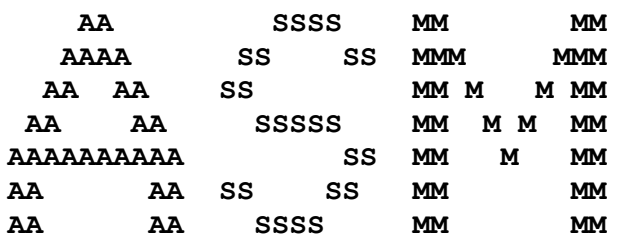
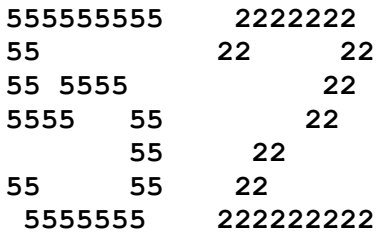




UŽIVATELSKÁ PŘÍRUČKA

Verze 1.0

Promis, Sněhurčina 700, 46015 Liberec

Vydání	Komentář	Datum
01	Originální vydání	1/92

PŘEDMLUVA

Tato příručka popisuje programování mikrokontrolérů řady MCS-8051, MCS-8052 a RUPI-44 v jazyce asembler. Obsahuje také instrukce pro ovládání asembleru ASM-52.

Pokud je použit termín MCS-51, rozumí se tím odkaz na všechny výše zmíněné typy procesorů. Všechny zmíněné typy mají stejnou architekturu a používají stejný instrukční soubor, liší se ale rozsahem interní paměti RAM i ROM a některými funkcemi speciálních funkčních registrů (dále jen SFR).

Manuál je rozdělen do několika kapitol:

Kapitola 1 - ÚVOD, popisuje programování v asembleru, obsahuje stručný přehled hardware MCS-51

Kapitola 2 - VÝRAZY A OPERANDY, popisuje, jak asembler ASM-52 vyčísluje výrazy, jaké jsou třídy výrazů, co jsou absolutní a co přemístitelné výrazy

Kapitola 3 - INSTRUKČNÍ SOUBOR, obsahuje kompletní soubor instrukcí MCS-51 v abecedním pořádku

Kapitola 4 - DIREKTIVY ASEMBLERU, popisuje způsob definice symbolů a použití všech direktiv

Kapitola 5 - ŘÍZENÍ ASEMBLERU, popisuje spuštění a ovládání průběhu činnosti asembleru

Kapitola 6 - CHYBOVÁ HLÁŠENÍ A FORMÁT VÝSTUPNÍHO SOUBORU, obsahuje seznam chybových hlášení a popis tvaru výstupního souboru (listingu).

Manuál se nezabývá detailním hardwarovým popisem MCS-51, všechny informace o hardware MCS-51 obsahuje příručka „MCS-51 User's Manual“ firmy Intel.

OBSAH

KAPITOLA 1 - ÚVOD

Tato příručka obsahuje návod na použití asembleru ASM-52 pro programování mikrokontrolérů řady MCS-51. Jako základní člen řady je uvažován typ 8051.

Asembler převádí přímo instrukce zapsané ve zdrojovém souboru na operační kódy instrukcí, které přímo informují procesor o typu požadované činnosti. Proto se programátoři používající asembler musí seznámit nejen s asemblerem, ale i s architekturou hardware použitého procesoru.

Tato kapitola jednak stručně popisuje asembler, současně si všímá některých důležitých vlastností hardware procesorů řady MCS-51.

Co je asembler

Asembler je softwarový nástroj. Je to základní prostředek pro tvorbu počítačových programů. Umožňuje provedení základní úlohy - převod symbolických instrukcí na proveditelný cílový kód. Cílový kód je možno naprogramovat do paměti programu procesoru a pak nechat provádět. Asembler Vám umožní ocenit své výhody při tvorbě programu oproti zápisu kódu přímo v hexadecimálním nebo binárním tvaru.

Mnemonický tvar operačních kódů instrukcí je výmluvný. Asembler rovněž umožňuje použít symbolická jména proměnných a určitých lokací programu, případně je možno použít symbolů jako operandy instrukcí. Jména symbolů mají být volena tak, aby zpřehledňovala program a činila jej čitelnější.

Program v asembleru obsahuje tři skupiny řádků:
- Strojové instrukce
- Direktivy
- Řídící řádky s příkazy asembleru

Strojová instrukce je řádek, obsahující mnemokód instrukce, proveditelné procesorem. Tyto instrukce jsou podrobně popsány v kapitole 3.

Direktiva asembleru je použita pro definici programových struktur a symbolů a generování neproveditelného cílového kódu (dat atp.). Direktivám asembleru je věnována kapitola 4.

Řízení asembleru je popsáno v kapitole 5. Řídící řádky (příkazy asembleru) ovlivňují průběh překladu programu.

Jak vyvíjet program

ASM-52 umožňuje uživateli modulární tvorbu programů. Následující odstavce jsou věnovány popisu základů modulárního programování.

Přednosti modulárního programování

Mnoho programů je příliš dlouhých nebo složitých, než aby mohly být napsány jako jeden zdrojový text. Programování se zjednodušší, pokud se celý program rozdělí na několik samostatných jednodušších částí, zvaných moduly. Moduly se snadněji vyvíjejí, ladí a udržují než monolitické programy.

Modulární programování si lze snadno představit jako techniku spojování funkčních celků. Jednotlivé celky jsou schopny samostatných aktivit, musí existovat pouze interfejs, který zprostředkuje přenos dat mezi jednotlivými funkčními bloky.

Jak postupovat efektivně při vývoji programu

Program se dá daleko rychleji napsat s využitím techniky modulárního programování, protože malé programové jednotky lze snadněji pochopit a vyvíjet než rozsáhlé programové celky. Programátor nadefinuje vstupní data modulu a výsledky jeho aktivity. Při ladění pak programátor použije jen nezbytné množství vstupních dat a zkontroluje výsledky. Odladěný modul pak lze včlenit do programu buď na úrovni zdrojového textu nebo pomocí linkeru.

Podprogramy a jejich vícenásobné využití

Části programů z jednoho modulu bývají často použitelné i v jiných programových modulech. Modulární programování umožňuje takové části programů odladit a uložit do knihovních souborů. Tyto podprogramy pak lze kdykoliv použít pro různé programové moduly. Naopak v monolitických programech bývají takové části programu skryty a nezřídka se opakují v podstatě tytéž programové konstrukce.

Ladění a změny snadno a rychle

Modulární programy se všeobecně snadněji ladí, než monolitické. Vzhledem k tomu, že jednotlivé moduly jsou prostřednictvím interfejsní části zpravidla dobře izolovány od ostatních programových částí, mohou být v těchto modulech problémy snadněji detekovány a odstraněny. Pokud je požadována nějaká změna, modulárně napsaný program tuto úlohu usnadňuje. Stačí pozměnit jediný programový modul a znovu jej spojit s ostatními částmi programu s jistotou, že ostatní části programu zůstaly nezměněny.

Vývoj modulárních programů s MCS-51

Tento oddíl krátce popisuje vývoj programů pomocí asembleru ASM-52, vytvářejícího přemístitelný kód.

Program, segmenty a moduly

Na počátku řešení úlohy vždy existuje problém zpracování

přesného popisu řešené úlohy a tvorby zadání. Následně pak může být úloha rozdělena na podúlohy a ty pak spojením představují řešení celkového problému.

Segment je ucelený blok paměti programu nebo dat. Segment může být absolutní nebo přemístitelný. Přemístitelný segment má jméno, je určitého typu a může mít další atributy. Segmenty se stejným jménem, ale z různých programových modulů jsou považovány za části jednoho a téhož segmentu a nazývají se "částečné segmenty". Částečné segmenty jsou zkombinovány pomocí linkeru do jediného výsledného segmentu. Absolutní segment nemá jméno a nemůže být kombinován s jinými segmenty.

Modul obsahuje jeden nebo více segmentů či částečných segmentů. Modul má jméno, které mu programátor přiřadí. Definice modulu určuje rozsah platnosti lokálních modulů, tj. takových, které jsou platné pouze uvnitř jediného modulu. Cílový soubor obsahuje jeden nebo více modulů. Spojit více modulů do výsledného cílového modulu můžete jednoduše např. pomocí příkazu COPY (např. COPY mod1+mod2 mod3).

Program je jediný absolutní modul, vzniklý spojením všech potřebných absolutních a přemístitelných modulů.

Tvorba a opravy zdrojových textů

Jakmile je problém definován, je možné začít vytvářet zdrojové texty. Zdrojovým textem je v případě asembleru ASM-52 diskový soubor, který vznikne pomocí libovolného textového editoru. Asembler přeloží zdrojový text a vytvoří výsledný soubor, obsahující údaje o překladu a o chybách, pokud se nějaké vyskytly. Chyby zdrojového textu se odstraňují opět pomocí textového editoru a celý cyklus se opakuje až do dosažení uspokojivého výsledku.

Překlad

Asembler ASM-52 přeloží zdrojový text do cílového kódu. Cílový kód produkovaný asemblerem ASM-52 je přemístitelný, pokud alespoň jeden segment modulu je přemístitelný. V opačném případě vzniká absolutní kód. Dále vzniká výsledný soubor (listing), obsahující údaje o průběhu činnosti asembleru. Pomocí příkazů lze do cílového kód začlenit ladící informace, vytvořit protokol o chybách nebo pokud je kód absolutní vytvořit přímo cílový kód v hexadecimálním tvaru (formát Intel-HEX).

Cílový soubor - je soubor, obsahující operační kód instrukcí a data. Obsahuje rovněž informace o tom, kde má být cílový kód v paměti umístěn. Může být zaveden do paměti a spuštěn. Pokud je cílový kód vytvořený ve formátu Intel-OMF, může obsahovat ladící informace (jména a hodnoty všech použitých symbolů). Asembler produkuje přemístitelný kód. Pokud ale modul obsahuje jen absolutní segmenty bez externích odkazů, je cílový kód také absolutní. Pak není potřeba použít ostatní pomocné programy (linker, konverzní programy z kódu OMF na kód HEX).

Výstupní soubor - je soubor, obsahující jednak původní zdrojový text, doplněný o generovaný cílový kód, chybová hlášení a ostatní údaje o průběhu překladu. Dále může obsahovat tabulku symbolů a křížové odkazy. Tvarem výstupního souboru se podrobně zabývá kapitola 6.

Přemísťování a spojování

Po přeložení všech programových modulů se používá pomocný program (linker) pro jejich spojení a umístění na absolutní adresy. Výstupem tohoto procesu je kompletní program a protokol o spojení.

Konverze na hexadecimální formát

Pokud je modul přeložený pomocí asembleru ASM-52 absolutní, lze požadovat generování hexadecimálního kódu přímo. Spojovací program generuje vždy soubor typu OMF, který však lze pomocí některých ladících prostředků (např. DEMON-52) přímo zavést do paměti a ladit. Některé ladící prostředky však umožňují pracovat pouze s cílovým souborem v hexadecimálním tvaru (např. UMON-52). Proto bývá potřeba konvertovat pomocí pomocného programu soubor ve tvaru OMF na tvar HEX. K tomuto účelu slouží konverzní program OH-52.

Konvence pro přípony souborů

Používání přípon souborů je důležité pro rozpoznání, o jaký meziprodukt činnosti se jedná. Programátor může ovlivnit jména všech souborů, stačí však, aby pojmenoval pouze zdrojový soubor. Je vhodné, aby přípona vyjadřovala skutečnost, že se jedná o zdrojový text (např. SRC, ASM nebo A51). Cílový kód produkovaný asemblerem ASM-52 má standardně příponu OBJ, výsledný soubor LST. Pokud se tvořil hexadecimální soubor, má příponu HEX a chybový protokol příponu ERR. Protokol o spojování mívá příponu M51, absolutní program ve tvaru OMF neužívá žádnou příponu. Vzhledem k tomu, že asembler ASM-52 nevytváří žádný pracovní soubor, není třeba se obávat žádné kolize se soubory s jinými příponami, ale se stejným jménem.

Tvorba, překlad a ladění programu pro MCS-51

Je třeba vykonat několik kroků, než realizujete svůj záměr a odladíte program pro kontrolér řady MCS-51.

Nejprve je třeba napsat zdrojový text programu. Tento text pak přeložíte asemblerem ASM-52, např. následujícím příkazovým řádkem:

```
C:\>ASM52 TEST.A52
```

Asembler se ohlásí hlavičkou, např.

```
DOS 3.30 MCS-51/52/44 Assembler Vers. 1.0
Copyright 1991 Promis
```

Asembler nyní překládá zdrojový text souboru TEST.A52. Během překladu je uživatel informován o čísle právě zpracovávaného řádku. Činnost asembleru se ukončí zprávou, např.:

```
14 Lines read
ASSEMBLY COMPLETE, 1 ERROR FOUND
```

Po překladu všech modulů se použije linker, který zkombinuje všechny segmenty do výsledného programu.

Nyní většinou následuje ladící práce. Ladění lze provádět např. pomocí ladicích prostředků UMON-52 a DEMON-52 nebo pomocí simulátorů EPROM či emulátorů. Po odladění programu následuje programování paměti EPROM, buď externí nebo na čipu, případně je možné použít tovární maskou programované procesory.

Přehled hardware MCS-51

Kompletní informace o hardware MCS-51 obsahuje příručka "MCS-51 User's Manual" firmy Intel. Zde se zaměříme pouze na popis nejdůležitějších vlastností řady procesorů MCS-51. Ty lze stručně shrnout do následujících bodů:

- Rezidentní paměť ROM nebo EPROM, až do rozsahu 16 kB, rozšiřitelná vnějšími obvody až na 64 kB
- Rezidentní paměť RAM 128 (192 pro RUPI-44, 256 pro MCS-52), obsahující 4 sady registrů a zásobník
- Možnost připojení externí paměti RAM s kapacitou až 64 kB
- 16-ti bitový programový čítač
- 8-mi bitový ukazatel zásobníku
- 2 (3 u MCS-52) programovatelné 16-ti bitové čítače/časovače
- min. 4 osmibitové obousměrné paralelní vstupně/výstupní brány
- minimálně pět zdrojů přerušení se dvěmi úrovněmi priority
- 111 instrukcí (55 typů funkcí)
- Booleovský procesor, 128 bitů a další vlajky se speciálními vlastnostmi
- Aritmeticko-logická jednotku pro funkce sčítání, odčítání, násobení, dělení a funkce AND, OR, EXCLUSIVE OR a COMPLEMENT

Adresování paměti

Procesor řady MCS-51 má pět adresových prostorů:

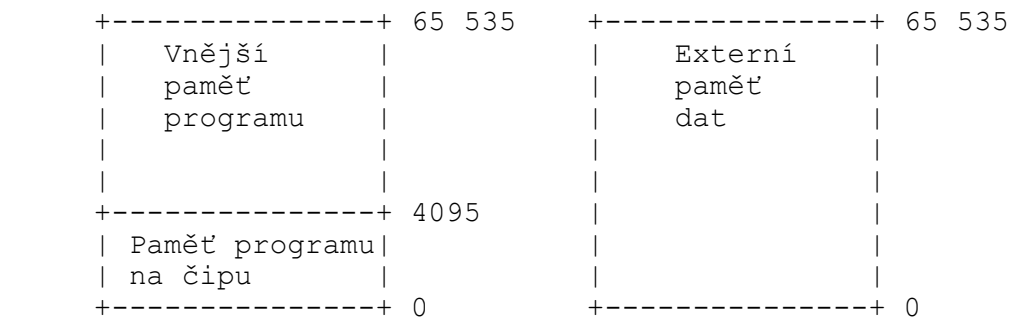
- paměť programu s rozsahem do 64 kB, z nichž až 16 kB se může nacházet přímo na čipu
- přímo adresovatelná paměť RAM na čipu s rozsahem 128 byte a další registry v oblasti SFR
- nepřímo adresovatelná paměť s rozsahem 128 byte (192 byte pro RUPI-44, 256 byte pro MCS-52) RAM na čipu, v níž je umístěn i zásobník
- externí paměť dat s rozsahem do 64 kB
- bitová paměť, společně využívané lokace přímo adresovatelné RAM na čipu

Paměť programu, externí paměť dat a paměť RAM na čipu jsou fyzicky odlišné paměťové prostory a jsou přístupné pomocí

odlišných adresovacích instrukcí. Rovněž paměť SFR a nepřímo adresovatelná paměť nad rozsah 128 byte jsou fyzicky odlišné paměťové prostory. Toto je důležité mít na zřeteli při popisu programování procesorů řady MCS-51.

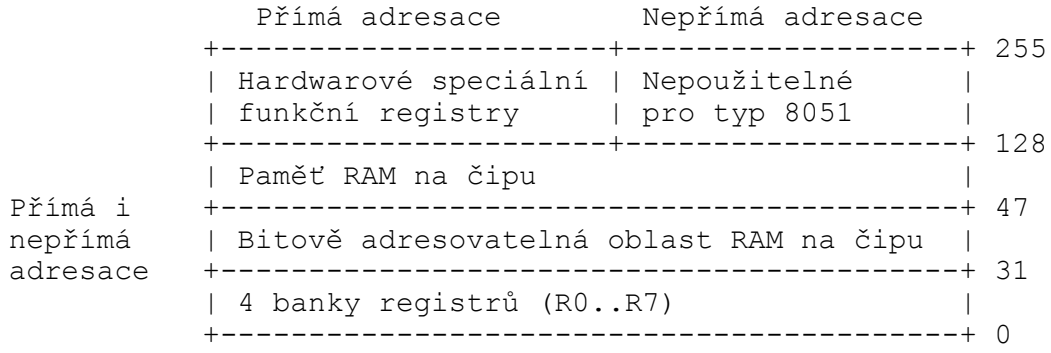
Pokud je specifikován symbol patřící do určité paměťové oblasti s chybným atributem, asembler ASM-52 generuje chybové hlášení. Způsobem definování symbolů a jejich atributů se zabývají následující kapitoly.

Pokud procesor využívá jak paměť programu na čipu, tak přídavnou vnější paměť programu, nevzniká z hlediska programování žádný rozdíl adres. Příklad takového uspořádání je na obr. 1-1.



Obr. 1-1 Příklad rozdělení paměti programu a externí paměti dat

Obrázek 1-2 znázorňuje paměť RAM na čipu, obsahující i bitově adresovatelnou oblast. Tato datová paměť obsahuje 4 banky všeobecně použitelných registrů v nejnižších 32 byte (adresy 0..1FH). Adresový prostor 20H..2FH je společně sdílený s bitově adresovatelným prostorem, tj. bit s adresou 0 je nultým bitem byte na adrese 20H. Přímo adresovatelné paměť na adresách 128-255 (80H-FFH) jsou vyhrazeny pro hardwarové speciální funkční registry (SFR), ale jen některé lokace jsou využity. Nepoužité adresy nejsou použitelné. Rozložení speciálních funkčních registrů je znázorněno v tab. 1-1.



Obr. 1-2 Paměťový prostor RAM na čipu

Adresa	Význam
*80H	Port 0 (P0)
81H	Stack pointer (SP)
82H	Data pointer, nižší byte (DPL)
83H	Data pointer, vyšší byte (DPH)
87H	Power control (PCON)
*88H	Timer - řízení (TCON)
89H	Timer - mód činnosti (TMOD)
8AH	Timer 0,nižší byte (TL0)
8BH	Timer 1,nižší byte (TL1)
8CH	Timer 0,vyšší byte (TH0)
8DH	Timer 1,vyšší byte (TH1)
*90H	Port 1 (P1)
*98H	Sériový interfejs - řízení (SCON)
99H	Sériový interfejs - data (SBUF)
*A0H	Port 2 (P2)
*A8H	Interrupt enable (IE)
*B0H	Port 3 (P3)
*B8H	Interrupt priority control (IP)
#*C8H	Timer 2 - řízení (T2CON)
#CAH	Timer 2 záchytný registr - nižší byte (RCAP2L)
#CBH	Timer 2 záchytný registr - vyšší byte (RCAP2H)
#CCH	Timer 2,nižší byte (TL2)
#CDH	Timer 2,vyšší byte (TH2)
*D0H	Stavové slovo programu (PSW)
*E0H	Akumulátor (ACC)
*F0H	Multiplikační registr (B)

* - bitově adresovatelné
# - pouze pro 8052

Tab. 1-1 Speciální funkční registry

Datové jednotky

Procesory řady MCS-51 umí manipulovat se čtyřmi skupinami datových objektů - bit, nibble (čtyři bity), byte a word (16 bitů).

Většina datových objektů jsou byte. Interní datová sběrnice má šířku 8 bitů a paměť programu, externí paměť dat i vnitřní datová paměť uchovávají data ve tvaru byte. Existuje ale i skupina instrukcí, která manipuluje s daty o velikosti 1 bit. Bit může být nastaven, vynulován, komplementován, může být prováděna logická operace s příznakem přenosu nebo může bit posloužit jako logická podmínka pro operaci skoku. Nibble je málo použit, je výhodný pro operaci s BCD kódem bez nutnosti další konverze. Instrukce používající šestnáctibitové adresy jsou realizovány pomocí datového ukazatele (DPTR) a programového čítače (PC). Oba tyto registry jsou šestnáctibitové. Relativně snadná je i implementace šestnáctibitové aritmetiky díky instrukcím "sečti s přenosem" (ADDC) a "odečti s výpůjčkou" (SUBB).

Aritmetické a logické funkce

Aritmetické instrukce jsou:

- ADD - sečtení ve dvojkově doplňkovém kódu
- ADDC - sečtení ve dvojkově doplňkovém kódu i s přenosem
- SUBB - odečtení ve dvojkově doplňkovém kódu i s výpůjčkou
- DA - dekadická korekce
- MUL - celočíselné násobení bez znaménka
- DIV - celočíselné dělení bez znaménka
- INC - zvětšení o jedničku ve dvojkově doplňkovém kódu
- DEC - zmenšení o jedničku ve dvojkově doplňkovém kódu

Příjemcem výsledků instrukcí ADD, ADDC, SUBB a DA je akumulátor. Instrukce DIV a MUL využívají dvojici registrů AB. Instrukce INC a DEC lze aplikovat na libovolný operand typu byte, včetně registrů a akumulátoru.

Logické funkce jsou:

- ANL - logický součin každého bitu dvou operandů typu byte/bit
- ORL - logický součet každého bitu dvou operandů typu byte/bit
- XRL - logický exkluzívní součet každého bitu dvou operandů typu byte
- CPL - logický komplement každého bitu dvou operandů typu byte/bit

Příjemcem výsledků logických funkcí s daty typu byte bývá akumulátor, s daty typu bit příznak přenosu. Některé logické funkce však mohou umístit svůj výsledek do libovolné proměnné typu byte nebo bit v datovém adresním prostoru.

Všeobecné registry

Procesory řady MCS-51 mají k dispozici až čtyři banky registrů pro všeobecné použití. Tyto banky jsou umístěny na nejnižších adresách paměti RAM na čipu (adresy 0..1FH). Přístup k těmto registrům je umožněn přímo pomocí speciálních symbolů asembleru R0, R1, R2, R3, R4, R5, R6 a R7. Pro změnu aktivní banky registrů je třeba modifikovat výběrové bity RS0 a RS1, obsažené ve stavovém slovu programu (PSW). Tabulka 1-2 znázorňuje adresaci registrů v závislosti na obsahu bitů RS0 a RS1.

RS1	RS0	Banka	Adresy v RAM na čipu
0	0	0	00H .. 07H
0	1	1	08H .. 0FH
1	0	2	10H .. 17H
1	1	3	08H .. 1FH

Tab. 1-2 Výběr banky registrů

Registry R0 a R1 mohou být použity pro nepřímé adresování v celém rozsahu RAM na čipu nebo pro nepřímé adresování externí paměti dat.

Zásobník

Zásobník je umístěn v RAM na čipu. Zásobník pracuje metodou LIFO (last-in, first-out). Je určen pro ukládání stavu programového čítače a důležitých pracovních registrů či proměnných umístěných v RAM na čipu, které se mají uchovat během volání podprogramů nebo při obsluze přerušení. Stav PC ukládá do zásobníku instrukce typu CALL nebo ho automaticky uloží akceptování požadavku na přerušení a obnovuje ho RET nebo RETI. Pracovní proměnné a registry se ukládají pomocí instrukce PUSH a obnovují prostřednictvím instrukce POP.

Ukazatel zásobníku (SP) obsahuje vždy adresu vrcholu zásobníku, tj. poslední adresy použité pro uschování nějakých dat. Při ukládání do zásobníku se jeho ukazatel (registr SP) zvětšuje. Po resetu je ukazatel zásobníku nastaven automaticky na hodnotu 7, tj. první data budou uložena na adresu 8. Ve většině aplikací je však nastavení SP ihned po resetu změněno tak, aby zásobník nekolidoval s bankami registrů. Pro tuto operaci se běžně používá instrukce MOV, např. MOV SP,6FH nastaví ukazatel zásobníku na adresu 6FH, takže první ukládání do zásobníku bude na adresu 70H.

Ukazatel zásobníku, registr SP v oblasti SFR, je osmibitový registr. Je nutné, aby programátor respektoval omezení, plynoucí z architektury procesorů řady 8051, resp. 8044. U procesoru typu 8051 nesmí hodnota SP nikdy překročit 127 (7FH), u 8044 191 (0BFH), jinak budou ze zásobníku vracena neplatná data. Přetečení zásobníku není ale nijak indikováno, ukazatel zásobníku se i v případě přetečení normálně zvětšuje, resp. zmenšuje.

Symbolické názvy speciálních funkčních registrů

Každý ze speciálních funkčních registrů je přístupný pomocí své adresy podle tab. 1-1, nicméně z hlediska lepší přehlednosti programu asembler ASM-52 podporuje symbolické názvy těchto registrů, které jsou předem definovány a jsou zvoleny příkazem asembleru MODE (viz. kap. 4). Symbolické názvy registrů pro procesory řady 8051 a 8052 jsou přehledně uvedeny v tab. 1-3.

