

W kolejnym odcinku naszego cyklu, którego celem jest poznanie i nauczenie się programowania mikrokontrolerów serii MCS-51, postanowiłem odłożyć na bok pozostałe do omówienia bloki funkcjonalne procesora 8051 (port szeregowy, system przerwań, specjalne tryby pracy), a „wrzucić” Ci, drogi Czytelniku garść informacji pochodzącej trochę z „innej beczki”. Chodzi mianowicie o krótkie, aczkolwiek wystarczające zapoznanie się z podstawowymi pojęciami dotyczącymi obsługi tych „wszystkich mądrości”, o których od kilku miesięcy uważnie czytasz – czyli o sposób programowania, czyli: „czym?, jak?, i dlaczego?... się ten procesor programuje”. Decyzja moja jest po części Waszą, prawdopodobnie przeciw większości z Was ma już swój „Komputer edukacyjny”, którego konstrukcja została opisywana w dwóch poprzednich numerach EdW. A skoro wydałeś na to swoje oszczędności, to dobrze by było, nie patrzeć tylko na te „cudeńko” ale je w końcu wypróbować!



Część 6 Assembler – język maszynowy procesora

Do wspólnego zrealizowania tego zadania niezbędnych jest kilka informacji oraz zrozumienie pojęcia „programowania” układu scalonego – w naszym przypadku mikroprocesora.

Zanim przejdę do wyjaśnienia tych pojęć, pragnę uspokoić wszystkich drobiazgowych Czytelników, że wspomniane pozostałe bloki funkcjonalne procesora omówię w następnych odcinkach naszego cyklu, ale tym razem... uwaga: na gorąco, czyli z wykorzystaniem naszego edukacyjnego układu.

Tym wszystkim, którzy nie zaopatrzyli się w dedykowany temu kursowi, komputer, radzę o jak najszybsze zmontowanie go, dzięki czemu będziecie, moi drodzy, na bieżąco praktycznie wykonywać wszystkie zadania.

A więc zacznijmy od najważniejszego stwierdzenia: „Mikrokontroler bez programu jest jak żołnierz bez...” wiesz bez czego. I jest to święta racja. Jak sam zdążyłeś się zorientować śledząc poprzednie odcinki kursu, że kostka 8051 zawiera w swojej strukturze całe mnóstwo użytecznych elementów takich jak np. pamięć programu, danych, programowane układy licznikowe, stos, itd. Wszystko fajnie... „tylko jak nad tym wszystkim zapamiętać?...”. Otóż aby odpowiednio wykorzystać zasoby kontrolera i zmusić je do pracy zgodnie z naszym zamysłem i przeznaczeniem konstruowanego układu, po-

trebny jest język porozumiewania się z mikroprocesorem. Ten język najczęściej nazywa się językiem maszynowym. Brzmi bardzo poważnie, prawda? Tak na prawdę język ten jest prostym zbiorem poleceń, dzięki którym możliwa jest nie tylko ingerencja we wszystkie wspomniane bloki funkcjonalne procesora, ale także wykonywanie określonych logicznych czynności: sprawdzanie warunków czy operacje arytmetyczno-logiczne. Sam język maszynowy to na pozór nieczytelny dla człowieka ciąg liczb. Aby sprawę uprościć stworzono postać jawną języka maszynowego – assembler. W języku tym kolejne polecenia opisywane są za pomocą słownych instrukcji uzupełnionych odpowiednimi do danej sytuacji argumentami. Tak postać jest akceptowalna przez programistę, dzięki temu program pisze się po prostu w dowolnym edytorze tekstowym (np. na komputerze), następnie dokonuje się zamiany (translacji) tak napisanego programu na wspomniany ciąg liczb – czyli język maszynowy procesora.

Tak jak istnieje na świecie wiele języków porozumiewania się między ludźmi, tak samo w rodzinach różnych mikroprocesorów, często pochodzących od różnych producentów, istnieje wiele języków, wszystkie jednak to języki maszynowe.

