Projekt i implementacja procesora Z80

Michał Bernacki-Janson, Justyna Bułach
Termin grupy: Środa 13:15 TN
Data: 27.05.2023

Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kurs: Architektura Komputerów 2 Projekt
Prowadzący: dr hab. inż. Tadeusz Tomczak

Spis treści

Streszczenie

Głównym celem poniższego projektu było zaprojektowanie procesora Zilog Z80 zgodnie z instrukcją. Mimo ograniczenia czasowego udało się zaimplementować znaczącą część instrukcji, zachowując zgodność z oryginalną konstrukcją procesora. Wzięliśmy za cel modularność i przejrzystość projektu, co można zauważyć w samej jego konstrukcji. Jedynymi przeszkodami na drodze do dalszego rozszerzenia możliwości urządzenia są ograniczenie czasowe i braki w wiedzy, których usunięcie pozwoliłoby na znaczącą optymalizację działania mikroprocesora jak i zaimplementowanie wszelkich brakujących rozkazów. Wykonanie owego projektu pozwoliło nam sprawdzić własne umiejętności.

Wprowadzenie

Zadaniem projektowym było zaprojektowanie i stworzenie procesora Z80 w jego podstawowej wersji. Z uwagi na ograniczony czas niemożliwym było zaimplementowanie wszystkich funkcji procesora, stąd też ograniczono działanie procesora do jego najważniejszych funkcji i flag. W projekcie zastosowano autorskie kody wewnętrzne oraz kody zewnętrzne zgodne ze standardami procesora. Aby urządzenie funkcjonowało został stworzony od podstaw automat sterujący. W dalszej części dokumentu będzie przedstawiony sposób działania, konstrukcja implementacji procesora, osiągi urządzenia, wnioski oraz sugerowane poprawki.

Do konstrukcji procesora użyliśmy programu Logisim Evolution (v3.8.0)

Zilog Z80 to mikroprocesor wydany w lipcu 1976 przez firmę ZiLOG, powstały na fali entuzjazmu po wydaniu procesora Intel 8080. To właśnie część pracowników filmy Intel stoi za powstaniem firmy ZiLOG i samego procesora.
Mikroprocesor posiada 8-bitową magistralę danych, 16-bitową magistralę adresową z możliwością zaadresowania 64 kB pamięci RAM i obszaru 64 kB przestrzeni in/out oraz zasilanie i poziomy logiczne zgodne ze standardem TTL.
Cechy charakterystyczne Z80:

- wszystkie sygnały sterujące i obie magistrale są dostępne wprost (bez konieczności multipleksowania)
- 158 wbudowanych rozkazów, z czego prawie połowa jest zgodna z Intel 8080
- obszerny zestaw rejestrów wewnętrznych ogólnego przeznaczenia wraz z zestawem alternatywnych rejestrów i rejestrami indeksowymi
- zestaw rozkazów operujących na 16-bitowych danych

Metodologia

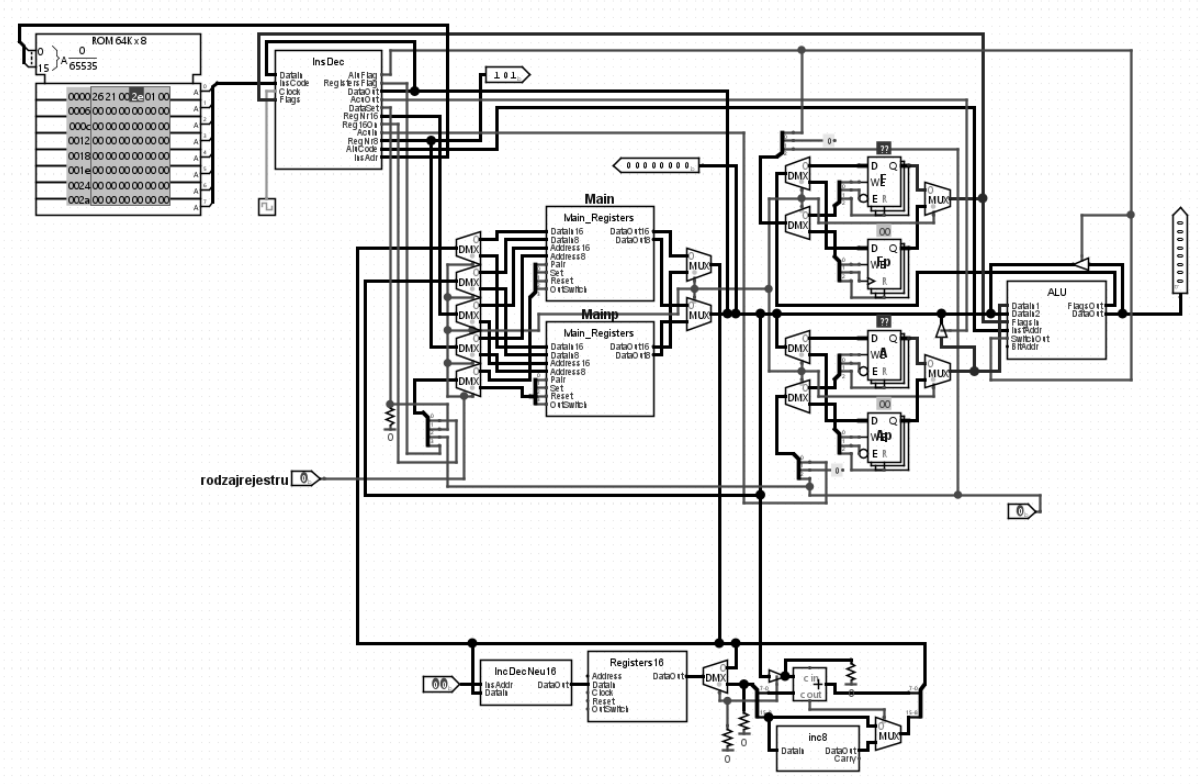
Początkowo wymagane było zapoznanie z ogólną konstrukcją samego procesora i utworzenie szkieletu samego projektu. Od początku było wiadome, że nie będzie możliwe zaimplementowanie wszystkiego, więc początkowy szkielet musiał być jak najprostszy. Sam układ procesora należy do tych prostych, zawierających dwa zestawy rejestrów: zwykły i alternatywny. Rejestry mają dwa tryby zapisu: podstawowy, jak i łączony, tworzący z dwóch rejestrów 8 bitowych, jeden 16 bitowy. Kolejnym krokiem była budowa ALU, jak i systemu flag, zapisywanych do zewnętrznego rejestru Flag. Po stworzeniu szkieletu utworzono pierwszy dekoder instrukcji wraz z automatem odpowiadającym za jej wykonywanie. W dalszej kolejności implementowano coraz to bardziej skomplikowane instrukcje, wymagające większej ilości cykli oraz działań na dodatkowej pamięci RAM.

