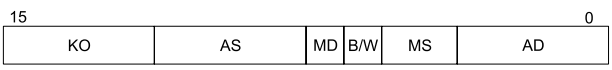
MSP430, część 2
Co ma takiego, czego inne nie mają?

Polscy konstruktorzy aparatury elektronicznej i bardziej ambitni amatorzy zaczynają coraz wyraźniej przekonywać się do mikrokontrolerów serii MSP430. Całkiem słusznie, nie wiadomo tylko, dlaczego z tak dużym opóźnieniem – MSP430 ma już przecież sporo ponad dziesięć lat! Czym wyróżnia się ten mikrokontroler na tle wielu innych, częstokroć nowszych, powszechnie znanych, łatwo dostępnych globalnie i masowo stosowanych?

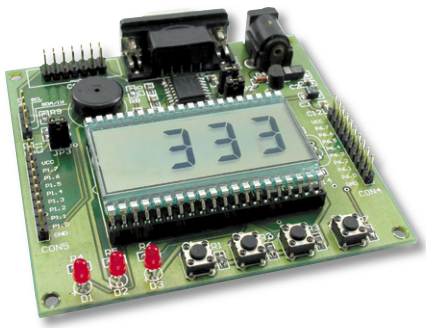
Sposób kodowania instrukcji dwuargumentowych jest przedstawiony na **rys. 3**.

Każdy rozkaz zawiera 4–bitowy kod operacji KO, co teoretycznie pozwala na zakodowanie 16 instrukcji. W rzeczywistości rozkazów dwuargumentowych jest tylko 12; pozostałe kombinacje wykorzystywane są przez instrukcje jednoargumentowe i skoki, w których odmienna jest interpretacja reszty słowa rozkazowego. Jest to często stosowana „sztuczka" konstrukcyjna. Jeden bit (B/W) zarezerwowano dla rozróżniania operacji na argumentach ośmio– i szesnastobitowych. Dwa 4–bitowe pola AS i AD, każde wskazujące jeden z 16 rejestrów uniwersalnych, stanowią części adresowe operandów. Interpretacja zawartości pierwszego z rejestrów zależy od związanego z nim dwubitowego modyfikatora adresu pierwszego argumentu (źródła, MS), co może sugerować, że dostępne są dla niego tylko 4 tryby adresowania (konkretnie: bezpośrednie rejestrowe, pośrednie, pośrednie z autoinkrementacją i indeksowe). Jednakże adresowanie indeksowe, w którym dla uzyskania adresu argumentu należy do zawartości wskazanego rejestru dodać umieszczoną po rozkazie 16–bitową liczbę (przesunięcie), przekształca się w adresowanie względne bieżące, jeżeli rejestrem jest licznik rozkazów. Podobnie adresowanie pośrednie z autoinkrementacją w odniesieniu do licznika rozkazów staje się po prostu adresowaniem natychmiastowym. Dana zawarta w słowie pamięci bezpośrednio za instrukcją jest wczytywana jako argument rozkazu, a licznik rozkazów jest inkrementowany i wskazuje następną instrukcję programu. Poza tym nie miałoby żadnego sensu wykorzystywanie rejestru statusu, zawierającego znaczniki, jako rejestru indeksowego. Próba taka jest wykrywana przez układ sterowania i zamiast zawartości rejestru R2 „podkładana" jest wartość zerowa, przy czym oryginalna zawartość rejestru nie ulega faktycznej zmianie. W takiej sytuacji adresowanie indeksowe przekształca się po prostu w bezpośrednie, bo przesunięcie dodawane jest do zera wskazując bezpośredni (absolutny) adres argumentu. W efekcie oryginalna koncepcja umieszczenia wśród równouprawnionych rejestrów uniwersalnych także licznika rozkazów i rejestru znaczników umożliwiła zakodowanie aż 7 trybów adresowania na zaledwie 2 bitach. Podobnie dla drugiego argumentu i wyniku – jednobitowy znacznik pozwala na wybór 4 trybów adresowania: rejestrowego, względnego bieżącego, indeksowego i bezpośredniego. Brak w tym przypadku adresowania

