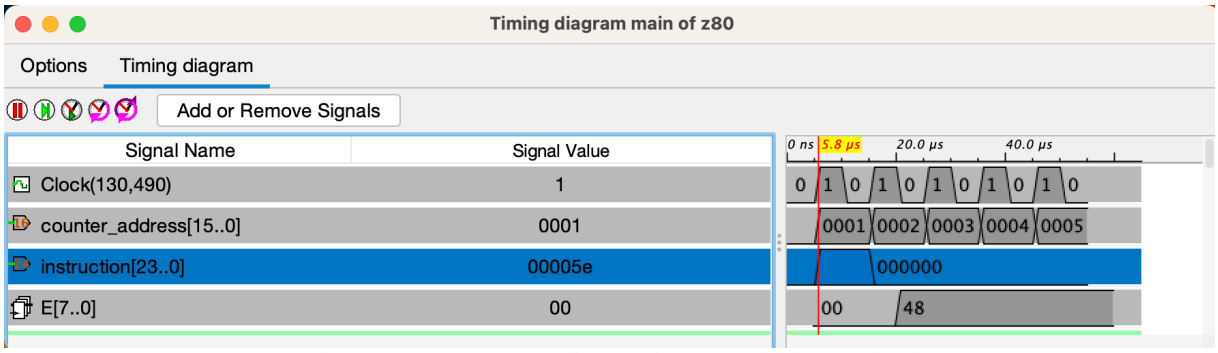
Wykres 4 Diagram czasowy dla instrukcji LD B, n (opcode 06 n).

Wpisanie wartości bezpośredniej n do B zajmuje dwa cykle zegarowe. W pierwszym cyklu zostaje pobrana instrukcja i przekazana na magistrale wraz z wartością n, która zostaje wpisana do rejestru B w trakcie drugiego cyklu zegara.



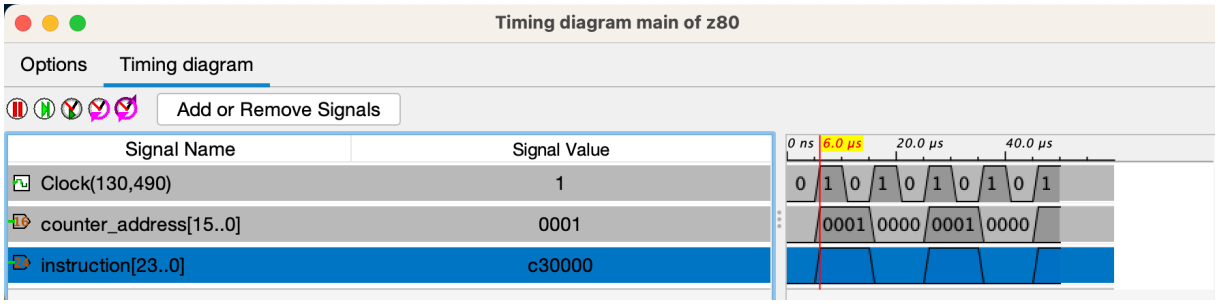
Wykres 5 Diagram czasowy dla instrukcji LD dd, nn (opcode 21 nn dla rozkazu 21dd15).

Wartość nn zostaje wpisana w trakcie drugiego cyklu odpowiednio do pary rejestrów HL.



