Základy programování 8051

1. Úvod do programování jednočipových mikropočítačů

Jednočipový mikropočítač řady 8051 je v současné době nepsaným standardem v mikroprocesorové technice. Jeho architektura a instrukční soubor se staly základem pro celou řadu vyšších typů jednočipových mikropočítačů.

Úkolem této části je popis elementárních programátorských technik a postupů, které je nutno zvládnout pro programování konkrétních aplikací. Předpokládá se alespoň základní znalost architektury 8051, jeho instrukcí a používání překladače.

2. Přesuny dat a zápis konstant

Přesuny dat v paměti procesoru provádí instrukce:

přesun MOV <oper1>,<oper2>

Začneme od nejjednoduššího příkladu, a to je zápis konstanty do registru.

```
MOV A,#12
```

Po provedení této instrukce se naplní obsah registru A číslem 12. Zde je nutno upozornit na znak '#', který je uveden před číslem 12. Tento znak je v assembleru 8051 velmi důležitý, odlišuje totiž zápis konstanty od zápisu adresy neboli tzv.přímé hodnoty, což je buď SFR registr nebo oblast vnitřní RAM. Špatné použití znaku '#' (ať už jeho vynechání či nadbytečnost) má za následek vznik těžko zjistitelných chyb a způsobuje **záhadné** chování programu.

Příklad:

```
MOV R0,#10 ;zápis 10 do registru R0
.
.
MOV A,#0 ;zápis 0 do A
.
.
MOV A,0 ;zápis 10 do A !!!!
```

Ve druhém případě použití registru A (třetí instrukce programu) nedošlo k jeho vynulování, ale k přesunu přímé hodnoty z adresy 0 do registru A. Protože na adrese 0 ve vnitřní paměti RAM leží registr R0, dojde k přepisu jeho hodnoty do registru A.

Instrukce MOV má má široké spektrum parametrů a je možno ji použít ve všech adresovacích módech.

Příklad:

```
MOV R0,#20H ;počáteční adresa ukládání
MOV B,#8 ;počet průchodů smyčkou
CYKL: MOV @R0,P1 ;načtení portu P1
INC R0 ;zvyš ukazatel
```

```
DJNZ B,CYKL ;proveď celkem 8x
```

Uvedený příklad ilustruje použití instrukce MOV při nepřímém adresování. Program 8x sejme hodnotu portu P1 a uloží ji postupně na adresy 20H až 27H do **vnitřní** paměti RAM.

Modifikace tohoto příkladu pro uložení hodnot do **vnější** paměti dat vypadá následovně:

```
MOV R0,#20H ;počáteční adresa ukládání
MOV B,#8 ;počet průchodů smyčkou
CYKL: MOV A,P1 ;načtení portu P1
MOVX @R0,A ;přesun do vnější paměti
INC R0 ;zvyš ukazatel
DJNZ B,CYKL ;proveď celkem 8x
```

Pro adresování **vnější** paměti dat je nutno použít instrukci MOVX (Move External - přesun z vnější paměti).

Všechny uvedené příklady zatím přesouvaly data v **paměti údajů** (ať už vnitřní, či vnější). Pro přesun dat z **paměti programu** slouží instrukce MOVC. Následující příklad ukazuje použití MOVC (Move Constant - přesun z pevné paměti) při přenosu bloku dat (např.tabulky) z paměti programu do paměti dat:

```
ORG 0
MOV DPTR,#TAB ;zápis adresy tabulky
MOV R7,#TAB_END ;zápis délky tabulky do R7
MOV R0,#20H ;tabulka se bude ukládat
;od adresy 20H ve vnitřní
;RAM
CYKL: MOV A,#0 ;nulování A
MOVC A,@A+DPTR ;přesun jednoho prvku tab.
MOV @R0,A ;do A a odtud do RAM
INC R0 ;zvyš ukazatel do RAM
INC DPTR ;zvyš ukazatel do ROM
DJNZ R7,CYKL ;opakuj přes celou délku
;tabulky
ORG 300H ;umístění tabulky
TAB: DB 1,2,3,4,5 ;jednotlivé prvky tabulky
DB 6,7,8,9,10 ;
TAB_END EQU $-TAB ;výpočet délky tabulky
;udělá překladač
```

V programu je použita instrukce MOV DPTR,#TAB, která do registru DPTR načte adresu tabulky. Instrukce s těmito parametry pracuje jako jediná se šestnácti bity, všechny ostatní parametry instrukce MOV pracují s osmibitovými údaji.

Poznámka:

Poslední řádek příkladu ukazuje jeden ze způsobů použití pseudoinstrukce EQU pro výpočet délky tabulky. Kdybychom místo posledního řádku uvedli:

```
TAB_END EQU 10
```

program by pracoval stejně, ale při jakékoli změně délky tabulky bychom tento údaj museli neustále počítat a měnit (což může zvláště u delších tabulek vést k chybám). Postupem uvedeným v příkladě přenecháme starost o výpočet délky tabulky překladači.

Pro přesun údajů je možno použít také instrukci:

výměna XCH A,<parametr>

která vymění obsah registru A s druhým parametrem. Takže sekvenci instrukcí:

```
;výměna obsahu A, R0
MOV B,A
MOV A,R0
MOV R0,B
```

je možno nahradit jedinou instrukcí:

```
XCH A,R0
```

3. Používání zásobníku

Zásobník je část paměti v oblasti vnitřní RAM. Je definován svým počátkem a může ležet kdekoli ve vnitřní oblasti RAM. V oblasti SFR registrů je definován ukazatel zásobníku - registr SP. Tento registr ukazuje vždy na vrchol zásobníku. Při ukládání dat na zásobník se nejprve hodnota SP registru zvýší o 1 - zásobník *roste* směrem do oblasti vyšších adres vnitřní RAM - a potom se data uloží tam, kam ukazuje SP. Při vybírání dat ze zásobníku je postup opačný - nejprve se uloží data, adresovaná SP a potom se SP sníží o 1.

Nastavení ukazatele zásobníku a starost o to, aby nedošlo ke kolizi zásobníku s ostatními údaji v RAM je zcela na zodpovědnosti programátora. Vždy je nutno pro zásobník vyhradit dostatečně velkou část paměti tak, aby mohl pojmout návratovou adresu i toho nejvíce vnořeného podprogramu. U systému s přerušením je nutno počítat s tím, že přerušena může být libovolná část programu, takže kapacita zásobníku musí být navržena s rezervou i pro tento případ.

Po resetu mikropočítače se hodnota SP nastaví na 7.

Zásobník je využíván pro odkládání návratových adres při volání podprogramů nebo při přerušení (viz bod 7).

Pro ukládání a vybírání dat ze zásobníku slouží instrukce:

ulož PUSH <parametr>

vyjmi POP <parametr>

Nejtypičtějším použitím zásobníku je přechodné ukládání mezivýsledků pro pozdější zpracování nebo pro úschovu pracovních registrů při vstupu do procedury.

Příklad:
```
ACALL PROC ;volání procedury
.
.
PROC: PUSH PSW ;úschova PSW
PUSH ACC ;úschova A
```

```
PUSH B ;úschova B
.
;nyní mohu v proceduře používat registry A,B
;před návratem z procedury je jejich obsah obnoven
.
.
POP B ;obnovení B
POP ACC ;obnovení A
POP PSW ;obnovení PSW
RET
```

4. Logické operace, posuvy a rotace

LOGICKÉ OPERACE

V instrukčním souboru 8051 jsou následující instrukce pro logické operace:

logické nasobení ANL <oper1>,<oper2>

logické sečítání ORL <oper1>,<oper2>

log.exkluzivní součet XRL <oper1>,<oper2>

Uvedené instrukce provádějí příslušnou logickou operaci mezi dvěma operandy <oper1>,<oper2> po jednotlivých bitech a výsledek se uloží do prvního operandu <oper1>.

Typickým příkladem použití instrukce ANL je vynulování jednotlivých bitů operandu. Používá se v těch případech, kdy potřebuji část operandu vynulovat.

Příklady:

```
;nulování bitu 0..3 registru B
ANL B,#11110000B
;nulování bitů 0,2,4,6 registru A
ANL B,#10101010B
;nulování celého registru B
ANL B,#0
```

Této operaci se říká **maskování** - operand se logicky vynásobí s tzv.maskou - v našem příkladě 11110000B,10101010B nebo 0. Ty bity prvního operandu, které se vynásobí s nulou v příslušném bitu druhého operandu (masce), jsou vynulovány (vymaskovány).

Příklad:

V registru R0 máme dvě BCD číslice. Úkolem programu bude tyto dvě číslice uložit do registrů R1 a R2, každou zvlášť. Tedy například:

```
R0=15H  ———>  R1=01H, R2=05H
```

Poznámka: BCD tvar číslic je způsob uchovávání číslic ve tvaru, kdy v jednom bytu jsou umístěny dvě číslice (v našem příkladě číslice 1 a 5.

```
MOV A,R0 ;přesun do A
ANL A,#0F0H ;nulování spodní části byte
MOV R1,A ;uschovej
MOV A,R0 ;znovu přesun
ANL A,#0FH ;nulování horní části byte
SWAP A ;zaměň horní a dolní část
MOV R2,A ;registru A a ulož
```

Instrukce SWAP A provede záměnu horní a dolní části registru A. Je-li obsah registru A před provedením instrukce SWAP např. 56H, po provedení této instrukce se obsah registru A změní na 65H.

Instrukce ORL provádí logický součet (po bitech operace OR) dvou operandů. Časté použití této instrukce je pro nastavení příslušných bitů daného operandu do 1.
Příklad:

```
;nastavení bitu 0 registru A do jedničky
ORL A,#1
;nastavení bitů 5,6,7 do jedničky
ORL A,#11100000B
```

Poznámka: Pro nastavení, nulování a negaci jednotlivých bitů existují speciální instrukce, které jsou popsány v následující kapitole.
Instrukce XRL provádí výhradní logický součet operandů (po bitech operace XOR) a výsledek ukládá do prvého operandu. Praktické využití této instrukce je v negaci vybraných bitů daného operandu.
Příklad:
Přečtěte z portu P0 hodnotu, spodní čtyři bity vynulujte, horní čtyři bity invertujte (negujte) a výsledek vyšlete na port P1.

```
MOV A,P0 ;načtení z portu P0 do A
ANL A,#0F0H ;nulování spodních bitů
XRL A,#0F0H ;invertování horních bitů
MOV P1,A ;vyslání na port P1
```

POSUVY A ROTACE

Pro rotace jsou v instrukčním souboru 8051 instrukce:
rotace doleva RL A
rotace doleva přes C RLC A
rotace doprava RR A
rotace doprava přes C RRC A
Posouvat (rotovat) operand je možno pouze v akumulátoru. Rotace mají široké použití v aritmetických programech (viz. kap.6). Ukážeme si další typické použití - vysílání tzv. **pochodující nuly** na port. Pochodující nula znamená, že na jeden port za sebou postupně vyšleme hodnoty:
11111110
11111101
11111011
11110111
11101111
11011111
10111111
01111111
Jak je vidět z tabulky, hodnoty, které se vysílají na port se liší pouze v pozici nuly - nula **pochoduje** přes celý port. Tato technika se používá při obsluze klávesnice - viz část II, zadání 19.

```
MOV R0,#8 ;počítadlo cyklů
MOV A,#11111110B ;první hodnota do A
CYKL: MOV P1,A ;vyslání na port
RL A ;rotace akumulátoru doleva
DJNZ R0,CYKL ;celkem 8x
```