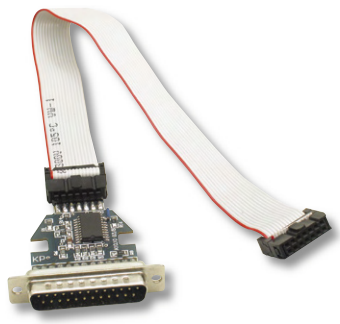


Rys. 3. Kodowanie instrukcji dwuargumentowych mikrokontrolera MSP430

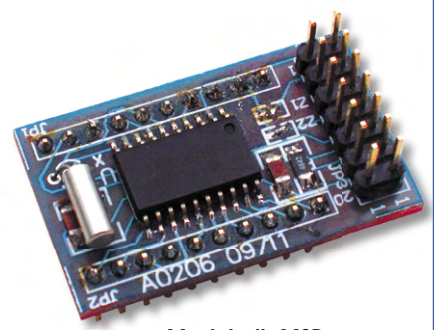
Za miesiąc przypomnimy nasze projekty na mikrokontrolerach z rodziny MSP430, m.in.:



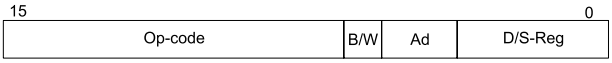
Zestaw uruchomieniowy



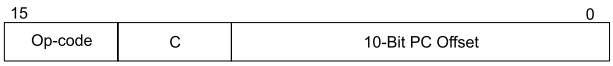
Programator JTAG



Moduł dipMSP



Rys. 4. Kodowanie instrukcji jednoargumentowych mikrokontrolera MSP430



Rys. 5. Kodowanie instrukcji skoków mikrokontrolera MSP430

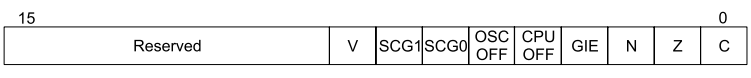
pośredniego nie stanowi wielkiej niedogodności, bo zawsze można zastosować w zamian adresowanie indeksowe z zerowym przesunięciem.

Siedem rozkazów jednoargumentowych rozpoczyna się unikalną kombinacją 0001, po której występuje dalszy ciąg kodu konkretnej operacji, wyróżnik bajt/słowo, a następnie 4–bitowy adres rejestru i dwubitowy modyfikator adresu identyczne, jak w rozkazach dwuargumentowych. Są zatem dostępne wszystkie z 7 trybów adresowania. Przedstawia to **rys. 4**.

Wyodrębnione rozkazy skoków warunkowych rozpoczynają się kombinacją 001, uzupełnioną trzema bitami kodującymi jeden z 8 konkretnych warunków skoku i 9–bitowym adresem względnym ze znakiem (**rys. 5**). Ponieważ skoki zawsze muszą być wykonywane pod adresy parzyste, więc nie istnieje potrzeba zapamiętywania najmłodszego, zawsze wyzerowanego bitu adresu skoku. W efekcie uzyskano imponująco duży zakres skoków względnych: aż ±512 słów (w nowocześniejszych przecież AVR–ach są to tylko ±64 słowa).

