Mikrokontrolery

Wojciech Kordecki
Zakład Pomiarowej i Medycznej Aparatury Elektronicznej
Wydział Podstawowych Problemów Techniki
Politechnika Wrocławska

Wrocław 2003

1 Wstęp

Materiały do wykładu i listy zadań na laboratorium będą umieszczane na stronie

`http://www.im.pwr.wroc.pl/~kordecki/`

Pytania do wykładowcy można zadawać również elektronicznie:

`kordecki@im.pwr.wroc.pl`

Literatura polecana

[1] P. Gałka, P. Gałka. *Podstawy programowania mikrokontrolerów.* Mikon, 1995.

[2] J. Majewski, K. Kardach. *Programowanie mikrokontrolerów z serii 8x51 w języku C.* Oficyna Wydawnicza P. Wr., Wrocław, 2002.

[3] R. Pełka. *Mikrokontrolery.* WKŁ, Warszawa, 1999.

[4] A. Rydzewski. *Mikrokontrolery jednoukładowe rodziny MCS – 51.* WNT, Warszawa, 1991.

Zawartość tematyczna poszczególnych godzin wykładowych

1. Architektura procesora 8051/52. Pamięć wewnętrzna SFR, pamięć programu, wybrane instrukcje.
2. Asembler, lista instrukcji: przesyłania, skoki, podprogramy, pętle, format Intel HEX
3. Pamięć zewnętrzna, porty, współpraca z urządzeniami zewnętrznymi.
4. Instrukcje operacji arytmetycznych i logicznych, kod U2, kod BCD, formaty liczb zmiennoprzecinkowych.
5. Przerwania i procedury obsługi przerwań, realizacja przerwań sprzętowych.

6. Programowanie liczników i zegarów. Programowanie portów do komunikacji szeregowej.

7. Techniki oprogramowania urządzeń do komunikacji mikrokontrolera z użytkownikiem (klawiatura, pola odczytowe)

8. Techniki oprogramowania sterowników działających w czasie rzeczywistym.

9. Komunikacja mikrokontrolera z wybranymi urządzeniami zewnętrznymi (przetworniki A/C, C/A i inne)

10. Programowanie mikrokontrolerów w języku wysokiego poziomu C lub Pascal.

11. Procesory bazujące na architekturze procesora 8051, przegląd rozwiązań.
12. Mikrokontrolery 16 i 32 bitowe, przykłady zastosowania.
13. Inne rozwiązania mikrokontrolerów (RISC, DSP)
14. Test sprawdzający nabytą przez studentów wiedzę.

Laboratorium, projekt – zawartość tematyczna

1. Środowisko do programowania mikrokontrolerów bazujących na architekturze procesora 8051: asembler, linker, debuger. Praktyczne zapoznanie się z środowiskiem programisty.

2. Praktyczne sprawdzenie zrozumienia zasady działania instrukcji wymiany danych między rejestrami oraz sposobów adresowania pamięci mikrokontrolera Realizacja i testowanie prostego algorytmu segregowania danych

3. Praktyczne testowanie instrukcji operacji na stosie i skoków warunkowych

4. Realizacja i testowanie programu z wieloma procedurami.

5. Testowanie instrukcji operacji logicznych i arytmetycznych, opracowanie i testowanie programu poszukiwania wartości maksymalnych.

6. Realizacja i testowanie algorytmów zamiany kodów.

7. Program obsługi klawiatury matrycowej i pola odczytowego LCD.

8. Instrukcje przerwań i ich wykorzystanie, realizacja i testowanie programu realizującego reakcje na zdarzenia zewnętrzne., i warunki programowe.

9. Programowanie liczników, praktyczna realizacja generatora przebiegów prostokątnych, obsługa portu szeregowego.

10. Praktyczne testowanie instrukcji wej/wyj, opracowanie prostego programu obsługi portów I/O: transmisji równoległej

11. Oprogramowanie komunikacji z urządzeniami zewnętrznymi wyposażonymi w interfejs SPI.

12. Poznanie praktyczne podstawowych zasad programowania mikrokontrolerów 8051 w języku C.

13. Oprogramowanie prostego algorytmu filtru cyfrowego. Na poziomie języka C i asemblera.

14. Sprawdzian zdobytej wiedzy – poprzez realizacje zadanego oprogramowania mikrokontrolera.

2 Procesor Intel 8051

2.1 Ogólne informacje o procesorach rodziny 8051

Procesory rodziny 8051 są ośmiobitowe, tzn. operują wyłącznie na danych ośmiobitowych. Produkowane są przez firmę Intel. Mają liczne wersje rozszerzone i zubożone.
Ich odpowiedniki są produkowane przez liczne firmy, m.in. Siemens, Philips, Atmel, AMD i in.

Format liczb:

- dziesiętny (decymalny) – liczba bez żadnych dodatkowych oznaczeń, np. 42,
- dwójkowy (binarny) – cyfry 0 i 1, na końcu litera B, np. $101010\text{B} = 42$,
- szesnastkowy (hexadecymalny) – cyfry $0, \ldots, 9, A, B, C, D, E, F$, na końcu litera H, gdzie $A = 10$, $B = 11$, $C = 12$, $D = 13$, $E = 14$, $F = 15$, np. $2A\text{H} = 42$.

Dla uniknięcia nieporozumień lepiej mówić „dwa-dwa szesnastkowo” (czyli 34), niż „dwadzieścia dwa szesnastkowo”.

Procesory Intel 8051/52 są jednoukładowe, tzn. w jednym układzie posiadają:

- wewnętrzna pamięć danych (Internal RAM)
- wewnętrzna pamięć programu (ROM)
- rejestry,
- układ czasowo-licznikowy,
- linie wejścia-wyjścia,
- port szeregowy,
- układ przerwań.