Ve většině případů je použití předdeklarovaného symbolu nebo přímé adresy jedinou možností, jak manipulovat s obsahem hardwarových speciálních funkčních registrů. Existují však vyjímky, speciálně v případě akumulátoru, pro nějž lze použít jak speciální symbol asembleru "A", tak i předdeklarovaný symbol "ACC" a totéž platí i pro příznak přenosu ("C" nebo "CY"). Je třeba upozornit na to, že ač některé instrukce, sémanticky správně zapsané, mohou být chybné, případně mohou mít stejný účinek, ale nebudou optimální co do délky generovaného kódu. Například instrukce

```
MOV  C,IE
```

zkopíruje bit IE (Interrupt Enable) do bitu příznaku přenosu, kdežto instrukce

```
MOV  CY,IE
```

má sémanticky tentýž význam, ale je syntakticky chybná. Obdobně jsou sémanticky totožné a syntakticky správné obě následující instrukce, ale generují odlišný kód:

```
     MOV  A,7          ;Délka 2 byte
     MOV  ACC,7        ;Délka 3 byte
```

Instrukcím a jejich operandům se podrobně věnuje kapitola 3.

Vzhledem k tomu, že oblast SFR je mapována jako přímá datová oblast, nejsou třeba žádné speciální instrukce pro uskutečnění vstupně-výstupních operací. Stav portu P0 lze tedy prostě zjistit instrukcí:

```
    MOV  A,P0
```

Symbol	Adresa	Význam
ACC	E0H	Akumulátor
B	F0H	Multiplikační registr
DPL	82H	Data pointer, nižší byte
DPH	83H	Data pointer, vyšší byte
IE	A8H	Interrupt enable
IP	B8H	Interrupt priority control
P0	80H	Port 0
P1	90H	Port 1
P2	A0H	Port 2
P3	B0H	Port 3
PCON	87H	Power control
PSW	D0H	Stavové slovo programu
#RCAP2L	CAH	Timer 2 záchytný registr - nižší byte
#RCAP2H	CBH	Timer 2 záchytný registr - vyšší byte
SBUF	99H	Sériový interfejs - data
SCON	98H	Sériový interfejs - řízení
SP	81H	Stack pointer
#T2CON	C8H	Timer 2 - řízení
TCON	88H	Timer 0 a 1 - řízení
TH0	8CH	Timer 0,vyšší byte
TH1	8DH	Timer 1,vyšší byte
#TH2	CDH	Timer 2,vyšší byte
TL0	8AH	Timer 0,nižší byte
TL1	8BH	Timer 1,nižší byte
#TL2	CCH	Timer 2,nižší byte
TMOD	89H	Timer 0 a 1 - mód činnosti

- pouze pro 8052

Tab. 1-3 Speciální funkční registry

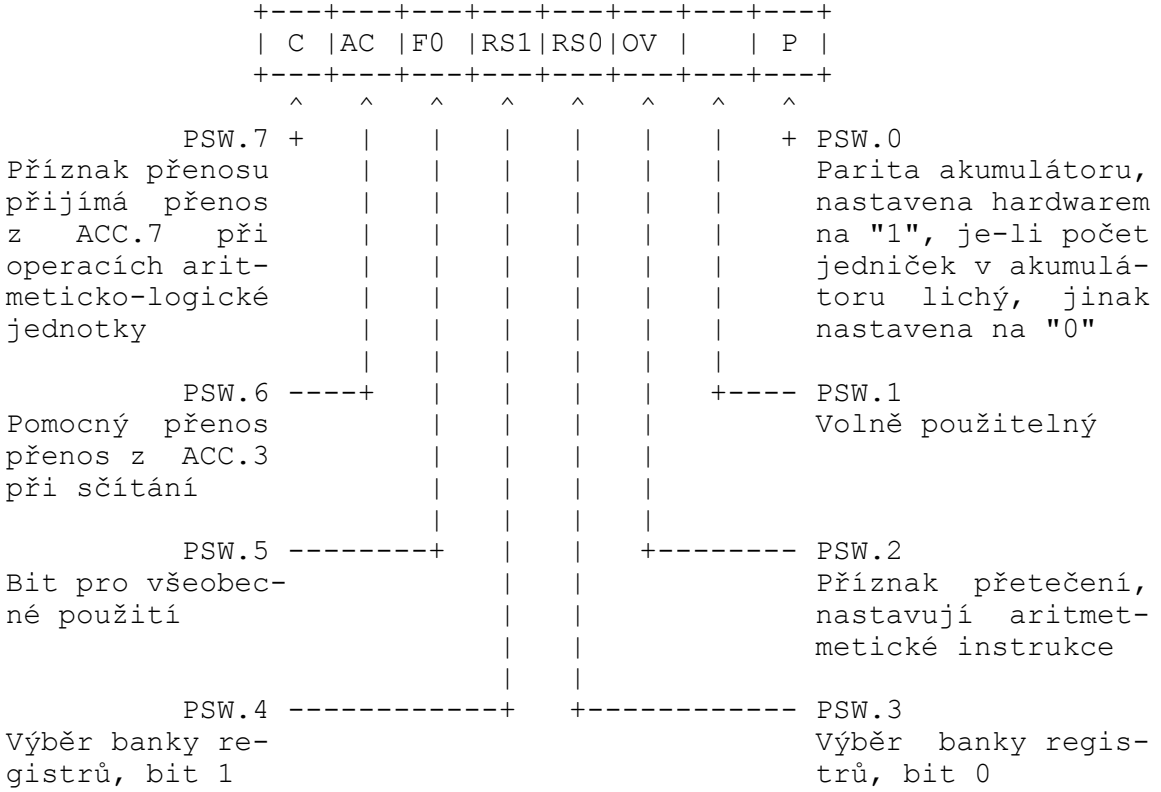
Bitová adresace

Některé ze speciálních funkčních registrů jsou bitově adresovatelné. Bity, obsažené v takových registrech lze tedy adresovat přímo, buď prostřednictvím adresy příslušného byte a offsetu bitu nebo pomocí předdeklarovaného symbolického jména bitu, pokud existuje. Pro bit 0 akumulátoru lze tedy použít

adresu zapsanou jako "ACC.0", pro příznak přenosu je možný zápis "PSW.7" a nebo je možné použít předdeklarovaný symbolický název "CY". Následující odstavce se věnují popisu speciálních funkčních registrů, které jsou bitově adresovatelné a jejichž bity mají předdeklarovaný symbolický název.

Stavové slovo programu

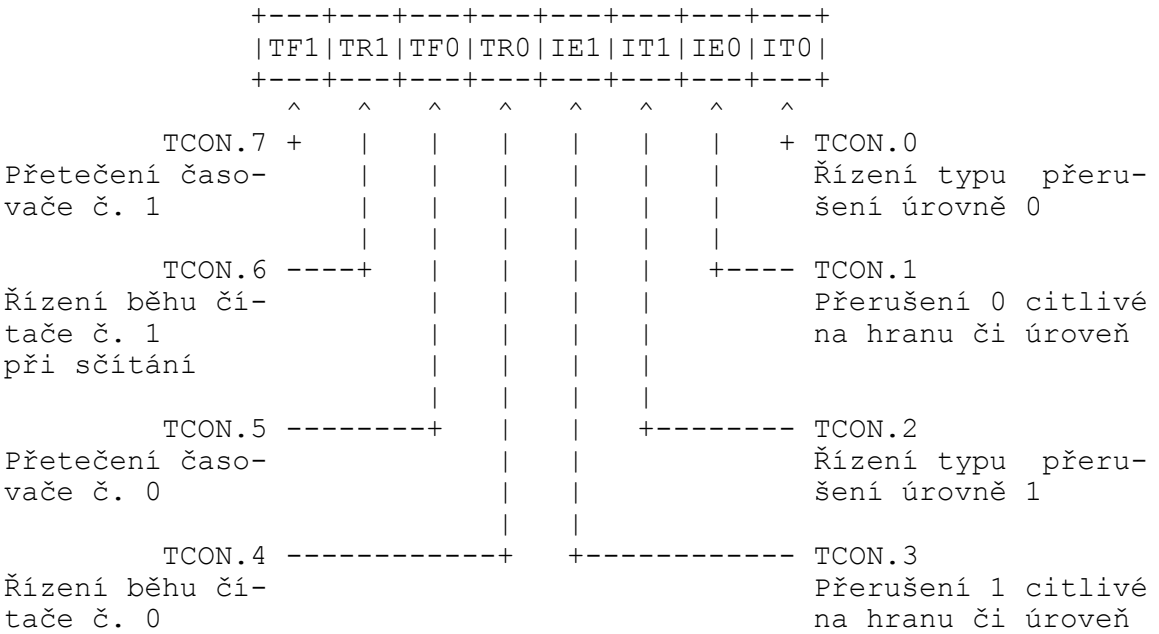
Stavové slovo programu (PSW) obsahuje různé stavové bity, které odpovídají stavu procesoru. Obr. 1-3 zobrazuje strukturu PSW i s předdeklarovanými bitovými symboly.



Obr. 1-3 Stavové slovo programu (PSW)

Čítače a časovače

Procesory řady 8051 mají dva nezávislé programovatelné čítače/časovače, řada 8052 pak disponuje ještě jedním dalším čítačem/časovačem. Čítače/časvoče jsou u řady 8051 řešeny jako šestnáctibitové a jsou řízeny pomocí dvou registrů, řízení modu (TMOD) a řízení běhu (TCON). Obr. 1-4 znázorňuje registr TCON i jeho bity. Pro detailní informace je třeba použít "MCS-51 User's Manual" nebo obdobnou publikaci.



Obr. 1-4 Registr řízení běhu čítačů/časovačů (TCON)

Vstupní a výstupní porty

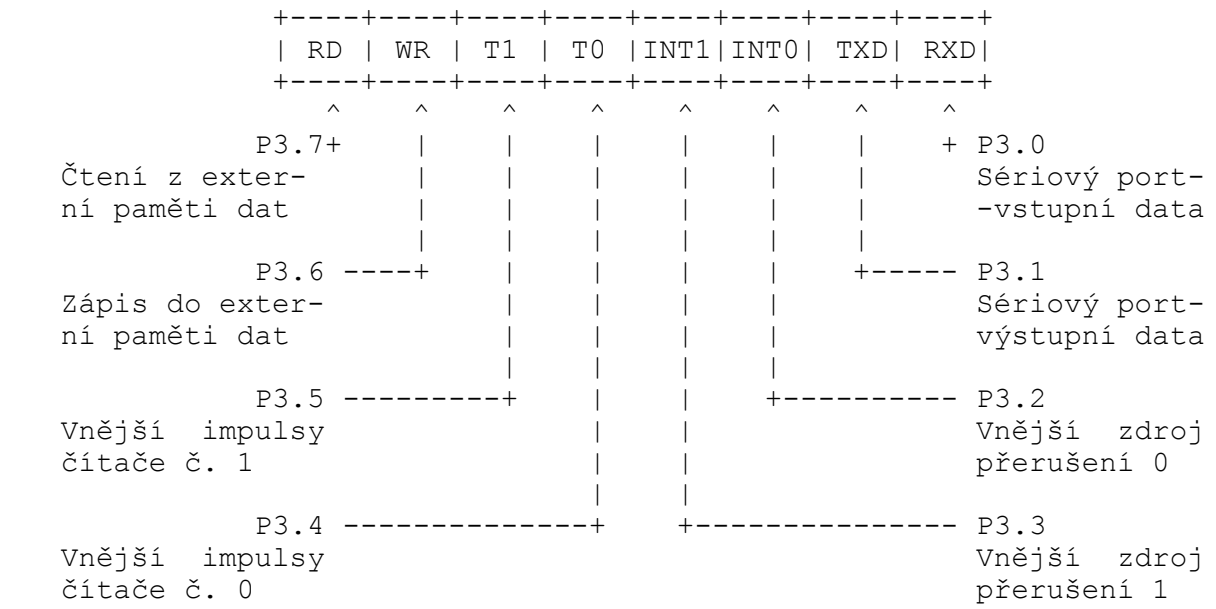
Procesory řady MCS-51 mají čtyři obousměrné osmibitové porty. Každý bit portu má vyvedený svůj pin na pouzdře procesoru. Všechny porty jsou řešeny jako obvody se záchytným registrem. Jsou adresovatelné buď jako celá osmice bitů (byte) nebo přímo pomocí adres jednotlivých bitů. Jak již bylo řečeno, umožňuje tato filozofie přímý vstup/výstup bez nutnosti existence speciálních instrukcí. Adresy jednotlivých portů jsou:

Port č.	Předdeklarovaný symbol	Adresa
0	P0	80H
1	P1	90H
2	P2	A0H
3	P3	B0H

Porty 0 a 2 jsou použité jako adresová a datová sběrnice v případě použití vnějších paměťových obvodů jak pro program, tak i pro externí data. Pokud nejsou tyto obvody použity, lze porty 0 a 2 využít jako obousměrné porty. Port 1 je určen pouze k použití jako obousměrný port, nemá žádnou speciální funkci (u typu 8051).

Port 3 lze použít také jako obousměrný port, jeho jednotlivé bity však mají speciální funkce. Jednak jsou to bity RD a WR, použité pro připojení externí datové paměti. Dále jsou zde bity T0 a T1, které slouží pro připojení zdrojů impulsů k příslušnému čítači, je-li použit ve funkci čítání externích událostí. Dále jsou zde piny INT0 a INT1, které jsou určeny pro připojení externích zdrojů přerušení a konečně piny RxD a TxD, což jsou datové signály sériového komunikačního rozhraní. Polohu jednotlivých bitů v portu 3 znázorňuje obr. 1-5.

Port 3 lze použít celý jako obousměrný port jen v tom případě, že není využito žádné ze speciálních vlastností jeho jednotlivých bitů.



Obr. 1-5 Speciální funkce bitů portu 3

Sériové rozhraní

Přítomnost sériového rozhraní přímo na čipu umožňuje použití kontroléru řady MCS-51 přímo pro komunikace bez nutnosti doplnění o komunikační obvody. Na MCS-51 je implementován duplexní UART, jehož činnost je řízena registrem řízení sériového rozhraní (SCON). Jeho struktura je naznačena na obr. 1-6, pro úplný popis je třeba použít "MCS-51 User's Manual".

SCON

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
7							0

SM0..SM2 - mód funkce sériového rozhraní
REN - povolení příjmu
RI - přerušení po ukončení příjmu
TI - přerušení po ukončení vysílání
RB8 - přijímaný bit č. 8
TB8 - vysílaný bit č. 8

Obr. 1-6 Řízení sériového rozhraní

Řízení přerušení

Pro řízení přerušovacího mechanismu jsou použity dva registry, registr povolení přerušení (IE) a registr priority přerušení (IP). Pokud je příslušný bit v registru IE nastaven na "1", pak může odpovídající zdroj přerušení přerušit činnost procesoru. Procesory řady MCS-51 nejsou schopny akceptovat požadavek na přerušení, dokud se neukončí prováděcí cyklus právě vykonávané instrukce.

Po akceptování požadavku na přerušení zakáže hardware možnost přerušení od zdroje se stejnou nebo nižší prioritou a

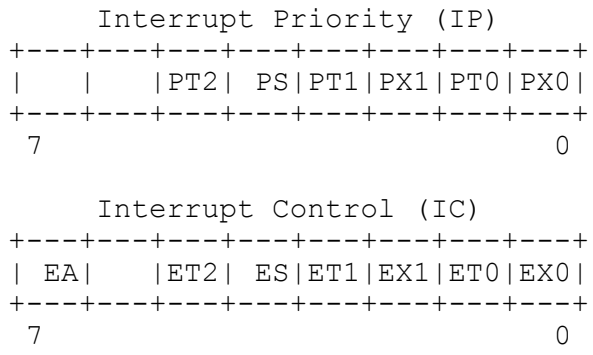
provede instrukci typu CALL na odpovídající lokaci v paměti programu. Na této lokaci musí být umístěna buď obslužná rutina přerušení a nebo, což je typičtější případ, odskok na takovou rutinu.

Rutina obsluhy přerušení musí být ukončena instrukcí RETI, čímž se znovu povolí přerušení na příslušné přerušovací úrovni. Tab. 1-4 obsahuje přehled návěští předdeklarovaných asemblerem ASM-52 a lokací paměti programu odpovídajících přerušení od daného zdroje.

Návěští	Adresa v paměti programu	Zdroj přerušení
RESET	0000H	signál reset
EXTI0	0003H	vnější přerušení 0
TIMER0	000BH	čítač/časovač 0
EXTI1	0013H	vnější přerušení 1
TIMER1	001BH	čítač/časovač 1
SINT	0023H	sériové rozhraní
TIMER2	002BH	čítač/časovač 2 (jen 8052)

Tab. 1-4 Zdroje přerušení u MCS-51

Procesory řady MCS-51 mají jen dvě přerušovací priority (0 a 1). Obr. 1-7 obsahuje popis struktury obou registrů pro řízení přerušení. Priorita přerušení 1 může způsobit přerušení obsluhy přerušení s prioritou 0, opačný případ není možný. Výskyt přerušení na libovolném zdroji nepřeruší obsluhu přerušení téže priority.



EA - povolení/zákaz všech přerušení
ET0..ET2 - povolení přerušení od čítačů/časovačů 0..2
EX0,EX1 - povolení vnějších přerušení 0 a 1
ES - povolení přerušení od sériového rozhraní
PT0..PT2 - priorita přerušení od čítačů/časovačů 0..2
PX0,PX1 - priorita vnějších přerušení 0 a 1
PS - priorita přerušení od sériového rozhraní

Obr. 1-7 Řízení přerušovacího mechanismu

Reset

Po signálu reset se nastaví důležité hardwarové registry procesoru do počátečního stavu. Tento stav popisuje tab. 1-5. Je důležité podotknout, že signál reset nemá vliv na obsah nepřímo adresovatelné paměti dat na čipu, tedy ani na obsah všeobecných registrů. Jejich stav po připojení napájení k procesoru je náhodný.

Registr	Hodnota
ACC	00H
B	00H
DPTR	0000H
IE	00H
IP	00H
P0	0FFH
P1	0FFH
P2	0FFH
P3	0FFH
PC	0000H
PCON	0xxxxxxxB u HMOS, 0xxx0000B u CHMOS verzí
PSW	00H
SCON	00H
SBUF	00H
SP	07H
TCON	00H
TMOD	00H
TH0	00H
TH1	00H
TL0	00H
TL1	00H

Tab. 1-5 Stav důležitých hardwarových registrů po resetu

Kapitola 2 - VÝRAZY A OPERANDY

Tato kapitola se zabývá popisem typů číselných výrazů a operandů, používaných asemblerem ASM-52. Je popsáno jejich použití i způsob, jakým je lze deklarovat v uživatelském programu. Kapitola se rovněž zabývá popisem vyhodnocování číselných výrazů.

Operandy

Všeobecně lze říct, že úplný zápis jednoho zdrojového řádku v asembleru ASM-52 má tento tvar:

[návěští:] instrukce [operand][,operand][,operand][;komentář]

Počet operandů závisí na typu instrukce. Operandy konkretizují činnost instrukce a ovlivňují výsledný generovaný kód.

Operandy lze rozdělit do šesti skupin:

- speciální symboly asembleru
- nepřímé adresy
- přímé datové operandy
- datové adresy (paměti na čipu)
- bitové adresy
- adresy v paměti programu

Speciální symbol asembleru je rezervované slovo, použitelné jako operand instrukce.

Nepřímé adresy používají obsah registru jako adresu dat na čipu.

Zbylé typy operandů (data, adresy bitů, dat na čipu nebo lokací v paměti programu) jsou číselné výrazy. Mohou být vyjádřeny pomocí symbolů, ale výsledkem musí být číslo. Pokud může být nějaký výraz kompletně vyčíslen už při překladu asemblerem, nazývá se absolutní výraz. V opačném případě se hovoří o přemístitelném výrazu. Rozsah povolených výsledných hodnot operandu závisí na kontextu, v jakém je výraz použit. Výraz se může skládat z předdefinovaných symbolů, ze symbolů definovaných uživatelem, čísel a asemblerských operátorů.

Jak bylo zmíněno v kapitole 1, procesory řady MCS-51 mají pět odlišných typů adresových oblastí. Těmto oblastem odpovídají i typy segmentů dle následujícího výčtu:

- bitově adresovatelný prostor	BIT
- prostor paměti programu	CODE
- přímo adresovatelná data na čipu	DATA
- nepřímo adresovatelný prostor na čipu	IDATA
- externí datová paměť	XDATA

Pokud má výraz pouze číselnou hodnotu bez příslušnosti k některému z uvedených typů segmnetů, zařadí jej asembler ASM-52 do skupiny čistě číselných hodnot (typ NUMBER).

Častým případem je, že výsledná číselná hodnota výrazu je platnou adresou pro více segmentových typů. Aby se předešlo nedorozumnění, asembler ASM-52 přiřadí každému výrazu jednu ze šesti uvedených typových tříd a při použití výrazu jako operandu pak kontroluje, zda třída výrazu odpovídá použitému kontextu (např. operandem v instrukci skoku může být pouze výraz třídy CODE nebo NUMBER).

Speciální symboly asembleru

Asembler má několik rezervovaných slov pro označení některých registrů a jejich použití jako operandů instrukcí. Tato skupinu rezervovaných slov se nazývá speciální symboly asembleru. Tabulka 2-1 shrnuje všechny tyto symboly.

Symbol	Význam
A	Akumulátor
R0,R1,R2,R3, R4,R5,R6,R7	Používá se pro označení jednoho z osmi registrů právě vybrané (aktivní) banky registrů
DPTR	Datový ukazatel - 16-ti bitový registr k adresaci paměti programu nebo externí paměti dat
PC	Programový čítač - 16-ti bitový registr obsahující adresu příští instrukce
C	Příznak přenosu (sčítaní), výpůjčka (odečítání)
AB	Dvojice registrů akumulátor/registr B použitá v instrukcích násobení (MUL) a dělení (DIV)
$	Okamžitá pozice v běžném segmentu (viz kap. 4)
AR0,AR1,AR2, AR3,AR4,AR5, AR6,AR7	Adresa registru v paměti dat na čipu. Závisí na čísle aktivní banky registrů, nastavené direktivou USING (viz kap. 4)

Tab. 2-1 Speciální symboly asembleru

Několik následujících příkladů demonstruje použití speciálních symbolů asembleru jako operandů instrukcí.

```
CLR  C              ;Nulování příznaku přenosu
SUBB A,R0           ;Odečtení obsahu registru 0
MOV  DPTR,#1234H    ;Naplnění datového ukazatele konstantou
PUSH AR7            ;Uložení obsahu R7 do zásobníku
```

Nepřímé adresování

Operandem typu nepřímá adresa se označuje použití registru, jehož obsah se chápe jako adresa dat v paměti na čipu nebo dat ve stránce externí paměti dat délky 256 byte. Pro tyto účely lze použít jedině registry R0 a R1 z aktivní banky registrů. Pro nepřímé adresování se v některých instrukcích používá šestnáctibitový registr (buď PC nebo DPTR), obsahující adresu v externí paměti dat nebo v paměti programu.

Pokud je použita nepřímá adresa dat na čipu (IDATA) a přitom nepřímá adresa má nepřípustný rozsah (např. u typu 8051 je větší než 127), pak v případě čtení dat se získají data s náhodným obsahem, v případě zápisu jsou data ztracena.

Použití registru jako nepřímé adresy indikuje použití symbolu "@" před registrem nebo symbolem registr označujícím. Platné instrukce s nepřímou adresou uvádí několik následujících příkladů:

```
MOVC A,@A+PC      ;Data z paměti programu do akumulátoru
ORL  A,@R0        ;Jejich logický součet s daty IDATA
MOVX @DPTR,A      ;Data z akumulátoru do oblasti XDATA
```

Přímý datový operand

Pokud je operandem instrukce číselný výraz, který se vyskytne jako součást cílového kódu asemblované instrukce, hovoříme o přímém datovém operandu. Jeho použití je indikováno znakem "" bezprostředně před číselným výrazem. Číselný výraz musí být platným výrazem, který se vyčíslí ještě během překladu asemblerem nebo ve fázi spojování (linkování).

Asembler ASM-52 zachází se všemi výrazy jako se šestnáctibitovými. Teprve kontext použití výrazu určí, jaká jeho část poslouží jako datový operand.

Ve většině případů je vyžadován přímý datový operand, zakódovatelný pomocí jednoho byte. V tomto případě asembler použije nižší byte šestnáctibitového výsledku. Z toho plyne, že tyto operandy mohou nabývat rozsahu -256..255 . Kladná čísla mají vyšší byte nulový, u záporných čísel obsahuje 0FFH. Použitím pouze spodního byte však informace o znaménku zanikne. Datový operand typu word nemá žádné takové omezení.

Přímé datové operandy nepodléhají kontrole příslušnosti k segmentu, což znamená, že ve výrazu, který reprezentuje přímý datový operand, lze použít libovolné symboly nezávisle na jejich příslušnosti k segmentům (třídě).

Následuje několik příkladů instrukcí s přímými datovými operandy.

```
MOV  R0,#10       ;Naplnění registru 0 konstantou
MOV  DPTR,#8000H  ;Naplnění datového ukazatele konstantou
ORL  A,#80H       ;Logický součet s obsahem akumulátoru
MOV  A,#PSW       ;Naplnění akumulátoru číslem 0D0H
```

Adresace dat

Pokud je číselný výraz použit ve významu adresa dat na čipu, není předcházen žádným speciálním symbolem. Výsledkem mohou být čísla v rozsahu -256..255. Adresuje se buď některá paměťová lokace na čipu z oblasti IDATA na adresách 0..127 nebo některý ze speciálních funkčních registrů. Pokud je ve výrazu použit symbol, musí patřit do třídy DATA, IDATA nebo NUMBER.

Přímé adresy v rozsahu 0..127 se vztahují na oblast dat na čipu, která je společná s oblastí IDATA (v celém rozsahu 0..127) a s oblastí třídy BIT (v rozsahu 32..47 neboli 20H..2FH). Všechny adresy v tomto rozsahu jsou použity, leží zde i všeobecné registry. Datová oblast s adresami 128..255 je nespojitá, pouze některé lokace jsou využity. Tab. 2-2 obsahuje přehled obsazení této poloviny oblasti DATA pro čipy 8051, 8052 a 8044.

