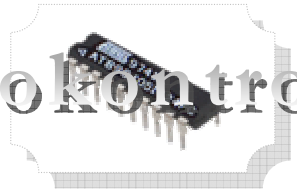
Wprowadzenie do programowania systemów mikroprocesorowych opartych na mikrokontrolerach rodziny MCS'51 - podstawy programowania w asemblerze



Mikrokontrolery

„Wprowadzenie do programowania systemów mikroprocesorowych opartych na mikrokontrolerach rodziny MCS’51 – podstawy programowania w asemblerze”

Wstęp... czyli po co i dlaczego to wszystko?

Praca ta (podręcznik?) powstała przede wszystkim z myślą o jej zastosowaniu przez uczniów klasy czwartej Technikum Nr 4 w Zespole Szkół Nr 6 im. Króla Jana III Sobieskiego w Jastrzębiu Zdroju. Mamy nadzieję, że będzie stanowić (cenną?) pomoc dydaktyczną podczas nauki programowania systemów mikroprocesorowych w ramach przedmiotu „Pracownia elektryczna i elektroniczna".

Nie można również pominąć faktu, iż do powstania tego podręcznika przyczynił się potrzeba stworzenia pracy dyplomowej na studiach podyplomowych w KISS'ie.

Wiemy już zatem po co, czas na odpowiedź na pytanie dlaczego?

- Jest wiele podręczników i materiałów na ten temat, ale żaden nie uwzględnia w pełni potrzeb szkoły w tym względzie tzn. albo są dostosowane do konkretnego systemu dydaktycznego np. tak jak książka autorstwa braci Gałków „Podstawy programowania mikrokontrolera 8051" dla Dydaktycznego Systemu Mikroprocesorowego DSM-51 firmy "MICROMADE", przez co zawężają punkt widzenia na układy mikroprocesorowe do jednej konfiguracji, albo bardzo ogólnie traktują temat, co nie jest wcale ich wadą a wręcz zaletą, ale wymaga większego poziomu przygotowania merytorycznego i zaangażowania ze strony czytającego.
  Powszechnie wiadomo, że jedno i drugie nie bardzo chcą iść w parze u tzw. przeciętnego ucznia, a wymagania są jednakowe dla wszystkich, jeżeli chodzi o minimum kompetencji zawodowych, stąd konieczność i próba połączenia „wody z ogniem", czyli uniwersalności z konkretem (czyli z DSM-51, bo w te zestawy wyposażona jest pracownia)
- Korzystanie z innych źródeł kosztuje (czasem nawet sporo), co zmusza niektórych do łamania praw autorskich. Ponieważ nasze źródło wiedzy tajemnej jest za friko, to może choć częściowo przyczyni się do zmniejszenia frustracji u cierpiących na niż finansowy lub ograniczy liczbę cierpiących na wyrzuty sumienia (?) wśród tych, którzy owszem mają wyż, ale nie bardzo lubią go trwonić (trudne słowo) na tak błahe cele jak kupowanie książek.

No i wreszcie...

- Skoro jest już tyle różnych źródeł to nie może powstać jeszcze jedno?

Głównym źródłem informacji użytych w tym opracowaniu jest Internet i strony głównych producentów układów mikrokontrolerów np. www.atmel.com.

Część informacji została zaczerpnięta lub powstała na podstawie materiałów zawartych w dokumentacji technicznej Dydaktycznego Systemu Mikroprocesorowego DSM-51 firmy MicroMade oraz „Podręcznym Katalogu Elektronika. Mikrokomputery jednoukładowe rodziny MCS-51." Andrzeja Rydzewskiego (WNT Warszawa 1997)

Na razie dostajemy do rąk (oczu?) pierwsze wydanie, które systematycznie będzie poszerzane i modyfikowane.

No to do dzieła!

1 Charakterystyka układów rodziny MCS'51... czyli słów kilka o mikrokontrolerze 8051.

Przedstawione tu informacje dotyczą podstawowych układów tej rodziny czyli 80C31, 80C51 oraz układów z nimi w pełni zgodnych np. AT 89C51. Wybór taki wynika z faktu zastosowania tych układów w zestawach dydaktycznych, które wykorzystywane są podczas zajęć w pracowni systemów mikroprocesorowych. Trzeba jednakże mieć świadomość, że dzisiaj rodzina tych układów to ogromna ilość różnych typów różniących się między sobą dodatkowymi wewnętrznymi układami scalonymi w strukturze (min. dekodery MP3, kontrolery USB). Informacje szczegółowe na ten temat można bez trudu uzyskać na stronach producentów tego typu układów takich jak ATMEL, SIEMENS czy PHILIPS oraz innych. To co jednak jest charakterystyczne to zgodność „w dół" z układami, które powstały jako pierwsze jeśli chodzi o listę rozkazów oraz podstawowe cechy.

Pod pojęciem ***mikrokontrolera*** rozumie się układ scalony z wyspecjalizowanym mikroprocesorem (inaczej ***mikrokomputer jednoukładowy***), którego konstrukcja spełnia dwa podstawowe kryteria:

- Zdolność do autonomicznej pracy, tzn. w najprostszych zastosowaniach nie wymaga przyłączenia zewnętrznych układów pomocniczych (przede wszystkim pamięci programu lub danych), sam stanowiąc niezależny system mikroprocesorowy
- Ze względu na przeznaczenie do pracy w systemach kontrolno-pomiarowych posiada rozbudowany system komunikacji z innymi elementami takich systemów zarówno przy użyciu sygnałów cyfrowych jak i analogowych.

1.1 Podstawowe cechy układów rodziny MCS'51.

Podstawowe cechy rodziny mikrokontrolerów '51:

- Układy 8-bitowe tzn. operacje przetwarzania danych wykonują na danych 8 bitowych
- Stała liczba instrukcji, niezależnie od rozbudowy układów wewnętrznych
- Multipleksowana zewnętrzna 8-bitowa magistrala danych i 16-bitowa magistrala adresowa
- Sprzętowe procedury dzielenia i mnożenia
- Wewnętrzna pamięć programu (opcjonalne) i danych
- Niezależny od jednostki arytmetyczno-logicznej układ transmisji szeregowej UART wygodny do zastosowania w standardzie RS 232C
- Duża liczba wejść/wyjść cyfrowych (od 16 do 48 linii w zależności od układu)
- Priorytetowy, 2-poziomowy układ kontroli przerwań
- Co najmniej dwa 16-bitowe układy czasowo-zliczające

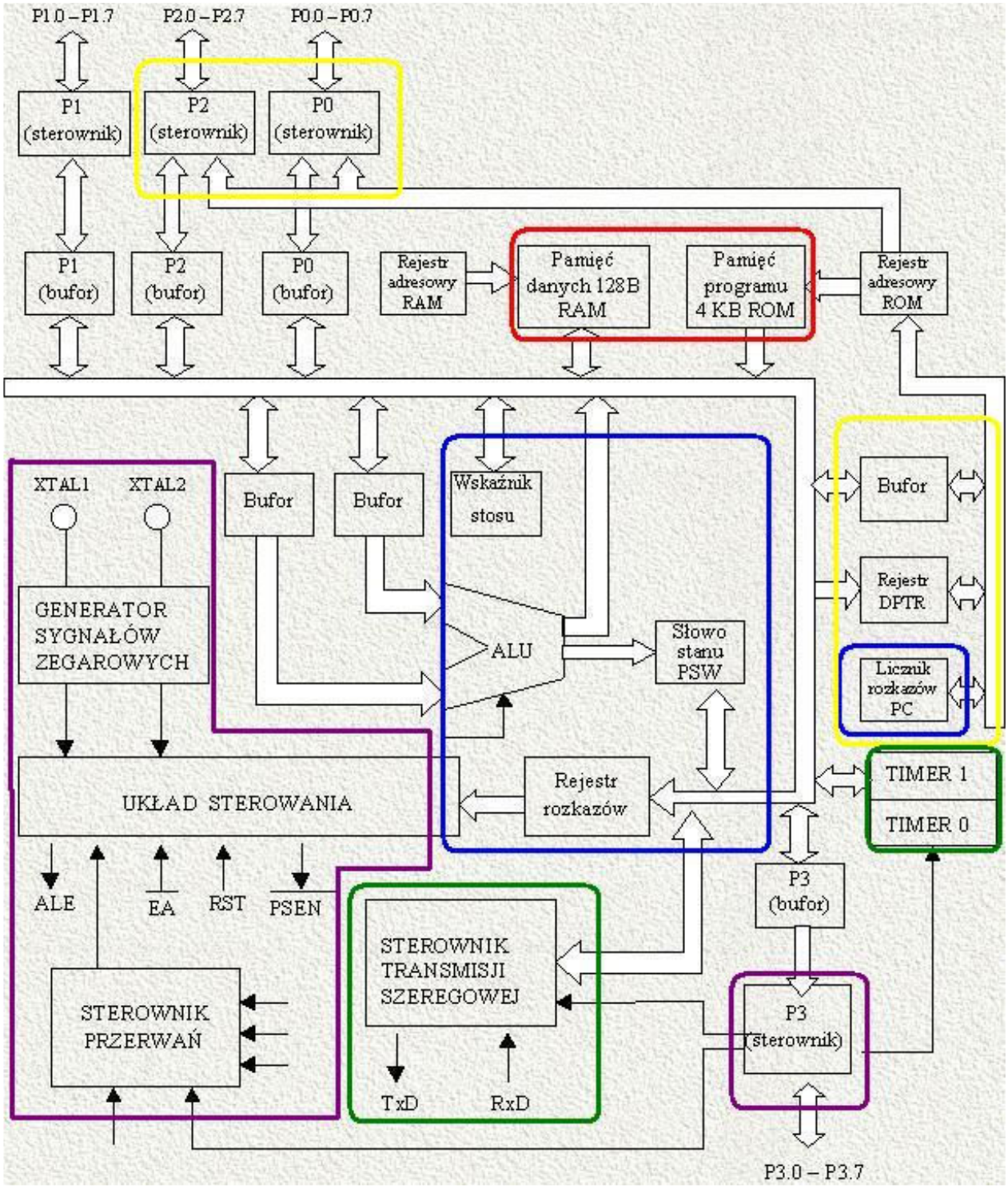
Podstawowe elementy struktury wewnętrznej są następujące:

- 8-bitowa magistrala danych
- 16-bitowa magistrala adresowa
- linie sterujące dostępem do zewnętrznej pamięci programu (/EA, /PSEN) i zewnętrznej pamięci danych (/RD, /WR)
- cztery 8-bitowe porty wejść/wyjść cyfrowych (P0, P1, P2, P3)
- układ portu szeregowego z programowaną szybkością transmisji, synchroniczną lub asynchroniczną
- dwa 16-bitowe liczniki
- kontroler przerwań reagujący na dwa sygnały zewnętrzne i trzy wewnętrzne wytwarzane przez układy liczników oraz układu transmisji szeregowej

- układ redukcji mocy pobieranej przez mikrokontroler

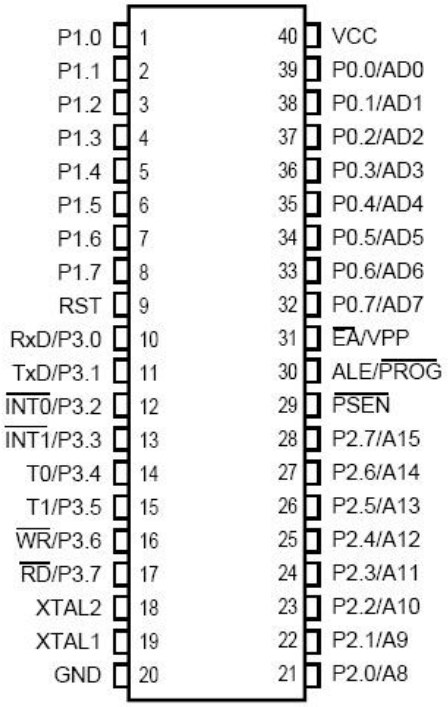
1.1.1 Schemat blokowy mikrokontrolera

Poniższy rysunek przedstawia schemat blokowy układu z zaznaczonymi podstawowymi blokami funkcjonalnymi:

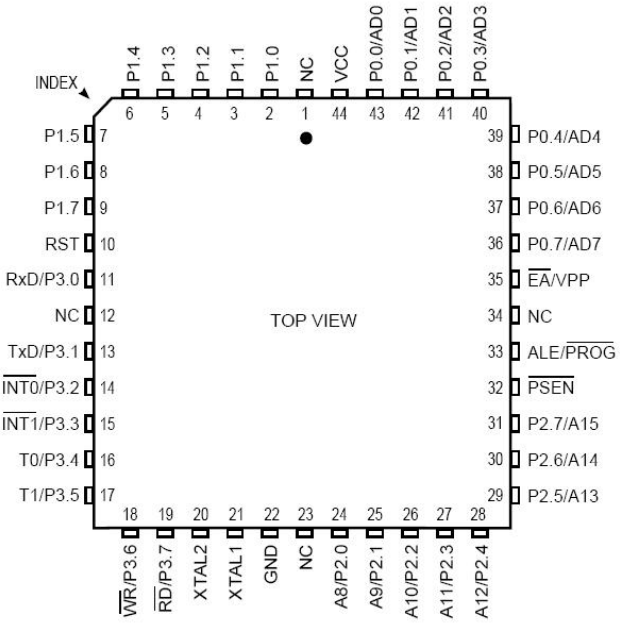


1.1.2 Opis wyprowadzeń układu

Rozkład wyprowadzeń układu mikrokontrolera 8051:



a) Typ 40 pin PDIP



b) Typ 44 pin PLCC

Szczegółowy opis wyprowadzeń:

Symbol	Numer wyprowadzenia dla obudowy		Kierunek I/O	Funkcja wyprowadzenia
	PDIP	PLCC		
ALE	**30**	**33**	**I/O**	**Address Latch Enable;** wyjście sygnału "zatrzaśnięcia" młodszego bajtu adresu w rejestrze magistrali adresowej podczas adresowania zewnętrznej pamięci. Normalnie sygnał generowany ze stałym współczynnikiem powtarzania równym 1/6 częstotliwości rezonatora systemowego i może być wykorzystany jako źródło sygnału zegarowego. Wytwarzany przy każdej operacji dostępu do zewnętrznej pamięci danych.
/EA	**31**	**35**	**I**	**External Access enable;** ustawienie z zewnątrz stanu niskiego na tym wejściu uaktywnia pobieranie przez układ kodu programu z zewnętrznej pamięci programu adresowanej w zakresie 0000H-0FFFFH; ustawienie z zewnątrz stanu wysokiego na tym wejściu wymusza pobieranie kodu programu z wewnętrznej pamięci programu w przypadku, kiedy zawartość licznika rozkazów jest mniejsza niż 0FFFH.
P0.0-P0.7	**39-32**	**43-36**	**I/O**	**Port 0** ; 8-bitowy dwukierunkowy port I/O typu „otwarty dren"; port wejściowy o wysokiej impedancji jeżeli ustawiony zostanie w stan wysoki poprzez zapis stanu 0FFH do rejestru portu; multipleksowana część magistrali adresowej (młodszy bajt adresu) i magistrali danych przy dostępie do zewnętrznej pamięci programu i danych (w tym trybie wykorzystuje wewnętrzny układ „podciągający" do potencjału zasilania).
P1.0-P1.7	**1-8**	**2-9**	**I/O**	**Port 1**; 8-bitowy dwukierunkowy port I/O z wewnętrznym układem „podciągającym"; jako wyjście może sterować 4 wejściami typu TTL; przy programowaniu i weryfikacji wewnętrznej pamięci ROM służy do przesyłania młodszego bajtu adresu.
P2.0-P2.7	**21-28**	**24-31**	**I/O**	**Port 2**; 8-bitowy dwukierunkowy port I/O z wewnętrznym układem „podciągającym; przy dostępie do zewnętrznej pamięci programu i danych służy do przesyłania starszego bajtu adresu magistrali adresowej; przy adresowaniu 8bitowym zewnętrznej pamięci danych (MOVX A,@Ri [i=0,1]) stan portu odpowiada zawartości rejestru specjalnego SFR P2
P3.0-P3.7	**10-17**	**11, 13-19**	**I/O**	**Port 3** ; 8-bitowy dwukierunkowy port I/O z wewnętrznym układem „podciągającym; Port P3 wykorzystywany jest także przez wewnętrzne układy mikrokontrolera jako:
	10	**11**	**I**	**RxD (P3.0)** wejście portu szeregowego
	11	**13**	**O**	**TxD (P3.1)** wyjście portu szeregowego
	12	**14**	**I**	**/INT0 (P3.2)** wejście przerwania zewnętrznego 0
	13	**15**	**I**	**/INT1 (P3.3)** wejście przerwania zewnętrznego 1
	14	**16**	**I**	**T0 (P3.4)** zewnętrzne wejście licznika T0
	15	**17**	**I**	**T1 (P3.5)** zewnętrzne wejście licznika T1
	16	**18**	**O**	**/WR (P3.6)** sygnał zapisu do zewnętrznej pamięci danych
	17	**19**	**O**	**/RD (P3.7)** sygnał odczytu z zewnętrznej pamięci danych
/PSEN	**29**	**32**	**O**	**Program Store Enable**; wyjście sygnału odczytu z zewnętrznej pamięci programu; przyjmuje stan aktywny dwa razy w każdym cyklu maszynowym (nieaktywny w przypadku pobierania instrukcji z wewnętrznej pamięci programu oraz przy dostępie do zewnętrznej pamięci danych)
				Reset; stan wysoki na tym wejściu przez czas równy

RST	**9**	**10**	**I**	dwóm cyklom maszynowym powoduje resetowanie układu (zapis adresu 0000H do licznika rozkazów PC)
XTAL1	**19**	**21**	**I**	**Crystal 1**; wejście odwracające wzmacniacza układu oscylatora oraz wejście wewnętrznego układu zegarowego
XTAL2	**18**	**20**	**O**	**Crystal 2** ; wyjście wzmacniacza układu oscylatora
GND	**20**	**22**	**I**	**Ground** ; masa układu zasilania
Vcc	**40**	**44**	**I**	**Power supply** ; wejście dodatniego potencjału zasilania

1.1.3 Magistrala multipleksowana.

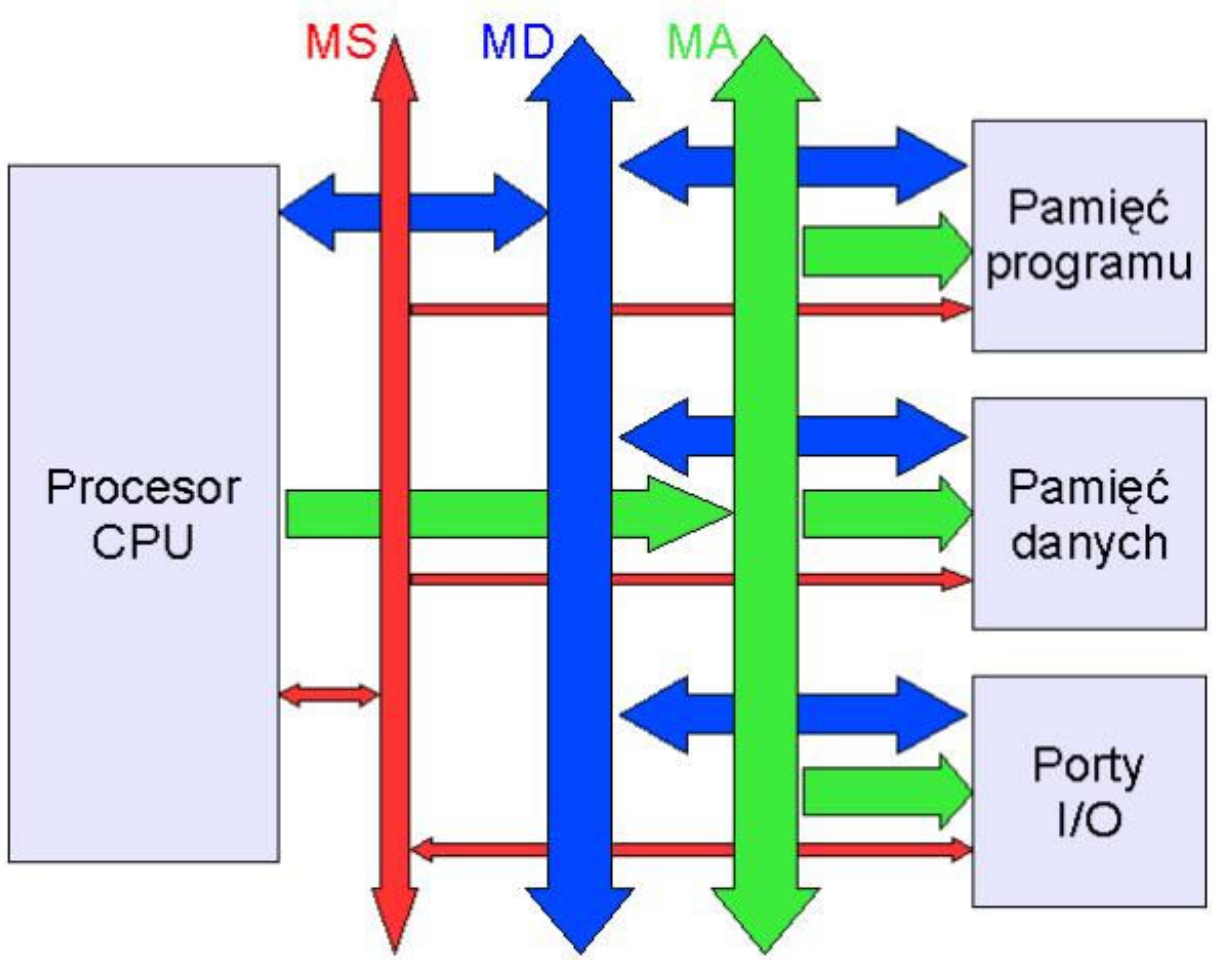
Pojęcie magistrali multipleksowanej systemu związane jest ze współpracą mikrokontrolera z układami zewnętrznymi, przede wszystkim z układami pamięci.

W klasycznym ujęciu aby stworzyć system mikroprocesorowy konieczne jest połączenie jednostki centralnej (CPU) czyli mikroprocesora z pamięcią programu, w której zapisany jest program działania jednostki centralnej, pamięcią danych, w której przechowywane są dane wykorzystywane przez procesor oraz układów portów wejścia/wyjścia (I/O) zapewniającymi współpracę systemu z urządzeniami zewnętrznymi takimi jak klawiatura czy wyświetlacz (monitor).

Z punktu widzenia technicznego działanie systemu w rzeczywistości sprowadza się jedynie do przesyłania danych pomiędzy poszczególnymi elementami systemu oraz ich przetwarzaniu przez procesor przy użyciu odpowiednich instrukcji. Zatem konieczne jest odpowiednie połączenie tych elementów zapewniające swobodny przepływ informacji w postaci elektrycznej. Rolę tą spełniają magistrale:

- Danych (MD) – jak wskazuje nazwa służy do przesyłu danych
- Adresowa (MA) – umożliwiająca wybór określonej informacji poprzez wskazanie jej źródła oraz miejsca zapisu; zasada działania systemów zakłada, że w danym momencie dostęp do magistrali danych może posiadać tylko jedno urządzenie będące źródłem danych oraz jeden odbiorca danych; identyfikacja poszczególnych urządzeń odbywa się przez nadanie im adresów czyli specyficznych danych przesyłanych za pomocą magistrali adresowej
- Sterująca (MS) – służy do zapewnienia odpowiedniej współpracy pomiędzy poszczególnymi zgodnie z opisaną powyżej zasadą poprzez przesył odpowiednich sygnałów sterujących (aktywacja układów oraz określenie typu operacji „zapis-odczyt” (RD-WR).

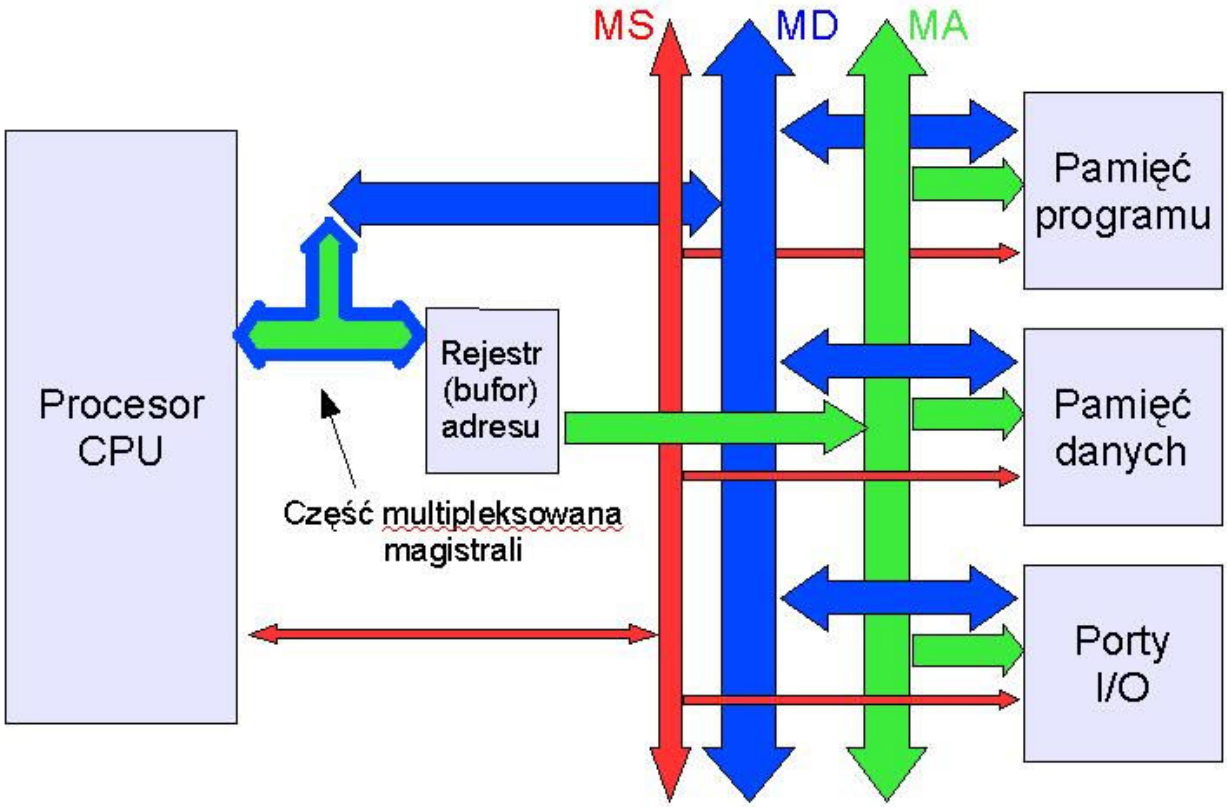
Powyższe założenia dla klasycznego ujęcia systemu ilustruje rysunek.



Magistrale łączą odpowiednie wyprowadzenia obudowy procesora oraz układów z nim współpracujących. Liczba tych wyprowadzeń uzależniona jest od rozmiarów magistral (liczby bitów danej lub adresu przesyłanych jednorazowo przez magistralę).

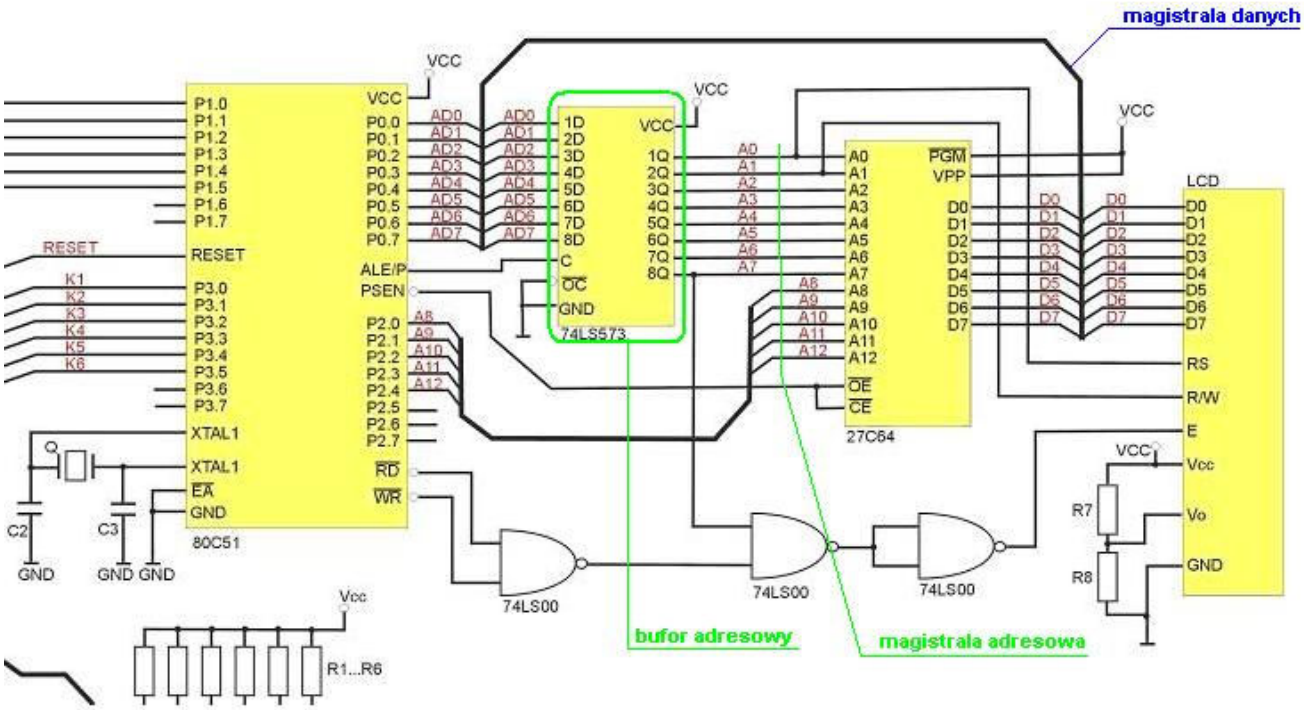
Aby zmniejszyć rozmiary obudowy procesora oraz ułatwić realizację techniczna magistral stosowany jest system tzw. multipleksowanej magistrali adresowej i danych. Polega on na tym, że te same linie magistrali wykorzystywane są zarówno do przesyłania danych jak i adresów. Wymaga to sekwencyjnego sterowania praca magistrali w jej części multipleksowanej oraz zastosowania dodatkowego układu rejestru, w którym przechowywany jest aktualny adres w trakcie przesyłania danych.

Sytuację tą ilustruje poniższy rysunek.



W przypadku mikrokontrolerów rodziny MCS'51 mamy do czynienia z możliwością wykorzystywania 8-bitowej zewnętrznej magistrali danych oraz 16-bitowej magistrali adresowej. W przypadku klasycznego rozwiązania systemu wymagałoby to wykorzystania 24 wyprowadzeń mikrokontrolera. Dzięki zastosowaniu multipleksacji magistrali wystarcza ich 16 (8-bitowy port P0 służy jako magistrala danych oraz jako część magistrali adresowej służąca do przesyłania 8 młodszych bitów adresu; drugą część magistrali adresowej stanowi port P2)

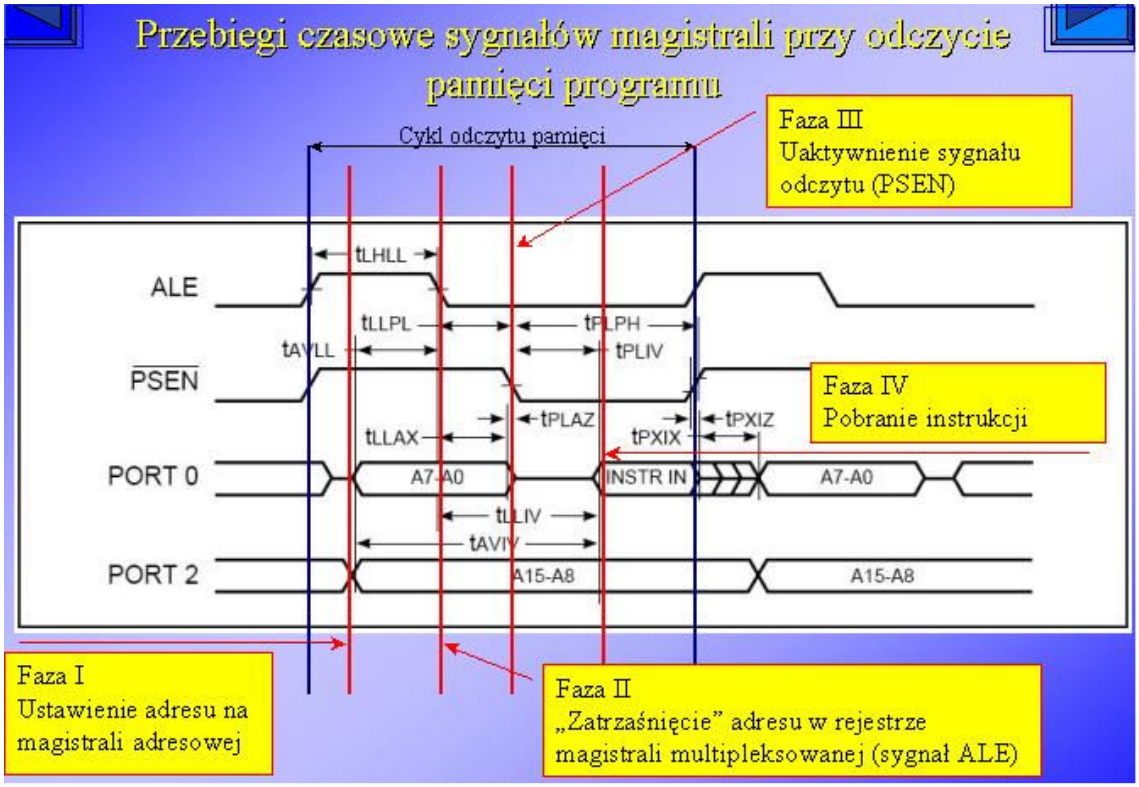
Ilustruje to rysunek przedstawiający fragment schematu ideowego systemu, w którym wykorzystano taki sposób rozwiązania magistrali.

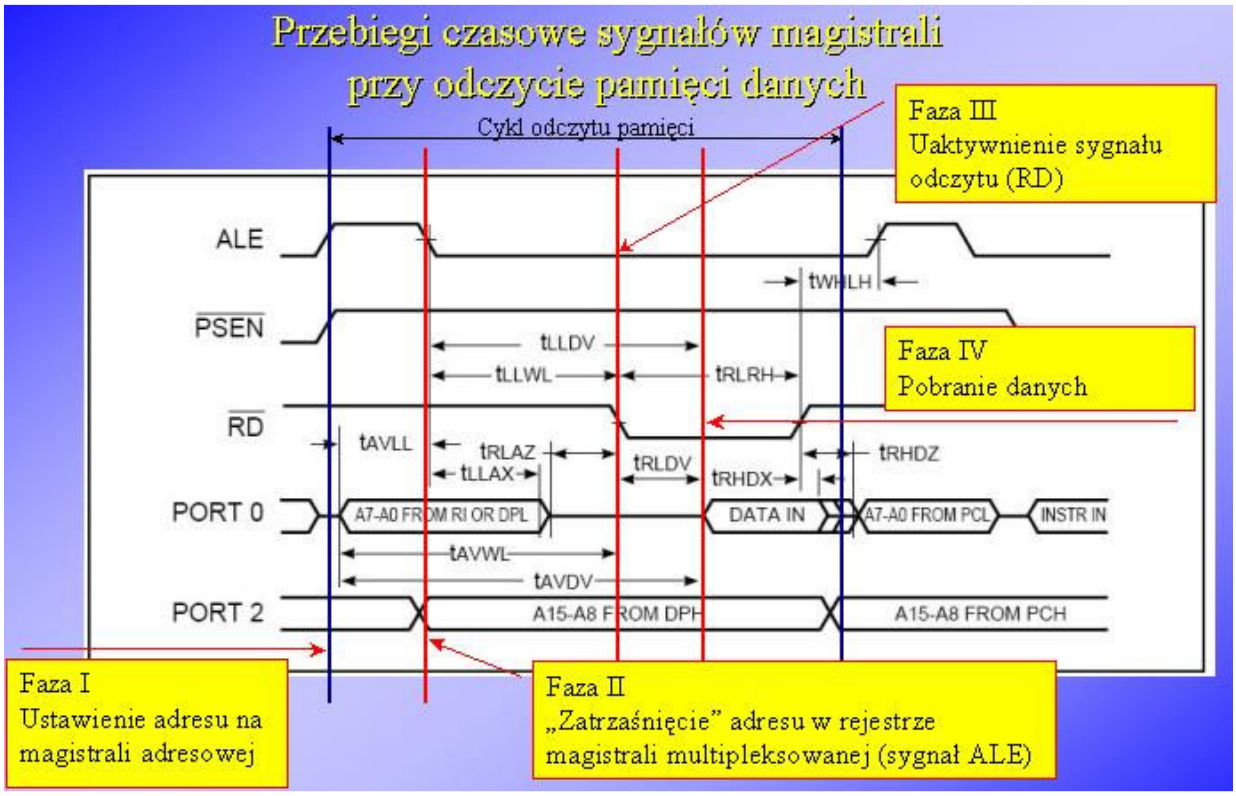


Praca magistrali multipleksowanej polega na następującej sekwencji zdarzeń:

1. w pierwszym kroku na magistralę podawany jest adres komórki pamięci zawierającej daną do odczytu lub do której dana będzie zapisana
2. cały adres lub jego część zapamiętywany jest w buforze adresowym i poprzez jego wyjścia wystawiany na wejścia adresowe układów w czasie trwania całej operacji zapisu lub odczytu danych
3. w kolejnym kroku przez magistralę przesyłane są dane

Przedstawia to rysunek przebiegów czasowych magistrali:

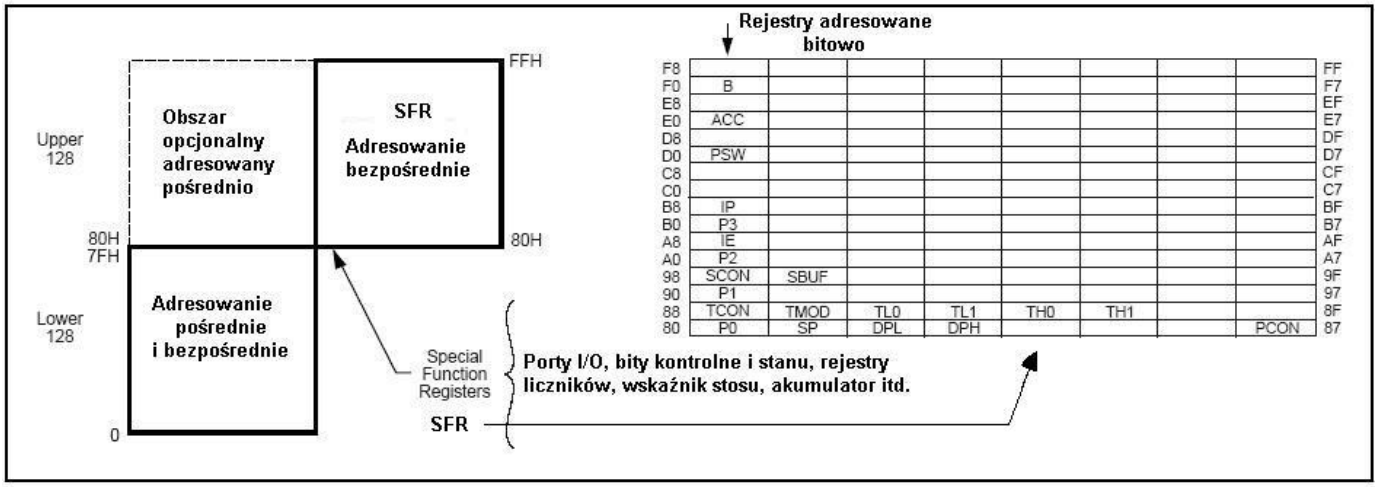




1.2 Struktura wewnętrznej pamięci RAM mikrokontrolera.

Wewnętrzna pamięć danych IRAM (Internal RAM) to w zasadzie najistotniejszy z punktu widzenia programisty element mikrokontrolera. Stanowi on miejsce przechowywania danych, wyników operacji i, co najistotniejsze dla mikrokontrolerów, jest wykorzystywany przez użytkownika do konfiguracji i sterowania pracą wewnętrznych układów mikrokontrolera.