Pamięć wewnętrzna.

- Można adresować nie tylko bajty, ale i bity.
- Można adresować do 256 bajtów (od 00h do FFh) i tyle samo bitów.
- Pierwsze 128 bajtów i 128 bitów znajduje się w pamięci danych, a pozostałe w w rejestrach specjalnych SFR (special function registers).

Program jest umieszczany w pamięci programu wewnętrznej lub zewnętrznej (PROM lun EPROM). Zewnętrzna pamięć programu (kodu) może mieć pojemność do 64 kB.
Dane są umieszczane w wewnętrznej lub zewnętrznej pamięci danych. Zewnętrzna pamięć danych (External RAM) może mieć pojemność do 64 kB.
Każdy z typów pamięci jest obsługiwany w inny sposób. Najwygodniejsza (zapis i odczyt) jest obsługa wewnętrzej pamięci danych. Mniej wygodna i wolniejsza (zapis i odczyt) jest obsługa zewnętrzej pamięci danych. Najmniej wygodna jest odczyt danych z pamięci programu.

2.2 Pamięć wewnętrzna i SFR

Pamięć wewnętrzna o adresach od 00h do 7Fh ma postać

Zakres	Nazwa	przeznaczenie
00h – 07h	R0 – R7	Rejestry użytkowe 1
08h – 0Fh	R0 – R7	Rejestry użytkowe 2
10h – 17h	R0 – R7	Rejestry użytkowe 3
18h – 1Fh	R0 – R7	Rejestry użytkowe 4
20h – 2Fh		Obszar adresowany bitowo
30h – 7Fh		Obszar danych

W obszarze danych umieszcza się stos.

Obszar 00h – 2Fh może być również traktowany tak, jak obszar 30h – 7Fh.

Niektóre rejestry z obszaru SFR.

- Akumulator A lub Acc – najbardziej uniwersalny rejestr.
- B – rejestr pomocniczy, używany m.in. przy dzieleniu i mnożeniu.
- DPH i DPL, razem dają DPTR – wskaźnik danych.
- SP – wskaźnik stosu.
- PSW – słowo stanu programu.

2.3 Pierwsze kroki w asemblerze

Rozkazy nie mogą zaczynać się od początku wiersza.
Od początku wiersza zaczyna się etykieta, po której następuje znak ':'
Tekst po znaku ';' nie jest przetwarzany.
Najczęściej spotykanymi rozkazami są MOV, SJMP, LJMP.
Rozkaz MOV ma dwa argumenty: pierwszy dokąd ładować (adres komórki pamięci), drugi skąd lub co ładować.
Rozkaz SJMP ma jeden argument, którym może być etykieta, powoduje skok do niej, ale nie dalej niż o 128 bajtów kodu.
Rozkaz LJMP działa podobnie, ale nie ma ograniczenia długości skoku.
Liczba n w rozkazie oznacza adres rejestru, a $\#n$ oznacza liczbę (wartość).

Dyrektywa ORG, której argumentem jest adres powoduje, że następujący po nim program jest asemblowany od tego adresu.

```
          LJMP start
          ORG 03h
          SJMP alarm    ;nie jest zwykle wykonywany (przerwanie)
          ORG 20H
start:    MOV 10h,#10   ;liczba 10 pod adres 16
          MOV A,10h     ;teraz do akumulatora
          MOV 11h,A     ;akumulator pod adres 17
          SJMP start    ;i od początku
alarm:    SJMP $
```

Inne często stosowane rozkazy:
INC – zwiększa o 1 zawartość komórki pamięci o adresie będącym argumetem rozkazu.
DEC – zmniejsza j.w.
JZ – skok jak w SJMP, ale tylko wtedy, gdy zawartość Acc jest równa zeru,
JNZ – j.w., ale gdy zawartość Acc jest różna od zera.

```
            LJMP start
            ORG 03h
            SJMP alarm
            ORG 20H
start:      MOV A,#250      ;liczba 250 do akumulatora
loop:       INC A           ;zwiększenie akumulatora o 1
            JNZ loop        ;aż się licznik przekręci
alarm:      SJMP $          ;do nieskończoności
```

Etykieta `$` oznacza „właśnie ta linijka programu”, czyli `SJMP $` powoduje „kręcenie się w miejscu” programu.

Uwaga. $255 + 1 = 256 = FF\text{H} \equiv 0$.

3 Asembler

3.1 Uwagi ogólne

Typowa linia programu (na przykładzie asemblera Dsm51ass):
`[<etykieta>] [<rozkaz>] [<operandy>] [;<komentarz>]`

`<etykieta>` – zaczyna się od litery lub znaku podkreślenia `_`, i może zawierać dowolną kombinację liter, cyfr i podkreśleń. Pierwszy znak etykiety jest pierwszym znakiem w linii. Jeśli etykieta jest zakończona dwukropkiem to jej wartością jest pozycja w kodzie źródłowym, czyli adres rozkazu z tej linii programu. Etykiety (symbole) stosowane z dyrektywami nadającymi im wartość nie są zakończone dwukropkiem.

`<rozkaz>` – mnemonik kodu maszynowego procesora, dyrektywa asemblera lub makro.

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

`<komentarz>` - wszystkie znaki występujące po średniku są ignorowane przez asembler.

Poszczególne pola linii programu są oddzielane co najmniej jednym znakiem spacji lub tabulacji. W programie mogą występować puste linie lub linie zawierające wyłącznie komentarz.

Stała liczbowa musi zaczynać się od cyfry.