Od początku wiedzieliśmy, że nie będziemy w stanie zbudować całego procesora, wzorując się więc na instrukcji zaczęliśmy kolejno dodawać funkcje. Dodatkowo postanowiliśmy podzielić system na podelementy np. rejestry. w ten sposób zachowując przejrzystość całości i zgodność ze schematami.

Rezultaty i obserwacje

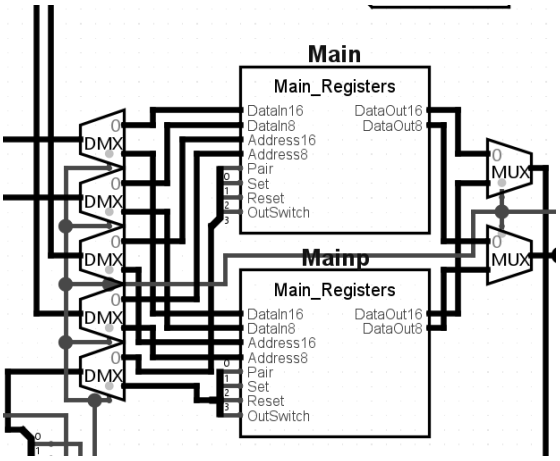
Konstrukcja układu:

Układ główny:



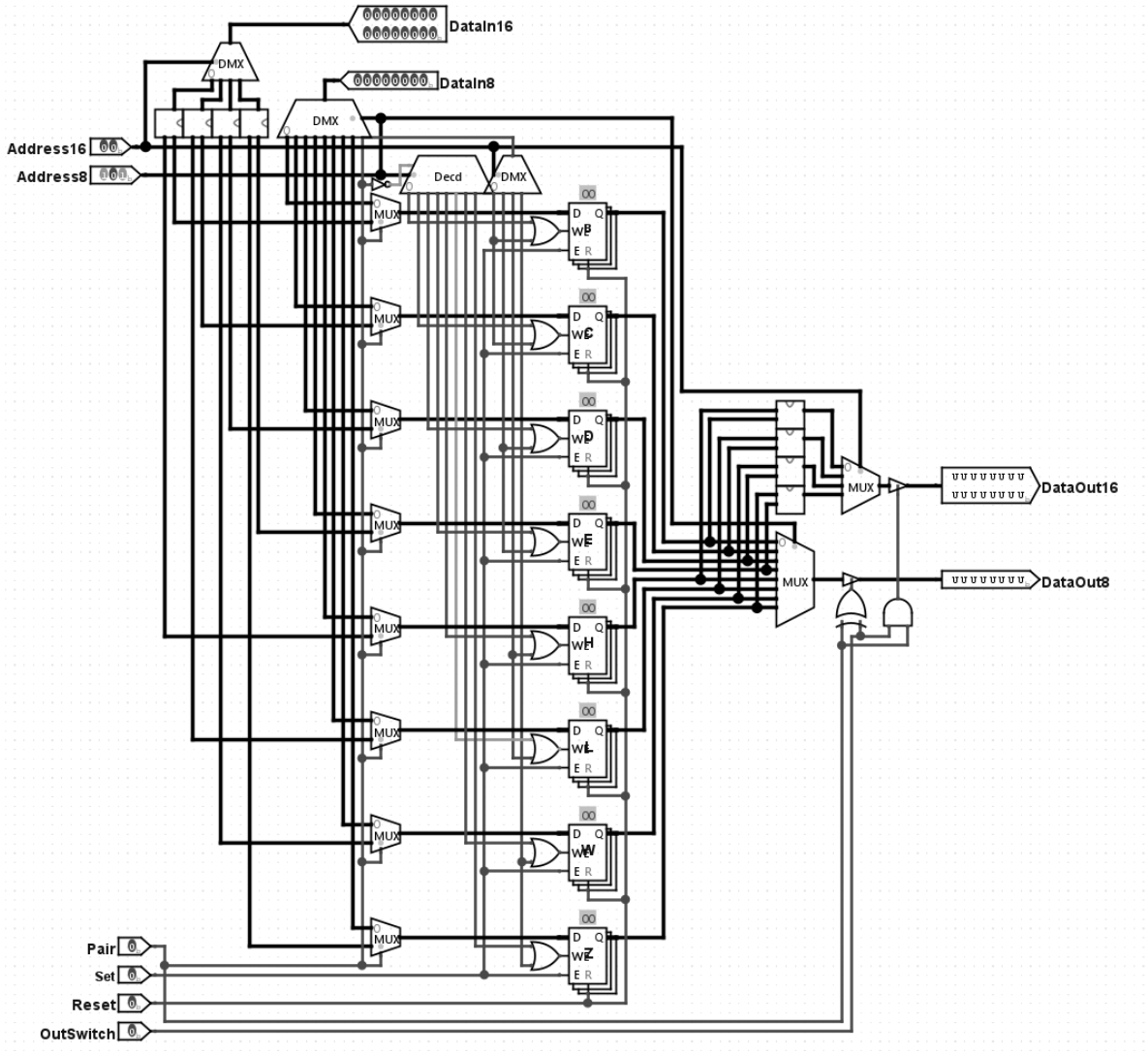
Układ główny służy do połączenia mniejszych elementów w całość, to w nim znajdują się wszystkie sygnały sterujące wychodzące z InsDec.