Mikrokontrolery rodziny ’51 w podstawowej wersji wyposażone są w 256-bajtową pamięć IRAM, której podział przedstawia rysunek.

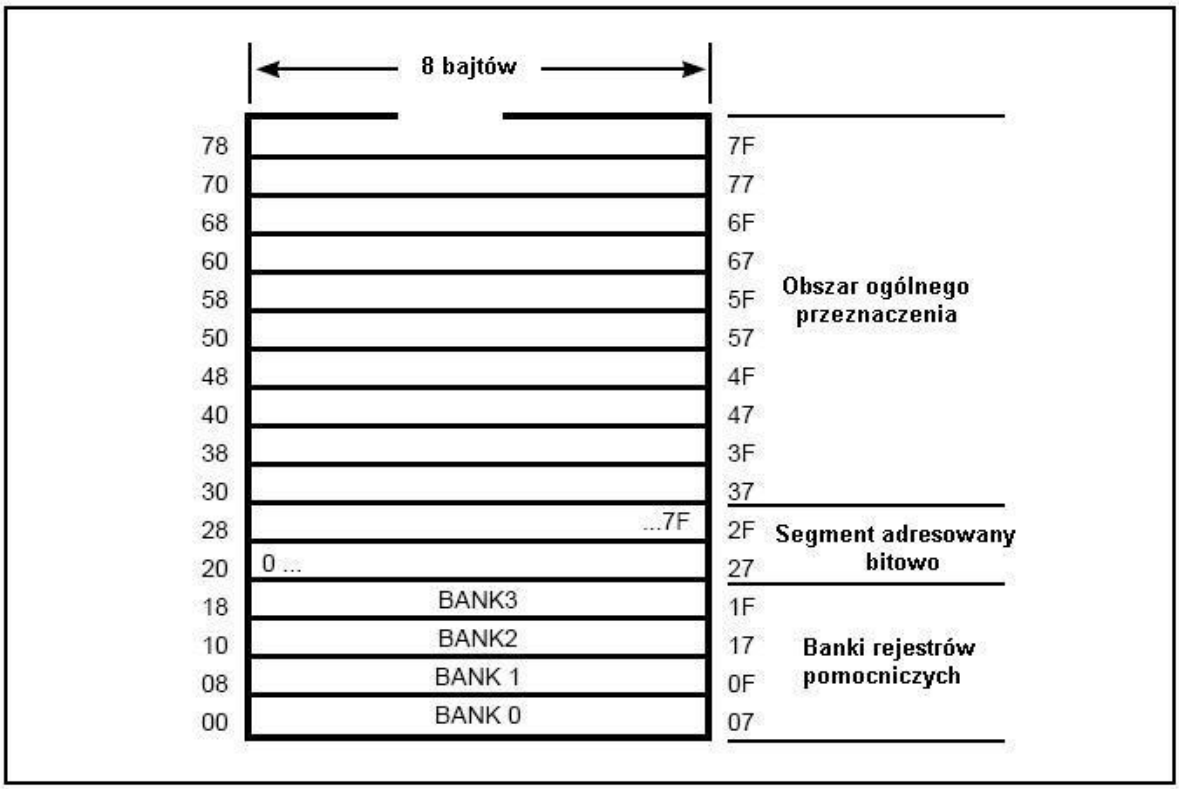


Jak widać, pamięć ta podzielona jest na 2 zasadnicze części różniące się przeznaczeniem oraz sposobem dostępu poprzez tryb adresowania:

- Upper memory – komórki pamięci o adresach od 127 do 255 (80H-0FFH) o adresowaniu bezpośrednim tworzące obszar rejestrów specjalnych (SFR –Special

Function Registers) zawierający podstawowe rejestry procesora oraz rejestry konfiguracji i sterowania wewnętrznych układów mikrokontrolera

- Lower memory – komórki pamięci o adresach od 0 do 127 (00H-7FH) o adresowaniu bezpośrednim oraz pośrednim będące „typową" pamięcią danych (jeśli pominąć charakterystyczne obszary banków rejestrów czy obszar adresowany bitowo); istotną cechą tego obszaru jest umiejscowienie w nim stosu czyli podręcznej pamięci wykorzystywanej przez procesor w procedurze wywołań podprogramów i obsługi przerwań
  W niektórych typach mikrokontrolerów występuję dodatkowy obszar pamięci danych pokrywający się z obszarem SFR (pamięć nakładkowa). Dostęp do tej pamięci możliwy jest jedynie poprzez adresowanie pośrednie.
  Strukturę obszaru „Lower memory" przedstawia rysunek.



Wszystkie komórki w tym obszarze są dostępne poprzez adresowanie bajtowe.

1.2.1 Banki rejestrów pomocniczych.

Komórki pamięci RAM o adresach 0-31 (00H-1FH) mogą być wykorzystane jako banki rejestrów pomocniczych. Każdy bank zawiera 8 rejestrów zatem banki są 4 (RB0, RB1, RB2 i RB3). W określonym momencie wykorzystywać można tylko jeden z nich, a o wyborze banku można decydować poprzez ustawienie odpowiednich bitów (RS0 i RS1) w rejestrze specjalnym PSW.

W przypadku rejestrów wchodzących w skład aktualnie wybranego banku możliwe jest zastosowanie składni instrukcji z użyciem zdefiniowanych nazw symbolicznych rejestrów zamiast ich adresów rzeczywistych. Są to odpowiednio nazwy R0, R1, R2,....R7 np., nazwa R0 odpowiada komórce o adresie 0 w przypadku wybrania banku RB0, komórce o adresie 8 w przypadku banku RB1, komórce o adresie 16 w przypadku banku RB2 lub komórce o adresie 24 w przypadku banku RB3 (oczywiście w danym momencie tylko jednej z tych komórek).

Wśród rejestrów pomocniczych dwa pełnia szczególną rolę – rejestry oznaczone jako R0 i R1 służą do adresowania pośredniego wewnętrznej i zewnętrznej pamięci RAM (dane zapisane w tych rejestrach odpowiadają adresom wykorzystywanych komórek)

Obszar banków wygląda następująco:

ADRES		
31 (1FH)	**Rejestr R7**	**Bank rejestrów RB3**
30 (1EH)	**Rejestr R6**	
29 (1DH)	**Rejestr R5**	
28 (1CH)	**Rejestr R4**	
27 (1BH)	**Rejestr R3**	
26 (1AH)	**Rejestr R2**	
25 (19H)	**Rejestr R1**	
24 (18H)	**Rejestr R0**	
23 (17H)	**Rejestr R7**	**Bank rejestrów RB2**
22 (16H)	**Rejestr R6**	
21 (15H)	**Rejestr R5**	
20 (14H)	**Rejestr R4**	
19 (13H)	**Rejestr R3**	
18 (12H)	**Rejestr R2**	
17 (11H)	**Rejestr R1**	
16 (10H)	**Rejestr R0**	
15 (0FH)	**Rejestr R7**	**Bank rejestrów RB1**
14 (0EH)	**Rejestr R6**	
13 (0DH)	**Rejestr R5**	
12 (0CH)	**Rejestr R4**	
11 (0BH)	**Rejestr R3**	
10 (0AH)	**Rejestr R2**	
9 (09H)	**Rejestr R1**	
8 (08H)	**Rejestr R0**	
7 (07H)	**Rejestr R7**	**Bank rejestrów RB0**
6 (06H)	**Rejestr R6**	
5 (05H)	**Rejestr R5**	
4 (04H)	**Rejestr R4**	
3 (03H)	**Rejestr R3**	
2 (02H)	**Rejestr R2**	
1 (01H)	**Rejestr R1**	
0 (00H)	**Rejestr R0**	

1.2.2 Obszar adresowany bitowo.

Obszar pamięci o adresach od 32 do 47 (20H-2FH) jest dostępny poprzez adresowanie bitowe. Tworzy go 16 rejestrów, które dostępne są jako bajty, dodatkowo można także operować każdym z bitów tych rejestrów z osobna. Daje to możliwości wykorzystywania tego obszaru jako np., zbioru flag, znaczników dostępnych przez instrukcji manipulacji bitowych lub co jest szczególnie ważne przez instrukcje skoków warunkowych.

Adres poszczególnych bitów może być tworzony w dwojaki sposób:

- Jako adres wewnętrzny bitu w rejestrze np. 22H.3 (adres bitu o numerze 3 w rejestrze o adresie 22H)
- Jako adres bezpośredni bitu – obszar ten składa się z 16*8=128 bitów i każdemu z kolejnych bitów w tym obszarze odpowiada numer od 0 do 127 (odpowiednio pierwszy bit w pierwszym rejestrze tego obszaru czyli bit 0 w rejestrze o adresie 32 (20H) może mieć adres 20H.0 a także 0, a np. bit o adresie 22H.3 ma adres bezpośredni 19 (bo 2*8+3=19 ; 2 =22H-20H[pierwszy rejestr w tym obszarze, 8 – bo rejestry zawierają 8 bitów, 3 – bo .... chyba nie trzeba tłumaczyć)

Mapa tego obszaru wygląda następująco:

Adres rejestru	**Numer bitu w rejestrze**								
	7	6	5	4	3	2	1	0	
47 (2FH)	**127**	**126**	**125**	**124**	**123**	**122**	**121**	**120**	**Adres bezpośredni bitu (dziesiętnie)**
46 (2EH)	**119**	**118**	**117**	**116**	**115**	**114**	**113**	**112**	
45 (2DH)	**111**	**110**	**109**	**108**	**107**	**106**	**105**	**104**	
44 (2CH)	**103**	**102**	**101**	**100**	**99**	**98**	**97**	**96**	
43 (2BH)	**95**	**94**	**93**	**92**	**91**	**90**	**89**	**88**	
42 (2AH)	**87**	**86**	**85**	**84**	**83**	**82**	**81**	**80**	
41 (29H)	**79**	**78**	**77**	**76**	**75**	**74**	**73**	**72**	
40 (28H)	**71**	**70**	**69**	**68**	**67**	**66**	**65**	**64**	
39 (27H)	**63**	**62**	**61**	**60**	**59**	**58**	**57**	**56**	
38 (26H)	**55**	**54**	**53**	**52**	**51**	**50**	**49**	**48**	
37 (25H)	**47**	**46**	**45**	**4**	**43**	**42**	**41**	**40**	
36 (24H)	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**	
35 (23H)	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	
34 (22H)	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	
33 (21H)	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	
32 (20H)	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	

1.2.3 Obszar rejestrów specjalnych SFR.

Obszar ten jest szczególnie istotny jeżeli chodzi o zarządzanie praca wewnętrznych układów wchodzących w skład mikrokontrolera. Zawarte w nim rejestry pamięci służą do sterowania ich praca poprzez ich konfigurację i aktywację.

Obszar ten zawiera komórki pamięci o adresach od 128 do 255 (80H-0FFH) i może być adresowany:

- Bezpośrednio jako rejestry 8-bitowe
- Bitowo (jak w poprzednim punkcie) dla rejestrów, których adres jest bez reszty podzielny przez 8 np.160; w przypadku adresów w formacie heksadecymalnym są to wszystkie komórki o najmłodszej pozycji równej 0 lub 8 np. 0A8H, 90H
  Dla rejestrów SFR adresowanych bitowo istnieją predefiniowane nazwy symboliczne bitów odpowiadające adresom bitowym, którymi można posługiwać się w składni instrukcji np. bit przeniesienia ustawiany przy operacjach arytmetycznych będący najstarszym bitem rejestru stanu procesora PSW (Program Status Word) może być adresowany przy użyciu adresu fizycznego 0D0H.7 (0D0H jest adresem bajtowym rejestru PSW) jak i poprzez adres bitowy 0D7H (215) powstały w wyniku dodania numeru bitu do adresu rejestru (patrz adresowanie bezpośrednie bitów w obszarze adresowanym bitowo. Można posłużyć się w tym przypadku także nazwą CY,C (CARRY) ,w miejsce której asembler wstawi adres fizyczny.
  Przykład.

Instrukcja zerowania bitu przeniesienia może być zapisana w następujący sposób

CLR 0D0H.7
CLR 0D7H
CLR C

Uwaga! Obszar SFR nie może być traktowany jako obszar do zapisu dowolnych danych. Wykonując operacje zapisu danych do pamięci IRAM należy zwrócić uwagę na prawidłowe adresowanie komórek pamięci. Przypadkowy, nieświadomy zapis danych do rejestrów SFR może spowodować niekontrolowane zachowanie mikrokontrolera np. poprzez przypadkowe uruchomienie systemu przerwań, układu licznika czy transmisji szeregowej.

Wszystkie rejestry SFR mają zdefiniowane nazwy symboliczne adresów, których skróty opisują funkcje spełniane przez te rejestry.

Ilość rejestrów SFR oraz ich funkcje zależne są od typu mikrokontrolera (dla wersji układu o rozbudowanej strukturze ich liczba ulega zwiększeniu z oczywistych powodów). Jednakże istnieje grupa rejestrów SFR, która jest charakterystyczna dla całej rodziny MCS'51. Ilustruje to poniższy rysunek obszaru SFR:

Rejestry adresowane bitowo

F8								FF
F0	B							F7
E8								EF
E0	ACC							E7
D8								DF
D0	PSW							D7
C8								CF
C0								C7
B8	IP							BF
B0	P3							B7
A8	IE							AF
A0	P2							A7
98	SCON	SBUF						9F
90	P1							97
88	TCON	TMOD	TL0	TL1	TH0	TH1		8F
80	P0	SP	DPL	DPH			PCON	87

REJESTRY SPECJALNE SFR

Symbol	Nazwa rejestru	Adres bezpośredni (HEX)	Adres bitu (HEX), adres symboliczny lub alternatywna funkcja portu								Wartość po restarcie
ACC	Akumulator (accumulator)	**E0H**	**E7**	**E6**	**E5**	**E4**	**E3**	**E2**	**E1**	**E0**	**00H**
B	Rejestr B	**F0H**	**F7**	**F6**	**F5**	**F4**	**F3**	**F2**	**F1**	**F0**	**00H**
DPH	Starszy bajt wskaźnikowego rejestru danych (Data Pointer Register [High])	**83H**									**00H**
DPL	młodszy bajt wskaźnikowego rejestru danych (Data Pointer Register [Low])	**82H**									**00H**
IE	Rejestr maski przerwań (Interrupt Enable)	**A8H**	**AF** **EA**	**AE** **-**	**AD** **-**	**AC** **ES**	**AB** **ET1**	**AA** **EX1**	**A9** **ET0**	**A8** **EX0**	**0XX00000B**
IP	Rejestr priorytetów przerwań (Interrupt Priority)	**B8H**	**BF** **-**	**BE** **-**	**BD** **-**	**BC** **PS**	**BB** **PT1**	**BA** **PX1**	**B9** **PT0**	**B8** **PX0**	**XXX00000B**
P0	Port 0	**80H**	**87** **P0.7** **AD7**	**86** **P0.6** **AD6**	**85** **P0.5** **AD5**	**84** **P0.4** **AD4**	**83** **P0.3** **AD3**	**82** **P0.2** **AD2**	**81** **P0.1** **AD1**	**80** **P0.0** **AD0**	**FFH**
P1	Port 1	**90H**	**97** **P1.7**	**96** **P1.6**	**95** **P1.5**	**94** **P1.4**	**93** **P1.3**	**92** **P1.2**	**91** **P1.1**	**90** **P1.0**	**FFH**
P2	Port 2	**A0H**	**A7** **P2.7** **A15**	**A6** **P2.6** **A14**	**A5** **P2.5** **A13**	**A4** **P2.4** **A12**	**A3** **P2.3** **A11**	**A2** **P2.2** **A10**	**A1** **P2.1** **A9**	**A0** **P2.0** **A8**	**FFH**
P3	Port 3	**B0H**	**B7** **P3.7** **/RD**	**B6** **P3.6** **/WR**	**B5** **P3.5** **T1**	**B4** **P3.4** **T0**	**B3** **P3.3** **/INT1**	**B2** **P3.2** **/INT0**	**B1** **P3.1** **TxD**	**B0** **P3.0** **RxD**	**FFH**
PCON	Zarządzanie poborem energii (Power CONtrol)	**87H**	**SMOD**	**-**	**-**	**-**	**GF1**	**GF0**	**PD**	**IDL**	**0XXX0000B**
PSW	Rejestr stanu programu (Program Status Word)	**D0H**	**D7** **CY**	**D6** **AC**	**D5** **F0**	**D4** **RS1**	**D3** **RS0**	**D2** **OV**	**D1** **-**	**D0** **P**	**00H**
SBUF	Bufor portu szeregowego (Serial data BUFfer)	**99H**									**XXXXXXXXB**
SCON	Rejestr konfiguracji portu szeregowego (Serial CONtroller)	**98H**	**9F** **SM0**	**9E** **SM1**	**9D** **SM0**	**9C** **REN**	**9B** **TB8**	**9A** **RB8**	**99** **TI**	**98** **RI**	**00H**
SP	Wskaźnik stosu (Stack Pointer)	**81H**									**07H**
TCON	Rejestr kontroli pracy liczników (Timer CONtrol)	**88H**	**8F** **TF1**	**8E** **TR1**	**8D** **TF0**	**8C** **TR0**	**8B** **IE1**	**8A** **IT1**	**89** **IE0**	**88** **IT0**	**00H**
TMOD	Rejestr trybu pracy liczników (Timer MODe)	**89H**	**GATE**	**C/T**	**M1**	**M0**	**GATE**	**C/T**	**M1**	**M0**	**00H**
TH0	Starszy bajt wyniku licznika T0 (Timer High 0)	**8CH**									**00H**
TH1	Starszy bajt wyniku licznika T1 (Timer High 1)	**8DH**									**00H**
TL0	Młodszy bajt wyniku licznika T0 (Timer Low 0)	**8AH**									**00H**
TL1	Młodszy bajt wyniku licznika T0 (Timer Low 0)	**8BH**									**00H**

Funkcje rejestrów związanych z konkretnymi układami wewnętrznymi np. liczników, opisane zostaną w poświęconych tym układom rozdziałach. W tym miejscu warto wspomnieć o tych rejestrach, które stanowią elementy pamięci procesora mikrokontrolera (czyli jego „jądra").

Do podstawowych rejestrów procesora zaliczamy:

1. **Akumulator (A, ACC)** – rejestr SFR o adresie 0E0H, dostępny poprzez adresowanie bitowe (adres bez reszty podzielny przez 8, nazwa symboliczna przy operacjach bitowych ACC); tak jak w wielu procesorach podstawowy rejestr związany z operacjami przetwarzania danych:
   W przypadku mikrokontrolera 8051 akumulatora dotyczą następujące własności:
   - Jest jedynym rejestrem procesora umożliwiającym komunikację procesora z układami połączonymi przez zewnętrzną magistralę systemowa z mikrokontrolerem; pełni rolę bufora danych wysyłanych do układów zewnętrznych magistrali systemowej np. wyświetlacza (dana, którą chcemy przesłać do układu zewnętrznego należy najpierw zapisać w akumulatorze) oraz danych odczytywanych z układów zewnętrznych (są one dostępne jedynie w akumulatorze po wykonaniu operacji odczytu)
   - Musi być wykorzystany przy wykonywaniu wszystkich operacji arytmetycznych na danych (przed wykonaniem operacji zawiera jeden z argumentów operacji a po jej wykonaniu wynik lub jego część)
2. **Rejestr stanu procesora PSW (Program Status Word – słowo stanu realizacji programu)** – podstawowa funkcja tego rejestru odpowiada rejestrowi flag (znaczników) procesora; rejestr o adresie 0D0H dostępny dla operacji bitowych tak jak akumulator

Numer bitu

	PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
0D0H	CY	AC	F0	RS1	RS0	OV	-	P
	0D7	0D6	0D5	0D4	0D3	0D2	0D1	0D0

Adres bitu

a) **CY (CARRY) – (PSW.7 lub 0D7H)** bit przeniesienia dla operacji arytmetycznych; w zależności od sposobu interpretacji danej przez programistę może spełniać następującą rolę:
   - Znacznik przekroczenia zakresu wartości 0-255 dla liczb bez znaku (8-bitowe dane wykorzystywane w programie mogą być interpretowane jako liczby naturalne czyli całkowite dodatnie); w przypadku wykonywania operacji arytmetycznego dodawania liczb 8-bitowych bez znaku można uzyskać liczbę 9-bitową (wynik z zakresu 256-511), zawarty wówczas w akumulatorze wynik stanowi jedynie uzupełnienie całkowitego wyniku do 256 (bit CY przyjmuje w tym przypadku wartość 1 i może być równie dobrze interpretowany jako 9 bit wyniku)
     Przykład. Wykonujemy operację dodawania ADD A,R5. Zawartość rejestrów przed wykonaniem operacji jest następująca:
     A=127=0111 1111B
     R5=160=1010 0000B
     W wyniku operacji dodawania powinniśmy uzyskać wynik 287=1 0001 1111B czyli wynik 9-bitowy; wynik zawarty w akumulatorze w tym przypadku to 31=0001 1111B zatem część całkowitego wyniku uzupełniająca go do 256, bit CY zaś przyjmie wartość 1 czyli taką jak 9 bit wyniku.
     W przypadku operacji odejmowania bit ten poprzez wartość 1 sygnalizuje wynik ujemny w przypadku kiedy odjemna jest mniejsza od odjemnika; uzyskany w tym przypadku wynik jest prawidłowy jako interpretowany w kodzie U2(uzupełnienia do 2)

Przykład. Wykonanie operacji odejmowania SUBB A,R5 dla tych samych danych powinno dać wynik 127-160=-33
Zawartość akumulatora po wykonaniu operacji to 1101 1111B. Jeżeli stan 1 bitu CY świadczący o tym, że wynik jest mniejszy niż 0 potraktujemy jako 9 bit wyniku (bit znaku) to otrzymamy wówczas wynik 1 1101 1111B co w kodzie U2 odpowiada liczbie –33.

- W przypadku wykonania operacji porównania dwóch danych 8-bitowch dla operacji skoku warunkowego CJNE stan tego bitu stanowi informacje o tym, która z porównywanych danych jest większa jako liczba bez znaku

b) AC (AUXILLIARY CARRY) – (PSW.6 lub 0D6H) flaga przeniesienia połówkowego; sygnalizuje przeniesienie występujące pomiędzy bitami b3 i b4 danych podczas operacji dodawania i odejmowania. W praktyce ma znaczenie jedynie dla operacji traktowanych jako operacje dla danych w kodzie BCD przy czym z flagi tej „korzysta" sam procesor dokonując na jej podstawie korekcji dziesiętnej wyniku dodawania lub odejmowania liczb w kodzie BCD

c) **F0 – (PSW.5 lub 0D5H)** flaga ogólnego przeznaczenia; nie jest zmieniana w wyniku operacji arytmetycznych; może posłużyć programiście jako flaga zmieniana programowo do sygnalizacji zdarzeń podczas działania programu

d) **RS1,RS0 (Register bank Select)– (PSW.4, PSW.3 lub 0D4H, 0D3H)**; para bitów służąca do wyboru banku rejestrów pomocniczych

RS1	RS0	
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

e) **OV (Overflow) – (PSW.2 lub 0D2H)** flaga przepełnienia, nadmiaru; w zależności od wcześniej wykonanej operacji ustawienie tej flagi (OV=1):
   - Sygnalizuje przekroczenie zakresu 8-bitowych liczb całkowitych ze znakiem w kodzie U2 (-128÷+127) dla operacji dodawania i odejmowania
   - Sygnalizuje uzyskanie dwubajtowego wyniku iloczynu (>255)
   - Sygnalizuje wykonanie operacji dzielenia przez 0

f) **P (Parity) – (PSW.0 lub 0D0H)** flaga parzystości; wartość tego bitu stanowi uzupełnienie liczby stanów „1" danej zawartej w akumulatorze do liczby parzystej; w praktyce może być wykorzystany np. w realizacji kontroli poprawności transmisji szeregowej danych wykorzystującej kontrolę parzystości lub nieparzystości.
Przykład. Jeżeli do akumulatora zapiszemy daną 97H=1001 0111B (5 stanów „1" czyli liczba nieparzysta stanów wysokich) wówczas bit parzystości P=1 (razem z „jedynkami" danej tworzy parzystą [6] liczbę stanów wysokich).

3. **Wskaźnik stosu SP (Stack Pointer)** – rejestr adresowy wykorzystywany przez procesor przy realizacji programu w przypadku wystąpienia procedur obsługi przerwań lub skoków do podprogramów (rejestr ten wskazuje adres komórki pamięci, w której zapisany jest adres powrotu z podprogramu wywołanego instrukcją CALL lub podprogramu obsługi przerwania).

Stos jest obszarem pamięci podręcznej (segment pamięci RAM) wykorzystywanej przez procesor lub program do przechowywania szczególnie wrażliwych danych, których

utrata najczęściej uniemożliwia prawidłową kontynuację programu. Dane te to zawartość podstawowych rejestrów procesora takich jak akumulator czy rejestr stanu procesora. Są one wykorzystywane przez różne programy (podprogramy) realizowane przez system, dlatego konieczna jest ochrona ich zawartości.

Stos jest automatycznie wykorzystywany przez procesor przy realizacji podprogramów (wywołanych w programie głównym lub w procedurze obsługi przerwania) do zapamiętania zawartości licznika rozkazów PC (Program Counter), który wskazuje adres pamięci zawierającej kod następnej instrukcji do wykonania. Zapis na stosie zawartości innych podstawowych rejestrów procesora musi być realizowany przez programistę w podprogramie lub przed jego wywołaniem (działanie takie świadczy o odpowiedzialności programisty i jest dobrym zwyczajem). Do realizacji operacji na stosie przeznaczona jest odrębna grupa instrukcji **PUSH i POP.**

Operowanie stosem jest o tyle wygodne, że nie wymaga określania w instrukcji adresu, pod który dana ma być zapisana. Adres ten stanowi aktualna zawartość wskaźnika stosu w momencie wykonania operacji zapisu na stos zwiększona o 1 (innymi słowy zawartość wskaźnika stosu w danym momencie jest adresem ostatnio zapisanej komórki stosu; po każdej operacji zapisu na stos układ sterowania zwiększa o 1 zawartość wskaźnika stosu). Wymaga to jednak stosowania **zasady LIFO (Last In First Out)** przy realizacji operacji na stosie. Zgodnie z tą zasadą ostatnia zapisana na stosie dana musi być odczytana w pierwszej kolejności i przesłana do miejsca skąd została odłożona. Inaczej: dane muszą być odczytywane ze stosu w odwrotnej kolejności niż były zapisywane na stos.

W przypadku mikrokontrolera 8051 istotna cechą wskaźnika stosu SP jest to, że w momencie restartu wskazuje on wartość 07 jako tzw. „dno stosu" tzn. że pierwszą komórką stosu jest komórka o adresie 08. Jest to o tyle niekorzystne, że odpowiada to komórkom banku rejestrów pomocniczych RB1 i w przypadku pozostawienia tej zawartości wskaźnika SP uniemożliwia korzystanie z tego banku a najczęściej także pozostałych banków za wyjątkiem RB0.

Korzystne z punktu widzenia korzystania z banków rejestrów jest umieszczenie stosu w obszarze pamięci nie kolidującym z bankami rejestrów, co można uczynić poprzez zapis odpowiedniej danej do wskaźnika stosu jako adresu początkowego stosu.
Np. instrukcja **MOV SP,#50** powoduje umieszczenie stosu w obszarze pamięci od adresu 50.

1.3 System przerwań mikrokontrolera.

1.3.1 Pojęcie przerwania.

Działanie systemu mikroprocesorowego polega na realizacji zasadniczego programu działania zwanego programem głównym oraz czasowej realizacji tzw. podprogramów czyli innych programów wywoływanych w trakcie działania programu głównego. Realizacja podprogramów może być związana podprogramów wywołaniem tych podprogramów za pomocą specjalnych instrukcji (CALL) i wówczas to programista określa moment ich uruchomienia. Zasadniczym problemem jest jednak zapewnienie odpowiedniej obsługi urządzeń zewnętrznych, szczególnie w przypadku kiedy ze względu na specyfikę pracy urządzenia nie da się przewidzieć kiedy będzie konieczna komunikacja z nim. Np. klawiatura jako urządzenie umożliwiające użytkownikowi wpływanie na pracę programu może być obsługiwana z różną prędkością przez różne osoby, dlatego konieczność jej obsługi może występować z różną częstotliwością i z różnym natężeniem.

Zasada obsługi tego typu urządzeń oparta jest na wykorzystaniu uniwersalnych podprogramów zwanych sterownikami. Możliwe jest cykliczne wywoływanie tej procedury i próba „trafienia" na moment, w którym urządzenie żądać będzie obsługi. Aby zwiększyć prawdopodobieństwo szybkiej reakcji na takie zdarzenie można procedurę obsługi wywoływać bardzo często, w krótkich odstępach czasu. W taki sposób realizuje się tzw. obsługę programową urządzenia.

Działanie to jest bardzo nieefektywne, gdyż większość prób komunikacji kończy się jałowym działaniem (urządzenie nie wymaga obsługi) i niepotrzebnie obciąża procesor uniemożliwiając w tym czasie realizację programu głównego.

Idealnym działaniem jest obsługa urządzenia tylko wtedy kiedy jest to konieczne. Sytuację taką zapewnia system przerwań systemu mikroprocesorowego.

Przerwanie jest chwilowym zatrzymaniem realizacji aktualnie wykonywanego programu, związane najczęściej z wystąpieniem istotnego dla pracy systemu zdarzenia, i wykonanie innego programu zwanego programem (podprogramem, procedurą) obsługi przerwania. Po realizacji obsługi przerwania konieczne jest odtworzenie stanu systemu przed wystąpieniem przerwania (patrz zastosowanie stosu) i dalsza realizacji programu od miejsca jego wstrzymania.

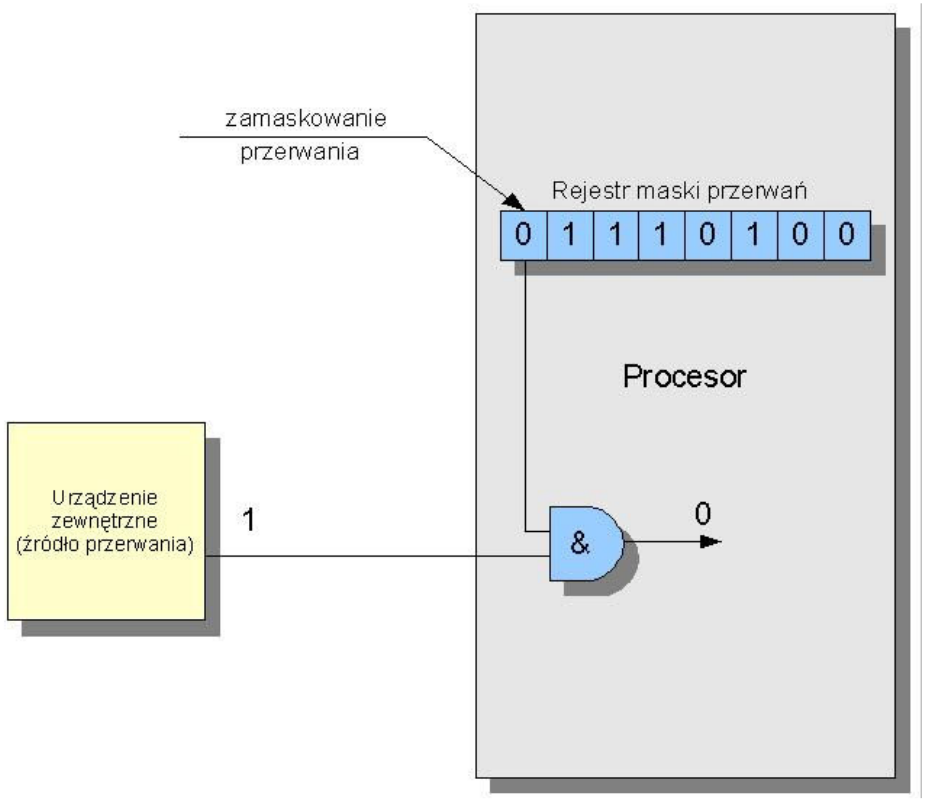
Przerwania można podzielić:

1. **ze względu na sposób wywołania przerwania:**
   - **programowe** – wywoływane przez programistę w sposób celowy przy użyciu odpowiednich instrukcji programowych
   - **sprzętowe** – istotne z punktu widzenia obsługi urządzeń zewnętrznych (szczególnie o charakterze wejściowym np. karta sieciowa, klawiatura itp.) wykorzystujące możliwość wpływania na pracę procesora przez urządzenie zewnętrzne za pomocą wytwarzanych przez to urządzenie odpowiednich sygnałów sterujących
2. **ze względu na sposób wyznaczenia adresu procedury obsługi przerwania**
   - proste – w których każde źródło przerwania ma jednoznacznie przypisany adres lokacji podprogramu obsługi (wektor); użytkownik nie ma wpływu na zmianę wektora obsługi
   - wektorowe – wykorzystują obszar pamięci RAM zwany tablicą wektorów przerwań do zapisania adresu podprogramu obsługi przerwania; w założeniu jest to system uniwersalny, umożliwiający przypisanie danemu urządzeniu dowolnego przerwania, dowolną lokalizację programu obsługi oraz swobodna zmianę tych parametrów (w praktyce swoboda ta jest ograniczona ze względu na trudność zapewnienia prawidłowej pracy systemu, szczególnie sygnałów przypadku rozbudowanych, skomplikowanych systemów)
3. **ze względu na konieczność obsługi przerwania**
   - niemaskowalne – zdarzenia mające zasadnicze znaczenie dla prawidłowej pracy systemu, które wymagają zawsze natychmiastowej reakcji systemu i realizacji obsługi np. „reset" procesora

- maskowalne – zdarzenia, które można „ukryć, zamaskować” przed procesorem, zatem można w dowolny sposób zmieniać reakcję systemu w danym momencie na wystąpienie określonego zdarzenia; maskowanie przerwań polega na zastosowaniu odpowiedniego rejestru zwanego rejestrem maski przerwań.

Ilustrację tego zagadnienia stanowi poniższy rysunek.

Zgłoszenie przerwania w tym przypadku oznacza ustawienie stanu wysokiego „1” na wejściu zgłaszania przerwań. Do maskowania przerwania wykorzystana została bramka NAND. Ustawienie stanu niskiego „0” bitu maski przerwania zgodnie z logiką działania bramki powoduje, że przerwanie nie zostanie zgłoszone.



Z systemem przerwań związane są pojęcia **priorytetów** oraz **poziomów** przerwania.

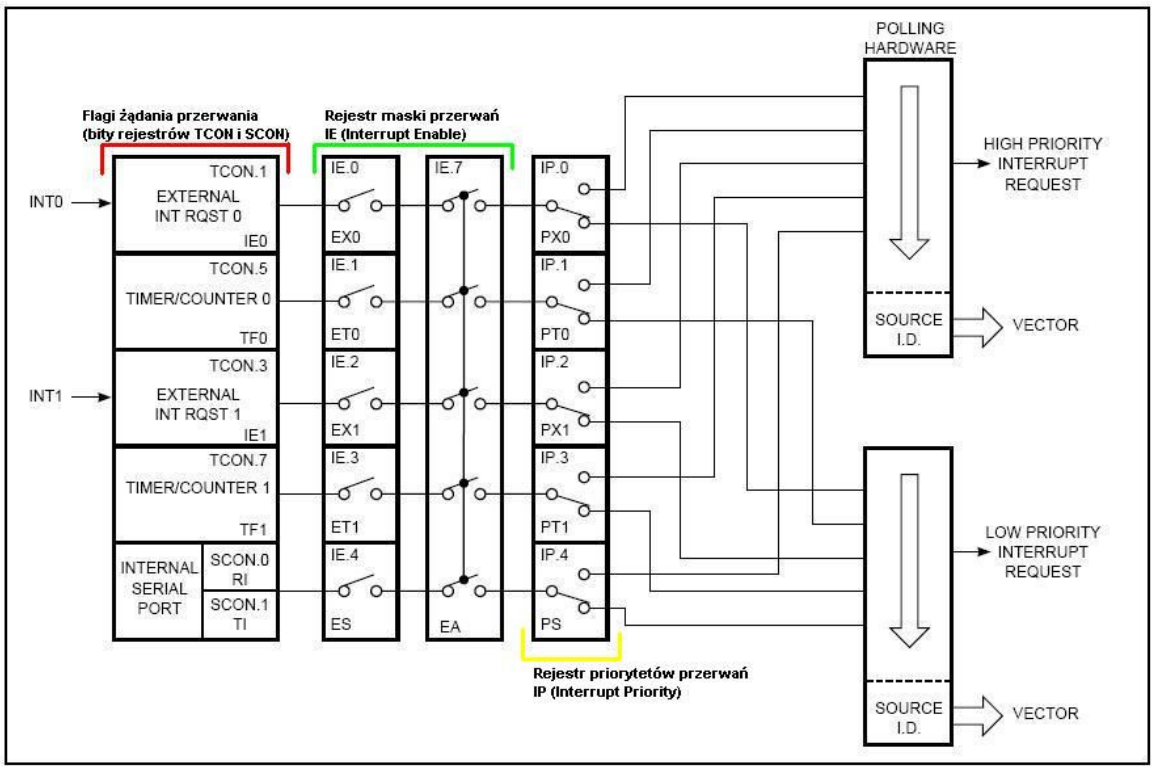
Priorytet przerwania jest określeniem ważności (wagi) danego przerwania w przypadku wystąpienia jednoczesnego co najmniej dwóch różnych przerwań. W danym momencie możliwa jest obsługa tylko jednego przerwania, dlatego konieczne jest tworzenie kolejki obsługi dla wielu przerwań występujących w tym samym czasie. Kolejność obsługi ustalana jest na podstawie priorytetów (im wyższy priorytet tym krótszy czas oczekiwania na obsługę).

Priorytety mogą być przydzielone na stałe lub mogą być okresowo zmieniane w celu usprawnienia działania systemu tzw. **rotacja priorytetów**, która pozwala uzyskać zbliżony średni czas oczekiwania na obsługę dla wszystkich przerwań.

Poziom przerwania związany jest z możliwością przerywania obsługi jednych przerwań na rzecz innych czyli tzw. **zagnieżdżona obsługa przerwań**.

1.3.2 Struktura systemu przerwań mikrokontrolera 8051, źródła oraz znaczniki przerwań .

Strukturę systemu przerwań związaną z rejestrami specjalnymi SFR przedstawia rysunek.



Jak wynika z rysunku przerwania w systemie mikrokontrolera 8051 mogą być generowane przez następujące źródła:

- 2 przerwania zewnętrzne **INT0** i **INT1** wykorzystywane przez urządzenia zewnętrzne z punktu widzenia mikrokontrolera
- 2 przerwania od układów liczników wewnętrznych **T0** i **T1**
- 2 przerwania od układu transmisji szeregowej (osobno dla nadajnika **TI** i odbiornika **RI**)

Każdemu przerwaniu przyporządkowana jest flaga zgłoszenia przerwania w postaci bitu odpowiedniego rejestru SFR(ustawienie stanu „1" dla bitu flagi przy odblokowanym systemie przerwań oznacza zgłoszenie przerwania). I tak:

- bity **IE0** oraz **IE1** rejestru **TCON** są znacznikami przerwań zewnętrznych INT0 oraz INT1
- bity **TF0** oraz **TF1** rejestru **TCON** są znacznikami przerwań od układów wewnętrznych liczników odpowiednio T0 oraz T1
  Rejestr TCON należy do rejestrów dostępnych przez adresowanie bitowe

Numer bitu

	TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0
88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
	8F	8E	8D	8C	8B	8A	89	88

Adres bitu (HEX)

- bity **TI** oraz **RI** rejestru **SCON** są znacznikami przerwań od nadajnika i odbiornika układu transmisji szeregowej UART
  Rejestr SCON jest adresowany bitowo

Numer bitu

	SCON.7	SCON.6	SCON.5	SCON.4	SCON.3	SCON.2	SCON.1	SCON.0
98H	SM0	SM1	SM2	REN	TB8	IRB8	TI	RI
	9F	9E	9D	9C	9B	9A	99	98

Adres bitu (HEX)

Innym zagadnieniem jest kasowanie znaczników przerwań po przyjęciu przerwania: Może to być realizowane:

- **sprzętowo** czyli automatycznie przez układ sterowania mikrokontrolera po uruchomieniu procedury obsługi przerwania; działanie to dotyczy znaczników liczników TF0 i TF1 oraz znaczników przerwań zewnętrznych IE0 i IE1 w przypadku kiedy przerwania te są zgłaszane przez pojawienie się opadającego zbocza sygnału zewnętrznego na wejściu zgłaszania przerwań.
- **programowo** czyli to programista w procedurze obsługi musi zapisać instrukcje kasowania znaczników przerwań; sytuacja taka ma miejsce dla znaczników układu transmisji szeregowej TI i RI oraz znaczników IE0 i IE1 przerwań zewnętrznych jeżeli są one zgłaszane przez ustalenie się stanu logicznego „0” sygnału zewnętrznego na wejściu zgłaszania przerwań.