Nižší nibble adresy	Vyšší nibble adresy: 8	9	A	B	C	D	E	F
F								
E								
D	TH1							
C	TH0							
B	TL1							
A	TL0							
9	TMOD	SBUF						
8	TCON	SCON	IE	IP				
7	PCON							
6								
5								
4								
3	DPH							
2	DPL							
1	SP							
0	P0	P1	P2	P3		PSW	ACC	B

Tab. 2-2 Speciální funkční registry čipu 8051

Nižší nibble adresy	Vyšší nibble adresy: 8	9	A	B	C	D	E	F
F								
E								
D	TH1				TH2			
C	TH0				TL2			
B	TL1				RCAP2L			
A	TL0				RCAP2H			
9	TMOD	SBUF						
8	TCON	SCON	IE	IP	T2CON			
7	PCON							
6								
5								
4								
3	DPH							
2	DPL							
1	SP							
0	P0	P1	P2	P3		PSW	ACC	B

Tab. 2-2 Speciální funkční registry čipu 8052

Nižší nibble adresy	Vyšší nibble adresy: 8	9	A	B	C	D	E	F
F		EBUF			DMA	FIFO3		
E		EINT			STAD	FIFO2		
D	TH1				RFL	FIFO1		
C	TH0				RBS	TBS		
B	TL1				RBL			
A	TL0				RCB	TCB		
9	TMOD				SMD	SIUST		
8	TCON		IE	IP	STS	NSNR		
7	PCON							
6								
5								
4								
3	DPH							
2	DPL							
1	SP							
0	P0	P1	P2	P3		PSW	ACC	B

Tab. 2-2 Speciální funkční registry čipu 8044

Adresace bitů

Bitová adresa představuje buď adresu bitu v bitově adresovatelném prostoru interní paměti RAM (na čipu) nebo adresu některého hardwarového bitu z oblasti speciálních funkčních registrů. Je nutno podotknout, že bitově adresovatelné jsou jen ty registry, které jsou na adresách x0H a x8H.

Existují dva způsoby, jak adresovat bitové operandy.

První způsob umožňuje zadat adresu bitu jako kombinaci adresy byte následované tečkou, za níž následuje offset bitu (celé číslo v rozsahu 0..7). Pro zadávání offsetu bitů je nutno použít takové výrazy, které se vyčíslí během činnosti asembleru (a jsou to tedy absolutní výrazy). Výraz pro vyjádření bázové adresy může být přemístitelný. Podmínkou při vyjádření bázové adresy pomocí symbolu patřícího do třídy DATA je, aby se při deklaraci segmentu, k němuž symbol patří, použilo atributu BITADDRESSABLE (viz kapitola 4).

Druhou možností je použít přímo adresu daného bitu. V tom případě lze použít použít pouze symbolů třídy BIT nebo NUMBER. Pokud se pro vyjádření bázové adresy segmentu třídy BIT použije symbolů, musí se jednat o výraz absolutní nebo jednoduše přemístitelný (viz kap. 4). Pro použití bitové adresy ve strojových instrukcích neplatí toto omezení.

Následuje několik příkladů vyjádření bitových adres:

```
SETB RS0     ;výběr sady registrů
CLR  FLAG    ;bitový symbol FLAG definovaný programátorem
MOV  C,ACC.7 ;bit 7 akumulátoru přemístit do příznaku přenosu
MOV  C,0E7H  ;tatáž instrukce s použitím absolutní adresy
```

Procesory řady MCS-51 mají celou řadu předdeklarovaných symbolů pro bity z oblasti speciálních funkčních registrů. Příloha 3 obsahuje jejich přehled pro procesory řady 8051, 8052 a 8044.

Adresace paměti programu

Adresa paměti programu je buď absolutní výraz s hodnotou v rozsahu 0..65535 (0..0FFFFH) nebo přemístitelný výraz třídy CODE. Existují tři typy instrukcí, vyžadující jako operand adresu v paměti programu. Jsou to relativní skoky, blokové skoky a volání ve 2kB bloku paměti programu a dlouhé skoky a volání.

Rozdílem při použití operandů je to, že zatímco operandy v případě dlouhých skoků a volání mohou nabývat neomezených hodnot, u ostatních dvou typů instrukcí je přípustný rozsah omezen. Kapitola 3 popisuje jednotlivé instrukce i přípustné operandy.

Relativní a podmíněné skoky

Tato skupina instrukcí umožňuje větvit program nezávisle na absolutní adrese. Cílová adresa se vypočte přidáním offsetu k adrese instrukce následující za instrukcí skoku. Offset je osmibitová hodnota, která omezuje rozsah cílových adres skoku na relativní vzdálenost v rozsahu -128..127 (dvojkově doplňkový kód).

Asembler kontroluje, zda výsledná hodnota výrazu vyčíslujícího cílovou adresu je třídy CODE nebo NUMBER a zda hodnota offsetu nepřesáhne povolené hranice.

Blokové skoky a volání

V případě blokového větvení programu se cílová adresa musí nacházet uvnitř téhož bloku programu, na němž je instrukce provádějící větvení. Uvnitř bloku se používá absolutní adresa o rozsahu jedenáct bitů, což omezuje délku bloku na 2kB. Blok je tudíž určen nejvyššími pěti bity okamžité hodnoty programového čítače, které nemohou být tímto typem instrukce ovlivněny.

Asembler kontroluje, zda výsledná hodnota výrazu vyčíslujícího cílovou adresu je třídy CODE nebo NUMBER a zda cílová adresa je uvnitř téhož bloku programu, na kterém se nachází instrukce větvení.

Dlouhé skoky a volání

Tato skupina instrukcí umožňuje větvení programu v celém paměťovém rozsahu. Cílová adresa je určena výrazem, jehož hodnota může být v rozsahu 0..65535 (0..0FFFFH).

Asembler kontroluje, zda výsledná hodnota výrazu vyčíslujícího cílovou adresu je třídy CODE nebo NUMBER.

Generické větvení programu

Asembler ASM-52 má dvě instrukce, které nepředstavují určitý operační kód. Jsou to instrukce JMP a CALL. Při použití těchto instrukcí asembler generuje operační kód podle toho, jaká je cílová adresa větvení.

Instrukce CALL se generuje jako ACALL, pokud cílová adresa volání leží uvnitř téhož bloku paměti programu a jeho vyjádření neobsahuje dopředné ani externí odkazy. V opačném případě se generuje LCALL.

Stejným pravidlem se řídí i instrukce JMP, navíc ale se může generovat i jako SJMP, pokud cílová adresa skoku leží v rozsahu -128..127 a její vyjádření neobsahuje dopředné ani externí odkazy.

Použití těchto generických instrukcí představuje efektivní způsob, jak zabránit velkým problémům při změně programu a přitom generovat částečně optimalizovaný kód. Je ale nutno poznamenat, že asembler generuje instrukce dlouhého skoku nebo volání i v případě, že cílová adresa je sice dosažitelná jedním z kratších způsobů, ale její vyjádření obsahuje dopředné odkazy.

Vyčíslení výrazů v průběhu překladu

Pojmem výraz v asembleru se rozumí kombinace číselných konstant, znakových řetězců, symbolů a operátorů, jíž lze vyčíslit jako šestnáctibitovou hodnotu. S vyjímkou některých příkazů může většina výrazů používat dopředné odkazy (na symboly, které nebyly zatím definovány) a všechny operátory asembleru.

Čísla

Asembler ASM-52 rozeznává čísla v soustavě šestnáctkové (přípona H), desítkové (bez přípony nebo s příponou D), osmičkové (přípona O nebo Q) a dvojkové (přípona B). Pokud číslo obsahuje příponu nebo znaky A,B,C,D,F, pak jsou povolena velká i malá písmena.

Reprezentace čísel

Z hlediska vnitřní reprezentace pracuje asembler se všemi čísly tak, jako by to byly 32-bitové veličiny. Záporná čísla jsou vyjádřena pomocí dvojkově-doplňkového kódu. Teprve přiřazením výrazu se rozsah výsledku omezí v závislosti na kontextu, v němž je výraz použit.

Použití dvojkově doplňkového kódu usnadňuje převod kladného čísla na záporné. Platí jednoduchý vztah:

```
-A = NOT(A)+1
```

Tato jednoduchost má ale za následek tu skutečnost, že nejde určit, zda hodnota představuje kladné či záporné číslo (-1=65535). Uživatel musí mít tuto skutečnost na zřeteli.

Znakové řetězce ve výrazech

Asembler ASM-52 umožňuje použití znakových řetězců ve výrazech. Všeobecně platí, že znakový řetězec o délce jeden či dva znaky může být použit jako číselná konstanta v libovolném kontextu. Vyjímku tvoří direktiva DB, která umožňuje použití znakových řetězců s 250-ti znaky.

Pokud má znakový řetězec délku jeden byte, je výsledná hodnota rovna ASCII reprezentaci tohoto znaku (vyšších 8 bitů má nulovou hodnotu). U dvouznakových řetězců je vyšších 8 bitů výsledku dáno ASCII reprezentací prvního znaku, nižších 8 bitů výsledku dáno ASCII reprezentací druhého znaku.

Znakový řetězec musí být ohraničený párem jednoduchých uvozovek ('). Pokud má řetězec obsahovat tento znak, lze jej nahradit dvěma těmito znaky bezprostředně za sebou.

Následující příklady dokumentují použití řetězců:

```
DB   1,'JEDEN',2,'DVA'  ;řetězce pro čísla 1 a 2
CJNE A,#'Z',$           ;čekání na znak 'Z' v akumulátoru
```

Zvláštnost představuje prázdný řetězec (dva znaky ''). Pokud je použit v kontextu číselná konstanta, vyčíslí se jako 0, jako argument direktivy DB však představuje prázdnou hodnotu, tj. nezpůsobí generování kódu.

Použití symbolů

Asembler poskytuje uživateli možnost práce s rozmanitými symboly. Jejich použitím programátor zvyšuje přehlednost zápisu zdrojového textu. Symboly mohou zastupovat registry, číselné konstanty, segmenty a adresy v různých oblastech paměti.

Každý použitý symbol má několik atributů:

- třída - příslušnost k určitému segmentu, může být BIT, CODE, DATA, IDATA, XDATA a NUMBER
- typ - registr, segment, adresa paměti, konstanta
- platnost - LOCAL, PUBLIC a EXTERNAL
- hodnota - jméno registru, bázová adresa segmentu, adresa v paměti nebo konstanta

Atribut typ klasifikuje význam symbolů:

- registr - symbolický název registru, nastavený pomocí direktivy EQU nebo SET
- segment - název přemístitelného segmentu paměti
- adresa - symbol reprezentuje adresu v paměti
- konstanta - symbol reprezentuje číselnou konstantu, použitelnou v libovolném kontextu

Atribut třída určuje příslušnost symbolu k určitému typu (oblasti) paměti. Tím je určeno, v jakém kontextu smějí být symboly použity. Vyjímku tvoří přídavný atribut BITADDRESSABLE pro třídu DATA, kde použití symbolu jako báze v konjunkci s bitovým offsetem vyjadřuje příslušnost k bitové oblasti paměti.

Atribut platnost určuje rozsah platnosti symbolu. Pokud je symbol místní (LOCAL), pak je použitelný pouze uvnitř jediného modulu, v němž byl definován. Symboly s atributem PUBLIC jsou použitelné i pro ostatní moduly, kde jsou naopak deklarovány pomocí atributu EXTERNAL. Pokud se deklaruje některý symbol s atributem EXTERNAL, nesmí se uvnitř modulu vyskytnout definice jiného symbolu se stejným jménem.

Atribut hodnota má následující významy:

- registr - hodnota je ASCII řetězec jména registru, který zastupuje
- segment - hodnota je bázová adresa přemístitelného segmentu, určí se teprve ve fázi spojování
- adresa - pro absolutní segmenty je to absolutní adresa paměťového místa, pro přemístitelné segmenty pak je to offset, jehož přičtením k bázi segmentu se získá výsledná hodnota
- konstanta - číselná hodnota nastavená direktivou EQU nebo SET

Jakmile programátor definuje symbol kdekoliv v programu, je použitelný kdekoliv v operandech stejně jako číselná konstanta, je ale nutné dodržovat konvence příslušnosti k segmentovým třídám. Bližší popis vytváření symbolů obsahuje kap. 4.

Asembler ASM-52 obsahuje některé předem deklarované symboly v závislosti na volbě MODE. Tabulka 2-2 obsahuje tyto předem deklarované symboly pro procesory řady 8051, 8052 a 8044. Další informace jsou obsaženy v příloze 3. Tyto symboly nelze předefinovat. Pokud není jejich existence žádoucí, lze je zrušit příkazem NOMODE (viz kap. 5).

Operátory ve výrazech

V asembleru ASM-52 existují čtyři skupiny operátorů: aritmetické, logické, relační a speciální. Všechny pracují s 32-bitovými hodnotami. Při konkrétním přiřazení pak jsou nejvyšší bity dle potřeby nahrazeny nulami. Operátory mohou být libovolně kombinovány nezávisle na jejich příslušnosti k určité skupině.

Aritmetické operátory

Tabulka 2-3 obsahuje aritmetické operátory.

Operátor	Význam
*	Násobení
/	Celočíselné dělení (ničí zbytek)
MOD	Zbytek po celočíselném dělení (ničí podíl)
+	Součet nebo unární plus
-	Rozdíl nebo unární mínus

Tab. 2-3 Aritmetické operátory asembleru

Příklady použití:

```
2+3
-1
7 MOD 3
4*5-2
```

Logické operátory

Tabulka 2-4 obsahuje logické operátory.

Operátor		Význam
OR	|	Logický součet
AND	&	Logický součin
XOR	^	Logický exkluzivní součet (modulo 2)
NOT	!	Logická negace

Tab. 2-3 Logické operátory asembleru

Příklady použití:

```
2 OR 3
!1
7 AND 3
4&5 XOR 2
```

Relační operátory

Tabulka 2-5 obsahuje relační operátory.

Operátor	Význam
EQ =	Rovnost
NE <>	Nerovnost
LT <	Menší než
LE <=	Menší nebo rovno
GT >	Větší než
GE >=	Větší nebo rovno

Tab. 2-5 Relační operátory asembleru

Relační operátory se liší od ostatních tím, že produkují pouze dvě výsledné hodnoty: false (0) a true (0FFFFFFFFH).

Příklady použití:
16>=14
16 GE 14

Speciální operátory asembleru

Tabulka 2-6 obsahuje speciální operátory.

Operátor	Význam
SHR	Posun bitů doprava
SHL	Posun bitů doleva
LOW	Výběr nižšího byte operandu 16 bit
HIGH	Výběr vyššího byte operandu 16 bit
()	Přednostní vyčíslení výrazu v závorkách

Tab. 2-6 Speciální operátory asembleru

Příklady použití:
LOW(00000111B SHL 4)
(HIGH 1234H) SHR 8

Pořadí vyhodnocování operátorů

Při vyhodnocování výrazů je důležitá jejich přednost. Operace s vyšší prioritou mají přednost před operacemi s nižší prioritou. Například výraz 2*3+5 dá výsledek 11, protože operace násobení má přednost před sčítáním (neboli násobení má vyšší prioritu). Následující seznam operátorů je seřazen tak, že první řádek je tvořen operátory s nejvyšší prioritou, další s nižší až nakonec poslední řádek obsahuje operátory s nejnižší prioritou.

- () výrazy uzavřené do závorek
- *, /, MOD, SHR, SHL
- +, - unární i binární
- EQ, NE, GE, GT, LE, LT, =, <>, >=, >, <=, <
- NOT, !
- AND, &
- OR, XOR, |, ^

Pokud je ve výrazu více operátorů se stejnou prioritou, jsou vyhodnocovány postupně zleva doprava.

Použití segmentů ve výrazech

Většina výrazů vyhodnocených asemblerem nemá žádnou příslušnost k některé třídě segmentů, resp. jsou třídy NUMBER. Jsou ale některé výrazy, které se vyhodnotí tak, že patří do určité segmentové třídy. Toto vyhodnocení se řídí následujícími pravidly:

1/ Výsledek unární operace (+, -, LOW, HIGH, NOT, !) má tutéž příslušnost k segmentu jako operand

2/ Výsledkem jakékoliv binární operaci s vyjímkou sčítání a odčítání je hodnota bez příslušnosti k segmentu (třída NUMBER)

3/ Pro binární sčítání a odčítání platí, že pokud má pouze jeden z operandů nějakou příslušnost k segmentu, má výsledek tutéž příslušnost; v opačném případě je třídy NUMBER

Máme-li např. výraz, zapsaný pomocí příslušnosti k segmentům jako DATA + (CODE1 - CODE2), je výsledkem hodnota s třídou DATA, neboť výsledkem výrazu CODE1-CODE2 je operand s třídou NUMBER.

Vyčíslení přemístitelných výrazů

Přemístitelný výraz je takový výraz, který obsahuje odkazy na přemístitelné symboly téhož modulu, popř. externí odkazy do jiného modulu. Takové výrazy nemohou být asemblerem vyčísleny, tuto činnost dokončí teprve spojovací a umísťovací program (linker).

Z toho plyne, že přemístitelný výraz může obsahovat pouze jediný odkaz na přemístitelný symbol. Je třeba si ovšem uvědomit, že relační operace mezi dvěma symboly se stejnou segmentovou třídou, případně jejich odečtení, dá jako výsledek absolutní číslo. Takové výrazy nejsou tedy přemístitelné, ale absolutní. Přemístitelný symbol může být modifikován přičtením či odečtením absolutní hodnoty. Následující schematické zápisy představují platné přemístitelné výrazy:

```
přemístitelný_symbol + absolutní_výraz
přemístitelný_symbol - absolutní_výraz
absolutní_výraz + přemístitelný_symbol
```

Jednoduše přemístitelné výrazy

V jednoduše přemístitelných výrazech může symbol představovat pouze adresu v přemístitelném paměťovém segmentu. Externí odkazy a odkazy na jména segmentů nejsou povolené.

Jednoduše přemístitelné výrazy jsou použitelné v následujících kontextech:

1/ operand direktivy ORG

2/ operand direktiv definujících symbol (EQU, SET, BIT, CODE, DATA, IDATA, XDATA)

3/ operand instrukce nebo direktiv DB a DW

Všeobecné přemístitelné výrazy

Všeobecně přemístitelný výraz nemá žádné z výše uvedených omezení, může používat externích odkazů i odkazů na báze segmentů. Takové výrazy lze však použít pouze jako operandy strojových instrukcí nebo jako operandy direktiv DB a DW. Všeobecně přemístitelné výrazy smějí být navíc předcházeny operátory LOW a HIGH.

Kapitola 3 - INSTRUKČNÍ SOUBOR

Úvod

Tato kapitola popisuje instrukční soubor asembleru ASM-52. Je psána jako referenční příručka. Instrukce jsou psány v abecedním pořádku. Pro jejich popis je použit stejný základní formát uspořádání.

Každá instrukce začíná na nové stránce. Záhlaví každé instrukce obsahuje stručný popis její činnosti, pak následuje několik hesel, obsahujících údaje o operandech, strojovém kódu a činnosti instrukce v souvislosti s registry procesoru a nakonec je uveden podrobný popis s příkladem použití. Použité symboly jsou přehledně uvedeny v tab. 3-1 níže.

U instrukcí je ještě uvedeno, zda se jich týkají nějaké poznámky. Tyto poznámky jsou uvedeny na konci kapitoly. Význam jednotlivých hesel v popisu instrukcí je:

Mnemonika: Vyjádření instrukce v jazyce symbolických adres. Je uvedeno velkými písmeny, ale asembler nedělá rozdíl mezi velkými a malými písmeny.

Operandy: Typy a přípustný rozsah operandů

Zápis: Tvar zápisu instrukce a pořadí operandů v textu

Strojový kód: Bitový obrazec strojového kódu instrukce tak, jak je uložen v paměti počítače

Činnost: Symbolický zápis činnosti instrukce

Délka: Počet byte, která zabírá instrukce v paměti programu

Doba trvání: Délka provádění instrukce, vyjádřeno ve strojových cyklech

Ovlivňuje: Znázornění ovlivňovaných příznaků ve stavovém slově programu (PSW)

Popis: Popis činnosti a použití instrukce

Příklad: Znázornění použití instrukce ve zdrojovém programu, jejího uložení v paměti ve formě bitového obrazce a obsah všech použitých operandů před a po provedení instrukce

Poznámky: Výčet poznámek, vztahujících se k popisované instrukci. Soupis těchto poznámek je na konci této kapitoly

Symbol	Význam
A	Akumulátor
AB	Dvojice registrů pro násobení a dělení
adresa bitu	Adresa bitu u obvodů 8051
adresa bloku	11-bitová adresa uvnitř 2KB bloku
B	Registr pro násobení a dělení
rel.offset	8-bitové relativní posunutí ve tvaru čísla ve dvojkovém doplňkovém kódu
C	Příznak přenosu
<adr>	Adresa v paměti programu
data	Přímý datový operand (konstanta)
datová adresa	8-bitová adresa v paměti RAM na čipu
DPTR	Ukazatel dat
PC	Programový čítač (čítač instrukcí)
Rr	Registr, r = 0 - 7 (celé číslo), při nepřímém adresování r = 0 nebo 1
SP	Ukazatel zásobníku
HIGH	Slabika vyšších řádů
LOW	Slabika nižších řádů
.n	Bit číslo n, n = 0 - 7 (celé číslo)
AND	Logický součin
NOT	Negace
OR	Logický součet
XOR	Součet modulo 2 (exkluzívní součet)
+	Plus
-	Mínus
*	Násobení
/	Dělení
=	Rovná se
(X)	Obsah prvku X
((X))	Obsah paměťového místa adresovaného obsahem prvku X
<	Menší než
>	Větší než
<>	Nerovná se
<-	Přesun hodnoty ne směru šipky
0	Číslice nula
#	Označení přímých dat v symbolické instrukci
@	Označení nepřímé adresy v symbolické instrukci

Tab. 3-1 Symboly použité při popisu instrukcí

ACALL

Absolutní volání podprogramu uvnitř 2kB bloku

Mnemonika: ACALL

Operandy: adresa v paměti programu

Zápis: ACALL <adr>

Strojový kód:

a a a 1 0 0 0 1	a a a a a a a a
7 0	7 0

Činnost:
(PC) <- (PC)+2
(SP) <- (SP)+1
((SP)) <- LOW(PC)
(SP) <- (SP)+1
((SP)) <- HIGH(PC)
(PC)0-10 <- adresa bloku

Délka: 2 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

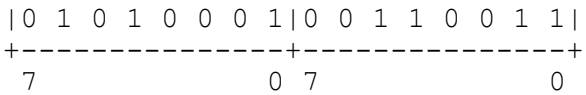
PSW

Popis: Tato instrukce uschová obsah programového čítače zvětšený o dvě (návratovou adresu) do zásobníku. Nižší byte programového čítače je vždy umístěn v zásobníku první. Pak nahradí jedenáct nejméně významných bitů programového čítače jedenácti adresními bity, zakódovanými v instrukci. Cílová adresa volání musí ležet uvnitř 2kB bloku paměti programu. Nejvýznamnější tři bity operačního kódu instrukce nahradí bity 10, 9 a 8 programového čítače, nižší byte programového čítače je nahrazen druhým byte instrukce.

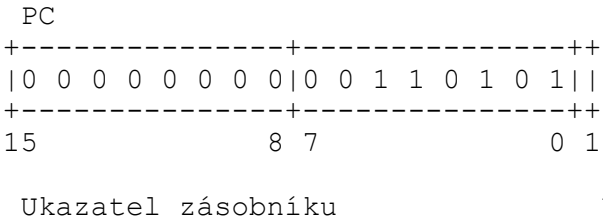
Příklad:

```
ORG 35H
ACALL SORT      ;volání podprogramu SORT
                ;(uložený v bloku na adrese 233H)
     .
     .
ORG 233H
SORT:
PUSH ACC        ;uloží obsah akumulátoru
     .
     .
RET             ;návrat z podprogramu
```

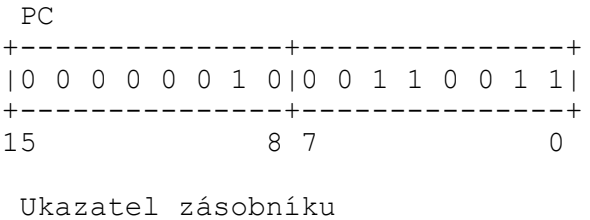
Kód instrukce:



Před provedením



Po provedení



Poznámky : 2,3

ADD

Součet obsahu akumulátoru a dat

Mnemonika: ADD

Operandy: A akumulátor
data -256 <= data <= +255

Zápis: ADD A, #data

Strojový kód:

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

Činnost: (A) <- (A) + data

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x	x				x		x

PSW

Popis: Tato instrukce provede součet 8-bitových dat a obsahu akumulátoru. Výsledek uloží do akumulátoru.