Další příklad ukazuje způsob rotace 16-ti bitového operandu doleva. K této rotaci se používají instrukce s přenosem do C bitu. Uvedený posuv se nazývá **logický posuv** a spočívá v tom, že bity D0-D14 se posunou o jedno místo doleva, do bitu D0 se přesune bit C a bit D15 se přesune do C.
Příklad:

```
;logický posuv 16-ti bitového čísla, uloženého
;v R0 a R1
MOV A,R0 ;načtení dolních 8 bitů do A
RLC A ;rotace doleva ms přenosem do C
XCH A,R1 ;uschovej a načti horních 8 bitů
RLC A ;rotuj horních 8 bitů
XCH A,R1 ;ulož horních 8 bitů
MOV R0,A ;ulož spodních 8 bitů
```

5. Booleovský procesor

Architektrura 8051 umožňuje pracovat přímo s jednotlivými bity. Instrukčním soubor obsahuje instrukce, které umožňují přímou adresaci jednotlivých bitů (ať už ve vnitřní RAM nebo v oblasti SFR registrů). Jsou to instrukce:
přesun MOV C,<bit>
komplementaci CPL <bit>
nulování CLR <bit>
nastavování SETB <bit>
logický součet ORL C,<bit>
logický součet s neg.bitem ORL C,/<bit>
logický součin ANL C,<bit>

logický součin s neg.bitem ANL C,/<bit>
Dále pak je možno stav kteréhokoli bitu testovat na hodnotu **true** nebo **false** instrukcemi podmíněných skoků:

skok,je-li bit roven jedné JB <adr>
skok, je-li bit roven nule JNB <adr>
skok, je-li bit roven jedné JBC <adr>
s následným nulováním bitu

Uvedené instrukce představují v praxi velmi silný nástroj pro řešení konkrétních aplikací.
Příklad:
Pokud je na vývodu T0 log.0, je na vývodu T1 stejná hodnota, jako je na vývodu P1.0. Pokud je na vývodu T0 log.1, hodnota na výstupu T1 se mění periodicky (střídání 0 a 1).

```
TEST: JB T0,KOMPL ;test na stav vývodu T0
MOV C,P1.0 ;je nulový - načti P1.0
MOV T1,C ;a přenes na T1
SJMP TEST ;návrat na začátek
KOMPL: CPL T1 ;komplementace vývodu T1
SJMP TEST ;návrat na začátek
```

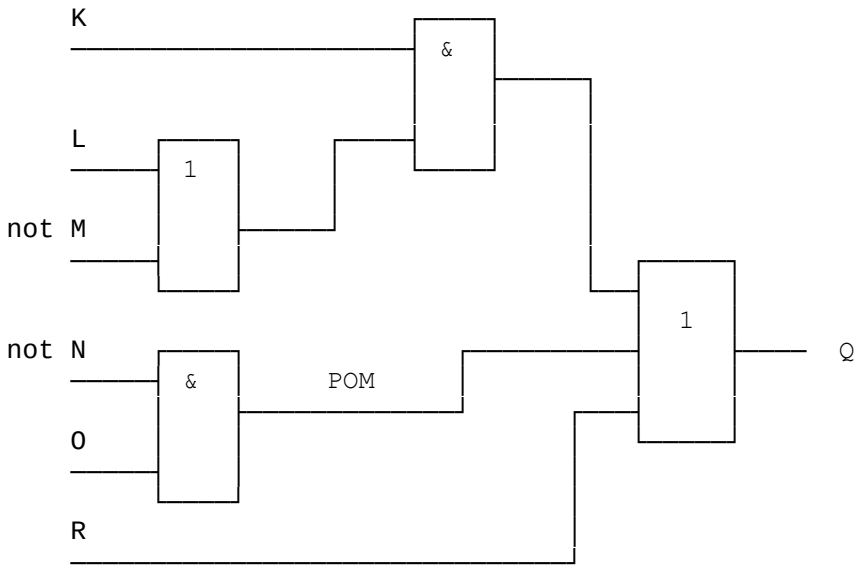
Příklad:
Synchronizace na přicházející signál. Úkolem je zjistit okamžik sestupné hrany (přechod z jedničky do nuly) na vývoru P2.5. Celý problém řeší zápis jedinné instrukce:

```
JB P2.5,$
```

nebo

```
ZDE: JB P2.5,ZDE
```

Oba zápisy jsou ekvivalentní, význam znaku **$** byl vysvětlen v kapitole 1. Činnost procesoru při provádění této instrukce je jednoduchá - pokud je na vývodu P2.5 logická jednička, provádí se skok na tutéž instrukci, tedy čeká se na okamžik, kdy dojde ke změně z log.1 na log.0. Podobně se dá realizovat čekání na vzestupnou hranu, celý puls, atd..
Příklad:
Další z možnosti využití instrukcí pracujících s bity je ve vyhodnocení booleovského výrazu. Představme si, že máme vyhodnotit booleovskou funkci:
Q= K . (L + not M) + (not N . O) + R



Řešení:

```
K BIT 0 ;definice jednotlivých
L BIT 1 ;proměnných v oblasti bitově
```

```
M BIT 2 ;adresovatelné RAM
N BIT 3
O BIT 4
R BIT 5
Q BIT 6
POM BIT 7
MOV C,O ;výpočet (not N.O)
ANL C,/N
MOV POM,C ;úschova mezivýsledku do POM
MOV C,L ;výpočet (not M + L)
ORL C,/M
ANL C,K ;and K
ORL C,POM ;or POM
ORL C,R ;or R
MOV Q,C ;výsledek ulož do Q
END ;konec programu
```

6. Větvení programů, cykly

VĚTVENÍ PROGRAMŮ

S některými instrukcemi pro větvení programu jsme se seznámili v předchozí kapitole. Mezi instrukce skoků patří dále:

skok, je-li C=1 JC <adr>
skok, je-li C=0 JNC <adr>
skok, je-li A=1 JZ <adr>
skok, je-li A=0 JNZ <adr>

Všechny uvedené instrukce (JB,JNB,JBC,JC,JNC,JZ,JNZ) jsou tzv. podmíněné skoky, kdy skok na zadanou adresu je podmíněn splněním určité podmínky - (bit nastave/vynulován, akumulátor je nulový...). Jejich další společnou vlastností je, že cíl skoku může ležet pouze v rozsahu <-128,+127> bajtů.
Další skupinou instrukcí, které umožňují větvení programu, jsou tzv.nepodmíněné instrukce:

nepodmíněný skok v rozsahu <-128,+127> SJMP <adr>
nepodmíněný skok v rozsahu 2 kB AJMP <adr>
nepodmíněný skok v celém adr. rozsahu LJMP <adr>
nepodmíněný skok v celém adr. rozsahu JMP @A+DPTR

Význam a použití prvních třech instrukcí pro nepodmíněný skok je zřejmý - liší se pouze v rozsahu, ve kterém může ležet adresa skoku. Povšimněme si blíže instrukce JMP @A+DPTR. Tato instrukce provede skok na cílovou adresu, která se vypočítá jako součet obsahu akumulátoru a DPTR. To je podstatný rozdíl oproti předchozím typům instrukcí, kdy cílovou adresu skoku jsme museli znát již při překladu instrukce do strojového kódu, kdežto adresa skoku u instrukce JMP @A+DPTR se vypočítává až **za běhu programu**!
Příklad:
Máme provést rozskok na jednu z osmi adres programu v závislosti na obsahu akumulátoru, který může nabývat hodnot z intervalu 1..8. Např. je-li A=2, skočí se na podprogram DRUHA.

```
MOV DPTR,#TAB ;adresa rozskokové tab.
DEC A ;korekce na počátek tab.
RL A ;násobení A dvěma (prvky
JMP @A+DPTR ;v tabulce jsou 2-bytové)

TAB: AJMP PRVNI
AJMP DRUHA
.
```

.
AJMP SEDMA

VYTVÁŘENÍ CYKLŮ

Při programování cyklů se používají v podstatě tři postupy, známe z vyšších programovacích jazyků: **repeat - until**, **while** a **for**.

---

repeat
.
.
.
until <podmínka>

---

while <podmínka> do
.
.
.
end

---

for <počet cyklů> do
.
.
.
end

A. U cyklu repeat-until se podmínka pro ukončení cyklu testuje vždy až na konci cyklu - vždy tedy dojde k tomu, že program projde cyklem alespoň jednou.
Příklad:

REPEAT:
.
.
;tělo cyklu
.
.
JC REPEAT ;until <podmínka>

Pokud je příznak C nastaven, program přejde na návěští REPEAT a celý cyklus se opakuje znovu.
B. Cyklus typu **while** vyhodnocuje podmínku pro vstup do cyklu na jeho počátku - může se tedy stát, že program cyklem neprojde ani jednou.

WHILE: JNC KONEC
.
.
;tělo cyklu
.
.
SJMP WHILE
KONEC:

Pokud příznak C není nastaven, provede se skok na návěští KONEC a program se do cyklu nedostane.

C. Cyklus **for** je tzv. tvrdý počítaný cyklus. Na rozdíl od předchozích typů je počet průchodů cyklu předem znám. Pro tento účel je přímo předeslána instrukce:
odečti a skoč, není-li výsledek nula DJNZ <oper>,<adr>

Příklad:
Jedno z možných použití tohoto cyklu je při vytváření časových smyček. Např zpoždění 100 Us při krystalu 6MHz:

```
;zpozdeni 100 mikrosecund (6 MHz)
DELAY: MOV R0,#24 ;naplň počítadlo cyklů
DJNZ R0,$ ;skáče sám na sebe (24x)
NOP
```

Poznámka: Pro výpočet časových smyček je vždy nutmo znát frekvenci krystalu, se kterouprocesor pracuje a potom délku každé instrukce v cyklech. Pro náš případ:

```
MOV R0,#23 ; 2 us
DJNZ R0,$ ; 24*4=96 us
NOP ; 2 Us
————————
100 us
```

K vytváření cyklů a větvení programu slouží také instrukce:
porovnej a skoč CJNE <oper1>,<oper2>,<adr>
při nerovnosti
Tato instrukce nejprve porovná oba operandy a pokud se nerovnají, provede relativní skok na zadanou adresu, jinak program pokračuje další instrukcí. Tato instrukce je v celém repertoáru instrukcí 8051 jedinná, která porovnává dva operandy a nastavuje příznak C beze změny hodnot operandů !!!!
Příklad:
Pro vytvoření cyklu repeat-until:

```
MOV B,#5 ;konečná hodnota
MOV A,#0 ;počáteční hodnota
REPEAT: INC A
CJNE A,B,REPEAT ;porovnej A a B, pokud se
;liší, skoč na REPEAT
```

Pro vytvoření cyklu for:

```
MOV R0,#0
FOR: .
;tělo cyklu for
.
INC R0
CJNE R0,#10,FOR ;dokud je R0 menší než 10,
;opakuj cyklus
```

7. Binární a dekadická aritmetika

Pro jednoduché aritmetické úlohy, tj.sčítání odčítání, násobení a dělení má 8051 tyto instrukce:
sčítání ADD A,<operand>
sčítání s přičtením příznaku C ADDC A,<operand>
odečítání SUBB A,<operand>
zvyš obsah o jedničku INC <operand>
sniž obsah o jedničku DEC <operand>
násobení MUL AB

dělení DIV AB
desítková úprava A DA A

Instrukce sčítání ADD přičte k obsahu akumulátoru zadaný operand. Instrukce ADDC přičte navíc k výsledku ještě obsah C.