Producenci oprogramowania tworzą bardziej zaawansowane tzw. języki

„wyższego poziomu”, slyszaleś zapewne o takich jak Pascal, C, Basic oraz inne. Tak naprawdę to są to translatory bardziej złożonych poleceń danego języka „wyższego poziomu” na kod maszynowy procesora. Zawsze na samym końcu obróbki programu użytkownika, czy powstał on w takim czy innym języku powstaje i zawsze kod maszynowy, który akceptuje dedykowany mikroprocesor.

W przypadku pisania większości programów na kontrolery 8051 (lub każde inne „jednoukładowce”), szczególnie podczas nauki, każdy początkujący musi poznać jego język maszynowy, czyli assembler. Dzięki temu później, kiedy nauczy się nim biegle posługiwać, będzie mieć dużą swobodę i możliwość obiektywnej oceny w wyborze dowolnego innego narzędzia wspomagającego programowanie tego procesora, ale to zupełnie inna historia.

Ogólnie można powiedzieć, że pisanie programu na procesor to po prostu tworzenie kolejnych poleceń, z których w efekcie powstaje cały program. „Tworzenie” to np. pisanie w dowolnym edytorze tekstowym ASCII w przypadku posiadaczy komputerów. Pozostałe osoby nie mające dostępu, mogą taki program napisać chociażby na kartce papieru (aczkolwiek w przypadku większych programów jest to bardzo trudne, czy wręcz niemożliwe).

Polecenie zwykle zawiera się w jednej linii programu. Jego składnia jest zasadniczo określona, z reguły można ją przedstawić jako:

instrukcja <argument_1>, <argument_2>.....(1)

Czasami przed instrukcją występuje także etykieta zakończona znakiem „:” (dwukropka), np. „etyk01:”, której zadaniem jest po prostu nazwanie danej linii polecenia. Ilość argumentów w linii polecenia może być różna w zależności od rodzaju instrukcji. W przypadku asemblera procesora 8051 i pochodnych liczba ta waha się od zera w przypadku instrukcji bezargumentowych do trzech. Zasadą jest że poszczególne argumenty oddziela się zawsze znakiem „,“: (przecinka), natomiast instrukcja oddzielona jest od pierwszego argumentu spacją (odstępem) lub znakiem tabulacji (to informacja dla komputerowców).

Podajmy przykład polecenia dla procesora 8051 w którym procesor wykonuje dodawanie zawartości dwóch jego rejestrów:

```
ADD A, B (2)
```

W poleceniu tym wystąpiły:

- instrukcja: „ADD”
- argumenty: A – rejestr A (akumulator) oraz rejestr B

W nazewnictwie asemblerowym pierwszą część polecenia, czyli instrukcję (np. ADD) nazywa się „mnemonikiem”, toteż od tego momentu będziemy posługiwać się tym zwrotem. Aby otrzymać wynik dodawania należy dodać do siebie dwa argumenty. Pierwszy argument (jakaś liczba) znajduje się w akumulatorze procesora 8051, druga zaś w tym przypadku w rejestrze B. W wyniku wykonania przytoczonego polecenia procesor doda do zawartości rejestru A wartość z rejestru B, a wynik umieści w rej. A (akumulatorze). Powiedzmy że chcemy dodać dwie liczby: 25 + 43. W tym celu musimy wpisać te wartości (składniki dodawania) do rejestrów procesora a potem jej dodać poleceniem (2). Można to zrobić w sposób następujący:

```
MOV A, #25 (3)
```

```
MOV B, #43 (4)
```

```
ADD A, B (5)
```

W liniach 3 i 4 rozkazaliśmy procesorowi wpisanie składników odpowiednio do rejestrów A i B, działanie polecenia w linii (5) jest Ci już znane.