System nasz budowaliśmy z myślą podziału rejestrów na zwykłe i alternatywne. Nie zdążyliśmy uruchamiać tego instrukcją, jednak wciąż przełączenie jest możliwe dzięki fladze “rodzajrejestru”. Dodatkowo same rejestry wewnętrznie posiadają możliwość łączenia dwóch rejestrów 8 bitowych w 16 bitowy.

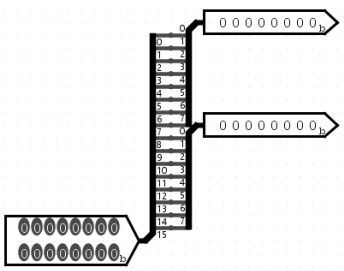


Budowa rejestru:

Na schemacie widać dobrze konstrukcje rejestrów i to w jaki sposób są one przełączane między danymi 8 bitowymi, a 16 bitowymi, gdzie każdy z nich odpowiada różnej adresacji (kolejno: 3 i 2 bitowej). Do podziału danych 16 bitowych zastosowaliśmy własny układ. Za wszystko odpowiada odpowiedni demultiplex, jak i dekoder.



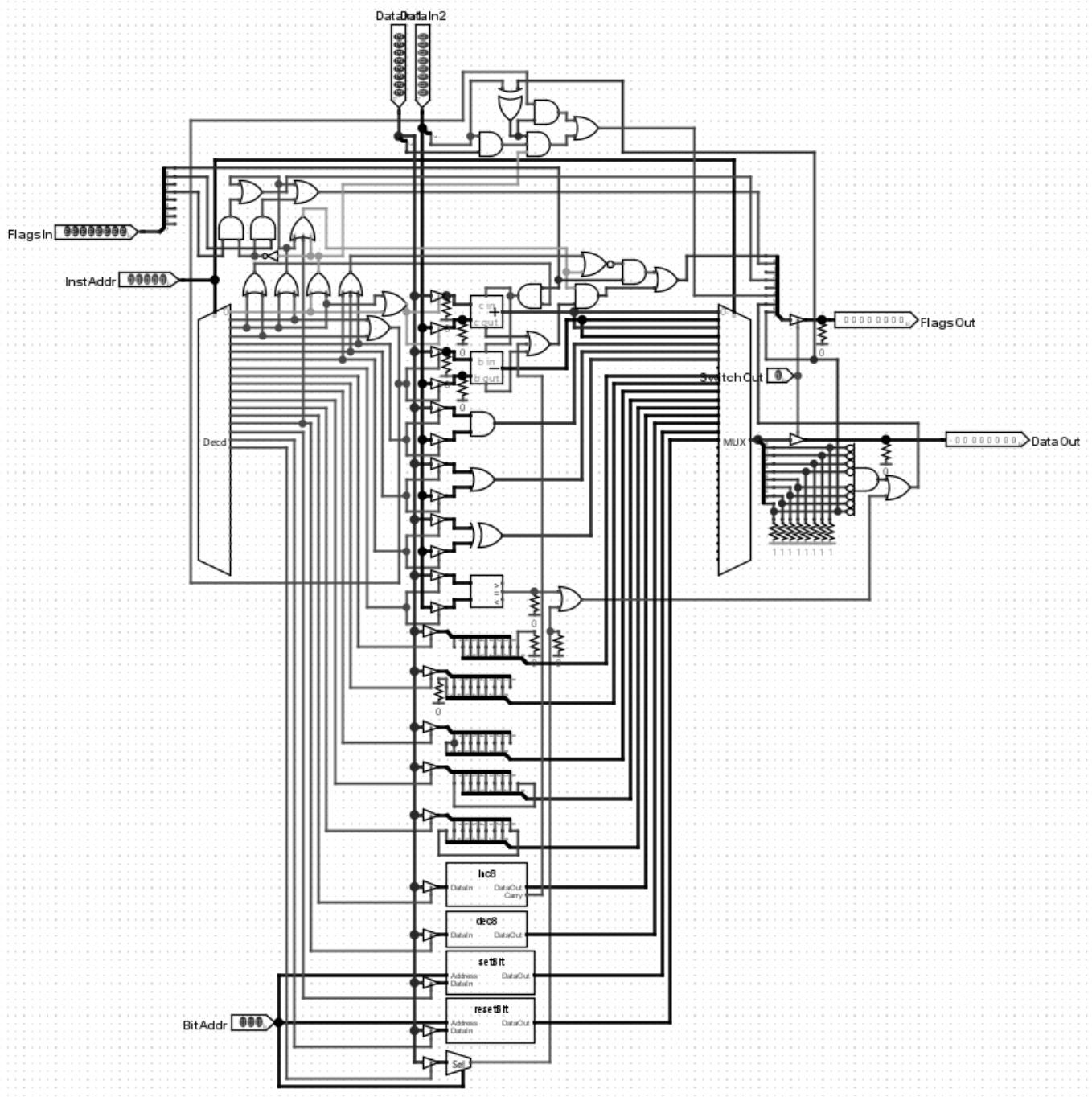
Układ odpowiadający za podział informacji 16 bitowych na dwie 8 biotwe:



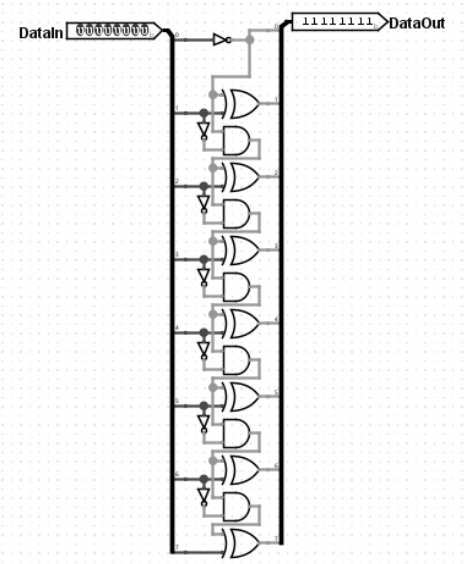
Budowa ALU:

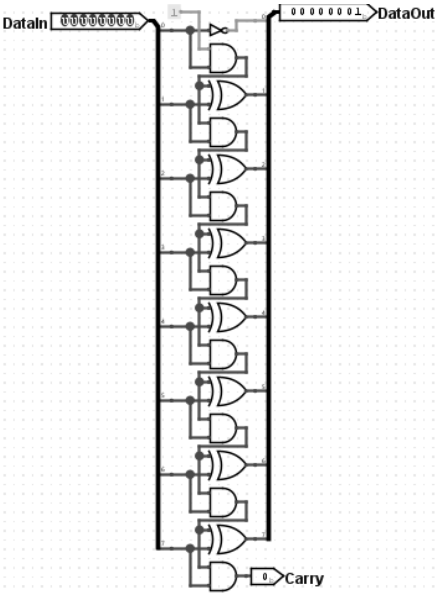
Ważnym punktem wykonywania operacji jest ALU. W naszym wypadku zostało ono zbudowane kombinacyjnie, przy pomocy naszych układów, jak i gotowych w programie logisim.

W samym ALU znajduje się wykrywanie i obsługa flag, która jest też potrzebna podczas skoków warunkowych.

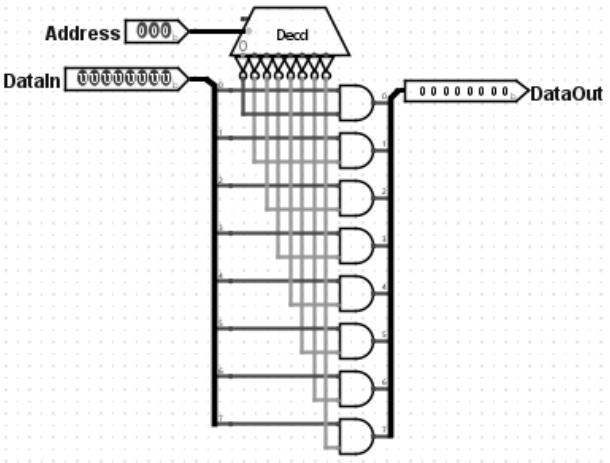


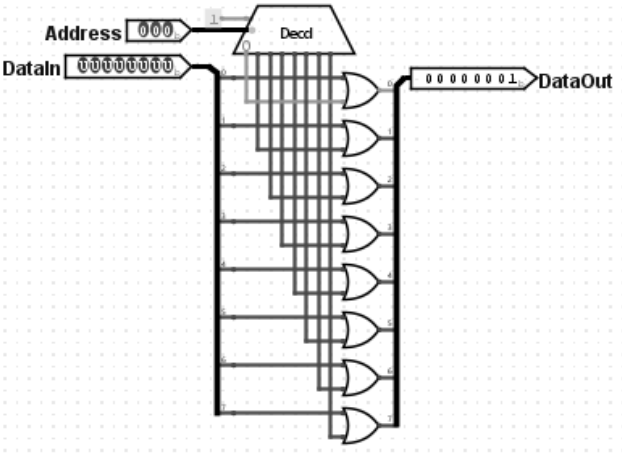
Nasze układy w ALU
Zostały one zbudowane z uwagi na brak odpowiednika w samym programie głównym lub te już istniejące zdawały się nie spełniać naszych potrzeb.
Dekrementor, inkrementor:





reset bit, set bit:



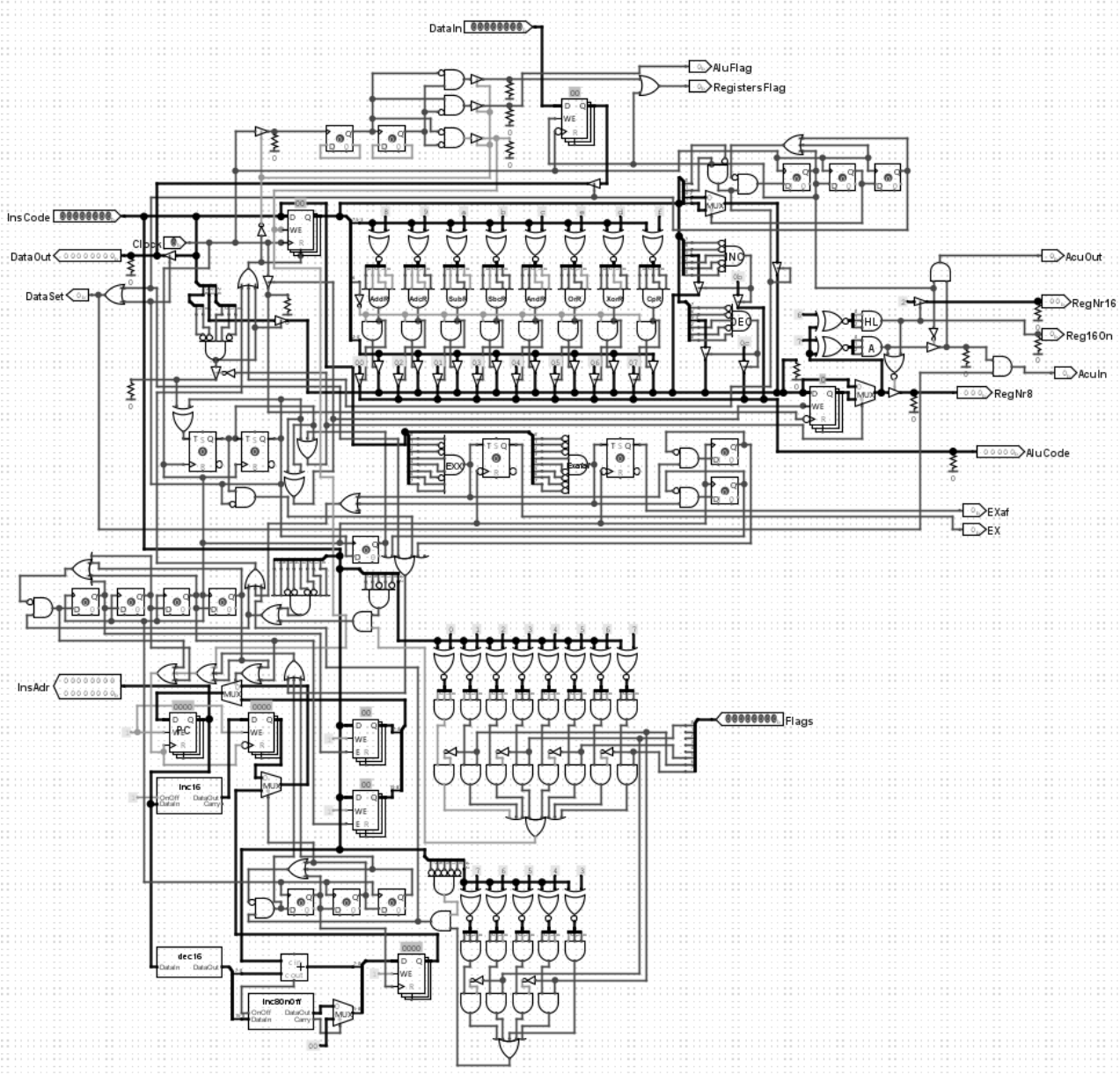


Budowa dekodera:

Najważniejszym elementem samego układu jest dekoder, w którym to umieściliśmy układ sterujący (automat).

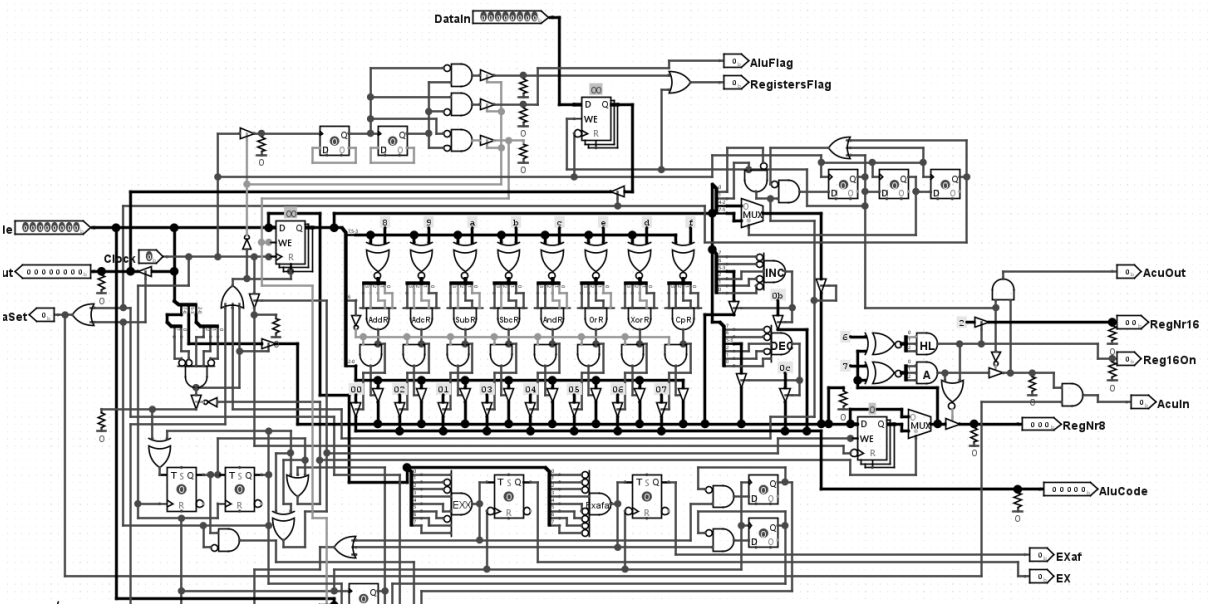
Z uwagi na wielkość układu najlepiej zacząć go analizować po fragmentach. Na początek całość, aby później było wiadomo, jaki element w danym momencie analizujemy:

Podczas budowy tego układu dowiedzieliśmy się, że nie należy stosować bramek na drodze zegara, jednak, z uwagi, że było to pod koniec czasu, zostało to wyeliminowane w maksymalnie jak największej ilości miejsc.

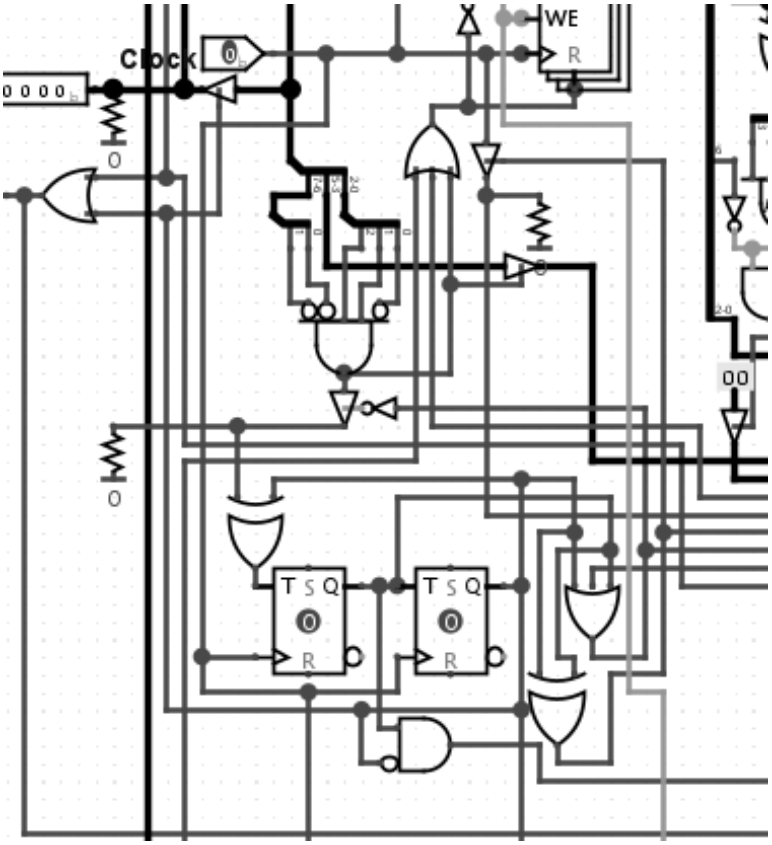


Najprostszą częścią i tą, która została zbudowana jako pierwsza jest dekodowanie instrukcji arytmetycznych oraz ich automat, która część znajduje się u góry układu.