W MSP430 zastosowano jeszcze jedną sprytną „sztuczkę" konstrukcyjną – tzw. generator stałych (*constant generator*) Jak już wspomniano nieco wyżej, w stosunku do rejestru statusu sensowne jest użycie tylko jednego trybu adresowania, czyli adresowania rejestrowego. Wybór każdego innego trybu nie zmienia zawartości rejestru, lecz pociąga za sobą wygenerowanie pewnej stałej programowej interpretowanej jako dana natychmiastowa. W ten sposób nie musi być ona fizycznie wpisywana w programie, nie zajmuje słowa w pamięci i nie trzeba tracić czasu na jej odczyt. Jako takie predefiniowane stałe wybrano 6 najczęściej występujących w rzeczywistych programach wartości: –1 (czyli 0FFFFh), 0, 1, 2, 4 i 8. Niektóre z nich generowane są przez odwołanie do rejestru statusu (R2), dla innych przeznaczono specjalny rejestr R3, który w takiej sytuacji nie może być wykorzystywany do żadnych innych celów. Poświęcenie jednego rejestru jest w sumie opłacalne, w zamian bowiem pozwala miedzy innymi na usunięcie z listy rozkazów całkiem sporej liczby zbędnych instrukcji. Przykładowo niepotrzebne stają się instrukcje inkrementacji i dekrementacji, nie tylko o jeden, ale także o dwa, bo zastąpić je można bez żadnych strat w objętości kodu i szybkości wykonywania zwykłymi rozkazami dodawania i odejmowania. Predefiniowane stałe wartości mogą być pomocne także w operacjach bitowych. Aby sprawdzić stan jakiegoś bitu najprościej jest wykonać operację iloczynu logicznego z maską, zawierającą jedynkę na testowanej pozycji. Bity o numerach 0, 1, 2 i 3, czyli o wagach 1, 2, 4 i 8 mogą być testowane najszybciej i bez konieczności przechowywania odpowiedniej maski w pamięci. Stanowi to wskazówkę przy optymalizacji programu. Tak właśnie rozmieszczono najczęściej wykorzystywane znaczniki N, Z i C w rejestrze SR (**rys. 6**).

Umieszczenie wśród rejestrów uniwersalnych wskaźnika stosu systemowego także pociąga za sobą znaczące konsekwencje. W większości mikrokontrolerów na stos można tylko jakąś daną wysłać lub ją ze stosu zdjąć. Obie operacje związane są z automatyczną zmianą położenia wierzchołku stosu, czyli zawartości rejestru SP. W niektórych mikrokontrolerach nawet to nie jest możliwe (PIC–e). Bardzo rzadko da się odczytać lub tym bardziej zmodyfikować zawartość wierzchołka stosu bez zmiany wskaźnika stosu, a możliwość wykonywania jakichś działań na elementach stosu poza jego szczytem praktycznie nie występuje. Tymczasem w MSP430 wszystko to jest jak najbardziej dostępne, stwarzając przy okazji zupełnie nowe możliwości programistyczne, bo wskaźnik stosu jest takim samym rejestrem jak wszystkie inne i można do niego stosować wszystkie tryby adresowania, w tym indeksowe. Warto także zaznaczyć, że poza stosem systemowym można także zaimplementować dodatkowy stos użytkownika, a nawet kilka takich stosów, o dowolnej organizacji – bajtowej lub słowowej. Nawiasem mówiąc, w MSP430 nie ma odrębnej instrukcji odczytu danej ze stosu (POP), bo zapewnia to już automatycznie tryb adresowania pośredniego z autoinkrementacją. Instrukcja wysyłania na stos (PUSH) musi istnieć, bo adresowania pośredniego z predekrementacją już się w żaden sposób nie udało w 16–bitowyjm słowie rozkazowym mikrokontrolera upchnąć.