Osoby znające asembler i możliwości 8051 pewnie się trochę zaśmieją z przykładu (3)...(5), bowiem dodawanie 2 stałych da się zapisać krócej w dwóch liniach, lecz nie o to nam chodzi, przynajmniej na tym etapie kursu.

Mnemonik „MOV” ma wiele zastosowań w języku procesorów '51, na język polski można ją przetłumaczyć jako „przemieszczenie” (przesunięcie) czegoś

gdzieś. Czego i gdzie to zależy od kontekstu, czyli od rodzaju argumentów, ale o tym za chwilę. Linia (3) oznacza dosłownie w tłumaczeniu na jęz. polski: „wpisz do akumulatora liczbę 25”, linia (4) – wpisz do rejestru B liczbę 43, linia (5) – dodaj do akumulatora zawartość rejestru B. W efekcie wykonania tych 3 poleceń w akumulatorze będzie wynik dodawania, czyli liczba 68 (binarnie będzie to 01000100). Jeżeli teraz każesz procesorowi wykonać np. polecenie

```
MOV P1, A (6)
```

spowoduje to że liczba 68 pojawi się na końcówkach portu P1 procesora, oczywiście w postaci binarnej. Aby to sprawdzić naocznie wystarczy dołączyć do każdej z 8-miu jego końcówek (piny 1...8 '8051/2) diodę świecącą z rezystorem, z pewnością zapalone zostaną diody dołączone do końcówek: 3 i 7 (na tych pozycjach liczby 68 w postaci binarnej jest „1”). Procesor po prostu w wyniku podania polecenia (6) wpisał zawartość akumulatora do rejestru portu P1.

No ale na razie wystarczy przykładów, wyjaśnimy sobie teraz, jak fizycznie procesor akceptuje instrukcje, no bo przecież „nie zje” od razu ciągu liter układających się w napis chociażby z przykładu (6). Otóż asembler procesora 8051 posiada określoną liczbę mnemoników oraz określone rodzaje argumentów.

Jeżeli jesteś ciekawy to powiem Ci że mnemoników jest nie tak wiele bo 42. W przyszłości będziesz musiał je poznać i zapamiętać, ale nie przejmuj się, nie jest to dużo, przy okazji praktycznego programowania same wpadną ci do głowy – i już zostaną...na zawsze.

Jeżeli chodzi o argumenty to wyróżnia się kilka ich rodzajów, nie będę teraz szczegółowo wyliczał ich wszystkich, pierwszy ich rodzaj – argument bezpośredni, poznałeś w przykładzie dodawania dwóch rejestrów – liczby: #25 i #43.

W zależności od rodzaju argumentów jakie występują po mnemoniku, różne jest działanie całego polecenia – instrukcji. W architekturze procesorów 8051 konstruktorzy wyróżnili 255 takich sytu-

cji i ponumerowali je od 0 do 255 (FFh szesnastkowo).

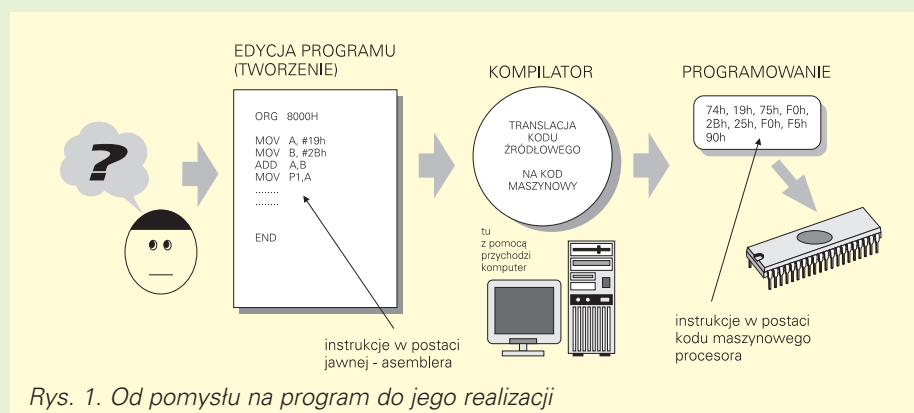
Tak powstało 255 instrukcji procesora (nie 256 bo jeden numer nie jest wykorzystany).