UWAGA. Jeżeli znacznik przerwania kasowany programowo nie zostanie wyzerowany to po zakończeniu procedury obsługi przerwania zostanie ono ponownie zgłoszone. Dodatkowo w przypadku reakcji na poziom sygnału dla przerwań zewnętrznych INT0 oraz INT1 jeżeli po zakończeniu procedury obsługi przerwania na wejściu zgłaszania przerwań nadal utrzymuje się niski poziom logiczny sygnału, ponownie zostanie zgłoszone przerwanie zewnętrzne mimo programowego kasowania flag. W tym przypadku konieczne jest zatem usunięcie zewnętrznej przyczyny przerwania lub jego zablokowanie przed wyjściem z procedury jego obsługi.

Konfiguracja sposobu zgłaszania przerwań zewnętrznych odbywa się za pomocą bitów IT0 oraz IT1 rejestru TCON. Ustawienie ich w stan „0” oznacza reakcję na poziom sygnału, zaś stan „1” odpowiada reakcji na zbocze sygnału.

1.3.3 Aktywacja, blokowanie oraz priorytety przerwań mikrokontrolera 8051.

Konfiguracji systemu przerwań dokonuje się za pomocą rejestrów:

1. **IE (Interrupt Enable) –- rejestr maski przerwań** o adresie 0A8H zatem dostępny także poprzez instrukcje manipulacji bitowych; ustawienie bitu (bit=1) oznacza aktywację funkcji natomiast zerowanie bitu oznacza wyłączenie funkcji

Numer bitu

	IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0
0A8H	EA	-	-	ES	ET1	EX1	ET0	EX0
	0AF	0AE	0AD	0AC	0AB	0AA	0A9	0A8

Adres bitu (HEX)

a) **EA (Enable All interrupts) – (IE.7 lub 0AFH)** bit zezwolenia na realizację działania systemu przerwań; pozwala w dowolnym momencie na załączenie (np. instrukcją **SETB EA**) lub wyłączenie (np. instrukcją **CLR EA**) całego systemu przerwań; bit o znaczeniu nadrzędnym w stosunku do pozostałych bitów tego rejestru

b) **ES (Enable Serial port interrupt) – (IE.4 lub 0ACH)** pozwala na indywidualne sterowanie przerwaniem dla portu szeregowego; załącza (ES=1 i EA=1) lub wyłącza (ES=0) przerwanie od portu szeregowego

c) **ET1 (Enable Timer 1 interrupt) – (IE.3 lub 0ABH)** pozwala na indywidualne sterowanie przerwaniem dla licznika T1; załącza (ET1=1 i EA=1) lub wyłącza (ET1=0) przerwanie od licznika T1

d) **EX1 (Enable eXternal 1 interrupt) – (IE.2 lub 0AAH)** pozwala na indywidualne sterowanie przerwaniem zewnętrznym o numerze 1 (INT1); załącza (EX1=1 i EA=1) lub wyłącza (EX1=0) przerwanie zewnętrzne o numerze 1 (INT1)
e) **ET0 (Enable Timer 0 interrupt) – (IE.1 lub 0A9H)** tak jak ET1 tyle że dla licznika T0
f) **EX0 (Enable eXternal 0 interrupt) – (IE.0 lub 0A8H)** tak jak EX1 tyle że dla przerwania zewnętrznego INT0

Przykład. Zapis do rejestru **IE** danej **10011000B** np**. MOV IE,#98B** lub sekwencja
SETB EA
SETB ES
SETB ET1
powoduje załączenie systemu przerwań oraz uaktywnienie przerwań od portu szeregowego oraz timera T1.

2. **IP (Interrupt Priority level) – rejestr poziomów priorytetów przerwań** o adresie 0B8H dostępny przez adresowanie bitowe.

Kolejność obsługi jednocześnie zgłaszanych przerwań zależna jest od ich wagi określonej przez priorytet. Producent układu mikrokontrolera określa następującą standardową hierarchię przerwań (od najwyższego do najniższego priorytetu):

- Przerwanie zewnętrzne INT0
- Przerwanie od licznika T0
- Przerwanie zewnętrzne INT1
- Przerwanie od licznika T1
- Przerwanie od układu transmisji szeregowej

Oznacza to, że jeżeli nie dokonamy programowo zmiany priorytetów, pozostawiając ustawienie producenta musimy liczyć się np. z sytuacją pierwszeństwa obsługi przerwania od licznika T0 przed obsługą jednocześnie zgłoszonego przerwania zewnętrznego INT1.

Dodatkowo jeżeli w trakcie obsługi przerwania INT1 nastąpi zgłoszenie przerwania o wyższym priorytecie np. od licznika T0, nastąpi zablokowanie obsługi przerwania INT1 i jego dokończenie po wcześniejszym zrealizowaniu obsługi przerwania T0.

Zmiana poziomów przerwań przez konfigurację rejestru IP pozwala np. nadać najwyższy w danym momencie priorytet np. przerwaniu od układu transmisji szeregowej. Ustawienie stanu „1" bitu dla określonego źródła przerwania pozwala zwiększyć wagę przerwania zwiększając jego poziom.

Obowiązuje jednak nadal standardowa kolejność obsługi jeżeli poziom przerwania zostanie zwiększony dla więcej niż jednego źródła.

Numer bitu

	IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
0B8H	-	-	-	PS	PT1	PX1	PT0	PX0
	0BF	0BE	0BD	0BC	0BB	0BA	0B9	0B8

Adres bitu (HEX)

a) **PS (Priority level of Serial port interrupt) – (IP.4 lub 0BCH)** zmiana poziomu przerwania portu szeregowego
b) **PT1 (Priority level of Timer 1 interrupt) – (IP.3 lub 0BBH)** zmiana poziomu przerwania licznika T1
c) **PX1 (Priority level of eXternal 1 interrupt) – (IP.2 lub 0BAH)** zmiana poziomu przerwania zewnętrznego INT1
d) **PT0 (Priority level of Timer 1 interrupt) – (IP.1 lub 0B9H)** zmiana poziomu przerwania licznika T0

e) **PX0 (Priority level of eXternal 0 interrupt) – (IP.0 lub 0B8H)** zmiana poziomu przerwania zewnętrznego INT0

Przykład. Zapis do rejestru IP danej **00011000B** spowoduje ustalenie następującej kolejności obsługi przerwań:

- Licznik T1
- Port szeregowy
- Przerwanie zewnętrzne INT0
- Licznik T0
- Przerwanie zewnętrzne INT1

1.3.4 Wektory przerwań systemu mikrokontrolera 8051.

Bardzo ważnym zagadnieniem związanym z obsługa przerwań jest właściwe umieszczenie programów ich obsługi (sterowników) w pamięci programu. Umożliwiają to tzw. wektory przerwań.

W systemie mikrokontrolera 8051 każde źródło przerwania jest jednoznacznie związane z adresem pamięci programu, który ładowany jest do licznika rozkazów PC w momencie wykrycia przerwania i zidentyfikowania jego źródła. Pod tym adresem musi zostać zapisany pierwsza instrukcja programu obsługi danego przerwania. Wektor przerwania jest właśnie takim adresem.

W systemach mikroprocesorowych część pamięci wykorzystywanej przez system pełni rolę tzw. tablicy wektorów przerwań. Komórki tego obszaru pamięci są przypisane do konkretnych przerwań i zawierają dane wprost będące adresami, pod którymi zapisano program obsługi danego przerwania bądź umożliwiają one wyliczenie tego adresu przez procesor.

Bardzo podobnie sytuacja wygląda w przypadku mikrokontrolera 8051. Wykorzystując w programie przerwania musimy pamiętać o tym, że fragmenty programu niezwiązane z przerwaniami nie mogą być lokowane w dowolnym miejscu pamięci. Początkowy obszar pamięci programu zarezerwowany jest właśnie do celów odpowiadających tablicy wektorów.

W przypadku korzystania z wszystkich możliwych przerwań a przede wszystkim wykorzystując przerwanie od układu transmisji szeregowej (największy wektor przerwania) właściwy program powinien być umieszczony co najmniej od adresu 26H. Spróbujmy to wyjaśnić.

W poniższej tabeli zamieszczono wektory wszystkich podstawowych przerwań mikrokontrolera 8051 oraz dodatkowe informacje omawiane we wcześniejszych fragmentach.

Źródło przerwania	Flaga żądania przerwania	Sposób obsługi (kasowania) flagi żądania przerwania	Wektor (adres) przerwania
Restart (reset) systemu (tak samo rozumie się także każde uruchomienie układu poprzez załączenie zasilania)	(RST)	-	0000H
Przerwanie zewnętrzne INT0	IE0	**Programowy** (by software) – aktywacja poziomem (level) sygnału na wejściu **Sprzętowy** (by hardware) – aktywacja zboczem sygnału na wejściu	0003H
Licznik T0	TF0	**Sprzętowy**	000BH
Przerwanie zewnętrzne INT1	IE1	**Programowy** (by software) – aktywacja poziomem (level) sygnału na wejściu **Sprzętowy** (by hardware) – aktywacja zboczem sygnału na wejściu	0013H
Licznik T0	TF0	**Sprzętowy**	000BH
Układ transmisji szeregowej UART	TI+RI	**Programowy**	0023H

Jak wynika z tabeli po załączeniu układu, co odpowiada restartowaniu systemu, zawsze do licznika rozkazów ładowany jest adres 0000H i dlatego każdy program dla systemu 8051 musi zaczynać się od tego adresu. W praktyce jest to pierwsza instrukcja skoku do właściwego programu głównego w obszarze pamięci pomijającym obszar zarezerwowany na obsługę przerwań (oczywiście jeśli nie wykorzystujemy systemu przerwań program może zajmować także obszar zarezerwowany na przerwania).

Widzimy także, że wektory kolejnych przerwań różnią się o 8 co oznacza, że dla każdego przerwania zarezerwowane jest 8 bajtów pamięci. W praktyce jest to niewystarczające do zapisania całej procedury obsługi przerwania i dlatego w tym miejscu umieszcza się odwołanie do właściwej procedury obsługi oraz ewentualnie instrukcję powrotu z obsługi przerwania.

Widzimy również, że w klasycznym układzie ostatni wektor przerwania dla układu UART wynosi 23H, co przy uwzględnieniu że instrukcja wywołania procedury przerwania zajmuje 3 bajty w pamięci uzasadnia wcześniejsze twierdzenie, że właściwy program główny można wówczas umieścić co najmniej od adresu 26H (przy odpowiednim zapisaniu procedury obsługi).

Właściwą lokację odpowiednich fragmentów programu umożliwia wykorzystanie odpowiednich dyrektyw Asemblera np. ORG.

Przykład. Oto sposób zapisania programu, który wykorzystuje system przerwań od urządzenia zewnętrznego dla wejścia INT0 oraz układu UART. Użyte w programie nazwy symboliczne np. GLOWNY, OBSLUGA_INT0, OBSLUGA_UART odpowiadają rzeczywistym adresom w pamięci programu, które są wyznaczane przez asembler podczas kompilacji programu.

Pierwsze instrukcje programu głównego konfigurują system przerwań poprzez zezwolenie na przerwania (SETB EA) oraz załączenie przerwań zewnętrznego INT0 (SETB EX0) oraz od układu UART (SETB ES)

```
        CSEG AT 0H ; umiejscowienie fragmentu od adresu 0H

        LJMP GLOWNY ; skok do programu głównego
        ORG 0003H     ; umieszczenie kolejnego fragmentu programu (obsługa przerwania INT0 od
        adresu
                      ; 0003H
        LCALL OBSLUGA_INT0 ; wywołanie podprogramu obsługi
        RETI     ; instrukcja powrotu z obsługi przerwania
        ORG 23H       ; od tego adresu umieszczamy obsługę przerwania UART
        LJMP OBSLUGA_UART       ; instrukcja skoku do obszaru gdzie
                                  ;znajduje się fragment programu
                                  ; dotyczący obsługi UART (w tym przypadku
                                  ; instrukcja powrotu z  obsługi przerwania RETI
                                  ; musi być umieszczona na końcu wywołanego
                                  ; fragmentu)
         ORG 60H      ; program główny umieszczony od adresu 60H zatem nie narusza obszaru
                      ;zarezerwowanego na przerwania
GLOWNY:
         SETB EA
         SETB EX0
         SETB ES
         .
         .
         .itd.
```

1.4 Układy liczników mikrokontrolera.

Liczniki (inaczej timery) jako układy wewnętrzne mikrokontrolera spełniają istotna rolę w działaniu systemu mikroprocesorowego i jako takie występują w każdym systemie. Jak sama nazwa wskazuje są to układy przeznaczone do zliczania impulsów, przy czym działanie to przede wszystkim wykorzystywane jest do realizacji zadań sterowania praca systemu względem czasu (tłumaczy to oryginalna nazwę tych układów).

Mikrokontrolery rodziny 8051 wyposażone są co najmniej w 2 układy liczników, T0 i T1.

Układy te mogą być obsługiwane poprzez wykorzystanie systemu przerwań i mogą być w dość dużym stopniu konfigurowane przez użytkownika poprzez rejestry specjalne SFR z nimi związane (rejestry związane z obsługą liczników poprzez system przerwań omówiono w poprzednim rozdziale)

1.4.1 Konfiguracja liczników mikrokontrolera - rejestry sterujące SFR.

Konfigurację pracy wewnętrznych układów liczników można przeprowadzić wykorzystując rejestry specjalne SFR:

1. **TCON (Timer/counter CONtrol register) – rejestr kontroli pracy układów czasowozliczajacych** o adresie 88H zatem dostępny także poprzez instrukcje manipulacji bitowych; 4 starsze bity tego rejestru wykorzystywane są do uruchamiania/zatrzymywania zliczania oraz jako flagi przepełnienia wykorzystywane także jako znaczniki przerwań od układów liczników

	Numer bitu							
	TCON.7	**TCON.6**	**TCON.5**	**TCON.4**	**TCON.3**	**TCON.2**	**TCON.1**	**TCON.0**
88H	**TF1**	**TR1**	**TF0**	**TR0**	**IE1**	**IT1**	**IE0**	**IT0**
	8F	**8E**	**8D**	**8C**	**8B**	**8A**	**89**	**88**
	Adres bitu (HEX)							

   a) **TF1 (Timer 1 overflow Flag) – (TCON.7 lub 8FH)** flaga przepełnienia licznika T1 oraz znacznik przerwania od układu licznika T1 w przypadku uaktywnienia systemu przerwań; liczniki mikrokontrolera pracują jako MOD N co oznacza, że po osiągnięciu wartości maksymalnej (zależnej od trybu pracy; maksymalnie 65535 dla licznika 16 bitowego) następuje zerowanie licznika i ponowne zliczanie od 0; przepełnienie licznika podczas zliczania sygnalizowane jest ustawieniem flagi TF1=1;
   UWAGA! W przypadku , kiedy do obsługi licznika nie jest wykorzystywany system przerwań (obsługa programowa czyli metoda pollingu) ponowne wykrycie przepełnienia licznika wymaga każdorazowego kasowania znacznika TF1 w programie obsługi)
   b) **TR1 (Timer 1 Run control bit) – (TCON.6 lub 8EH)** uruchamia (TR1=1) lub zatrzymuje (TR1=0) pracę licznika T1
   c) **TF0 (Timer 0 overflow Flag) – (TCON.5 lub 8DH)** flaga przepełnienia licznika T0 oraz znacznik przerwania od układu licznika T0 w przypadku uaktywnienia systemu przerwań; przepełnienie licznika podczas zliczania sygnalizowane jest ustawieniem flagi TF0=1;
   UWAGA! W przypadku , kiedy do obsługi licznika nie jest wykorzystywany system przerwań (obsługa programowa czyli metoda pollingu) ponowne wykrycie przepełnienia licznika wymaga każdorazowego kasowania znacznika TF0 w programie obsługi)
   d) **TR0 (Timer 0 Run control bit) – (TCON.4 lub 8CH)** uruchamia (TR0=1) lub zatrzymuje (TR0=0) pracę licznika T0
2. **TMOD (Timer/counter Mode control register) – rejestr konfiguracji trybu pracy liczników**; rejestr dostępny jedynie poprzez adresowanie bajtowe (niedostępny dla operacji

manipulacji bitowych); 4 starsze bity rejestru służą do konfiguracji trybu pracy licznika T1, 4 młodsze bity odpowiadają licznikowi T0

TMOD

89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0
	TIMER 1				TIMER 0			

a) **GATE** – bit sprzętowej (hardware) zewnętrznej kontroli pracy liczników; jeżeli GATE=1 oraz TRx(x=0,1)=1 wówczas odpowiedni licznik T0 lub T1 uruchamiany jest jeżeli na wyprowadzeniu INTx(x=0,1) mikrokontrolera występuje poziom logiczny wysoki „1”; jeżeli GATE=0 wówczas odpowiedni licznik T0 lub T1 uruchamiany jest jedynie poprzez bit TRx(x=0,1) rejestru TCON

b) **C/T (Counter or Timer)** – ustawienie bitu w stan „1” powoduje, że licznik pracuje jako COUNTER tzn. zlicza impulsy z zewnętrznego źródła doprowadzone poprzez port P3 (odpowiednio P3.5[T1] dla licznika T1 i P3.4[T0] dla licznika T0;
Stan „0” tego bitu oznacza, że licznik pracuje jako TIMER tzn. zlicza impulsy z układu wewnętrznego generatora zegarowego o częstotliwości f=fosc/12

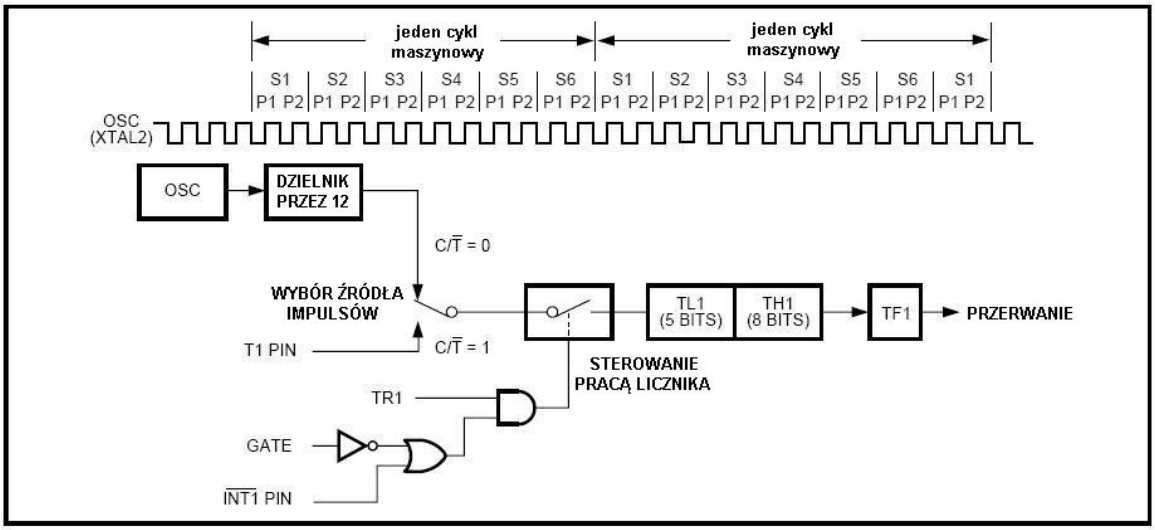
c) **M1, M0 (Mode selector bits)** – bity wyboru trybu pracy; kombinacja tych dwóch bitów pozwala na wybór 1 z 4 trybów pracy licznika

M1	M0	**Tryb pracy liczników**
0	0	Timer/counter 13-bitowy
0	1	Timer/counter 16-bitowy
1	0	Timer/counter 8-bitowy z autoładowaniem
1	1	Tylko licznik 0 jako dwa 8-bitowe liczniki (licznik 1 zatrzymany)

1.4.2 Charakterystyka pracy liczników dla poszczególnych trybów.

1.4.2.1 Tryb 0.

Na rysunku przedstawiono konfigurację licznika (w tym przypadku licznika T1) w trybie 0 wraz z odpowiednimi bitami rejestrów TMOD i TCON odpowiadającymi za konfigurację i sterowanie pracą licznika.



Oba liczniki w trybie 0 pracują jako 8-bitowe z prescalerem dzielącym przez 32 (2^5), którego funkcję spełnia rejestr młodszego bajtu bufora licznika TLx (na rysunku TL1).

Rejestr wynikowy licznika w tym przypadku ma długość 13 bitów, na które składają się wszystkie bity rejestru THx oraz 5 młodszych bitów rejestru TLx (3 najstarsze bity tego rejestru są w tym przypadku pomijane).

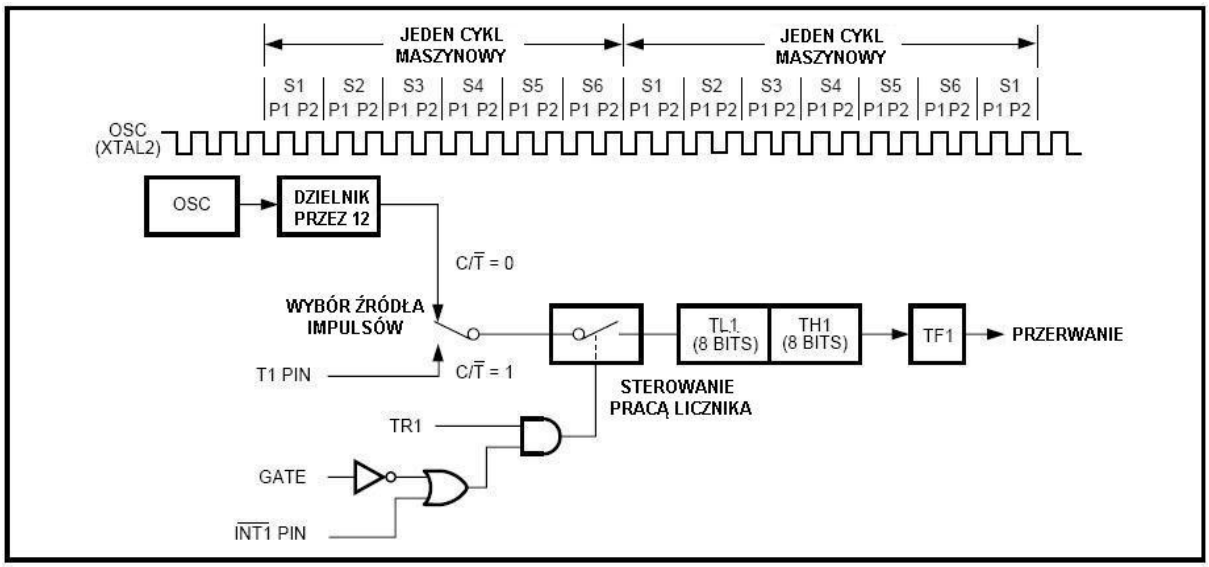
Przepełnienie licznika (przejście od stanu 1 1111 1111 1111 do stanu 0 0000 0000 0000) powoduje generowanie sygnału przerwania poprzez ustawienie flagi przepełnienia TFx.

Dołączenie wejścia impulsów zliczanych czyli uruchomienie licznika ma miejsce wtedy, kiedy TRx=1 oraz kiedy GATE=0 lub /INTx=1.

Ustawienie GATE=1 powoduje, że praca licznika jest kontrolowana zewnętrznie poprzez odpowiednie wejście portu P.3 (piny /INT0 oraz /INT1). Czas trwania impulsu sterującego na tym wejściu określa czas zliczania przez licznik.

Przykład. Zapisanie do rejestru TMOD danej 10000100B (GATE[1]=1, C/T[1]=1, M1M0[1]=0, GATE[0]=0, C/T[0]=1, M1M0[0]=0) np. MOV TMOD,#86H , powoduje, że licznik T1 będzie zliczał impulsy wewnętrznego generatora (timer) ; zliczanie będzie miało miejsce wtedy, kiedy na wejściu mikrokontrolera /INT1 będzie stan „1” zewnętrznego sygnału sterującego oraz kiedy TR1=1; licznik T0 będzie zliczał impulsy z zewnętrznego źródła z wejścia T0 mikrokontrolera; zliczanie będzie miało miejsce kiedy TR0=1; oba liczniki skonfigurowano w trybie 0.

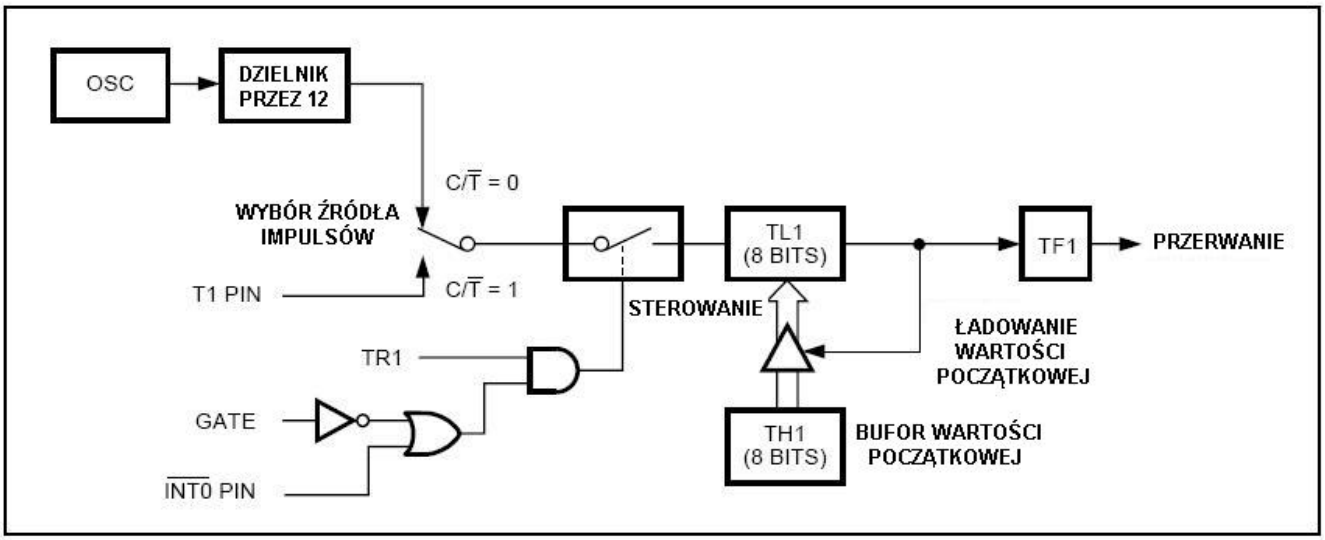
1.4.2.2 Tryb 1.



Tryb 1 jest identyczny z trybem 0 poza tym, że bufor licznika jest 16-bitowy (THxTLx) a przepełnienie licznika i wygenerowanie sygnału przerwania ma miejsce po zmianie stanu z FFFFH na 0000H.

1.4.2.3 Tryb 2.

Rysunek przedstawia konfigurację licznika (w tym przypadku T1) w trybie 2.



W trybie 2 liczniki pracują jako 8-bitowe wykorzystując jako bufor wynikowy rejestr TLx z automatycznym przeładowywaniem licznika.

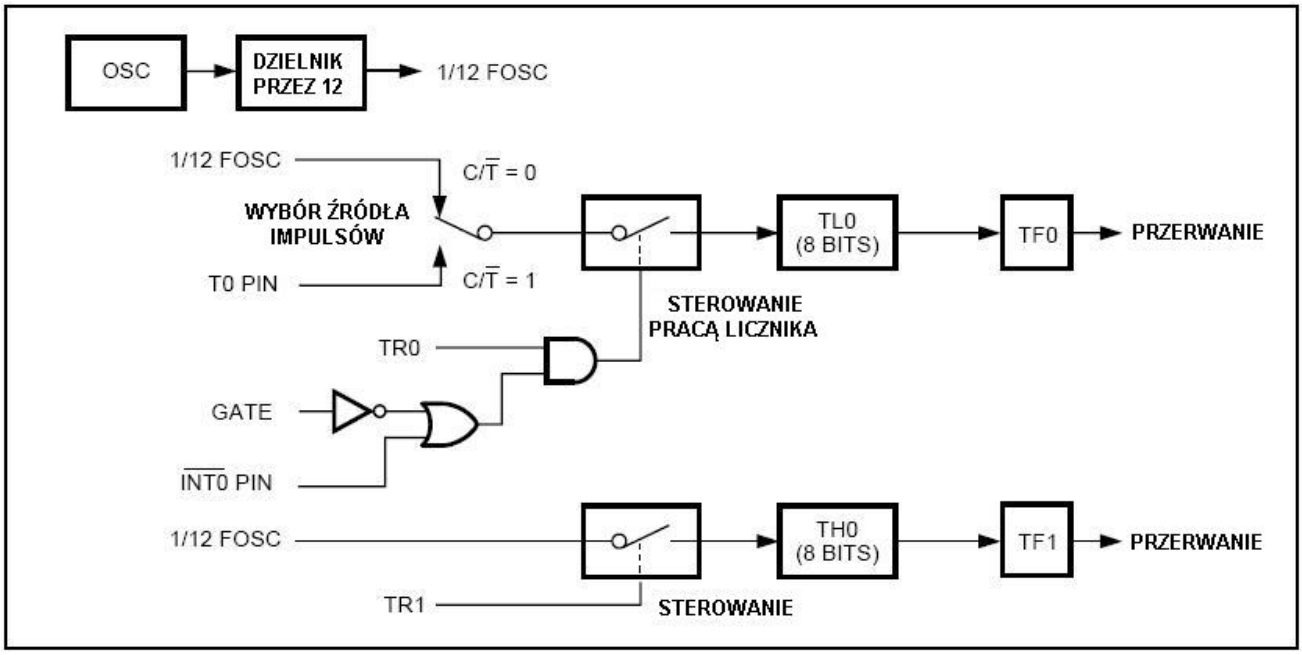
Każde przepełnienie licznika (zmiana stanu z 1111 1111 na 0000 0000) powoduje zarówno ustawienie flagi wywołania przerwania TFx jak i załadowanie wartości początkowej, którą jest zawartość bufora THx (można ją zmieniać dowolnie programowo w zakresie 00H-FFH). Przeładowanie nie zmienia zawartości bufora wartości początkowej THx.

UWAGA! Ten tryb pracy licznika T1 jest szczególnie istotny, gdyż wykorzystywany jest do taktowania pracy układu transmisji szeregowej UART).

1.4.2.4 Tryb 3.

UWAGA! Ustawienie trybu 3 dla licznika T1 uniemożliwia jego wykorzystywanie.

Rysunek przedstawia konfigurację licznika w trybie 3.



W trybie 3 rejestry TL0 i TH0 licznika T0 wykorzystywane są jako odrębne liczniki 8-bitowe.

Do sterowania i kontroli pracy licznika T0 w trybie 3 wykorzystywane są bity licznika T1 (bit startu TR1 oraz flaga przepełnienia TF1) dlatego licznik T1 nie może być wykorzystywany w aplikacjach, które wykorzystywałyby te bity.

Dla TL0 w trybie 3 logika sterowania i kontroli działania jest identyczna jak dla trybów 0 i 1. W przypadku TH0 możliwe jest jedynie zliczanie impulsów wewnętrznych (timer) a do uruchamiania licznika służy jedynie bit TR1.

Ustawienie trybu 3 dla licznika T0 ma tę zaletę, że pozwala wówczas uzyskać możliwość korzystania z 3 liczników (8-bitowych TL0 i TH0 oraz licznika T1 w trybie 0, 1 lub 2). Sytuacja taka może np. dotyczyć zastosowania wykorzystywania T1 do taktowania prędkości pracy układu UART, gdyż nie wymaga on zastosowania systemu przerwań i stałej kontroli pracy licznika.

1.5 Układ transmisji szeregowej UART mikrokontrolera.

Układ transmisji szeregowej umożliwia mikrokontrolerowi komunikację z innymi urządzeniami wykorzystującymi ten sposób wymiany danych jak i innymi mikrokontrolerami w systemach wieloprocesorowych

1.5.1 Układy interfejsów szeregowych – podstawowe pojęcia.

Równoległa transmisja danych jest praktyczna i szybka w przypadku transmisji lokalnych, ale nie jest to rozwiązanie wygodne, gdy transmisja odbywa się na odległości większe niż kilka metrów. Jest to związane z wysokimi kosztami wielożyłowych przewodów ale przede wszystkim, ze względu na występowanie zakłóceń powodowanych przez sąsiadujące przewody wytwarzające pole elektromagnetyczne przy przesyle sygnałów, ograniczona jest częstotliwość sygnałów, a co za tym idzie prędkość transmisji W systemach mikrokomputerowych transmisje danych do urządzeń zewnętrznych oraz pomiędzy systemami zwykle realizuje się stosując transmisję szeregową.

W najprostszej postaci transmisji szeregowej dane przesyła się linią dwuprzewodową. W danym czasie dane przesyła się tylko w jednym kierunku . Jedno z urządzeń pełni rolę nadajnika a drugie odbiornika informacji. Ten sposób transmisji nazywa się jednokierunkowym *(**half duplex**)* i jest stosowany, gdy dane przesyła się z zasady w jednym kierunku np. do drukarki.

Alternatywnym sposobem jest stosowanie łącza z dwoma przewodami, ze wspólnym przewodem zwrotnym do przesyłania danych w obu kierunkach równocześnie, czyli w trybie transmisji dwukierunkowej *(**full duplex**)*. Przykładem tego rodzaju transmisji jest ciągle używany standard RS232C szczególnie w zastosowaniach technicznych.

Stosowane są dwa rodzaje transmisji: **asynchroniczny** i **synchroniczny**. Dotyczy to występowania tzw. sygnału zegarowego, który wyznacza momenty nadawania informacji i wymusza odpowiednie momenty jej odbierania tak aby transmisja odbywała się bez błędów.

W trybie **asynchronicznym** nie występuje przesyłanie sygnału synchronizującego. Powoduje to, że w dłuższym czasie następuje rozsynchronizowanie pracy układu nadajnika i odbiornika (częstotliwości pracy obu układów różnią się w stopniu uniemożliwiającym prawidłową współpracę) i poprawna transmisja nie może mieć miejsca. Dlatego w tym trybie przesyła się osobno pojedyncze bajty lub znaki; stanowią one pojedynczy rekord danych. Czas ich transmisji jest na tyle krótki, że prawdopodobieństwo braku synchronizacji jest znikome. Synchronizacja pomiędzy nadajnikiem i odbiornikiem następuje na początku przesyłania pojedynczego rekordu. Dodatkowym ograniczeniem jest konieczność stosowania tej samej częstotliwości pracy generatorów taktujących pracę układów nadawczo-odbiorczych (wymaga to wzajemnej umowy i dokonania jednakowych ustawień przez użytkowników) Rekordy można przesyłać w dowolnych odstępach czasu, pomiędzy nimi system jest w stanie oczekiwania tzw. stan „ciszy”.

Podstawowa wadą transmisji asynchronicznej jest to, że do informacji dodaje się pewną liczbę bitów nadmiarowych, mających znaczenie kontroli transmisji, przez co efektywność transmisji przy przesyłaniu dużej ilości danych jest mała, szczególnie, gdy dane muszą być przesyłane z dużą szybkością.

W przypadku transmisji synchronicznej dane przesyła się w dużych blokach wielobajtowych, synchronizując transmisję tylko na początku bloku danych. W tym trybie transmisji zegary nadajnika i odbiornika muszą zapewnić synchronizację w czasie przesyłania całego bloku danych, albo przez stosowanie układów pętli fazowej PLL albo przez przesyłanie częstotliwości zegara osobnym przewodem. W pewnych przypadkach stosuje się odtwarzanie częstotliwości zegarowej z przesyłanego sygnału danych.

W obu trybach transmisji stosuje metody detekcji błędów transmisji: w trybie asynchronicznym jest przesyłanie bitu parzystości (nieparzystości) dodawanego do każdego bajtu lub znaku, w trybie synchronicznym stosuje się przesyłanie cyklicznych kodów nadmiarowych (CRC).

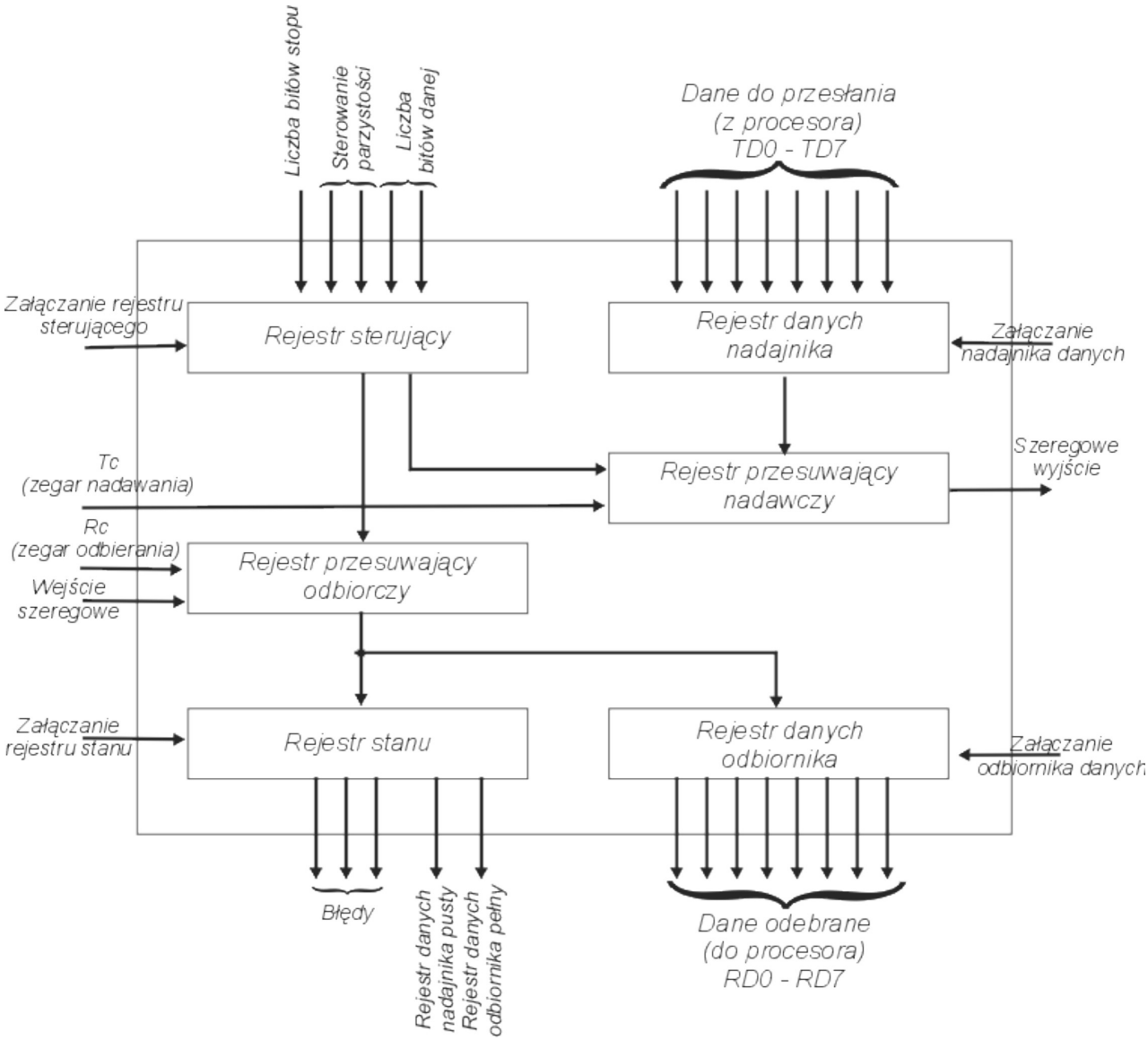
W asynchronicznym trybie transmisji, w stanie spoczynkowym („ciszy”) sygnał ma wartość 1. Na początku transmisji każdego znaku lub bajtu wysyła się bit startu. Ma on poziom 0, czas trwania równy czasowi przesłania jednego bitu i urządzenie odbiorcze traktuje bit startu jako sygnał synchronizujący początek transmisji. Po odebraniu bitu startu urządzenie odbiorcze sprawdza poziom sygnału w czasie następnych *n* bitów i dekoduje na tej podstawie transmitowane dane. Liczba przesyłanych bitów informacji może wynosić od 5 do 8. W systemach komputerowych zwykle transmituje 7 lub 8 bitów, ponieważ dane binarne są 8-bitowe, a znaki alfanumeryczne 7-bitowe. Po ciągu bitów danych może być przesłany opcjonalnie bit parzystości (nieparzystości). Stanowi on element najprostszej metody kontroli poprawności transmisji skuteczny jednakże tylko w przypadku przekłamania jednego bitu transmisji.