Typ	Składnia	Przykład
Dziesiętny	`<cyfry>`	`125`
Szesnastkowy	`<cyfra><cyfry szesnastkowe>H`	`0FFFFH`
Ósemkowy	`<cyfry ósemkowe>O`	`77770`
Binarny	`<cyfry binarne>B`	`10101B`
Znakowy	`'<znak>'`	`'A'`

Stałą jest rozkaz NOP – tam gdzie został umieszczony, wstawia zero w kodzie programu w pamięci programu. Jego wykonanie trwa jeden cykl.

Dyrektywy danych (w różnych asemblerach różnie):

`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.

Przykład.

```
ok_message DB 'OK'
two_int DW 0403h,0201h
const_zero EQU 0
bit_1 BIT 1
```

Predefiniowanym symbolem jest ACC, który jest równy E0h, jest to adres akumulatora w SFR.

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.

Przykład.

```
1         0000   const_next EQU 0
2         [01]         IF const_next
3                            SJMP next ;jezeli const_next<>0
4         [01]         ELSE
5  0000:  80 FE               SJMP $    ;jezeli const_next=0
6         [00]         ENDIF
7  0002:  00                  NOP
8  0003:  80 FE   next:       SJMP $
9                           END
```

```
1         0001   const_next EQU 1
2         [01]       IF const_next
3  0000: 80 01             SJMP next ;jezeli const_next<>0
4         [01]       ELSE
5                         SJMP $    ;jezeli const_next=0
6         [00]       ENDIF
7  0002: 00                NOP
8  0003: 80 FE  next:      SJMP $
9                        END
```

Dyrektywy makrodefinicji:

`MACRO` – początek definicji makra,

`ENDM` – koniec definicji makra.

W makrodefinicji można umieszczać parametry, nie można lokalnych etykiet (w tym asemblerze).

Przykład.

```
five  MACRO ptr ;umieszcza liczbę 5 pod adresem ptr
          MOV R0,ptr
          MOV @R0,#5

      ENDM
          five #5
          five #6
          five #7

          END
```

Po zasemblowaniu otrzymujemy

```
     1                    five  MACRO ptr ;umieszcza liczbe 5 pod
     2                              MOV R0,ptr
     3                              MOV @R0,#5
     4                          ENDM
     5
     6                              five #5
>    1  0000: 78 05                 MOV R0,#5
>    2  0002: 76 05                 MOV @R0,#5
>    3                          ENDM
     7                              five #6
>    1  0004: 78 06                 MOV R0,#6
>    2  0006: 76 05                 MOV @R0,#5
```

```
>     3                         ENDM
      8                             five #7
>     1  0008: 78 07                MOV R0,#7
>     2  000A: 76 05                MOV @R0,#5
>     3                         ENDM
      9
     10                             END
```

3.2 Przesłania dla pamięci wewnętrznej

Rozkaz przesłania danej jednobajtowej MOV.
Pierwszy argument wskazuje dokąd należy przesłać bajt. Może to być: akumulator A, adres bezpośredni, rejestr z aktywnego zbioru rejestrów (od R0 do R7), adres pośredni (zawarty w rejestrze R0 lub R1 z aktywnego zbioru rejestrów, czyli @R0 lub @R1).
Drugi argument wskazuje co należy przesłać. Może to być jak poprzednio: akumulator A, adres bezpośredni, rejestr, adres pośredni lub liczba (postać #n).
Nie można przesłać w jednym rozkazie umieścić dwukrotnie symbolu rejestru.

Nielegalne są też rozkazy

```
MOV A,ACC
MOV ACC,A
```

Rozkazy przesłań do i z akumulatora są wykonywane szybciej od wielu innych. W szczególności rozkaz

```
MOV ACC,R0
```

zajmuje 2 cykle, a rozkaz

```
MOV A,R0
```

tylko jeden cykl.

Zestaw aktywnych rejestrów Rr jest wybierany przez dwubitową liczbę RS1 RS0 w rejestrze PSW.

Rozkazy wymiany XCH: pierwszym argumentem jest zawsze A, drugim adres bezpośredni, rejestr lub adres pośredni.
Wszystkie przesłania i wymiany operują na danych ośmiobitowych, z wyjątkiem rozkazu

```
MOV DPTR,#nn
```

gdzie #nn jest liczbą szesnastobitową.

3.3 Skoki i rozkazy sterujące

Skok długi LJMP – jego argumentem jest liczba szesnatobitowa – adres w pamięci programu. Jest ona wpisywana do PC i program wykona rozkaz umieszczony pod tym adresem. Jako argument LJMP należy umieścić etykietę.

Rozkaz

```
JMP @A+DPTR
```

spowoduje skok pod adres umieszczony w DPTR i zwiększony o zawartość akumulatora.

Rozkaz AJMP ma znaczenie historyczne i nie należy go używać.

Skoki krótkie – tylko o wielkość w zakresie $-128, 127$. Jako argument należy podać etykietę. Asembler sam obliczy wielkość skoku lub da komunikat błędu o za długim skoku.
SJMP – skok bezwarunkowy,
JC – jak SJMP, ale zostanie wykonany, gdy bit CY jest ustawiony.
JNC – jak JC, ale gdy CY jest wyzerowany,
JB, JNB – jak JC i JNC, ale argumentem rozkazu jest adres dowolnego bitu,
JBC – jak JB, po wykonaniu skoku bit jest zerowany,
JZ i JNZ – jak poprzednio, ale gdy akumulator jest lub nie jest wyzerowany.