Příklad: ADD A,#32H ;přičte 32H k obsahu akumulátoru

Kód instrukce:

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

Před provedením

Akumulátor

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

Po provedení

Akumulátor

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

Poznámky: 4, 5, 6, 7

ADD

Součet obsahu akumulátoru a obsahu nepřímé adresy

Mnemonika: ADD

Operandy:
A	akumulátor
Rr	registr r ={ 1, 2}

Zápis: ADD A,@Rr

Strojový kód:

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

Činnost: (A) <- (A) + ((Rr))

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x	x				x		x

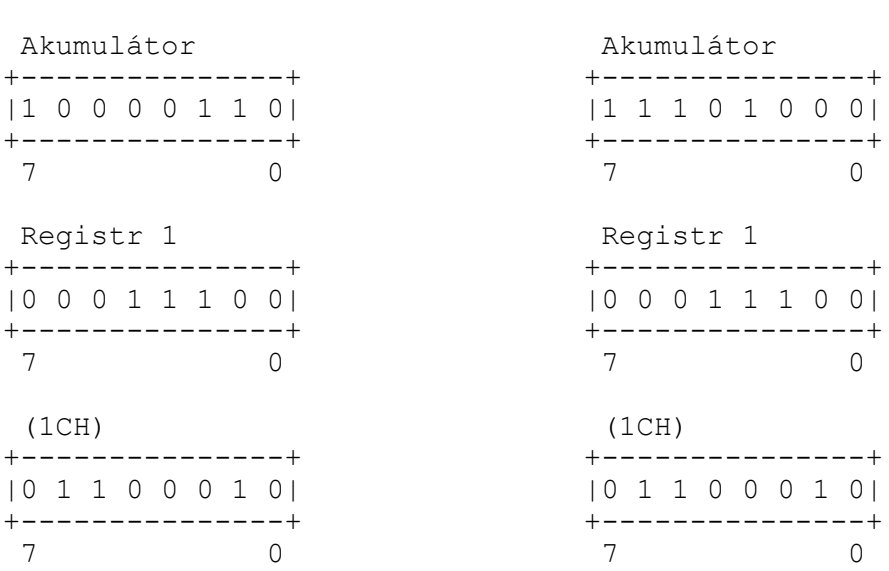
PSW

Popis: Tato instrukce provede součet obsahu akumulátoru a obsahu paměťového místa na adrese určené registrem r. Výsledek uloží do akumulátoru.

Příklad:
```
ADD A,@R1        ;součet    akumulátoru   a  obsahu
                 ;paměťového místa na adrese určené
                 ;registrem R1
```

Kód instrukce:

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



Poznámky: 5, 6, 7, 15

ADD

Součet obsahu akumulátoru a obsahu registru

Mnemonika: ADD

Operandy: A akumulátor
Rr registr 0 <= r <= 7

Zápis: ADD A,Rr

Strojový kód:

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

Činnost: (A) <- (A) + (Rr)

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x	x				x		x

PSW

Popis: Tato instrukce provede součet obsahu akumulátoru a obsahu registru r. Výsledek uloží do akumulátoru.

Příklady: ADD A,R6 ;součet obsahu R6 a akumulátoru

Kód instrukce:

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

Před provedením

Akumulátor

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

Registr 6

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

Po provedení

Akumulátor

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

Registr 6

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

Poznámky: 5, 6, 7

ADD

Součet obsahu akumulátoru a obsahu datové adresy

Mnemonika: ADD

Operandy: A akumulátor
datová adresa 0 <= datová adresa <= 255

Zápis: ADD A,datová adresa

Strojový kód:

0 0 1 0 0 1 0 1	datová adresa
7 0	7 0

Činnost: (A) <- (A) + (datová adresa)

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x	x				x		x

PSW

Popis: Tato instrukce provede součet obsahu akumulátoru a obsahu paměťového místa určeného datovou adresou. Výsledek uloží do akumulátoru.

Příklad:
```
ADD   A,32H          ;součet akumulátoru a dat na
                     ;adrese 32H
```
Kód instrukce:

0 0 1 0 0 1 0 1	0 0 1 1 0 0 1 0
7 0	7 0

Před provedením

Akumulátor

0 0 1 0 0 1 1 0
7 0

(32H)

0 1 0 1 0 0 1 1
7 0

Po provedení

Akumulátor

0 1 1 1 1 0 0 1
7 0

(32H)

0 1 0 1 0 0 1 1
7 0

Poznámky: 5, 6, 7, 8

ADDC

Součet obsahu akumulátoru, příznaku přenosu a dat

Mnemonika: ADDC

Operandy: A akumulátor
data -256 <= data <= +255

Zápis: ADDC A,#data

Strojový kód:

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

Činnost: (A) <- (A) + (C) + data

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x	x				x		x

PSW

Popis: Tato instrukce provede součet obsahu akumulátoru, hodnoty příznaku přenosu (0 nebo 1) a 8-bitových dat. Příznak přenosu se znovu nastaví a výsledek součtu se uloží do akumulátoru. Příznak přenosu a akumulátor jsou výsledkem součtu všech tří hodnot.

Příklad:
```
ADDC   A,#0AFH     ;součet akumulátoru, hodnoty AFH
                   ;a příznaku přenosu
```

Kód instrukce:

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

Před provedením

Akumulátor

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

Příznak přenosu

1

Po provedení

Akumulátor

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

Příznak přenosu

1

Poznámky: 4, 5, 6, 7

ADDC

Součet obsahu akumulátoru, příznaku přenosu a obsahu nepřímé adresy

Mnemonika: ADDC

Operandy:
	A	akumulátor
	Rr	registr r ={ 1, 2}

Zápis: ADDC A,@Rr

Strojový kód:

0	0	1	1	0	1	1	r
7							0

Činnost: (A) <- (A) + (C) +((Rr))

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x	x				x		x
			PSW				

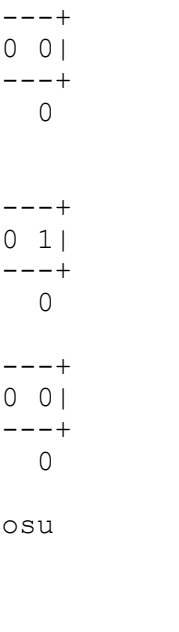
Popis: Tato instrukce provede součet obsahu akumulátoru, hodnoty příznaku přenosu (0 nebo 1) a obsahu paměťového místa na adrese určené registrem r. Příznak přenosu se znovu nastaví a výsledek součtu se uloží do akumulátoru. Příznak přenosu a akumulátor jsou výsledkem součtu všech tří hodnot.

Příklady:
```
ADDC   A,@R1        ;součet akumulátoru, příznaku
                    ;přenosu a obsahu paměti na
                    ;adrese určené registrem R1
```

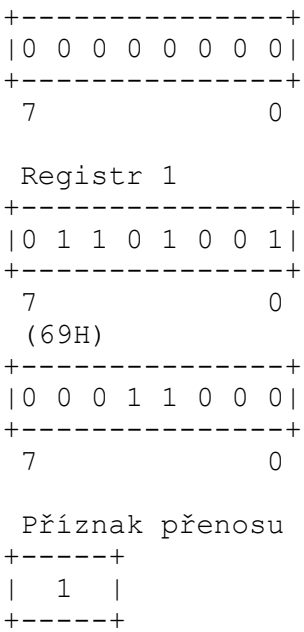
Kód instrukce:

0	0	1	1	0	1	1	1
7							0

Před provedením



Po provedení



Poznámky: 5, 6, 7, 15

ADDC

Součet obsahu akumulátoru, příznaku přenosu a obsahu registru

Mnemonika: ADDC

Operandy:
A	akumulátor	
Rr	registr	0 <= r <= 7

Zápis: ADDC A,Rr

Strojový kód:

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

Činnost: (A) <- (A) + (C) + (Rr)

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x	x				x		x

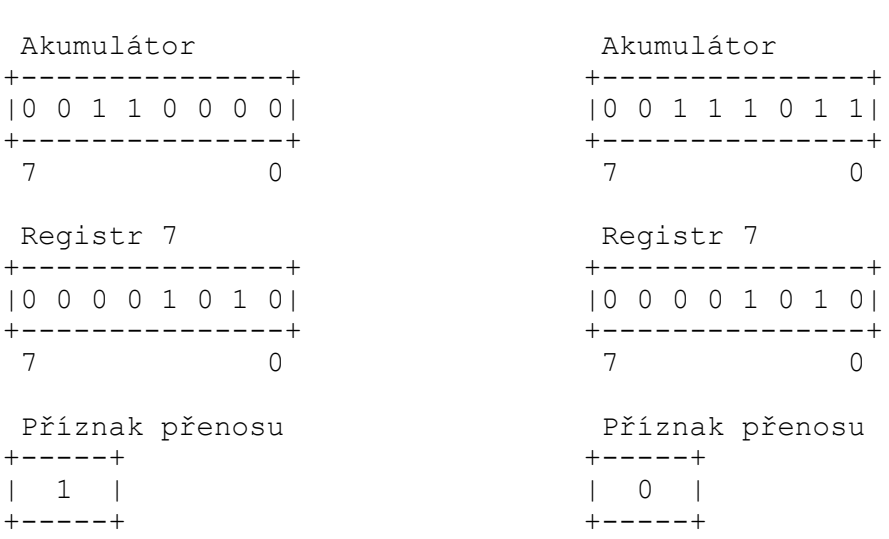
PSW

Popis: Tato instrukce provede součet obsahu akumulátoru, hodnoty příznaku přenosu (0 nebo 1) a obsahu registru r. Příznak přenosu se znovu nastaví a výsledek součtu se uloží do akumulátoru. Příznak přenosu a akumulátor jsou výsledkem součtu všech tří hodnot.

Příklad:
```
ADDC   A,R7          ;součet akumulátoru, příznaku
                     ;přenosu a obsahu registru 7
```

Kód instrukce:

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



Poznámky: 5, 6, 7

ADDC

Součet obsahu akumulátoru, příznaku přenosu obsahu datové adresy

Mnemonika: ADDC

Operandy:
A	akumulátor
datová adresa	0 <= datová adresa <= 255

Zápis: ADDC A,datová adresa

Strojový kód:

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	datová adresa							

Činnost: (A) <- (A) + (C) + (datová adresa)

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x	x				x		x

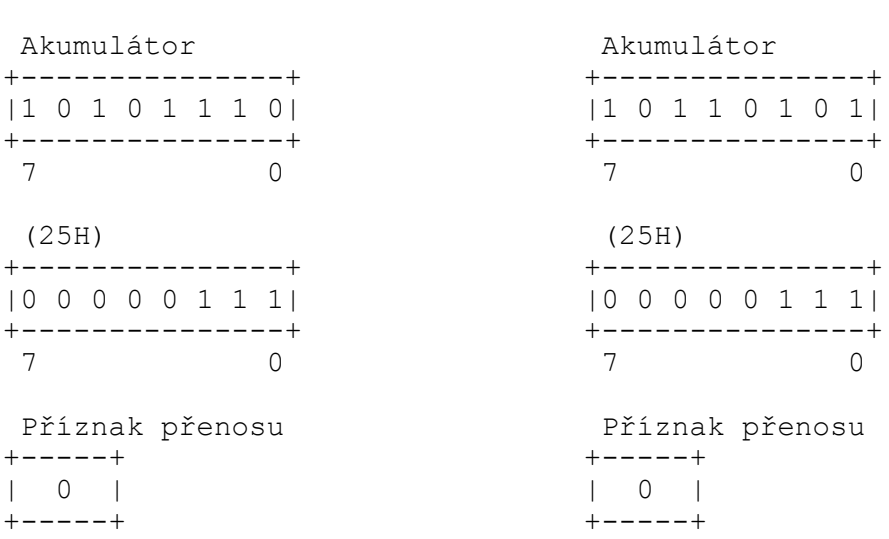
PSW

Popis: Tato instrukce provede součet obsahu akumulátoru, hodnoty příznaku přenosu (0 nebo 1) a obsahu paměťového místa určeného datovou adresou. Příznak přenosu se znovu nastaví a výsledek součtu se uloží do akumulátoru. Příznak přenosu a akumulátor jsou výsledkem součtu všech tří hodnot.

Příklady:
```
ADDC   A,25H        ;součet akumulátoru, příznaku
                    ;přenosu a obsahu paměťového
                    ;místa na adrese 25H
```

Kód instrukce:

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



Poznámky: 5, 6, 7, 8

AJMP

Absolutní skok uvnitř 2kB bloku

Mnemonika: AJMP

Operandy: adresa v paměti programu

Zápis: AJMP <adr>

Strojový kód:

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

Činnost:
(PC) <- (PC) + 2
(PC)0-10 <- adresa bloku

Délka: 2 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce inkrementuje obsah programového čítače a potom nahradí jedenáct nižších bitů programového čítače jedenácti bity kódové adresy. Cílová adresa bude v témže 2 kB bloku paměti, ve které je umístěna instrukce následující po AJMP.

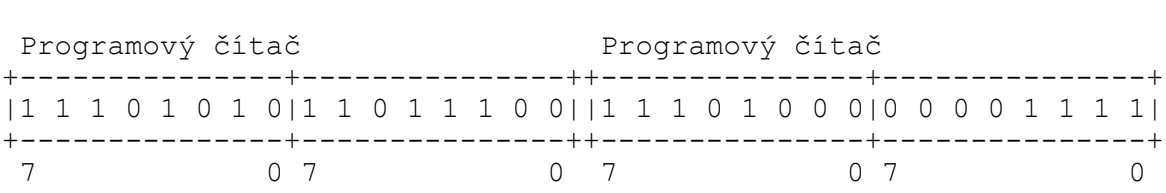
Jedenáctibitová adresa uvnitř 2 kB bloku je složena ze tří nejvyšších bitů operačního znaku a z osmi bitů druhé slabiky strojového kódu instrukce AJMP.

Příklad:

```
      ORG 0E80FH
TOPP: MOV   A,R3
         .
         .
      ORG 0EADCH
      AJMP TOPP     ;skok zpět na návěští TOPP  na
                    ;adresu 0E80FH
```

Kód instrukce:

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



Poznámky: žádné

ANL

Logický součin dat s obsahem akumulátoru

Mnemonika: ANL

Operandy: A akumulátor
data -256 <= data <= +255

Zápis: ANL A,#data

Strojový kód:

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

Činnost: (A) <- (A) AND data

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce provede logický součin hodnoty 8-bitových dat s obsahem akumulátoru. Bit n výsledku bude nastaven na hodnotu 1, bude-li bit n obou operandů roven 1. Jinak je bit n výsledku vynulován. Výsledek je umístěn v akumulátoru.

Příklad:
```
ANL A,#00001000B    ;maska vyloučí všechny bity
                    ;kromě bitu 3
```

Kód instrukce:

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

Před provedením

Akumulátor

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

Po provedení

Akumulátor

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

Poznámky: 4, 5

ANL

Logický součin obsahu nepřímé adresy s obsahem akumulátoru

Mnemonika: ANL

Operandy:
A	akumulátor
Rr	registr r ={ 1, 2}

Zápis: ANL A,@Rr

Strojový kód:

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

Činnost: (A) <- (A) AND ((Rr))

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

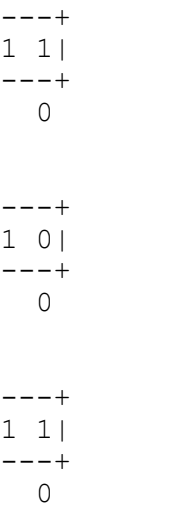
Popis: Tato instrukce provede logický součin obsahu paměťového místa o adrese určené registrem r s obsahem akumulátoru. Bit n výsledku bude nastaven na hodnotu 1, bude-li bit n obou operandů roven 1. Jinak je bit n výsledku vynulován. Výsledek je umístěn v akumulátoru.

Příklad:
```
ANL   A,@R0        ;logický součin obsahu nepřímé
                    ;adresa obsahu akumulátoru
```

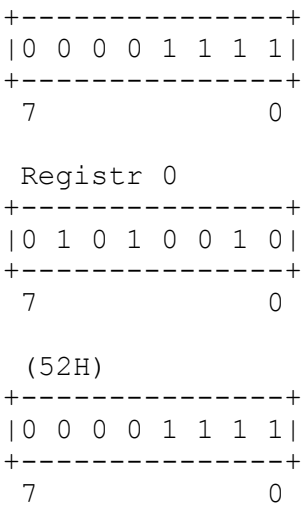
Kód instrukce:

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

Před provedením



Po provedení



Poznámky: 5, 15

ANL

Logický součin obsahu registru s obsahem akumulátoru

Mnemonika: ANL

Operandy: A akumulátor
Rr registr 0 <= r <= 7

Zápis: ANL A,Rr

Strojový kód:

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

Činnost: (A) <- (A) AND (Rr)

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce provede logický součin obsahu registru r s obsahem akumulátoru. Bit n výsledku bude nastaven na hodnotu 1, bude-li bit n obou operandů roven 1. Jinak je bit n výsledku vynulován. Výsledek je umístěn v akumulátoru.

Příklad:

```
MOV    R4,10000000B  ;přesun masky do registru 4
ANL    A,R4           ;logický součin R4 a akumulá-
                      ;toru
```

Kód instrukce:

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

Před provedením

Akumulátor

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

Registr 4

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

Po provedení

Akumulátor

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

Registr 4

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

Poznámky: 5

ANL

Logický součin obsahu datové adresy s obsahem akumulátoru

Mnemonika: ANL

Operandy: A akumulátor
datová adresa 0 <= datová adresa <= 255

Zápis: ANL A,datová adresa

Strojový kód:

0 1 0 1 0 1 0 1	datová adresa
7 0	7 0

Činnost: (A) <- (A) AND (datová adresa)

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce provede logický součin obsahu datové adresy s obsahem akumulátoru. Bit n výsledku bude nastaven na hodnotu 1, bude-li bit n obou operandů roven 1. Jinak je bit n výsledku vynulován. Výsledek je umístěn v akumulátoru.

```
Příklad:      ANL A,37H           ;logický součin obsahu na adrese
                                  ;37H a akumulátoru
```

Kód instrukce:

0 1 0 1 0 1 0 1	0 0 1 1 0 1 1 1
7 0	7 0

Před provedením

Akumulátor

0 1 1 1 0 1 1 1
7 0

(37H)

1 1 1 1 0 0 0 0
7 0

Po provedení

Akumulátor

0 1 1 1 0 0 0 0
7 0

(37H)

1 1 1 1 0 0 0 0
7 0

Poznámky: 5, 8

ANL

Logický součin obsahu bitové adresy s příznakem přenosu

Mnemonika: ANL

Operandy: C příznak přenosu
adresa bitu 0 <= adresa bitu <= 255

Zápis: ANL C,adresa bitu

Strojový kód:

1 0 0 0 0 0 1 0	adresa bitu
7	0 7 0

Činnost: (C) <- (C) AND (adresa bitu)

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x							

PSW

Popis: Tato instrukce provede logický součin hodnoty příznaku přenosu s hodnotou bitu určeného adresou. Mají-li oba bity logickou hodnotu 1, je výsledkem logická 1. Jinak je výsledkem logická 0. Výsledek je umístěn do bitu příznaku přenosu.

Příklad:
```
ANL C,37.3          ;logický součin bitu 3 ve slabice
                    ;na adrese 37H a příznaku přenosu
```

Kód instrukce:

1 0 0 0 0 0 1 0	0 0 1 0 1 0 1 1
7 0	7 0

Před provedením

Příznak přenosu

1

(37)

0 0 1 0 1 1 1 0
7 0

Po provedení

Příznak přenosu

1

(37)

0 0 1 0 1 1 1 0
7 0

Poznámky: žádné

ANL

Logický součin negace obsahu bitové adresy s příznakem přenosu

Mnemonika: ANL

Operandy: C příznak přenosu
adresa bitu 0 <= adresa bitu <= 255

Zápis: ANL C,/adresa bitu

Strojový kód:

1 0 1 1 0 0 0 0	adresa bitu
7 0	7 0

Činnost: (C) <- (C) AND NOT (adresa bitu)

Délka: 2 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x							

PSW

Popis: Tato instrukce provede logický součin hodnoty příznaku přenosu s negací hodnoty bitu určeného adresou bitu. Výsledek bude mít logickou hodnotu 1, bude-li mít příznak přenosu logickou hodnotu 1 a adresovaný bit logickou hodnotu 0. Hodnota bitu adresovaného bitovou adresou se nemění. Výsledek je umístěn v příznaku přenosu.

Příklad:
```
ANL C,/40.5        ;logický součin negace bitu 5 ve
                   ;slabice na adrese 40H a přízna-
                   ;ku přenosu
```

Kód instrukce:

1 0 1 1 0 0 0 0	0 1 0 0 0 1 0 1
7 0	7 0

Před provedením

Příznak přenosu

1

(40)

0 1 0 1 1 0 0 0
7 0

Po provedení

Příznak přenosu

1

(40)

0 1 0 1 1 0 0 0
7 0

Poznámky: žádné

ANL

Logický součin dat s obsahem datové adresy

Mnemonika: ANL

Operandy: datová adresa 0 <= datová adresa <= 255
data -256 <= data <= +255

Zápis: ANL datová adresa,#data

Strojový kód:

0 1 0 1 0 0 1 1	datová adresa	data
7 0	7 0	7 0

Činnost: (datová adresa) <- (datová adresa) AND data

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce provede logický součin osmibitových dat s obsahem určené datové adresy. Bit n výsledku bude nastaven do logické hodnoty 1, bude-li bit n obou operandů mít hodnotu 1. Jinak je bit n výsledku nulován. Výsledek se umístí do paměti na určenou datovou adresu.

Příklad:
```
MOV   57H,PSW      ;přesun PSW na adresu 57H
ANL   57H,#01H     ;maska pro zachování příznaku P
```

Kód instrukce:

0 1 0 1 0 0 1 1	0 1 0 1 0 1 1 1	0 0 0 0 0 0 0 1
7 0	7 0	7 0

Před provedením

(57H)

0 1 1 1 0 1 1 1
7 0

Po provedení

(57H)

0 0 0 0 0 0 0 1
7 0

Poznámky: 4,7

ANL

Logický součin obsahu datové adresy s obsahem akumulátoru

Mnemonika: ANL

Operandy: datová adresa 0 <= datová adresa <= 255
A akumulátor

Zápis: ANL datová adresa,A

Strojový kód:

0 1 0 1 0 0 1 0	datová adresa
7 0	7 0

Činnost: (datová adresa) <- (datová adresa) AND A

Délka: 2 byte

Doba trvání: 1 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce provede logický součin obsahu určené datové adresy s obsahem akumulátoru. Bit n výsledku bude nastaven do logické hodnoty 1, bude-li bit n obou operandů mít hodnotu 1. Jinak je bit n výsledku nulován. Výsledek se umístí do paměti na určenou datovou adresu.

Příklad:
```
MOV   A,10000001B ;uložení masky do akumulátoru
ANL   10H,A       ;maska pro zachování bitů 0 a 7
```

Kód instrukce:

0 1 0 1 0 0 1 0	0 0 0 1 0 0 0 0
7 0	7 0

Před provedením

Akumulátor

1 0 0 0 0 0 0 1
7 0

(10H)

0 0 1 1 0 0 0 1
7 0

Po provedení

Akumulátor

1 0 0 0 0 0 0 1
7 0

(10H)

0 0 0 0 0 0 0 1
7 0

Poznámky: 9

CALL

Generovaná instrukce volání

Mnemonika: CALL

Operandy: adresa v paměti programu

Zápis: CALL <adr>

Strojový kód: Přeloží se jako ACALL nebo LCALL podle následujícího popisu

Činnost: viz buď ACALL nebo LCALL

Délka: 2 nebo 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce se přeloží jako LCALL nebo ACALL. Instrukce bude přeložena jako ACALL, pokud cílová adresa volání leží uvnitř 2kB bloku a její vyjádření neobsahuje žádné dopředné odkazy. V opačném případě se přeloží jako LCALL, ačkoliv může být cílová adresa dosažitelná i pomocí instrukce ACALL. Pro popis činnosti viz kapitoly LCALL a ACALL.

Příklad:

```
ORG 23H
SORT:
PUSH ACC            ;uloží obsah akumulátoru
     .
     .
RET                 ;návrat z podprogramu
     .
     .
     .
ORG 35H
CALL  SORT          ;volání podprogramu SORT
                    ;přeloží se jako ACALL
```

Kód instrukce: viz ACALL nebo LCALL

Poznámky : 1,2,3

CJNE

Porovnání obsahu nepřímé adresy a datového operandu,
provede skok, pokud se nerovnají

Mnemonika: CJNE

Operandy:
```
Rr              registr r ={ 1, 2}
data            -256 <= data <= +255
adresa v paměti programu
```

Zápis: CJNE @Rr,#data,<adr>

Strojový kód:

1 0 1 1 0 1 1 r	d a t a	rel. offset
7 0	7 0	7 0

Činnost:
```
(PC) <- (PC) + 3
IF ((Rr))<>data THEN (PC) <- (PC) + rel. offset
IF ((Rr)) <data THEN (C) <- 1
                ELSE (C) <- 0
```

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

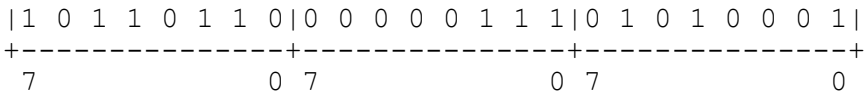
C	AC	F0	RS1	RS0	OV		P
x							

PSW

Popis: Tato instrukce porovnává hodnotu datového operandu s obsahem nepřímé adresy, obsažené v registru Rr. Pokud jsou tyto hodnoty různé, provede se skok na uvedenou adresu v paměti programu. Pokud jsou hodnoty stejné, pokračuje se dále ve vykonávání programu.
Pokud je hodnota datového operandu obsaženého v instrukci větší než hodnota, obsažená na nepřímé adrese v paměti dat, nastaví se příznak přenosu na hodnotu "1", v opačném případě se příznak přenosu nastaví na "0".
Programový čítač se zvětší o tři. Pokud se operandy nerovnají, přičte se ještě hodnota relativního offsetu.