Příklady:

Sečtěte dvě 16-ti bitová čísla a výsledek uložte na místo prvního z nich.

```
OPER1 EQU 20H
EQU 21H
OPER2 EQU 22H
MOV A,OPER1
ADD A,OPER2
MOV VYSL,A
```

8. Používání podprogramů

Pro používání podprogramů jsou v instrukčním souboru instrukce:

volání podprogramu v rozsahu 2 kb ACALL <adr>
volání podpr. v celém adres.rozsahu LCALL <adr>
návrat z podprogramu RET
návrat z obsluhy přerušení RETI

Instrukce ACALL a LCALL při své aktivaci **zvýší** ukazatel zásobníku SP o 2, uloží adresu následující instrukce na zásobník a předá řízení programu na cílovou adresu.

Instrukce RET předá řízení na adresu, uloženou na vrcholu zásobníku a poté odečte od SP dvojku - dochází tedy ke **snížení** zásobníku o jednu úroveň.

Činnost instrukce RETI z hlediska předání řízení programu na adresu uloženou na zásobníku je totožná s RET, ale navíc RETI povoluje přerušení - používá se tedy při návratu z obsluhy přerušení.

Příklad:

Pp vyhodnocení přerušení od sériové linky se řízení předá na adresu 23H. Jeden z možných způsobů obsluhy přerušení může vypadat např. takto:

```
ORG 23H
LJMP SER_INT ;skok na obsluhu přerušení
.
.
.
ORG 500H ;adresa obsluhy přerušení
;může být n alibovolném místě
;zde např. 500H
SER_INT:
PUSH ACC
PUSH B ;úschova geristrů
PUSH PSW
.
. ;obsluha přerušení
.
POP PSW
POP B ;obnovení registrů
POP ACC
RETI ;návrat z přerušení, povolení
;dalších přerušení
```

PROGRAMOVÁNÍ APLIKACÍ

Cílem této části je seznámit se se základními aplikacemi mikropočítače a se základními typy vstupních a výstupních periferií, možnými způsoby jejich připojení a tvorbou obslužných programů pro obsluhu těchto standardních systémových periferií.

Jednotlivé typické periferie mikropočítače jsou realizovány samostatnými moduly, který je možno připojit na jeden 8-bitový port mikropočítače. Takto navržená modulární stavebnice umožňuje připojit k základnímu modulu s obvodem mikroprocesoru (mikropočítače) jeden nebo dva moduly standardní periferie.

Obvykle připojujeme jeden modul vstupní periferie (na port P3) a jeden modul výstupní periferie (na port P1). Tato konvence bodu připojení vstupního a výstupního modulu je dodržena ve všech vypracovaných programech.

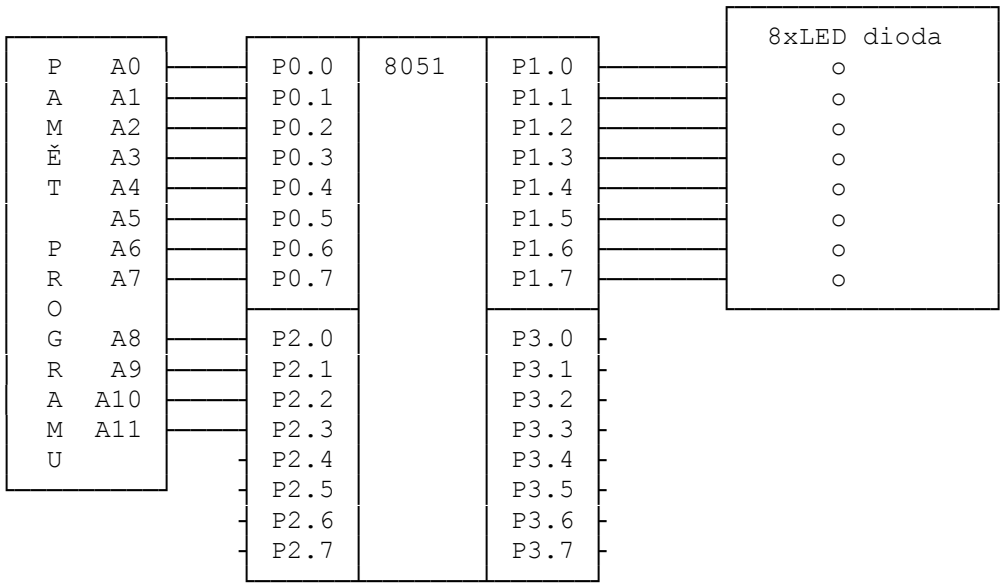
Jednotlivé navržené aplikace (zapojení modulů) jsou doplněny několika úlohami, které ilustrují použití daného modulu a způsob jeho programové obsluhy. Výčet navržených sestav a jejich zadání samozřejmě není konečný a jejich množství záleží pouze na fantazii uživatele.

Také způsob řešení daného zadání není jediný možný, nýbrž jeden z mnoha možných způsobů realizace daného zadání. Cílem však bylo postupovat od jednodušších zadání (a řešení) k náročnějším úlohám. Zároveň některé řešení demonstrují obvyklý způsob programové realizace obsluhy standardních úloh, použití direktiv překladače apod.

1. SESTAVA CPU + 8xLED

VŠEOBECNĚ : K základnímu modulu s procesorem je připojen pouze modul 8xLED na port P1. Cílem je osvojit si základní metody obsluhy V/V portů a různé způsoby generování zpoždění.

LED diody se rozsvěcují nulovou úrovní na portu. Proto je nutno provádět negaci dat vysílaných na port, ke kterému jsou připojeny LED diody.

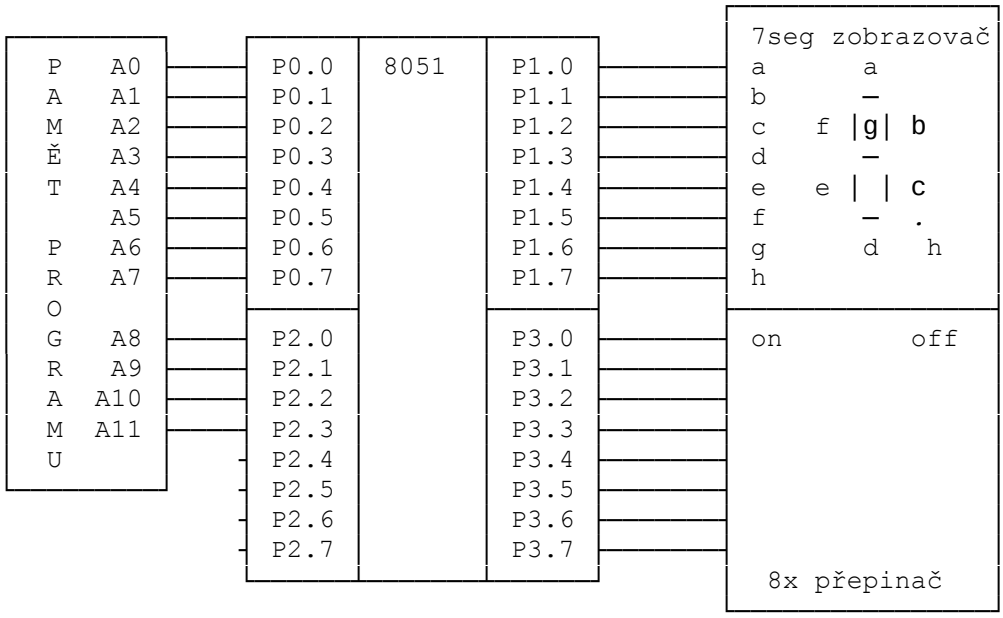


ZADÁNÍ 1 : V libovolném časovém intervalu rozsviťte a zhasněte libovolnou z osmi LED diod. Časový interval realizujte programovou smyčkou, samostatným podprogramem, pomocí čítače a pomocí přerušení generovaného čítačem.
POZNÁMKY : Zpoždění generované programovou smyčkou je vhodné pro realizaci kratších zpoždění. Použití podprogramu (nebo souboru podprogramů - viz modul DELAY.INC) umožňuje vytvořit si soustavu různě dlouhých zpoždění, která se jednoduše vyvolávají instrukcí CALL. Přesnost časového zpoždění generovaného uvedeným způsobem je poměrně vysoká, pokud není v programu použit přerušovací systém. Při vyvolání obsluhy přerušení může být totiž časovací podprogram přerušen na dobu obsluhy přerušení a výsledný čas zpoždění generovaného programovým cyklem je pak delší v závislosti na četnosti vyvolávaných přerušení a délce jejich obsluhy.
Časové zpoždění generované čítačem je možno použít pro generování časového zpoždění jehož délku lze dynamicky modifikovat i za běhu programu. Časové zpoždění generované časovačem s využitím přerušení se používá především při programování časově řízených procesů v úlohách s více procesy (např. multiplexní obsluha dynamických zobrazovacích jednotek).
ZADÁNÍ 2 : Postupně rozsviťte všechny LED diody v určitém
časovém intervalu. Ovládání LED diod realizujte samostatným řízením jednotlivých bitů a řízením celého portu v programové smyčce.
POZNÁMKY : Řešení ilustruje rozdíl mezi bitovým řízením portu a bajtovým řízením portu. Pro bitové řízení se používají instrukce SETB bit, CLR bit, CPL bit a MOV bit. Pro bajtové řízení portů můžeme použít instrukcí ANL port a ORL port a MOV port.
ZADÁNÍ 3 : V daném časovém intervalu rotujte jednu svíticí diodu vpravo nebo vlevo. Rotaci realizujte jednosměrně (cyklicky) nebo obousměrně (vratně).
POZNÁMKY : Úlohu lze řešit také pomocí přímého čtení stavu portu do střadače (registr ACC), rotací a zpětným zápisem na port.
ZADÁNÍ 4 : V daném časovém intervalu zobrazte na LED diodách binární tvar (formát) kódů 0 až 255.
POZNÁMKY : Změnou časové konstanty je možno měnit rychlost \ zobrazování.

2. SESTAVA CPU + 7 seg. zobrazovač + 8xDIP

VŠEOBECNĚ : K základnímu modulu je připojen modul 7 segmentového statického zobrazovače resp. modul DIP přepínačů. Statická obsluha 7 segmentového zobrazovače znamená, že data zapsaná na V/V port, ke kterému je zobrazovač připojen jsou trvale zobrazena. Nový zápis na port řízení statického displeje se provede až při změně zobrazovaného údaje.

Protože rozsvícení segmentů se děje nulovou úrovní, je nutno provádět negaci dat vysílaných na port s připojeným displejem.



ZADÁNÍ 5 : V daném časovém intervalu zobrazujte na 7 segmentovém displeji číslice 0 až 9

POZNÁMKY : Převodní tabulka CODE definuje ke každému číslu z intervalu 0 až 9 jeho kombinaci segmentů, která zobrazí tvar daného čísla na segmentovce. Přiřazení segmentů jednotlivým bitům je následující :

d0

```
    ___
d5 | d6 | d1
    ___
d4 |    | d2
    ___    .
```

d3 d7

Protože je tabulka definována programem a tím pádem uložena v paměti programu, je nutno pro vyzvednutí kódů z tabulky použít instrukci MOVC (move from code memory).