Rozkaz CJNE porównuje pierwszy argument z drugim i gdy nie są równe, to wykonuje skok (krótki) o wielkość określoną przez trzeci argument. Trzeci argument powinien być etykietą. Jeżeli pierwszym argumentem jest A, to drugi może być albo adres bezpośredni albo liczba (postaci #n). Jeżeli pierwszym argumentem jest rejestr Rr, to drugim musi być liczba. Innych Możliwości nie ma.
Rozkaz DJNZ zmniejsza pierwszy argument o 1 i jeżeli stanie się on zerem, to zostanie wykonany skok (krótki) o wielkość określoną przez drugi argument. Drugi argument powinien być etykietą. Pierwszy argument musi być albo rejestrem Rr albo adresem bezpośrednim.

3.4 Podprogramy

Stos jest umieszczony w pamięci wewnętrznej. Wierzchołek stosu jest wskazywany przez rejestr SP. Rozkaz PUSH powoduje położenie jego argumentu, który jest adresem bezpośrednim na stosie i zwiększenie SP o 1, a rozkaz POP powoduje zdjęcie ze stosu bajtu i zapisanie jego wartości pod adresem, który jest argumentem rozkazu, a także zmniejszenie SP o 1.
Na początku programu powinno się ustalić wartość SP, czyli określić dno stosu.
Po zresetowaniu procesora, do rejestru SP wpisywana jest wartość 7. Nie należy polegać na domyślnych wartościach.

Rozkaz LCALL ma podobnie jak LJMP jeden argument, ale dodatkowo odkłada na stos adres powrotu, najpierw młodszy, potem starszy bajt. Bezargumentowy rozkaz RET wykonuje skok pod położony na stosie adres i zdejmuje go ze stosu. Wykorzystując rozkazy PUSH i POP można zmienić adres powrotu.

3.5 Format HEX Intela

Kod produkowany przez asembler ma najczęściej format HEX Intela. Z tego formatu korzystają też najczęściej urządzenia programujące pamięć EPROM.

Przykład.

```
:10008000AF5F67F0602703E0322CFA92007780C361
:1000900089001C6B7EA7CA9200FE10D2AA00477D81
:0B00A00080FA92006F3600C3A00076CB
:00000001FF
```

Linie z wyjątkiem ostatniej:

- Pierwszy znak (:) oznacza początek linii danych.
- Następne dwa znaki oznaczają długość danych w linii (10h w tym przypadku).
- Następne cztery znaki oznaczają adres zapisywania danych (0080h w tym przypadku).
- Następne dwa znaki oznaczają typ danych.
- Dalej następują dane.
- Ostatnie dwa znaki są sumą kontrolną, tzn. suma wszystkich znaków (cyfr) w linii jest podzielna przez $256 = FF\mathrm{H}$.

Ostatnia linia ma specjalne znaczenie i zawsze wygląda tak jak powyżej.
Typy danych:

- 00 – dane.
- 01 – koniec pliku,

Pozostałe, 02 – 05, nie będą (na razie) potrzebne.

4 Pamięć i urządzenia zewnętrzne

4.1 Pamięć danych i pamięć programu

Do procesora można przyłączyć zewnętrzną pamięć danych o rozmiarze do 64 kB oraz zewnętrzną pamięć programu o rozmiarze do 64 kB. Niektóre wersje procesorów rodziny Intel 51 nie mają wewnętrznej pamięci programu.
W czasie wykonywania programu adres rozkazu w pamięci programu jest umieszczany w rejestrze PC.
Dane mogą być umieszczone w kodzie programu, np. dyrektywami asemblera DB oraz DW. Odczytywanie tych danych jest możliwe rozkazami MOVC A,@A+DPTR oraz MOVC A,@A+PC.

Rozkaz MOVC A,@A+DPTR powoduje umieszczenie w akumulatorze zawartości bajtu zapisanego w pamięci programu pod adresem, który jest sumą zawartości akumulatora i rejestru DPTR. Program

```
        MOV DPTR,#data
        MOV A,#3
        MOVC A,@A+DPTR
        ORG 0020h
data:   DB 'Ala ma kota'
        END
```

umieszcza w akumulatorze 20h, czyli kod spacji.

Rozkaz MOVC A,@A+PC powoduje umieszczenie w akumulatorze zawartości bajtu zapisanego w pamięci programu pod adresem, który jest sumą zawartości akumulatora i rejestru PC. PC jest równe adresowi po pobraniu rozkazu. Program

```
MOV A,#3
MOVC A,@A+PC
DB 'Ala ma kota'
END
```

umieszcza w akumulatorze 20h, czyli kod spacji.

Pamięć danych jest pamięcią zewnętrzną. Odczyt i zapis danej jest dokonywany rozkazem MOVX za pośrednictwem akumulatora.
MOVX A,@DPTR – bajt do akumulatora z pamięci zewnętrznej, adres wskazywany umieszczony w DPTR,
MOVX @DPTR,A – bajt z akumulatora do pamięci zewnętrznej, adres wskazywany umieszczony w DPTR.
Jest też możliwe adresowanie przy pomocy rejestru R0 lub R1 (bajt młodszy adresu), gdy ustalony jest bajt starszy adresu.