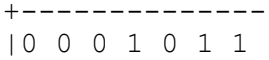
Příklad:
```
ORG 110H
CJNE @R0,#7H,NEQ  ;skok, když obsah  nepřímé adresy
                  ;není 7
ORG 164H
NEQ:
MOV  F0,C         ;uschování příznaku přenosu
```

Kód instrukce:



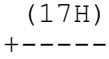
Před provedením

Registr 0

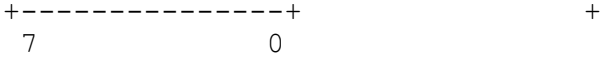


(17H)

Příznak přenosu

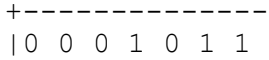


Programový čítač



Po provedení

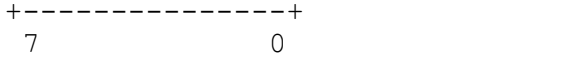
Registr 0



(17H)

Příznak přenosu

Programový čítač



Poznámky: 4,10,11,12,15

CJNE

Porovnání datového operandu s obsahem akumulátoru,
provede skok, pokud se nerovnají

Mnemonika: CJNE

Operandy:
A	akumulátor
data	-256 <= data <= +255
adresa v paměti programu	

Zápis: CJNE A,#data,<adr>

Strojový kód:

1 0 1 1 0 1 0 0	d a t a	rel. offset
7 0	7 0	7 0

Činnost:
```
(PC) <- (PC) + 3
IF (A)<>data THEN (PC) <- (PC) + rel. offset
IF (A) <data THEN (C) <- 1
             ELSE (C) <- 0
```

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

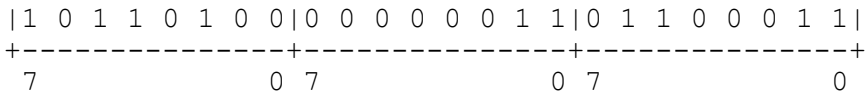
C	AC	F0	RS1	RS0	OV		P
x							

PSW

Popis: Tato instrukce porovnává hodnotu datového operandu s obsahem akumulátoru. Pokud jsou tyto hodnoty různé, provede se skok na uvedenou adresu v paměti programu. Pokud jsou hodnoty stejné, pokračuje se dále ve vykonávání programu.
Pokud je hodnota datového operandu obsaženého v instrukci větší než hodnota, obsažená v akumulátoru, nastaví se příznak přenosu na hodnotu "1", v opačném případě se příznak přenosu nastaví na "0".
Programový čítač se zvětší o tři. Pokud se operandy nerovnají, přičte se ještě hodnota relativního offsetu.

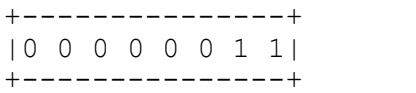
Příklad:
```
ORG 11H
CJNE A,#3H,DCR    ;skok, když obsah ACC<>3

ORG 77H
DCR:
DEC  A
```

Kód instrukce:

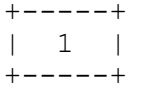


Před provedením

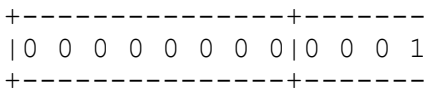
Akumulátor



Příznak přenosu

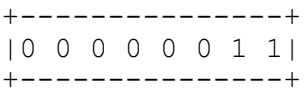


Programový čítač

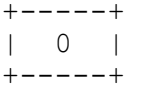


Po provedení

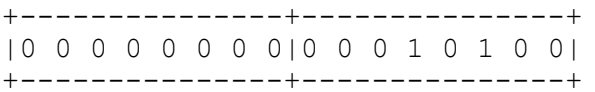
Akumulátor



Příznak přenosu



Programový čítač



Poznámky: 4,10,11,12

CJNE

Porovnání obsahu obsahu datové adresy a akumulátoru,
provede skok, pokud se nerovnají

Mnemonika: CJNE

Operandy:
A	akumulátor
datová adresa	0 <= datová adresa <= 255
adresa v paměti programu	

Zápis: CJNE A,datová adresa,<adr>

Strojový kód:

1 0 1 1 0 1 0 1	datová adresa	rel. offset
7 0	7 0	7 0

Činnost:
```
(PC) <- (PC) + 3
IF (A) <> (datová adresa)
THEN (PC) <- (PC) + rel. offset
IF (A)  < (datová adresa) THEN (C) <- 1
                          ELSE (C) <- 0
```

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x							

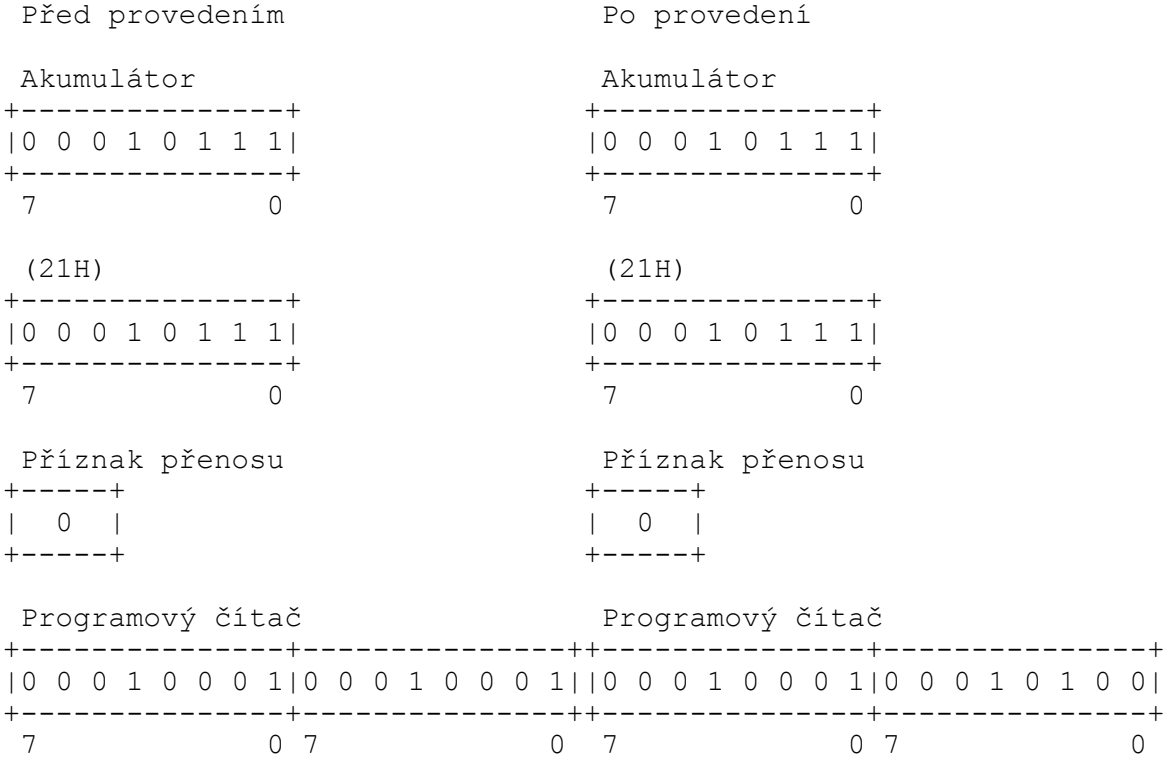
PSW

Popis: Tato instrukce porovnává hodnotu operandu obsaženého na zadané adrese s obsahem akumulátoru. Pokud jsou tyto hodnoty různé, provede se skok na uvedenou adresu v paměti programu. Pokud jsou hodnoty stejné, pokračuje se dále ve vykonávání programu.
Pokud je hodnota operandu v paměti dat větší než hodnota, obsažená v akumulátoru, nastaví se příznak přenosu na hodnotu "1", v opačném případě se příznak přenosu nastaví na "0".
Programový čítač se zvětší o tři. Pokud se operandy nerovnají, přičte se ještě hodnota relativního offsetu.

Příklad:
```
ORG 1111H
CJNE A,21H,RES    ;skok, když operand na adrese 21H
                  ;není roven obsahu akumulátoru
RES:
```

Kód instrukce:

1 0 1 1 0 1 0 1	0 0 1 0 0 0 0 1	0 0 0 0 0 0 0 0
7 0	7 0	7 0



Poznámky: 8,10,11,12

CJNE

Porovnání obsahu registru a datového operandu,
provede skok, pokud se nerovnají

Mnemonika: CJNE

Operandy:
```
Rr              registr  0 <= r <= 7
data            -256 <= data <= +255
adresa v paměti programu
```

Zápis: CJNE Rr,#data,<adr>

Strojový kód:

1 0 1 1 1 r r r	d a t a	rel. offset
7 0	7 0	7 0

Činnost:
```
(PC) <- (PC) + 3
IF (Rr)<>data THEN (PC) <- (PC) + rel. offset
IF (Rr) <data THEN (C) <- 1
               ELSE (C) <- 0
```

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x							

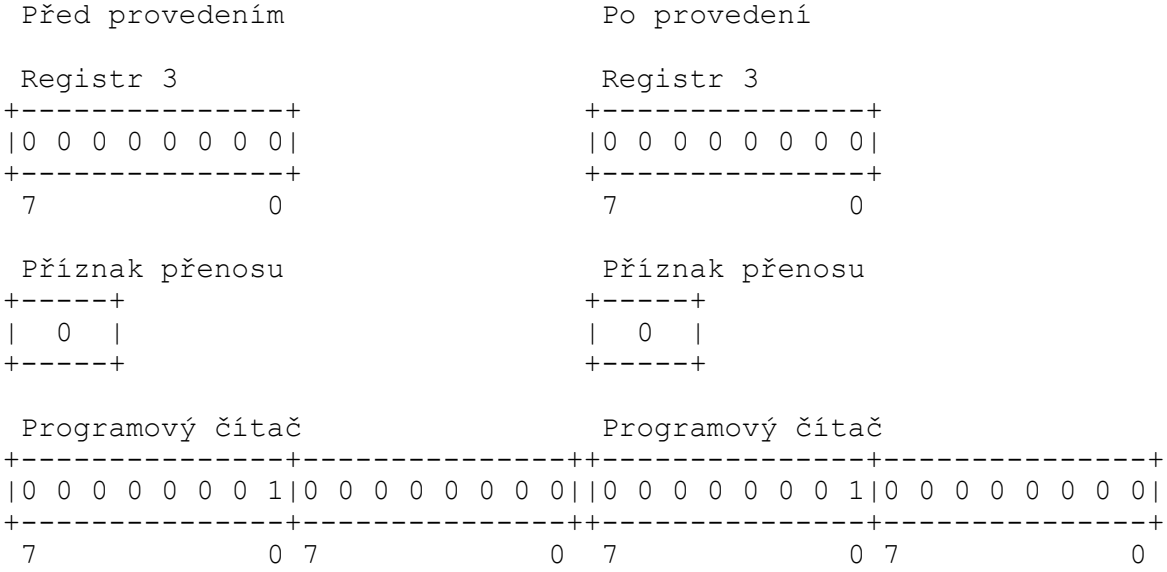
PSW

Popis: Tato instrukce porovnává hodnotu datového operandu s obsahem registru Rr. Pokud jsou tyto hodnoty různé, provede se skok na uvedenou adresu v paměti programu. Pokud jsou hodnoty stejné, pokračuje se dále ve vykonávání programu.
Pokud je hodnota datového operandu obsaženého v instrukci větší než hodnota, obsažená v registru, nastaví se příznak přenosu na hodnotu "1", v opačném případě se příznak přenosu nastaví na "0".
Programový čítač se zvětší o tři. Pokud se operandy nerovnají, přičte se ještě hodnota relativního offsetu.

Příklad:
```
CJNE R3,#0H,$     ;nekonečná smyčka, dokud R3=0
```

Kód instrukce:

1 0 1 1 1 0 1 1	0 0 0 0 0 0 0 0	1 1 1 1 1 1 0 1
7 0	7 0	7 0



Poznámky: 4,10,11,12,15

CLR

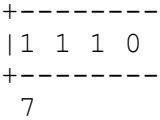
Nulování obsahu akumulátoru

Mnemonika: CLR

Operandy: A akumulátor

Zápis: CLR A

Strojový kód:



Činnost: (A) <- 0

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

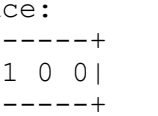
C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce nastaví obsah akumulátoru na hodnotu "0".

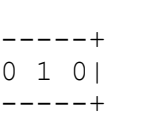
Příklad: CLR A

Kód instrukce:



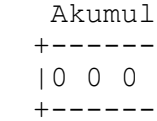
Před provedením

Akumulátor



Po provedení

Akumulátor



Poznámky: 5

CLR

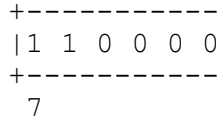
Nulování obsahu příznaku přenosu

Mnemonika: CLR

Operandy: C příznak přenosu

Zápis: CLR C

Strojový kód:



Činnost: (C) <- 0

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

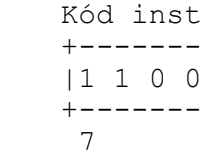
C	AC	F0	RS1	RS0	OV		P
x							

PSW

Popis: Tato instrukce nastaví obsah příznaku přenosu na hodnotu "0".

Příklad: CLR C

Kód instrukce:



Před provedením	Po provedení
Příznak přenosu	Příznak přenosu
1	0

Poznámky: Žádná

CLR

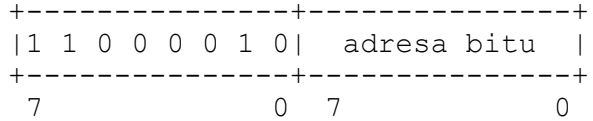
Nulování obsahu bitu

Mnemonika: CLR

Operandy: adresa bitu 0 <= adresa bitu <=0

Zápis: CLR adresa bitu

Strojový kód:



Činnost: (adresa bitu) <- 0

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

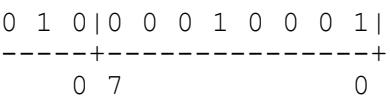
C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce nastaví obsah bitu na hodnotu "0".

```
Příklad:      CLR  22H.1     ;Nastav bit 1 u byte na adrese 22H
                             ;na hodnotu "0"
```

Kód instrukce:

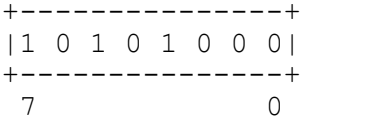


Před provedením

(22H)

Po provedení

(22H)



Poznámky: 18

CPL

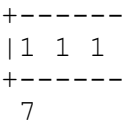
Komplement obsahu akumulátoru

Mnemonika: CPL

Operandy: A akumulátor

Zápis: CPL A

Strojový kód:



Činnost: (A) <- NOT (A)

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

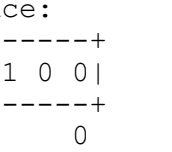
C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce neguje každý bit akumulátoru.

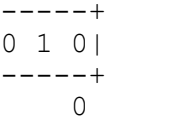
Příklad: CPL A

Kód instrukce:



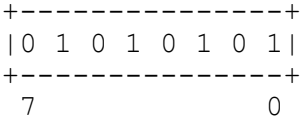
Před provedením

Akumulátor



Po provedení

Akumulátor



Poznámky: Žádná

CPL

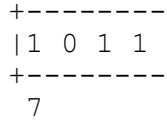
Komplement obsahu příznaku přenosu

Mnemonika: CPL

Operandy: C příznak přenosu

Zápis: CPL C

Strojový kód:



Činnost: (C) <- NOT (C)

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

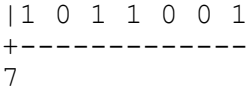
C	AC	F0	RS1	RS0	OV		P
x							

PSW

Popis: Tato instrukce neguje příznak přenosu.

Příklad: CPL C

Kód instrukce:



Před provedením	Po provedení
Příznak přenosu	Příznak přenosu
0	1

Poznámky: Žádná

CPL

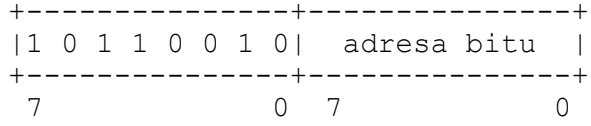
Komplement obsahu bitu

Mnemonika: CPL

Operandy: adresa bitu 0 <= adresa bitu <= 255

Zápis: CPL adresa bitu

Strojový kód:

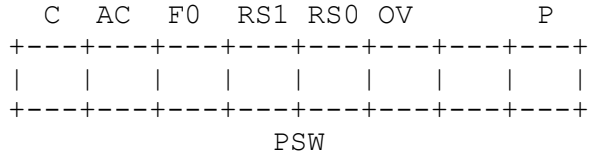


Činnost: (adresa bitu) <- NOT (adresa bitu)

Délka: 2 byte

Doba trvání: 1 strojový cyklus

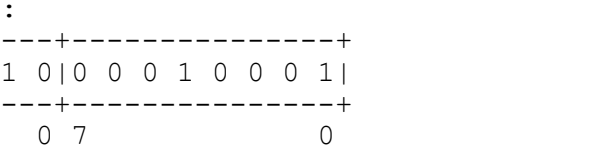
Ovlivňuje:



Popis: Tato instrukce neguje obsah bitu na zadané adrese.

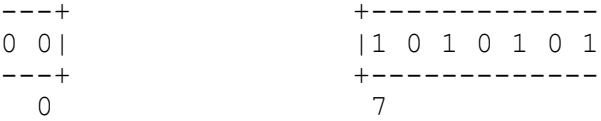
Příklad: CPL 22H.1 ;Neguj bit 1 u byte na adrese 22H

Kód instrukce:



Před provedením

(22H)



Po provedení

(22H)

Poznámky: 18

DA

Desítková úprava obsahu akumulátoru

Mnemonika: DA

Operandy: A akumulátor

Zápis: DA A

Strojový kód:

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

Činnost: Upraví obsah akumulátoru na desítkový tvar.

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x							x

PSW

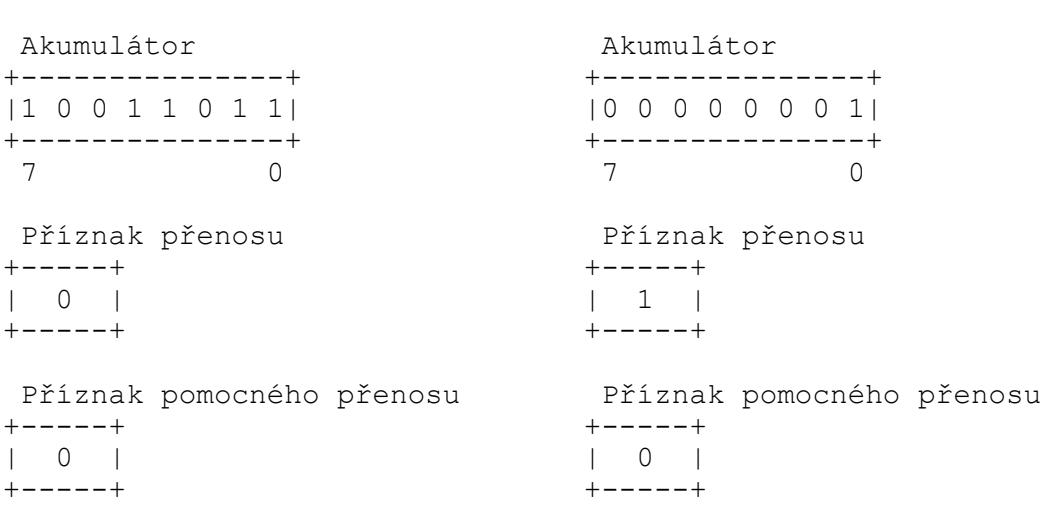
Popis: Tato instrukce upravuje obsah akumulátoru tak, aby odpovídal zhuštěnému tvaru dvojkově kódovaných desítkových číslic (kód BCD). Instrukce DA může být použita pouze po instrukcích sčítání dvou čísel, která jsou v kódu BCD. Má-li příznak pomocného přenosu logickou hodnotu 1 nebo obsahují-li nižší řády (bity 0 až 3) akumulátoru číslici větší než 9, přičte se k obsahu akumulátoru číslo 6. Má-li příznak přenosu před nebo po tomto přičtení logickou hodnotu 1 nebo obsahují-li vyšší řády (bity 4 až 7) číslici větší než 9, přičte se k obsahu akumulátoru číslo 60H. Akumulátor a příznak přenosu obsahují výsledné hodnoty po provedené desítkové úpravě.

Příklad:

```
ADD   A,R1          ;součet dvou čísel
DA    A             ;úprava obsahu akumulátoru na
                    ;desítkový tvar
```

Kód instrukce:

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


Před provedením
Akumulátor
1 0 0 1 1 0 1 1
7 0
Příznak přenosu
0
Příznak pomocného přenosu
0
Po provedení
Akumulátor
0 0 0 0 0 0 0 1
7 0
Příznak přenosu
1
Příznak pomocného přenosu
0

Poznámky: 5, 6

DEC

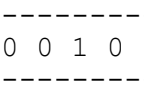
Snížení obsahu nepřímé adresy o jedničku

Mnemonika: DEC

Operandy: Rr registr 0 <= r <= 7

Zápis: DEC @Rr

Strojový kód:

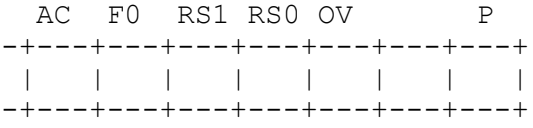


Činnost: ((Rr)) <-((Rr)) - 1

Délka: 1 byte

Doba trvání: 1 strojový cyklus

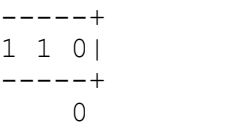
Ovlivňuje:

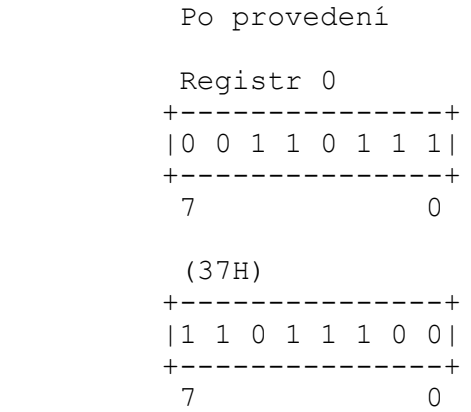


Popis: Tato instrukce odečte jedničku od obsahu paměťového místa na adrese určené obsahem registru r. Výsledek je uložen v adresovaném paměťovém místě.

Příklad:
```
DEC   @R0           ;snížení obsahu čítače o 1
```

Kód instrukce:





Poznámky: 15

DEC

Snížení obsahu akumulátoru o jedničku

Mnemonika: DEC

Operandy: A akumulátor

Zápis: DEC A

Strojový kód:

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

Činnost: (A) <- (A) - 1

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce odečte jedničku od obsahu akumulátoru
Výsledek je uložen v akumulátoru.

Příklad: DEC A ;snížení obsahu akumulátoru o 1

Kód instrukce:

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

Před provedením

Akumulátor

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

Po provedení

Akumulátor

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

Poznámky: 5

DEC

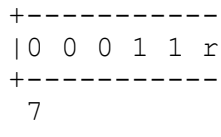
Snížení obsahu registru o jedničku

Mnemonika: DEC

Operandy: Rr registr 0 <= r <= 7

Zápis: DEC Rr

Strojový kód:

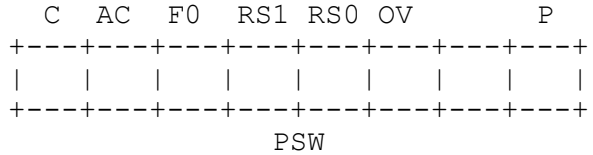


Činnost: (Rr) <- (Rr) - 1

Délka: 1 byte

Doba trvání: 1 strojový cyklus

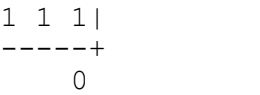
Ovlivňuje:



Popis: Tato instrukce odečte jedničku od obsahu registru r.
Výsledek je uložen v registru r.

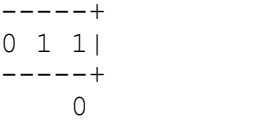
Příklad: DEC R7 ;snížení obsahu registru 7 o 1

Kód instrukce:



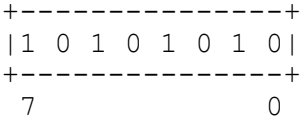
Před provedením

Registr 7



Po provedení

Registr 7



Poznámky: žádné

DEC

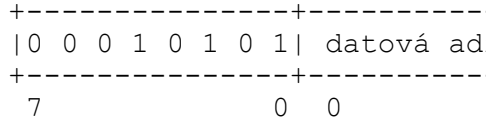
Snížení obsahu datové adresy o jedničku

Mnemonika: DEC

Operandy: datová adresa 0 <= datová adresa <= 255

Zápis: DEC datová adresa

Strojový kód:

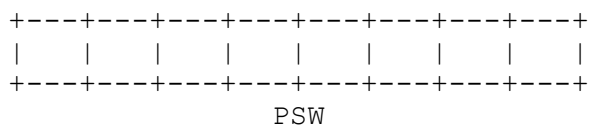


Činnost: (datová adresa) <- (datová adresa) - 1

Délka: 2 byte

Doba trvání: 1 strojový cyklus

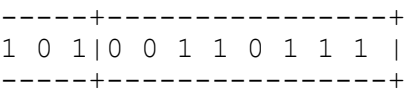
Ovlivňuje:



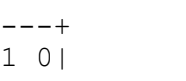
Popis: Tato instrukce odečte jedničku od obsahu určené datové adresy. Výsledek je uložen v dané datové adrese.

Příklad: DEC 37H ;snížení čítače o 1

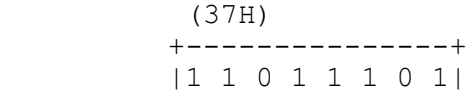
Kód instrukce:



Před provedením



Po provedení



Poznámky: 9, 18

DIV

Dělení obsahu akumulátoru obsahem registru B

Mnemonika: DIV

Operandy: AB dvojice registrů

Zápis: DIV AB

Strojový kód:

1	0	0	0	0	1	0	0
7							0

Činnost: (AB) <- (A) / (B)

Délka: 1 byte

Doba trvání: 4 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x					x		x

PSW

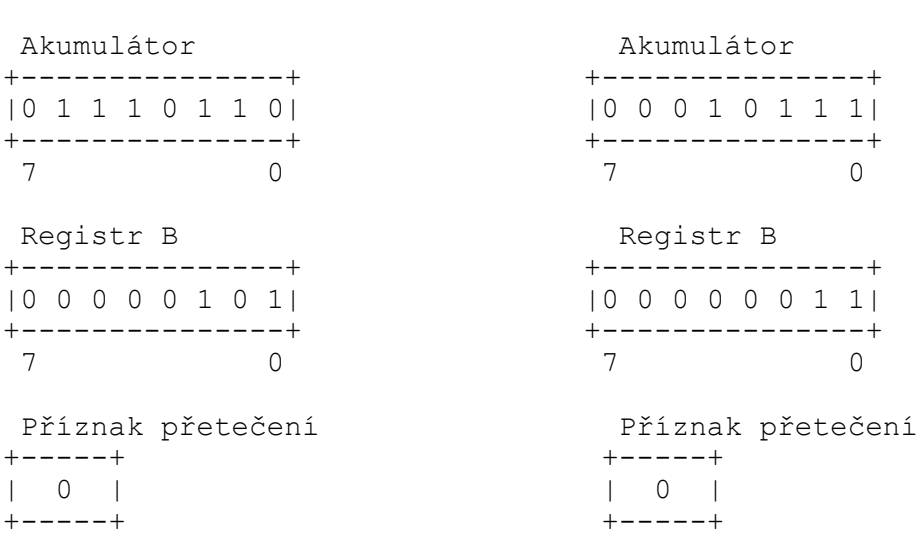
Popis: Tato instrukce vydělí obsah akumulátoru obsahem registru B. Oba operandy jsou pokládány za celá čísla bez znaménka. Po dělení obsahuje akumulátor podíl a registr B obsahuje zbytek. Příznak přenosu se vždy vynuluje. Při dělení nulou se příznak přetečení nastaví na logickou hodnotu 1, jinak má příznak přetečení logickou hodnotu 0.

