Wrocław, 31.05.2023

Projekt i implementacja procesora Intel 8051

Igor Sosnowicz, Jakub Dostal

1. Wprowadzenie

Celem projektu było samodzielne zaprojektowanie oraz zaimplementowanie procesora Intel 8051.

Projekt wykonany został z wykorzystaniem symulatora logiki cyfrowej Logisim Evolution. Całość przechowywana była w repozytorium projektu na GitHubie.

Autorami projektu są Jakub Dostal (263959) oraz Igor Sosnowicz (263897).

W naszej implementacji wykorzystaliśmy kilka podstawowych instrukcji procesora, o których szerzej w następnym dziale.

2. Specyfikacja

Nasz procesor wyposażony jest w jednostkę arytmetyczno logiczną (ALU) wykonującą operacje inkrementacji, dekrementacji, dodawania, odejmowania, mnożenia, dzielenia, logicznego AND, logicznego OR, porównania (w przypadku funkcji skoku warunkowego) oraz rotacji. Cały procesor składa się z połączonych bloków, z których każdy odpowiada za co innego (ALU, PSW, …).

Procesor wyposażony jest w rejestry R0 – R7, akumulator oraz rejestr B (a także kilka rejestrów pomocniczych, niedostępnych z poziomu użytkownika).

Zaimplementowane flagi to carry, zero, auxiliary carry, overflow oraz sign.

Poniżej znajduje się lista oraz krótki opis zaimplementowanych przez nas instrukcji:

- MOV – instrukcja skopiowana wartości ze źródła do miejsca docelowego. Zaimplementowane zostały przez nas dwa warianty tej instrukcji – z rejestru (R0 – R7) do akumulatora oraz z akumulatora do rejestru (R0 – R7).
- XCH – instrukcja zamiany miejscami wartości w rejestrze (R0 – R7) oraz wartości w akumulatorze. Instrukcja wykorzystuje jako pośrednika do wymiany rejestr B.
- JC / JNC – instrukcja skoku warunkowego odpowiednio dla wartości flagi Carry równej 1 oraz 0.
- JNZ – kolejna instrukcja skoku warunkowego, tym razem dla zerowej flagi zero.
- ADD / SUBB / ADDC– instrukcja dodawania, odejmowania oraz dodawania z uwzględnieniem flagi carry, zaimplementowane w dwóch wariantach – wartość z rejestru z wartością z akumulatora oraz wartość z akumulatora z wartością immediate.
- RR / RL – operacja rotacji, odpowiednio przesunięcie ostatniego bitu na początek i pierwszego bitu na koniec.
- INC / DEC / ANL / ORL – zwiększenie wartości o jeden, zmniejszenie wartości o jeden, logiczna operacja AND oraz logiczna operacja OR.

3. Idea działania

Nasz procesor oparty jest na odpowiednio zakodowanych ciągach bitów, przechowywanych w pamięciach ROM w postaci kodu i odpowiednio rozbijany na odpowiednie sygnały na magistrali.

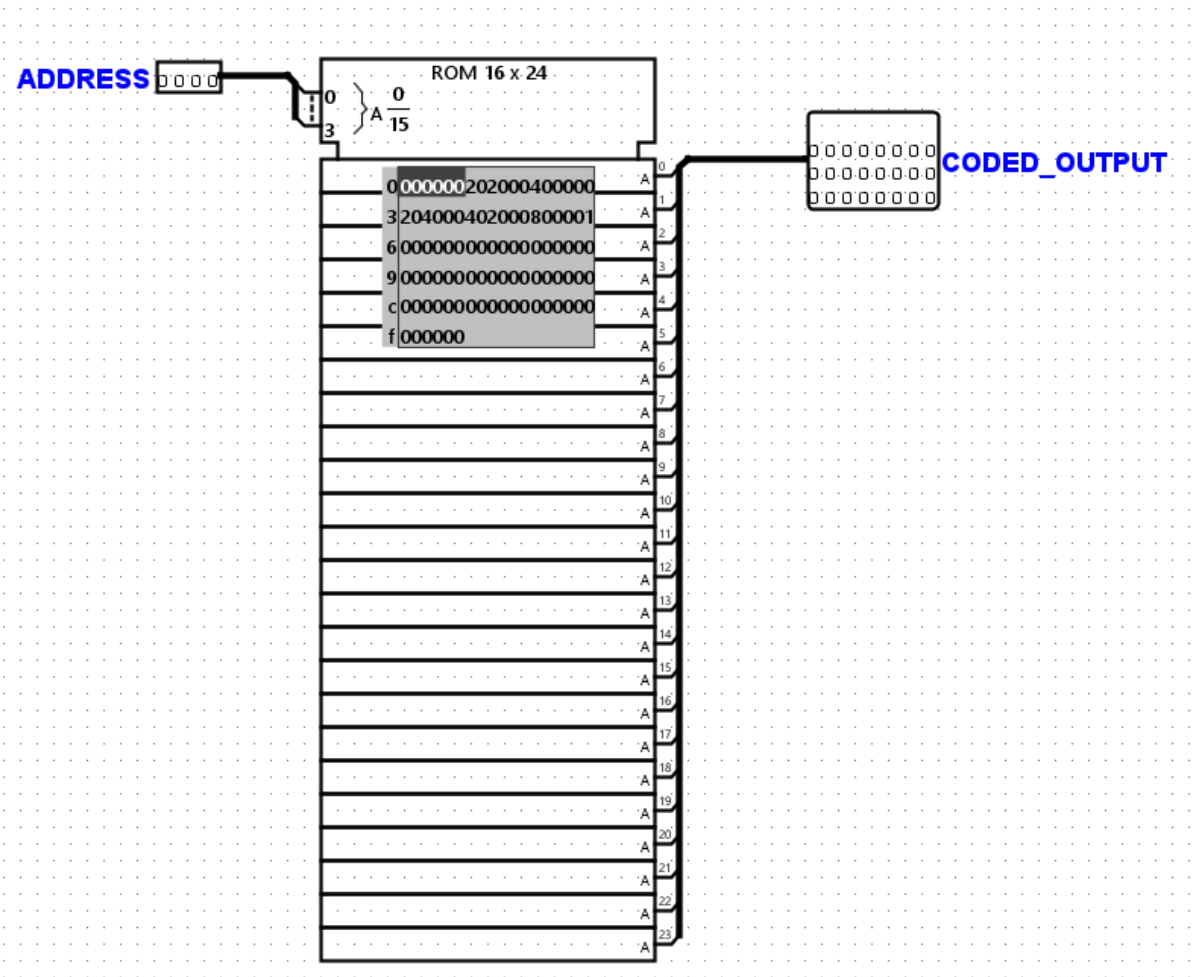
Poniżej przedstawiony jest każdy z symboli w ciągu wraz z jego długością w bitach, cały ciąg ma długość 24 bitów i obsługuje jedną operację:

1	1	1	4	5	1	4	4	1	1	1
increment PC	register store	register load	ALU opcode	register address	indirect	satisfied	not satisfied	check carry	check zero	finish

Dla przykładu – poniżej znajduje się lista ciągów bitowych dla instrukcji rotacji:

	1	1	1	4	5	1	4	4	1	1	1
	increment PC	register store	register load	ALU opcode	register address	indirect	satisfied	not satisfied	check carry	check zero	finish
0	0	0	0	0000	00000	0	0000	0000	0	0	0
1	0	0	1	0000	00010	0	0000	0000	0	0	0
2	0	1	0	0000	00000	0	0000	0000	0	0	0
3	0	0	1	0000	00100	0	0000	0000	0	0	0
4	0	1	0	0000	00010	0	0000	0000	0	0	0
5	1	0	0	0000	00000	0	0000	0000	0	0	1

Każde słowo jest konwertowane do postaci heksadecymalnej i zapisywane w odpowiednim ROMie:



Krótki opis każdego elementu słowa kodowego:

- Increment PC oznacza zwiększenie Program Countera dla danego słowa
- Register store / load pozwalają procesorowi odpowiednio na zapis lub odczyt do rejestru, w zależności od potrzeby
- ALU opcode to kod dla jednostki arytmetyczno logicznej, aby wiedziała, jaką operację należy wykonać
- Register address – adres rejestru, do którego zapisujemy lub z którego odczytujemy
- Indirect – jeżeli adresujemy niebezpośrednio, flaga indirect jest ustawiona na 1
- (not) Satisfied - informacja dla procesora, do której instrukcji ma udać się w przypadku skoku warunkowego w zależności od tego, czy jego wymaganie jest spełnione czy nie
- Carry / zero check – informacja dla procesora, którą flagę ma sprawdzić w przypadku skoku warunkowego
- Finish wyznacza koniec konretnej operacji

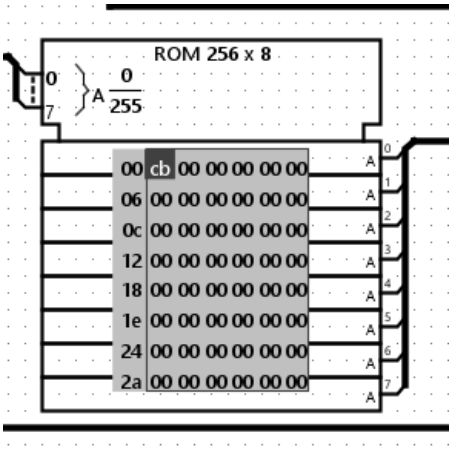
4. Obsługa procesora

Aby skorzystać z procesora, należy w jego pamięci ROM wpisać odpowiednie kody instrukcji wraz z ewentualnym parametrem, takim jak wartość dodawana, miejsce, w które należy skoczyć itp.

Tabela z kodami instrukcji znajduje się pod poniższym linkiem (link jest również dostępny w repozytorium projektu na GitHubie oraz w folderze z projektem):

https://developer.arm.com/documentation/101655/0961/8051-Instruction-Set-Manual/Opcodes

Na przykład dla operacji zamiany wartości z R3 i wartości z A (XCH), należy podać w naszej pamięci kodu wartość CB:



Pamięć kodu znajduje się w bloku **main**, na samym dole układu.

5. Trudności i wnioski

Największą trudnością było dla nas zaimplementowanie prawidłowo działającej pamięci procesora – dlatego w naszej implementacji brakuje jakichkolwiek operacji na pamięci, wszystkie wykonywane są na rejestrach. Spowodowane jest to najpewniej niedokładnym zrozumieniem działania modułu pamięci w procesorze, a wszelkie próby dodania takiego modułu kończyły się niepowodzeniem.

Niemniej jednak procesor spełnia swoje zadanie i jest w stanie wykonywać obliczenia, a także działać według ciągu instrukcji znajdującym się w przechowywanym przez niego kodzie. Nie jest to jednostka najbardziej wydajna, ale nie jest też bardzo wolna – jest to prosta procesor zaimplementowany w najbardziej intuicyjny dla nas sposób.