Na końcu przesyłanego słowa przesyła się pewną liczbę bitów stopu, mających poziom 1. Bity stopu stanowią separator pomiędzy kolejnymi przesyłanymi znakami zapewniający, że urządzenie odbiorcze będzie mogło wykryć bit startu następnego słowa. Stosuje się przesyłanie 1, 2 lub 1,5 bitu stopu. Ostatni przypadek oznacza, że przesyła się na zmianę w kolejnych słowach 1 i 2 bity stopu.

Szybkość transmisji określa się w bodach, dla danych binarnych jest liczba bitów na sekundę. Przyjętymi standardowo szybkościami są: 50, 75, 110, 150, 300, 600, 1200, 2400, 4800, 9600, i 19200 bodów. Szybkość ta nie uwzględnia przerwy pomiędzy kolejnymi słowami. W przypadku transmisji lokalnych lub synchronicznych stosuje się szybkości większe, do 115 K bodów.

Transmisję szeregową realizuje się przy użyciu układów UART (*Universal Asynchronous Receiver-Transmitter)* czyli dosłownie uniwersalny asynchroniczny układ odbiorczo-nadawczy lub USART *(U Synchronous ART*), który dodatkowo może realizować transmisję w trybie synchronicznym.

Przykładowe bloki wchodzące w skład standardowego układu transmisji szeregowej przedstawia rys.2.1.



Schemat blokowy uniwersalnego asynchronicznego nadajnika/odbiornika

Złącza urządzeń zawierają trzy typy informacji, zwykle przechowywanych w trzech rejestrach:

- Rejestr danych – zawiera daną do przesłania do jednostki złącza szyny lub stanowi odbiornik danej z tego złącza
- Rejestr stanu – zawierający informacje, kiedy dane dostępne są do wprowadzenia lub, kiedy rejestr może przyjąć dane
- Rejestr sterujący – zawierający informacje używane do zainicjowania trybów pracy oraz do określenia logicznych i fizycznych właściwości kanału danych

1.5.2 Charakterystyka układu transmisji szeregowej mikrokontrolera 8051.

Układ transmisji szeregowej mikrokontrolera 8051 pozwala na jednoczesną transmisję danych w obu kierunkach czyli w trybie full duplex.

Dodatkowo układ odbiornika posiada buforowany rejestr odbiorczy, co umożliwia rozpoczęcie odbioru kolejnego rekordu przed odczytaniem poprzedniego. Jednakże jeżeli zakończony zostanie odbiór kolejnego znaku przed odczytaniem poprzedniego następuje utrata wcześniejszej informacji przez jej nadpisanie (overwrite).

Buforom nadawczemu i odbiorczemu odpowiada rejestr specjalny SFR o tym samym adresie w przestrzeni adresowej RAM i tej samej nazwie SBUF (Serial BUFfer). W rzeczywistości są to dwa różne rejestry fizyczne a ich rozróżnienie odbywa się poprzez wykonywaną operację (rejestr SBUF nadajnika może być jedynie zapisywany [WR] natomiast odbiornika tylko odczytywany [RD]).

Konfigurację pracy układu transmisji szeregowej przeprowadza się przy użyciu rejestrów SCON oraz PCON.

1.5.3 Rejestry specjalne SFR związane z układem transmisji szeregowej.

Podstawowym rejestrem związanym z pracą układu transmisji szeregowej jest rejestr SCON.

SCON (Serial port CONtrol) – rejestr sterujący portu szeregowego o adresie 98H zatem dostępny także poprzez instrukcje manipulacji bitowych

Numer bitu

	SCON.7	SCON.6	SCON.5	SCON.4	SCON.3	SCON.2	SCON.1	SCON.0
98H	**SM0**	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**
	9F	**9E**	**9D**	**9C**	**9B**	**9A**	**99**	**98**

Adres bitu (HEX)

a) **SM0, SM1 (Serial Mode specifier) – (SCON.7 i SCON.6 lub 9FH i 9EH)** bity wyboru trybu pracy portu szeregowego

SM0	SM1	Tryb pracy portu szeregowego	Częstotliwość transmisji danych
0	0	Tryb 0 – synchroniczna 8-bitowa	Stała, **f_{osc}/12**
0	1	Tryb1 – asynchroniczna 8-bitowa	Zmienna , programowana
1	0	Tryb2 – synchroniczna 9-bitowa	Stała, **f_{osc}/32 lub f_{osc}/64**
1	1	Tryb3 – asynchroniczna 9-bitowa	Zmienna, programowana

b) **SM2 (Serial Mode specifier) – (SCON.5 lub 9DH)** uaktywnia (SM2=1) tryb dla pracy wieloprocesorowej;

c) **REN (Reception ENable) – (SCON.4 lub 9CH)** pozwala na programowe sterowanie pracą odbiornika portu szeregowego; załącza (REN=1) lub wyłącza (REN=0) odbiór danych przez port szeregowy

d) **TB8 (Transmitted Bit nr 8) – (SCON.3 lub 9BH)** – 9 bit transmitowanej danej w trybie 2 i 3 pracy portu szeregowego (wartość ustawiana programowo)

e) **RB8 (Received Bit nr 8) – (SCON.2 lub 9AH)** – 9 bit odebranej danej w trybie 2 i 3 pracy portu szeregowego; w trybie 1 jeżeli SM2=0, RB8 odpowiada wartości odebranego bitu stopu; w trybie 0 nie używany

f) **TI (Transmit Interrupt) – (SCON.1 lub 99H)** – flaga żądania przerwania od układu nadajnika transmisji szeregowej; ustawiana sprzętowo po zakończeniu transmisji 8 bitu danej w trybie 0 lub po rozpoczęciu transmisji bitu stopu w pozostałych trybach pracy; kasowanie znacznika musi być realizowane programowo

g) **RI (Receive Interrupt) – (SCON.0 lub 98H)** – flaga żądania przerwania od układu odbiornika transmisji szeregowej; ustawiana sprzętowo po zakończeniu odbioru 8 bitu danej w trybie 0 lub w połowie czasu trwania odbioru bitu stopu w pozostałych trybach pracy (z wyjątkami dotyczącymi konfiguracji bitu SM2); kasowanie znacznika musi być realizowane programowo

2 Lista rozkazów mikrokontrolera 8051... czyli co możesz dla mnie zrobić?

W rozdziale tym omówione zostaną wszystkie możliwe składnie instrukcji mikrokontrolera 8051.
W części pierwszej przedstawione zostaną dokładne informacje dotyczące każdej instrukcji wraz z przykładami ich zapisu w rzeczywistych przypadkach. Druga część zawiera Wersję tabelaryczną, którą zawsze warto mieć pod ręką zwłaszcza w początkach nauki kiedy jeszcze nie opanowaliśmy jej na tyle, aby „sypać nimi z rękawa" (co po napisaniu odpowiedniej liczby programów przychodzi samo z siebie czy tego chcemy czy nie).

Na początku warto wiedzieć, że:

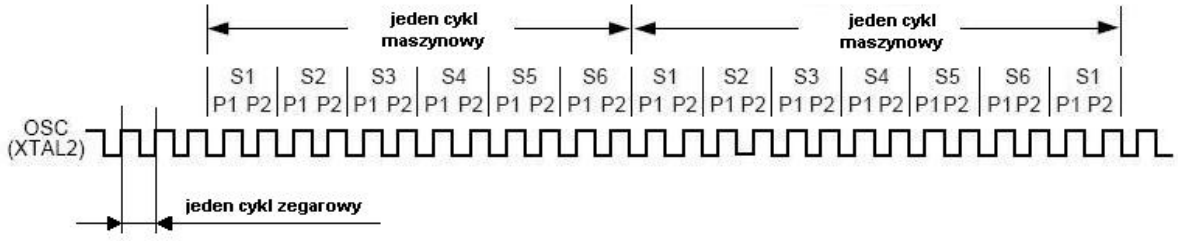
- **Mnemonik** – składnia instrukcji zawierająca anglojęzyczny skrót opisu operacji wraz z argumentami instrukcji (dane lub adresy danych) np. **ADD A,Rn (w praktyce np. ADD A,R5)**; rozumienie znaczenia tego skrótu znacznie ułatwia zapamiętanie poszczególnych instrukcji a co najważniejsze pozwala na rozumienie i sensowne stosowanie danej instrukcji w programie; użyte w opisie ogólnym symbole wyjaśnione zostały w legendzie załączonej do listy.
- **Operacja** – przedstawienie w symbolicznej formie działania instrukcji; w pełnej wersji opisu dodatkowo skomentowane w bardziej przystępnej formie np.
  Operacja: A ← A or (Ri) , wykonuje sumę logiczną danych zawartych w akumulatorze
  i komórce wewnętrznej pamięci RAM o adresie podanym pośrednio jako dana w rejestrze
  Ri (R0 lub R1) aktualnie wybranego banku rejestrów. Wynik umieszcza w akumulatorze
- **Struktura bajtów** – kod maszynowy instrukcji czyli przedstawienie w formie binarnej kodu instrukcji, który w rzeczywistości zostanie umieszczony w pamięci programu; informacja ta jest istotna przede wszystkim dla osób zajmujących się tworzeniem kompilatorów, asemblerów;
  dla „normalnego" użytkownika bardzo istotna jest informacja o ilości bajtów tworzących kod instrukcji ze względu na ograniczony rozmiar pamięci dostępny dla programisty np. 2KB, 4KB (max 64 KB); jest to jedna z podstawowych zalet tworzenia programu w asemblerze, że stale możemy kontrolować „wielkość" programu co przy językach wyższego poziomu jest w praktyce niemożliwe.

Przykład. Poniższy fragment programu zajmie w pamięci 17 bajtów

```
                    DISPLAY:
0000 C0E0               PUSH ACC                ; 2 bajty
0002 908002             MOV DPTR,#STAN          ; 3 bajty
0005                WAIT:
0005 E0                 MOVX A,@DPTR            ; 1 bajt
0006 20E7FC             JB ACC.7,WAIT           ; 3 bajty
0009 908001             MOV DPTR,#WR_ZNAK       ; 3 bajty
000C D0E0               POP ACC                 ; 1 bajt
000E F0                 MOVX @DPTR,A            ; 1 bajt
000F 020000   F         LJMP KLAWISZ            ; 3 bajty
                                                ----------------------
                                                Razem: 17 bajtów
```

Adres_pierwszego_bajtu Struktura_bajtów

- **Cykle** – ilość cykli maszynowych niezbędnych do wykonania instrukcji; na podstawie tej informacji oraz danej częstotliwości rezonatora zegara systemowego możemy obliczyć czas wykonania dowolnego fragmentu programu; jest to kolejna zaleta programowania w asemblerze szczególnie istotna przy tworzeniu systemów czasu rzeczywistego.
  Cykl maszynowy to najkrótszy czas wykonania pojedynczej instrukcji przez procesor (wiele instrukcji wymaga kilku cykli); w przypadku mikrokontrolera 8051 jeden cykl maszynowy odpowiada 12 cyklom zegarowym (okresom rezonatora użytego w systemie).



Przykład.
Jeżeli użyjemy rezonatora o częstotliwości f_{rez}=24 MHz to cykl maszynowy $_M=12T_c=12/f_{rez}=0.5\mu s$ wówczas najkrótszy czas wykonania fragmentu z poprzedniego przykładu będzie wynosił 8µs

```
DISPLAY:
        PUSH ACC                ; 2 cykle
        MOV DPTR,#STAN          ; 2 cykle
WAIT:
        MOVX A,@DPTR            ; 2 cykle
        JB ACC.7,WAIT           ; 2 cykle
        MOV DPTR,#WR_ZNAK       ; 2 cykle
        POP ACC                 ; 2 cykle
        MOVX @DPTR,A            ; 2 cykle
        LJMP KLAWISZ            ; 2 cykle
                                ----------------------
                Razem: 16 cykli*0.5µs=8µs
```

- **Znaczniki (flagi)** – zmieniane przez instrukcję bity flag w rejestrze stanu procesora PSW; bardzo istotna informacja dla wielu instrukcji a przede wszystkim dla operacji arytmetycznych oraz skoków warunkowych ; dopiero po uwzględnieniu bitów flag możemy uzyskać pełny i prawdziwy wynik wielu operacji np. wynik dodawania dwóch

liczb 8-bitowych może być liczba 9-bitową co uzyskujemy po uwzględnieniu znacznika przeniesienia CY (CARRY).

.

Oznaczenia i symbole użyte w opisie instrukcji mikrokontrolerów rodziny MCS'51:

A	-	akumulator – kiedy mówimy o adresowaniu bajtowym
ACC	-	akumulator – kiedy mówimy o adresowaniu bitowym
B	-	rejestr B
Rn	-	rejestr roboczy, R0...R7 (n = 0 dla R0, n = 7 dla R7)
Ri	-	rejestr roboczy – wskaźnik danych (i = 0 dla R0, i = 1 dla R1)
rrr	-	rejestr R0...R7 (rrr=000 dla R0, rrr=111 dla R7)
DPTR	-	wskaźnik danych (DPH.DPL)
PC	-	licznik rozkazów
SP	-	wskaźnik stosu
C, CY	-	znacznik przeniesienia
AC	-	znacznik przeniesienia pomocniczego
OV	-	znacznik nadmiaru
dana	-	zmienna 8 bitowa
dana_16	-	zmienna 16 bitowa
adr	-	adres pierwszych 128 bajtów wewnętrznej pamięci RAM (adr=0...7FH) lub rejestrów specjalnych SFR (adr=80H...0FFH)
rel	-	8 - bitowe przesunięcie o wartościach z przedziału (-128,127)
adr_11	-	11 – bitowy adres pamięci programu
adr_16	-	16 – bitowy adres pamięci programu
and	-	iloczyn logiczny
or	-	suma logiczna
xor	-	suma modulo 2 (różnica symetryczna)
not	-	negacja logiczna
( )	-	zawartość komórki pamięci danych lub programu o adresie podanym w nawiasie
@	-	w mnemoniku poprzedza adres pośredni
#	-	w mnemoniku poprzedza argument bezpośredni
X	-	w zapisie operacji oznacza zawartość rejestru X
(X)	-	w zapisie operacji oznacza zawartość pamięci o adresie X
<dest>	-	miejsce pobrania argumentu i zapisania wyniku operacji
<src>	-	miejsce pobrania drugiego argumentu operacji

2.1 Alfabetyczna lista rozkazów mikrokontrolera 8051.

ACALL adr_11 (ang. *Absolute call*; *subroutine call on page)*

skocz do programu na stronie

Zawartość licznika rozkazów zwiększona o 2 przy pobraniu rozkazu jest ładowana na stos (pierwszy bajt mniej znaczący); zawartość wskaźnika stosu jest zwiększana o 2, a do bitów 0-10 licznika rozkazów jest wpisywany 11 - bitowy adres bezpośredni. Pięć bardziej znaczących bitów licznika rozkazów nie zmienia się. Skok do podprogramu wykonuje się pod adresem na tej stronie (o pojemności 2K), na której jest umieszczony pierwszy bajt rozkazu następnego po ACALL.

Operacja: $PC \leftarrow PC + 2$
$SP \leftarrow SP + 1$ $(SP) \leftarrow PC_{7-0}$
$SP \leftarrow SP + 1$ $(SP) \leftarrow PC_{15-8}$
$PC_{10-0} \leftarrow$ adr_11
PC_{15-11} *bez zmian*

Struktura bajtów:

adr_{10} adr_9 adr_8 0	0 0 0 1

Adr_{7-4}	adr_{3-0}

Bajty: 2
Cykle: 2

Przykład: ACALL 500H
ACALL PODPROGRAM

Mnemonik: Add

ADD A, <src_byte> (ang. add to accumulator)
dodaj do akumulatora

Do zawartość akumulatora jest dodawany wskazany argument, a wynik jest wpisywany do akumulatora. Zgodnie z wynikiem operacji są ustawiane znaczniki: CY, AC i OV. Są możliwe cztery tryby adresowania argumentu <src>.

ADD A,Rn – do danej zawartej w akumulatorze dodaje daną zapisaną w jednym z rejestrów pomocniczych np. R5
Operacja: $A \leftarrow A + Rn$
Struktura bajtów:

0 0 1 0	1 r r r

Bajty: 1
Cykle: 1

Przykład: ADD A,R5 (0010 1101)

ADD A,adr - do danej zawartej w akumulatorze dodaje daną zapisaną w pamięci wewnętrznej RAM w komórce wskazanej przez 8-bitowy adres zapisany w instrukcji
Operacja: A ← A + (adr)
Struktura bajtów:

0 0 1 0	0 1 0 1

Adr

Bajty: 2
Cykle: 1
Przykład: ADD A,50H (0010 0101 0101 0000)

ADD A,@Ri - do danej zawartej w akumulatorze dodaje daną zapisaną w komórce wewnętrznej pamięci o adresie wskazanym przez 8-bitową daną zawartą w rejestrze pomocniczym Ri (R0 lub R1). Uwaga: Jest to jedyny możliwy do zastosowania tryb adresowania w przypadku kiedy dana źródłowa umieszczona jest w obszarze pamięci IRAM o adresie <adr> w zakresie 128-255.
Operacja: A ← A + (Ri)
Struktura bajtów:

0 0 1 0	0 1 1 i

Bajty: 1
Cykle: 1
Przykład: ADD A,@R0 (0010 0110)
ADD A,#dana - do danej zawartej w akumulatorze dodaje 8-bitową daną wskazaną bezpośrednio w instrukcji
Operacja: A ← A + dana
Struktura bajtów:

0 0 1 0	0 1 0 0

Dana

Bajty: 2
Cykle: 1
Przykład: ADD A,#80 (0010 0100 0101 0000)

Mnemonik: ADDC

ADDC A,<src_byte> (ang. *add to accumulator with carry)*
dodaj do akumulatora z przeniesieniem

Do zawartości akumulatora jest dodawana zawartość znacznika przeniesienia CY

i wskazany argument. Wynik jest wpisywany do akumulatora. Zgodnie z wynikiem operacji są ustawiane znaczniki: CY, AC i OV. Są możliwe cztery tryby adresowania argumentu <src>.

Działanie instrukcji różni się od poprzednio omawianej jedynie tym, że wynik dodawania uwzględnia dodatkowo stan znacznika CY jako argumentu dodawania o wartości 0 lub 1.

ADDC A,Rn

Operacja: A ← A + C + Rn

Struktura bajtów:

0 0 1 1	1 r r r

Bajty: 1

Cykle: 1

Przykład: ADDC A,R4 (0011 1100)

ADDC A,adr

Operacja: A ← A + C + (adr)

Struktura bajtów:

0 0 1 1	0 1 0 1

Adr

Bajty: 2

Cykle: 1

Przykład: ADDC A,60 (0011 0101 0011 1100)

ADDC A,@Ri

Uwaga: Jest to jedyny możliwy do zastosowania tryb adresowania w przypadku kiedy dana źródłowa umieszczona jest w obszarze pamięci IRAM o adresie <adr> w zakresie 128-255.

Operacja: A ← A + C + (Ri)

Struktura bajtów:

0 0 1 1	0 1 1 i

Bajty: 1

Cykle: 1

Przykład: ADDC A,@R1 (0011 0111)

ADDC A,#dana

Operacja: A ← A + C + dana

Struktura bajtów:

0 0 1 1	0 1 0 0

Dana

Bajty: 2

Cykle: 1

Przykład: ADDC A,#100 (0011 0100 0110 0100)

Mnemonik: AJMP (Absolute Jump)

AJMP adr_11 (ang. *unconditional jump on page)*
skocz bezwarunkowo na stronie

Do bitów 0 - 10 licznika rozkazów (po zwiększeniu jego zawartości o 2 przy pobraniu rozkazu) jest wpisywany 11 - bitowy adres bezpośredni. Pięć bardziej znaczących bitów licznika rozkazów nie zmienia się. Skok wykonuje się pod adres na tej stronie (o pojemności 2K) na której jest umieszczony pierwszy bajt rozkazu następującego po AJMP.

Operacja: $PC_{10-0} \leftarrow$ adr_11
PC 15-11 *bez zmian*

Struktura bajtów:

adr_{10} adr_9 adr_8 0	0 0 1 0

adr_{7-0}

Bajty: 2
Cykle: 2
Przykład: AJMP 200H
AJMP DALEJ

Mnemonik:ANL (Logical-AND for byte variables)

ANL <dest_byte>, <src_byte> (ang. *logical AND)*
pomnóż logicznie

Wykonywany jest iloczyn logiczny (bit po bicie) wskazanych argumentów. Wynik jest wpisywany do miejsca , z którego został pobrany argument <dest_byte>. Jest możliwych sześć kombinacji trybów adresowania argumentów <dest> i <src>.

ANL A,Rn – iloczyn logiczny danej zawartej w akumulatorze oraz danej zapisanej w rejestrze pomocniczym aktualnie wybranego banku rejestrów

Operacja: A ← A and Rn
Struktura bajtów:

0 1 0 1	1 r r r

Bajty: 1
Cykle: 1
Przykład: ANL A,R6 (0101 1110)

ANL A,adr – iloczyn logiczny danej zawartej w akumulatorze oraz danej zapisanej w komórce wewnętrznej pamięci RAM o adresie wskazanym bezpośrednio w instrukcji

Operacja: A ← A and adr
Struktura bajtów:

0 1 0 1	0 1 0 1

Adr

Bajty: 2

Cykle: 1
Przykład: ANL A,120 (0101 0101 0111 1000)

ANL A,@Ri – iloczyn logiczny danej zawartej w akumulatorze oraz danej zapisanej w komórce wewnętrznej pamięci RAM o adresie wskazanym przez 8-bitową daną zawartą w rejestrze pomocniczym Ri (R0 lub R1). Uwaga: Jest to jedyny możliwy do zastosowania tryb adresowania w przypadku kiedy dana źródłowa umieszczona jest w obszarze pamięci IRAM o adresie <adr> w zakresie 128-255.
Operacja: A ← A and (Ri)
Struktura bajtów:

0 1 0 1	0 1 1 i

Bajty: 1
Cykle: 1
Przykład: ANL A,@R0 (0101 0110)

ANL A,#dana – iloczyn logiczny danej zawartej w akumulatorze oraz 8-bitowej danej wskazanej bezpośrednio w instrukcji
Operacja: A ← A and dana
Struktura bajtów:

0 1 0 1	0 1 0 0

Dana

Bajty: 2
Cykle: 1
Przykład: ANL A,#0F0H (0101 0100 1111 0000)

ANL adr,A - iloczyn logiczny danej zawartej w akumulatorze oraz danej zapisanej w komórce wewnętrznej pamięci RAM o adresie wskazanym bezpośrednio w instrukcji
Operacja: (adr) ← (adr) and A
Struktura bajtów:

0 1 0 1	0 0 1 0

Adr

Bajty: 2
Cykle: 1
Przykład: ANL 33,A (0101 0010 0010 0001)

ANL adr,#dana - - iloczyn logiczny danej zapisanej w komórce wewnętrznej pamięci RAM o adresie wskazanym bezpośrednio w instrukcji oraz 8-bitowej danej zapisanej bezpośrednio w instrukcji
Operacja: (adr) ← (adr) and dana
Struktura bajtów:

0 1 0 1	0 0 1 1

Adr

Dana

Bajty: 3
Cykle: 2
Przykład: ANL 44,# 11 (0101 0011 0010 1100)
ANL BUFOR,#MASKA

Mnemonik: ANL (Logical-AND for bit variables)

ANL C, bit (ang. *logical AND carry with direct bit)*
pomnóż logicznie przez bit

Jeżeli wartość logiczna bitu o podanym adresie bezpośrednim jest równa 0, to znacznik przeniesienia CY (akumulator procesora boolowskiego) jest zerowany, w przeciwnym wypadku zawartość CY nie zmienia się (jest ona mnożona logicznie przez zawartość zaadresowanego bitu).
Operacja: C ← C and bit
Struktura bajtów:

1 0 0 0	0 0 1 0

adres bitu

Bajty: 2
Cykle: 2
Przykład: ANL C,20
ANL C,40H.1
ANL C,ACC.2

ANL C, /bit (ang. *logical AND carry with complement of direct bit)*
pomnóż logicznie przez negację zawartości bitu

Jeżeli wartość logiczna bitu o podanym adresie bezpośrednim jest równa 1, to znacznik przeniesienia CY (akumulator przeniesienia boolowskiego) jest zerowany, w przeciwnym wypadku zawartość CY nie zmienia się (jest ona mnożona logicznie przez negację zawartości zaadresowanego bitu).
Operacja: C ← C and (not bit)
Struktura bajtów:

1 0 1 1	0 0 0 0

Bajty: 2
Cykle: 2
Przykład: ANL C,/55
ANL C,/ACC.5

Mnemonik: CJNE (Compare and Jump if Not Equal)

CJNE <r>, <s>, d (ang. *compare and jump if not equal*)
porównaj argumenty i skocz, gdy nie są równe

Porównywane są wskazane argumenty. Jeżeli nie są one równe, to do zawartości licznika rozkazów jest dodawane przesunięcie *d* (liczba ze znakiem w kodzie U2), tzn. jest wykonywany skok względny. Dodawanie jest wykonywane po pobraniu kodu rozkazu skoku, zatem skok następuje względem adresu pierwszego bajtu rozkazu następnego po CJNE. jeżeli wartość argumentu <r> (jako liczby bez znaku) jest mniejsza niż argumentu <s>, to do znacznika CY wpisuje się jedynka; w przeciwnym razie znacznik CY jest zerowany. Stan innych znaczników oraz argumentów nie zmienia się. Są możliwe cztery kombinacje trybów adresowania argumentów <r> i <s>.

CJNE A,adr,rel – jeżeli dane zawarte w akumulatorze i komórce wewnętrznej pamięci RAM o adresie wskazanym bezpośrednio w instrukcji nie są równe wykonywany jest skok względny; w przeciwnym przypadku wykonywana jest kolejna zapisana po tej operacji instrukcja

Operacja: PC ← PC + 3
jeśli A < (adr) to C ← 1
jeśli A >/ (adr) to C ← 0
jeśli A <> (adr) to PC ← PC + rel

Struktura bajtów:

1 0 1 1	0 1 0 1

Adr

Rel

Bajty: 3
Cykle: 2
Przykład: CJNE A,30H,40
CJNE A,BUFOR,NIE_ROWNE

CJNE A,#dana,rel - jeżeli dana zawarta w akumulatorze i dana wskazana bezpośrednio w instrukcji nie są równe wykonywany jest skok względny; w przeciwnym przypadku wykonywana jest kolejna zapisana po tej operacji instrukcja

Operacja: PC ← PC + 3
jeśli A < dana to C ← 1
jeśli A >/ dana to C ← 0
jeśli A < > dana to PC ← PC + rel

Struktura bajtów:

1 0 1 1	0 1 0 0

Dana

Rel

Bajty: 3
Cykle: 2
Przykład: CJNE A,#66,123
CJNE A,#STALA,ROZNE

CJNE Rn,#dana,rel - jeżeli dana zawarta w rejestrze pomocniczym aktualnie wybranego banku rejestrów i dana wskazana bezpośrednio w instrukcji nie są równe wykonywany jest skok względny; w przeciwnym przypadku wykonywana jest kolejna zapisana po tej operacji instrukcja

Operacja: PC ← PC + 3
jeśli Rn < dana to C ← 1
jeśli Rn >/ dana to C ← 0
jeśli Rn < > dana to PC ← PC + rel

Struktura bajtów:

1 0 1 1	1 r r r

Dana

Rel

Bajty: 3
Cykle: 2
Przykład: CJNE R3,#0FH,0C0H
CJNE R4,#WSKAZNIK,POWTORZ

CJNE @Ri,#dana,rel - jeżeli dana zawarta w komórce wewnętrznej pamięci RAM o adresie wskazanym przez daną zapisaną w rejestrze pomocniczym Ri (R0 lub R1) i dana wskazana bezpośrednio w instrukcji nie są równe wykonywany jest skok względny; w przeciwnym przypadku wykonywana jest kolejna zapisana po tej operacji instrukcja

Operacja: PC ← PC + 3
jeśli (Ri) < dana to C ← 1
jeśli (Ri) >/ dana to C ← 0
jeśli (Ri) < > dana to PC ← PC + rel

Struktura bajtów:

1 0 1 1	0 1 1 I

Dana

Rel

Bajty: 3
Cykle: 2
Przykład: CJNE @R0,#100,50
CJNE @R1,#DANA,NEXT

Mnemonik: CLR (Clear)

CLR A (ang. *clear accumulator*)
wyzeruj akumulator

Zerowana jest zawartość akumulatora.

Operacja: A ← 0

Struktura bajtów:

1 1 1 0	0 1 0 0

Bajty: 1

Cykle: 1

Mnemonik: CLR bit (Clear bit)

CLR C (ang. *clear carry flag*)
wyzeruj znacznik przeniesienia

Zerowany jest znacznik przeniesienia CY(akumulator procesora boolowskiego)

Operacja: C ← 0

Struktura bajtów:

1 1 0 0	0 0 1 1

Bajty: 1
Cykle: 1

CLR bit (ang. *clear bit*)
wyzeruj bit

Zerowany jest bit o podanym adresie bezpośrednim.

Operacja: bit ← 0

Struktura bajtów: :

1 1 0 0	0 0 1 0

adres bitu

Bajty: 2
Cykle: 1
Przykład: CLR 2
CLR P1.1
CLR 20H.2

Mnemonik: CPL A (Complement Accumulator)

CPL A (ang. *complement accumulator*)
neguj akumulator

Negowany jest (bit po bicie) zawartość akumulatora. Tylko przy pomocy tego rozkazu w sekwencji :

CPL A
INC A

jest możliwa zmiana znaku liczby w kodzie U2.

Operacja: A ← not A

Struktura bajtów:

1 1 1 1	0 1 0 0

Bajty: 1
Cykle: 1

Mnemonik: CPL bit (Complement bit)

CPL C (ang. *complement carry flag*)
neguj znacznik przeniesienia

Negowany jest znacznik przeniesienia CY (akumulator procesora boolowskiego).

Operacja: C ← > not C

Struktura bajtów:

1 0 1 1	0 0 1 1

Bajty: 1
Cykle: 1

CPL bit (ang. *complement bit*)
neguj bit

Negowany jest bit o podanym adresie bezpośrednim.

Operacja: bit ← not bit

Struktura bajtów:

1 0 1 1	0 0 1 0

adres bitu

Bajty: 2
Cykle: 1
Przykład: CPL P2.3
CPL 10
CPL 21H.1

Mnemonik: DA A (Decimal-adjust Accumulator for Addition)

DA A (ang. *decimal adjust*)
wykonaj korekcję dziesiętną

Uwaga! Nie wykonuje się korekcja dziesiętna w wyniku odejmowania.

Wykonywana jest korekcja dziesiętna w wyniku dodawania. Operacja ta sprowadza wynik do postaci dwóch cyfr dziesiętnych (kod BCD), jeżeli argumenty były w kodzie BCD. Zgodnie z wynikiem operacji jest ustawiany znacznik CY (wskazuje, że liczba dziesiętna otrzymana w wyniku dodawania jest większa niż 99). Nie zmienia się stan znaczników AC i OV.

Korekcja przebiega w następujący sposób: jeżeli zawartość bitów 0 - 3 akumulatora jest większa niż 9 lub jest ustawiony znacznik AC, to do zawartości akumulatora jest dodawana liczba 6, po czym jeżeli zawartość bitów 4 - 7 akumulatora jest większa niż 9 lub jest ustawiony znacznik CY, to do tych bitów jest dodawana liczba 6. Jeżeli podczas tej ostatniej operacji wystąpi przeniesienie, to do znacznika CY jest wpisywana jedynka - w przeciwnym razie stan znacznika nie zmienia się.

Rozkaz DA A powinien być użyty wyłącznie w połączeniu z rozkazem dodawania (ADD lub ADDC). Również zliczanie w kodzie BCD powinno być wykonywane za pomocą rozkazu dodawania, na przykład:

```
ADD A, #1
DA A
```

Nie należy w takim przypadku używać rozkazu INC, ponieważ jego wykonanie nie powoduje ustawienia znaczników AC i CY.

Operacja: jeśli $A_{3-0} > 9$ lub AC = 1 to $A_{3-0} \leftarrow A_{3-0} + 6$
jeśli $A_{7-4} > 9$ lub C = 1 to $A_{7-4} \leftarrow A_{7-4} + 6$

Struktura bajtów:

0 0 0 1	0 1 0 1

Bajty: 1
Cykle: 1

Mnemonik: DEC byte (Decrement byte)

DEC <src> (ang. *decrement*)
zmniejsz o jeden

Od wskazanego argumentu jest odejmowana jedynka. Nie zmienia się stan znaczników. Są możliwe cztery tryby adresowania argumentu.

Uwaga! Jeżeli rozkaz jest użyty do zmiany stanu wyjścia, to jest odczytywana i modyfikowana zawartość rejestru wyjściowego portu (a nie stan logiczny z końcówek układu). Nie ustawia znacznika pożyczki!

DEC A – zmniejsza o 1 zawartość akumulatora
Operacja: A ← A – 1
Struktura bajtów:

0 0 0 1	0 1 0 0

Bajty: 1
Cykle: 1

DEC Rn – zmniejsza o 1 zawartość rejestru pomocniczego aktualnie wybranego banku rejestrów
Operacja: Rn ← Rn – 1
Struktura bajtów:

0 0 0 1	1 r r r

Bajty: 1
Cykle: 1
Przykład: DEC R2

DEC adr – zmniejsza o 1 zawartość komórki wewnętrznej pamięci RAM o adresie wskazanym bezpośrednio w instrukcji

Operacja: (adr) ← (adr) – 1

Struktura bajtów:

0 0 0 1	0 1 0 1

adr

Bajty: 2

Cykle: 1

Przykład: DEC 40
DEC LICZNIK

DEC @Ri - zmniejsza o 1 zawartość komórki wewnętrznej pamięci RAM o adresie wskazanym pośrednio jako dana w rejestrze Ri (R0 lub R1)

Operacja: (Ri) ← (Ri) – 1

Struktura bajtów:

0 0 0 1	0 1 1 i

Bajty: 1

Cykle: 1

Przykład: DEC @R1

Mnemonik: DIV (Divide)

DIV AB (ang. *divide*)
dziel

8-bitowa liczba bez znaku, zawarta w akumulatorze, jest dzielona przez 8-bitową liczbę bez znaku z rejestru B. Część całkowita ilorazu wpisuje się do akumulatora, a reszta - do rejestru B. Zerowane są znaczniki CY i AC.

Uwaga! Jeżeli w rejestrze B jest zero (00H), to w wyniku wykonania rozkazu zawartość akumulatora i rejestru B jest nieokreślona, a do znacznika OV wpisuje się jedynka.

Operacja: A ← Int (A/B), iloraz OV = 1 jeśli przed
B ← Mod (A/B), reszta dzieleniem B = 0

Struktura bajtów:

1 0 0 0	0 1 0 0

Bajty: 1 Cykle: 4

Mnemonik: DJNZ (Decrement and Jump if Not Zero)

DJNZ <src>, rel (ang. *decrement and jump if not zero*)
zmniejsz o 1 i skocz, gdy nie zero

Od wskazanego argumentu jest odejmowana jedynka. Jeżeli po tej operacji argument nie jest równy zeru, to do zawartości licznika rozkazów dodaje się przesunięcie d (liczba ze znakiem w kodzie U2), tzn. jest wykonywany skok względny. Dzieje się to po pobraniu kodu rozkazu skoku, a zatem skok następuje względem adresu pierwszego bajtu rozkazu następnego po DJNZ. Nie zmienia się stan znaczników. Argument może być umieszczony w rejestrze roboczym lub komórce wewnętrznej pamięci danych, adresowanej bezpośrednio (też w SFR).

Uwaga! Jeżeli rozkaz jest użyty do zmiany stanu wyjścia, to jest odczytywana i modyfikowana zawartość rejestru wyjściowego portu (a nie stan logiczny z końcówek układu).

DJNZ Rn,rel – zmniejsza o 1 zawartość rejestru pomocniczego aktualnie wybranego banku rejestrów i jeżeli wartość po zmianie jest różna od 0 wykonywany jest skok względny; w przeciwnym przypadku wykonywana jest kolejna zapisana po tej operacji instrukcja

Operacja: PC ← PC + 2
Rn ← Rn – 1
jeśli Rn < > 0 to PC ← PC + rel

Struktura bajtów:

1 1 0 1	1 r r r
Rel	

Bajty: 2
Cykle: 2
Przykład: DJNZ R7,POWTORZ

DJNZ adr,rel - zmniejsza o 1 zawartość komórki wewnętrznej pamięci RAM o adresie wskazanym bezpośrednio w instrukcji i jeżeli wartość po zmianie jest różna od 0 wykonywany jest skok względny; w przeciwnym przypadku wykonywana jest kolejna zapisana po tej operacji instrukcja

Operacja: PC ← PC + 2
(adr) ← (adr) + 3
jeśli (adr) < > 0 to PC ← PC + rel

Struktura bajtów:

1 1 0 1	0 1 0 1

Adr

Rel

Bajty: 3
Cykle: 2
Przykład: DJNZ 77,NASTEPNY

DJNZ LICZNIK,NEXT

Mnemonik: INC byte (Increment byte)

INC <src> (ang. *increment*)
zwiększ o jeden

Do wskazanego argumentu dodawana jest wartość 1. Nie zmienia się stan znaczników. Są możliwe cztery tryby adresowania argumentu.

Uwaga! Jeżeli rozkaz jest użyty do zmiany stanu wyjścia, to jest odczytywana i modyfikowana zawartość rejestru wyjściowego portu (a nie stan logiczny z końcówek układu). Nie ustawia znacznika przeniesienia.

INC A – zwiększa o 1 zawartość akumulatora
Operacja: A ← A + 1
Struktura bajtów:

0 0 0 0	0 1 0 0

Bajty: 1
Cykle: 1

INC Rn - zwiększa o 1 zawartość rejestru pomocniczego aktualnie wybranego banku rejestrów
Operacja: Rn ← Rn + 1
Struktura bajtów:

0 0 0 0	1 r r r

Bajty: 1
Cykle: 1
Przykład: INC R4

INC (adr) - zwiększa o 1 zawartość komórki wewnętrznej pamięci RAM o adresie wskazanym bezpośrednio w instrukcji
Operacja: (adr) ← (adr) + 1
Struktura bajtów:

0 0 0 0	0 1 0 1

adr

Bajty: 2
Cykle: 1
Przykład: INC 127
INC BUFOR

INC @Ri - zwiększa o 1 zawartość komórki wewnętrznej pamięci RAM o adresie wskazanym pośrednio jako dana w rejestrze Ri (R0 lub R1)
Operacja: (Ri) ← (Ri) + 1
Struktura bajtów:

0 0 0 0	0 1 1 i

Bajty: 1
Cykle: 1
Przykład: INC @R0

Mnemonik: Increment Data Pointer

INC DPTR (ang. *increment data pointer*)
zwiększ o 1 wskaźnik danych

Do 16-bitowego wskaźnika danych DPTR, złożonego z rejestrów specjalnych DPH (bardziej znaczący bit) i DPL (mniej znaczący bajt), jest dodawana jedynka. Nie zmienia się stan znaczników.

Operacja: DPTR ← DPTR + 1
Struktura bajtów:

1 0 1 0	0 0 1 1

Bajty: 1
Cykle: 2

Mnemonik: JB (Jump if Bit set)

JB bit,rel (ang. *jump if bit is set*)
skocz, gdy bit ustawiony

Jeżeli wartość bitu o podanym adresie bezpośrednim jest jedynką, to do zawartości licznika rozkazów dodaje się przesunięcie <rel> (liczba ze znakiem w kodzie U2), tzn. jest wykonywany skok względny. Dzieje się to po pobraniu kodu rozkazu skoku, a zatem skok następuje względem adresu pierwszego bajtu rozkazu następnego po JB. Testowany bit nie ulega zmianie.