Příklad:

```
MOV  B,5          ;naplnění registru B hodnotou 5
DIV  AB           ;dělení obsahu akumulátoru pěti
```

Kód instrukce:

1	0	0	0	0	1	0	0
7							0


Před provedením
Akumulátor
0 1 1 1 0 1 1 0
7 0
Registr B
0 0 0 0 0 1 0 1
7 0
Příznak přetečení
0
Po provedení
Akumulátor
0 0 0 1 0 1 1 1
7 0
Registr B
0 0 0 0 0 0 1 1
7 0
Příznak přetečení
0

Poznámky: 5

DJNZ

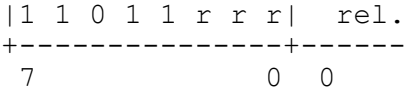
Zmenšení obsahu registru o jedničku,
skok při nenulovém výsledku

Mnemonika: DJNZ

Operandy: Rr registr 0 <= r <= 7
adresa v paměti programu

Zápis: DJNZ Rr,<adr>

Strojový kód:

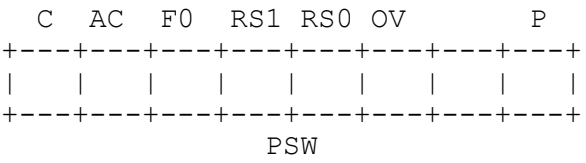


Činnost:
(PC) <- (PC)+2
(Rr) <- (Rr) -1
IF (Rr)<>0 THEN (PC) <- (PC) + rel. offset

Délka: 2 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

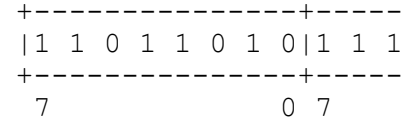


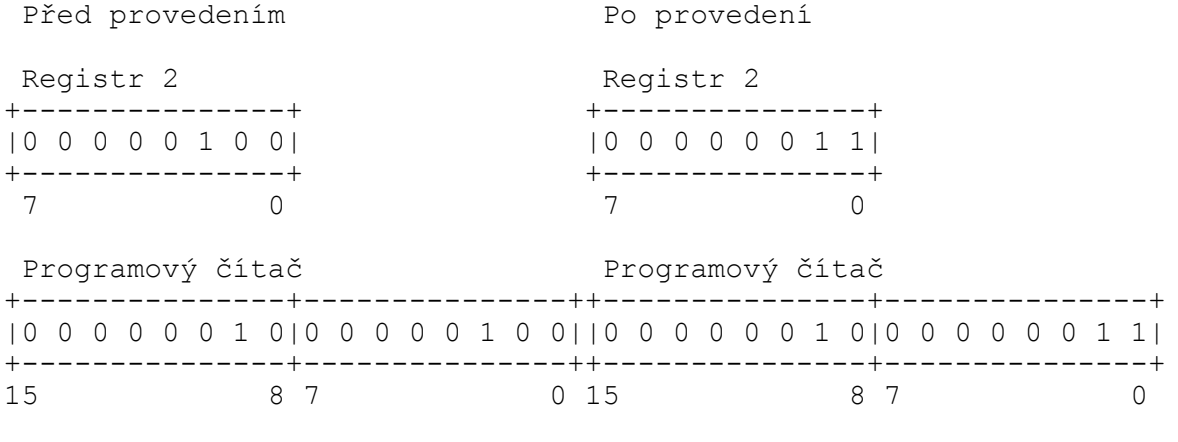
Popis: Tato instrukce sníží hodnotu obsaženou v registru o 1. Pokud je výsledek nenulový, provede skok na adresu určenou druhým operandem. V případě nulového výsledku se provede následující instrukce.

Programový čítač se zvětší o dva. Pokud je výsledek nenulový, přičte se ještě hodnota relativního offsetu.

Příklad:
```
ORG 23H
LOOP:
INC  DPTR
DJNZ R2,LOOP
```

Kód instrukce:





Poznámky: 10,11,12

DJNZ

Zmenšení obsahu datové adresy o jedničku,
skok při nenulovém výsledku

Mnemonika: DJNZ

Operandy: datová adresa 0 <= datová adresa <= 255
adresa v paměti programu

Zápis: DJNZ datová adresa,<adr>

Strojový kód:

1 1 0 1 0 1 0 1	datová adresa	rel. offset
7 0	7 0	7 0

Činnost:
(PC) <- (PC)+3
(datová adresa) <- (datová adresa) -1
IF (datová adresa)<>0 THEN (PC) <- (PC)+rel. offset

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

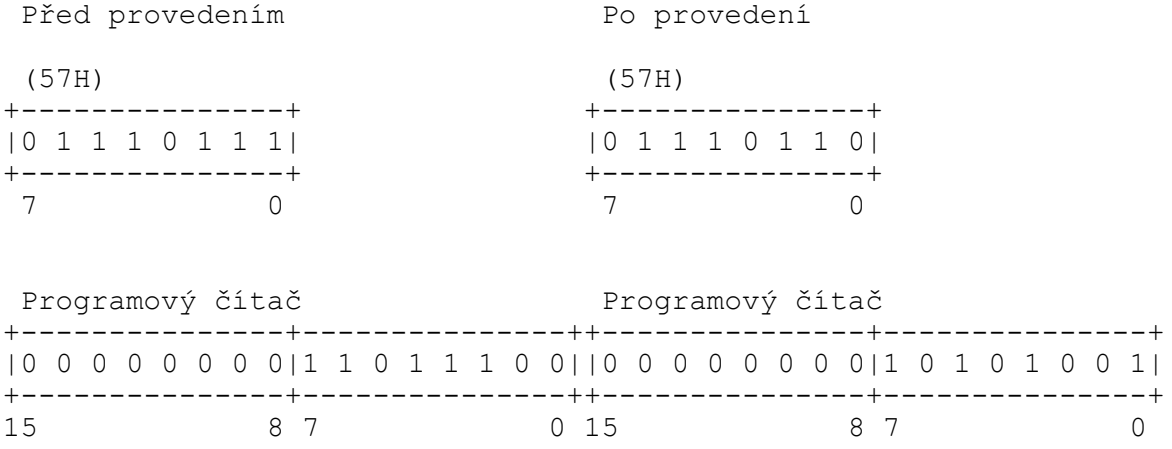
Popis: Tato instrukce sníží hodnotu obsahu datové adresy o 1. Pokud je výsledek nenulový, provede skok na adresu určenou druhým operandem. V případě nulového výsledku se provede následující instrukce.

Programový čítač se zvětší o dva. Pokud je výsledek nenulový, přičte se ještě hodnota relativního offsetu.

Příklad:
```
LOOP: MOV R7,57H    ;nastavení indexu cyklu do R7
        .
        .
      DJNZ 57H,LOOP ;snížení obsahu na adrese 57H o 1
      INC  A        ;skok  zpět na LOOP ( 51 slabik zpět
                    ;od instrukce INC  A)
```

Kód instrukce:

1 1 0 1 0 1 0 1	0 1 0 1 0 1 1 1	1 1 0 0 1 0 1 0
7 0	7 0	7 0



Poznámky: 9, 10, 11, 12

INC

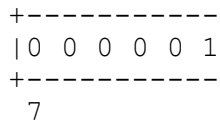
Zvýšení obsahu nepřímé adresy o jedničku

Mnemonika: INC

Operandy: Rr Registr 0 <= r <= 1

Zápis: INC @Rr

Strojový kód:



Činnost: ((Rr)) <- ((Rr)) +1

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

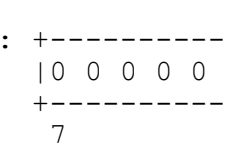
PSW

Popis: Tato instrukce zvýší obsah paměťového místa na adrese určené registrem r o 1. Výsledek je uložen na určené paměťové místo.

Příklad:

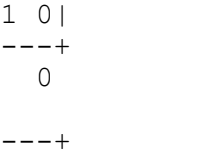
```
INC   @R0           ;zvýšení obsahu čítače o 1
```

Kód instrukce:

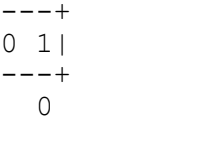


Před provedením

Registr 0

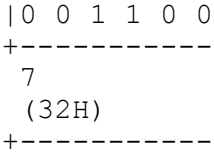


(32H)

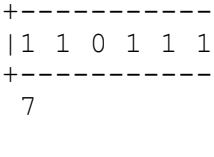


Po provedení

Registr 0



(32H)



Poznámky : 15

INC

Zvýšení obsahu akumulátoru o jedničku

Mnemonika: INC

Operandy: A Akumulátor

Zápis: INC A

Strojový kód:

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

Činnost: (A) <- (A) +1

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce zvýší obsah akumulátoru o 1. Výsledek je uložen v akumulátoru.

Příklad:
```
INC   A             ;zvýšení obsahu akumulátoru o 1
```

Kód instrukce:

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

Před provedením

Akumulátor

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

Po provedení

Akumulátor

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

Poznámky : 5

INC

Zvýšení obsahu ukazatele dat o jedničku

Mnemonika: INC

Operandy: DPTR Ukazatel dat

Zápis: INC DPTR

Strojový kód:

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

Činnost: (DPTR) <- (DPTR) +1

Délka: 1 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce zvýší obsah 16-bitového ukazatele dat o 1. Výsledek je uložen v ukazateli dat.

Příklad:
```
INC   DPTR         ;zvýšení obsahu ukazatele dat o 1
```

Kód instrukce:

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

Před provedením

Ukazatel dat

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

Po provedení

Ukazatel dat

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

Poznámky : žádné

INC

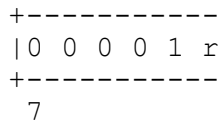
Zvýšení obsahu registru o jedničku

Mnemonika: INC

Operandy: Rr Registr 0 <= r <= 7

Zápis: INC Rr

Strojový kód:

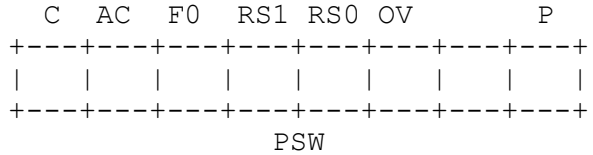


Činnost: (Rr) <- (Rr) +1

Délka: 1 byte

Doba trvání: 1 strojový cyklus

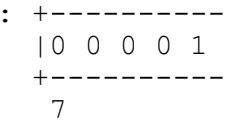
Ovlivňuje:



Popis: Tato instrukce zvýší obsah registru r o 1.
Výsledek je uložen v registru r.

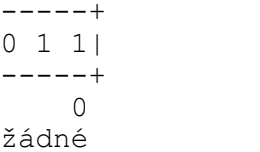
Příklad: INC R7 ;zvýšení obsahu registru 7 o 1

Kód instrukce:



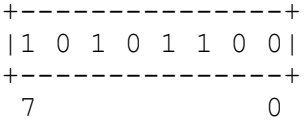
Před provedením

Registr 7



Po provedení

Registr 7



Poznámky : žádné

INC

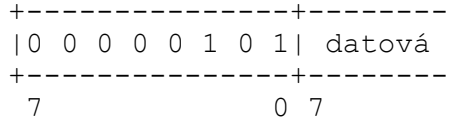
Zvýšení obsahu datové adresy o jedničku

Mnemonika: INC

Operandy: datová adresa 0 <= datová adresa <= 255

Zápis: INC datová adresa

Strojový kód:

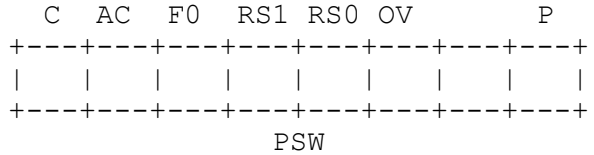


Činnost: (datová adresa) <- (datová adresa) +1

Délka: 2 byte

Doba trvání: 1 strojový cyklus

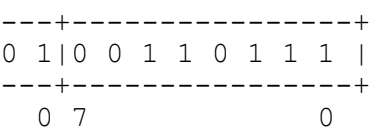
Ovlivňuje:



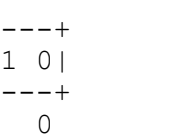
Popis: Tato instrukce přičte jedničku k obsahu určené datové adresy. Výsledek je uložen v dané datové adrese.

Příklad: INC 37H ;zvýšení obsahu čítače o 1

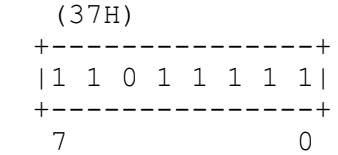
Kód instrukce:



Před provedením



Po provedení



Poznámky: 9

JB

Skok při nenulovém bitu

Mnemonika: JB

Operandy: adresa v paměti programu
adresa bitu 0 <= adresa bitu <= 255

Zápis: JB adresa bitu,<adr>

Strojový kód:

0 0 1 0 0 0 0 0	adresa bitu	rel.offset
7 0	7 0	7 0

Činnost: (PC) <- (PC) + 3
IF (adresa bitu) = 1 THEN (PC) <- (PC) + rel.offset

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

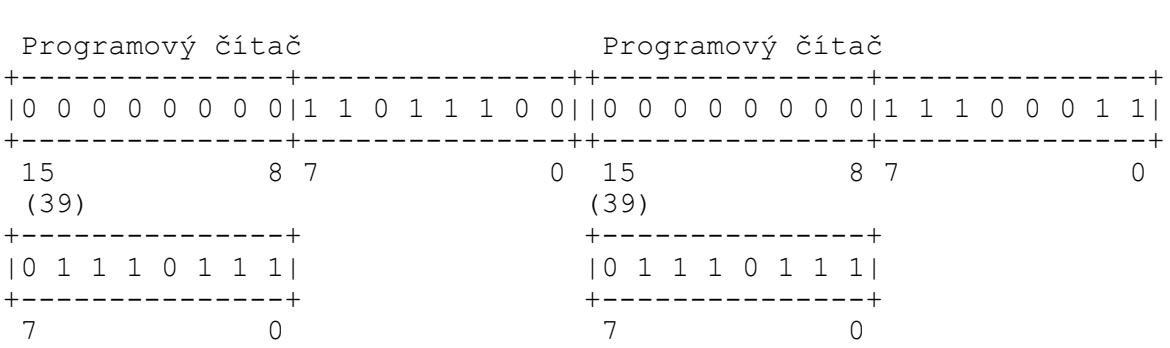
PSW

Popis: Tato instrukce testuje stav bitu určeného adresou bitu. Má-li bit logickou hodnotu 1, přechází řízení na určenou adresu v paměti programu. Obsahuje-li bit logickou hodnotu 0, pokračuje program následující instrukcí.
Programový čítač je inkrementován a ukazuje na následující instrukci. Byla-li splněna podmínka testu, přičte se k inkrementovanému programovému čítači relativní posun a bude se provádět instrukce umístěná na takto vzniklé adrese.

Příklad:
```
        JB   39.6,HOP      ;skok na návěští HOP, je-li bit
                           ;6 ve slabice 39 roven 1
             .
             .
HOP:    MOV A,39           ;přesun obsahu adresy 39 do
                           ;akumulátoru
```

Kód instrukce:

0 0 1 0 0 0 0 0	0 0 1 1 1 1 1 0	0 0 0 0 0 1 0 1
7 0	7 0	7 0



Poznámky: 10, 11, 12

JBC

Skok při nenulovém bitu s jeho vynulováním

Mnemonika: JBC

Operandy: adresa v paměti programu
adresa bitu 0 <= adresa bitu <= 255

Zápis: JBC adresa bitu,<adr>

Strojový kód:

0 0 0 1 0 0 0 0	adresa bitu	rel.offset
7 0	7 0	7 0

Činnost:
```
(PC) <- (PC) + 3
IF (adresa bitu) = 1 THEN
                     (adresa bitu) <- 0
                     (PC) <- (PC) + rel.offset
```

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

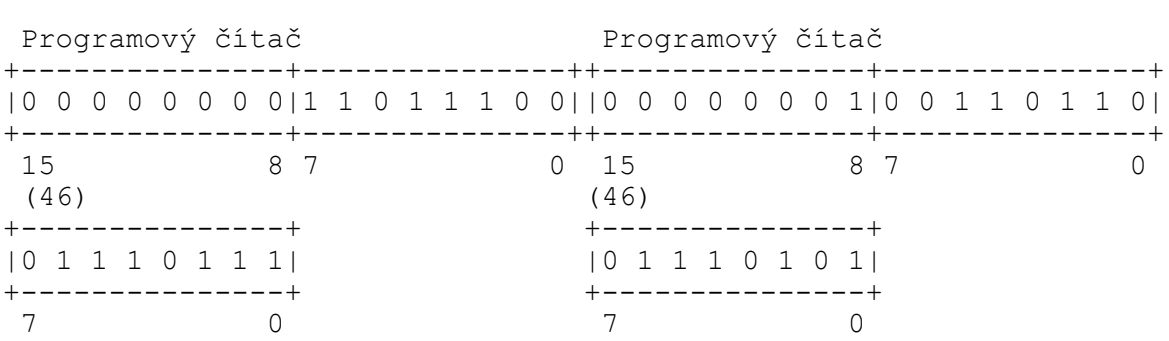
PSW

Popis: Tato instrukce testuje stav bitu určeného adresou bitu. Má-li bit logickou hodnotu 1, vynuluje se a řízení přechází na určenou adresu v paměti programu. Obsahuje-li bit logickou hodnotu 0, pokračuje program následující instrukcí.
Programový čítač je inkrementován a ukazuje na následující instrukci. Byla-li splněna podmínka testu, přičte se k inkrementovanému programovému čítači relativní posun a bude se provádět instrukce umístěná na takto vzniklé adrese.

Příklad:
```
        JBC   46.1,HOP1     ;skok na  návěští HOP1 a vynu -
                            ;lování je-li bit 1 ve slabice
                            ;46  roven 1
                  .
                  .
HOP1:   MOV A,H39
```

Kód instrukce:

0 0 0 1 0 0 0 0	0 1 1 1 0 0 0 1	0 1 0 1 0 1 1 1
7 0	7 0	7 0



Poznámky: 10, 11, 12

JC

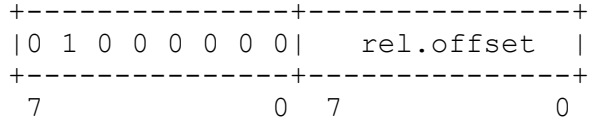
Skok při nenulovém příznaku přenosu

Mnemonika: JC

Operandy: adresa v paměti programu

Zápis: JC <adr>

Strojový kód:

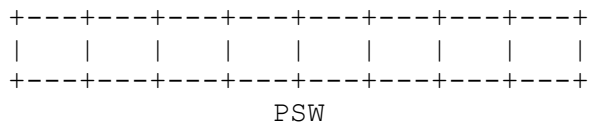


Činnost:
(PC) <- (PC) + 2
IF (C) = 1 THEN (PC) <- (PC) + rel.offset

Délka: 2 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

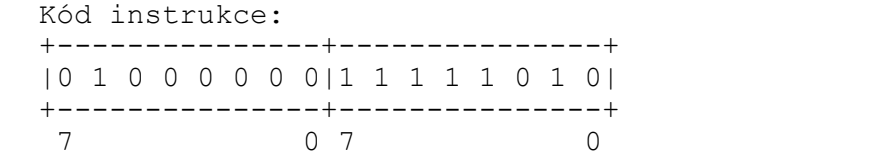


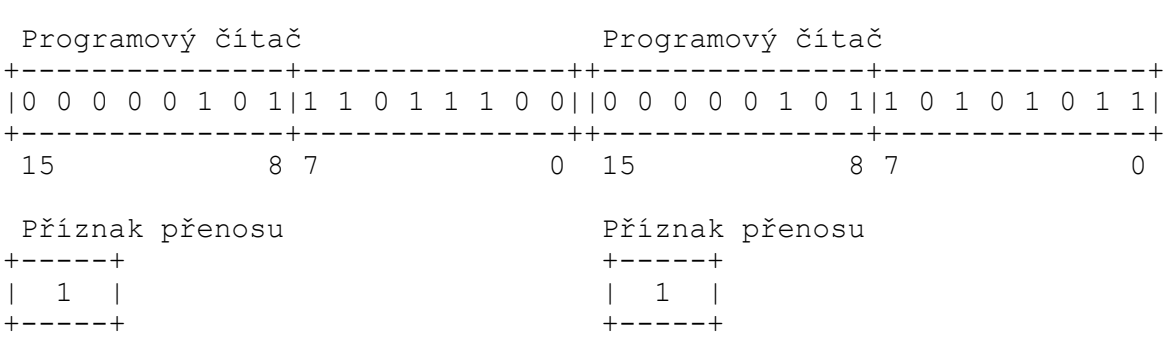
Popis: Tato instrukce testuje stav bitu příznaku přenosu. Má-li bit logickou hodnotu 1, přechází řízení na určenou adresu v paměti programu. Obsahuje-li bit logickou hodnotu 0, pokračuje program následující instrukcí.
Programový čítač je inkrementován a ukazuje na následující instrukci. Byla-li splněna podmínka testu, přičte se k inkrementovanému programovému čítači relativní posun a bude se provádět instrukce umístěná na takto vzniklé adrese.

Příklad:

```
NUL:  CLR    C          ;vynulování příznaku přenosu
      CJNE   A,3,$+3
      JC     NUL        ;je-li přenos, provede skok
                        ;na návěští NUL
```

Kód instrukce:





Poznámky: 10, 11, 12

JMP

Generovaná instrukce skoku

Mnemonika: JMP

Operandy: adresa v paměti programu

Zápis: JMP <adr>

Strojový kód: Přeloží se jako AJMP, SJMP nebo LJMP podle následujícího popisu.

Činnost: viz buď AJMP ,SJMP nebo LJMP

Délka: 2 nebo 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce se přeloží jako AJMP,SJMP nebo AJMP. Jako SJMP se přeloží tehdy, neobsahuje-li cílová adresa skoku dopředné odkazy a nachází-li se v rozsahu -128 až +127 od adresy následující instrukce. Instrukce bude přeložena jako AJMP, pokud cílová adresa skoku leží uvnitř 2kB bloku a její vyjádření neobsahuje žádné dopředné odkazy. V opačném případě se přeloží jako LJMP, ačkoliv může být cílová adresa dosažitelná i pomocí instrukce AJMP nebo SJMP. Pro popis činnosti viz kapitoly AJMP,LJMP a SJMP.

Příklad:

```
ORG 23H
SORT:
PUSH ACC            ;uloží obsah akumulátoru
     .
     .
ORG 35H
JMP  SORT           ;skok na návěští SORT
                    ;přeloží se jako SJMP
```

Kód instrukce: viz AJMP, SJMP nebo LJMP

Poznámky : 1,2,3

JMP

Skok podle součtu obsahu akumulátoru a ukazatele dat

Mnemonika: JMP

Operandy: A Akumulátor

DPTR Ukazatel dat

Zápis: JMP @A+DPTR

Strojový kód:

0	1	1	1	0	0	1	1
7							0

Činnost: (PC) <- (A) + (DPTR)

Délka: 1 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

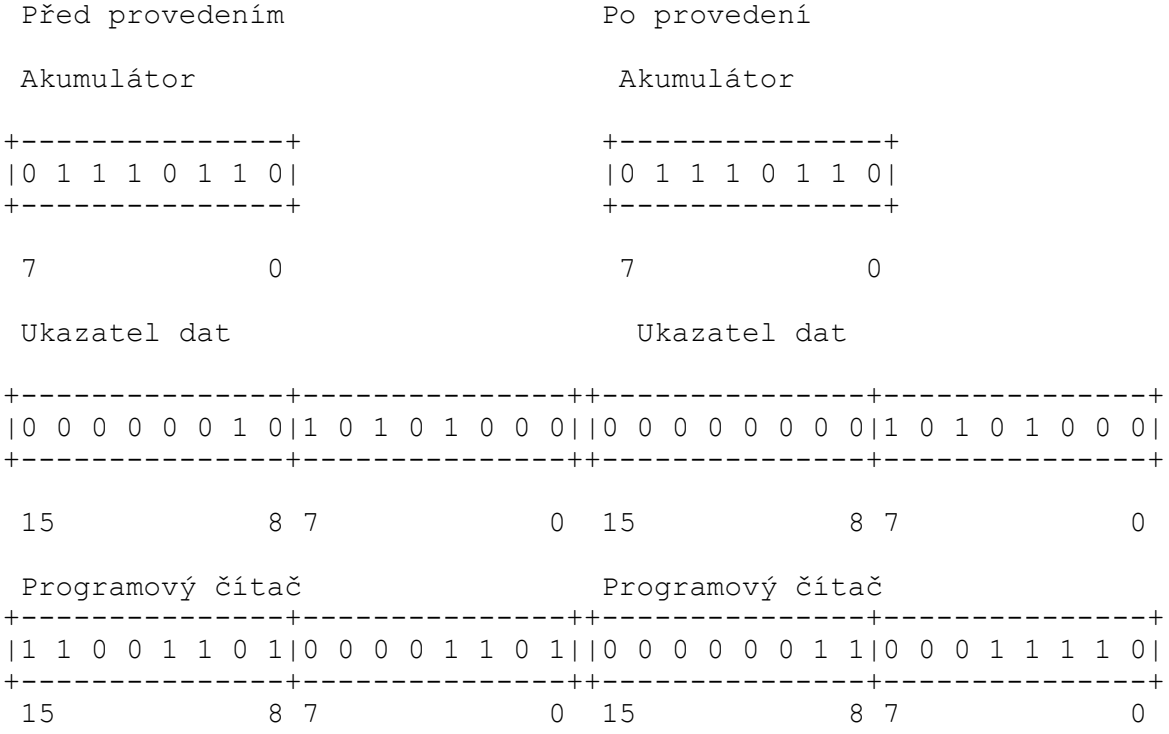
PSW

Popis: Tato instrukce sečte obsah akumulátoru s obsahem ukazatele dat. Řízení se předá na adresu vytvořenou tímto součtem.

