2-PODSTAWY PIC18 TUTORIAL

Piotr Chmura

2 października 2021

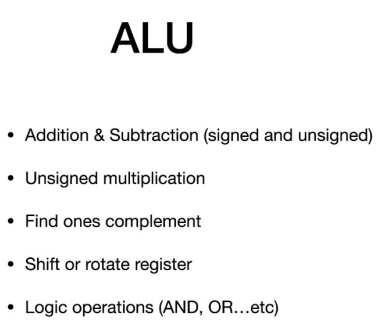
1 Wstęp

Część druga omawiająca podejście teoretyczne dotyczące programowania mikrokontrolerów.

2 Procesor

Ostatnią z trzech części którą należy omówić jest procesor. Choć jest to bardzo uproszczone myślenie a sam procesor jest o wiele bardziej skomplikowany, podzielić go można na 3 główne części. Jednostkę arytmetyczno-logiczną (ALU), Rejestry generalnego oraz specjalnego przeznaczenia jak i jednostkę kontrolną.

2.1 Jednostka Arytmetyczno-Logiczna

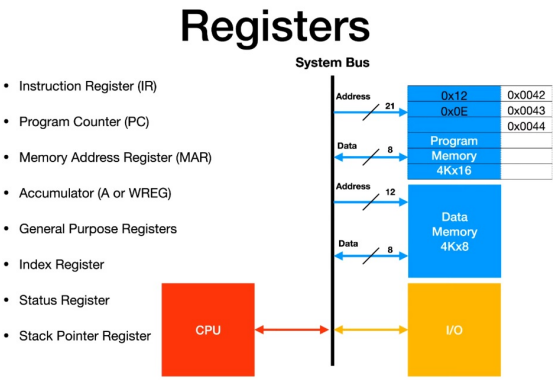


Rysunek 1: ALU

Działania jakie jednostka może wykonywać to między innymi:
Dodawanie i odejmowanie ze znakiem lub bez.
Mnożenie bez możliwości rozróżnienia znaków oraz brak możliwości dzielenia, chyba że napiszemy do tego algorytm.
Znajdowanie dopełnienia do 1.
Przesuwanie oraz rotacje rejestrów.
Tworzenie funkcji logicznych.

Każdy z powyższych podpunktów potrzebuje danych na których będzie mógł operować. Dane te składowane mogą być w pamięci danych, lecz zazwyczaj CPU posiada rejestry wewnętrzne. CPU jak wspomniano wcześniej posiada szerokie grono specjalnych rejestrów. Posiadanie wewnętrznych rejestrów w CPU czyni operacje które chcemy wykonać znacznie szybsze z racji braku konieczności adresacji i oczekiwania na pewną wartość co zmniejsza czas dokonywania operacji, zajmują się tym rejestry generalnego przeznaczenia (GPR). CPU posiada również rejestry specjalne.

2.2 IR



Rysunek 2: ALU

Rejestr Instrukcji (IR). Gdy piszemy program zauważyć możemy iż zapisywany jest w pamięci programu w której znajdują się poszczególne zapisane wcześniej dane. Używamy notacji Heksadecymalnej. Podczas pracy procesora nastąpi odczyt wartości zawartej w pamięci programu pod adresem np. 0x12 oraz 0X0E. Pamiętać należy iż mamy do czynienia z pamięcią bajtu. Instrukcja taka zostanie zapisana przez procesor w IR jako na przykład 0E12.

2.3 PC

Program Counter. Licznik programu (PC). Odpowiedzialny jest za wskazywanie użytkownikowi gdzie aktualnie znajduje się w programie. Dla przykładu załóżmy iż chcemy z pamięci odczytać daną pod adresem 0x0042. Licznik programu mówi nam jaki jest adres następnej instrukcji do której będziemy przechodzić w trakcie wykonywania naszego programu. Następnym adresem będzie 0x0043. PC wiedział już będzie iż kolejną instrukcją jaką należy wyegzekwować to 0x0044. Niektórzy z was mogą myśleć, iż w czasie odczytywania programu sposób tegoż odczytywania jest sekwencyjny (instrukcje następują jedna po drugiej). Generalnie tak jest, lecz instrukcje mogą prowadzić do operacji skoku. Operację tą możemy porównać do funkcji napisanej w C, egzekwowanej z main(), działającej na tej samej zasadzie skoku do funkcji, wykonania zawartego w funkcji algorytmu oraz powrotu do egzekwowania części głównej programu.

2.4 MAR

MAR (Memory Adres Register) jest to rejestr przechowujący adres którym zainteresowany jest procesor i jest to adres znajdujący się w pamięci danych . Załóżmy iż odczytywanym adresem jest FA1. Oznacza to że procesor zainteresowany jest lokalizacją tego adresu w pamięci danych i chce go odczytać. Składuje więc adres wewnętrznie w MAR, chcąc dokonać wpisania lub odczytania wartości, CPU nie musi generować adresu na nowo ponieważ znajduje się on już w MAR.