Przykład. Przepisanie łańcucha znaków z pamięci programu do pamięci zewnętrznej.

```
tab_x     EQU 10h
          MOV R0,#12          ;rozmiar tablicy
          MOV DPTR,#tab_x     ;adres tablicy - dane
          CLR A               ;adres - kod
          MOV B,A             ;  i do B
loop:     PUSH DPH            ;zapamiętanie adresu
          PUSH DPL            ;  dla danych
          MOV DPTR,#tab_c     ;adres tablicy - kod
          MOVC A,@A+DPTR      ;bajt w ACC
          POP DPL             ;odtworzenie adresu
```

```
        POP DPH             ;  dla danych
        MOVX @DPTR,A        ;bajt do pam. zewn.
        INC B               ;nast. adres - kod
        MOV A,B             ;  i do ACC
        INC DPTR            ;nast. adres - pam. zewn.
        DJNZ R0,loop
        SJMP $
        ORG 30h
tab_c:  DB 'Ala ma kota'
        DB 0                ;koniec napisu
```

4.2 Porty

Procesor 8051/52 ma cztery porty P0, P1, P2, P3.
Na ogół P2 służy do adresowania starszego bajtu, a P0 młodszego przy korzystaniu z pamięci zewnętrznej.
Rozkaz MOVX A,@DPTR m.in. wpisuje do portu P2 zawartość DPH, a do P0 zawartość DPL - tylko w czasie operacji odczytu/zapisu.
Jeżeli do P2 wpisany jest starszy bajt, zamiast rozkazu
MOVX A,@DPTR można użyć MOVX A,@Ri
zamiast
MOVX @DPTR,A można użyć MOVX @Ri,A
gdzie Ri jest albo R0 albo R1.

Port P1 służy do bezpośredniej komunikacji ze światem zewnętrznym.
Port P3 pełni również inne funkcje.
Przed odczytem danej z portu lub jednej jego linii, należy przedtem wpisać tam jedynkę.

4.3 Urządzenia zewnętrzne

Urządzenia zewnętrzne mają tę samą przestrzeń adresowa, co i zewnętrzna pamięć danych.
Jeżeli np. wyświetlacz ma podłączone kolejne litery pod adresy począwszy od 10h, to program przepisywania tablicy umieści napis 'Ala ma kota' na tablicy świetlnej. Na końcu łańcucha jest zero (jak w C).
Inna wersja tego programu, wyświetla napis aż do końca, czyli do naptkania znaku o kodzie 0.

```
tab_x     EQU 10h
          MOV R0,#100          ;maks. rozmiar tablicy
          MOV DPTR,#tab_x      ;adres tablicy - dane
          CLR A                ;adres - kod
          MOV B,A              ;  i do B
loop:     PUSH DPH             ;zapamiętanie adresu
          PUSH DPL             ;  dla danych
          MOV DPTR,#tab_c      ;adres tablicy - kod
          MOVC A,@A+DPTR       ;bajt w ACC
          JZ stop              ;jesli koniec napisu
          POP DPL              ;odtworzenie adresu
          POP DPH              ;  dla danych
```

```
          MOVX @DPTR,A          ;bajt do pam. zewn.
          INC B                 ;nast. adres - kod
          MOV A,B               ;  i do ACC
          INC DPTR              ;nast. adres - pam. zewn.
          DJNZ R0,loop
stop:     SJMP $
          ORG 30h
tab_c:    DB 'Ala ma kota'
          DB 0                  ;koniec napisu
```

5 Operacje arytmetyczne i logiczne

5.1 Arytmetyka

Dodawanie liczb x i y w systemie dwójkowym:
$x = x_7x_6x_5x_4x_3x_2x_1x_0$,
$y = y_7y_6y_5y_4y_3y_2y_1y_0$.

```
CY=0;
for i:=0 to 7 do
begin
     z[i]:=(x[i]+y[i]+CY) mod 2;
     CY:=(x[i]+y[i]+CY) div 2;
end;
```

Odejmowanie liczb x i y w systemie dwójkowym:

Niech $\bar{y} = \bar{y}_7\bar{y}_6\bar{y}_5\bar{y}_4\bar{y}_3\bar{y}_2\bar{y}_1\bar{y}_0$

Wtedy $x - y = x + (\bar{y} + 1)$

Przykład.

Wykonujemy $3 - 5 = -2$.

$y = +5$	00000101
$\bar{y}$	11111010
$+1$	1
$-y = -5$	11111011

Teraz dodajemy $3 + (-5)$

$+3$	00000011
-5	11111011
$=$	11111110

To ma być -2, czyli

$z = +2$	00000010
$\bar{z}$	11111101
$+1$	1
$-z = -2$	11111110

Jest to kod uzupełnień do dwóch.

Można w nim reprezentować liczby 0d -128 d0 127.

Uwaga. Taki właśnie jest zasięg skoków krótkich.

Kod BDC (binary decimal code): każdy bajt reprezentuje dwie cyfry dziesiętne, każdą przez połówkę bajtu.

Przykład. Liczba 29 w kodzie BDC jest równa $\underbrace{0010}_{2}\ \underbrace{1001}_{9}$, a liczba 78 jest równa $\underbrace{0111}_{7}\ \underbrace{1000}_{8}$

5.2 Rozkazy arytmetyczne

- Dodawanie: ADD dodaje do akumulatora drugi argument,
- Dodawanie z przeniesieniem ADDC dodaje do akumulatora drugi argument i dodaje bit CY,
- Odejmowanie (z przeniesieniem) SUBB odejmuje od akumulatora drugi argument i odejmuje bit CY, wynik w kodzie uzupełnień do dwóch.

Pierwszym argumentem jest zawsze A, drugim może być adres bezpośredni, rejestr, adres pośredni lub liczba.
Wynik jest zawsze umieszczony w akumulatorze. Jeśli wynik jest większy od FFh, to ustawiane są bity CY, AC i OV.

Rozkaz INC zwiększa o 1, a rozkaz DEC zmniejsza o 1 argument, którym może być akumulator A, adres bezpośredni, rejestr lub adres pośredni. Nie zmieniają się znaczniki.
Argumentem rozkazu INC może być też DPTR.