Příklad: JMP @A+DPTR

Kód instrukce:

0	1	1	1	0	0	1	1
7							0



Poznámky: žádné

JNB

Skok při nulovém bitu

Mnemonika: JNB

Operandy: adresa v paměti programu
adresa bitu 0 <= adresa bitu <= 255

Zápis: JNB adresa bitu, <adr>

Strojový kód:

0 0 1 1 0 0 0 0	adresa bitu	rel.offset
7 0	7 0	7 0

Činnost: PC <- (PC) + 3
IF NOT (adresa bitu) THEN PC <- (PC) + rel.offset

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

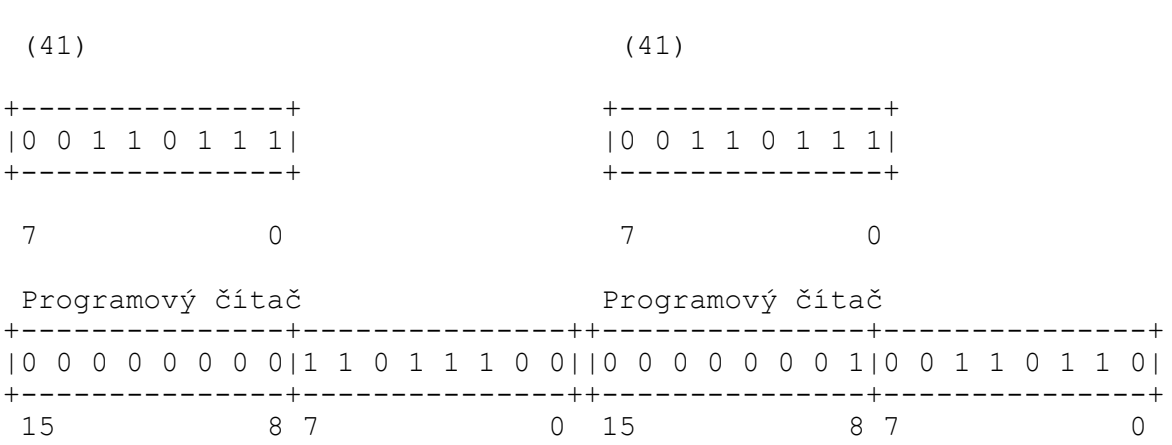
Popis: Tato instrukce testuje stav bitu určeného adresou bitu. Má-li bit logickou hodnotu 0, přechází řízení na určenou adresu v paměti programu. Obsahuje-li bit logickou hodnotu 1, pokračuje program následující instrukcí. Programový čítač je inkrementován a ukazuje na následující instrukci. Byla-li splněna podmínka testu, přičte se k inkrementovanému programovému čítači relativní posun a bude se provádět instrukce umístěná na takto vzniklé adrese.

Příklad:

```
        ORG   0DCH
        JNB   41.6,SKOK  ;skok na návěští SKOK, je-li bit 6
                         ;ve slabice 41 roven 0
              .
              .
              .
SKOK:   ADD   A,41
```

Kód instrukce:

0 0 1 1 0 0 0 0	0 1 0 0 1 1 1 0	0 1 0 1 0 1 1 1
7 0	7 0	7 0



Poznámky: 10, 11, 12

JNC

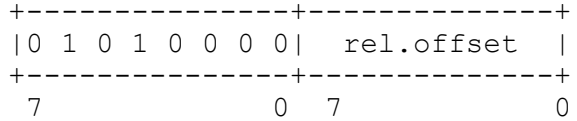
Skok při nulovém příznaku přenosu

Mnemonika: JNC

Operandy: adresa v paměti programu

Zápis: JNC <adr>

Strojový kód:

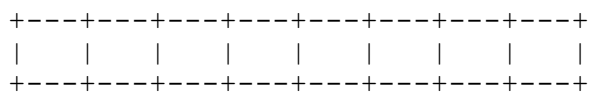


Činnost:
```
PC <- (PC) + 2
IF (C)  =  0  THEN  PC <- (PC) + rel.offset
```

Délka: 2 byte

Doba trvání: 2 strojové cykly

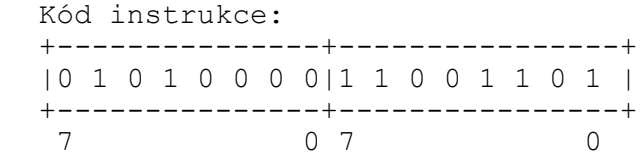
Ovlivňuje:

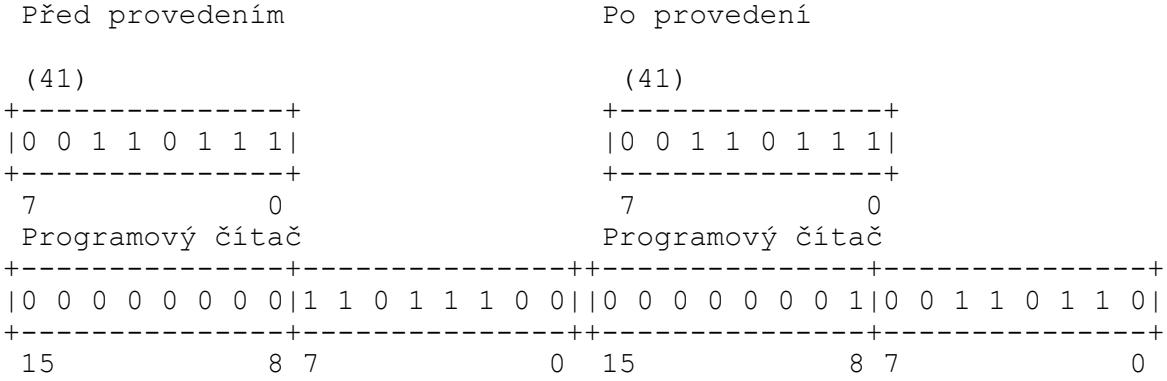


Popis: Tato instrukce testuje stav bitu příznaku přenosu. Má-li bit logickou hodnotu 0, přechází řízení na určenou adresu v paměti programu. Obsahuje-li bit logickou hodnotu 1, pokračuje program následující instrukcí. Programový čítač je inkrementován a ukazuje na následující instrukci. Byla-li splněna podmínka testu, přičte se k inkrementovanému programovému čítači relativní posun a bude se provádět instrukce umístěná na takto vzniklé adrese.

Příklad:
```
ZAC:  MOV   A,R5
      .
      .
      JNC   ZAC     ;skok na  návěští  ZAC,  je-li
                    ;nulový příznak přenosu
```

Kód instrukce:





Poznámky: 10, 11, 12

JNZ

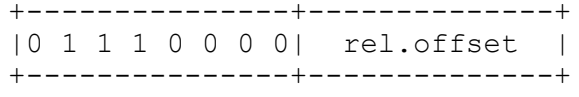
Skok při nenulovém obsahu akumulátoru

Mnemonika: JNZ

Operandy: adresa v paměti programu

Zápis: JNZ <adr>

Strojový kód:

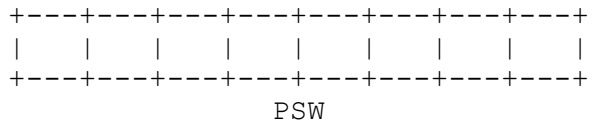


Činnost: PC <- (PC) + 2
IF (A) <> 0 THEN PC <- (PC) + rel.offset

Délka: 2 byte

Doba trvání: 2 strojové cykly

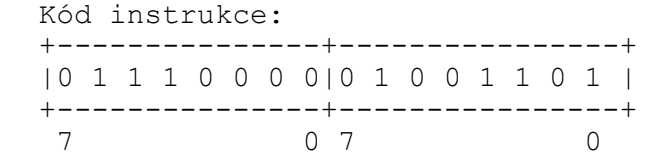
Ovlivňuje:

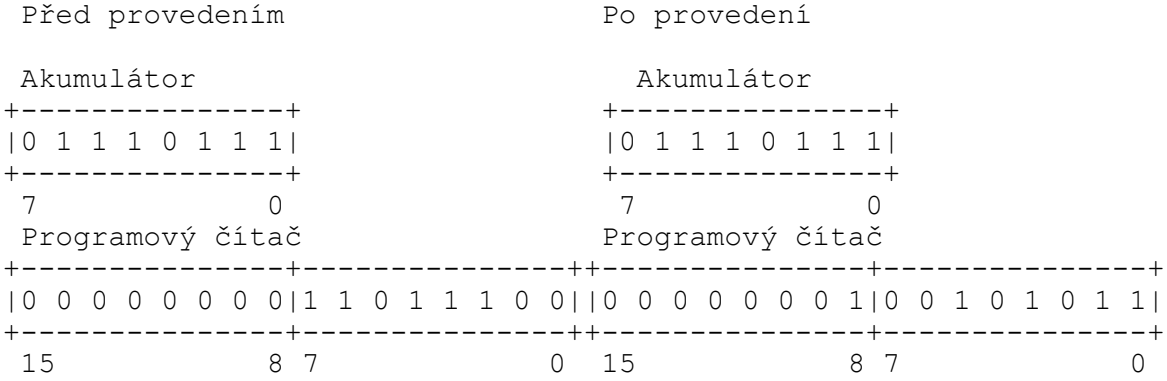


Popis: Tato instrukce testuje obsah akumulátoru. Je-li nenulový, přechází řízení na určenou adresu v paměti programu. Jinak program pokračuje následující instrukcí.
Programový čítač je inkrementován a ukazuje na následující instrukci. Byla-li splněna podmínka testu, přičte se k inkrementovanému programovému čítači relativní posun a bude se provádět instrukce umístěná na takto vzniklé adrese.

Příklad:
```
      JNZ    POKR        ;skok při nenulovém obsahu aku-
             .            ;mulátoru na návěští POKR
             .
POKR: MOV    R5,A
```

Kód instrukce:





Poznámky: 10, 11, 12

JZ

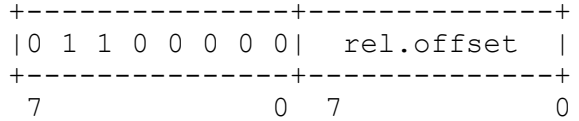
Skok při nulovém obsahu akumulátoru

Mnemonika: JZ

Operandy: adresa v paměti programu

Zápis: JZ <adr>

Strojový kód:

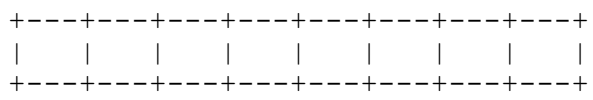


Činnost:
```
PC <- (PC) + 2
IF  (A)  =  0  THEN  PC <- (PC) + rel.offset
```

Délka: 2 byte

Doba trvání: 2 strojové cykly

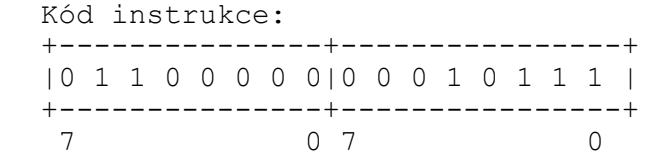
Ovlivňuje:

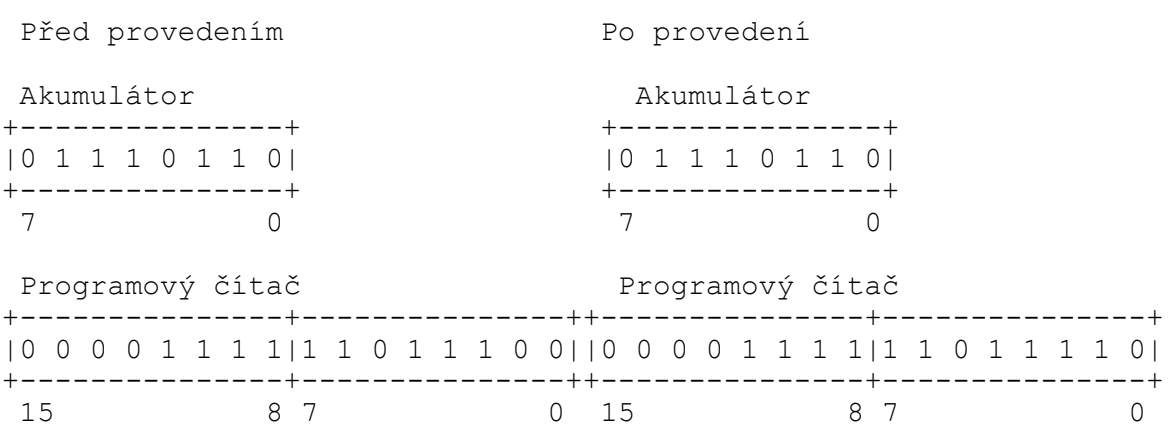


Popis: Tato instrukce testuje obsah akumulátoru. Je-li nulový, přechází řízení na určenou adresu v paměti programu. Jinak program pokračuje následující instrukcí.
Programový čítač je inkrementován a ukazuje na následující instrukci. Byla-li splněna podmínka testu, přičte se k inkrementovanému programovému čítači relativní posun a bude se provádět instrukce umístěná na takto vzniklé adrese.

Příklad:
```
        JZ    POKR          ;skok při nulovém obsahu aku-
              .             ;mulátoru na návěští POKR
              .
POKR:MOV    R5,A
```

Kód instrukce:




Před provedením
Po provedení
Akumulátor
Akumulátor
0 1 1 1 0 1 1 0
0 1 1 1 0 1 1 0
7 0
7 0
Programový čítač
Programový čítač
0 0 0 0 1 1 1 1 1 1 0 1 1 1 0 0
0 0 0 0 1 1 1 1 1 1 0 1 1 1 1 0
15 8 7 0
15 8 7 0

LCALL

Absolutní volání podprogramu v rozsahu celé programové paměti

Mnemonika: LCALL

Operandy: adresa v paměti programu

Zápis: LCALL <adr>

Strojový kód:

7							0	7							0	7							0
0	0	0	1	0	0	1	0	HIGH(<adr>)								LOW(<adr>)							

Činnost:
(PC) <- (PC)+3
(SP) <- (SP)+1
((SP)) <- LOW(PC)
(SP) <- (SP)+1
((SP)) <- HIGH(PC)
(PC) <- <adr>

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

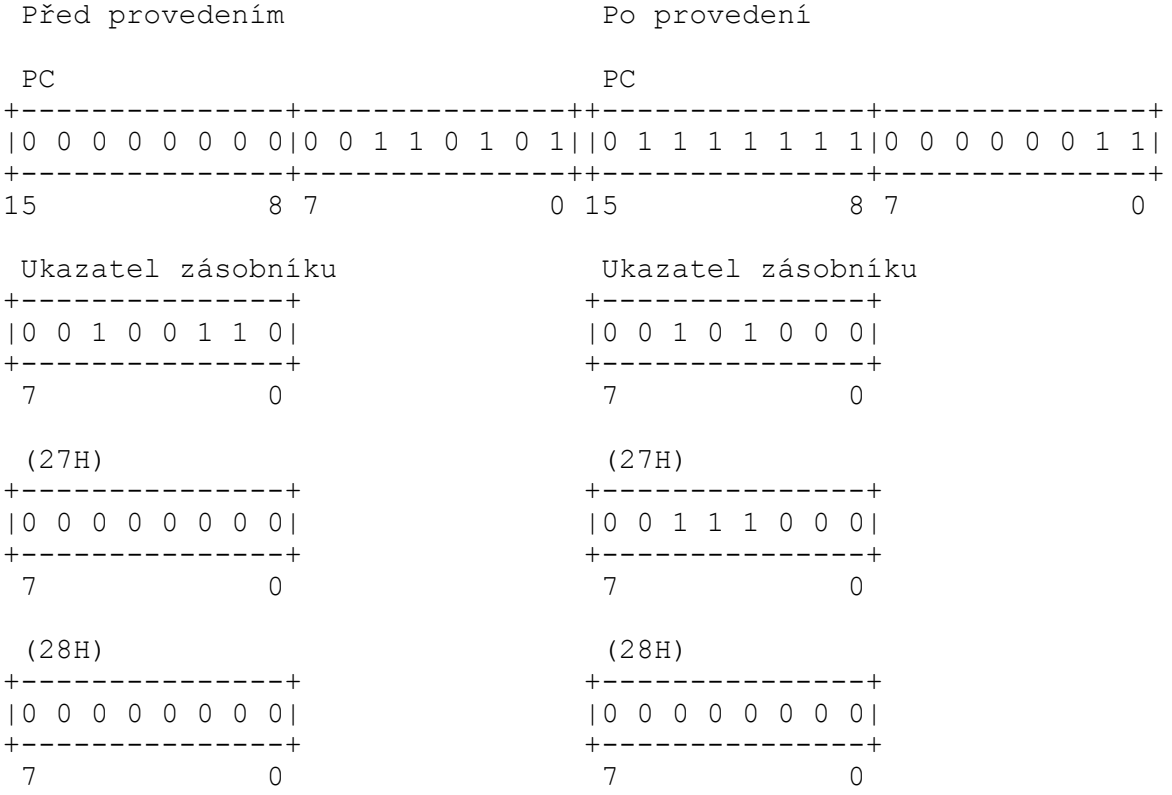
Popis: Tato instrukce uschová obsah programového čítače zvětšený o tři (návratovou adresu) do zásobníku. Nižší byte programového čítače je vždy umístěn v zásobníku první. Pak nahradí obsah programového čítače adresou, která je operandem instrukce. Cílová adresa volání může ležet kdekoliv v paměti programu.

Příklad:

```
ORG 23H
LCALL SORT     ;volání podprogramu SORT
      .
      .
ORG 7F03H
SORT:
      .
      .
      .
RET
```

Kód instrukce:

7							0	7							0	7							0
0	0	0	1	0	0	1	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1



Poznámky : 1,2,3

LJMP

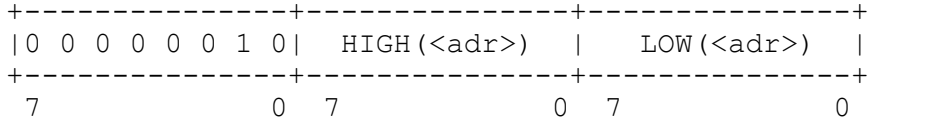
Absolutní skok v rozsahu celé programové paměti

Mnemonika: LJMP

Operandy: adresa v paměti programu

Zápis: LJMP <adr>

Strojový kód:

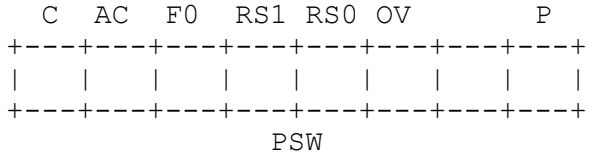


Činnost: (PC) <- <adr>

Délka: 3 byte

Doba trvání: 2 strojové cykly

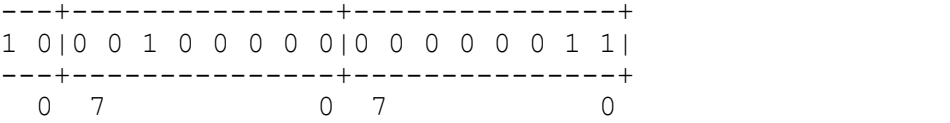
Ovlivňuje:



Popis: Tato instrukce provede skok na adresu, která je operandem instrukce.

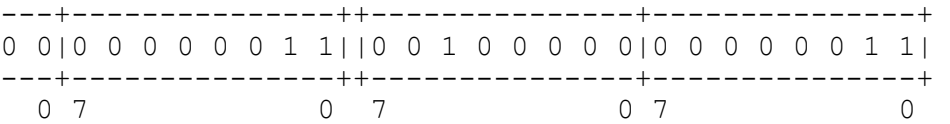
Příklad:
```
ORG 3
LJMP 2003H
```

Kód instrukce:



Před provedením

Programový čítač



Po provedení

Programový čítač

Poznámky: žádné

MOV

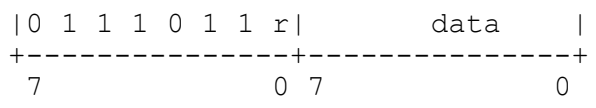
Přesun dat na nepřímou adresu

Mnemonika: MOV

Operandy: Rr registr 0 <= r <= 1
data -256 <= data <= +255

Zápis: MOV @Rr,#data

Strojový kód:

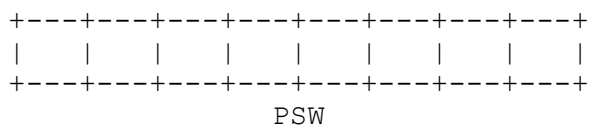


Činnost: ((Rr)) <- data

Délka: 2 byte

Doba trvání: 1 strojový cyklus

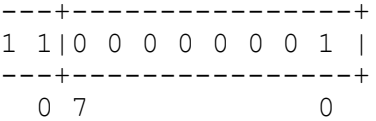
Ovlivňuje:



Popis: Tato instrukce provede přesun hodnoty 8-bitových dat do paměťového místa na adrese určené obsahem registru r.

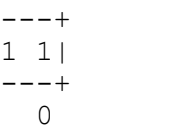
Příklad: MOV @R1,#01H ;přesun hodnoty 1 na místo
;určené obsahem registru R1

Kód instrukce:

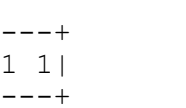


Před provedením

Registr 1

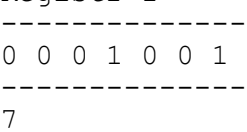


(13H)

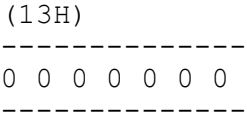


Po provedení

Registr 1



(13H)



Poznámky: 4, 15

MOV

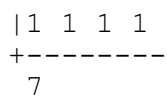
Přesun obsahu akumulátoru na nepřímou adresu

Mnemonika: MOV

Operandy: Rr registr 0 <= r <= 1
A akumulátor

Zápis: MOV @Rr,A

Strojový kód:

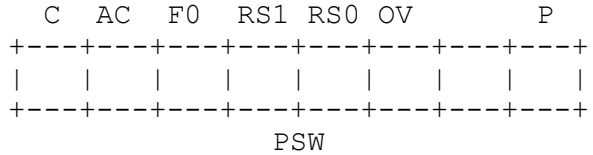


Činnost: ((Rr)) <- (A)

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

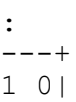


Popis: Tato instrukce provede přesun obsahu akumulátoru do paměťového místa na adrese určené obsahem registru r.

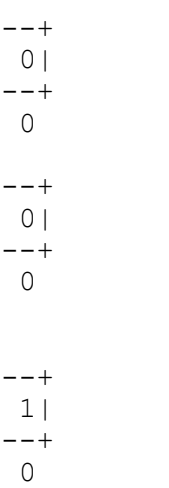
Příklad:

```
MOV  @R0,A          ;přesun obsahu akumulátoru
                    ;na místo určené  obsahem
                    ;registru R1
```

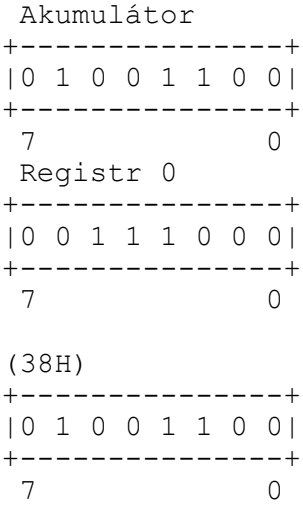
Kód instrukce:



Před provedením



Po provedení



Poznámky: 15

MOV

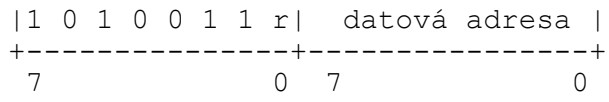
Přesun obsahu datové adresy na nepřímou adresu

Mnemonika: MOV

Operandy: Rr registr 0 <= r <= 1
datová adresa 0 <= datová adresa <= 255

Zápis: MOV @Rr,datová adresa

Strojový kód:

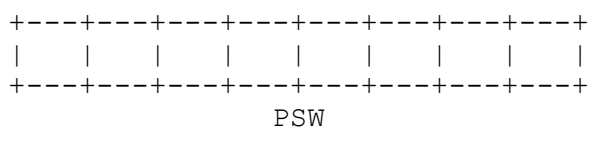


Činnost: ((Rr)) <- (datová adresa)

Délka: 2 byte

Doba trvání: 2 strojové cykly

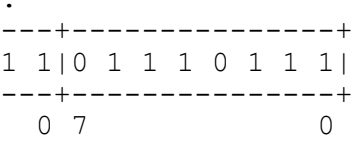
Ovlivňuje:

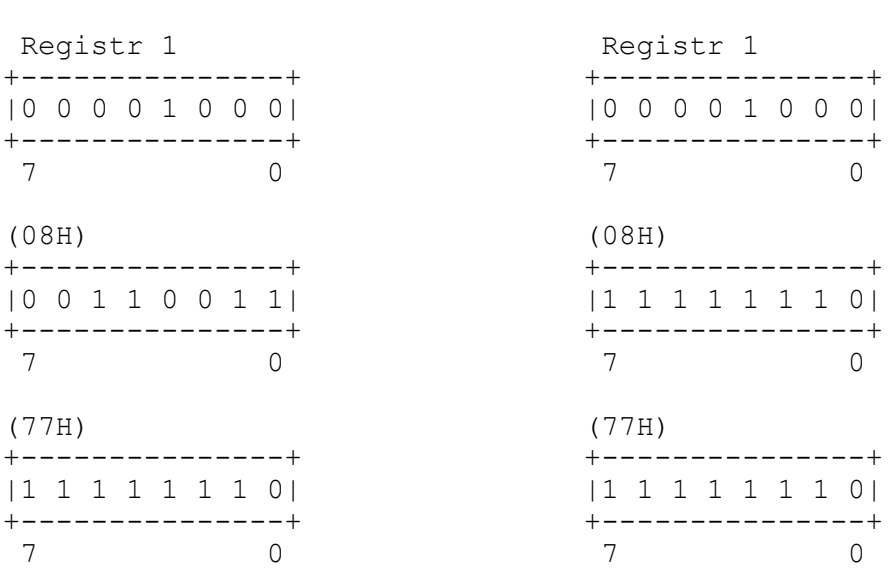


Popis: Tato instrukce provede přesun obsahu datové adresy do paměťového místa na adrese určené obsahem registru r.

```
Příklad:      MOV  @R1,77H        ;přesun obsahu adresy 77H
                                  ;na místo určené  obsahem
                                  ;registru R1
```

Kód instrukce:





Poznámky: 8, 15

MOV

Přesun dat do akumulátoru

Mnemonika: MOV

Operandy:
A	akumulátor
data	-256 <= data <= +255

Zápis: MOV A,#data

Strojový kód:

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

Činnost: (A) <- data

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce provede přesun hodnoty 8-bitových dat do akumulátoru.