I jak się pewnie niedługo przekonasz z punktu widzenia programisty – użytkownika liczba ta jest mniejsza, to jednak należy zapamiętać ten fakt.

Jeżeli zatem cały zbiór instrukcji procesora daje się przedstawić jako liczba wraz z towarzyszącymi jej ewentualnie argumentami, z których każdy także daje się przetłumaczyć na liczbę, w efekcie można wywnioskować, że cały program pisany przez użytkownika w postaci źródłowej (literowej) można przetłumaczyć na ciąg liczb. Mało tego, wszystkie te liczby mogą zawierać się jedynie w 8 bitach, co idealnie pasuje architektury naszego procesora – jest on przecież prawdziwym ośmiobitowcem. Zamieniony do takiej postaci Twój program wystarczy teraz śmiało wpisać do poszczególnych komórek pamięci EPROM bądź samego procesora (gdy ten pracuje w trybie z wewnętrzną pamięcią programu – np.87C51) lub do kostki EPROM z której później procesor będzie pobierał instrukcje.

Operację programowania pamięci EPROM przyprowadza się oczywiście za pomocą specjalizowanych narzędzi do programowania procesorów – tzw. programatorów. Na pocieszenie powiem Ci że w ofercie handlowej AVT pod nazwą AVT-320 znajduje się taki programator idealnie nadający się do programowania wszystkich dostępnych na rynku kostek serii MCS-51. W przyszłości, kiedy nabędziesz już umiejętności swobodnego „surfowania” (to ostatnio bardzo popularne słowo) wśród rodziny '51-nek, z pewnością takie narzędzie Ci się przyda. To przyszłość, na razie do tego etapu jeszcze wspólnie nie doszliśmy.

Całą drogę od pomysłu na program poprzez jego napisanie i zamianę do postaci akceptowanej przez procesor i jednocześnie dającej wpisać się do pamięci programu procesora obrazuje **rysunek 1**.



Rys. 1. Od pomysłu na program do jego realizacji

Też to potrafisz

Wróćmy na chwilę do naszego prostego przykładu dodawania dwóch liczb. Zapiszmy instrukcje (3)...(5) z argumentami dodawania w postaci szesnastkowej:

```
MOV A, #19h (7)
MOV B, #2Bh (8)
ADD A, B (9)
```

Liczba 25 dziesiętnie można zapisać w postaci heksadecymalnej jako „19”, a liczba 43 jako „2B”, dodając na końcu każdej z nich małą literkę „h” co oznacza zapis że zapisaliśmy liczbę w takiej właśnie postaci. Spróbujmy teraz zamienić te trzy linie na postać liczbową (bajtową) akceptowaną przez procesor. W liście rozkazów 8051 instrukcja:

```
MOV A, "jakaś_liczba_8_bitowa"
```

ma numer (odtąd będziemy ten numer nazywać kodem rozkazu): „74h”. Ale w linii (7) występuje jeszcze argument – liczba stała „19h”, dlatego ostatecznie linię tę w kodzie maszynowym (liczbowo) można zapisać jako: „74h, 19h”. Podobnie tłumaczymy linię (8). Odwołując się do listy instrukcji (którą całą niebawem poznasz), sekwencję:

```
MOV B, "jakaś_liczba_8_bitowa"
```

tłumaczymy jako: „75h, F0h, 2Bh”. Trzecia linia zaś będzie miała postać: „25h, F0h”. Skąd to wszystko wiem? Ano ze wspomnianej listy instrukcji. Nie martw się w tej chwili nie jest Ci ona potrzebna, ważne jest abyś uzmysłowił sobie w jaki sposób pisze się program w języku assemblera i jak go potem tłumaczy się na język maszynowy procesora – czyli postać liczbową.