Jeżeli warunek nie zostanie spełniony wykonywana jest instrukcja zapisana jako następna po JB.

Operacja: PC ← PC + 3
jeśli bit = 1 to PC ← PC + rel
Struktura bajtów:

0 0 1 0	0 0 0 0

adres bitu

Rel

Bajty: 3
Cykle: 2

Przykład: JB 22H.0,100
JB ACC.7,CZEKAJ

Mnemonik: JBC (Jump if Bit is set and Clear bit)

JBC bit,rel (ang. *jump if bit is set and clear bit*)
jeśli bit jest ustawiony, to zeruj go i skocz

Jeżeli wartość bitu o podanym adresie bezpośrednim jest jedynką, to jest on zerowany i do zawartości licznika rozkazów dodaje się przesunięcie <rel> (liczba ze znakiem w kodzie U2) jest wykonywany skok względny. Dzieje się to po pobraniu kodu rozkazu skoku, a zatem skok następuje względem adresu pierwszego bajtu rozkazu następnego po JBC.

Jeżeli warunek nie zostanie spełniony wykonywana jest instrukcja zapisana jako następna po JBC.

Uwaga! Jeżeli rozkaz jest użyty do zmiany stanu wyjścia, to jest odczytywana i modyfikowana zawartość rejestru wyjściowego portu (a nie stan logiczny z końcówek układu).

Operacja: PC ← PC + 3
jeśli bit = 1 to PC ← PC + rel i bit ← 0

Struktura bajtów:

0 0 0 1	0 0 0 0

adres bitu

Rel

Bajty: 3
Cykle: 2
Przykład: JBC 2BH.5,40
JBC TF0,TIME

Mnemonik: JC (Jump if Carry is set)

JC rel (ang. *jump if carry is set*)
skocz, jeśli jest przeniesienie

Jeżeli jest ustawiony znacznik przeniesienia (CY = 1), to do zawartości licznika rozkazów dodaje się przesunięcie <rel> (liczba ze znakiem w kodzie U2), tzn. Jest wykonany skok względny. Dzieje się to po pobraniu kodu rozkazu skoku, a zatem skok następuje względem adresu pierwszego bajtu rozkazu następnego po JC.

Jeżeli warunek nie zostanie spełniony wykonywana jest instrukcja zapisana jako następna po JC.

Operacja: PC ← PC + 2
Jeśli C = 1 to PC ← PC + rel

Struktura bajtów:

0 1 0 0	0 0 0 0

Rel

Bajty: 2
Cykle: 2
Przykład: JC MNIEJSZE
JC 127

Mnemonik: JMP (Jump indirect)

JMP @A+DPTR (ang.*jump indirect*)
skocz pośrednio

Do licznika rozkazów jest wpisywana suma zawartości 16-bitowego rejestru DPRT i akumulatora. Wykonywany jest skok pod adres umieszczony w DPTR z przesunięciem zapisanym w akumulatorze. Zawartość akumulatora jest traktowana jako liczba dwójkowa bez znaku (z zakresu 0-255). Dodawanie jest wykonywane mod 2^{16} tzn. że w przypadku kiedy suma przekroczy 2^{16}=65536 skok zostanie wykonany pod adres (A+DPTR)-2^{16}.

Operacja: PC ← A + DPTR

Struktura bajtów:

0 1 1 1	0 0 1 1

Bajty: 1

Cykle: 2

Mnemonik: JNB (Jump if Bit Not set)

JNB bit,rel (ang. *jump if bit is not set*)
skocz, jeśli bit jest zerowy

Jeżeli wartość bitu o podanym adresie bezpośrednim jest zerem, to do zawartości licznika rozkazów dodaje się przesunięcie <rel> (liczba ze znakiem w kodzie U2), tzn. jest wykonywany skok względny. Dzieje się to po pobraniu kodu rozkazu skoku, a zatem skok następuje względem adresu pierwszego bajtu rozkazu następnego po JNB. Testowany bit nie ulega zmianie.

Jeżeli warunek nie zostanie spełniony wykonywana jest instrukcja zapisana jako następna po JNB.

Operacja: PC ← PC + 3

jeśli bit = 0 to PC ← PC + rel

Struktura bajtów:

0 0 1 1	0 0 0 0

adres bitu

Rel

Bajty: 3

Cykle: 2

Przykład: JNB 8,10
JNB ACC.7,KLAW
JNB 20H.0,FLAGA

Mnemonik: JNC (Jump if Carry Not set)

JNC rel (ang. *jump if carry is not set*)
skocz, jeśli nie ma przeniesienia

Jezeli nie jest ustawiony znacznik przeniesienia (CY = 0), to do zawartości licznika rozkazów dodaje się przesunięcie <rel> (liczba ze znakiem w kodzie U2), tzn. jest wykonywany skok względny. Dzieje się to po pobraniu kodu rozkazu skoku, a zatem skok następuje względem adresu pierwszego bajtu rozkazu następnego po JNC.

Jeżeli warunek nie zostanie spełniony wykonywana jest instrukcja zapisana jako następna po JNC.

Operacja: PC ← PC + 2
jeśli C = 0 to PC ← PC + rel

Struktura bajtów:

0 1 0 1	0 0 0 0

Rel

Bajty: 2
Cykle: 2
Przykład: JC WIEKSZE
JC 127

Mnemonik: JNZ (Jump if Accumulator Not Zero)

JNZ rel (ang. *jump if accumulator not zero*)
skocz jeśli akumulator nie jest zerowy

Jeżeli zawartość akumulatora nie jest zerowa, to do zawartości licznika rozkazów dodaje się przesunięcie <rel> (liczba z znakiem w kodzie U2)tzn. jest wykonywany kod względny. Dzieje się to po pobraniu kodu rozkazu skoku a zatem skok następuje względem adresu pierwszego bajtu rozkazu następnego po JNZ.

Jeżeli warunek nie zostanie spełniony wykonywana jest instrukcja zapisana jako następna po JNZ.

Operacja: PC ← PC + 2
jeśli A < > 0 to PC ← PC + rel

Struktura bajtów:

0 1 1 1	0 0 0 0

Rel

Bajty: 2
Cykle: 2
Przykład: JNZ 100
JNZ NACISNIETY

Mnemonik: JZ (Jump if Accumulator Zero)

JZ rel (ang. *jump if accumulator zero*)
skocz, jeśli akumulator jest zerowy

Jeżeli zawartość akumulatora nie jest zerowa, to do zawartości licznika rozkazów dodaje się przesunięcie <rel> (liczba z znakiem w kodzie U2)tzn. jest

wykonywany kod względny. Dzieje się to po pobraniu kodu rozkazu skoku, a zatem skok następuje względem adresu pierwszego bajtu rozkazu następnego po JZ.

Operacja: PC ← PC + 2
jeśli A = 0 to PC ← PC + rel

Struktura bajtów:

0 1 1 0	0 0 0 0

Rel

Bajty: 2
Cykle: 2
Przykład: JZ 100
JZ CZEKAJ_KLAW

Mnemonik: LCALL (Long call)

LCALL adr_16 (ang. *subroutine call*)
skocz do podprogramu

Zawartość licznika rozkazów zwiększona o 3 przy pobraniu rozkazu jest ładowana na stos (pierwszy bajt mniej znaczący), zawartość wskaźnika stosu jest zwiększana o 2, a do licznika rozkazów jest wpisywany 16-bitowy adres pośredni. Rozkaz ten pozwala na skok do podprogramu, pod dowolny adres w pamięci programu.

Operacja: PC ← PC + 3
SP ← SP + 1 (SP) ← PC_{7-0}
SP ← SP + 1 (SP) ← PC_{15-8}
PC ← adr_16

Struktura bajtów:

0 0 0 1	0 0 1 0

Adr $_{15-8}$	adr $_{7-0}$

Bajty: 2
Cykle: 2
Przykład: LCALL 0FF0AH
LCALL OBSLUGA

Mnemonik: LJMP (Long Jump)

LJMP adr_16 (ang. *unconditional jump*)
skocz bezwarunkowo

Do licznika rozkazów jest wpisywany 16-bitowy adres bezpośredni. Rozkaz ten pozwala na skok pod dowolny adres w całej przestrzeni adresowej pamięci programu.

Operacja: PC ← adr_16

Struktura bajtów:

0 0 0 0	0 0 1 0

adr $_{15-8}$

adr $_{7-0}$

Bajty: 3
Cykle: 2
Przykład: LJMP 0AF00H
LJMP START

Mnemonik: MOV (Move byte variable)

MOV <dest_byte>, <src_byte> (ang. *move data*)
prześlij dane

Ośmiobitowe dane z miejsca wskazanego przez drugi argument <src> są kopiowane i przesyłane do miejsca wskazanego przez pierwszy argument <dest>. Jest możliwych piętnaście trybów adresowania danych.

MOV A,Rn – do akumulatora kopiuje daną z rejestru pomocniczego aktualnie wybranego banku rejestrów
Operacja: A ← Rn
Struktura bajtów:

1 1 1 0	1 r r r

Bajty: 1
Cykle: 1
Przykład: MOV A,R3

MOV A,adr - do akumulatora kopiuje daną z komórki wewnętrznej pamięci RAM o adresie bezpośrednio wskazanym w instrukcji
Operacja: A ← (adr)
Struktura bajtów:

1 1 1 0	0 1 0 1

Adr

Bajty: 2
Cykle: 1
Przykład: MOV A,90
MOV A,MNOZNA

MOV A,@Ri – do akumulatora kopiuje zawartość komórki wewnętrznej pamięci RAM o adresie wskazanym pośrednio jako dana w rejestrze Ri (R0 lub R1)
Uwaga: Jest to jedyny możliwy do zastosowania tryb adresowania w przypadku kiedy dana źródłowa umieszczona jest w obszarze pamięci IRAM o adresie w zakresie 128-255.
Operacja: A ← (Ri)
Struktura bajtów:

1 1 1 0	0 1 1 i

Bajty: 1
Cykle: 1
Przykład: MOV A,@R1

MOV A,#dana – do akumulatora kopiuje 8-bitową daną bezpośrednio wskazaną w instrukcji
Operacja: A ← dana
Struktura bajtów:

0 1 1 1	0 1 0 0

Dana

Bajty: 2
Cykle: 1
Przykład: MOV A,#10
MOV A,#CZYNNIK

MOV Rn,A - do rejestru pomocniczego aktualnie wybranego banku rejestrów kopiuje daną z akumulatora
Operacja: Rn ← A
Struktura bajtów:

1 1 1 1	1 r r r

Bajty: 1
Cykle: 1
Przykład: MOV R4,A

MOV Rn,adr - do rejestru pomocniczego aktualnie wybranego banku rejestrów kopiuje daną z komórki wewnętrznej pamięci RAM o adresie bezpośrednio wskazanym w instrukcji.
Operacja: Rn ← (adr)
Struktura bajtów:

1 0 1 0	1 r r r

Adr

Bajty: 2
Cykle: 1
Przykład: MOV R7,20
MOV R6,SETKI

MOV Rn,#dana - do rejestru pomocniczego aktualnie wybranego banku rejestrów kopiuje 8-bitową daną bezpośrednio wskazaną w instrukcji
Operacja: Rn ← dana
Struktura bajtów:

0 1 1 1	1 r r r

Dana

Bajty : 2

Cykle: 1
Przykład: MOV R5,#100
MOV R3,#SET_L

MOV adr,A - do komórki wewnętrznej pamięci RAM o adresie bezpośrednio wskazanym w instrukcji kopiuje daną z akumulatora
Operacja: (adr) ← A
Struktura bajtów:

1 1 1 1	0 1 0 1

Adr

Bajty: 2
Cykle: 1
Przykład: MOV 30,A
MOV STAN,A

MOV adr,Rn - do komórki wewnętrznej pamięci RAM o adresie bezpośrednio wskazanym w instrukcji kopiuje daną z rejestru pomocniczego aktualnie wybranego banku rejestrów
Operacja: (adr) ← Rn
Struktura bajtów:

1 0 0 0	1 r r r

Adr

Bajty: 2
Cykle: 2
Przykład: MOV 34,R6
MOV LOSOWA,R3

MOV adr,adr1 - do komórki wewnętrznej pamięci RAM o adresie <adr> bezpośrednio wskazanym w instrukcji kopiuje daną z komórki wewnętrznej pamięci RAM o adresie <adr1> bezpośrednio wskazanym w instrukcji.
Operacja: (adr) ← (adr1)
Struktura bajtów:

1 0 0 0	0 1 0 1

adr1

adr

Bajty: 3
Cykle: 2
Przykład: MOV 30,127
MOV TEMP,STALA

MOV adr,@Ri - do komórki wewnętrznej pamięci RAM o adresie <adr> bezpośrednio wskazanym w instrukcji kopiuje daną z komórki wewnętrznej pamięci RAM o adresie wskazanym pośrednio jako dana w rejestrze Ri (R0 lub R1)
Uwaga: Jest to jedyny możliwy do zastosowania tryb adresowania w przypadku kiedy dana źródłowa umieszczona jest w obszarze pamięci IRAM o adresie <adr> w zakresie 128-255.
Operacja: (adr) ← (Ri)
Struktura bajtów:

1 0 0 0	0 1 1 i

adr

Bajty: 2
Cykle: 2
Przykład: MOV 20H,@R1
MOV TEMP,@R0

MOV adr,#dana - do komórki wewnętrznej pamięci RAM o adresie <adr> bezpośrednio wskazanym w instrukcji kopiuje 8-bitową daną bezpośrednio wskazaną w instrukcji
Operacja: (adr) ← dana
Struktura bajtów:

0 1 1 1	0 1 0 1

adr

dana

Bajty: 3
Cykle: 2
Przykład: MOV 40,#40
MOV SEKUNDNIK,#IL_SEK

MOV @Ri,A – do komórki wewnętrznej pamięci RAM o adresie wskazanym pośrednio jako dana w rejestrze Ri (R0 lub R1) kopiuje daną z akumulatora
Uwaga: Jest to jedyny możliwy do zastosowania tryb adresowania w przypadku kiedy dana ma być umieszczona w obszarze pamięci IRAM o adresie <adr> w zakresie 128-255.
Operacja: (Ri) ← A
Struktura bajtów:

1 1 1 1	0 1 1 i

Bajty: 1
Cykle: 1

MOV @Ri,adr - do komórki wewnętrznej pamięci RAM o adresie wskazanym pośrednio jako dana w rejestrze Ri (R0 lub R1) kopiuje daną z komórki wewnętrznej pamięci RAM o adresie bezpośrednio wskazanym w instrukcji.

Uwaga: Jest to jedyny możliwy do zastosowania tryb adresowania w przypadku kiedy dana ma być umieszczona w obszarze pamięci IRAM o adresie w zakresie 128-255.

Operacja: (Ri) ← (adr)

Struktura bajtów:

1 0 1 0	0 1 1 i

adr

Bajty : 2

Cykle: 2

Przykład: MOV @R0,50
MOV @R1,BUFOR

MOV @Ri,#dana - do komórki wewnętrznej pamięci RAM o adresie wskazanym pośrednio jako dana w rejestrze Ri (R0 lub R1) kopiuje 8-bitową daną bezpośrednio wskazaną w instrukcji.

Uwaga: Jest to jedyny możliwy do zastosowania tryb adresowania w przypadku kiedy dana ma być umieszczona w obszarze pamięci IRAM o adresie w zakresie 128-255.

Operacja: (Ri) ← dana

Struktura bajtów:

0 1 1 1	0 1 1 i

dana

Bajty: 2

Cykle: 1

Przykład: MOV @R0,#200
MOV @R1,#CONST

Mnemonik: MOV <dest_bit>,<src_bit> (Move bit data)

MOV bit,C (ang. *move carry to direct bit*)
prześlij znacznik CY do bitu

Zawartość znacznika przeniesienia (akumulatora procesora boolowskiego) jest przesyłana do bitu o podanym adresie bezpośrednim.

Operacja: C ← bit

Struktura bajtów:

1 0 1 0	0 0 1 0

adres bitu

Bajty: 2

Cykle: 2

Przykład: MOV ACC.0,C
MOV 0,C

MOV C,bit (ang. *move direct bit to carry*)
prześlij bit do znacznika CY

Zawartość bitu o podanym adresie bezpośrednim jest przesyłana do znacznika przeniesienia CY (akumulatora procesora boolowskiego).

Operacja: bit ← C

Struktura bajtów:

1 0 0 1	0 0 1 0

adres bitu

Bajty: 2

Cykle: 2

Przykład: MOV C,ACC.0
MOV C,0

MOV DPTR,#dana16 (ang. *load data pointer with 16-bit constant*)
wpisz 16-bitową stała do wskaźnika danych

Do 16-bitowego rejestru DPRT (DPH.DPL) jest wpisywany argument bezpośredni, podany jako drugi i trzeci bajt rozkazu. Pierwszy bajt (bardziej znaczący) jest wpisywany do rejestru specjalnego DPH, a drugi (mniej znaczący) do DPL.

Operacja: DPTR ← dana_16

Struktura bajtów:

1 0 0 1	0 0 0 0

dana $_{15-8}$

dana $_{7-0}$

Bajty: 3

Cykle: 2

Przykład: MOV DPTR,#60000
MOV DPTR,#0FF21H
MOV DPTR,#ADRES_LCD

Mnemonik: MOVC (Move Code byte)

MOVC A, @A + <base_reg>
(ang. *move data from progrm memory to accumulator*)
prześlij bajt z pamięci programu do akumulatora

Zawartość komórki pamięci programu o adresie będącym sumą zawartości akumulatora (8-bitowa liczba dwójkowa bez znaku) i zawartości 16-bitowego rejestru bazowego <base_reg> jest przesyłana do akumulatora. Są tu możliwe dwa rejestry bazowe: wskaźnik danych DPTR i licznik rozkazów PC. Jeżeli rejestrem bazowym

jest licznik rozkazów, to adresem bazowym jest jego zawartość po pobraniu rozkazu, czyli adres pierwszego bajtu rozkazu następnego po MOVC.

MOVC A,@A+DPTR – do akumulatora kopiuje daną z komórki pamięci programu (CODE) a adresie wyznaczonym jako suma danych zawartych w akumulatorze przed wykonaniem instrukcji oraz zawartej w rejestrze DPTR
Operacja: $A \leftarrow (A + DPTR)_{CODE}$
Struktura bajtów:

1 0 0 1	0 0 1 1

Bajty: 1
Cykle: 2

MOVC A,@A+PC - do akumulatora kopiuje daną z komórki pamięci programu (CODE) a adresie wyznaczonym jako suma danych zawartych w akumulatorze przed wykonaniem instrukcji oraz stanu licznika rozkazów PC po pobraniu rozkazu, czyli adresu pierwszego bajtu rozkazu następnego po MOVC.
Operacja: $PC \leftarrow PC + 1$
$A \leftarrow (A + PC)_{CODE}$
Struktura bajtów:

1 0 0 0	0 0 1 1

Bajty: 1
Cykle: 2

Mnemonik: MOVX (Move External)

MOVX A, @<dp>
(ang. *move data from external data memory to accumulator*)
przeslij bajt z zewnętrznej pamięci danych do akumulatora

Dane z komórki zewnętrznej pamięci danych o adresie pośrednim zawartym we wskaźniku danych <dp> są wpisywane do akumulatora. Wskaźnikiem danych może być rejestr roboczy R0 lub R1 albo rejestr specjalny DPTR. Jeżeli wskaźnikiem danych jest rejestr R0 lub R1, to 8-bitowy adres jest wysyłany tylko poprzez port P0 (UWAGA! Przy wykorzystywaniu pełnej magistrali adresowej starszy bajt adresu odpowiada wówczas stanowi portu P2 w momencie wykonywania instrukcji).

Jeżeli wskaźnikiem danych jest rejestr DPTR , to 16-bitowy adres jest wysyłany prze port P0 (z DPL - osiem mniej znaczących bitów) i P2 (z DPH - osiem bardziej znaczących bitów). W obu przypadkach na wyjściu “not RD” (P3.7)(/RD) jest generowany impuls będący sygnałem sterującym odczytywania.

Jako zewnętrzna pamięć RAM uważane są urządzenia zewnętrzne np. klawiatura, LCD itp. Obsługiwane przez system poprzez multipleksowaną magistralę danych i adresową. Jest to wówczas jedyny możliwy tryb adresowania umożliwiający zapis i odczyt danych z tych urządzeń.

MOVX A,@Ri – do akumulatora kopiuje daną z komórki zewnętrznej pamięci RAM o młodszym bajcie adresu wskazywanym pośrednio przez daną zawartą w rejestrze Ri (R0 lub R1) aktualnie wybranego banku rejestrów

Operacja: $A \leftarrow (256 * P2 + Ri)_{XDATA}$
Struktura bajtów:

1 1 1 0	0 0 1 i

Bajty: 1
Cykle: 2

MOVX A,@DPTR - do akumulatora kopiuje daną z komórki zewnętrznej pamięci RAM o adresie wskazywanym pośrednio przez daną zawartą w 16-bitowym rejestrze DPTR
Operacja: $A \leftarrow (DPTR)_{XDATA}$
Struktura bajtów:

1 1 1 0	0 0 0 0

Bajty: 1
Cykle: 2

MOVX @<dp>, A

(ang. *move data from accumulator to external data memory*)
prześlij bajt z akumulatora do zewnętrznej pamięci danych

Do komórki zewnętrznej pamięci danych o adresie pośrednim zawartym we wskaźniku danych <dp> są wpisywane dane z akumulatora. Wskaźnikiem danych może być rejestr roboczy R0 lub R1 albo rejestr specjalny DPTR. Jeżeli wskaźnikiem danych jest rejestr R0 lub R1, to 8-bitowy adres jest wysyłany tylko poprzez port P0 (UWAGA! Przy wykorzystywaniu pełnej magistrali adresowej starszy bajt adresu odpowiada wówczas stanowi portu P2 w momencie wykonywania instrukcji).

. Jeżeli natomiast wskaźnikiem danych jest rejestr DPTR, to 16-bitowy adres jest wysyłany prze port P0 (z DPL - osiem mniej znaczących bitów) i P2 (z DPH - osiem bardziej znaczących bitów). W obu przypadkach na wyjściu “not WR” (P3.6)(/WR) jest generowany impuls będący sygnałem sterującym zapisywania.

MOVX @Ri,A - do komórki zewnętrznej pamięci RAM o młodszym bajcie adresu wskazywanym pośrednio przez daną zawartą w rejestrze Ri (R0 lub R1) aktualnie wybranego banku rejestrów kopiuje daną z akumulatora

Operacja: $(256 * P2 + Ri)_{XDATA} \leftarrow A$
Struktura bajtów:

1 1 1 1	0 0 1 i

Bajty: 1

Cykle: 2

MOVX @DPTR,A - do komórki zewnętrznej pamięci RAM o adresie wskazywanym pośrednio przez daną zawartą w 16-bitowym rejestrze DPTR kopiuje daną z akumulatora.
Operacja: $(DPTR)_{XDATA} \leftarrow A$
Struktura bajtów:

1 1 1 1	0 0 0 0

Bajty: 1
Cykle: 2

Mnemonik: MUL (Multiply)

MUL AB (ang. *multiply*)
pomnóż

Ośmiobitowa liczba bez znaku, znajdująca się w akumulatorze, jest mnożona przez 8-bitową liczbę bez znaku z rejestru B. Szesnastobitowy wynik jest wpisywany w następujący sposób: osiem bardziej znaczących bitów do rejestru B, osiem mniej znaczących bitów do akumulatora. Jeżeli wynik mnożenia jest większy niż 255, to jest ustawiany znacznik OV; w przeciwnym razie OV jest zerowany. Znacznik CY jest zerowany.
Operacja: BA ← A * B OV = 1 jeśli iloczyn > 255
Struktura bajtów:

1 0 1 0	0 1 0 0

Bajty: 1
Cykle: 4

Mnemonik: NOP (No Operation)

NOP (ang. *no operation*)
nic nie rób

Nie jest wykonywana żadna operacja (a czas płynie).
Operacja: brak działania
Struktura bajtów:

0 0 0 0	0 0 0 0

Bajty: 1
Cykle: 1

Mnemonik: ORL (Logical-OR for byte variables)

ORL <dest_byte>, <src_byte> (ang. *logical OR*)
sumuj logicznie

Wykonywana jest suma logiczna (bit po bicie) wskazanych argumentów. Wynik jest wpisywany do miejsca, z którego został pobrany argument <dest>. Jest możliwych sześć kombinacji trybów adresowania argumentów.

Uwaga! Jeżeli danego rozkazu użyto do zmiany stanu wyjścia, to zostaje odczytana i zmodyfikowana zawartość rejestru wyjściowego portu (a nie stan logiczny z końcówek układu).

ORL A,Rn – wykonuje sumę logiczną danych zawartych w akumulatorze i rejestrze pomocniczym aktualnie wybranego banku rejestrów. Wynik umieszcza w akumulatorze.
Operacja: A ← A or Rn
Struktura bajtów:

0 1 0 0	1 r r r

Bajty: 1
Cykle: 1
Przykład: ORL A,R6

ORL A,adr - wykonuje sumę logiczną danych zawartych w akumulatorze i komórce wewnętrznej pamięci RAM o adresie podanym bezpośrednio w instrukcji. Wynik umieszcza w akumulatorze.
Operacja: A ← A or (adr)
Struktura bajtów:

0 1 0 0	0 1 0 1

adr

Bajty: 2
Cykle: 1
Przykład: ORL A,120
ORL A,MASKA_BIT

ORL A,@Ri - wykonuje sumę logiczną danych zawartych w akumulatorze i komórce wewnętrznej pamięci RAM o adresie podanym pośrednio jako dana w rejestrze Ri (R0 lub R1) aktualnie wybranego banku rejestrów. Wynik umieszcza w akumulatorze.
Operacja: A ← A or (Ri)
Struktura bajtów:

0 1 0 0	0 1 1 i

Bajty: 1
Cykle: 1

ORL A,#dana - wykonuje sumę logiczną danej zawartej w akumulatorze i danej zapisanej bezpośrednio w instrukcji. Wynik umieszcza w akumulatorze.
Operacja: A ← A or dana

Struktura bajtów:

0 1 0 0	0 1 0 0

dana

Bajty: 2
Cykle: 1
Przykład: ORL A,#00110100B
ORL A,#MASKA

ORL adr,A - wykonuje sumę logiczną danych zawartych w akumulatorze i komórce wewnętrznej pamięci RAM o adresie podanym bezpośrednio w instrukcji. Wynik umieszcza w komórce wewnętrznej pamięci RAM o adresie podanym bezpośrednio w instrukcji.
Operacja: (adr) ← (adr) or A
Struktura bajtów:

0 1 0 0	0 0 1 0

adr

Bajty: 2
Cykle: 1
Przykład: ORL 100,A
ORL SET_RS,A

ORL adr,#dana - wykonuje sumę logiczną danej zawartej w komórce wewnętrznej pamięci RAM o adresie podanym bezpośrednio w instrukcji i danej zapisanej bezpośrednio w instrukcji. Wynik umieszcza w komórce wewnętrznej pamięci RAM o adresie podanym bezpośrednio w instrukcji.
Operacja: (adr) ← (adr) or dana
Struktura bajtów: :

0 1 0 0	0 0 1 1

adr

dana

Bajty: 3
Cykle: 2
Przykład: ORL 30H,#00001111B
ORL WSKAZ,#MASKA

Mnemonik: ORL (Logical-OR for bit variables)

ORL C,bit (ang. *logical OR carry with direct bit)*
bit sumuj logicznie z bitem

Jeżeli wartość logiczna bitu o podanym adresie bezpośrednim jest równa 1, to do znacznika przeniesienia CY (akumulator procesora boolowskiego) wpisuje się jedynka. W przeciwnym razie zawartość CY nie zmienia się (zawartość CY jest sumowana logicznie z zawartością zaadresowanego bitu).

Operacja: C ← C or bit
Struktura bajtów:

1 1 1 1	0 0 1 0

adres bitu

Bajty: 2
Cykle: 2
Przykład: ORL C,20
ORL C,21H.3
ORL C,TI

ORL C,/bit

(ang. *logical OR carry with complement of direct bit*)
sumuj logicznie z negacją bitu

Jeżeli wartość logiczna bitu o podanym adresie bezpośrednim jest równa 0, to do znacznika przeniesienia CY (akumulator procesora boolowskiego) wpisuje się jedynka. W przeciwnym razie zawartość CY nie zmienia się (zawartość CY jest sumowana logicznie z zawartością zaadresowanego bitu).
Operacja: C ← C or (not bit)
Struktura bajtów: :

1 0 1 0	0 0 0 0

adres bitu

Bajty: 2
Cykle: 2
Przykład: ORL C,/20
ORL C,/21H.3
ORL C,/TI

Mnemonik: POP (Pop from stack)

POP adr (ang. *pop from stack*)
zdejmij ze stosu

Dane z wierzchołka stosu, tzn. komórki wewnętrznej pamięci danych o adresie zawartym we wskaźniku stosu SP, są wpisywane do komórki wewnętrznej pamięci danych (lub rejestru specjalnego) o podanym adresie bezpośrednim, po czym zawartość SP jest zmniejszana o 1.
Operacja: (adr) ← (SP)
SP ← SP – 1
Struktura bajtów:

1 1 0 1	0 0 0 0

Adr

Bajty: 2
Cykle: 2
Przykład: POP 20H

POP ACC

Mnemonik: PUSH (Push onto stack)

PUSH adr (ang. *push onto stack*)
ładuj na stos

Zawartość wskaźnka stosu SP jest zwiększana o 1, po czym na wierzchołek stosu, czyli do komórki wewnętrznej pamięci danych o adresie zawartym w SP, jest wpisywana zawartość komórki wewnętrznej pamięci danych (lub rejestru specjalnego) o podanym adresie bezpośrednim.

Operacja: SP ← SP + 1
(SP) ← (adr)

Struktura bajtów:

1 1 0 0	0 0 0 0

Adr

Bajty: 2
Cykle: 2
Przykład: POP 20H
POP ACC

UWAGA! W przypadku instrukcji POP oraz PUSH nie można stosować nazw symbolicznych rejestrów pomocniczych banków rejestrów (należy stosować bezpośrednie adresy tych rejestrów jako komórek pamięci RAM) natomiast dla akumulatora należy używać nazwy ACC.

Mnemonik: RET (Return from subroutine)

RET ***PODPROGRAM*** (ang. *return from subroutine*)
wróć z podprogramu

Adres powrotu (najpierw bardziej, następnie mniej znaczący bajt) jest wpisywany ze stosu do licznika rozkazów. Zawartość wskaźnika stosu jest zmniejszana o 2.

Operacja: $PC_{15-8} \leftarrow (SP)$ $SP \leftarrow SP - 1$
$PC_{7-0} \leftarrow (SP)$ $SP \leftarrow SP - 1$

Struktura bajtów:

0 0 1 0	0 0 1 0

Bajty: 1
Cykle: 2

Mnemonik: RETI (Return from interrupt)

RETI (ang. *return from interrupt*)
wróć z przerwania

Adres powrotu (najpierw bardziej, następnie mniej znaczący bajt) jest wpisywany ze stosu do licznika rozkazów. Zawartość wskaźnika stosu jest zmniejszana o 2. Wykonanie rozkazu RETI jest dla systemu przerwań sygnałem zakończenia obsługi przerwania, czyli żadne zgłoszenie przerwania z tego samego lub niższego niż obsługiwane poziomu nie będzie wcześniej przyjęte.

Operacja: $PC_{15-8} \leftarrow (SP)$ $SP \leftarrow SP - 1$
$PC_{7-0} \leftarrow (SP)$ $SP \leftarrow SP - 1$

Struktura bajtów:

0 0 1 1	0 0 1 0

Bajty: 1
Cykle: 2

Mnemonik: RL (Rotate Accumulator Left)

RL A (ang. *rotate left*)
przesuń cyklicznie w lewo

Zawartość akumulatora jest przesuwana cyklicznie w lewo o jeden bit, tzn. bit 1 przyjmuje wartość 0, bit 2 - wartość bitu 1,..., bit 7 - wartość bitu 6, a bit 0 - wartość bitu 7.

Operacja: $(An+1) \leftarrow An$, $n = 0 - 6$
$A0 \leftarrow A7$

Struktura bajtów:

0 0 1 0	0 0 1 1

Bajty: 1
Cykle: 1

Mnemonik: RLC (Rotate Accumulator Left through the Carry flag)

RLC A (ang. *rotate left through carry*)
przesuń cyklicznie w lewo z CY

Zawartość akumulatora jest przesuwana cyklicznie w lewo o jeden bit, tzn. bit 1 przyjmuje wartość bitu 0, bit 2 – wartość bitu 1,..., bit 7 – wartość bitu 6, znacznik CY – wartość bitu 7, a bit 0 – wartość znacznika CY. Wykonanie tego rozkazu przy zerowym CY, np. w sekwencji:

```
CLR C
RLC A
```

odpowiada pomnożeniu przez 2 liczby (w naturalnym kodzie dwójkowym) zawartej w akumulatorze. Znacznik przeniesienia CY jest ustawiany zgodnie z wynikiem operacji. Inne znaczniki pozostają bez zmian.

Operacja: $(An+1) \leftarrow An$, $n = 0 - 6$

A0 ← C C ← A7

Struktura bajtów:

0 0 1 1	0 0 1 1

Bajty: 1

Cykle: 1

Mnemonik: RR (Rotate Accumulator Right)

RR A (ang. *rotate right*)
przesuń cyklicznie w prawo

Zawartość akumulatora jest cyklicznie przesuwana w prawo o jeden bit, tzn. bit 0 przyjmuje wartość bitu 1, bit 1 – wartość bitu 2,..., bit 6 – wartość bitu 7, a bit 7 – wartość bitu 0.

Operacja: An ← (An+1), n = 0 – 6
A7 ← A0

Struktura bajtów:

0 0 0 0	0 0 1 1

Bajty: 1

Cykle: 1

Mnemonik: RRC (Rotate Accumulator Right through the Carry flag)

RRC A (ang. r*otate right through carry*)
przesuń cyklicznie w prawo z CY

Zawartość akumulatora jest cyklicznie przesuwana w prawo o jeden bit, tzn. bit 0 przyjmuje wartość bitu 1, bit 1 – wartość bitu 2,..., bit 6 – wartość bitu 7, bit 7 – wartość znacznika CY, a znacznik CY – zawartość bitu 0. Wykonanie tego rozkazu przy zerowym CY, np. w sekwencji:

```
CLR C
RLC A
```

odpowiada podzieleniu przez 2 liczby (w naturalnym kodzie dwójkowym) zawartej w akumulatorze. Znacznik przeniesienia CY jest ustawiany zgodnie z wynikiem operacji. Inne znaczniki pozostają bez zmian.

Operacja: An ← (An+1), n = 0 – 6
A7 ← C C ← A0

Struktura bajtów:

0 0 0 1	0 0 1 1

Bajty: 1

Cykle: 1

Mnemonik: SETB (Set Bit)

SETB C (ang. s*et carry flag*)
ustaw znacznik przeniesienia

Do znacznika przeniesienia CY (akumulatora procesora boolowskiego) jest wpisywana jedynka.

Operacja: C ← 1

Struktura bajtów:

1 1 0 1	0 0 1 1

Bajty: 1

Cykle: 1

SETB bit (ang. *set bit*)
ustaw bit

Do bitu o podanym adresie bezpośrednio wpisywana jest jedynka.

Operacja: bit ← 1

Struktura bajtów:

1 1 0 1	0 0 1 0

adres bitu

Bajty: 2

Cykle: 1

Przykład: SETB 1
SETB 20H.1
SETB P3.5

Mnemonik: SJMP (Short Jump)

SJMP rel (ang. *jump relative*)
skocz bezwarunkowo względem PC

Od zawartości licznika rozkazów jest dodawane przesunięcie <rel> (8 – bitowa liczba ze znakiem w kodzie U2, z przedziału <-127, +127>). Dodawanie jest wykonywane po pobraniu kodu rozkazu skoku, kiedy w liczniku rozkazów znajduje się adres pierwszego bajtu następnego rozkazu. Skok jest wykonywany względem tego adresu.

Operacja: PC ← PC + 2
PC ← PC + rel

Struktura bajtów:

1 0 0 0	0 0 0 0

Rel

Bajty: 2

Cykle: 2

Przykład: SJMP 0F0H
SJMP WROC

Mnemonik: Subtract with borrow

SUBB A, <src_byte> (ang. *subtract from accumulator with borrow*)
odejmij od akumulatora z pożyczką

Od zawartości akumulatora jest odejmowany wskazany argument oraz zawartość znacznika przeniesienia CY. Wynik operacji jest wpisywany do akumulatora, - są ustawiane znaczniki CY, AC, i OV. Są możliwe cztery tryby adresowania argumentu <src>.

SUBB A,Rn - Od zawartości akumulator odejmowana jest dana zawarta w rejestrze pomocniczym aktualnie wybranego banku rejestrów oraz zawartość znacznika przeniesienia CY. Wynik operacji jest wpisywany do akumulatora.
Operacja: A ← A – C – Rn
Struktura bajtów:

1 0 0 1	1 r r r

Bajty: 1
Cykle: 1
Przykład: SUBB A,R2

SUBB A,adr - Od zawartości akumulator odejmowana jest dana zawarta w komórce wewnętrznej pamięci RAM o adresie bezpośrednio zapisanym w instrukcji oraz zawartość znacznika przeniesienia CY. Wynik operacji jest wpisywany do akumulatora.
Operacja: A ← A – C – (adr)
Struktura bajtów:

1 0 0 1	0 1 0 1

Adr

Bajty: 2
Cykle: 1
Przykład: SUBB A,45
SUBB A,ODJEMNIK

SUBB A,@Ri - Od zawartości akumulator odejmowana jest dana zawarta w komórce wewnętrznej pamięci RAM o adresie wskazanym przez 8-bitową daną zawartą w rejestrze pomocniczym Ri (R0 lub R1) aktualnie wybranego banku rejestrów oraz zawartość znacznika przeniesienia CY. Wynik operacji jest wpisywany do akumulatora.
Uwaga: Jest to jedyny możliwy do zastosowania tryb adresowania w przypadku kiedy dana źródłowa umieszczona jest w obszarze pamięci IRAM o adresie <adr> w zakresie 128-255.
Operacja: A ← A – C – (Ri)

Struktura bajtów:

1 0 0 1	0 1 1 i

Bajty: 1
Cykle: 1

SUBB A,#dana - Od zawartości akumulator odejmowana jest dana bezpośrednio zapisana w instrukcji oraz zawartość znacznika przeniesienia CY. Wynik operacji jest wpisywany do akumulatora.
Operacja: A ← A – C – dana
Struktura bajtów:

1 0 0 1	0 1 0 0

Dana

Bajty: 2
Cykle: 1
Przykład: SUBB A,#12
SUBB A,#ODJEMNIK

Mnemonik: SWAP (Swap nibbles within the Accumulator)

SWAP A (ang. *swap nibbles within accumulator*)
wymień półbajty (tetrady)w akumulatorze

Wymieniona zostaje zawartość bitów 0 – 3 (mniej znacząca tetrada) i bitów 4 – 7 (bardziej znacząca tetrada) akumulatora. Jest to równoważne czterokrotnemu cyklicznemu przesunięciu zawartości akumulatora.
Operacja: $A_{7-4} \Leftrightarrow A_{3-0}$
Struktura bajtów:

1 1 0 0	0 1 0 0

Bajty: 1
Cykle: 1

Mnemonik: XCH (Exchange Accumulator with byte variable)

XCH A, <src_byte> (ang. *exchange accumulator with memory contents*)
wymień akumulator z zawartością pamięci

Zawartość akumulatora jest wymieniana z zawartością wskazanego argumentu. Są możliwe trzy tryby adresowania argumentu <src>.