Rys. 6. Rejestr znaczników mikrokontrolera MSP430

Ortogonalność MSP430 pociąga za sobą bardzo ciekawe konsekwencje w instrukcjach skoków i – przede wszystkim – skoków ze śladem (CALL). Są one traktowane tak, jak każdy inny rozkaz i wobec tego można w nich używać wszystkich siedmiu trybów adresowania. Zazwyczaj programista ma do dyspozycji tylko wywołanie podprogramu spod adresu bezpośredniego, zawartego w rozkazie i stanowiącego stałą programową. Znacznie rzadziej istnieje możliwość skoku ze śladem pod adres wyliczany, który można w trakcie realizacji programu modyfikować. Tymczasem w MSP430 mamy pełną dowolność, z adresowaniem indeksowym włącznie. Rozpatrzmy bardzo typową, wziętą z życia sytuację: niech system mikroprocesorowy realizuje jedną z – powiedzmy – 16 funkcji, przekazywanych mu przez standardowy port szeregowy. Kodowanie tych poleceń jest najprostsze z możliwych, gdyż stanowią one binarną liczbę zawartą w najbardziej znaczących 4 bitach transmitowanej informacji. Najmłodsze 4 bity zarezerwowane są dla przyszłych rozszerzeń i/lub przekazywania pewnych parametrów związanych z poleceniami. W '51 program obsługi rozpoczynałby się zapewne następująco:

```
MOV A,SBUF                ;odbiór polecenia
ANL A,#0F0h               ;skasowanie najmłod-
szych 4 bitów
CJNE A,#00h,nie_pierwsze      ;sprawdzenie nu-
meru polecenia
JMP pierwsze_polecenie
nie_pierwsze:       CJNE A,#10h,nie_drugie
JMP drugie_polecenie
nie_drugie:         CJNE A,#20h,nie_trzecie
                JMP trzecie_polecenie
nie_trzecie:        CJNE A,#30h,nie_czwarte
```

i tak w sumie aż 16 razy.

Gdyby '51 miał takie możliwości, jak MSP430, to program mógł-

by wyglądać następująco:

```
        MOV R5,SBUF
        ANL R5,#0F0h
        CALL @R5, początek_tablicy_z_pod-
programami
        następny rozkaz programu
```

Prawda, że prościej, bardziej przejrzyście i elegancko? W instrukcji CALL zastosowano nieistniejący w rzeczywistości w '51 tryb adresowania indeksowego. Skok ze śladem następuje pod adres będący sumą adresu początku tablicy zawierającej 16 rozmieszczonych co 16 bajtów podprogramów i względnego adresu danego podprogramu w tablicy, znajdującego się w rejestrze R5. W przykładzie tym nie zastosowano oryginalnej transkrypcji i skrótów mnemonicznych MSP430, bo dla wielu Czytelników mogą się one wydawać na pierwszy rzut oka obce, a każdy interesujący się mikrokontrolerami mnemoniki '51 raczej zna. W przykładzie tym nawet ważniejsze od zwartości i szybkości wykonywania jest właśnie wykorzystanie instrukcji CALL, a nie JUMP. Ma to wielkie zalety, szczególnie wtedy, gdy te same polecenia mogą pochodzić z różnych źródeł (na przykład z portu szeregowego, modemu GSM lub klawiatury). Oczywiście nic nie stoi na przeszkodzie, by w razie potrzeby wykorzystać instrukcję JUMP.

Architektura jednostki centralnej mikrokontrolera MSP430 wydaje się być niezwykle starannie przemyślana i dopracowana pod względem nawet najdrobniejszych szczegółów – warto chociażby przypomnieć wspomniane wyżej rozmieszczenie najczęściej używanych znaczników w rejestrze statusu. Lista rozkazów wygląda na pierwszy rzut oka bardzo skromnie, ale bliższa analiza wykazuje, że właściwie żadnej istotnej instrukcji nie pominięto. Jest nawet rozkaz dodawania dwóch 16–bitowych liczb, reprezentujących 4 cyfry BCD każda. Bardzo obszerny jest asortyment skoków warunkowych. Wiele typowych rozkazów usunięto, bo mogą być one realizowane w inny sposób poprzez wykorzystywanie specyficznych cech jednostki centralnej. Dla ułatwienia pracy programistom te dodatkowe instrukcje są rozróżniane przez asembler i można z nich swobodnie korzystać – nazwano je instrukcjami emulowanymi i jest ich też 27. Przykładem mogą być wspomniane rozkazy skoków, inkrementacji, dekrementacji, zerowania komórki pamięci czy też rejestru i wiele innych. Nie ma rozkazów logicznych, ani przesłań operujących na pojedynczych bitach.

Maciej Nowiński