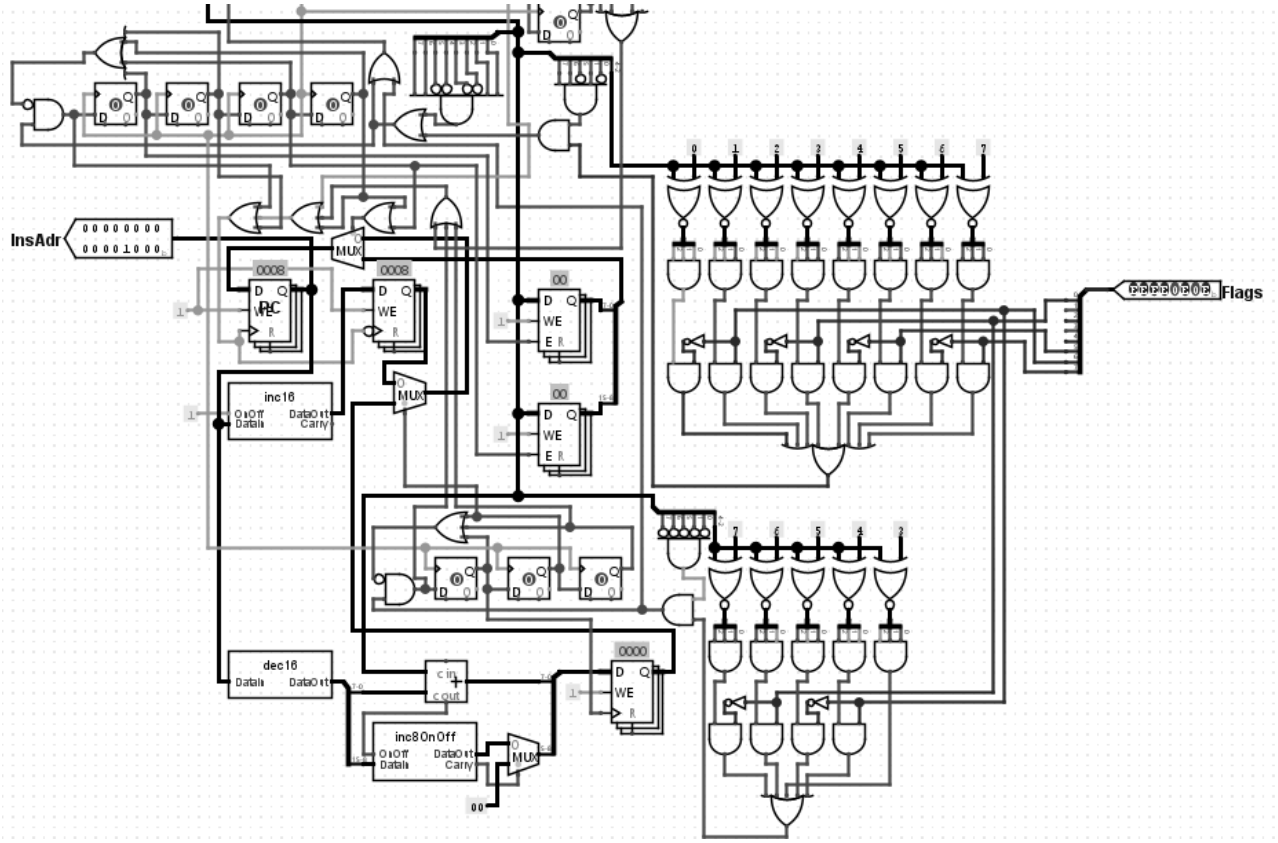
Z uwagi na to, że instrukcje zawierające kod rejestru, zawierają również odniesienia do innych struktur pamięciowych, więc potrzebne było dodatkowe sprawdzenie adresacji.



Część odpowiadająca za wpisywanie danych do rejestrów znajduje się tuż pod powyższym układem. Zawiera ona dodatkowy rejestr zapisujący cel danych (jest on podłączony do systemu dekodującego adresację) oraz układ przerzutników wskazujący na kolejne cykle wykonywania.

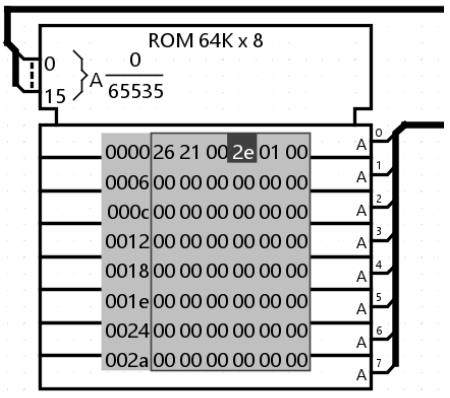


Pod tym układem znajduje się obsługa kolejnych instrukcji, wraz z rejestrem PC zapisującym numer obecnie wykonywanej instrukcji. Znajdują się tam instrukcje skoków zwykłych oraz warunkowych różnego typu.



Jak uruchomić?