XCH A,Rn – zamienia zawartość akumulatora i rejestru pomocniczego aktualnie wybranego banku rejestrów.
Operacja: A ⇔ Rn
Struktura bajtów:

1 1 0 0	1 r r r

Bajty: 1
Cykle: 1
Przykład: XCH A,R6

XCH A,adr - zamienia zawartość akumulatora i komórki wewnętrznej pamięci RAM o adresie bezpośrednio zapisanym w instrukcji.
Operacja: A ⇔ (adr)
Struktura bajtów:

1 1 0 0	0 1 0 1

Adr

Bajty: 2
Cykle: 1
Przykład: XCH A,55
XCH A,BUFOR

XCH A,@Ri - zamienia zawartość akumulatora komórce wewnętrznej pamięci RAM o adresie wskazanym przez 8-bitową daną zawartą w rejestrze pomocniczym Ri (R0 lub R1) aktualnie wybranego banku rejestrów.
Operacja: A ⇔ (Ri)
Struktura bajtów:

1 1 0 0	0 1 1 i

Bajty: 1
Cykle: 1

Mnemonik: XCHD (Exchange Digit)

XCHD A,@Ri (ang. *exchange nibble of accumulator with memory*)
wymień półbajty (tetrady) z akumulatora i pamięci

Zawartość bajtów 0 – 3 (mniej znacząca tetrada) akumulatora zostaje wymieniona z zawartością bitów 0 – 3 komórki wewnętrznej pamięci danych o adresie zawartym w rejestrze R0 lub R1 z ustawionego w danej chwili zbioru. Bity 4 –7 akumulatora oraz komórki pamięci pozostają bez zmiany.
Operacja: $A_{3-0} \Leftrightarrow (Ri)_{3-0}$
Struktura bajtów:

1 1 0 1	0 1 1 i

Bajty: 1
Cykle: 1
Przykład: XCHD A,@R0

Mnemonik: XRL (Logical Exclusive-OR for byte variables)

XRL <dest_byte>, <src_byte> (ang. *logical XOR*)
sumuj mod 2

Wykonywana jest suma mod 2 (różnica symetryczna) wskazanych argumentów (bit po bicie). Wynik jest wpisywany do miejsca, z którego został obrany argument <dest>. Jest możliwych sześć kombinacji trybów adresowania argumentów <dest> i <src>.

XRL A,Rn - Wykonywana jest suma mod 2 (różnica symetryczna) na danych zawartych w akumulatorze i rejestrze pomocniczym Rn aktualnie wybranego banku rejestrów. Wynik umieszczany w akumulatorze.
Operacja: A ← A xor Rn
Struktura bajtów:

0 1 1 0	1 r r r

Bajty: 1
Cykle: 1
Przykład: XRL A,R5

XRL A,adr - Wykonywana jest suma mod 2 (różnica symetryczna) na danych zawartych w akumulatorze i komórce wewnętrznej pamięci RAM o adresie bezpośrednio zapisanym w instrukcji. Wynik umieszczany w akumulatorze.
Operacja: A ← A xor (adr)
Struktura bajtów:

0 1 1 0	0 1 0 1

Adr

Bajty: 2
Cykle: 1
Przykład: XRL A,70
XRL A,SBUF

XRL A,@Ri - Wykonywana jest suma mod 2 (różnica symetryczna) na danych zawartych w akumulatorze i komórce wewnętrznej pamięci RAM o adresie wskazanym przez 8-bitową daną zawartą w rejestrze pomocniczym Ri (R0 lub R1) aktualnie wybranego banku rejestrów. Wynik umieszczany w akumulatorze.
Operacja: A ← A xor (Ri)
Struktura bajtów:

0 1 1 0	0 1 1 i

Bajty: 1
Cykle: 1
Przykład: XRL A,@R1

XRL A,#dana - Wykonywana jest suma mod 2 (różnica symetryczna) na zawartości akumulatora i danej zapisanej bezpośrednio w instrukcji. Wynik umieszczany w akumulatorze.

Operacja: A ← A xor dana

Struktura bajtów:

0 1 1 0	0 1 0 0

Dana

Bajty: 2

Cykle: 1

Przykład: XRL A,#10101100B
XRL A,#MASKA_BIT

XRL adr,A - Wykonywana jest suma mod 2 (różnica symetryczna) na danych zawartych w akumulatorze i komórce wewnętrznej pamięci RAM o adresie bezpośrednio zapisanym w instrukcji. Wynik umieszczany w komórce wewnętrznej pamięci RAM o adresie bezpośrednio zapisanym w instrukcji.

Operacja: (adr) ← (adr) xor A

Struktura bajtów:

0 1 1 0	0 0 1 0

Adr

Bajty: 2

Cykle: 1

Przykład: XRL 70,A
XRL SBUF,A

XRL adr,#dana - wykonywana jest suma mod 2 (różnica symetryczna) na zawartości komórki wewnętrznej pamięci RAM o adresie bezpośrednio zapisanym w instrukcji i danej zapisanej bezpośrednio w instrukcji. Wynik umieszczany w komórce wewnętrznej pamięci RAM o adresie bezpośrednio zapisanym w instrukcji.

Operacja: (adr) ← (adr) xor dana

Struktura bajtów:

0 1 1 0	0 0 1 1

Dana

Bajty: 3

Cykle: 2

Przykład: XRL 40H,#10101100B
XRL SBUF,#MASKA_BIT

Uwaga! Jeżeli danego rozkazu użyto do zmiany stanu wyjścia, to zostaje odczytana i zmodyfikowana zawartość rejestru wyjściowego portu (a nie stan logiczny z końcówek układu).

2.2 Skrócona lista rozkazów z podziałem funkcjonalnym.

Instrukcje wymiany danych						
Mnemonik	**Operacja**	**Struktura bajtów**			**Cykle**	**Znaczniki**
MOV A,Rn	A ← Rn	1110	1 r r r		1	P
MOV A,adr	A ← (adr)	1110	0101	[adr]	1	P
MOV A,@Ri	A ← (Ri)	1110	011 i		1	P
MOV A,#dana	A ← dana	0111	0100	[dana]	1	P
MOV Rn,A	Rn ← A	1111	1 r r r		1	-
MOV Rn,adr	Rn ← (adr)	1010	1 r r r	[adr]	1	-
MOV Rn,#dana	Rn ← dana	0111	1 r r r	[dana]	1	-
MOV adr,A	(adr) ← A	1111	0101	[adr]	1	-
MOV adr,Rn	(adr) ← Rn	1000	1 r r r	[adr]	2	-
MOV adr,adr1	(adr) ← (adr1)	1000	0101	[adr1] [adr]	2	-
MOV adr,@Ri	(adr) ← (Ri)	1000	011 i	[adr]	2	-
MOV adr,#dana	(adr) ← dana	0111	0101	[adr] [dana]	2	-
MOV @Ri,A	(Ri) ← A	1111	011 i		1	-
MOV @Ri,adr	(Ri) ← (adr)	1010	011 i	[adr]	2	-
MOV @Ri,#dana	(Ri) ← dana	0111	011 i	[dana]	1	-
MOV DPTR,#dana16	DPTR ← dana_16	1001	0000	$[dana_{15-8}]$ $[dana_{7-0}]$	2	-
MOVC A,@A+DPTR	$A \leftarrow (A + DPTR)_{CODE}$	1001	0011		2	P
MOVC A,@A+PC	$A \leftarrow (A + PC)_{CODE}$	1000	0011		2	P
MOVX A,@Ri	$A \leftarrow (256 * P2 + Ri)_{XDATA}$	1110	001 i		2	P
MOVX A,@DPTR	$A \leftarrow (DPTR)_{XDATA}$	1110	0000		2	P
MOVX @Ri,A	$(256 * P2 + Ri)_{XDATA} \leftarrow A$	1111	001 i		2	-
MOVX @DPTR,A	$(DPTR)_{XDATA} \leftarrow A$	1111	0000		2	-
POP adr	(adr) ← (SP)	1101	0000	[adr]	2	-
PUSH adr	SP ← SP + 1 (SP) ← (adr)	1100	0000	[adr]	2	- -
XCH A,Rn	A ← Rn	1100	1 r r r		1	P
XCH A,adr	A ← (adr)	1100	0101	[adr]	1	P
XCH A,@Ri	A ← (Ri)	1100	011 i		1	P
XCHD A,@Ri	$A_{3-0} \leftarrow (Ri)_{3-0}$	1101	011 i		1	P
NOP	brak działania	0000	0000		1	-

Instrukcje manipulacji bitowych					
Mnemonik	**Operacja**	**Struktura bajtów**		**Cykle**	**Znaczniki**
CLR C	C ← 0	1100 0011		1	C
CLR bit	bit ← 0	1100 0010	[adr bitu]	1	C
SETB C	C ← 1	1101 0011		1	C
SETB bit	bit ← 1	1101 0010	[adr bitu]	1	-
CPL C	C ← not C	1011 0011		1	C
CPL bit	bit ← not bit	1011 0010	[adr bitu]	1	-
ANL C,bit	C ← C and bit	1000 0010	[adr bitu]	2	C
ANL C,/bit	C ← C and (not bit)	1011 0000	[adr bitu]	2	C
ORL C,bit	C ← C or bit	1111 0010	[adr bitu]	2	C
ORL C,/bit	C ← C or (not bit)	1010 0000	[adr bitu]	2	C
MOV C,bit	C ← bit	1010 0010	[adr bitu]	2	C
MOV bit,C	bit ← C	1001 0010	[adr bitu]	2	-

Instrukcje arytmetyczne					
Mnemonik	**Operacja**	**Struktura bajtów**		**Cykle**	**Znaczniki**
ADD A,Rn	A ← A + Rn	0010 1 r r r		1	C
ADD A,adr	A ← A + (adr)	0010 0101	[adr]	1	AC
ADD A,@Ri	A ← A + (Ri)	0010 011i		1	OV
ADD A,#dana	A ← A + dana	0010 0100	[dana]	1	P
ADDC A,Rn	A ← A + C + Rn	0011 1 r r r		1	C
ADDC A,adr	A ← A + C + (adr)	0011 0101	[adr]	1	AC
ADDC A,@Ri	A ← A + C + (Ri)	0011 011 i		1	OV
ADDC A,#dana	A ← A + C + dana	0011 0100	[dana]	1	P
INC A	A ← A + 1	0000 0100		1	P
INC Rn	Rn ← Rn + 1	0000 1 r r r		1	-
INC (adr)	(adr) ← (adr) + 1	0000 0101	[adr]	1	-
INC @Ri	(Ri) ← (Ri) + 1	0000 011 i		1	-
INC DPTR	DPTR ← DPTR + 1	1010 0011		2	-
SUBB A,Rn	A ← A – C – Rn	1001 1 r r r		1	C
SUBB A,adr	A ← A – C – (adr)	1001 0101	[adr]	1	AC
SUBB A,@Ri	A ← A – C – (Ri)	1001 011 i		1	OV
SUBB A,#dana	A ← A – C – dana	1001 0100	[dana]	1	P
DEC A	A ← A – 1	0001 0100		1	P
DEC Rn	Rn ← Rn – 1	0001 1 r r r		1	-
DEC (adr)	(adr) ← (adr) – 1	0001 0101	[adr]	1	-
DEC @Ri	(Ri) ← (Ri) – 1	0001 011 i		1	
DIV AB	A ← Int (A/B), iloraz OV = 1 jeśli przed B ← Mod (A/B), reszta dzieleniem B = 0	1000 0100		4	C = 0 OV, P
MUL AB	BA ← A * B OV = 1 jeśli iloczyn > 255	1010 0100		4	C = 0 OV, P
DA A	jeśli A_{3-0} > 9 lub AC = 1 to A3 - 0 ← A3 - 0 + 6 jeśli A_{7-4} > 9 lub C = 1 to A7-4 ← A7-4 + 6	0001 0101		1	AC C, P

Instrukcje logiczne					
Mnemonik	**Operacja**	**Struktura bajtów**		**Cykle**	**Znaczniki**
ANL A,Rn	A ← A and Rn	0101 1 r r r		1	P
ANL A,adr	A ← A and (adr)	0101 0101	[adr]	1	P
ANL A,@Ri	A ← A and (Ri)	0101 011 i		1	P
ANL A,#dana	A ← A and dana	0101 0100	[dana]	1	P
ANL adr,A	(adr) ← (adr) and A	0101 0010	[adr]	1	-
ANL adr,#dana	(adr) ← (adr) and dana	0101 0011	[adr] [dana]	2	-
ORL A,Rn	A ← A or Rn	0100 1 r r r		1	P
ORL A,adr	A ← A or (adr)	0100 0101	[adr]	1	P
ORL A,@Ri	A ← A or (Ri)	0100 011 i		1	P
ORL A,#dana	A ← A or dana	0100 0100	[dana]	1	P
ORL adr,A	(adr) ← (adr) or A	0100 0010	[adr]	1	-
ORL adr,#dana	(adr) ← (adr) or dana	0100 0011	[adr] [dana]	2	-
XRL A,Rn	A ← A xor Rn	0110 1 r r r		1	P
XRL A,adr	A ← A xor (adr)	0110 0101	[adr]	1	P
XRL A,@Ri	A ← A xor (Ri)	0110 011 i		1	P
XRL A,#dana	A ← A xor dana	0110 0100	[dana]	1	P
XRL adr,A	(adr) ← (adr) xor A	0110 0010	[adr]	1	-
XRL adr,#dana	(adr) ← (adr) xor dana	0110 0011	[adr] [dana]	2	-
CLR A	A ← 0	1110 0100		1	P
CPL A	A ← not A	1111 0100		1	-
RL A	(An+1) ← An, n = 0 – 6 A0 ← A7	0010 0011		1	-
RLC A	(An+1) ← An, n = 0 – 6 A0 ← C C ← A7	0011 0011		1	C P
RR A	An ← (An+1), n = 0 – 6 A7 ← A0	0000 0011		1	-
RRC A	An ← (An+1), n = 0 – 6 A7 ← C C ← A0	0001 0011		1	C P
SWAP bA	A_{7-4} <=> A_{3-0}	1100 0100		1	-

Instrukcje skoków i wywołań podprogramów – część I.					
Mnemonik	**Operacja**	**Struktura bajtów**		**Cykle**	**Znaczniki**
AJMP adr_11	PC $_{10-0}$ ← adr_11 PC $_{15-11}$ *bez zmian*	adr_{10} adr_9 adr_8 0 0010	adr_{7-0}	2	-
LJMP adr_16	PC ← adr_16	0000 0010	[adr $_{15-8}$] [adr_{7-0}]	2	-
SJMP rel	PC ← PC + 2 PC ← PC + rel	1000 0000	[rel]	2	-
JMP @A+DPTR	PC ← A + DPTR	0111 0011		2	-
JZ rel	PC ← PC + 2 jeśli A = 0 to PC ← PC + rel	0110 0000	[rel]	2	-
JNZ rel	PC ← PC + 2 jeśli A < > 0 to PC ← PC + rel	0111 0000	[rel]	2	-
JC rel	PC ← PC + 2 Jeśli C = 1 to PC ← PC + rel	0100 0000	[rel]	2	-
JNC rel	PC ← PC + 2 jeśli C = 0 to PC ← PC + rel	0101 0000	[rel]	2	-
JB bit,rel	PC ← PC + 3 jeśli bit = 1 to PC ← PC + rel	0010 0000 [adr bitu]	[rel]	2	-
JNB bit,rel	PC ← PC + 3 jeśli bit = 0 to PC ← PC + rel	0011 0000 [rel]	[adr bitu]	2	-
JBC bit,rel	PC ← PC + 3 jeśli bit = 1 to PC←PC + rel i bit←0	0001 0000 [rel]	[adr bitu]	2	-
CJNE A,adr,rel	PC ← PC + 3 jeśli A < (adr) to C ← 1 jeśli A >/ (adr) to C ← 0 jeśli A <> (adr) to PC ← PC + rel	1011 0101 [adr]	[rel]	2	C
CJNE A,#dana,rel	PC ← PC + 3 jeśli A < dana to C ← 1 jeśli A >/ dana to C ← 0 jeśli A < > dana to PC ← PC + rel	1011 0100 [rel]	[dana]	2	C
CJNE Rn,#dana,rel	PC ← PC + 3 jeśli Rn < dana to C ← 1 jeśli Rn >/ dana to C ← 0 jeśli Rn < > dana to PC ← PC + rel	1011 1 r r r [rel]	[dana]	2	C

Instrukcje skoków i wywołań podprogramów – część II.					
Mnemonik	**Operacja**	**Struktura bajtów**		**Cykle**	**Znaczniki**
CJNE @Ri,#dana,rel	PC ← PC + 3 jeśli (Ri) < dana to C ← 1 jeśli (Ri) >/ dana to C ← 0 jeśli (Ri) < > dana to PC ← PC + rel	1011 011 i [dana] [rel]		2	C
DJNZ Rn,rel	Rn ← Rn -1 PC ← PC +2 jeśli Rn <> 0 to PC ← PC + rel	1101 1 r r r	[rel]	2	-
DJNZ adr,rel	(adr)←(adr) - 1 PC←PC +3 jeśli (adr) <> 0 to PC←PC + rel	1101 0101 [rel]	[adr]	2	-
ACALL adr_11	PC ← PC + 2 SP ← SP + 1 (SP) ← PC7 - 0 SP ← SP + 1 (SP) ← PC15 – 8 PC_{10-0} ← adr_11 PC_{15-11} *bez zmian*	adr_{10} adr_9 adr_8 0 0001	adr_{7-0}	2	-
LCALL adr_16	PC ← PC + 3 SP←SP + 1 (SP)←PC_{7-0} SP←SP + 1 (SP)←PC_{15-8} PC ← adr_16	0001 0010 adr_{7-0}	adr_{15-8}	2	-
RET*PODPROGRAM*	PC_{15-8}←(SP) SP←SP – 1 PC_{7-0} ←(SP) SP←SP – 1	0010 0010		2	-
RETI*PRZERWANIE*	PC_{15-8}←(SP) SP←SP – 1 PC_{7-0}←(SP) SP←SP – 1	0011 0010		2	-

Legenda:

A	akumulator – kiedy mówimy o adresowaniu 8 bitów
ACC	akumulator – kiedy mówimy o adresowaniu bitowym
B	rejestr B
Rn	rejestr roboczy, R0...R7 (n = 0 dla R0, n = 7 dla R7)
Ri	rejestr roboczy – wskaźnik danych (i = 0 dla R0, i = 1 dla R1)
Rrr	rejestr R0...R7 (rrr=000 dla R0, rrr=111 dla R7)
DPTR	wskaźnik danych (DPH.DPL)
PC	licznik rozkazów
SP	wskaźnik stosu
C, CY	znacznik przeniesienia
AC	znacznik przeniesienia pomocniczego
OV	znacznik nadmiaru
dana	zmienna 8 bitowa
dana_16	zmienna 16 bitowa
Adr	adres pierwszych 128 bajtów wewnętrznej pamięci RAM (adr=0...7FH) lub rejestrów specjalnych SFR (adr=80H...0FFH)
Rel	8 - bitowe przesunięcie o wartościach z przedziału (-128,127)
Adr_11	11 – bitowy adres pamięci programu
Adr_16	16 – bitowy adres pamięci programu
And	iloczyn logiczny
Or	suma logiczna
Xor	suma modulo 2 (różnica symetryczna)
Not	negacja logiczna
( )	zawartość komórki pamięci danych lub programu o adresie podanym w nawiasie
@	w mnemoniku poprzedza adres pośredni
#	w mnemoniku poprzedza argument bezpośredni

3 Podstawy Asemblera dla układów MCS'51... czyli jak dogadać się z mikrokontrolerem?

Aby stworzyć plik programu, który następnie można będzie umieścić w pamięci systemu mikroprocesorowego potrzebujemy następujących narzędzi:

1. dowolny edytor tekstowy np. NOTEPAD, edytor systemowy dostępny w każdej nakładce na DOS np. NortonCommander czy DosNavigator (uruchamiane kombinacją klawiszy SHIFT+F4);
   program tworzymy jako plik tekstowy koniecznie z rozszerzeniem .ASM i umieszczamy w tym samym katalogu co wykorzystywany przez nas kompilator czyli program asemblera np. DSM51ASS.exe

Przykładowy plik **glowny.asm**:

```
KEY1  XDATA 0F000H
KEY2  XDATA KEY1+1

PUBLIC      KLAWISZ
EXTRN CODE (DISPLAY)
      CSEG AT 0
KLAWISZ:
      MOV DPTR,#KEY1
      MOVX A,@DPTR
      CJNE A,#0FFH,_0_7
      INC DPTR
      MOVX A,@DPTR
      CJNE A,#0FFH,_8_9
      SJMP KLAWISZ
_0_7:
      CPL A
      MOV R2,#0FFH
      CLR C
ROT:
      RRC A
      INC R2
      JNC ROT
      MOV A,R2
      ADD A,#30H
      LJMP DISPLAY
_8_9:
      JNB ACC.0,_8
      JB ACC.1,KLAWISZ
      MOV A,#'9'
      LJMP DISPLAY
_8:
      MOV A,#'8'
      LJMP DISPLAY
      END
```

2. dowolne środowisko programistyczne dla mikrokontrolerów rodziny MCS'51 zawierające wspomniany kompilator; programy tego typu udostępniane są bardzo często przez samych producentów układów w tym przypadku opisane zostaną programy dostępne na stronie www.micromade.pl firmy „MICROMADE" producenta *Dydaktycznego Systemu Mikroprocesorowego DSM-51* oraz producenta oprogramowania dla mikrokontrolerów '51 firmy KEIL www.keil.com.

I to w zasadzie tyle na początek.

3.1 Asembler 8051 DSM51ASS ver.1.01

Informacje zawarte w tym podrozdziale zaczerpnięte zostały z dokumentacji Dydaktycznego Systemu Mikroprocesorowego DSM-51 firmy „MICROMADE" z Piły.

3.1.1 Opis programu.

Program DSM51ASS jest asemblerem mikrokontrolera przeznaczonym do stosowania przy nauce programowania tego mikrokontrolera z wykorzystaniem Dydaktycznego Systemu Mikroprocesorowego DSM-51 i jest rozprowadzany razem z tym zestawem.

Podstawowe cech asemblera DSM51ASS:

- Jest makroasemblerem jednoprzebiegowym tzn., że wykorzystuje tzw. makroinstrukcje (fragmenty programu) zdefiniowane przez użytkownika przy użyciu odpowiednich dyrektyw asemblera, czyli instrukcji własnych kompilatora, nie będących częścią programu samego mikrokontrolera
- Komunikaty wypisuje w języku polskim
- Pozwala asemblować tylko pojedynczy plik wejściowy (nie ma fazy linkowania; *linkowanie* to łączenie różnych programów tzw. *segmentów* w jeden program przy użyciu programu zwanego *linkerem,* który najczęściej jest elementem wykorzystywanego środowiska programistycznego)
- Asemblowany program może liczyć maksymalnie około 20 000 linii listingu (zależnie od dostępnej pamięci)
- Zezwala na stosowanie rozbudowanych wyrażeń arytmetycznych o postaci zbliżonej do wyrażeń języka C
- Wszystkie wyrażenia liczy na liczbach w postaci 32-bitowej ze znakiem
- Wartości wszystkich symboli pamięta w postaci liczb 16-bitowych bez znaku

3.1.2 Wywołanie programu (asemblacja pliku źródłowego)

W przypadku korzystania z DOSa np. przy użyciu NortonCommandera lub DOSNavigatora wywołujemy działanie asemblera przez wpisanie do linii komend sekwencji

DSM51ASS <nazwa>

gdzie <nazwa> jest nazwą pliku zawierającego kod źródłowy programu. Jeśli w nazwie nie zawarto rozszerzenia to domyślnie przyjmowane jest rozszerzenie .asm.

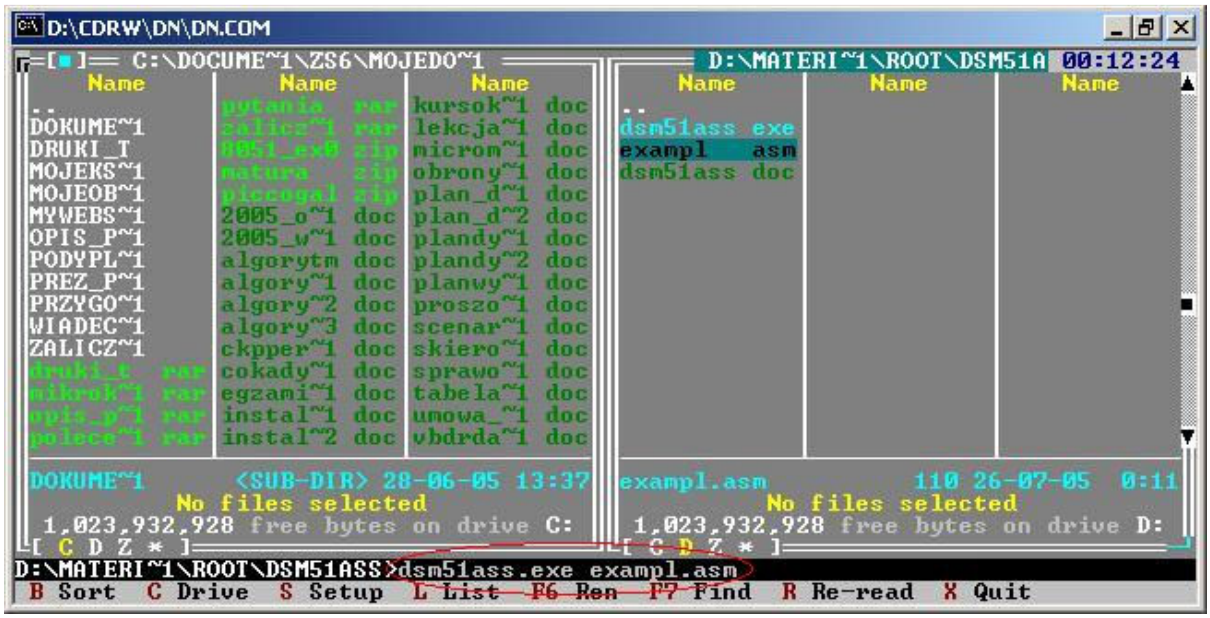
W wyniku działania programu powstają następujące zbiory:

<nazwa>.hex – plik w formacie INTELHEX przeznaczony dla programatora pamięci

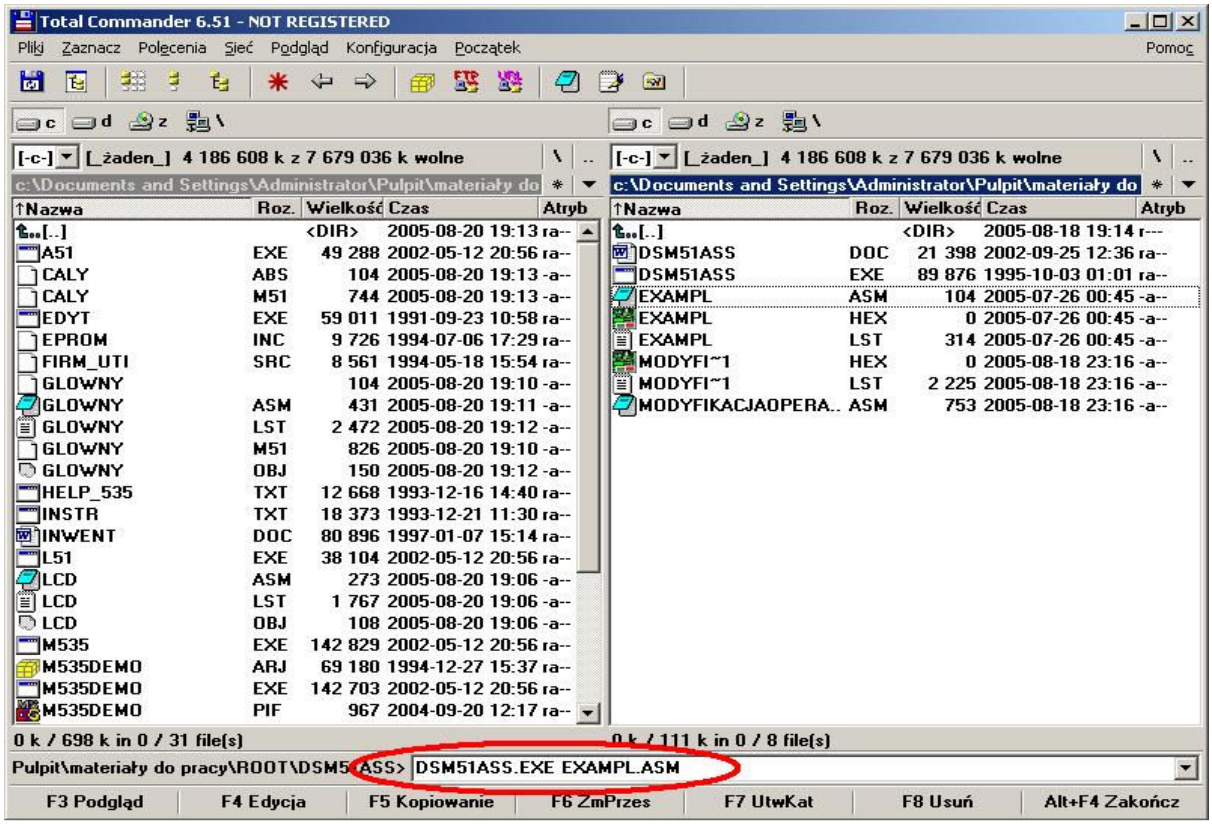
<nazwa>.lst – listing programu zawierający zarówno program w postaci źródłowej jak i w postaci kodu maszynowego; w przypadku wystąpienia błędów składni w pliku tym podana jest informacja o rodzaju błędu oraz jego umiejscowieniu w programie

Przykłady.

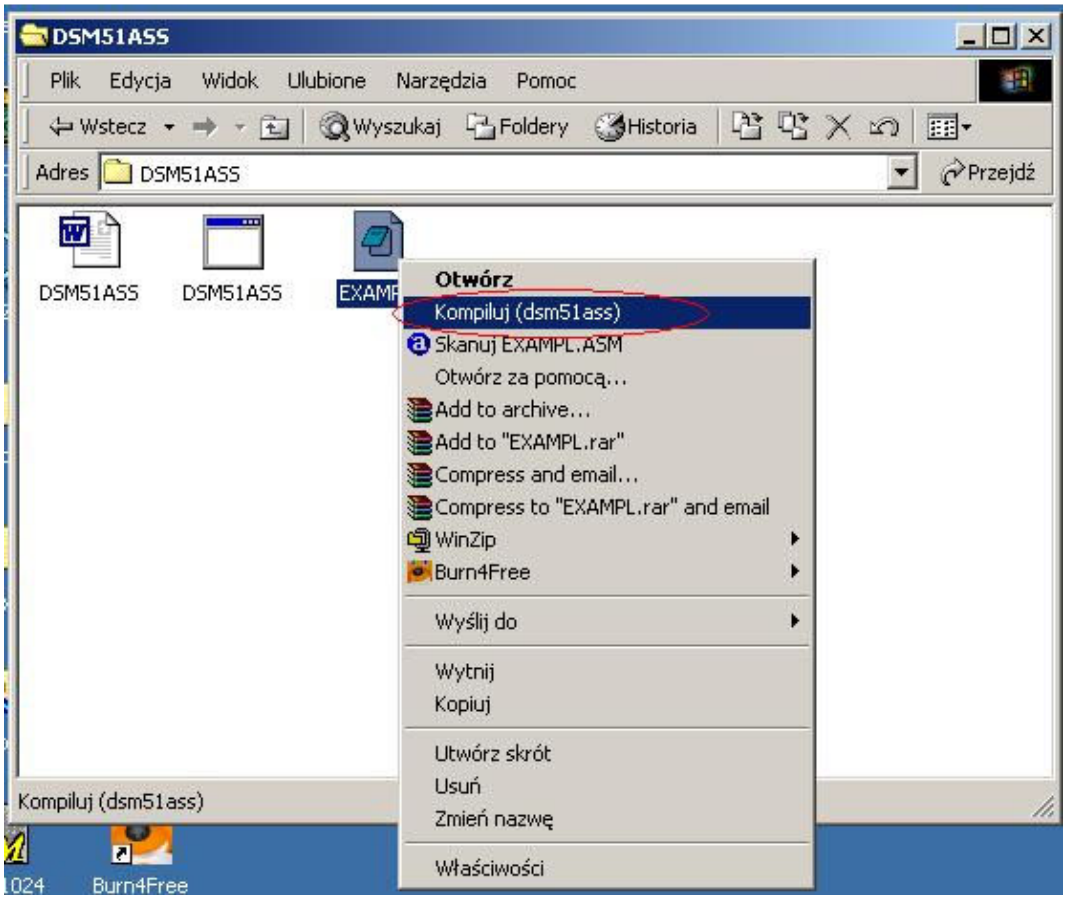
Wywołanie asemblacji pliku o nazwie EXAMPL.ASM (musi znajdować się w tym samym katalogu co DSMA51ASS) przy użyciu DOSNavigatora



Sytuacja wygląda identycznie w przypadku zastosowania np. TOTALCOMMANDERA dla Windows.



W przypadku wersji dla WINDOWS wywołanie asemblacji odbywa się np. poprzez użycie prawego klawisza myszy dla wybranego pliku źródłowego i wybór opcji z menu podręcznego lub techniką „przenieś i upuść” (plik .ASM „przenosimy i upuszczamy” na DSM51ASS.EXE).



Zawartość pliku EXAMPL.HEX czyli zestaw danych wykorzystywanych przy transmisji szeregowej do programatora

```
:03000000020100FA
:06010000780A790A740A76
:00000001FF
```

Zawartość pliku EXAMPL.LST

```
1  0000: 02 01 00      LJMP POCZ
2  0100:               ORG 100H
3  0100:           POCZ:
4  0100: 78 0A         MOV R0,#10
5  0102: 79 0A         MOV R1,#0AH
6  0104: 74 0A         MOV A,#00001010B
7                      END
```

Kolorem żółtym zaznaczono adres komórki pamięci, w której zapisany zostanie pierwszy bajt rozkazu zapisanego w danej linii.
Kolor turkusowy obejmuje kod maszynowy instrukcji programu w postaci heksadecymalnej (szesnastkowej).
Kolorem zielonym zaznaczono postać znakową programu zapisaną w asemblerze w pliku .asm.

Zawartość pliku .LST w przypadku wystąpienia błędu:

```
   1  0000: 02 01 00       LJMP POCZ
   2  0100:                ORG 100H
   3  0100:          POCZ:
   4                       MOV R0,#10
***** BŁĄD 2 : NIEZNANY MNEMONIK *****
   5  0100: 79 0A          MOV R1,#0AH
   6  0102: 74 0A          MOV A,#00001010B
   7                       END
```

W tym przypadku błąd polega na tym, że instrukcja została zapisana na pozycji zarezerwowanej dla etykiet.

3.1.3 Format linii programu.

Typowa linia programu w asemblerze ma postać:

[<etykieta>][<mnemonik rozkazu>][<operandy>][;<komentarz>]

znaczenie poszczególnych pól linii programu jest następujące:

<etykieta> - symbol umieszczony na samym początku linii (dla uniknięcia błędów składni warto zwrócić uwagę na to, że etykieta powinna być jako jedyny element pliku umieszczana od początku wiersza tzn. pierwszy znak etykiety musi być pierwszym znakiem w linii, pozostałe pola linii warto, także dla lepszej czytelności pliku źródłowego, poprzedzić odstępem przez użycie klawisza tabulacji 'Tab'). Etykieta musi zaczynać się od litery lub znaku podkreślenia '_', i może zawierać dowolną kombinację liter, cyfr i podkreśleń (z tą dowolnością lepiej nie przesadzać, etykieta powinna być ciągiem znaków, nazwą symboliczną, która wyraża cel jej użycia np. PETLA, JESZCZE_RAZ, KONWERSJA_BCD itp. co znacznie zwiększa czytelność programu). Jeśli etykieta zakończona jest dwukropkiem to nadawana jest jej wartość określająca jej pozycję w kodzie źródłowym (adres rozkazu z tej linii programu). Innymi słowy etykiecie przypisywana jest przez asembler wartość równa adresowi pamięci, pod którym zapisany zostanie pierwszy bajt kodu maszynowego instrukcji programu poprzedzonej etykietą. Stosowanie etykiet jest konieczne najczęściej wtedy, kiedy w programie pojawiają się alternatywne drogi jego dalszej realizacji (instrukcje skoków warunkowych) lub kiedy istnieje konieczność realizacji instrukcji zapisanych w pliku źródłowym wcześniej, czyli pod niższym adresem w pamięci niż aktualnie wykonywana instrukcja). Dzięki etykietom programista zwolniony jest z konieczności wyliczania adresów bezwzględnych i względnych dla poszczególnych linii programu (obowiązek ten spada na asembler podczas kompilacji pliku źródłowego).
Etykiety w innym zastosowaniu pozwalają zastąpić dane liczbowe, którymi w rzeczywistości operuje procesor, nazwami symbolicznymi, które pozwalają na skojarzenie danej z konkretną informacją określającą konkretne zdarzenie.

Przykład.

JNB 20H.0,$
CLR 20H.0
JNB 20H.1,(tu należy wpisać adres instrukcji **SETB EX1,** czyli trzeba wiedzieć gdzie ta instrukcja będzie zapisana w pamięci; jest to oczywiście możliwe ale wymagałoby żmudnych obliczeń i wykorzystania informacji z listy rozkazów o strukturze bajtowej użytych w programie instrukcji)
CLR 20H.1
SJMP (tu należy wpisać adres ostatniej instrukcji w tym fragmencie programu

‘MOV A,R2’)
SETB EX1
SJMP (tu należy wpisać adres pierwszej instrukcji w tym fragmencie programu
‘JNB 20H.0,$’
MOV A,R2

Wprowadzenie etykiet pozwala przekazać informacje o znaczeniu poszczególnych danych jako argumentów instrukcji oraz umożliwia wskazanie kolejności wykonywania instrukcji zapisanych w programie.

Konkretnie:

Adresowi bitu komórki 20H.0 nadano nazwę FL_INT (flaga przerwania (interrupt)), gdyż ustawienie tego bitu komórki oznacza wystąpienie wcześniej przerwania.

Adresowi bitu komórki 20H.1 nadano nazwę FL_KON (flaga konwersji), gdyż ustawienie tego bitu związane jest z konwersją sygnału przez przetwornik A/C

Pozostałe nazwy dotyczą adresów linii programu i pozwalają uzyskać właściwą zależność pomiędzy poszczególnymi instrukcjami:

- **PON_INT** (ponownie przerwanie)
- **CZEKAJ** (chyba nie trzeba tłumaczyć co oznacza)
- **WYNIK** (wskazanie instrukcji, która rozpoczyna fragment programu związany z odczytem wyniku czasu przetwarzania)

PON_INT:
JNB FL_INT,$
CLR FL_INT
JNB FL_KON,CZEKAJ (w ten sposób pokazano, że po spełnieniu określonego warunku należy wykonać jako następną instrukcję poprzedzoną etykietą **‘CZEKAJ’ czyli SETB EX1)**
CLR FL_KON
SJMP WYNIK(w ten sposób pokazano, że jako następną należy wykonać instrukcję poprzedzoną etykietą **‘WYNIK’ czyli MOV A,R2)**
CZEKAJ:
SETB EX1
SJMP PON_INT (w ten sposób pokazano, że jako następną należy wykonać instrukcję poprzedzoną etykietą **‘PON_INT’ czyli JNB FL_INT,$)**
WYNIK: MOV A,R2
Podczas asemblacji pliku źródłowego w miejsce etykiet podstawione zostaną oczywiście konkretne wartości danych.

```
002D:          PON_INT:
002D: 30 01 FD      JNB FL_INT,$
0030: C2 01         CLR FL_INT
0032: 30 00 04      JNB FL_KON,CZEKAJ
0035: C2 00         CLR FL_KON
0037: 80 04         SJMP WYNIK
0039:          CZEKAJ:
0039: D2 AA         SETB EX1
003B: 80 F0         SJMP PON_INT
003D: EA       WYNIK:    MOV A,R2
```

<mnemonik rozkazu> - mnemonik kodu maszynowego procesora, dyrektywa asemblera lub makro; jest to skrót wyrażenia w języku angielskim opisującego działanie danej instrukcji, funkcję

dyrektywy asemblera; rozumienie znaczenia tego skrótu znacznie ułatwia zapamiętanie poszczególnych instrukcji a co najważniejsze pozwala na rozumienie i sensowne stosowanie danej instrukcji w programie; mnemoniki poszczególnych rozkazów szczegółowo zostały wyjaśnione w rozdziale 3 (Alfabetyczna lista rozkazów mikrokontrolera 8051) .

<operandy> - informacje wymagane przez mnemonik, dyrektywę asemblera lub makro. Poszczególne operandy są oddzielane przecinkami.

Operandami mogą być:

- adresy programów i lokacji programowych
- adresy wewnętrznej pamięci danych 8051
- adresy pośrednie
- adresy bitów
- stałe
- symbole specjalne (nazwy rejestrów)

<komentarz> - wszystkie znaki występujące po średniku są traktowane jako komentarz i ignorowane przez asembler.