2.5 Akumulator - rejestr W

Jest to jeden z najważniejszych rejestrów CPU. W dokumentacji PIC18F odwołanie do niego to WREG. Niektóre z CPU oparte są na akumulatorze inne z nich nie są. Akumulator to miejsce w którym podczas dokonywanej operacji na przykład dodawania liczb owe liczby zostają składowane w celu późniejszego działania na nich. Po zakończeniu wykonywania operacji akumulator zostaje zwolniony, wpisując z powrotem wartości liczb do pamięci danych. Uogólniając schemat działania, akumulator aktualnie zajmuje się pewnymi danymi na których pracuje ALU, po zakończeniu swojej pracy przenosząc je do pamięci danych. Rejestry generalnie rzecz ujmując są znacznie szybsze niż pamięci, lecz zajmują one przestrzeń przeznaczoną dla CPU co zwiększa fizyczną wielkość układu scalonego. Nie zajmują one tylko przestrzeni, generują one również zwiększony pobór mocy. Jest to drogie rozwiązanie rozpatrując w terminach produkcji, mocy, przestrzeni. Jeśli w CPU zostaje użyty rejestr, musi on mieć funkcjonalność pozwalającą na przyspieszenie zdolności obliczeniowej. Są to rejestry pozwalające procesorowi dokonywać obliczeń znacznie szybciej. Owe rejestry to miejsce składowania danych. Procesor odczytuje dane z pamięci, wczytując je do GPR. Ten temat zostanie jeszcze poruszony, ponieważ będziemy pisać operacje pośredniczące właśnie do GPR przed usunięciem oraz zwróceniem do pamięci danych.

2.6 Rejestr Indeksu

Zachowuje on indeks tablicy. Pomaga on nam zwracając wartość indeksu na którym się znajdujemy w tablicy 2D.

2.7 Rejestr Stanu

Każdy z bitów rejestru statusu niesie ze sobą ważną informację. Odczytując poszczególne bity będące flagą oznaczającą iż pewna z operacji została dokonana o czym świadczy pojawienie się w rejestrze logicznej jedynki lub zera na określonym bicie, dostajemy informacje na temat zachodzących procesów. Niektóre z tych flag to flagi przeniesienia. Dodając kilka liczb może wystąpić przeniesienie, jeśli przeniesienie jest logiczną jedynką pewien z bitów zostanie ustawiony w stan wysoki. Kolejnym przykładem jest flaga zero. Mówi ona o dokonaniu operacji o rezultacie zero. Ową flagę rozpoznać możemy również po stanie wysokim na bicie odpowiedzialnym za informację na temat tej flagi. Istnieją również inne flagi, dla wartości pozytywnych i negatywnych jak i tylko pozytywnych. Dla przykładu gdy dokonaliśmy operacji której wynikiem była negatywna liczba, odzwierciedlać się to będzie ustawieniem bitu flagi w stan wysoki.

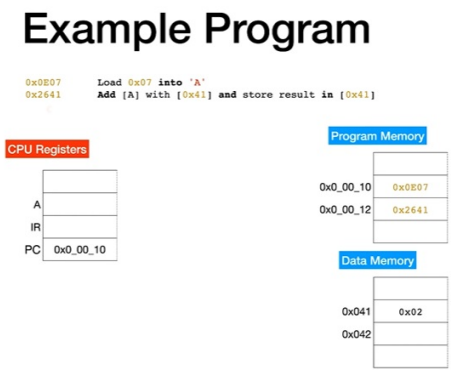
2.8 Rejestr wskaźnika stosu

Stack Pointer Register. Zapisuje on adres oraz dokonuje wpisu do licznika programu (PC).

3 Jednostka kontrolna

O jednostce kontrolnej myśleć możemy jako o maszynie stanów skończonych. W celu dodania kilku liczb, maszyna stanu udaje się do pierwszego z nich w którym pobiera pierwszą z wartości będącą w pewnej lokalizacji.Następnie przechodzi do drugiego stanu, będącego pobraniem drugiej wartości, udając się do trzeciego stanu gdzie wysyła wartość do ALU w określoną lokalizację. Czwartym stanem jest dodanie wartości przez ALU, piątym pobranie wartości oraz umieszczenie jej gdzieś w pamięci danych itd. Jest to uproszczone spojrzenia na jednostkę kontrolną, z racji kontroli bardzo dużej ilości procesów. Jednostka kontrolna połączona jest do szyny systemowej, odpowiedzialna jest za komunikację z pamięcią poprzez linie adresacji, sprawdzając czy linia danych gotowa jest do odebrania informacji co kontroluje wpisywanie/odczytywanie wartości do pamięci.