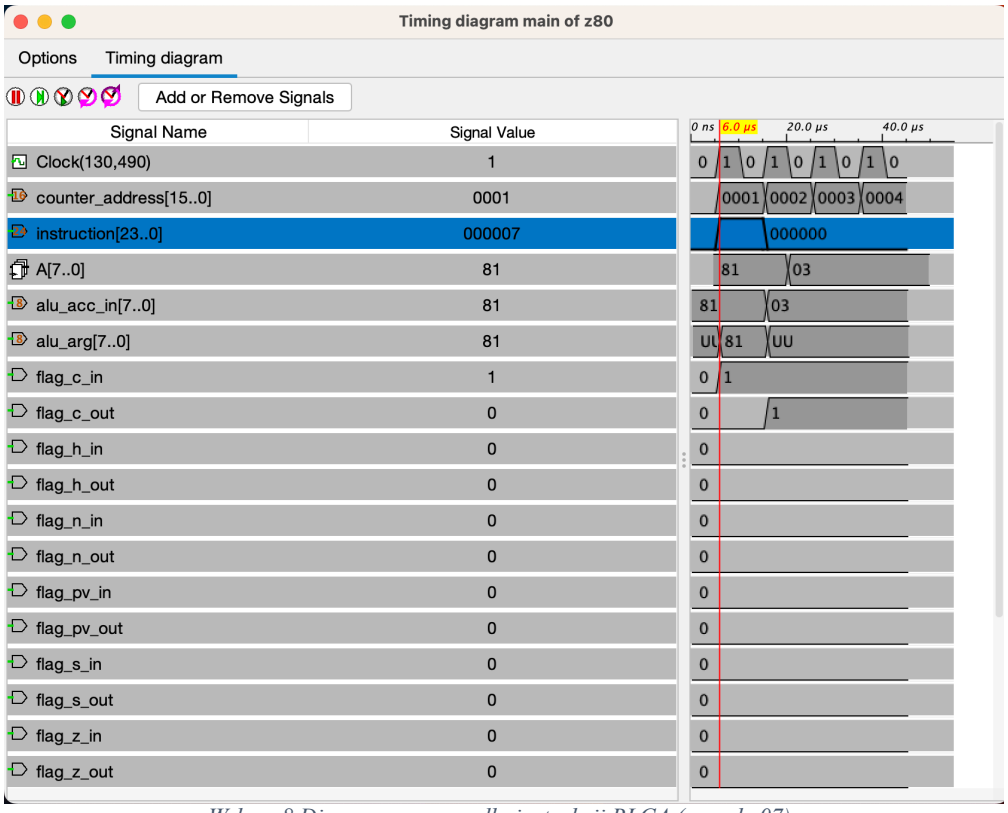
Wykres 6 Diagram czasowy dla instrukcji LD E, (HL) (opcode 5e).

Skopiowanie zawartości pamięci pod wskazanym przez HL adresem zajmuje dwa cykle zegara tak jak pozostałe badane instrukcje load.



Wykres 7 Diagram czasowy dla instrukcji JP nn (opcode C30000)

Skok bezwarunkowy JP nn wykonuje się w ciągu jednego cyklu zegara. Widoczne są skoki licznika do 0000.

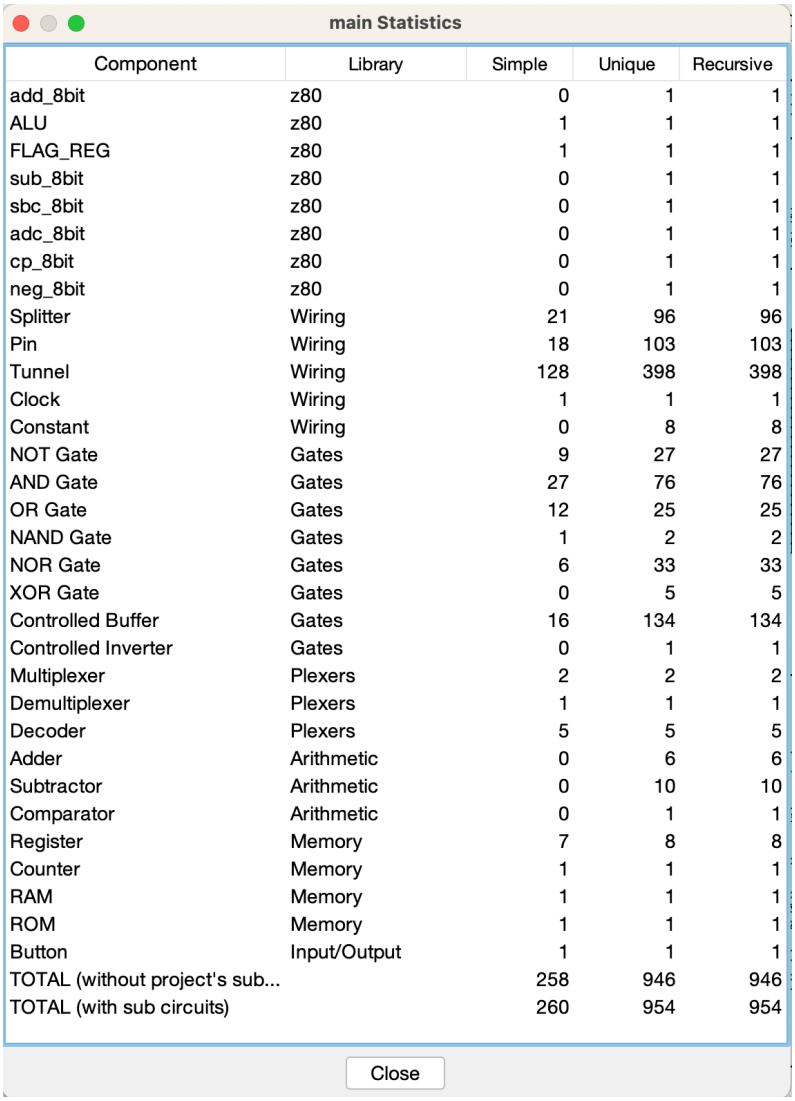


Wykres 8 Diagram czasowy dla instrukcji RLCA (opcode 07).