Za pomocą średnika można w szybki sposób eliminować daną instrukcję z programu co często może być przydatne w trakcie uruchamiania i testowania nowego programu.

Poszczególne pola linii programu muszą być oddzielone między sobą co najmniej jednym znakiem spacji (lub tabulacji).
W programie mogą występować puste linie lub linie zawierające wyłącznie komentarz.
Przykład:

```
;*************************************************************
****
;**********************OBSŁUGA
PRZERWANIA**************************
;*************************************************************
****
        ORG 0FFA0H
OBS_INT:
                CLR TR0     ; zatrzymanie pracy timera
```

Pierwsze trzy linie poprzedzone średnikiem odpowiadają liniom zawierającym wyłącznie komentarz; w tym przypadku w sposób wizualny zaznaczono początek specyficznego fragmentu programu.

Następnie użyto dyrektywy o mnemoniku ORG, która służy w tym przypadku do określenia lokacji tego fragmentu programu od adresu 0FFA0H w pamięci programu (operand 0FFA0H jest w tym przypadku adresem bezpośrednim lokacji programu).

W kolejnej linii użyto etykiety, która wystąpi także w innym fragmencie programu (konkretnie np. w instrukcji wywołania tego fragmentu jako podprogramu LCALL OBS_INT). Etykieta ta zastępuje adres lokacji 0FFA0H a poprzez skojarzenie czyni program bardziej czytelnym niż w przypadku stosowania wartości liczbowych)

Następna linia to typowa linia programu z rozkazem o mnemoniku CLR (CLEAR BIT) z operandem TR0 w postaci predefiniowanej nazwy symbolicznej adresu bitowego jednego z bitów sterujących rejestru specjalnego związanego z układem wewnętrznego licznika mikrokontrolera i komentarzem wyjaśniającym zastosowanie tej instrukcji w tym fragmencie programu.

3.1.4 Wyrażenia arytmetyczne.

Wyrażenia arytmetyczne służą do określenia wartości parametrów wymagających podania wartości liczbowej.

Wyrażenia składają się ze stałych liczbowych i symboli (etykiety, nazwy stałych lub zmiennych) połączonych operatorami arytmetycznymi.

Składnia wyrażeń asemblera DSM51ASS została zaczerpnięta z języka C. Wprowadzono kilka dodatkowych operatorów stosowanych powszechnie w asemblerach oraz zmieniono priorytet operacji bitowych - są one wykonywane przed operacjami porównań. Operatory porównań dają wartość 1 jeżeli warunek jest prawdziwy oraz 0 jeżeli jest fałszywy. Operatory logiczne (!,&&,||) każdą wartość różną od zera traktują jako prawdę, a wartość 0 jako fałsz. Jako wynik operacji logicznych również uzyskujemy wartość 1 lub 0.

W asemblerze DSM51ASS wszystkie obliczenia wykonywane są na liczbach 32 bitowych ze znakiem. Oznacza to, że wartość wyrażenia jest wyliczona prawidłowo dopóki wyniki pośrednie mieszczą się w zakresie -2,147,483,648 do 2,147,483,647. Przekroczenie tego zakresu w czasie obliczeń nie jest sygnalizowane.

Wartości symboliczne używane w programie są przechowywane jako liczby 16 bitowe bez znaku.

Asembler kontroluje wartość i typ wyrażenia. Jeżeli wartość wyrażenia arytmetycznego wykracza poza zakres dopuszczony dla aktualnego parametru lub typ wyrażenia jest nieodpowiedni to generowany jest błąd.

Przykład:

```
FOSC_MHZ        EQU  24
CYKL_MASZ_MIKROS        EQU 12/FOSC_MHZ
```

Powyższy ciąg wyrażeń spowoduje że wyrażeniu CYKL_MASZ_MIKROS zostanie przypisana wartość 2 jako czas cyklu maszynowego wykonania instrukcji określony w µs, po podstawieniu do wyrażenia 12/FOSC_MHZ wartości 24 odpowiadającej wyrażeniu FOSC_MHZ jako częstotliwości rezonatora generatora zegara systemowego wyrażonej w MHz.

3.1.5 Stałe liczbowe.

Stała liczbowa musi zaczynać się od cyfry (w przypadku liczb szesnastkowych zaczynających się od litery np. C500H należy ją poprzedzić nieznaczącym zerem czyli zapisać jako 0C500H)

DSM51ASS akceptuje następujące typy stałych liczbowych:

Typ	Składnia	Przykład
Dziesiętny	<cyfry>	125
Szesnastkowy	<cyfra><cyfry szesnastkowe>H	0FFFFH
Ósemkowy	<cyfry ósemkowe>O	7777O
Binarny	<cyfry binarne>B	10101B
Znakowy	'<znak>'	'A'

3.1.6 Symbole.

Symbole reprezentowane są przez ciąg znaków zaczynający się od litery lub znaku podkreślenia ‘_’ i składający się z dowolnej sekwencji liter, cyfr i podkreśleń.
Asembler rozpoznaje pierwsze 32 znaki symbolu.

W asemblerze DSM51ASS zdefiniowano standardowe symbole reprezentujące poszczególne rejestry i bity procesora 8051. Poza tym asembler rozpoznaje symbole określające adresy urządzeń

systemu DSM-51 i zawartych w jego pamięci EPROM podprogramów standardowych np. CSDA jako adres przetwornika cyfrowo-analogowego czy DELAY_100MS jako nazwę podprogramu odmierzenia opóźnienia o wielokrotności 100 ms.
Listę tych symboli umieszczono w rozdziale „Predefiniowane symbole” w końcowej części tego podrozdziału.

3.1.7 Operatory arytmetyczne, logiczne i bitowe.

Operatory w/g priorytetu ich wykonywania:

(),	Nawiasy
!, ~, +, -, <, >	Modyfikacje wartości
.0, .1, .2, .3, .4, .5, .6, .7	Selekcja bitów
*, /, %	Mnożenie, dzielenie
+, -	Dodawanie, odejmowanie
<<, >>	Przesunięcia bitowe
&	Bitowe AND
^	Bitowe XOR
|	Bitowe OR
<, <=, >, >=, =, ==, !=	Porównania
&&	Logiczne AND
||	Logiczne OR

Znaczenie poszczególnych operatorów:

() - Nawiasy określają kolejność wykonywania działań. Nie ma ograniczenia liczby zastosowanych zagłębionych nawiasów.

Operatory modyfikujące wartość następującego po nich operandu:
! - Negacja logiczna. Zmienia wartość różną od 0 na 0, a wartość równą 0 na 1.
~ - Negacja bitowa. Zmienia wszystkie 32 bity w operandzie na odwrotne.
+ - Nie zmienia wartości.
- - Zmienia znak operandu na przeciwny.
< - Najmłodszy bajt operandu (wyrażenie '< operand' jest równoważne wyrażeniu '(operand & 0FFH)' lub '(operand /256)').

> - Starsze 3 bajty operandu (wyrażenie '> operand' jest równoważne wyrażeniu '(operand >> 8)' lub '(operand / 256)').
Przykład:

```
1   0000   NEG_LOG_1   EQU !20 ;wartość 20 zamieniono na 0
                                (negacja logiczna)
2   0001   NEG_LOG_2   EQU !0  ;wartość 0 zamieniono na 1
                                (negacja logiczna)
3   FFFE   NEG_BIT   EQU   ~1   ;   wartość   1   czyli   bitowo
0000000000000001B
                    zamieniono   na   FFFEH   czyli   bitowo
              1111111111111110B
                    (negacja bitowa)
4   00EF   LOW_OP    EQU <9ACDEFH ; z 3-bajtowej liczby 9ACDEF
                                wydzielono najmłodszy bajt czyli
                                EF
5   9ACD   HIGH3_OP  EQU >9ACDEFH ; z 3-bajtowej liczby 9ACDEF
                                wydzielono 2 starsze bajty czyli
                                9ACD
```

Operatory selekcji bitów w bitowo adresowalnych rejestrach.

.n - Występując po adresie (nazwie) rejestru oblicza adres wskazanego bitu tego rejestru. Jeżeli dany adres nie jest adresem bitowo adresowalnego rejestru sygnalizowany jest błąd. 'n' jest cyfrą z zakresu 0…7.
Przykład:

```
000E   BIT_14_E    EQU 21H.6 ; przyporządkowuje adres 14 (0EH) w obszarze
                    adresowalnym bitowo bitowi o numerze 6 rejestru o adresie
                21H.
```

Operatory mnożenia i dzielenia.

***** - Mnożenie.
/ - Dzielenie.
% - Dzielenie 'modulo' (reszta z dzielenia).

Operatory dodawania i odejmowania.

+ - Dodawanie.
- - Odejmowanie.

Przykład:

```
0004   WYNIK_4   EQU   (6*4+1)/5-1   ;   wyrażeniu   WYNIK_4
przyporządkowano wynik działania arytmetycznego (6*4+1)/5-1
```

Operatory przesunięć bitowych.

<< - Przesunięcie w lewo. Operand występujący z lewej strony operatora jest przesuwany w lewo o liczbę bitów określoną przez operand występujący z prawej strony. Na zwalniane bity wchodzą zera.

Przykład:
03C=(00111100B) ROT_L EQU 1111B<<2

>> - Przesunięcie w prawo. Operand występujący z lewej strony operatora jest przesuwany w prawo o liczbę bitów określoną przez operand występujący z prawej strony. Na zwalniane bity wchodzą zera.
Przykład:
01=(0001B) ROT_P EQU 1111B>>3

Operatory bitowe.

& - AND między bitami operandów. Odpowiedni bit w wyniku ma wartość '1' tylko wtedy, gdy odpowiadające mu bity w obu operandach mają wartość '1'. W pozostałych przypadkach ma wartość '0'. Operacja ta jest wykonywana dla wszystkich 32 bitów.

Operand1	Operand2	Wynik
0	**0**	**0**
0	1	0
1	0	0
1	**1**	**1**

Przykład:
08H=1000B _AND EQU 1100B&1001B
^ -

Operand1	Operand2	Wynik
0	**0**	**0**
0	**1**	**1**
1	0	1
1	**1**	**0**

Przykład:
05H=0101B _XOR EQU 1100B^1001B

| - (OR) suma logiczna między bitami operandów. Odpowiedni bit w wyniku ma wartość '1', gdy co najmniej jeden z odpowiadających mu bitów w operandach ma wartość '1'. W przeciwnym przypadku ma wartość '0'. Operacja ta jest wykonywana dla wszystkich 32 bitów.

Operand1	Operand2	Wynik
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

Przykład:

```
0DH=1101B      _OR      EQU 1100B|1001B
```

Operatory porównań.

< - Mniejszy. Prawda (=1) gdy wartość lewego operandu jest mniejsza od wartości prawego operandu.

<= - Mniejszy lub równy. Prawda (=1) gdy wartość lewego operandu jest mniejsza od lub równa wartości prawego operandu.

> - Większy. Prawda (=1) gdy wartość lewego operandu jest większa od wartości prawego operandu.

>= - Większy lub równy. Prawda (=1) gdy wartość lewego operandu jest większa od lub równa wartości prawego operandu.

= - Równy. Prawda (=1) gdy wartość lewego operandu jest równa wartości prawego operandu.

== - Równy. Prawda (=1) gdy wartość lewego operandu jest równa wartości prawego operandu.

!= - Różny. Prawda (=1) gdy wartość lewego operandu jest różna od wartości prawego operandu.

Przykład:

```
0001     POR_1    EQU 2<3 ;(PRAWDA)
0000     POOR_2   EQU 4<=3 ;(FAŁSZ)
0000     POR_3    EQU 'A'='B' ;(FAŁSZ)
0001     POR_4    EQU 'A'!='B' ;(PRAWDA)
```

Operatory logiczne.

&& - AND logiczne. Prawda (=1) gdy oba operandy mają wartości różne od zera.

|| - OR logiczne. Prawda (=1) gdy co najmniej jeden z operandów ma wartość różną od zera.

3.1.8 Dyrektywy asemblera.

W poszczególnych liniach programu oprócz mnemoników oznaczających poszczególne rozkazy procesora mogą wystąpić dyrektywy asemblera. Umożliwiają one wstawianie danych w treść programu, przypisywanie wartości symbolom, sterowanie przebiegiem asemblacji i budowanie makr (zestawów poleceń wywoływanych pojedynczą nazwą).

Asembler DSM51ASS akceptuje następujące dyrektywy:

Dyrektywy danych:

DB - wstawienie w kod wartości numerycznych i tekstowych,
DW - wstawienie w kod dwubajtowych wartości numerycznych,
EQU- definiowanie stałej,
BIT - definiowanie stałej typu bit,
REG- definiowanie stałej typu rejestr,

SET - definiowanie zmiennej.

Dyrektywy sterujące:

IF - początek bloku warunkowej asemblacji,
ELSE- początek alternatywnego bloku warunkowej asemblacji,
ENDIF- koniec bloku warunkowej asemblacji,
ORG- ustawienie adresu dla następnego bloku kodu,
END- koniec programu.

Dyrektywy makrodefinicji:

MACRO - początek definicji makra,
ENDM (MACEND) - koniec definicji makra.

Dyrektywy danych.

DB - wstawienie w kod wartości numerycznych i tekstowych
Składnia:
[<etykieta>] DB <parametry>

Przykład:

```
TAB_KOD:        DB 01H,0A8H,25
TEXT :          DB ‘Przyklad’
0006: 01 A8 19  TAB_KOD:        DB 01H,0A8H,25
0009: 50 72 7A  TEXT:           DB 'Przyklad'
000C: 79 6B 6C
000F: 61 64
```

Wpisuje w treść programu wartości parametrów. Poszczególne parametry oddzielane są przecinkami. Parametry mogą być wyrażeniami arytmetycznymi lub ciągami znaków ujętymi w znaki ' lub ".
Wartości kolejnych parametrów będących wyrażeniami arytmetycznymi są wpisywane w kolejne bajty treści programu.
Parametry będące ciągami znaków są wpisywane w całości do treści programu. W ciągu znaków ujętym w znaki " może wystąpić znak ' i odwrotnie.

DW - wstawienie w kod dwubajtowych wartości numerycznych
Składnia:
[<etykieta>] DW <parametry>
Przykład:

```
0000: 02 0A FF    TABLICA:  DW 2*256+10,65535,0ABBAH
0003: FF AB BA
```

Wpisuje w treść programu wartości parametrów. Poszczególne parametry oddzielane są przecinkami. Parametry są wyrażeniami arytmetycznymi.
Wartość każdego parametru jest wpisywana w dwa kolejne bajty treści programu (najpierw starszy).

EQU - definiowanie stałej.
Składnia:
<symbol> EQU <wyrażenie>
Symbolowi <symbol> przypisywana jest wartość wyrażenia. Typ symbolu ustalany jest na podstawie wyrażenia.
Każda wartość zdefiniowana dyrektywą EQU jest stałą i nie może być zmieniana w trakcie asemblacji.
Przykład:

```
0022          STALA       EQU 22H
0011: E5 22                MOV A,STALA
```

BIT - definiowanie stałej typu bit.
Składnia:
<symbol> BIT <wyrażenie>
Symbolowi <symbol> przypisywana jest wartość wyrażenia. Kontroluje typ wyrażenia. Zdefiniowany symbol może być używany wyłącznie jako adres bitu.
Wartość symbolu nie może być zmieniana w trakcie asemblacji.
Przykład:

```
008C      STOPER BIT TR0
D2 8C          SETB STOPER
```

REG - definiowanie stałej typu rejestr.
Składnia:
<symbol> REG <wyrażenie>
Symbolowi <symbol> przypisywana jest wartość wyrażenia. Kontroluje typ wyrażenia. Zdefiniowany symbol może być używany wyłącznie jako adres rejestru wewnętrznego mikrokontrolera lub komórki wewnętrznej pamięci RAM.
Wartość symbolu nie może być zmieniana w trakcie asemblacji.
Przykład:

```
0099      BUFOR   REG SBUF
E5 99          MOV A,BUFOR
```

SET - definiowanie zmiennej.
Składnia:
<symbol> SET <wyrażenie>
Symbolowi <symbol> przypisywana jest wartość wyrażenia. Typ symbolu ustalany jest na podstawie wyrażenia.
Wartości zdefiniowane dyrektywą SET mogą być dowolnie wiele razy modyfikowane przez ponowne użycie dyrektywy SET. Zmiana typu symbolu w trakcie kolejnego przypisania powoduje wygenerowanie ostrzeżenia.
Przykład:

```
0022      ZMIENNA     SET 22H
7A 22                 MOV R2,#ZMIENNA
0033      ZMIENNA     SET 33H
7B 33                 MOV R3,#ZMIENNA
```

Dyrektywy sterujące.

IF - początek bloku warunkowej asemblacji.
Składnia:

IF <wyrażenie>

Jeżeli wartość wyrażenia jest różna od '0' (prawda) to kod występujący za tą dyrektywą jest asemblowany. Jeżeli wartość wyrażenia jest równa '0' (fałsz) i istnieje dyrektywa ELSE to kod występujący po ELSE jest asemblowany.
Dyrektywa ENDIF zamyka blok kodu asemblowanego warunkowo.
Dopuszczalne jest zagłębianie dyrektyw IF do 16 poziomów.

ELSE - początek alternatywnego bloku warunkowej asemblacji.
Składnia:

ELSE

Używana w połączeniu z dyrektywą **IF**. Jeśli wyrażenie testowane w dyrektywie IF ma wartość '0' to alternatywny kod zaznaczony przez ELSE jest asemblowany.

ENDIF - koniec bloku warunkowej asemblacji.
Składnia:

ENDIF

Zakończenie bloku warunkowej asemblacji rozpoczętego dyrektywą **IF**.

Przykład:

```
        IF 2>3
        MOV A,B
        ELSE
        MOV B,A
        ENDIF
```

Po asemblacji:

```
[01]            IF 2>3  (FAŁSZ)
                MOV A,B
[01]            ELSE
 F5 F0          MOV B,A
[00]            ENDIF
```

Przykład:

```
IF 4>3
        MOV A,B
        ELSE
        MOV B,A
        ENDIF
```

Po asemblacji:

```
[01]            IF 4>3 (PRAWDA)
E5 F0           MOV A,B
[01]            ELSE
```

```
                MOV B,A
[00]            ENDIF
```

ORG - ustawienie adresu dla następnego bloku kodu.
Składnia:

ORG <wyrażenie>

Ustawienie adresu dla następującego po tej dyrektywie bloku kodu. Adres dla następnej instrukcji procesora jest ustalany poprzez wyliczenie wartości wyrażenia. Możliwe jest jedynie zwiększanie aktualnego adresu kodu. Próba zmniejszenia adresu jest sygnalizowana jako błąd.
Standardowo kod programu jest umieszczany rozpoczynając od adresu 0.
W przypadku tworzenia tzw. modułów relokowalnych i łączenia ich przy użyciu linkera dyrektywa ORG blokuje możliwość umieszczania innych modułów relokowalnych pod adresami mniejszymi od argumentu użytego wraz z dyrektywą.
Przykład:

```
MOV A,R1
MOV A,R2
ORG 4444H
CLR STOPER
ORG 5555H
SETB TR1
```

Po asemblacji:

```
001D: E9          MOV A,R1
001E: EA          MOV A,R2
4444:       ORG 4444H
4444: C2 8C       CLR STOPER
5555:       ORG 5555H
5555: D2 8E       SETB TR1
```

W tym przypadku dyrektywa ORG spowodowała umieszczenie fragmentów występujących po dyrektywie pod adresem wskazanym w składni (4444H oraz 5555H). Jednocześnie niemożliwe byłoby umieszczenie przez linker innych modułów w tym przypadku w obszarach 001FH-4443H i 4446H-5554H.

END - koniec programu.
Składnia:

END

Zaznaczenie końca programu. Linie występujące w pliku źródłowym po tej dyrektywie nie są asemblowane.
Użycie tej dyrektywy w programie nie jest konieczne. Przy jej braku końcem programu jest koniec pliku.
Przykład:

```
MOV A,R1
MOV A,R2
ORG 4444H
CLR STOPER
END
```

```
        ORG 5555H
        SETB TR1
```
Po asemblacji:
```
001D: E9        MOV A,R1
001E: EA        MOV A,R2
4444:           ORG 4444H
4444: C2 8C     CLR STOPER
                END
```
Dyrektywy makrodefinicji.

MACRO - początek definicji makra.
Składnia:

<Nazwa> MACRO <parametry>

Makro to zestaw instrukcji asemblera. Ciąg instrukcji występujący po linii zawierającej dyrektywę MACRO, aż do najbliższej dyrektywy ENDM, tworzy makro o nazwie <Nazwa>.
Po zdefiniowaniu cały taki zestaw może być włączony w kod źródłowy programu poprzez wywołanie makra.
W treści makr mogą występować bez ograniczeń wywołania innych makr, ale nie może wystąpić definicja innego makra. Próba wywołania przez makro samego siebie (lub innego zapętlenia wywoływań prowadzącego do nieskończonego rozwijania makr) jest wykrywana i sygnalizowana jako błąd.
Parametry to oddzielone przecinkami symbole (parametry formalne makra), które mogą być wykorzystywane w treści makra.

Makro jest wywoływane poprzez umieszczenie jego nazwy w polu rozkazu danej linii programu. Przy wywołaniu podawane są parametry aktualne makra.
W czasie wstawiania makra w kod programu asembler zastępuje wszystkie parametry formalne parametrami aktualnymi.
Asembler nie umożliwia tworzenia etykiet lokalnych w makrach. Jeśli występuje taka potrzeba to można podawać etykietę (lub jej fragment) jako jeden z parametrów makra. Jest to możliwe, gdyż zastępowanie parametrów formalnych parametrami aktualnymi odbywa się na drodze podmieniania tekstów przed asemblacją linii programu. Teksty te są podmieniane niezależnie od tego, czy występują w etykiecie, nazwie mnemonika, czy w treści operandu.

ENDM (MACEND) - koniec definicji makra.
Składnia:

ENDM

Dyrektywa **ENDM** kończy definicję makra. Alternatywną nazwą tej dyrektywy jest **MACEND**.
Przykład:
```
            SETB TR1
AKUM        MACRO
            MOV A,R1
            MOV A,R2
            MACEND
            SETB STOPER
            AKUM
            CLR TR1
            AKUM
```

Po asemblacji:

```
5555: D2 8E             SETB TR1
            AKUM        MACRO
                        MOV A,R1
                        MOV A,R2
                        MACEND
5557: D2 8C             SETB STOPER
            AKUM
>   1  5559: E9         MOV A,R1
>   2  555A: EA         MOV A,R2
>   3                   MACEND
555B: C2 8E             CLR TR1
            AKUM
>   1  555D: E9         MOV A,R1
>   2  555E: EA         MOV A,R2
>   3                   MACEND
```

3.1.9 Predefiniowane symbole.

Rejestry funkcji specjalnych:

```
Symbol      Nazwa (opis)                          Adres
=====================================================
ACC   akumulator                                  0E0H
B     rejestr B                                   0F0H
PSW   rejestr stanu                               0D0H
SP    wskaźnik stosu                              81H
DPTR  dwubajtowy wskaźnik danych
  DPL młodszy bajt                                 82H
  DPH starszy bajt                                 83H
P0    Port 0                                       80H
P1    Port 1                                       90H
P2    Port 2                                      0A0H
P3    Port 3                                      0B0H
IP    rejestr kontroli priorytetów przerwań       0B8H
IE    rejestr zezwoleń na przerwania              0A8H
TMOD  timery - tryby pracy                         89H
TCON  timery - sterowanie                          88H
TH0   timer 0 starszy bajt                         8CH
TL0   timer 0 młodszy bajt                         8AH
TH1   timer 1 starszy bajt                         8DH
TL1   timer 1 młodszy bajt                         8BH
SCON  sterownie transmisją szeregową               98H
SBUF  bufory transmisji szeregowej                 99H
PCON  sterowanie trybami 'power down'              87H


Bity adresowalne bezpośrednio

Symbol      Nazwa (opis)                      Adres
=====================================================
P     PSW.0                                   0D0H
OV    PSW.2                                   0D2H
RS0   PSW.3                                   0D3H
```

```
RS1    PSW.4                              0D4H
F0     PSW.5                              0D5H
AC     PSW.6                              0D6H
CY     PSW.7                              0D7H

RXD    P3.0                               0B0H
TXD    P3.1                               0B1H
INT0   P3.2                               0B2H
INT1   P3.3                               B3H
T0     P3.4                               0B4H
T1     P3.5                               0B5H
WR     P3.6                               0B6H
RD     P3.7                               0B7H

PX0    IP.0                               0B8H
PT0    IP.1                               0B9H
PX1    IP.2                               0BAH
PT1    IP.3                               0BBH
PS     IP.4                               0BCH

EX0    IE.0                               0A8H
ET0    IE.1                               0A9H
EX1    IE.2                               0AAH
ET1    IE.3                               0ABH
ES     IE.4                               0ACH
EA     IE.7                               0AFH

IT0    TCON.0                              88H
IE0    TCON.1                              89H
IT1    TCON.2                              8AH
IE1    TCON.3                              8BH
TR0    TCON.4                              8CH
TF0    TCON.5                              8DH
TR1    TCON.6                              8EH
TF1    TCON.7                              8FH

RI     SCON.0                              98H
TI     SCON.1                              99H
RB8    SCON.2                              9AH
TB8    SCON.3                              9BH
REN    SCON.4                              9CH
SM2    SCON.5                              9DH
SM1    SCON.6                              9EH
SM0    SCON.7                              9FH
```

UWAGA! Poniższe symbole zdefiniowane są jedynie dla asemblera DSM51ASS.

Urządzenia zewnętrzne systemu DSM-51

```
Symbol      Nazwa (opis)                          Adres
=================================================
CSIC  sterownik przerwań                          00H
CSDA  przetwornik cyfrowo/analogowy               08H
CSAD  przetwornik analogowo/cyfrowy               10H
CSMX  multiplekser analogowy                      18H
CSKB0 klawiatura matrycowa, klawisze 0..7         21H
CSKB1 klawiatura matrycowa, klawisze 8..          22H
CS55A układ 8255 rejestr portu A                  28H
CS55B układ 8255 rejestr portu B                  29H
CS55C układ 8255 rejestr portu C                  2AH
CS55D układ 8255 rejestr sterujący                2BH
CSDS  wyświetlacz 7-segm, wybór wskaźnika         30H
CSDB  wyświetlacz 7-segm, bufor danych            38H
```

```
CSMOD dekoder adresów (przełączanie trybu)      40H
LCDWC wyświetlacz LCD, wpis rozkazów            80H
LCDWD wyświetlacz LCD, wpis danych              81H
LCDRC wyświetlacz LCD, odczyt stanu             82H
LCDRD wyświetlacz LCD, odczyt danych            83H
CSX   zewnętrzna magistrala systemowa           0C0H
```

Podprogramy standardowe w pamięci EPROM systemu DSM-51

Symbol	Nazwa (opis)	Adres
WRITE_TEXT	wypisanie tekstu na LCD	8100H
WRITE_DATA	wypisanie znaku na LCD	8102H
WRITE_HEX	wypisanie liczby hex na LCD	8104H
WRITE_INSTR	wysłanie rozkazu do LCD	8106H
LCD_INIT	inicjalizacja LCD	8108H
LCD_OFF	wygaszenie LCD	810AH
LCD_CLR	ustawienie w stan początkowy	810CH
DELAY_US	opóźnienie (2*A+6)*12/11.059 us	810EH
DELAY_MS	opóźnienie A ms	8110H
DELAY_100MS	opóźnienie A * 100ms	8112H
WAIT_ENTER	"PRESS ENTER." i czeka na ENTER	8114H
WAIT_ENTER_NW	czekanie na klawisz ENTER	8116H
TEST_ENTER	sprawdzenie klawisza ENTER	8118H
WAIT_ENT_ESC	czekanie na ENTER lub ESC	811AH
WAIT_KEY	czekanie na dowolny klawisz	811CH
GET_NUM	wczytanie liczby BCD (4 cyfry)	811EH
BCD_HEX	zamiana BCD na HEX	8120H
HEX_BCD	zamiana HEX na BCD	8122H
MUL_2_2	mnożenie liczb 2 bajtowych	8124H
MUL_3_1	mnożenie 3bajty * 1bajt	8126H
DIV_2_1	dzielenie 2bajty / 1bajt	8128H
DIV_4_2	dzielenie 4bajty / 2bajty	812AH

3.2 Asembler KEIL A51

3.2.1 Wprowadzenie do asemblera A51.

Niniejsza instrukcja obejmuje wszystkie ważniejsze zasady realizowania języka asemblera, jednak szczegółowo odnosi się do asemblera A51 firmy KEIL. Z powodu znacznej uniwersalności opisu słowo ASEMBLER dotyczy równocześnie języka procesora 8051 i programu tłumaczącego A51. Większość zasad dotyczących Asemblera A51 zgodnych jest z opisywanym wcześniej Asemblerem DSM51ASS (zatem przedstawione w poprzednim rozdziale przykłady są aktualne także w tym przypadku i zostaną one w większości pominięte, przy czym możliwości A51 są większe np. pozwala on tworzyć pliki typu .OBJ umożliwiające łączenie różnych programów w całość przy spełnieniu określonych zasad tworzenia segmentów programów.

I jeszcze jedno. Wybór A51 do opisu może budzić wątpliwości (program powstał w 1985 roku i jest przeznaczony dla systemu DOS). Odpowiedź jest stosunkowo prosta a wynika ona jak to zwykle bywa z możliwości sprzętowych i finansowych jakimi dysponuje szkoła. Ciągle jesteśmy zmuszeni do wykorzystywania komputerów, których konfiguracja nie pozwala na swobodne wykorzystywanie środowiska WINDOWS (i chyba dobrze, bo dzięki temu niektórzy uczniowie naprawdę przekonują się o tym, że bez Windows'a komputer też działa!!!).

3.2.2 Wywołanie programu (asemblacja pliku źródłowego).

Stworzony plik .asm poddajemy kompilacji przy użyciu asemblera A51.exe. Zasadniczą różnicą w stosunku do opisywanego wcześniej DSM51ASS jest to, że w wyniku działania programu A51 nie powstaje plik w standardzie INTELHEX (.hex).

Plikiem wyjściowym po kompilacji jest plik .OBJ pozwalający na zastosowanie w programie linkera do łączenia z innymi modułami programów przy spełnieniu określonych zasad opisanych w dalszej części tego opracowania.

Podobnie jak w przypadku DSM51ASS tworzony jest natomiast plik .LST pełniący taką samą funkcję.

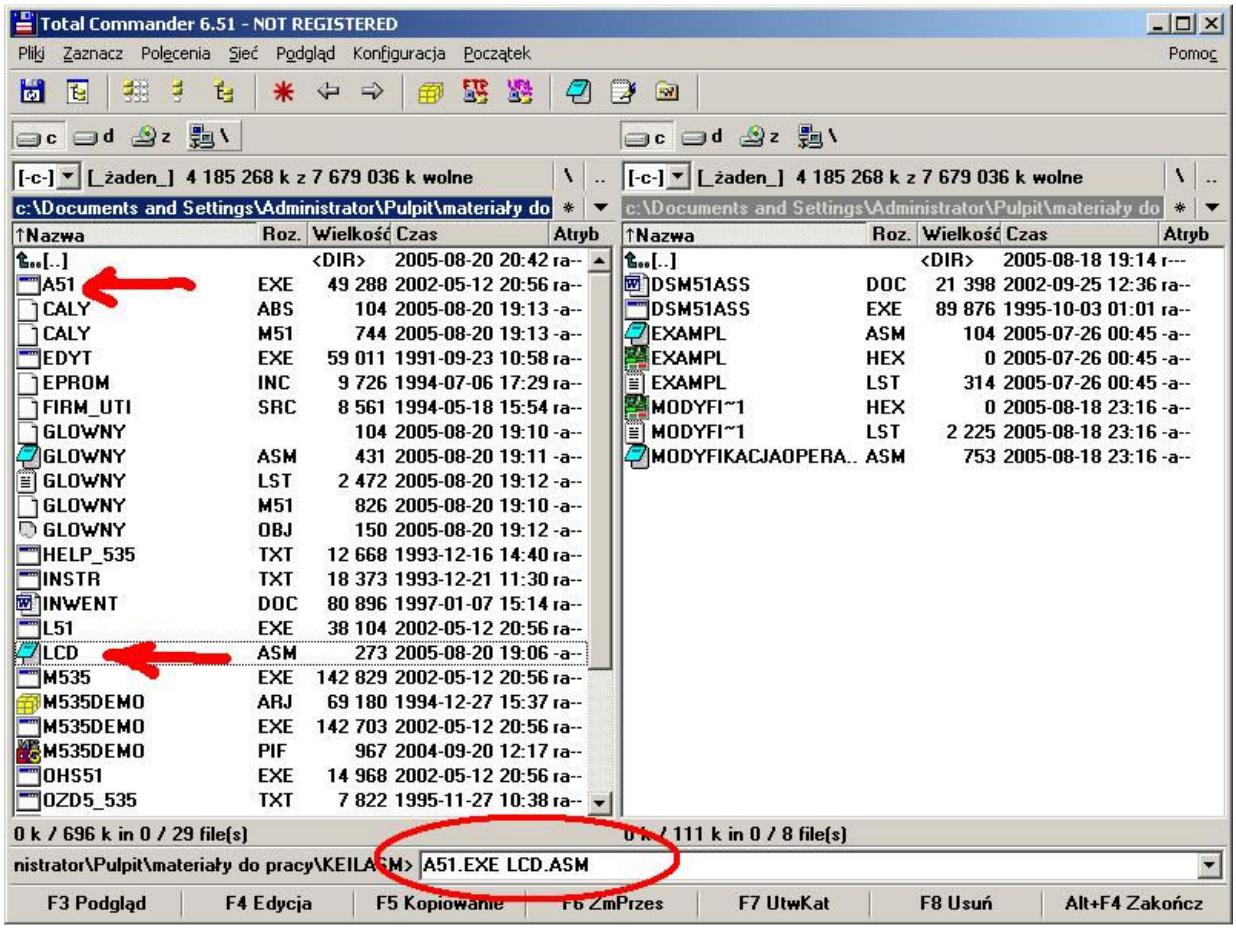
Uzyskanie pliku .HEX możliwe jest po poddaniu pliku .OBJ kolejnemu stopniowi kompilacji przy użyciu programu OHS51.EXE stanowiącego wraz z A51 część całego środowiska programistycznego

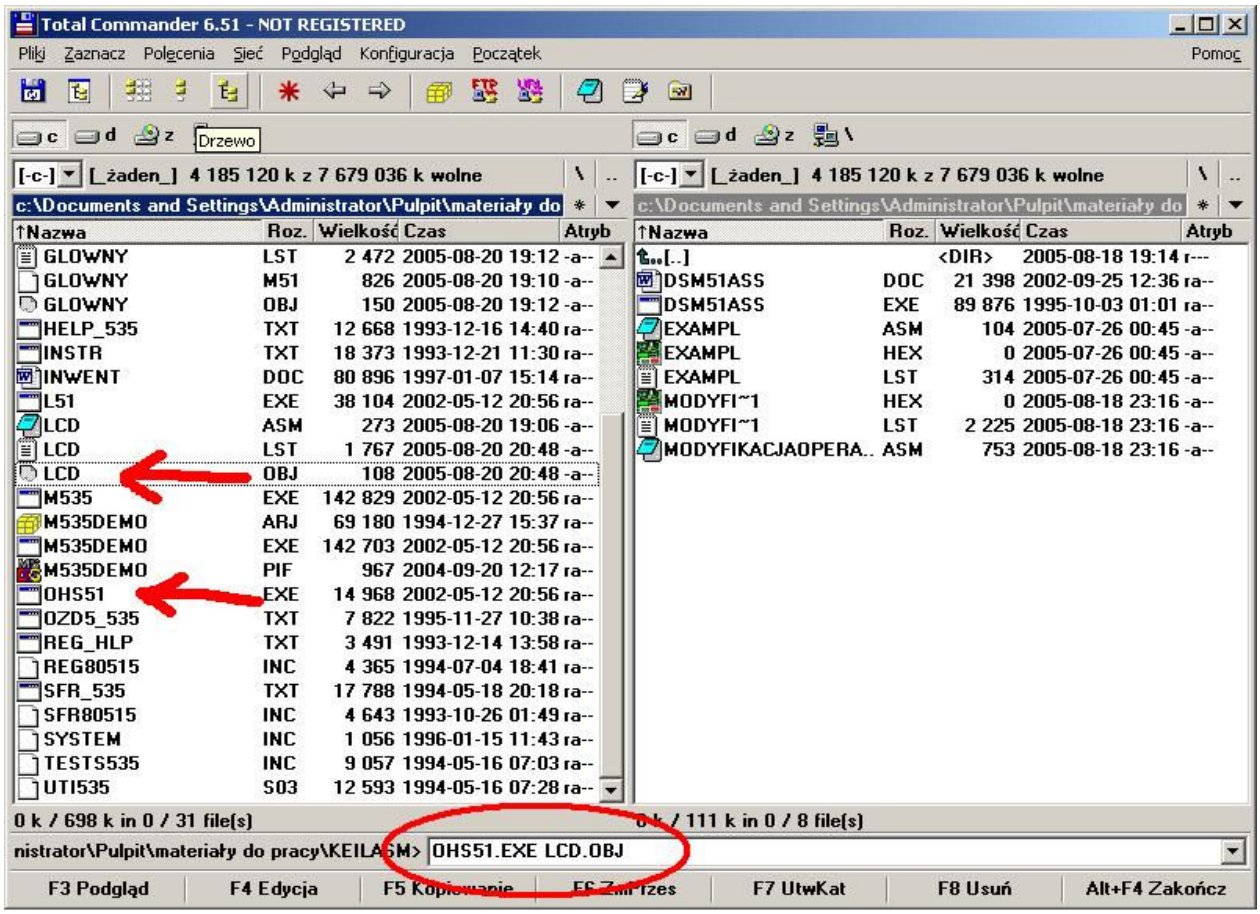
Przykłady.

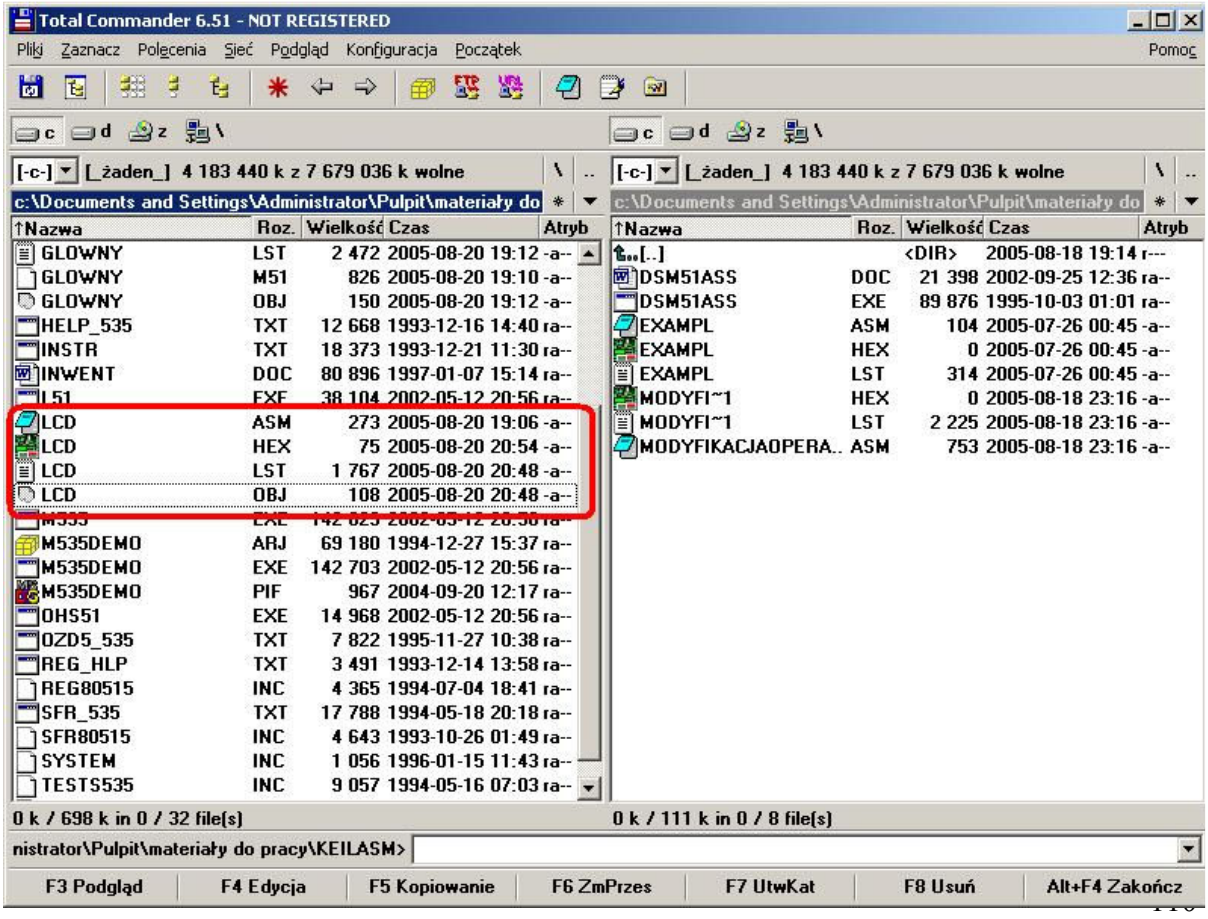
Wywołanie asemblacji pliku o nazwie LCD.ASM (musi znajdować się w tym samym katalogu co A51) przy użyciu DOSNavigatora.