Řídicí proměnnou cyklu (programové smyčky) je registr R0, který obsahuje číslo 0 až 9, které je zobrazováno na segmentovce. Vlastní programová smyčka se skládá z testu na horní mez, vyzvednutí kódu segmentovky pro dané číslo a generování časové prodlevy, která určuje dobu zobrazení daného čísla na displeji (rychlost změny čísla na segmentovce).

ZADÁNÍ 6 : Na 7 segmentovém displeji zobrazujte nastavenou kódovou kombinaci na 4xDIP přepínači. Nastavená kombinace na DIP přepínači nechť je interpretována jako hexadecimální kód 0h až Fh. Změnu hodnoty DIP přepínače simulujte v IVP pomocí HSI souboru.

POZNÁMKY : Základní programová smyčka představuje událostně řízený proces. V cyklu je testován stav portu P3 (DIP přepinače) s předcházejícím stavem a pokud nastala změna je teprve vyvolána událost (proces zobrazení na segmentovku).

Proces zobrazení je realizován funkcí (podprogramem) GETKOD, který sejme hexadecimální číslo nastavené na spodní půlce DIP přepinače a zobrazí ho jako znak 0-9,A-F na segmentovce. Převodní tabulka je za tímto účelem rozšířena o segmentové kombinace pro hexadecimální čísla A až F.

Příklad zároveň demonstruje vztah (převod) mezi binární soustavou (DIP) a hexadecimální soustavou (segmentovka).

V programuje použita definice symbolického pojmenování registru pseudoinstrukcí EQU.

3. SESTAVA CPU + maticový zobrazovač 5x7 + 8xDIP

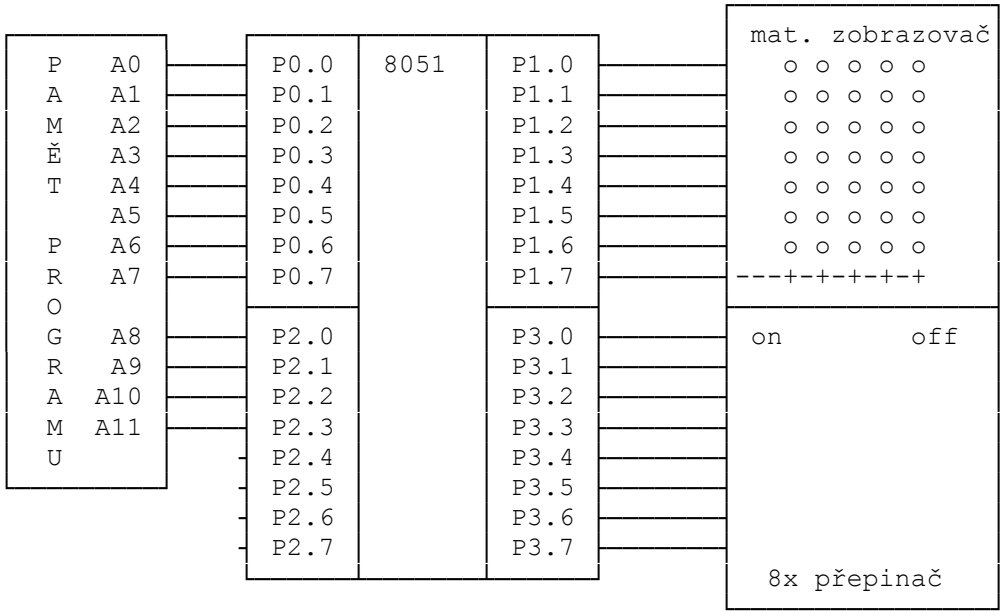
VŠEOBECNĚ : K základnímu modulu je připojen maticový zobrazovač resp. modul DIP přepínačů. Cílem zadání je ukázat způsob obsluhy dynamicky řízeného zobrazovače. Maticový zobrazovač obsahuje 5 sloupců, které mají 7 bitovou šířku. Dynamická obsluha znamená, že k portu P1 se cyklicky (časový multiplex) připojuje první až pátý sloupec zobrazovače, přičemž jsou na portu nastavena aktuální data pro připojený sloupec maticového zobrazovače.

Dostatečně rychlým přepínáním sloupců a vzhledem k určité setrvačnosti lidského oka a dosvitu segmentů vidíme svítit všechny sloupce současně.

Inkrementace čítače aktuální pozice se provádí pulzem na nejvyšším datovém bitu (MSB) portu P1. Pokud je pulz delší než 100 mikrosec, provede se vynulování čítače aktuální pozice sloupce.

Základ obsluhy maticového zobrazovače tvoří podprogram ZOBRM, který zobrazí 5 byte z vnitřní paměti dat VIDEO od adresy dané ukazovatelem. Postupnou změnou ukazatele o jedničku lze vytvořit iluzi posouvání znaku (případně textu) na zobrazovači.

Změnou časové konstanty multiplexu v podprogramu ZOBRM je možno demonstrovat vliv této konstanty na svítivost multiplexovaných sloupců (zmenšováním časové konstanty) a blikání multiplexovaných sloupců (zvětšováním časové konstanty).



ZADÁNÍ 7 : Zobrazte libovolný motiv na maticovém zobrazovači.

POZNÁMKY : Na počátku programu je nutno nejdříve inicializovat (prvopočátečně nastavit) port obsluhy maticového zobrazovače.

Jelikož úloha obsahuje jediný proces - zobrazování na maticovém displeji - je možno realizovat časový multiplex hlavní programovou smyčkou. Zobrazený grafický motiv je zadefinovaný pseudoinstrukcí DB na návěští MOTIV. První bajt motivu představuje data pro první sloupec zleva maticového zobrazovače. LSB bit dat řídí první řádek shora. Výsledek zobrazení uvidíme, pokud MOTIV otočíme o 90 proti směru hodinových ručiček.

Základní rutiny obsluhy maticového zobrazovače a převodní tabulka motivů ASCII znaků ' ' (20h) až '_' (5Fh) jsou zahrnuty do modulu MATDSP.INC, který je do programu začleněn direktivou překladače INCLUDE.

ZADÁNÍ 8 : V daném časové intervalu zobrazujte na zobrazovači jednotlivé znaky podle ASCII tabulky znaků.

POZNÁMKY : Program zobrazuje ASCII znaky od kódu 20h až po kód 5Fh včetně. Tvary (motivy) jednotlivých znaků je možno upravit změnou tabulky MASCII. Doplněním tabulky o kódy 60h až 7Fh a úpravou programu je možno zobrazit i malá písmena.

Rozšířením tabulky MASCII i o motivy pro kódy větší než 80h lze zadefinovat znaky národní abecedy a tím pádem zobrazovat i texty s použitím diakritiky.

ZADÁNÍ 9 : Na maticovém zobrazovači zobrazte znak jehož ASCII kód je nastaven na modulu DIP přepínačů.

POZNÁMKY : Úloha objasní základní zákonitosti ASCII tabulky znaků, jejíž znaky zadávány v hexadekadické soustavě jsou zobrazovány na zobrazovači.

Základní smyčka provádí neustálé zobrazování motivu na zobrazovači, přičemž je kontrolován stav DIP přepinače a pokud nastala změna oproti poslednímu stavu, pak je proveden převod motivu nového znaku z paměti programu (instrukcí MOVC) do vnitřní paměti dat, ze které probíhá zobrazování na displej rutinou ZOBRM.

Program obsahuje také funkci testu na zadaný interval kódů. Pokud leží zadaný kód mimo tento interval je zobrazen na displej standartní (DEFAULT) motiv.

ZADÁNÍ 10 : Realizujte vodorovné posouvání znaku, jehož ASCII kód je zadán DIP přepinačem, směrem zprava doleva na zobrazovači danou rychlostí.

POZNÁMKY : Úloha je realizována dvěma procesy. Proces na pozadí (vykonávaný v hlavní programové smyčce po většinu času) provádí zobrazování motivu od adresy dané ukazovatelem do oblasti VIDEO na displej. Proces vyvolávaný pomocí přerušení od časovače provádí posun ukazovatele do oblasti VIDEO.

Nastavení ukazatele definuje obsah dat, která budou zobrazeny v prvním sloupci displeje. Posunutím tohoto ukazatele o jedničku se v prvním sloupci displeje začne zobrazovat obsah druhého sloupce a tím se vytvoří iluze posunutí znaku o sloupec vlevo.

Pomocí konstant řídicích četnost vyvolání přerušení (inicializační obsah čítače a obsah registru čítání počtu přerušení) lze nastavit (měnit) rychlost posuvu znaku na displeji. Konstanta SHRATE je nepřímo úměrná rychlosti a konstanta SHSEED je přímo úměrná rychlost posuvu.

ZADÁNÍ 11 : Realizujte vodorovné posouvání daného textu na maticovém displeji vlevo danou rychlostí.

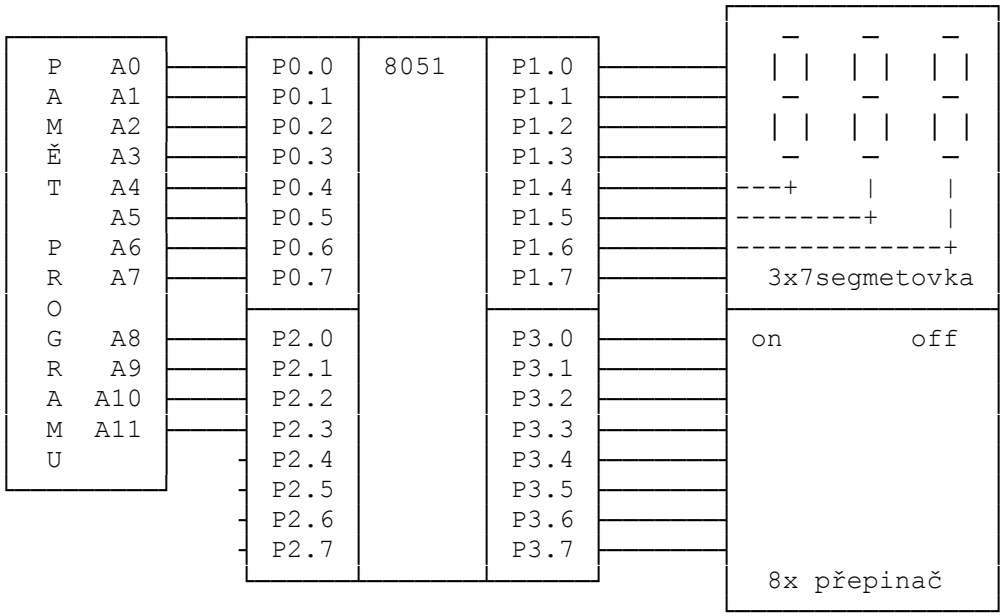
POZNÁMKY : Program provede nejdříve sestavení motivů zobrazovaných znaků textu do oblasti vnitřní paměti dat pojmenované VIDEO přičemž si uloží směrník na začátek pole motivů - adresa dat prvního sloupce prvního motivu (PRVNI) a konce pole motivů - adresa dat prvního sloupce posledního motivu.