Procesor według wykresów czasowych dla instrukcji load i arytmetycznych jest w stanie wczytać i odkodować instrukcje w czasie pierwszego cyklu, a w czasie drugiego wpisać wyniki instrukcji na odpowiednie wejścia i pobrać kolejną instrukcje. Dla instrukcji skoku bezwarunkowego w ciągu jednego cyklu jest skok do innego adresu pamięci ROM.

3.3 Analiza złożoności

Ze względu na wykorzystanie gotowych układów z biblioteki Logisim-evolution takich jak dekodery o różnych wielkościach adresów, gotowych układów pamięci lub układów arytmetycznych nie jesteśmy w stanie oszacować dokładnej powierzchni układu procesora. Logisim-evolution daje możliwość zliczenia występować danych układów i bramek w bieżącym układzie nie wliczając tzw. black boxów [4] czyli podukładów (simple) i wliczając elementy w podukładach (unique).



Component	Library	Simple	Unique	Recursive
add_8bit	z80	0	1	1
ALU	z80	1	1	1
FLAG_REG	z80	1	1	1
sub_8bit	z80	0	1	1
sbc_8bit	z80	0	1	1
adc_8bit	z80	0	1	1
cp_8bit	z80	0	1	1
neg_8bit	z80	0	1	1
Splitter	Wiring	21	96	96
Pin	Wiring	18	103	103
Tunnel	Wiring	128	398	398
Clock	Wiring	1	1	1
Constant	Wiring	0	8	8
NOT Gate	Gates	9	27	27
AND Gate	Gates	27	76	76
OR Gate	Gates	12	25	25
NAND Gate	Gates	1	2	2
NOR Gate	Gates	6	33	33
XOR Gate	Gates	0	5	5
Controlled Buffer	Gates	16	134	134
Controlled Inverter	Gates	0	1	1
Multiplexer	Plexers	2	2	2
Demultiplexer	Plexers	1	1	1
Decoder	Plexers	5	5	5
Adder	Arithmetic	0	6	6
Subtractor	Arithmetic	0	10	10
Comparator	Arithmetic	0	1	1
Register	Memory	7	8	8
Counter	Memory	1	1	1
RAM	Memory	1	1	1
ROM	Memory	1	1	1
Button	Input/Output	1	1	1
TOTAL (without project's sub...		258	946	946
TOTAL (with sub circuits)		260	954	954

Rysunek 8 Statystyki głównego układu

3.4 Zaimplementowane instrukcje

Instrukcje do zaimplementowania były wybrane, aby było minimalnie instrukcji potrzebnych do stworzenia jakiegoś prostego programu wykonującego się automatycznie. Instrukcje te były wybierane również na podstawie różnych grup rozkazów, aby były jakieś z możliwie jak najwięcej grup. W dodatku, aby instrukcje posiadały różne argumenty w tym dane z rejestrów, pamięci lub argumenty natychmiastowe.
Z takimi więc kryteriami, ostatecznie zaimplementowano instrukcje ładujące dane, wykonujące różne operacje arytmetyczne i przesuwne na danych i instrukcje skoków by móc tworzyć pętle. Wszystkie konkretne instrukcje zaimplementowane są widoczne w tabeli poniżej [tabela 1].

r – rejestr (A, B, C, D, E, H, L)
r' – drugi inny rejestr
n / nn – argument natychmiastowy 8-bitowy / 16-bitowy
(HL) – adres pamięci z pary rejestrów H i L
cc - warunek