- Plik należy uruchomić w Logisim evolution 3.8.0
- Wybieramy opcję łapki i klikamy na ROM
- Każda para zer to nowa instrukcja zapisana szesnastkowo, po kliknieciu na nią, możemy nadpisać zawartość komórki, należy tam wpisać numer instrukcji szesnastkowo [czyli instrukcja LD r,n do rejestru C: 0 0 0 0 1 1 1 0 powinna być zapisana jako 0e)
- Aby uruchomić cykl zegara można kliknąć na zegar lub użyć skrótu Ctrl + “t” (pół cyklu) lub F9 (pełen cykl)



Instrukcje:

informacja do rejestrów i ich adresowania (dana trzy bitowa r)

A	111
B	000
C	001
D	010
E	011
H	100
L	101

Końcowo udało nam się zaimplementować instrukcje:

1. LD r, n

Instrukcja: 0 0 [3 bit r] 1 1 0 2. [dana 8 bit]

Opis: funkcja załadowuje 8-bitową liczbę całkowitą do dowolnego rejestru r

Potrzebna ilość cykli: 4

2. LD r, r'

Instrukcja: 0 1 [3 bit r'] [3 bit r]

Opis: funkcja załadowuje zawartość rejestru r' do drugiego dowolnego registru r

Potrzebna ilość cykli: 4

3. LD r, (HL)

Instrukcja: 0 1 [3 bit r] 1 1 0

Opis: funkcja załadowuje 8-bitową zawartość z danej lokacji pamięci do rejestru r

Potrzebna ilość cykli: 2

4. JP nn

Instrukcja: 1 1 0 0 0 0 1 1 2. [dana 8 bit] 3. [dana 8 bit]

Opis: funkcja załadowuje operand nn do pary rejestru Program Counter i odczytuje kolejną instrukcję ze wskazanego miejsca

Potrzebna ilość cykli: 6

5. JP cc, nn

Instrukcja: 1 1 [3 bit cc] 0 1 0 2. [dane 8 bit] 3. [dane 8 bit]

Opis: wykonuje skok warunkowy na podstawie wskazanej flagi

000	Non Zero
001	Zero
010	No Carry
011	Carry
100	Parity Odd
101	Parity Even
110	Sign Positive
111	Sing Negative

Potrzebna ilość cykli: 4

6. JR e

Instrukcja: 0 0 0 1 1 0 0 0 2. [dane 8 bit e-2]

Opis: wykonuje skok o wartość podaną

Potrzebna ilość cykli: 3

7. JR C, e

Instrukcja: 0 0 1 1 1 0 0 0 2. [dana 8 bit e-2]

Opis: wykonuje skok o wartość podaną zależnie od flagi carry (gdy jest 1)

Potrzebna ilość cykli: 3

8. JR NC, e

Instrukcja: 0 0 1 1 0 0 0 0 2. [dana 8 bit e-2]

Opis: wykonuje skok o wartość podaną zależnie od flagi carry (gdy jest 0)

Potrzebna ilość cykli: 3

9. JR Z, e

Instrukcja: 0 0 1 0 1 0 0 0 2. [dana 8 bit e-2]

Opis: wykonuje skok o wartość podaną zależnie od flagi zero (gdy jest 1)

Potrzebna ilość cykli: 3

10. JR NZ, e

Instrukcja: 0 0 1 0 0 0 0 0 2. [dana 8 bit e-2]

Opis: wykonuje skok o wartość podaną zależnie od flagi zero (gdy jest 0)

Potrzebna ilość cykli: 3

11. ADD A, r

Instrukcja: 1 0 0 0 0 [3 bit r]

Opis: zawartość rejestru r zostaje dodana do akumulatora, gdzie zostanie przechowany wynik

Potrzebna ilość cykli: 2

12. ADC A, r

Instrukcja: 1 0 0 0 1 [3 bit r]

Opis: Operand r wraz z flagą przeniesienia zostaje dodany do akumulatora, gdzie zostanie przechowany wynik

Potrzebna ilość cykli: 2

13. SUB r

Instrukcja: 1 0 0 1 0 [3 bit r]

Opis: Operand r jest odjęty od zawartości akumulatora, gdzie zostanie przechowany wynik

Potrzebna ilość cykli: 2

14. SBC A, r

Instrukcja: 1 0 0 1 1 [3 bit r]

Opis: Operand r wraz z flagą przeniesienia zostaje odjęty od akumulatora, gdzie zostanie przechowany wynik

Potrzebna ilość cykli: 2

15. AND A, r

Instrukcja: 1 0 1 0 0 [3 bit r]

Opis: Operacja logiczna AND zostaje wykonana na bajcie określonym przez operand r oraz na bajcie przechowywanym w akumulatorze. Wynik zostanie przechowany w akumulatorze.

Potrzebna ilość cykli: 2

16. OR A, r

Instrukcja: 1 0 1 1 0 [3 bit r]

Opis: Operacja logiczna OR zostaje wykonana na bajcie określonym przez operand r oraz na bajcie przechowywanym w akumulatorze. Wynik zostanie przechowany w akumulatorze.

Potrzebna ilość cykli: 2

17. XOR A, r

Instrukcja: 1 0 1 0 1 [3 bit r]

Opis: Operacja logiczna XOR zostaje wykonana na bajcie określonym przez operand r oraz na bajcie przechowywanym w akumulatorze. Wynik zostanie przechowany w akumulatorze

Potrzebna ilość cykli: 2

18. CP A, r

Instrukcja: 1 0 1 1 1 [3 bit r]

Opis: Zawartość operanda r zostaje porównana z zawartością akumulatora. Jeśli zostanie zwrócone true, zostaje ustawiona flaga A. Nie wpływa na zawartość akumulatora.

Potrzebna ilość cykli: 2

19. ADD A, (HL)

Instrukcja: 1 0 0 0 0 1 1 0

Opis: Bajt w komórce pamięci wskazany przez parę rejestru HL jest dodany do zawartości akumulatora, gdzie zostanie przechowany wynik.

Potrzebna ilość cykli: 2

20. ADC A, (HL)

Instrukcja: 1 0 0 0 1 1 1 0

Opis: Bajt wskazany przez HL wraz z flagą przeniesienia zostaje dodany do akumulatora.

Potrzebna ilość cykli: 2

21. SUB A, (HL)

Instrukcja: 1 0 0 1 0 1 1 0

Opis: Bajt wskazany przez HL jest odjęty od zawartości akumulatora

Potrzebna ilość cykli: 2

22. SBC A, (HL)

Instrukcja: 1 0 0 1 1 1 1 0

Opis: Bajt wskazany przez HL wraz z flagą przeniesienia zostaje odjęty od akumulatora.

Potrzebna ilość cykli: 2

23. AND A, (HL)

Instrukcja: 1 0 1 0 0 1 1 0

Opis: Operacja logiczna AND zostaje wykonana na bajcie wskazanym przez HL oraz na bajcie przechowywanym w akumulatorze. Wynik zostanie przechowany w akumulatorze.