4 Przykładowy program

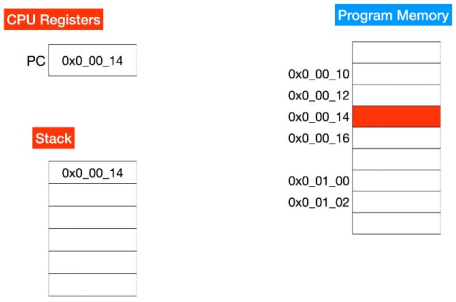


Rysunek 3: Przykładowy program

Wspomniany wcześniej program składa się z dwóch instrukcji u góry Rysunku 3. 2 Kody operacyjne pozwolą zilustrować jak powinna wyglądać interakcja między komponentami omówionymi wcześniej. Po prawej stronie rysunku widzimy pamięć programu oraz pamięć danych. Z lewej natomiast widzimy niektóre z ważniejszych rejestrów CPU (dla rozpatrywanego programu). Szyna systemowa powinna znajdować się pomiędzy wymienionymi komponentami, wyjścia I/O lecz nie ma ich by nie zaciemniać przykładu. Powyższy program zapisywany jest w pamięci programu. Instrukcję 0x0E07 składujemy w pamięci programu pod adresem 0x0-00-10 kolejną pod 0x0-00-12. Używamy notacji HEX co za tym idzie każda z cyfr lub liter odzwierciedla stan 4 bitów. Dwie instrukcje mające 4 znaki HEX mnożymy przez 4 bity na jeden co daje 16 bitów [b] czyli 2 bajty [B] stąd różnica adresów o 2 - adresacja bajtowa. Na potrzeby tego programu do lokalizacji 0x041 wpisujemy wartość 0x02. W tym momencie zawartość akumulatora i rejestru instrukcji jest zerowa, ponieważ program który rozpatrujemy nie uwzględnia wcześniejszych manipulacji oraz zjawisk które mogły zajść podczas jego działania. To co nas interesuje to Licznik Programu - PC. Mówi on nam iż adres następnej instrukcji to 0x0-00-10. Przejdziemy przez kroki wykonywane przez program. CPU musi dowiedzieć się co znaczy instrukcja którą musi wyegzekwować. Udaje się więc do PC w celu odczytania adresu na który on wskazuje. Adres zostaje

podany na linię adresową pamięci programu w celu odczytania zawartości komórki pamięci. Odczytuje on instrukcje, zapisując ją do rejestru instrukcji IR, w międzyczasie wykonując kilka operacji dekodowania oraz multipleksowania. W ten sposób dokonuje interpretacji komendy zawartej w IR. Komentarz przy adresie 0x0E07 nakazuje załadowanie wartości 0x07 HEX do akumulatora A. CPU rozbierze na części pierwsze instrukcję, jak to robi będziemy mogli zobaczyć w następnych programach. Uogólniając Jednostka kontrolna wie iż 0E to komenda ładowania wartości do akumulatora a 07 to wartość którą należy tam wpisać. Po wpisaniu wartości do akumulatora PC powinien przenieść się do następnego adresu którym jest 0x0-00-12, w celu wykonania dalszej części programu. Widzimy więc iż PC wartość adresu wskazywana przez PC uległa zwiększeniu, do akumulatora została wpisana żądana liczba. Następuje zmiana zawartości rejestru instrukcji na nową wartość wskazywaną przez PC. Jednostka kontrolna znów próbuje rozebrać komendę na części pierwsze oraz dokonać jej interpretacji. Instrukcja 0x2641 nakazuje pobranie wartości akumulatora, przejście do pamięci danych pod adresem 0x41 pobierając zawartość pod owym adresem. Następnie dokonuje ona (ALU) operacji dodawania składując jej rezultat w komórce 0x41 (0x09) nadpisując wartość pamięci danych. Po zakończeniu procesu należy przesunąć dalej PC. Analizując ten program dowiedzieliśmy się jak działają podstawowe komponenty takie jak akumulator, licznik programu oraz rejestr instrukcji w praktyce.