1. Tabela zaimplementowanych instrukcji

Grupa	Instrukcja	Argumenty	Opis
8-bit load	LD	r, n	Wpisuje wartość n do rejestru r
	LD	r, r'	Kopiuje zawartość r' do r
	LD	(HL), r	Kopiuje zawartość r do miejsca w pamięci wskazanego przez parę rejestrów HL
	LD	r, (HL)	Kopiuje zawartość miejsca w pamięci wskazanego przez parę rejestrów HL do rejestru r
16-bit load	LD	dd, nn	Wczytuje wartość nn do pary rejestrów (w projekcie zaimplementowano tylko dla HL)
8-bit arithmetic	ADD	A, r	Dodanie zawartości r do akumulatora. Wynik w akumulatorze.
	ADD	A, (HL)	Dodanie zawartości miejsca w pamięci wskazanego przez HL do akumulatora. Wynik w akumulatorze.
	ADD	A, n	Dodanie wartości bezpośredniej do akumulatora. Wynik w akumulatorze.
	ADC	A, r	Dodanie wartości r do akumulatora z uwzględnieniem przeniesienia (flagi c). Wynik w akumulatorze.
	ADC	A, (HL)	Dodanie zawartości miejsca w pamięci wskazanego przez HL do akumulatora z uwzględnieniem przeniesienia (flagi c). Wynik w akumulatorze.
	SUB	A, r	Operacja odejmowania zawartości r od akumulatora. Wynik w akumulatorze.
	SBC	A, r	Operacja odejmowania zawartości r od akumulatora z uwzględnieniem pożyczki (flaga c). Wynik w akumulatorze.
	AND	r	Operacja logiczna AND rejestru r i akumulatora. Wynik w akumulatorze.
	OR	r	Operacja logiczna OR rejestru r i akumulatora. Wynik w akumulatorze.
	XOR	r	Operacja logiczna XOR rejestru r i akumulatora. Wynik w akumulatorze.
	CP	r	Porównanie zawartości akumulatora i rejestru r oraz ustawienie odpowiednich flag.
General purpose arithmetic and CPU control	CPL		Zawartość rejestru A zostaje odwrócona.
	NEG		Zawartość rejestru A zostaje zanegowana.
	CCF		Flaga carry rejestru F zostaje odwrócona.
	SCF		Flaga carry rejestru F zostaje ustawiona.
	NOP		Procesor nie wykonuje żadnej instrukcji w czasie tego cyklu zegarowego.
Rotate and shift	RLCA		Zawartość akumulatora jest obracana w lewo o jedną pozycje. Najbardziej znaczący bit jest kopiowany do flagi carry oraz na bit zerowy (najmniej znaczącego).
	RLA		Zawartość akumulatora jest obracana w lewo jedną pozycje do flagi carry, a wcześniejsza zawartość tej flagi zostaje skopiowana do bitu zerowego.
	RRCA		Zawartość akumulatora jest obracana w prawo o jedną pozycje. Najmniej znaczący bit jest kopiowany do flagi carry oraz na bit siódmy (najbardziej znaczący)
	RRA		Zawartość akumulatora jest obracana w prawo jedną pozycje do flagi carry, a wcześniejsza zawartość tej flagi zostaje skopiowana do bitu siódmego (najbardziej znaczącego).
Jump	JP	nn	Skok bezwarunkowy do miejsca w liczniku wykonywania programu (PC) na pozycje określoną w nn.
	JP	cc, nn	Skok warunkowy do miejsca w liczniku wykonywania programu (PC) na pozycje określoną w nn jeśli warunek cc jest spełniony.

Flagi są zmieniane odpowiednio jak w dokumentacji Z80.

4. Podsumowanie

Budowa procesora Z80 była na początku trudna. Dużo czasu zostało poświęcone na znalezienie odpowiednich źródeł informacji oraz przejrzenie dokumentacji. Im prace nad projektem postępowały tym łatwiej było implementować kolejne rozwiązania i instrukcje.

Co można zmienić?

- Na początku budowania procesora planowano podzielić go na oddzielne podukłady (jak układ ALU i FLAG_REG). Miało to na celu uporządkowanie i zmniejszenie ilości komponentów w głównym układzie procesora. Przed zaimplementowaniem zautomatyzowanego wyboru instrukcji, dla ręcznego wpisywania wartości na magistrali to działało, lecz im więcej instrukcji dokładano tym bardziej komplikowało to obsługiwanie na zewnątrz i wewnątrz komponentu ALU. Możliwe, że gdyby z doświadczeniem po zakończeniu projektu od początku budowano procesor to nie wybrano by dzielenia go na podukłady.
- Obsługa instrukcji wybieranych z ROM została wykonana w formie bramek logicznych (AND dla bitów 1 i NOR dla bitów 0). Można byłoby to zastąpić odpowiednimi dwoma dekoderami i AND. Przy realizacji większej ilości instrukcji byłoby to lepsze rozwiązanie. Dla mniejszej ilości i łatwiejszego testowania zostawiliśmy w formie bramek.

5. Bibliografia

[1] Zilog. (2016). *Z80 CPU User Manual UM008011-0816.* Retrieved from zany80: https://zany80.github.io/documentation/Z80/UserManual.html

[2] Hutchens et al. (n.d.). *logisim-evolution*. Retrieved from Github: https://github.com/logisim-evolution/logisim-evolution

[3] Toaster, D. (n.d.). *Z80 opcode instructions*. Retrieved from ClrHome: https://clrhome.org/table/

[4] Burch, C. (2011). *Logisim Project Menu*. Retrieved from cburch logisim: http://www.cburch.com/logisim/docs/2.7/en/html/guide/menu/project.html