Pak jsou spuštěny dva procesy a stejně jako v zadání 10 probíhá jednak neustálé zobrazování daného výřezu z pole motivů znaků textu a v časovém intervalu daném přerušením od čítače je posouván směrník ukazující na data prvního zobrazovaného sloupce v poli motivů textu, čímž vzniká iluze posunu textu na maticovém zobrazovači.

Pro zadání textu je v programu použita konvence jazyka PASCAL, kdy řetězec je definován v prvním bajtu počtem znaků, za kterým následují vlastní znaky řetězce.

4. SESTAVA CPU + 3x7 seg. zobrazovač + 8xDIP

VŠEOBECNĚ : Připojení třímístného BCD displeje k 8 bitovému portu je realizováno pomocí dynamické obsluhy. Dynamická obsluha spočívá, podobně jako při maticovém zobrazovači, v časově multiplexované obsluze jednotlivých pozic třímístného displeje. 8 bitový port je přitom rozdělen na dvě poloviny. Dolní polovina (nibble) obsahuje BCD kódované číslo, které se má zobrazit. Horní polovina obsahuje číslo pozice zobrazovače na displeji v kódu 1 z N, na kterém se má daná hodnota zobrazit. V našem případě jsou využity pouze tři pozice ze čtyř možných.



ZADÁNÍ 12 : Napište program, který bude na třímístném displeji postupně zobrazovat čísla 0 až 99. Vedoucí nuly nechť jsou zobrazeny.

POZNÁMKY : Obsluha segmentového displeje je založena, podobně jako u maticového zobrazovače, na vytvoření oblasti paměti, jejíž obsah je v daném časovém intervalu zapisován na port řízení displeje. Tato oblast je opět symbolicky pojmenována VIDEO.

Úloha je opět rozdělena na dva procesy. Proces na pozadí v hlavní programové smyčce inkrementuje v daném časové intervalu proměnnou cyklu, jejíž hodnota je pak konvertována do oblasti VIDEO podprogramem TOCIS.

Proces zobrazování čísla na displeji zapíše při každém přerušení hodnotu a číslo pozice na daný port a posune ukazatel řízení zobrazení na další pozici. Při dosažení pravé krajní pozice je ukazatel nastaven zpět na levou krajní pozici.

ZADÁNÍ 13 : Napište program, který bude na třímístném displeji postupně zobrazovat čísla 0 až 999. Vedoucí nuly nechť jsou potlačeny.

POZNÁMKY : Program je rozšířen z bajtového formátu čítání na dvoubajtový formát (word) čítání, ale stále v BCD soustavě. Zároveň je procedura TOCIS doplněna o mechanizmus testování první platné (nenulové) pozice a potlačování nevýznamných nul.

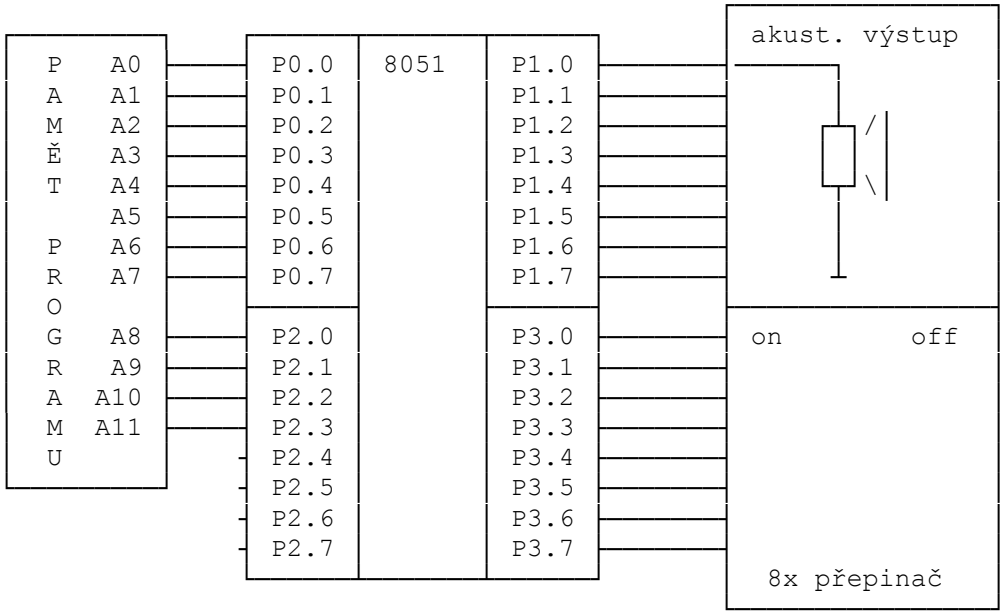
ZADÁNÍ 14 : K portu P3 připojte modul 8xDIP. Napište program, který převede nastavenou kombinaci na DIP přepínačích (port P3) na dekadické číslo a zobrazí jej na třímístném displeji. Kombinace nechť je chápána buď jako 2xBCD číslo v rozsahu 0 až 99 (nedovolené kombinace budou ignorovány) nebo jako BC číslo v rozsahu 0 až 255.

POZNÁMKY : První řešení (zobrazení BCD čísla) je obdobné jako předchozí. U varianty se zobrazováním BC čísla je program doplněn o proceduru BCBCD výstupní konverze. Výstupní konverze spolu se vstupní konverzí představuje velmi častou úlohu řešenou v mikroprocesorových systémech tam, kde dochází ke vstupu (zadávání) číselných údajů a jejich výstupu (zobrazování).

V programu je použita výstupní konverze metodou postupného dělení vzhledem k tomu, že obvod obsahuje instrukce dělení.

5. SESTAVA CPU + reproduktor

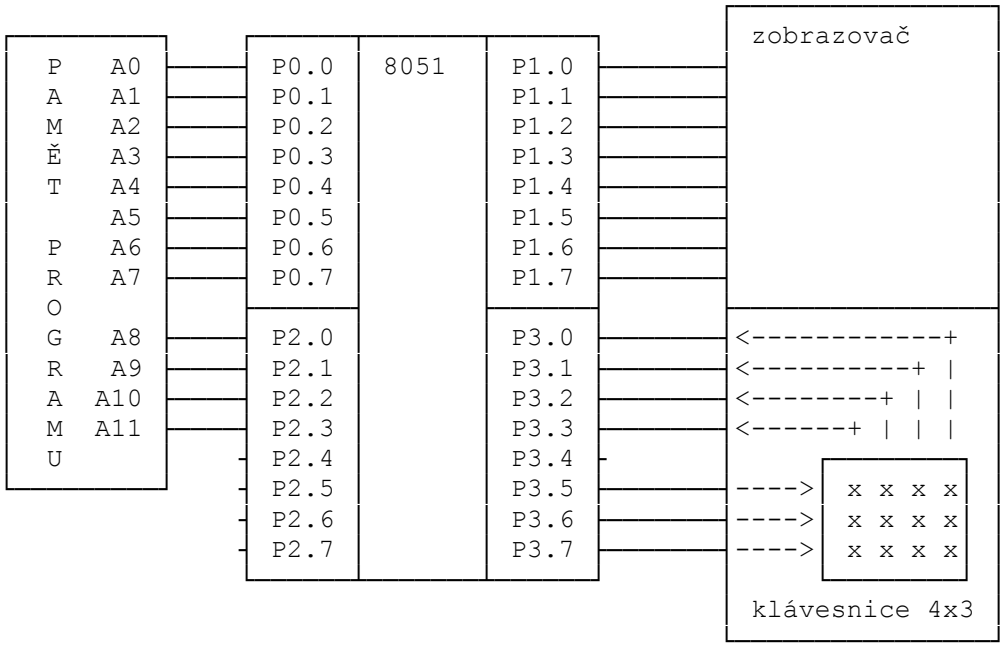
VŠEOBECNĚ : Pomocí akustického výstupu (reproduktoru, piezoelektrického měniče) je možno vytvářet širokou škálu akustických signálů, které mají v časté použití v aplikacích řízených mikroprocesory.
Nejjednodušší řízení akustické periferie se provádí pomocí jednoho drátu (bitu V/V portu). Tím že v určitém intervalu měníme stav tohoto bitu (instrukcí CPL bit) vytvářímе tón určitého kmitočtu.
Tím, že tuto změnu bitu provádíme po určitou dobu, řídíme délku trvání akustického signálu.
Všechny procesy používající akustický signál jsou silně závislé na správném výpočtu frekvence signálu a např. změna kmitočtu krystalu procesoru značně ovlivňuje výsledný zvukový efekt.



ZADÁNÍ 15 : Realizujte akustický signál (pípnutí) dané délky a kmitočtu. Poměr mezi akustických signálem a klidem nechť je stejný (symetrické pípání) nebo volitelný (asymetrické pípání).
POZNÁMKY : Úloha je opět rozdělena na dva procesy. Proces na pozadí (v hlavní programové smyčce) provádí řízení délky trvání tónu (a následující pauzy) pomocí konstanty KONDUR. Proces aktivovaný přerušením mění stav bitu řízení akustického výstupu.
Konstanta časovače generujícího přerušení KONFRE určuje výšku tón signálu (jeho kmitočet).
ZADÁNÍ 16 : Realizujte akustický signál, jehož kmitočet je dán nastavením DIP přepinačem (256 kmitočtů).
POZNÁMKY : Hodnota sejmutá z DIP přepinače přímo ovlivňuje časovou konstantu násady časovače generujícího kmitočet akustického signálu.
Aby byl výsledný tón v oblasti slyšitelnosti lidského ucha, je vlastní hodnota sejmutá z DIP vynásobena dvěma a přičtena ke konstantě SEED. Aktuální stav konstanty je uložen v paměti dat, což umožňuje jeho dynamickou změnu za běhu programu.
ZADÁNÍ 17 : Realizujte monochromatickou stupnici.
POZNÁMKY : Pro realizaci stupnice potřebujeme znát kmitočty jednotlivých tónů stupnice a z nich pak vypočítat velikost časových konstant pro generování daného kmitočtu (v závislosti na krystalu procesoru).
Pro použitý krystal 11.059 MHz jsou tyto údaje vypočteny a uvedeny v modulu MELODY.INC pro jednu oktávu.
Vlastní program pak v procesu na pozadí vybírá v zadaném časovém intervalu jednotlivé časové konstanty pro daný tón stupnice a ukládá je do proměnné TON pro potřebu druhého procesu, který tento údaj používá pro řízení intervalu přerušení čítačem (kmitočet zvuku) za běhu programu.
ZADÁNÍ 18 : Realizujte různé poplašné akustické signály.
POZNÁMKY : Obsluha procesu přerušení probíhá stejně jako v předešlých příkladech.
Proces na pozadí ovšem provádí postupnou změnu konstanty čítače a to v daném intervalu. Interval této změny určuje rychlost změny akustického signálu a interval změny pak určuje rozsah změny signálu.

6. SESTAVA CPU + Klávesnice + zobrazovač

VŠEOBECNĚ : Obsluha klávesnice je jedním ze základních rutin programů používaných v mikroprocesorově řízených přístrojích a zařízeních.
Základ obsluhy klávesnice spočívá ve vytestovávání křížového pole, které si můžeme představit jako matici. Do řádků matice vysíláme postupně rotující nulu a ze sloupců matice snímáme odezvu. Pokud byla stlačena nějaké klávesa, pak se to projeví nulovou úrovní bitu v daném sloupci.