Mnożenie: MUL AB. Wynikiem jest liczba 16 bitowa, starszy bajt jest umieszczony w B, młodszy w A. Jeśli wynik jest większy od FFh, to bit OV jest ustawiany, a CY zerowany.
Dzielenie (całkowitoliczbowe) DIV AB. Liczbę zawartą w A dzielimy przez liczbę zawartą w B. Część całkowita wyniku jest wpisywana do A, reszta z dzielenia do B.

Korekcja dziesiętna DA A daje w połączenie z rozkazami ADD lub ADDC poprawny wynik , gdy dodajemy liczby w kodzie BDC. Ustawia bit CY, gdy wynik jest większy od 99.

Przykład. Dodając $29 + 78 = 107$ w kodzie BDC otrzymujemy :

$$\begin{array}{ll} 29 & 0010\,1001 \\ 78 & 0111\,1000 \\ \hline 107 & 1010\,0001 \end{array}$$

Powinno zaś być $\underbrace{0000}_{0}\ \underbrace{0111}_{7}$ i ustawiony znacznik przeniesienia, bit CY.

5.3 Rozkazy logiczne na bajtach

Rozkazy ANL, ORL, XRL są odpowiednio koniunkcją, alternatywą i alternatywą wykluczającą bitów argumentów. Pierwszym argumentem może być A lub adres bezpośredni.

- Jeżeli pierwszym argumentem jest A, to drugim może być adres bezpośredni, rejestr, adres pośredni lub liczba.
- Jeżeli pierwszym argumentem jest adres bezpośredni, to drugim jest albo A albo liczba.

Rozkaz CLR A powoduje wyzerowanie akumulatora. Daje ten sam efekt, co MOV A,#0 i wykonywany jest w tym samym czasie – jeden cykl rozkazowy.
Rozkaz CPL A wpisuje do A negację (bit po bicie) zawartości A.
Rozkaz SWAP A zamienia w akumulatorze starszą i młodszą połówkę bajtu.

Rozkazy przesunięć RL, RLC, RR, RRC jako argument zawsze mają A. Oznaczmy $A = a_7a_6a_5a_4a_3a_2a_1a_0$.

- RL A: $a_0 = a_7$, $a_{i+1} = a_i$ dla $i = 0, 1, \ldots, 6$,
- RLC A: $a_0 = CY$, $a_{i+1} = a_i$ dla $i = 0, 1, \ldots, 6$, $CY = a_7$,
- RR A: $a_7 = a_0$, $a_{i-1} = a_i$ dla $i = 1, 2, \ldots, 7$,
- RRC A: $a_7 = CY$, $a_{i-1} = a_i$ dla $i = 1, 2, \ldots, 7$, $CY = a_0$.

5.4 Rozkazy na bitach

Rozkaz CLR zeruje, a rozkaz SETB ustawia bit. Argumentem jest adres bezpośredni bitu lub bit C.
Rozkaz CPL daje negację bitu. Argumentem jest adres bezpośredni bitu lub bit C.
Rozkazy ANL i ORL dają koniunkcję lub alternatywę bitów. Pierwszym argumentem jest C, a drugim adres bezpośredni bitu.
Rozkaz MOV nadaje pierwszemu argumentowi wartość drugiego argumentu. Jednym argumentem jest C, drugim adres bezpośredni bitu.

Uwaga.
Bity można adresować przez podanie nazwy bajtu w SFR i numeru bitu w postaci
nazwa.numer
np. ACC.3 adresuje bit o numerze 4 w akumulatorze, P0.1 adresuje bit o numerze 1 portu P0.
Można też bit adresować przez nazwę bitu.
Przykład.
Słowo stanu programu PSW.

CY=PSW.7 ustawiany lub zerowany sprzętowo podczas wykonywania operacji arytmetycznych, sygnalizuje przeniesienie lub pożyczkę z bitu 7. Pełni rolę akumulatora boolowskiego. Dostępny też przez

nazwę C.

AC=PSW.6 znacznik przeniesienia pomocniczego przy używaniu zapisu BDC.

F0=PSW.5 bit do wszystkiego, zmieniany programowo.

RS1=PSW.4, RS0=PSW.3 liczba RS1 RS0 wskazuje na zestaw rejestrów.

OV=PSW.2 znacznik nadmiaru ustawiany lub zerowany sprzętowo, wskazuje na przekroczenie zakresu liczb w kodzie U2.

P=PSW.0 znacznik parzystości ustawiany lub zerowany sprzętowo, (P=1 – parzysta, P=0 – nieparzysta liczba jedynek w A.

5.5 Typy zmiennoprzecinkowe

Liczby zmiennoprzecinkowe są podzbiorem liczb wymiernych postaci

$$z = (-1)^s m p^w, \tag{5.5.1}$$

- p – podstawa (liczba naturalna > 1),
- m – mantysa (nieujemna liczba wymierna o skończonym rozwinięciu przy podstawie p),
- s – znak ($s = 0$ lub $s = 1$),
- w – wykładnik (liczba całkowita).

Jeżeli

$$1 \leq m < p, \tag{5.5.2}$$

to przedstawienie liczby $z \neq 0$ (5.5.1) jest jednoznaczne.
W komputerze zawsze jest $p = 2$ (notacja binarna) natomiast do „normalnego" użytku jest $p = 10$ (notacja dziesiętna). W notacji dziesiętniej stosuje się zapis $z = x\,\mathrm{E}\,y = x10^y$, gdzie y jest liczbą całkowitą, a x jest liczbą wymierną o skończonym rozwinięciu dziesiętnym.
Dla $p = 2$, jeżeli spełniony jest warunek (5.5.2), to część całkowita mantysy jest równa 1. Można wtedy przyjąć ją za domyślną i nie zapamiętywać, a za pamiętywać tylko część ułamkową. Dla takich m jest to liczba znormalizowana.