Potrzebna ilość cykli: 2

24. OR A, (HL)

Instrukcja: 1 0 1 1 0 1 1 0

Opis: Operacja logiczna OR zostaje wykonana na bajcie wskazanym przez HL oraz na bajcie przechowywanym w akumulatorze. Wynik zostanie przechowany w akumulatorze.

Potrzebna ilość cykli: 2

25. XOR A, (HL)

Instrukcja: 1 0 1 0 1 1 1 0

Opis: Operacja logiczna XOR zostaje wykonana na bajcie wskazanym przez HL oraz na bajcie przechowywanym w akumulatorze. Wynik zostanie przechowany w akumulatorze.

Potrzebna ilość cykli: 2

26. CP A, (HL)

Instrukcja: 1 0 1 1 1 1 1 0

Opis: Zawartość bajta wskazanego przez HL zostaje porównana z zawartością akumulatora. Jeśli zostanie zwrócone true, zostaje ustawiona flaga A. Nie wpływa na zawartość akumulatora.

Potrzebna ilość cykli: 2

27. EXX

Instrukcja: 1 1 0 1 1 0 0 1

Opis: Przełącza rejestry zwykłe na alternatywne i odwrotnie, wraz z rejestrami WZ

Potrzebna ilość cykli: 4

28. EX AF, AF'

Instrukcja: 0 0 0 0 1 0 0 0

Opis: Przełącza rejestry akumulatora i frag na alternatywne i odwrotnie

Potrzebna ilość cykli: 4

29. INC r

Instrukcja: 0 0 [3 bit r] 1 0 0

Opis: Inkrementuje wartość z rejestru r

Potrzebna ilość cykli: 2

30. DEC r

Instrukcja: 0 0 [3 bit r] 1 0 1

Opis: Dekrementuje wartość z rejestru r

Potrzebna ilość cykli: 2

31. INC (HL)

Instrukcja: 0 0 1 1 0 1 0 0

Opis: Bajt zawarty w adresie, wskazanym przez HL jest inkrementowany

Potrzebna ilość cykli: 2

32. DEC (HL)

Instrukcja: 0 0 1 1 0 1 0 1

Opis: Bajt zawarty w adresie, wskazanym przez HL jest dekrementowany

Potrzebna ilość cykli: 2

Z uwagi na to, że udało się zaimplementować instrukcje wymagające różnej ilości cykli, jak i danych, bylibyśmy w stanie (przy dodatkowym czasie) zaimplementować większą ilość instrukcji, a nawet wszystkie, dostępne w procesorze.

Instrukcje, które były gotowe do implementacji (zbudowano pod nie już szkielet) i zabrakło jedynie czasu na dekoder to:

BIT b, r

SET b, r

RES b, r

i instrukcje przesunięć

Do samej konstrukcji procesora zostały wykorzystane:

Rozdzielacz	134
Pin	134
Rezystor ściągający	36
Stała	50
NOT (negacja)	46
AND (koniunkcja)	143
OR (alternatywa)	53
NOR (zaprzeczony OR)	2
XOR (alternatywa wykluczająca)	69
XNOR (zaprzeczony XOR)	30
Bufor sterowany	57
Multiplekser	30
1 bit adresowych	24
2 bit adresowych	2
3 bit adresowych	2
4 bit adresowych	2
Demultiplekser	19
1 bit adresowych	10
2 bit adresowych	4
3 bit adresowych	2
4 bit adresowych	1
Dekoder	5
3 bit adresowych	4
4 bit adresowych	1
D Flip-Flop	6
T Flip-Flop	2
Rejestr	29
rejestr 16	5
rejestr 8	23
rejestr 3	1
dodawanie	1
odejmowanie	1
porównanie	1

Znając ilość wykorzystanych układów, przyjmując wielkość tranzystora jako: $2 * 10^{-14}$ m^2 oraz znając budowę wykorzystanych układów, jest możliwe przybliżone określenie wielkości całego procesora.
Uzyskaliśmy wielkość: $1{,}92 * 10^{-06}$ m^2 co jest wartością zbliżoną do rzeczywistej wielkości procesora, która wynosi około $1{,}18 * 10^{-05}$ m^2. Odstępstwo od wartości oczekiwanej może być spowodowane:

- mniejszą ilością układów, z uwagi na nie zaimplementowanie wszystkich instrukcji
- błędy pomiarowe, związane z wielkością tranzystora, czy samych poszczególnych układów
- największy wpływ na wynik końcowy miała ilość i gęstość połączeń, przez co, błąd w ich obliczaniu, może mocno rzutować na sam wynik końcowy
- możliwe też jest niedokładne uwzględnienie odstępów między układami

Należy też uwzględnić, że przy liczeniu wielkości nie została wzięta pod uwagę pamięć ROM.
Całość musiała być szacowana bez pomocy dodatkowych programów, czyli “ręcznie”.

Analiza i wnioski

Analizując nasze rezultaty, doszliśmy do wniosku, iż strategia dzielenia układu i budowania go modularnie sprawdziła się, gdyż ułatwia późniejsze poprawki, jak i dodawanie kolejnych elementów procesora. Sama rozbudowa jest w ten sposób ułatwiona. Podczas konstrukcji popełniliśmy parę błędów związanych z brakiem wiedzy, jednak za każdym razem staraliśmy się zmniejszyć ich oddziaływanie na projekt.

Zdajemy sobie sprawę, że wiele instrukcji dałoby się wykonać używając mniejszą ilość bramek, czy cykli, jednak jest to najmniejsza możliwa ilość, jaką udało nam się osiągnąć.

Największym problemem projektu był ograniczony czas, jednak udało nam się sporo nauczyć i wynieść podczas jego budowy.

Bibliografia

Zilog, Inc. 2016 Z80 CPU User Manual:
https://www.zilog.com/docs/z80/um0080.pdf

Załączniki

Zostaną one umieszczone na e-portalu. Będą zawierać:

- Elektroniczna wersja sprawozdania
- Układ stworzony w Logisim evolution