Stlačená klávesa je pak identifikována číslem řádku a sloupce, což může představovat vnitřní kód klávesnice. Tomuto vnitřnímu kódu pak lze přiřadit libovolný kód, podle potřeby dané aplikace. Pro naše úlohy budeme z důvodu demonstrace vstupní konverze přiřazovat ASCII kód stlačené klávesy, i když by bylo možno použít jednodušší systém kódování tlačítek klávesnice.

Vlastní obsluha klávesnice sestává ze tří základních úrovní :

KEY - test stlačení klávesy (KEYpressed)

KBD - sejmutí znaku, pokud byla stlačena klávesa (KeyBoard)

CI - čekání na stlačení klávesy a sejmutí znaku (Character Input)

Podle potřeb dané úlohy je pak použita příslušná rutina.

ZADÁNÍ 19 : Pomocí připojeného modulu 8xLED ověřte algoritmus vytestovávání stlačené klávesy v křížovém poli 4x3.

POZNÁMKY : Vysílání rotující jedničky provádíme velmi zpomaleně a stav portu obsluhy klávesnice zapisujeme zároveň na řadu LED diod. Stlačíme-li libovolnou klávesu, můžeme si ověřit ve kterém okamžiku se na pravé straně řady LED diod rozsvítí odezva k příslušnému aktivovanému sloupci, které jsou zase zobrazovány v levé části řady LED diod.

ZADÁNÍ 20 : Zobrazte na připojeném 7 segmentovém displeji kód stlačené klávesy. Program čeká na stlačení klávesy.

POZNÁMKY : Program využívá modul KLA4x3.INC, ve kterém jsou implementovány tři základní rutiny obsluhy klávesnice. Základní rutinou je podprogram KEY, který testuje křížové pole klávesnice a pokud je nějaká klávesa stlačena, pak podprogram vrací ve střadači číslo řádku a sloupce (první) stlačené klávesy. Jinak vrací nulu.

Druhou rutinou je podprogram KBD, který pomocí podprogramu KEY otestuje, zda je stlačena nějaká klávesa a pokud ano, pak vypočte vnitřní kód klávesnice (číslo z intervalu 0-15), kterému pak přiřadí ASCII kód dané klávesy.

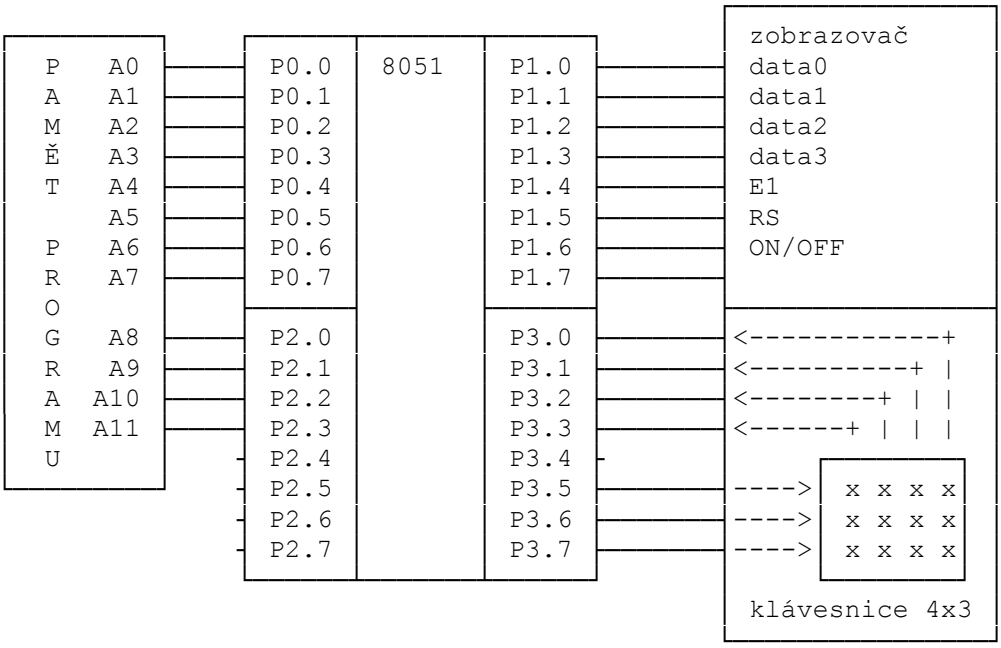
Rutina CI nejdříve čeká na puštění předchozí klávesy a pak čeká na stisk klávesy opakovaným voláním podprogramu KBD.

Z uvedeného vyplývá, že během obsluhy klávesnice jsou k dispozici tři druhy kódů klávesnice, podle úrovně obsluhy. Volba typu kódu je dána konkrétní aplikací, ve které je klávesnice použita.

ZADÁNÍ 21 : Na připojeném 3místném 7 segmentovém displeji zobrazujte postupně čísla 0 až 999. Zároveň testujte klávesnici a při stlačení libovolné nebo definované klávesy proveďte vynulování čísla na displeji.
POZNÁMKY : Základ programu je převzat ze zadání 13. Procedura generující zpoždění v hlavní programové smyčce je nahrazena opakovaným voláním procedury, která generuje kratší zpoždění. V rámci opakovaného volání procedury zpoždění je volán test stlačení klávesy.
Pokud bylo zjištěno stlačení klávesy, pak se provede požadovaná akce.
ZADÁNÍ 22 : Pomocí klávesnice zadejte dekadické číslo v intervalu 0 až 999. Průběh zadávání zobrazujte na 3místném displeji. Ukončení zadání proveďte stiskem klávesy A. Ostatní klávesy jsou ignorovány
POZNÁMKY : V tomto programu je ukázána jak vstupní konverze, tak i výstupní konverze. Vstupní konverze spočívá ve vynásobení čísla desíti a přičtení nové číslice.
Výstupní konverze BCBCD je dvoubajtová (na rozdíl od zadání 13) a je provedena metodou postupného odečítání. Číselné hodnoty jednotlivých řádů jsou ukládány do pole adresovaného registrem R1, přičemž nejvyšší dva řády jsou ignorovány (v uvedené aplikaci nemohou nastat).

7. SESTAVA CPU + Alfanumerický displej + klávesnice

VŠEOBECNĚ : Pomocí alfanumerického displeje je možno ilustrovat dokonalejší způsob zobrazování textů a editace zadávaných údajů. Alfanumerický LCD displej již obsahuje vlastní obvod řízení multiplexu LCD displeje a řadiče displeje s integrovanými základními funkcemi jako je smazání displeje, ovládání kurzoru (tvaru, pozice, viditelnosti), režim a format zápisu apod.



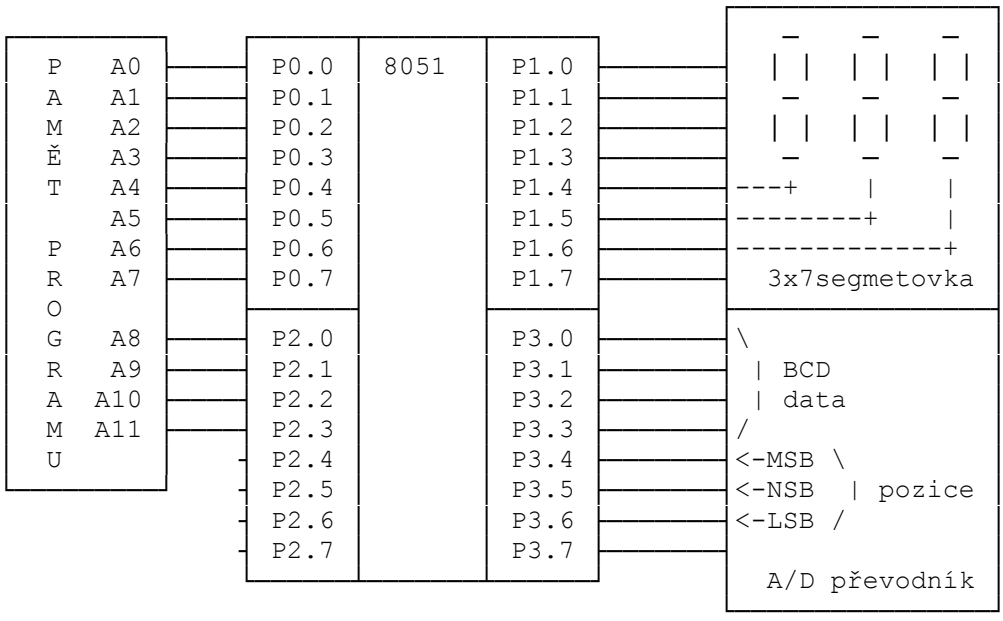
Komunikace procesoru s displejem probíhá pomocí 4 datových bitů a 3 řídicích signálů. 8-bitová data jsou do řadiče displeje vysílána ve dvou 4-bitových přenosech. Pomocí řídicích bitů se zajišťuje řízení komunikace mezi řadičem displeje a procesorem a rozlišení přenášených údajů na data k zobrazení a příkazy pro řadič. Podrobný popis je uveden v příloze.
Obsluha displeje vyžaduje určité počáteční nastavení (inicializaci), ve které se nastaví režim komunikace, formát přenášených dat a režim činnosti displeje. Poté pomocí podprogramu CO (Character Output) je možno provádět výstup dat na alfanumerický displej. Pro často používané činnosti jsou vytvořeny podprogramy, které jsou uvedeny ve stejném modulu DISPLAY.INC.
ZADÁNÍ 23 : Vypište na displej libovolný text, který je zadefinován podle konvence pro znakový řetězec (String) v jazyku PASCAL a jazyku C.

ZADÁNÍ 24 : Vypište na displej libovolný text, který je delší než délka řádku displeje. Prohlížení textu realizujte pomocí kláves řízení kurzoru (libovolné dvě tlačítka na klávesnici) tak, že při dosažení libovolného okraje displeje kurzorem se text posune v daném směru. Po stlačení definované klávesy nechť se nastaví kurzor na začátek textu.
POZNÁMKY : Program vypíše do obou řádků displeje text. Klávesa vlevo od nuly slouží pro posun kurzoru vlevo, klávesa vpravo od nuly pro posun kurzoru vpravo. Po stisku klávesy 0 se kurzor nastaví na začátek textu. Pro posun jak textu, tak i kurzoru se používají příslušné příkazy řadiče displeje.
Pro pohodlnější obsluhu je možno modifikovat obsluhu klávesnice tak, aby držení tlačítka generovalo stisknutý znak v daném intervalu (funkce autorepeat).

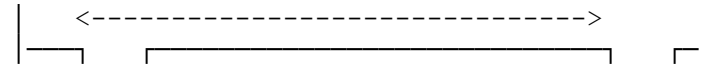
ZADÁNÍ 25 : Znaky stlačené na klávesnici vypisujte na displej. Při dosažení pravého okraje displeje, nechť je obsah displeje posunut vlevo a nový znak zapsán na pravý kraj. Vyhrazenou klávesu použijte pro funkci mazání displeje (Clear) a mazání posledního znaku na displeji (BackSpace).
ZADÁNÍ 26 : Naprogramujte celočíselnou kalkulačku pro aritmetické operace sčítání, odčítání, násobení a dělení v rozsahu 1 bajtu. Písmena A-D klávesnice použijte pro zadání aritmetické operace (+,-,*,/), čímž se ukončí zadání prvního operandu. Písmeno E použijte pro ukončení zadání druhého operandu a výpočet výsledku (ve významu symbolu =). Písmeno F použijte pro mazání posledního zadaného číselného znaku. Po stlačení libovolné klávesy nastavte displej pro zadání nového příkladu. Nutno použít klávesnici 4x4.