W efekcie po przetłumaczeniu naszego przykładu otrzymamy sekwencję liczb: „74h, 19h, 75h, F0h, 2Bh, 25h, F0h „ (10)

Jeżeli teraz poszczególne liczby wpiszesz do kolejnych komórek pamięci programu (czy to zewnętrznej czy to wewnętrznej) to po uruchomieniu układu procesor wykona dokładnie to czego do niego oczekujesz, czyli załaduje dwa składniki do dwóch rejestrów procesora a następnie dokona operacji dodania ich.

Uff, wyglądało to dość mozolnie, bo trzeba było napisać instrukcje w postaci assemblerowej czyli jawnej (linie 7,8,9), potem na podstawie bliżej Ci nie znanej (na razie) tabeli instrukcji zamienić program do postaci maszynowej, wreszcie umieścić sekwencję w pamięci programu.

W praktyce dzięki zastosowaniu dowolnego komputera proces ten da się przyspieszyć.

O ile każdy program w postaci assemblera trzeba wstukać z klawiatury i zapisać na dysku w postaci pliku tekstowego (np. korzystając z edytora popularnego Norton Commandera), to do przetłumaczenia postaci źródłowej za wykonywalną (maszynowej) służą specjalne narzędzia

(programy) zwane kompilatorami. Dzięki nim proces zamiany – zwany dalej kompilacją – kodu źródłowego na maszynowy trwa często bardzo krótko, a tłumaczenie nawet kilku tysięcy linii nie trwa dłużej niż kilkanaście sekund. W efekcie działania kompilatora powstaje zbiór z programem zapisany w postaci maszynowej, czyli ciągu liczb jak zilustrowałem na przykładzie dodawania liczb. Plik taki najczęściej jest gotowy do użycia przez programatory pamięci EPROM (lub programatory procesorów). Taki zbiór świetnie nadaje się też do zapisania w pamięci operacyjnej twojego komputera edukacyjnego opisywanego w trzech ostatnich numerach EdW. Jeżeli posiadasz dowolny komputer klasy PC, będziesz mógł nabyć dyskietkę z takim kompilatorem, dzięki któremu efekty twojej pracy będą natychmiastowe. Szczegółowe informacje zawarte są w 3 części opisującej komputerek edukacyjny na 8051 w tym numerze EdW.

Jeżeli nie masz dostępu do komputera, będziesz zmuszony do tłumaczenia przykładów z naszego kursu, ręcznie na kartce papieru, korzystając ze „ściągawki”, którą będzie lista instrukcji procesora 8051. Lista taka ukaże się

w przyszłym numerze EdW. Będziesz ją mógł wyciąć i w razie potrzeby zafolować, chroniąc ją tym samym przez zniszczeniem.

I choć wszystkie przykłady w czasie programowania będą dość proste i dające się zrealizować „na papierze”, to powinieneś już teraz pomyśleć o wyposażeniu swego domowego kącika, nawet w przestarzały komputer klasy AT-286 lub w muzealną wersję XT.

Użytkownicy komputerów innych rodzajów, np. Amiga, Commodore, Atari, posiadający interfejs szeregowy zgodny z RS232c będą mogli także ich używać, do przesyłania kodu maszynowego pisanych przez siebie programów z komputera do naszego systemiku edukacyjnego. Musicie jednak kochani poszperać wśród swoich kolegów i namierzyć kompilator na procesory 8051 działający na waszym komputerze, bowiem na dyskietce oferowanej do naszego kursu znajduje się zestaw programów na komputery klasy PC.

W tym odcinku dość nietypowo, umieszczamy pierwsze 5 ćwiczeń w części III opisu zestawu AVT-2250. Zapraszam do lektury w tym numerze EdW.

Sławomir Surowiński