4.1 Stack - stos



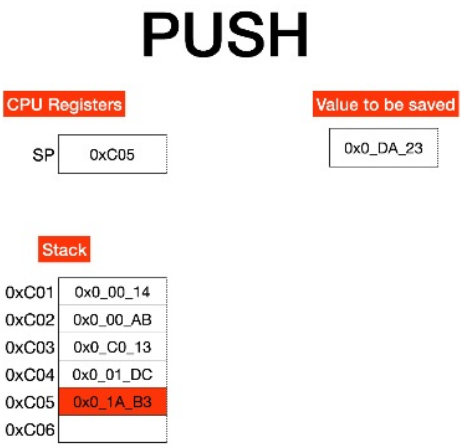
Rysunek 4: Przykładowy program

Stack z którym mamy do czynienia działa na zasadzie LIFO - Last In First Out (bufor). Jest to zbiór pamięci w którym składować możemy pewne komendy w celu powrotu do wykonywania poprzedniego zadania. Dlaczego potrzebujemy stosu, jakie jest jego zastosowanie oraz jak go używać? Nauczymy się wpisywać oraz odczytywać poszczególne wartości ze stosu. Załóżmy iż mamy do czynienia z podobnym programem, jaki napisaliśmy kilka minut temu.

4.2 PUSH

Zauważyć możemy że licznik programu wskazuje dokładnie na adres komórki zaznaczonej na czerwono. Procesor rozpocznie interpretację. Następnym krokiem będzie przesunięcie PC do następnej komórki.Dokona jakiejś operacji znów przesuwając PC. Załóżmy, że instrukcja na którą wskazuje pod 14 jest wywołaniem

kolejnej subrutyny. Wracając do licznika programu. Nakazujemy przeskoczenie PC W celu wykonania subrutyny znajdującej się w innych komórkach pamięci. Po zakończeniu wykonywania są subrutyny nastąpi zwrócenia wartości oraz powrót do ostatniej instrukcji zapisanej na stack. Możemy o tym myśleć jak wywołaniu funkcji w języku C w main().

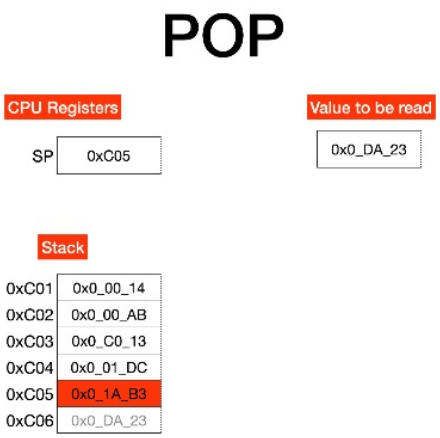


Rysunek 5: Wpisanie na stos

Postaramy się zrozumieć co dokładnie dzieje się podczas tej operacji. Procesor mówi, iż licznik programu aktualnie odnosi się do adresu 0x0-00-14 zapisując ten adres na stosie. Tak jak wcześniej założyliśmy, pod adresem znajduje się instrukcja skoku, co zmienia nam zawartość licznika programu, a co za tym idzie odczytywane miejsce w pamięci. Owa subrutyna jest bardzo krótka (2 instrukcje) z czego w drugiej wracamy do wcześniej wykonywanego działania. Aktualnie zapisana wartość na stosie zostaje ponownie odczytana umożliwiając powrót do wykonywania programu. Innymi słowy procesor uda się do stosu, z pytaniem o miejsce do którego ma wrócić. Po odczytaniu owej wartości ze stosu, następuje jego wyczyszczenie. Stos jest nam potrzebny, gdy mamy do czynienia z zagnieżdżonymi operacjami skoku między poszczególnymi subrutynami. Z każdym powrotem przemieszczać się będziemy w górę stosu. PIC18F posiada 32 rejestrowy stos. Omówimy proces wypychania danych (pop).

4.3 POP

Wspomniany wcześniej rejestr wskaźnika stosu znajduje się na pewnym adresie w środku stosu. Posiadając nową wartość 0x0-1A-D3 którą chcemy wpisać na stos, należy użyć pewnych komend, w uproszczeniu powyższa wartość zostanie wpisana na stos, powodując przesunięcie wskaźnika stosu na adres 0xC05. Z następną wartością schemat jest identyczny. Dokonaliśmy zapełnienia stosu, chcąc wrócić do poprzednich subrutyn należy odczytać ostatnią wpisaną wartość stosu, na którą wskazuje wskaźnik stosu. Odczytywanie wartości ze stosu to tzw. POP. Następnie należy zmniejszyć o 1 wartość wskaźnika stosu. Ostatnia wpisana wartość przeniesie się na adres 0xC05 co spowoduje bram możliwości powrotnego odczytu ze stosu. Jeśli chodzi o hardware owa wartość nie znika,ponieważ jest częścią pamięci. Jednak wskaźnik utracił referencję do owego adresu co sprawia iż po prostu nie może go odczytać. Więc gdy napotkamy nową wartośc do wpisania na stos, zostanie ona nadpisana w miejscu podprogramu w wartości 0xC06. Wypchnięcie powoduje zwolnienie możliwości wpisania wartości na stos.



Rysunek 6: Wypchnięcie ze stosu