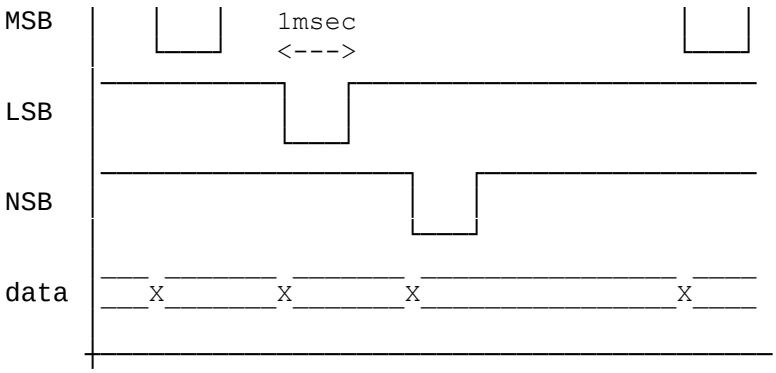
8. SESTAVA CPU + A/D převodník + zobrazovač

VŠEOBECNĚ : Modul A/D převodníku je vybaven obvodem C520 D. Tento A/D převodník převádí analogovou úroveň 0 - 0.999 V na 3 BCD cifry. Obvod C852 D provádí cyklicky převod analogové úrovně na vstupu a na výstupu generuje 7-bitový údaj formátu :
pppdddd
kde **ppp** je číslo pozice BCD cifry v kódy 1 z N
dddd je BCD hodnota na dané pozici.



Jednotlivé 7-bitové data jsou převodníkem vysílaná v pořadí MSB (nejvýznamější cifra), LSB (nejméně významná cifra), NSB (prostřední cifra). Celková doba vysílání změřené hodnoty A/D převodníkem trvá cca 6 msec.
6 msec





Časový diagram výstupu obvodu C520 D

Obsluha A/D převodníku pak spočívá v zachycení počátku vysílání tří údajů a následném zachycení všech tří 7-bitových dat v daném pořadí. Formát dat připojených na V/V port je následující :

---	LSB	NSB	MSB	BCD3	BCD2	BCD1	BCD0

.7 .6 .5 .4 .3 .2 .1 .0

ZADÁNÍ 27 : Připojením libovolného vícemístného zobrazovače realizujte voltmetr s jedním rozsahem. Přetečení a podtečení voltmetru na daném rozsahu indikujte na zobrazovači.

POZNÁMKY : Vypracovaný program řeší zadání pomocí dvou procesů. Oba procesy jsou událostně řízené (plánované). Základní proces snímání hodnoty z A/D převodníku (měření) probíhá v základní programové smyčce na pozadí spolu s generováním časové prodlevy aktivace procesu měření. Tento proces je aktivován přibližně 3 krát za sekundu. Kratší perioda způsobuje příliž častou změnu měřené hodnoty na nejnižším řádu displeje, která může působit až rušivě.

Proces zpracování změřené hodnoty představuje její zobrazování na multiplexně řízeném displeji. Zobrazování probíha v rámci obsluhy přerušení, jejíž interval je dán vlastnostmi multiplexovaného displeje.

Jednotlivé procesu spolu "komunikují" pomocí sdílené paměti dat, do které jeden proces data zapisuje a druhý proces je z této paměti vyčítá. Protože oba procesy pracují asynchronně (navzájem časově nezávisle), je nutno provést blokování čtení dat procesem zobrazování během zápisu dat procesem měření. Jinak by mohlo dojít ke čtení části dat z předcházející změřené hodnoty a části dat z nové změřené hodnoty.

Při pouhém zobrazování dat lze tento jev zanedbat vzhledem k času změření jednoho vzorku, periodě měření a periodě vypisování dat na displej.

Vzhledem k řízení procesů pomocí přerušení je nutno ještě ošetřit čtení hodnoty z A/D převodníku, které probíhá v reálném čase, opět nazávisle na procesu zobrazování.

Jiný způsob ošetření kolizí nezávisle aktivovaných procesů sdílejících společnou datovou oblast je použitím vhodně definovaných priorit přerušení jednotlivých procesů.

Při použití takového typu převodníku je také důležité v rámci počáteční inicializace provést otestování, zda obvod vysílá vůbec nějaké data.

Pro náročnější aplikace je také nutno ošetřit smyčky, které se za určitých okolností mohou stát nekonečné, tj. smyčky v podprogramech ADBYTE a CTIAD. Vhodný mechanizmus pro tyto účely je buď obvod WATCH-DOG, který ovšem na procesoru 8051 není integrován nebo mechanizmus zadefinování konečného počtu pokusů o přečtení bezchybné hodnoty.

ZADANÍ 28 : Připojte modul maticového zobrazovače a pomocí potenciometru nastavujte rychlost posuvu textu na zobrazovači.

Kapitola 3. : Architektura 8051 - stručný popis

3.1 Organizace paměti

Jednočipové mikropočítače řady 8051 mají čtyři základní adresové prostory:

64 kB paměti programu

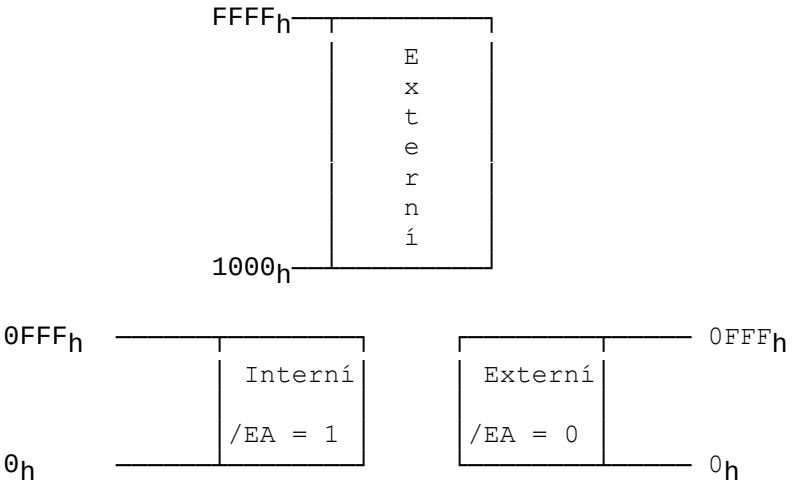
64 kB vnější paměti dat

128 B vnitřní paměti dat

21 speciálních funkčních registrů (SFR registers)

3.1.1 Paměť programu

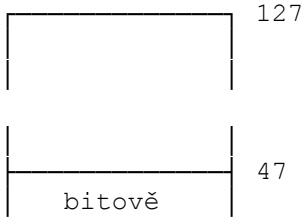
Adresový prostor paměti programu se skládá z vnitřní (interní) a vnější (externí) části - viz obr.1.

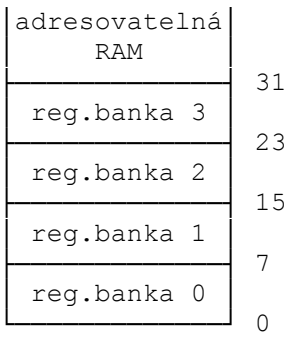


Pokud je na vývod /EA přivedena hodnota logická 1, provádí 8051 program z interní oblasti paměti, pokud adresa nepřekročí hodnotu
$0FFF_h$. Obsahy adres 1000_h až $FFFF_h$ se vybírá z vnější paměti programu. Pokud je na vývod /EA připojena logická nula, vybírají se všechny instrukce z vnější paměti programu.

3.1.2 Paměť dat

Paměť dat se skládá z vnějšího a vnitřního adresového prostoru. Externí paměť dat je přístupná pouze pomocí instrukcí typu MOVX; její velikost je 64 kB.
Oblast vnitřní paměti dat je rozdělena do dvou částí:
adresový prostor RAM
(nižších 128 bytů vnitřní paměti dat)
adresový prostor speciálních funkčních registrů (SFR)
(vyšších 128 bytů vnitřní paměti dat)
V nižší části (spodních 128 bytů) vnitřní paměti jsou uloženy čtyři banky registrů, které zabírají prvních 32 bytů (adresy 0-31). Následujících 16 bytů na adresách 32-47 paměti RAM obsahuje 128 (8x16) bitově adresovatelných pozic pro přímou práci s bity. Zbytek adresového prostoru do adresy 127 je volně k použití aplikačním programům.
Rozložení adresového prostoru vnitřní paměti RAM:





V druhé polovně adresového prostoru vnitřní paměti dat se nachází oblast speciálních funkčních registrů (SFR). Je třeba poznamenat, že tento adresový prostor není využit celý - je obsazených pouze 21 adres. Z těchto 21 speciálních funkčních registrů je 11 bitově adresovatelných.

3.2 Způsoby adresování

Jednočipové mikropočítače řady 8051 používají 5 adresovacích módů:
- adresování s registry
- přímé adresování
- nepřímé adresování s registrem
- přímý operand
- nepřímé adresování s bázovým a indexovým registrem

3.2.1 Adresování s registry

Tento způsob adresování se týká osmi pracovních registrů právě aktivní registrové sady. Nejnižší tři bity operačního kódu instrukce označují, který registr bude použit. Jako registry je možno taká adresovat ACC, B, DPTR a CY.

3.2.2 Přímé adresování

Přímé adresování je jedinný způsob adresování , který umožňuje práci se speciálními funkčními registry. Umožňuje rovněž pracovat s nižšími 128 byty vnitřní paměti RAM.

3.2.3 Nepřímé adresování s registrem

Adresování nepřímé s registrem používá obsahy registrů R0 nebo R1 v právě aktuální sadě registrů jako pointer do paměťového bloku dat. Tímto blokem může být:
- nižších 128 bytů vnitřní RAM
- 256 bytů vnější paměti dat

Tímto způsobem **nelze** adresovat oblast speciálních funkčních registrů.
Celou vnější paměť dat 64 kB je možno adresovat pouze prostředníctvím DPTR registru, který je šetnáctibitový.
Tohoto způsobu adresování používají rovněž instrukce pro práci se zásobníkem PUSH a POP. Funkci adresovacího registru ve vnitřní paměti dat přebírá registr SP, který může ukazovat kamkoli do vnitřní paměti RAM.

3.2.4 Adresování s přímým operandem

Tento způsob adresování umožňuje umístit konstantu (=přímý operand) přímo do strojového kódu instrukce.

3.2.5 Nepřímé adresování s bázovým a indexovým registrem

Použití bázového a indexového registru umožňuje přístup k datům z paměti programu. Cílová adresa se získá sečtením obsahu bázového registru (DPTR nebo PC) a indexového registru (ACC).