Wykładnik spełnia nierówność $w_{\min} < w < w_{\max}$.
Liczby

$$w_{\min} + 1 \text{ oraz } w_{\max} - 1$$

określają najmniejszy i największy wykładnik liczby.
Dla liczb znormalizowanych z

$$2^{w_{\min}+1} \leq |z| < 2 \cdot 2^{w_{\max}-1} = 2^{\cdot \max} \tag{5.5.3}$$

Wartość $w_{\max}$ jest zarezerwowana dla reprezentacji symboli, a $w_{\min}$ – dla zera i liczb zdenormalizowanych.

Zero i liczby zdenormalizowane

Jeżeli $w = w_{\min}$, to z założenia $m < 1$ i wtedy $z = (-1)^s m 2^{w_{\min}+1}$.
Wobec tego

$$z = 0 \iff m = 0 \wedge w = w_{\min}, \tag{5.5.4}$$

oraz dla $|z| < 2^{w_{\min}+1}$

$$z = (-1)^s m \cdot 2^{w_{\min}+1} \iff m < 1 \wedge w = w_{\min}. \tag{5.5.5}$$

Symbole

Nieskończoności:

$$+\infty \iff m = 1.0 \wedge w = w_{\max} \wedge s = 0,$$
$$+\infty \iff m = 1.0 \wedge w = w_{\max} \wedge s = 1.$$

Nieliczby (NaN (ang. *Not a Number*):

$$m \neq 1 \wedge w = w_{\max}.$$

Formaty zmiennoprzecinkowe:

- pojedynczy, 32 bity,
- podwójny, 64 bity,
- chwilowy, 80 bitów.

Zapis liczby w trzech polach od najstarszego do najmłodszego bitu: pole znaku s, pole wykładnika w, pole mantysy m.
Pole znaku ma zawsze 1 bit.
Pole wykładnika zawiera wykładnik w dwójkowym kodzie spolaryzowanym w_B (polaryzacja=ang. $bias$), wg wzoru:

$$w = w_B - B$$

gdzie B jest polaryzacją wyznaczoną tak, że

$$w_{\min} = w_B - B \text{ dla } w_B = 000\ldots000\text{B},$$
$$w_{\max} = w_B - B \text{ dla } w_B = 111\ldots111\text{B}.$$

Pole mantysy m zawiera część ułamkową m' mantysy w naturalnym kodzie dwójkowym. W formacie chwilowym również jeden bit części całkowitej.

Format	s	Wykładnik w_B	I	Część ułamk. m'
32 bity	31	30 23		22 0
64 bity	63	62 52		51 0
80 bitów	79	78 64	63	62 0

Daje to następujące rozmiary pól:

Format	Wykładnik	Część ułamk.
32 bity	8	23
64 bity	11	52
80 bitów	15	63

6 Przerwania, zegary i liczniki

6.1 Przerwania

Odebranie przez procesor zgłoszenia przerwania powoduje zawieszenie wykonywanego programu, zapisanie na stos adresu powrotu (aktualnego stanu PC) i wykonanie skoku do podprogramu. Podprogram musi kończyć się rozkazem RETI. Wykonanie takiego podprogramu nazywa się obsługą przerwania.
Przerwania mogą mieć różne priorytety. Jeżeli w czasie obsługi przerwania procesor odbierze zgłoszenie przerwania o wyższym priorytecie, to przerywa obsługę przerwania o niższym priorytecie i kończy ją po zakończeniu przerwania o wyższym priorytecie.

Zgłoszone przerwanie o niższym lub równym priorytecie jest obsługiwanie po zakończeniu bieżącej obsługi przerwania.
Procesor 8051 może przyjąć zgłoszenie z 5 źródeł:

- z wejścia INT0,
- z wejścia INT1,
- z przepełnienia licznika T0,
- z przepełnienia licznika T1,
- z portu szeregowego – koniec nadawania lub odbierania znaku (dwa osobne znaczniki).

Podprogram obsługi przerwania ma ustalony sprzętowo dla niego adres.

Przerwanie z danego źródła powoduje ustawienie na 1 odpowiedniego znacznika (bitu). Bity te można też ustawić programowo przez SETB. Przyjęcie zgłoszenia zeruje znaczniki, z wyjątkiem przychodzących z portu szeregowego. Wszystkie bity można zerować programowo przez CLR.
System przerwań można włączyć i wyłączyć programowo, a także każde zgłoszenie przerwania może być indywidualnie zamaskowane przez ustawienie odpowiedniego bitu w bajcie IE (interrupt enable).
Jeśli EA=1, to zgłoszenia przerwań są przyjmowane, a jeśli EA=0, to nie są.
Priorytety można zmienić bitami w bajcie IP (interrupt priority)

Znaczniki (bity) w bajtach EA i IP i znaczniki zgłoszenia przerwań (Zn) w bajtach TCON i SCON, adresy podprogramów obsługi przerwań.

IE	IP	Zn	Adres	Przerwanie	Priorytet
EX0	PX0	IT0	0003h	zewnętrzne INT0	najwyższy
ET0	PT0	TF0	000Bh	od licznika/ zegara T0	
EX1	PX1	IT0	0013h	zewnętrzne INT1	
ET1	PT1	TF1	001Bh	od licznika/ zegara T1	
ES	PS	TI/RI	0023h	od portu szeregowego	najniższy
EA				system przerwań	

Bity IE0 i IE1 sterują sposobem przyjęcia przerwania:

- IEi=0 – zgłoszenie niskim poziomem sygnału na wejściu
- IEi=1 – zgłoszenie opadającym zboczem sygnału.