Uzyskany w ten sposób plik LCD.OBJ poddajemy działaniu programu OHS51 tak jak pokazano na rys.

Sytuacja wygląda identycznie w przypadku zastosowania np. TOTALCOMMANDERA dla Windows.




Total Commander 6.51 - NOT REGISTERED
Pliki Zaznacz Polecenia Sieć Podgląd Konfiguracja Początek Pomoc
Drzewo
[-c-] [_żaden_] 4 185 120 k z 7 679 036 k wolne
c:\Documents and Settings\Administrator\Pulpit\materiały do
↑Nazwa Roz. Wielkość Czas Atryb
GLOWNY LST 2 472 2005-08-20 19:12 -a--
GLOWNY M51 826 2005-08-20 19:10 -a--
GLOWNY OBJ 150 2005-08-20 19:12 -a--
HELP_535 TXT 12 668 1993-12-16 14:40 ra--
INSTR TXT 18 373 1993-12-21 11:30 ra--
INWENT DOC 80 896 1997-01-07 15:14 ra--
L51 EXE 38 104 2002-05-12 20:56 ra--
LCD ASM 273 2005-08-20 19:06 -a--
LCD LST 1 767 2005-08-20 20:48 -a--
LCD OBJ 108 2005-08-20 20:48 -a--
M535 EXE 142 829 2002-05-12 20:56 ra--
M535DEMO ARJ 69 180 1994-12-27 15:37 ra--
M535DEMO EXE 142 703 2002-05-12 20:56 ra--
M535DEMO PIF 967 2004-09-20 12:17 ra--
OHS51 EXE 14 968 2002-05-12 20:56 ra--
OZD5_535 TXT 7 822 1995-11-27 10:38 ra--
REG_HLP TXT 3 491 1993-12-14 13:58 ra--
REG80515 INC 4 365 1994-07-04 18:41 ra--
SFR_535 TXT 17 788 1994-05-18 20:18 ra--
SFR80515 INC 4 643 1993-10-26 01:49 ra--
SYSTEM INC 1 056 1996-01-15 11:43 ra--
TESTS535 INC 9 057 1994-05-16 07:03 ra--
UTI535 S03 12 593 1994-05-16 07:28 ra--
[-c-] [_żaden_] 4 185 120 k z 7 679 036 k wolne
c:\Documents and Settings\Administrator\Pulpit\materiały do
↑Nazwa Roz. Wielkość Czas Atryb
[..] <DIR> 2005-08-18 19:14 r---
DSM51ASS DOC 21 398 2002-09-25 12:36 ra--
DSM51ASS EXE 89 876 1995-10-03 01:01 ra--
EXAMPL ASM 104 2005-07-26 00:45 -a--
EXAMPL HEX 0 2005-07-26 00:45 -a--
EXAMPL LST 314 2005-07-26 00:45 -a--
MODYFI~1 HEX 0 2005-08-18 23:16 -a--
MODYFI~1 LST 2 225 2005-08-18 23:16 -a--
MODYFIKACJAOPERA.. ASM 753 2005-08-18 23:16 -a--
0 k / 698 k in 0 / 31 file(s)
0 k / 111 k in 0 / 8 file(s)
nistrator\Pulpit\materiały do pracy\KEILASM> OHS51.EXE LCD.OBJ
F3 Podgląd F4 Edycja F5 Kopiowanie F6 ZmPrzes F7 UtwKat F8 Usuń Alt+F4 Zakończ


Total Commander 6.51 - NOT REGISTERED
Pliki Zaznacz Polecenia Sieć Podgląd Konfiguracja Początek Pomoc
[-c-] [_żaden_] 4 183 440 k z 7 679 036 k wolne
c:\Documents and Settings\Administrator\Pulpit\materiały do
↑Nazwa Roz. Wielkość Czas Atryb
GLOWNY LST 2 472 2005-08-20 19:12 -a--
GLOWNY M51 826 2005-08-20 19:10 -a--
GLOWNY OBJ 150 2005-08-20 19:12 -a--
HELP_535 TXT 12 668 1993-12-16 14:40 ra--
INSTR TXT 18 373 1993-12-21 11:30 ra--
INWENT DOC 80 896 1997-01-07 15:14 ra--
L51 EXE 38 104 2002-05-12 20:56 ra--
LCD ASM 273 2005-08-20 19:06 -a--
LCD HEX 75 2005-08-20 20:54 -a--
LCD LST 1 767 2005-08-20 20:48 -a--
LCD OBJ 108 2005-08-20 20:48 -a--
M535DEMO ARJ 69 180 1994-12-27 15:37 ra--
M535DEMO EXE 142 703 2002-05-12 20:56 ra--
M535DEMO PIF 967 2004-09-20 12:17 ra--
OHS51 EXE 14 968 2002-05-12 20:56 ra--
OZD5_535 TXT 7 822 1995-11-27 10:38 ra--
REG_HLP TXT 3 491 1993-12-14 13:58 ra--
REG80515 INC 4 365 1994-07-04 18:41 ra--
SFR_535 TXT 17 788 1994-05-18 20:18 ra--
SFR80515 INC 4 643 1993-10-26 01:49 ra--
SYSTEM INC 1 056 1996-01-15 11:43 ra--
TESTS535 INC 9 057 1994-05-16 07:03 ra--
[-c-] [_żaden_] 4 183 440 k z 7 679 036 k wolne
c:\Documents and Settings\Administrator\Pulpit\materiały do
↑Nazwa Roz. Wielkość Czas Atryb
[..] <DIR> 2005-08-18 19:14 r---
DSM51ASS DOC 21 398 2002-09-25 12:36 ra--
DSM51ASS EXE 89 876 1995-10-03 01:01 ra--
EXAMPL ASM 104 2005-07-26 00:45 -a--
EXAMPL HEX 0 2005-07-26 00:45 -a--
EXAMPL LST 314 2005-07-26 00:45 -a--
MODYFI~1 HEX 0 2005-08-18 23:16 -a--
MODYFI~1 LST 2 225 2005-08-18 23:16 -a--
MODYFIKACJAOPERA.. ASM 753 2005-08-18 23:16 -a--
0 k / 698 k in 0 / 32 file(s)
0 k / 111 k in 0 / 8 file(s)
nistrator\Pulpit\materiały do pracy\KEILASM>
F3 Podgląd F4 Edycja F5 Kopiowanie F6 ZmPrzes F7 UtwKat F8 Usuń Alt+F4 Zakończ

3.2.3 Format linii programu

Na program w języku asemblera 8051składają się: instrukcje sterujące i rozkazy, gdzie jednemu rozkazowi maszynowemu procesora odpowiada jeden wiersz programu asemblerowego o następującej postaci:

[*ETYKIETA:*] ROZKAZ [*OPERAND 1*] [*,OPERAND 2*] [*,OPERAND 3*] ; [KOMENTARZ]

Parametry w nawiasach kwadratowych występują opcjonalnie. Komentarz (tekst po średniku) jest ignorowany przez program asemblera i służy tylko dokumentowaniu programu oraz uczynieniu go czytelnym dla osób postronnych.

Operandami mogą być:

- adresy programów i lokacji programowych
- adresy wewnętrznej pamięci danych 8051
- adresy pośrednie
- adresy bitów
- stałe
- symbole specjalne (nazwy rejestrów)
- symbole programowe (SEGMENT, EXTERN, …)

3.2.4 Liczby:

Asembler A51 pozwala na wykorzystywanie następujących formatów liczb do reprezentowania programie danych i adresów:

PODSTAWA	PRZYROSTEK	CYFRY	PRZYKŁADY
16 (heksadecymalne)	H,h	0-9, A-F, a-f	1234H, 98h, 0FFFFH, 0A85Dh
10 (dziesiętne)	D, d, lub brak	0-9	1234, 65345D, 20d, 198
8 (ósemkowe)	O, o, Q, q	0-7	177q, 123o, 177777O
2 (dwójkowe)	B, b	0, 1	1111B, 10010000B

Praca na „długich” liczbach niesie niedogodność ich nieczytelności. Dlatego, aby uniknąć pomyłek i ułatwić sobie pracę dopuszcza się stosowanie znaku $ rozdzielającego grupy liczb, np.:

1001$0111$1100$1010B co oznacza 1001011111001010B
5$1$4$2$3 co oznacza 51423

3.2.5 Stałe znakowe:

Przez STAŁĄ ZNAKOWA w asemblerze A51 rozumiemy każdy napis utworzony ze znaków ASCII. Wyrażenie mogą tworzyć co najwyżej dwa znaki zawarte pomiędzy podwójnym apostrofem, niedopuszczalna jest stała znakowa pusta, np.:

‘c’ stała znakowa o wartości 0063H (bardzo przydatne przy zapisywaniu w programie znaków alfanumerycznych, które przy asemblacji automatycznie zamieniane są na odpowiadające im wartości kodu ASCII

‘Ok’ stała znakowa o wartości 4F6BH

” pusta stała znakowa – BŁĄD

‘napis’ stała znakowa, która nie może być użyta wyrażeniach będących instrukcjami mikrokontrolera jako operand (ciąg taki reprezentuje 5-bajtową czyli 40-bitową daną zatem niemożliwe byłoby umieszczenie tego typu danej w żadnym z rejestrów mikrokontrolera ’51 [wszystkie rejestry są 8-bitowymi rejestrami za wyjątkiem 16-bitowego rejestru DPTR]); tego typu ciągi mogą być częścią pliku programu jedynie w postaci tablicy danych w wydzielonym obszarze pamięci programu dostępnym poprzez wykorzystanie instrukcji wymiany danych MOVC), wykorzystywanie tego typu ciągów w programie możliwe jest tylko przy wykorzystaniu dyrektyw asemblera najczęściej dyrektywy DB.

W programie stałe znakowe mogą wyglądać w następujący:

```
Zapisz:   MOV   @R1, #’A’
          ADD   A, #‘B’

Tekst:    DB    ‘Stary misiek mocno chrapie’, 0, ‘a misiowa mocno sapie’, 0
```

3.2.6 Nazwy symboliczne:

Nazwy symboliczne służą w asemblerze do oznaczania np.: stałych, adresów pamięci danych, adresów podprogramów, itd., jednak niektóre nazwy nie mogą być stosowane gdyż występują w funkcji mnemonika rozkazu asemblera. Nazwy symboliczne muszą spełniać następujące warunki:

maksymalna długość;	31 znaków
pierwszy znak nazwy:	duża lub mała litera, znak podkreślenia (‘_‘), znak’?’
kolejne znaki nazwy:	jak pierwszy znak nazwy oraz znaki cyfr

W asemblerze używa się także etykiet, które są nazwami symbolicznymi i pełnią funkcję oznacznika lokacji pamięci programu lub danych. Prezentowane nazwy symboliczne odnoszą się do zasobów programowych i nie mogą być wykorzystywane w innym kontekście:

A	akumulator
R0..R7	rejestry robocze aktualnie wybranego banku rejestrów
DPTR	rejestr wskaźnika DPTR
PC	licznik rozkazów
C	znacznik Carry (przeniesienia)
AB	operand wskaźnika MUL lub DIV
AR0..AR7	w połączeniu z dyrektywą USING oznaczają absolutne adresy rejestrów aktualnie wybranego banku rejestrów.

Znak '$' odczytujemy jako znacznik aktualnej wartości licznika rozkazów, np.:

```
MSG:    DB   LEN, 'TO JEST WIADOMOŚĆ'
LEN     EQU  $-MSG      ; LEN ma wartość równą długości tekstu
Dead:   SJMP $          ; to samo co        SJMP Dead
```

W asemblerze A51 stosuje się następujące operatory w wyrażeniach:

- arytmetyczne: +, -, *, /, MOD, ^, (,),
- binarne: NOT, HIGH, LOW, SHL, SHR, AND, OR, XOR,
- dla porównań: >=, <=, <>, =, <, >.

3.2.7 Oznaczenie adresowania danych:

Adresowanie bezpośrednie rozumiemy jako stosowanie konkretnych nazw symbolicznych lub liczb wpisywanych wprost do rozkazu bądź instrukcji, np.:

```
MOV A, #0E0H        ; załadowanie wartości 0E0H do akumulatora (A)
MOV DPTR, #8000h    ; załadowanie wskaźnika DPTR wartością 8000h
MOV A, 128          ; przesłanie do akumulatora (A) wartości z komórki
                      DATA o adr. 128
MOV R0, #BUFOR      ; załadowanie wartości symbolu BUFOR do rejestru
                      roboczego R0.
```

Przywołane nazwy symboliczne odpowiadają zasobom procesora 8051 (nazwy rejestrów SFR, np.: P0, P1, P2, SCON, SBUB, TMOD, TCON, PSW, ACC, B). Dla oznaczania adresów bitów stosuje się zapis z kropką:

```
SETB    FLAGI.3
CLR     P0.6
```

Możliwe jest także stosowanie symboli (nazw) bitów zasobów 8051 (np.: C, F0, P, SM0, SM1, EA,…)

```
SETB    EA
MOV     C, P1.0
MOV     P, C
```

3.2.8 Dyrektywy asemblera:

Dyrektywy asemblera dzielimy na następujące grupy:

1. Definiujące symbole:
   SEGMENT, EQU, SET, DATA, IDATA, XDATA, BIT, CODE.

2. Rezerwacja i inicjacja pamięci:
   DS, DB, DW, DBIT.

3. Udostępniające nazwy:
   PUBLIC, EXTRN, NAME.

4. Sterujące:
   ORG, END, RSEG, CSEG, DSEG, XSEG, ISEG, BSEG, USING.

3.2.8.1 Dyrektywa SEGMENT:

Celem dyrektywy SEGMENT jest określenie nazwy, typu i sposobu relokacji segmentu relokowanego, tak jak na poniższym schemacie:

Nazwa_Segmentu SEGMENT Typ_Segmentu [Sposób_Relokacji]

Mówiąc inaczej: każdy program lub jego fragment musi być umieszczony (ulokowany) w określonym obszarze pamięci, od określonego adresu początkowego. Adresy te może określić programista przy użyciu określonych dyrektyw asemblera i nie pozwolić na ich dowolną modyfikację przez inne programy (mam tu na myśli program służący do łączenia różnych programów zwanych modułami czyli tzw. linker). W takim przypadku mówimy o segmencie absolutnym, nierelokowalnym, który zawsze musi być umieszczony w pamięci od tego samego adresu np. inicjacja procedury obsługi przerwania.

Inne moduły mogą być zadeklarowane jako relokowalne czyli takie, które program linkera może umieścić w dowolnym niewykorzystywanym przez inne moduły obszarze pamięci. Obszar ten jest wybierany w czasie łączenia modułów po uwzględnieniu wszystkich segmentów absolutnych oraz dyrektyw rezerwujących pamięć do innych celów.

Nazwa_Segmentu musi spełniać wymagania dla nazw symbolicznych.
Typ_Segmentu np.:

CODE obszar pamięci obszaru (PGM)
XDATA obszar zewnętrznej pamięci danych
DATA obszar wewnętrznej pamięci danych (adresy: 0..127)

IDATA obszar wewnętrznej pamięci danych pośredni adresowalny (0..127/255)
BIT obszar wewnętrznej pamięci danych adresowalny bitowo (20H..2FH)

[Sposób_relokacji] może być jednym z: PAGE, INPAGE, INBLOCK, BITADDRESSABLE oraz UNIT.

Przykłady deklaracji z użyciem nazw segmentu stosu:

```
STACK   SEGMENT     IDATA

        RSEG        STACK
        DS          10H

        CSEG
        …
        MOV         SP, #STACK – 1
```

3.2.8.2 Dyrektywy EQU i SET:

Zasady wykorzystania dyrektyw jak dla DSM51ASS.

EQU jest wskaźnikiem zmiennika nazwy symbolicznej, z uwzględnieniem tego, że nazwa może być, w przeciwieństwie do dyrektywy SET, deklarowana wyłącznie jednokrotnie (stała czasu kompilacji). SET pozwala na redefinicję nazwy symbolicznej czyli wielokrotne przypisywanie wartości tej samej nazwie symbolicznej w trakcie pisania programu:

```
LIMIT   EQU     1200
VALUE   SET     LIMIT-200
SERIAL  EQU     SBUF
LICZNIK EQU     R3

VALUE   SET     VALUE / 2
TEMP    SET     VALUE * VALUE
```

3.2.8.3 Dyrektywy BIT, DATA, IDATA, XDATA i CODE:

Podobnie jak dyrektywy EQU i SET służą do przypisywania wartości symbolowi. Pełnią funkcję określeń nazw symbolicznych odnoszących się do odpowiednich obszarów pamięci tzn.

wybranym symbolom wskazanym w dyrektywie programu przypisują wartość będącą adresem wybranego obszaru pamięci. Odpowiednio:

- CODE – nadaje symbolowi wartość (0-65535) będącą adresem w obszarze pamięci programu
- DATA - nadaje symbolowi wartość (0-255) będącą adresem w obszarze wewnętrznej pamięci danych adresowanej bezpośrednio
- IDATA - nadaje symbolowi wartość (0-255) będącą adresem w obszarze wewnętrznej pamięci danych adresowanej pośrednio
- XDATA - nadaje symbolowi wartość (0-65535) będącą adresem w obszarze zewnętrznej pamięci danych
- BIT - nadaje symbolowi wartość (0-255) będącą adresem bitu w obszarze wewnętrznej pamięci danych oraz obszarze SFR adresowanych bitowo

```
FLAGI       SEGMENT     DATA BITADDRESSABLE
KOPIA       SEGMENT     XDATA


            RSEG        SEG_DANE
BASE:       DS 1

ALERT       BIT         BASE.0
CZUJ1       BIT         ALERT+1
ON          BIT         60              ; absolutny adres bitu
OFF         BIT         25H.3           ; absolutny adres bitu

RS_232      DATA        SBUF
BUFOR       DATA        40H
RES         DATA        BUFOR+20

            RSEG        KOPIA
            ORG 100H
REZERW      EQU 5
DATA:       DS  REZERW
CZAS        XDATA       DATA+REZERW

START   CODE        0
INT_EX0     CODE        START + 3
INT_EX1     CODE        START + 0BH
```

3.2.8.4 Dyrektywy DS, DB, DW, DBIT:

Dyrektywy DS, DB, DW, DBIT rezerwują pamięć, np.:

```
R_BUF:    DS   20    ;rezerwacja 20 bitów pamięci w aktualnym segmencie
TIME:     DS   5
```

Należy zwrócić uwagę na to, że dyrektywy DB może inicjować pamięć bajtowo, a dyrektywa DW słowowo (2 bajty), np.:

```
KOMUNIKAT:   DB      'WYBIERZ KLAWISZ'
TEKST:       DB      'KISS MY ...'
TABLICA:     DB      '0123456789ABCDEF'
TABLICA1:    DW      TEKST, 0A88H, 0, 0, 0, 0
             DW      1, 2, 3
```

3.2.8.5 Dyrektywy EXTRN, PUBLIC, NAME:

Niniejszymi dyrektywami posługujemy się w programach wielomodułowych w celu przekazania nazwy symbolicznej do innego modułu **(PUBLIC)** albo dla importowania nazwy z innego modułu **(EXTRN)**. Dyrektywa **NAME** służy dla deklaracji nazwy modułu.

Przy pisaniu modułów programów jak wspominano wielokrotnie wcześniej używa się etykiet, nazw symbolicznych. Musza być one unikalne, charakterystyczne tylko dla jednego modułu jeżeli planujemy w przyszłości jego wykorzystanie do łączenia z innymi programami. Aby umożliwić współpracę takich modułów ze sobą konieczne jest upublicznienie nazw używanych w danym module za pomocą dyrektywy PUBLIC, tzn. wskazanie, że określone w dyrektywie nazwy symboliczne mogą być wykorzystane w innych modułach do odwołania się do bieżącego modułu.

Z drugiej strony pojawienie się nazwy symbolicznej niezdefiniowanej jako etykieta w danym module byłoby traktowane przez asembler jako błąd. Aby tego uniknąć nazwy takie wskazuje się jako zdefiniowane w innym module poprzez użycie dyrektywy EXTRN.

Dla lepszego zrozumienia zagadnienia posłużmy się przykładem.
Tworzymy dwa moduły programów, które będą się wzajemnie odwoływać do siebie:

- Moduł GLOWNY, który realizuje operację sprawdzania klawiatury i kodowania klawisza jako cyfry; moduł ten zostanie ustalony jako absolutny od adresu 0
- Moduł LCD, który realizuje operację wyświetlania pojedynczego znaku na wyświetlaczu LCD; moduł zostanie określony jako relokowalny

Współzależność obu modułów polegać będzie na tym, że w przypadku naciśnięcia klawisza jego kod ma być wyświetlony na wyświetlaczu po czym ponownie powinna być sprawdzana klawiatura.

Moduł GLOWNY.ASM ma postać:

```
KEY1    XDATA    0F000H
KEY2    XDATA    KEY1+1

PUBLIC  KLAWISZ
EXTRN   CODE (DISPLAY)
    CSEG AT 0
```

```
KLAWISZ:                ;zdefiniowanie nazwy etykiety KLAWISZ
    MOV DPTR,#KEY1
    MOVX A,@DPTR
    CJNE A,#0FFH,_0_7
    INC DPTR
    MOVX A,@DPTR
    CJNE A,#0FFH,_8_9
    SJMP KLAWISZ
_0_7:
    CPL A
    MOV R2,#0FFH
    CLR C
ROT:
    RRC A
    INC R2
    JNC ROT
    MOV A,R2
    ADD A,#30H
    LJMP DISPLAY        ;użycie niezdefiniowanej w tym module etykiety DISPLAY
_8_9:
    JNB ACC.0,_8
    JB ACC.1,KLAWISZ
    MOV A,#'9'
    LJMP DISPLAY        ;użycie niezdefiniowanej w tym module etykiety DISPLAY
_8:
    MOV A,#'8'
    LJMP DISPLAY        ;użycie niezdefiniowanej w tym module etykiety DISPLAY
    END
```

Jak widać na zaznaczonym fragmencie w module tym pojawia się zdefiniowana etykieta KLAWISZ, która zostanie wykorzystana w module LCD.ASM dlatego została udostępniona za pomocą dyrektywy PUBLIC. Dyrektywa EXTRN wskazuje, że użyta w tym module nazwa DISPLAY została zdefiniowana w innym module (w tym przypadku w module LCD).

Moduł LCD.ASM ma postać:

```
WR_ROZKAZ   XDATA 8000H
WR_ZNAK     XDATA 8001H
STAN        XDATA 8002H

PUBLIC  DISPLAY
EXTRN   CODE (KLAWISZ)
ZNAK    SEGMENT   CODE
    RSEG ZNAK
DISPLAY:                ;zdefiniowanie nazwy etykiety DISPLAY
    PUSH ACC
    MOV DPTR,#STAN
WAIT:
    MOVX A,@DPTR
    JB ACC.7,WAIT
    MOV DPTR,#WR_ZNAK
    POP ACC
    MOVX @DPTR,A
```

```
LJMP KLAWISZ          ;użycie niezdefiniowanej w tym module etykiety KLAWISZ
END
```

W tym przypadku dyrektywa PUBLIC została użyta do udostępnienia w innych modułach zdefiniowanej w tym module nazwy DISPLAY, natomiast dyrektywa EXTRN pozwoliła importować do tego modułu nazwę KLAWISZ

3.2.8.6 Dyrektywy sterujące:

- END
  Dyrektywa kończąca moduł programu – dyrektywa END w pliku źródłowym jest jednocześnie „rozkazem" dla asemblera sugerującym zakończenie tłumaczenia,

- ORG arg
  Pozwala ustalić wskaźnik w bieżącym segmencie na wartość będącą argumentem tej dyrektywy

- RSEG Nazwa
  Umożliwia ustawienie relokowanego segmentu Nazwa jako bieżącego,

- CSEG, DSEG, XSEG, ISEB, BIEG:
  Ich funkcją jest ustalenie absolutnego segmentu (odpowiednio pamięci programu, pamięci danych, zewnętrznej pamięci danych, pamięci danych adresowanych pośrednio, obszaru pamięci danych dostępnego bitowo) jako bieżącego – dyrektywa może wystąpić z opcjonalnym parametrem AT i absolutnym adresem obszaru pamięci, np.:

```
DATA_SEG1     SEGMENY     DATA
CODE_SEG1     SEGMENT     CODE

              BSEG    AT  70H       ; segment absolutny
DECMODE:      DBIT    1
CHARMODE:     DBIT    1

              RSEG    DATA_SEG1     ; segment relokowalny
TOTAL:        SD      1
COUNT:        DS      2

              RSEG    CODE_SEG1
START:
              MOV SP, #STACK-1
              …
              END
```

3.2.8.7 Dyrektywa USING:

Daje możliwość wyboru bieżącego banku rejestrów dla nazw symbolicznych AR0..AR7, np.:

```
USING   3
PUSH    AR2     ; push dotyczy rejestru R2 w 3 banku rejestrów

USING   0
PUSH    AR0     ; push dotyczy rejestru R0 w 0 banku rejestrów
```

3.2.9 Najważniejsze instrukcje sterujące asemblera A51:

Plik źródłowy jest docelowym miejscem instrukcji asemblera, które są tam umieszczane w celu określenia dodatkowych warunków tłumaczenia.
Instrukcje występują w wierszach rozpoczynających się znakiem $:

- DEBUG/NODEBUG

  Włączanie opcji DEBUG spowoduje dołączenie do pliku object programu wszystkich nazw symbolicznych (mogą być wykorzystane przez program uruchamiający),

- ERRORPRINT/NOERRORPRINT

  Można podać opcjonalny argument w postaci nazwy zbioru, który będzie zawierał listę błędów tłumaczenia programu źródłowego, np.:

  $ ERRORPRINT (BLEDY.ERR)

- OBJECT

  Można zdefiniować nazwę zbioru wynikowego [object] (domyślnie przyjmowana jest nazwa taka jak dla pliku .ASM):

  $ OBJECT (nazwa.obj)

- PRNT/NOPRINT

  Zdefiniowanie nazwy zbioru LST (domyślnie przyjmowana jest nazwa jak dla pliku .ASM) albo rezygnacja z tworzenia tego zbioru (lepiej tego nie robić aby nie pozbawiać się informacji o rodzaju i miejscu wystąpienia ewentualnych błędów)

- INCLUDE

Umożliwia włączenie do tłumaczonego pliku innego pliku źródłowego Zastosowanie tego polecenia pozwala rozszerzyć zakres predefiniowanych nazw rejestrów dla różnych typów mikrokontrolerów np. port P5, który występuje tylko w wybranych układach

$ INCLUDE (REG80535.INC) – rozszerza zakres predefiniowanych symboli dla mikrokontrolera 80535

Zawartość pliku REG80535.INC:

```
;Siemens 80515
;Special Function Register definition for Archimedes A8051 Assembler
;

I3CC0   EQU  P1.0    ;Ext Int 3 / Capture Compare 0 pin
I4CC1   EQU  P1.1    ;Ext Int 4 / Capture Compare 1 pin
I5CC2   EQU  P1.2    ;Ext Int 5 / Capture Compare 2 pin
I6CC3   EQU  P1.3    ;Ext Int 6 / Capture Compare 3 pin
INT2    EQU  P1.4    ;External Interrupt 2 input pin
T2EX    EQU  P1.5    ;Timer 2 External reload trigger input pin
CLKOUT  EQU  P1.6    ;System Clock output pin
T2      EQU  P1.7    ;Timer 2 input pin

IEN0    EQU  0A8H    ;Interrupt Enable Register 0

; The following symbols are included to show complete registers,
; but are commented out to avoid re-defining symbols which are
; built into the Archimedes A8051 assembler
;
;EX0     EQU  IEN0.0  ;External Interrupt 0 enable
;ET0     EQU  IEN0.1  ;Timer 0 overflow interrupt enable
;EX1     EQU  IEN0.2  ;External Interrupt 1 enable
;ET1     EQU  IEN0.3  ;Timer 1 overflow interrupt enable
;ES      EQU  IEN0.4  ;Serial port interrupt enable
ET2     EQU  IEN0.5  ;Timer 2 overflow interrupt enable
WDT     EQU  IEN0.6  ;Watchdog timer reset flag
EAL     EQU  IEN0.7  ;Master interrupt enable

IEN1    EQU  0B8H    ;Interrupt Enable Register 1
EADC    EQU  IEN1.0  ;A/D converter interrupt enable
EX2     EQU  IEN1.1  ;External interrupt 2 interrupt enable
EX3     EQU  IEN1.2  ;External interrupt 3 interrupt enable
EX4     EQU  IEN1.3  ;External interrupt 4 interrupt enable
EX5     EQU  IEN1.4  ;External interrupt 5 interrupt enable
EX6     EQU  IEN1.5  ;External interrupt 6 interrupt enable
SWDT    EQU  IEN1.6  ;Watchdog timer start/reset bit
EXEN2   EQU  IEN1.7  ;Timer 2 external reload interrupt enable

IP0     EQU  0A9H    ;Interrupt Priority Register 0
IP1     EQU  0B9H    ;Interrupt Priority Register 1

IRCON   EQU  0C0H    ;Interrupt Request Control Register
IADC    EQU  IRCON.0 ;A/D converter interrupt request flag
IEX2    EQU  IRCON.1 ;External interrupt 2 edge flag
IEX3    EQU  IRCON.2 ;External interrupt 3 edge flag
IEX4    EQU  IRCON.3 ;External interrupt 4 edge flag
IEX5    EQU  IRCON.4 ;External interrupt 5 edge flag
IEX6    EQU  IRCON.5 ;External interrupt 6 edge flag
TF2     EQU  IRCON.6 ;Timer 2 overflow flag
EXF2    EQU  IRCON.7 ;Timer 2 external reload flag
```

```
CCEN    EQU  0C1H    ;Compare/Capture Enable Register
CCL1    EQU  0C2H    ;Compare/Capture Register 1, Low Byte
CCH1    EQU  0C3H    ;Compare/Capture Register 1, High Byte
CCL2    EQU  0C4H    ;Compare/Capture Register 2, Low Byte
CCH2    EQU  0C5H    ;Compare/Capture Register 2, High Byte
CCL3    EQU  0C6H    ;Compare/Capture Register 3, Low Byte
CCH3    EQU  0C7H    ;Compare/Capture Register 3, High Byte

T2CON   EQU  0C8H    ;Timer 2 control register
T2I0    EQU  T2CON.0 ;Timer 2 input selection bit 0
T2I1    EQU  T2CON.1 ;Timer 2 input selection bit 1
T2CM    EQU  T2CON.2 ;Compare mode bit
T2R0    EQU  T2CON.3 ;Timer 2 reload mode select bit 0
T2R1    EQU  T2CON.4 ;Timer 2 reload mode select bit 1
I2FR    EQU  T2CON.5 ;External Interrupt 2 Rise/Fall select bit
I3FR    EQU  T2CON.6 ;External Interrupt 3 Rise/Fall select bit
T2PS    EQU  T2CON.7 ;Prescaler select bit

CRCL    EQU  0CAH    ;COMPARE/RELOAD/CAPTURE REGISTERS
CRCH    EQU  0CBH    ;...
TL2     EQU  0CCH    ;Timer 2 counter low byte
TH2     EQU  0CDH    ;Timer 2 counter high byte

ADCON   EQU  0D8H    ;A/D Converter Control Register
MX0     EQU  ADCON.0 ;A/D input channel selection bit 0
MX1     EQU  ADCON.1 ;A/D input channel selection bit 1
MX2     EQU  ADCON.2 ;A/D input channel selection bit 2
ADM     EQU  ADCON.3 ;A/D conversion mode
BSY     EQU  ADCON.4 ;A/D busy bit
CLK     EQU  ADCON.6 ;System clock enable
BD      EQU  ADCON.7 ;Baud rate enable

P6      EQU  0D9H    ;A/D Converter Data Register
ADDAT   EQU  0D9H    ;A/D Converter Data Register
DAPR    EQU  0DAH    ;A/D Converter Program Register

P4      EQU  0E8H    ;Port 4
P5      EQU  0F8H    ;Port 5

ET2I    EQU  02BH    ;Timer 2 interrupt vector
IADCV   EQU  043H    ;A/D converter interrupt vector
IEX2V   EQU  04BH    ;External Interrupt 2 vector
IEX3V   EQU  053H    ;External Interrupt 3 vector
IEX4V   EQU  05BH    ;External Interrupt 4 vector
IEX5V   EQU  063H    ;External Interrupt 5 vector
IEX6V   EQU  06BH    ;External Interrupt 6 vector
```

$ INCLUDE (SYSTEM.INC) – pozwala dołączyć do pliku źródłowego nazwy podprogramów charakterystycznych dla zestawu dydaktycznego

Zawartość pliku SYSTEM.INC:

```
;Siemens 80515
;Special Function Register definition for Archimedes A8051 Assembler
;

I3CC0   EQU  P1.0    ;Ext Int 3 / Capture Compare 0 pin
I4CC1   EQU  P1.1    ;Ext Int 4 / Capture Compare 1 pin
I5CC2   EQU  P1.2    ;Ext Int 5 / Capture Compare 2 pin
I6CC3   EQU  P1.3    ;Ext Int 6 / Capture Compare 3 pin
INT2    EQU  P1.4    ;External Interrupt 2 input pin
T2EX    EQU  P1.5    ;Timer 2 External reload trigger input pin
CLKOUT  EQU  P1.6    ;System Clock output pin
```

```
T2      EQU  P1.7    ;Timer 2 input pin

IEN0    EQU  0A8H    ;Interrupt Enable Register 0

; The following symbols are included to show complete registers,
; but are commented out to avoid re-defining symbols which are
; built into the Archimedes A8051 assembler
;
;EX0     EQU  IEN0.0  ;External Interrupt 0 enable
;ET0     EQU  IEN0.1  ;Timer 0 overflow interrupt enable
;EX1     EQU  IEN0.2  ;External Interrupt 1 enable
;ET1     EQU  IEN0.3  ;Timer 1 overflow interrupt enable
;ES      EQU  IEN0.4  ;Serial port interrupt enable
ET2     EQU  IEN0.5  ;Timer 2 overflow interrupt enable
WDT     EQU  IEN0.6  ;Watchdog timer reset flag
EAL     EQU  IEN0.7  ;Master interrupt enable

IEN1    EQU  0B8H    ;Interrupt Enable Register 1
EADC    EQU  IEN1.0  ;A/D converter interrupt enable
EX2     EQU  IEN1.1  ;External interrupt 2 interrupt enable
EX3     EQU  IEN1.2  ;External interrupt 3 interrupt enable
EX4     EQU  IEN1.3  ;External interrupt 4 interrupt enable
EX5     EQU  IEN1.4  ;External interrupt 5 interrupt enable
EX6     EQU  IEN1.5  ;External interrupt 6 interrupt enable
SWDT    EQU  IEN1.6  ;Watchdog timer start/reset bit
EXEN2   EQU  IEN1.7  ;Timer 2 external reload interrupt enable

IP0     EQU  0A9H    ;Interrupt Priority Register 0
IP1     EQU  0B9H    ;Interrupt Priority Register 1

IRCON   EQU  0C0H    ;Interrupt Request Control Register
IADC    EQU  IRCON.0 ;A/D converter interrupt request flag
IEX2    EQU  IRCON.1 ;External interrupt 2 edge flag
IEX3    EQU  IRCON.2 ;External interrupt 3 edge flag
IEX4    EQU  IRCON.3 ;External interrupt 4 edge flag
IEX5    EQU  IRCON.4 ;External interrupt 5 edge flag
IEX6    EQU  IRCON.5 ;External interrupt 6 edge flag
TF2     EQU  IRCON.6 ;Timer 2 overflow flag
EXF2    EQU  IRCON.7 ;Timer 2 external reload flag

CCEN    EQU  0C1H    ;Compare/Capture Enable Register
CCL1    EQU  0C2H    ;Compare/Capture Register 1, Low Byte
CCH1    EQU  0C3H    ;Compare/Capture Register 1, High Byte
CCL2    EQU  0C4H    ;Compare/Capture Register 2, Low Byte
CCH2    EQU  0C5H    ;Compare/Capture Register 2, High Byte
CCL3    EQU  0C6H    ;Compare/Capture Register 3, Low Byte
CCH3    EQU  0C7H    ;Compare/Capture Register 3, High Byte

T2CON   EQU  0C8H    ;Timer 2 control register
T2I0    EQU  T2CON.0 ;Timer 2 input selection bit 0
T2I1    EQU  T2CON.1 ;Timer 2 input selection bit 1
T2CM    EQU  T2CON.2 ;Compare mode bit
T2R0    EQU  T2CON.3 ;Timer 2 reload mode select bit 0
T2R1    EQU  T2CON.4 ;Timer 2 reload mode select bit 1
I2FR    EQU  T2CON.5 ;External Interrupt 2 Rise/Fall select bit
I3FR    EQU  T2CON.6 ;External Interrupt 3 Rise/Fall select bit
T2PS    EQU  T2CON.7 ;Prescaler select bit

CRCL    EQU  0CAH    ;COMPARE/RELOAD/CAPTURE REGISTERS
CRCH    EQU  0CBH    ;...
TL2     EQU  0CCH    ;Timer 2 counter low byte
TH2     EQU  0CDH    ;Timer 2 counter high byte

ADCON   EQU  0D8H    ;A/D Converter Control Register
```

```
MX0     EQU  ADCON.0 ;A/D input channel selection bit 0
MX1     EQU  ADCON.1 ;A/D input channel selection bit 1
MX2     EQU  ADCON.2 ;A/D input channel selection bit 2
ADM     EQU  ADCON.3 ;A/D conversion mode
BSY     EQU  ADCON.4 ;A/D busy bit
CLK     EQU  ADCON.6 ;System clock enable
BD      EQU  ADCON.7 ;Baud rate enable

P6      EQU  0D9H    ;A/D Converter Data Register
ADDAT   EQU  0D9H    ;A/D Converter Data Register
DAPR    EQU  0DAH    ;A/D Converter Program Register

P4      EQU  0E8H    ;Port 4
P5      EQU  0F8H    ;Port 5

ET2I    EQU  02BH    ;Timer 2 interrupt vector
IADCV   EQU  043H    ;A/D converter interrupt vector
IEX2V   EQU  04BH    ;External Interrupt 2 vector
IEX3V   EQU  053H    ;External Interrupt 3 vector
IEX4V   EQU  05BH    ;External Interrupt 4 vector
IEX5V   EQU  063H    ;External Interrupt 5 vector
IEX6V   EQU  06BH    ;External Interrupt 6 vector
```

4 Zakończenie... czyli ciąg dalszy nastąpi.

Przedstawione w tym opracowaniu treści maja posłużyć wprowadzeniu do zagadnienia programowania systemów mikroprocesorowych wykorzystujących mikrokontrolery rodziny MCS'51. Praca zostanie teraz poddana surowej i mamy nadzieję sprawiedliwej oraz konstruktywnej ocenie tych, do których jest skierowana czyli uczniów Technikum Elektrycznego w Zespole Szkół Nr 6 w Jastrzębiu Zdroju.

Kolejne etapy jej powstawania dotyczyć już będą konkretnych przykładów zastosowań i będą one sukcesywnie powstawać w oparciu o doświadczenia autorów wynikające z możliwości realizacji oraz potrzeb wynikających z reformy systemu edukacji zawodowej.

Oby ta praca miała sens!!!