3.3 Seznam SFR registrů 8051

Registr Adresa Registr Adresa
* P0 80H SP 81H
DPL 82H DPH 83H
PCON 87H * TCON 88H
TMOD 89H TL0 8AH
TL1 8BH TH0 8CH
TH1 8DH * P1 90H
* SCON 98H SBUF 99H
* P2 A0H * IE A8H
* P3 B0H * IP B8H
* PSW D0H * ACC E0H
* B F0H
* bitově orientované

3.4 Symbolická pojmenování bitů 8051

7 6 5 4 3 2 1 0
P3 RD WR T1 T0 INT1 INT0 TxD RxD
PSW CY AC F0 RS1 RS0 OV - P
IP - - - PS PT1 PX1 PT0 PX0
IE EA - - ES ETI EX1 ET0 EX0
SCON SM0 SM1 SM2 REN TB8 RB8 TI RI
TCON TF1 TR1 TF0 TR0 IE1 IT1 IE0 IT0

3.5 Seznam instrukcí 8051

obj. code menmonic operand byte cycle
_____

11xx ACALL adr11 2 2
31xx ACALL adr11 2 2
51xx ACALL adr11 2 2
71xx ACALL adr11 2 2
91xx ACALL adr11 2 2
B1xx ACALL adr11 2 2
D1xx ACALL adr11 2 2
F1xx ACALL adr11 2 2
28 ADD A,R0 1 1
29 ADD A,R1 1 1
2A ADD A,R2 1 1
2B ADD A,R3 1 1

2C ADD A,R4 1 1
2D ADD A,R5 1 1
2E ADD A,R6 1 1
2F ADD A,R7 1 1
25xx ADD A,direct 2 1
26 ADD A,@R0 1 1
27 ADD A,@R1 1 1
24xx ADD A,#DATA 2 1
38 ADDC A,R0 1 1
39 ADDC A,R1 1 1
3A ADDC A,R3 1 1
3B ADDC A,R2 1 1
3C ADDC A,R4 1 1
3D ADDC A,R5 1 1
3E ADDC A,R6 1 1
3F ADDC A,R7 1 1
obj. code menmonic operand byte cycle

35xx ADDC A,direct 2 1
36 ADDC A,@R0 1 1
37 ADDC A,@R1 1 1
34xx ADDC A,#DATA 2 1
01xx AJMP adr11 2 2
21xx AJMP adr11 2 2
41xx AJMP adr11 2 2
61xx AJMP adr11 2 2
81xx AJMP adr11 2 2
A1xx AJMP adr11 2 2
C1xx AJMP adr11 2 2
E1xx AJMP adr11 2 2
58 ANL A,R0 1 1
59 ANL A,R1 1 1
5A ANL A,R2 1 1
5B ANL A,R3 1 1
5C ANL A,R4 1 1
5D ANL A,R5 1 1
5E ANL A,R6 1 1
5F ANL A,R7 1 1
55xx ANL A,direct 2 1
56 ANL A,@R0 1 1
57 ANL A,@R1 1 1
54xx ANL A,#DATA 2 1
52xx ANL direct,A 2 1
53xxxx ANL direct,#data 3 2
82xx ANL C,bit 2 2
B0xx ANL C,/bit 2 2
B5xxxx CJNE A,direct,rel 3 2
B4xxxx CJNE A,#data,rel 3 2
B8xxxx CJNE R0,#data,rel 3 2
B9xxxx CJNE R1,#data,rel 3 2
obj. code menmonic operand byte cycle

BAxxxx CJNE R2,#data,rel 3 2

BBxxxx CJNE R3,#data,rel 3 2
BCxxxx CJNE R4,#data,rel 3 2
BDxxxx CJNE R5,#data,rel 3 2
BExxxx CJNE R6,#data,rel 3 2
BFxxxx CJNE R7,#data,rel 3 2
E4 CLR A 1 1
C2xx CLR bit 2 1
C3 CLR C 1 1
F4 CPL A 1 1
B2xx CPL bit 2 1
B3 CPL C 1 1
D4 DA A 1 1
14 DEC A 1 1
15xx DEC direct 2 1
18 DEC R0 1 1
19 DEC R1 1 1
1A DEC R2 1 1
1B DEC R3 1 1
1C DEC R4 1 1
1D DEC R5 1 1
1E DEC R6 1 1
1F DEC R7 1 1
16 DEC @R0 1 1
17 DEC @R1 1 1
84 DIV AB 1 4
D5xxxx DJNZ direct,rel 3 2
D8xx DJNZ R0,rel 2 2
D9xx DJNZ R1,rel 2 2
DAxx DJNZ R2,rel 2 2
DBxx DJNZ R3,rel 2 2
DCxx DJNZ R4,rel 2 2
obj. code menmonic operand byte cycle

DDxx DJNZ R5,rel 2 2
DExx DJNZ R6,rel 2 2
DFxx DJNZ R7,rel 2 2
04 INC A 1 1
05xx INC direct 2 1
A3 INC DPTR 1 2
08 INC R0 1 1
09 INC R1 1 1
0A INC R2 1 1
0B INC R3 1 1
0C INC R4 1 1
0D INC R5 1 1
0E INC R6 1 1
0F INC R7 1 1
06 INC @R0 1 1
07 INC @R1 1 1
20xxxx JB bit,rel 3 2
10xxxx JBC bit,rel 3 2
40XX JC rel 2 2

73 JMP @A+DPTR 1 2
30xxxx JNB bit,rel 3 2
50xx JNC rel 2 2
70xx JNZ rel 2 2
60xx JZ rel 2 2
12xxxx LCALL adr16 3 2
02xxxx LJMP adr16 3 2
E8 MOV A,R0 1 1
E9 MOV A,R1 1 1
EA MOV A,R2 1 1
EB MOV A,R3 1 1
EC MOV A,R4 1 1
ED MOV A,R5 1 1
obj. code menmonic operand byte cycle

EE MOV A,R6 1 1
EF MOV A,R7 1 1
E5xx MOV A,direct 2 1
E6 MOV A,@R0 1 1
E7 MOV A,@R1 1 1
74xx MOV A,#DATA 2 1
A2xx MOV C,bit 2 1
92xx MOV bit,C 2 2
F8 MOV R0,A 1 1
F9 MOV R1,A 1 1
FA MOV R2,A 1 1
FB MOV R3,A 1 1
FC MOV R4,A 1 1
FD MOV R5,A 1 1
FE MOV R6,A 1 1
FF MOV R7,A 1 1
A8xx MOV R0,direct 2 2
A9xx MOV R1,direct 2 2
AAxx MOV R2,direct 2 2
ABxx MOV R3,direct 2 2
ACxx MOV R4,direct 2 2
ADxx MOV R5,direct 2 2
AExx MOV R6,direct 2 2
AFxx MOV R7,direct 2 2
78xx MOV R0,#data 2 1
79xx MOV R1,#data 2 1
7Axx MOV R2,#data 2 1
7Bxx MOV R3,#data 2 1
7Cxx MOV R4,#data 2 1
7Dxx MOV R5,#data 2 1
7Exx MOV R6,#data 2 1
7Fxx MOV R7,#data 2 1
obj. code menmonic operand byte cycle

F5xx MOV direct,A 2 1
88xx MOV direct,R0 2 2
89xx MOV direct,R1 2 2
8Axx MOV direct,R2 2 2

8Bxx MOV direct,R3 2 2
8Cxx MOV direct,R4 2 2
8Dxx MOV direct,R5 2 2
8Exx MOV direct,R6 2 2
8Fxx MOV direct,R7 2 2
85xxxx MOV direct,direct 3 2
86xx MOV direct,@R0 2 2
87xx MOV direct,@R1 2 2
75xxxx MOV direct,#data 3 2
F6 MOV @R0,A 1 1
F7 MOV @R1,A 1 1
A6xx MOV @R0,direct 2 2
A7xx MOV @R1,direct 2 2
76xx MOV @R0,#data 2 1
77xx MOV @R1,#data 2 1
90xxxx MOV DPTR,#data16 3 2
93 MOVC A,@A+DPTR 1 2
83 MOVC A,@A+PC 1 2
E0 MOVX A,@DPTR 1 2
E2 MOVX A,@R0 1 2
E3 MOVX A,@R1 1 2
F0 MOVX @DPTR,A 1 2
F2 MOVX @R0,A 1 2
F3 MOVX @R1,A 1 2
A4 MUL AB 1 4
00 NOP 1 1
48 ORL A,R0 1 1
49 ORL A,R1 1 1
obj. code menmonic operand byte cycle

4A ORL A,R2 1 1
4B ORL A,R3 1 1
4C ORL A,R4 1 1
4D ORL A,R5 1 1
4E ORL A,R6 1 1
4F ORL A,R7 1 1
45xx ORL A,direct 2 1
46 ORL A,@R0 1 1
47 ORL A,@R1 1 1
44xx ORL A,#DATA 2 1
42xx ORL direct,A 2 1
43xxxx ORL direct,#data 3 2
72xx ORL C,bit 2 2
A0xx ORL C,/bit 2 2
D0xx POP direct 2 2
C0xx PUSH direct 2 2
22 RET 1 2
32 RETI 1 2
23 RL A 1 1
33 RLC A 1 1
03 RR A 1 1
13 RRC A 1 1

D2xx SETB bit 2 1
D3 SETB C 1 1
98 SUBB A,R0 1 1
99 SUBB A,R1 1 1
9A SUBB A,R2 1 1
9B SUBB A,R3 1 1
9C SUBB A,R4 1 1
9D SUBB A,R5 1 1
9E SUBB A,R6 1 1
9F SUBB A,R7 1 1
obj. code menmonic operand byte cycle

95xx SUBB A,direct 2 1
96 SUBB A,@R0 1 1
97 SUBB A,@R1 1 1
94xx SUBB A,#DATA 2 1
C4 SWAP A 1 1
C5xx XCH A,direct 2 1
C8 XCH A,R0 1 1
C9 XCH A,R1 1 1
CA XCH A,R2 1 1
CB XCH A,R3 1 1
CC XCH A,R4 1 1
CD XCH A,R5 1 1
CE XCH A,R6 1 1
CF XCH A,R7 1 1
C6 XCH A,@R0 1 1
C7 XCH A,@R1 1 1
68 XRL A,R0 1 1
69 XRL A,R1 1 1
6A XRL A,R2 1 1
6B XRL A,R3 1 1
6C XRL A,R4 1 1
6D XRL A,R5 1 1
6E XRL A,R6 1 1
6F XRL A,R7 1 1
65xx XRL A,direct 2 1
66 XRL A,@R0 1 1
67 XRL A,@R1 1 1
64xx XRL A,#DATA 2 1
62xx XRL direct,A 2 1
63xxxx XRL direct,#data 3 2

Poznámka k použitým symbolům:
Rn .... registr R0-R7 z aktivní banky registrů
direct .... 8-mi bitová adresa do vnitřní paměti dat v rozsahu 0..255. (0-127 .. vnitřní RAM, 128-255 .. oblast SFR)
@Ri .... 8-mi bitová adresa ve vnitřní paměti dat RAM (0-127) nebo ve vnější paměti dat (v rozsahu 0-255)
#data .... 8-mi bitová konstanta

#data16.... 16-ti bitová konstanta
adr16 .... 16-ti bitová adresa. Umožňuje adresování v celém rozsahu 64 kB paměti programu
adr11 .... 11-ti bitová adresa. Umožňuje adresování ve 2 kB bloku paměti programu
rel .... 8-mi bitová adresa. Umožňuje adresování v rozsahu -128..+127 bytů
bit .... přímo adresovaný bit ve vnitřní datové paměti RAM nebo v SFR