Wejściem jest INT0=P3.2 lub INT1=P3.3

Uwaga. Różnica między rozkazami RET i RETI.

Rozkaz RETI kończy obsługę przerwania, a rozkaz RET nie kończy. Do momentu wystąpienia RETI nie będzie przyjęte żadne zgłoszenie przerwania o niższym lub równym priorytecie.

Ponieważ na podprogram obsługi przerwania procesor rezerwuje zaledwie 8 bajtów, a obszar pamięci programu zarezerwowany na te podprogramy nie powinien być wykorzystywany do innych celów, to typowy program ma strukturę:

```
LJMP start      ;skok do właściwego programu
ORG 0003h
LJMP int0proc   ;skok do obsługi INT0
ORG 000Bh
LJMP t0proc     ;skok do obsługi T0
ORG 0013h
LJMP int1proc   ;skok do obsługi INT1
ORG 001Bh
LJMP t1proc     ;skok do obsługi T1
ORG 0023h
LJMP rs_proc    ;skok do obsługi RS
```

```
        ORG 0030h
;procedury obsługi przerwań
int0proc:       ;obsługa przerwania od INT0
        RETI
t0proc:         ;obsługa przerwania od T0
        RETI
int1proc:       ;obsługa przerwania od INT1
        RETI
t1proc:         ;obsługa przerwania od T1
        RETI
rs_proc:        ;obsługa przerwania od RS
        RETI
```

```
start:
;tu zaczyna się właściwy program
        SETB EA  ;zezwolenie na przyjmowanie przerwań
        SETB EX0 ;odblokowanie przerwań z INT0
wait:   SJMP $
;czekamy na przerwanie z INT0, obsługujemy je
;i wracamy do linii z etykietą wait
```

6.2 Zegary i liczniki

Do mierzenia czasu lub zliczania impulsów wykorzystywane są dwa 16-bitowe liczniki T0 i T1. Każdy z nich składa się z dwóch bajtów THi i TLi, które mogą być wykorzystywane również niezależnie. Przepełnienie się licznika powoduje ustawienie znacznika TFi (o ile EA=1 oraz ETi=1). Sposób działania licznika jest określony przez ustawienie bitów w słowach TMOD i TCON.

TMOD:

$$\underbrace{\text{GATE C/T M1 M0}}_{\text{T1}}\ \underbrace{\text{GATE C/T M1 M0}}_{\text{T0}}$$

TCON:

TF1 TR1 TF0 TR0 IE1 IT1 IE0 IT0

- M1 M0 – tryb pracy od 0 do 3,
- C/T=0 – funkcja zegara, C/T=1 – funkcja licznika.
- Przyjmiemy, że GATE=0 (tak jest przy inicjacji systemu)
- TRi – uruchomienie licznika Ti

Omówione zostaną tylko tryby 1 i 2.

W trybie 1 odpowiedni licznik 16-bitowy Ti jest zwiększany o 1 przy wystąpieniu impulsu na wejściu T0=P3.4 lub T1=P3.5 (dla C/T=1) lub w każdym cyklu (dla C/T=0). Przy częstotliwości zegara 12 MHz, jest to co 1 μs. Przy przepełnieniu, tzn. przy przejściu z FFFFh na 0, generowane jest przerwanie. Impulsem jest zmiana poziomu sygnału na wejściu T0 lub T1 próbkowana w dwóch kolejnych cyklach.

Do właściwego odmierzania czasu trzeba ustawić odpowiednią wartość na początku, a potem w procedurze obsługi przerwania doładować licznik.

Przykład. Mamy odmierzać czas 40 ms w systemie z zegarem 12 MHz. Wartością początkową musi być 64C0h = FFFFh − 40000.
Ponieważ od chwili przepełnienia licznika do przyjęcia przerwania może upłynąć nieznany okres czasu, to licznik trzeba doładować w locie. Zakładamy, że ten okres jest mniejszy od 64. Wtedy doładowanie ma postać:

```
ORL TL0,#0C0h   ;dodanie do mniej znaczącego bajtu
MOV TH0,#64h    ;wpisanie bardziej znaczącego bajtu
```

W trybie 2 licznik Ti pracuje jako 8-bitowy (TLi) z automatycznym przepisaniem (reload) wartości początkowej z THi w chwili przepłnienia licznika. Maksymalny okres przerwań wynosi 256 cykli, czyli 256 μs przy zegarze 12 MHz.
Pozostałe reguły są takie jak w trybie 1.
Przykład. Zegar 11.0592 MHz. Jeden cykl trwa $t_c = 12/11.0592\,\mu$s. Jeśli wpiszemy do THi wartość 244, to przepełnienie nastąpi co

$$t_c \cdot 256 \cdot (256 - 244)\,\mu\text{s} = 10/3\,\text{ms}.$$

```
;Stałe przy kwarcu 11.0592MHz
;Stała do odmierzania 10/3 ms
set11     EQU 244   ;256-244=12
clk1      EQU 3     ;3 tyknięcia na 10ms
```

```
clk5      EQU 15    ;15 tyknięć na 50ms
fblink    EQU 20    ;co 0.2 sek.
```

Doładowanie ogranicza się do

```
        ;mniej znaczący bajt TL0 jest zerem
        ;w momencie przepełnienia
        MOV TH0,#set11    ;wpisanie bardziej znaczącego bajtu
```