Příklad:
```
MOV  A,#01H          ;přesun hodnoty 01H do
                     ;akumulátoru
```

Kód instrukce:

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

Před provedením

Akumulátor

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

Po provedení

Akumulátor

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

Poznámky: 4, 5

MOV

Přesun obsahu nepřímé adresy do akumulátoru

Mnemonika: MOV

Operandy:
A	akumulátor
Rr	registr 0 <= r <= 1

Zápis: MOV A,@Rr

Strojový kód:

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

Činnost: (A) <- ((Rr))

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Přesune obsah nepřímé adresy do akumulátoru.

Příklad: MOV A,@R1 ;obsah nepřímé adresy do akumulátoru

Kód instrukce:

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

Před provedením

Akumulátor

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

Registr 1

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

(1CH)

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

Po provedení

Akumulátor

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

Registr 1

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

(1CH)

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

Poznámky: 5, 15

MOV

Přesun obsahu registru do akumulátoru

Mnemonika: MOV

Operandy:

A	akumulátor
Rr	registr 0 <= r <= 7

Zápis: MOV A,Rr

Strojový kód:

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

Činnost: (A) <- (Rr)

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce provede přesun obsahu registru r do akumulátoru.

Příklad:

```
MOV A,R6            ;přesun obsahu registru R6
                    ;do akumulátoru
```

Kód instrukce:

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

Před provedením

Akumulátor

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

Registr 6

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

Po provedení

Akumulátor

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

Registr 6

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

Poznámky: 5

MOV

Přesun obsahu datové adresy do akumulátoru

Mnemonika: MOV

Operandy: A akumulátor
datová adresa 0 <= datová adresa <= 255

Zápis: MOV A,datová adresa

Strojový kód:

1 1 1 0 0 1 0 1	datová adresa
7 0	7 0

Činnost: (A) <- (datová adresa)

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce provede přesun obsahu datové adresy do akumulátoru.

Příklad:
```
MOV A,90H           ;přesun obsahu adresy 90H
                    ;do akumulátoru
```

Kód instrukce:

1 1 1 0 0 1 0 1	1 0 0 1 0 0 0 0
7 0	7 0

Před provedením

Akumulátor

0 0 1 0 0 1 1 0
7 0

(90H)

0 1 1 1 1 0 0 1
7 0

Po provedení

Akumulátor

0 1 1 1 1 0 0 1
7 0

(90H)

0 1 1 1 1 0 0 1
7 0

Poznámky: 5, 8

MOV

Přesun bitu do příznaku přenosu

Mnemonika: MOV

Operandy: C příznak přenosu
adresa bitu 0 <= adresa bitu <= 255

Zápis: MOV C,adresa bitu

Strojový kód:

1 0 1 0 0 0 1 0	adresa bitu
7 0	7 0

Činnost: (C) <- (adresa bitu)

Délka: 2 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x							

PSW

Popis: Tato instrukce provede přesun obsahu bitu určeného adresou bitu do příznaku přenosu.

```
Příklad:      MOV C,33.2          ;přesun  hodnoty  bitu  2
                                  ;na adrese 33 do příznaku
                                  ;přenosu
```

Kód instrukce:

1 0 1 0 0 0 1 0	0 0 0 0 1 0 1 0
7 0	7 0

Před provedením

(33)

0 0 1 1 0 1 1 0
7 0

Příznak přenosu

0

Po provedení

(33)

0 0 1 1 0 1 1 0
7 0

Příznak přenosu

1

Poznámky: žádné

MOV

Přesun dat do ukazatele dat

Mnemonika: MOV

Operandy: DPTR ukazatel dat
data 0 <= data <= 65535

Zápis: MOV DPTR,#data

Strojový kód:

1 0 0 1 0 0 0 0	HIGH data	LOW data
7 0	7 0	7 0

Činnost: (DPTR) <- data

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce provede přesun hodnoty 16-bitových dat do ukazatele dat.

Příklad: MOV DPTR,#0F4FH ;ukazatel dat = 0xF4F

Kód instrukce:

1 0 0 1 0 0 0 0	0 0 0 0 1 1 1 1	0 1 0 0 1 1 1 1
7 0	7 0	7 0

Před provedením

Ukazatel dat

0 0 0 0 0 0 0 0	1 1 0 1 1 1 0 0
15 8	7 0

Po provedení

Ukazatel dat

0 0 0 0 1 1 1 1	0 1 0 0 1 1 1 1
15 8	7 0

Poznámky: žádné

MOV

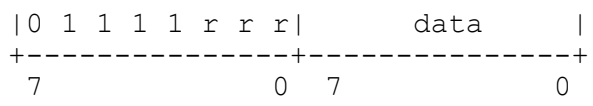
Přesun dat do registru

Mnemonika: MOV

Operandy: Rr — Registr 0 <= r <= 7
data — -256 <= data <= +255

Zápis: MOV Rr,#data

Strojový kód:

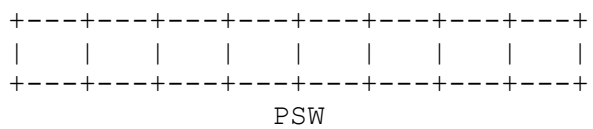


Činnost: (Rr) <- data

Délka: 2 byte

Doba trvání: 1 strojový cyklus

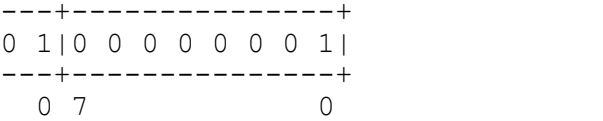
Ovlivňuje:



Popis: Tato instrukce provede přesun hodnoty 8-bitových dat do registru r.

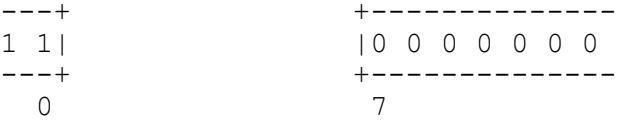
Příklad: MOV R5,#01H ;přesun dat do registru 5

Kód instrukce:



Před provedením

Registr 5



Po provedení

Registr 5

Poznámky: 4

MOV

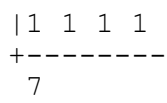
Přesun obsahu akumulátoru do registru

Mnemonika: MOV

Operandy: Rr — Registr 0 <= r <= 7
A — Akumulátor

Zápis: MOV Rr,A

Strojový kód:

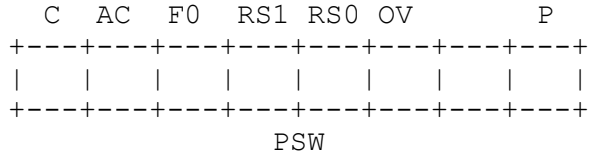


Činnost: (Rr) <- (A)

Délka: 1 byte

Doba trvání: 1 strojový cyklus

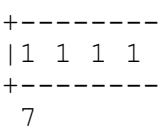
Ovlivňuje:



Popis: Tato instrukce provede přesun obsahu akumulátoru do registru r.

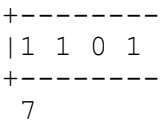
Příklad:
```
MOV R7,A            ;přesun obsahu  akumulátoru
                    ;do registru 7
```

Kód instrukce:

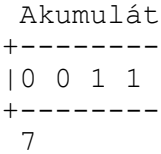


Před provedením

Registr 7



Akumulátor



Po provedení

Registr 7

Akumulátor

Poznámky: žádné

MOV

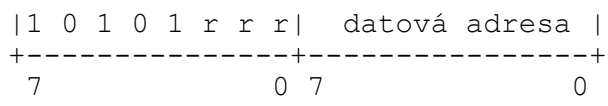
Přesun obsahu datové adresy do registru

Mnemonika: MOV

Operandy: Rr — Registr 0 <= r <= 7
datová adresa — 0 <= datová adresa <=255

Zápis: MOV Rr,datová adresa

Strojový kód:

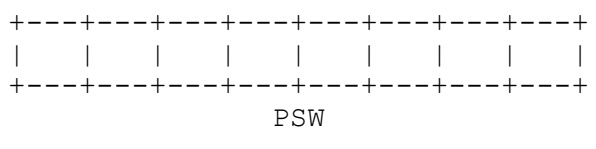


Činnost: (Rr) <- (datová adresa)

Délka: 2 byte

Doba trvání: 2 strojové cykly

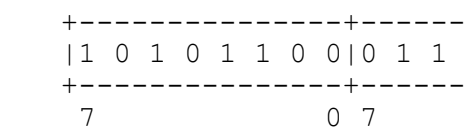
Ovlivňuje:



Popis: Tato instrukce provede přesun obsahu datové adresy do registru r.

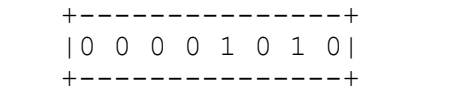
Příklad:
```
MOV R4,69H          ;přesun obsahu na adrese 69H
                    ;do registru 4
```

Kód instrukce:

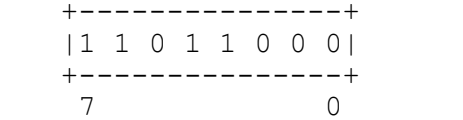


Před provedením

Registr 4

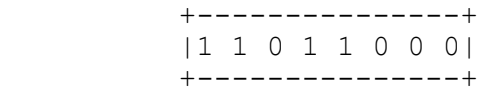


(69H)

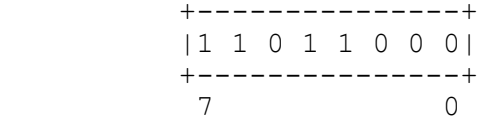


Po provedení

Registr 4



(69H)



Poznámky: 8

MOV

Přesun příznaku přenosu do bitu

Mnemonika: MOV

Operandy: adresa bitu 0 <= adresa bitu <= 255
C příznak přenosu

Zápis: MOV adresa bitu,C

Strojový kód:

1 0 0 1 0 0 1 0	adresa bitu
7 0	7 0

Činnost: (adresa bitu) <- (C)

Délka: 2 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x							

PSW

Popis: Tato instrukce provede přesun obsahu příznaku přenosu do bitu určeného adresou bitu.

Příklad:
```
MOV ACC.7,C          ;přesun hodnoty příznaku
                     ;přenosu do sedmého bitu
                     ;akumulátoru
```

Kód instrukce:

1 0 0 1 0 0 1 0	1 1 1 0 0 1 1 1
7 0	7 0

Před provedením	Po provedení
Příznak přenosu	Příznak přenosu
1	1
Akumulátor	Akumulátor
0 0 1 1 0 1 1 0	1 0 1 1 0 1 1 0
7 0	7 0

Poznámky: žádné

MOV

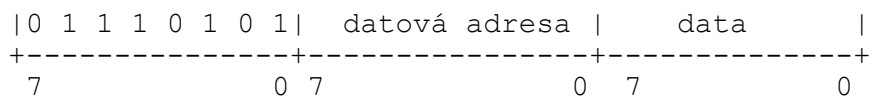
Přesun dat na datovou adresu

Mnemonika: MOV

Operandy: datová adresa 0 <= datová adresa <= 255
data -256 <= data <= +255

Zápis: MOV datová adresa,#data

Strojový kód:

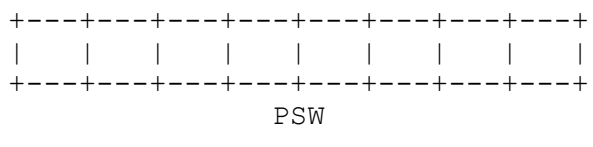


Činnost: (datová adresa) <- data

Délka: 3 byte

Doba trvání: 2 strojové cykly

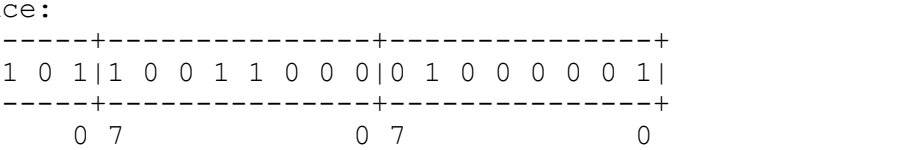
Ovlivňuje:



Popis: Tato instrukce provede přesun dat o délce 1 byte do paměti na adresu danou prvním operandem.

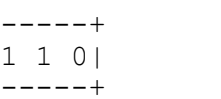
Příklad: MOV SBUF,#'A' ;znak 'A' do sériového bufferu

Kód instrukce:



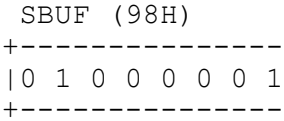
Před provedením

SBUF (98H)



Po provedení

SBUF (98H)



Poznámky: 4, 9, 18

MOV

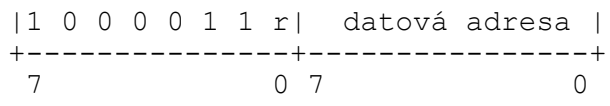
Přesun obsahu nepřímé adresy na datovou adresu

Mnemonika: MOV

Operandy: datová adresa 0 <= datová adresa <= 255
Rr Registr 0 <= r <= 1

Zápis: MOV datová adresa,@Rr

Strojový kód:

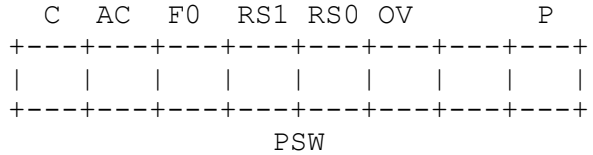


Činnost: (datová adresa) <- ((Rr))

Délka: 2 byte

Doba trvání: 2 strojové cykly

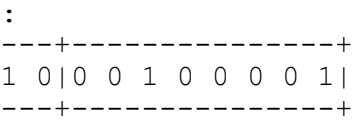
Ovlivňuje:



Popis: Tato instrukce provede přesun dat z paměti adresované nepřímo pomocí registru Rr do paměti na adresu danou operandem.

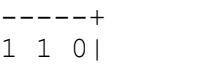
Příklad:
```
MOV 21H,@R0          ;přesun z nepřímé adresy na
                     ;adresu 21H
```

Kód instrukce:

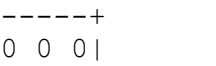


Před provedením

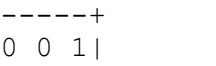
(21H)



Registr 0

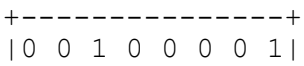


(20H)

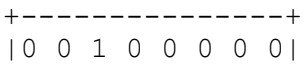


Po provedení

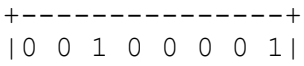
(21H)



Registr 0



(20H)



Poznámky: 9, 15

MOV

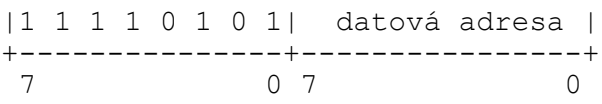
Přesun obsahu akumulátoru na datovou adresu

Mnemonika: MOV

Operandy: datová adresa 0 <= datová adresa <= 255
A Akumulátor

Zápis: MOV datová adresa,A

Strojový kód:

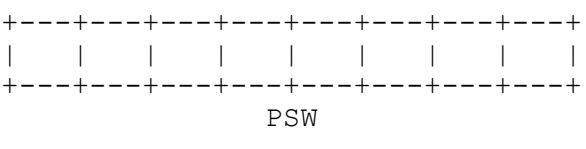


Činnost: (datová adresa) <- (A)

Délka: 2 byte

Doba trvání: 1 strojový cyklus

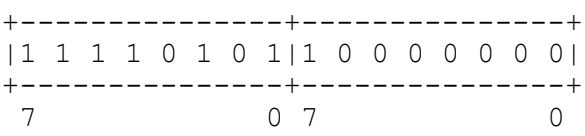
Ovlivňuje:



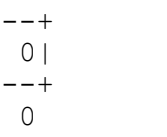
Popis: Tato instrukce provede přesun obsahu akumulátoru do paměti na adresu danou operandem.

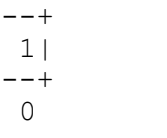
```
Příklad:      MOV P0,A            ;přesun obsahu  akumulátoru  na
                                  ;port 0
```

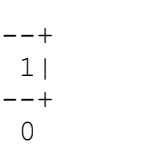
Kód instrukce:



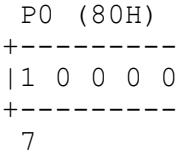
Před provedením

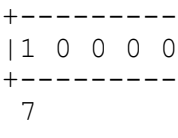


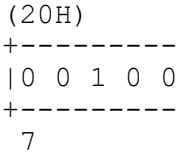




Po provedení







Poznámky: 9, 15

MOV

Přesun obsahu registru na datovou adresu

Mnemonika: MOV

Operandy: datová adresa 0 <= datová adresa <= 255
Rr Registr 0 <= r <= 7

Zápis: MOV datová adresa,Rr

Strojový kód:

1 0 0 0 1 r r r	datová adresa
7 0	7 0

Činnost: (datová adresa) <- ((Rr))

Délka: 2 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
PSW							

Popis: Tato instrukce provede přesun obsahu registru Rr do paměti na adresu danou operandem.

Příklad:
```
MOV 11H,R7          ;přesun  obsahu registru  R7 na
                    ;adresu 11H
```

Kód instrukce:

1 0 0 0 1 1 1 1	0 0 0 1 0 0 0 1
7 0	7 0

Před provedením

(11H)

0 0 0 0 0 0 0 0
7 0

Registr 7

0 0 1 0 0 0 0 1
7 0

Po provedení

(11H)

0 0 1 0 0 0 0 1
7 0

Registr 7

0 0 1 0 0 0 0 1
7 0

Poznámky: 9

MOV

Přesun obsahu datové adresy na datovou adresu

Mnemonika: MOV

Operandy: datová adresa 1 0 <= datová adresa <= 255
datová adresa 2 0 <= datová adresa <= 255

Zápis: MOV datová adresa 1,datová adresa 2

Strojový kód:

1 0 0 0 0 1 0 1	datová adresa 1	datová adresa 2
7 0	7 0	7 0

Činnost: (datová adresa 1) <- (datová adresa 2)

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce provede přesun obsahu datové adresy 2 (zdrojové adresy) do paměti na adresu danou datovou adresou 2 (cílovou adresou).

Příklad:
```
MOV P1,27H          ;obsah paměti na adresu  27H na
                    ;port 1
```

Kód instrukce:

1 0 0 0 0 1 0 1	0 0 1 0 0 1 1 1	1 0 0 1 0 0 0 0
7 0	7 0	7 0

Před provedením

P1 (90H)

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

(27H)

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

SBUF (98H)

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

Po provedení

P1 (90H)

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

(27H)

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

SBUF (98H)

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

Poznámky: 4, 9

MOVC

Přesun obsahu programové paměti adresované datovým ukazatelem do akumulátoru

Mnemonika: MOVC

Operandy:

A	Akumulátor
DPTR	Datový ukazatel

Zápis: MOVC A,@A+DPTR

Strojový kód:

1	0	0	1	0	0	1	1
7							0

Činnost: (A) <- ((A) + (DPTR))

Délka: 1 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce sečte obsah datového ukazatele s obsahem akumulátoru. Vzniklá šestnáctibitová adresa je zdrojovou adresou paměti programu, jejíž obsah se přesune do akumulátoru.

Příklad:

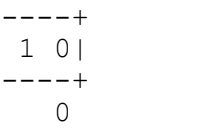
```
MOVC A,@A+DPTR      ;výběr tabulkové hodnoty
```

Kód instrukce:

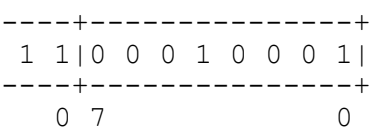
1	0	0	1	0	0	1	1
7							0

Před provedením

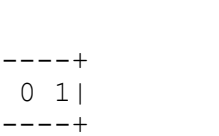
Akumulátor



Datový ukazatel

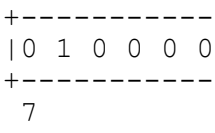


(8F17H)

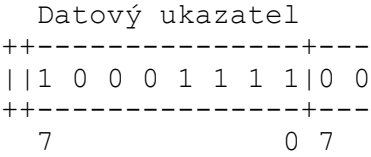


Po provedení

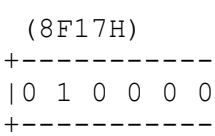
Akumulátor



Datový ukazatel



(8F17H)



Poznámky: 5

MOVC

Přesun obsahu programové paměti adresované programovým čítačem do akumulátoru

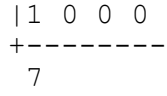
Mnemonika: MOVC

Operandy:

A	Akumulátor
PC	Programový čítač

Zápis: MOVC A,@A+PC

Strojový kód:



Činnost:
(PC) <- (PC) + 1
(A) <- ((A) + (PC))

Délka: 1 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

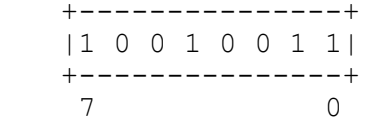
PSW

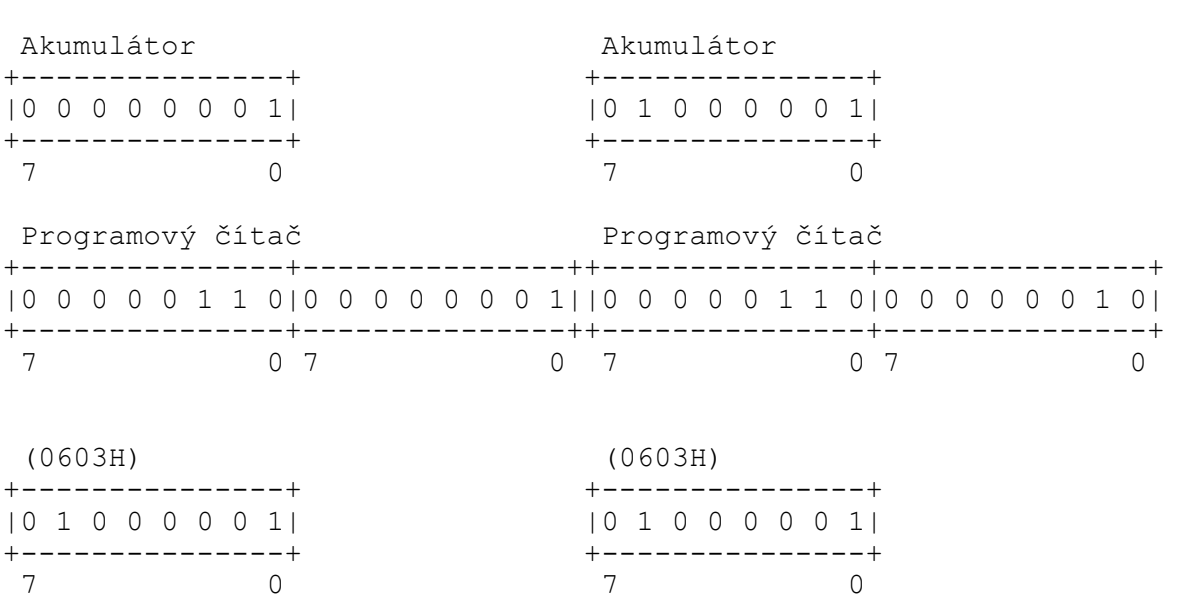
Popis: Tato instrukce zvětší obsah programového čítače o 1 a pak sečte obsah programového čítače s obsahem akumulátoru. Takto vzniklá šestnáctibitová adresa je zdrojovou adresou paměti programu, jejíž obsah se přesune do akumulátoru.

Příklad:

```
MOVC A,@A+PC        ;výběr tabulkové hodnoty
```

Kód instrukce:




Před provedením
Po provedení
Akumulátor
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 1
7 0
Programový čítač
0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 1 0
0 0 0 0 0 0 1 0
(0603H)
0 1 0 0 0 0 0 1

Poznámky: 5

MOVX

Přesun obsahu akumulátoru do externí datové paměti adresované datovým ukazatelem

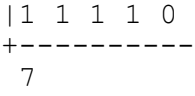
Mnemonika: MOVX

Operandy:

DPTR	Datový ukazatel
A	Akumulátor

Zápis: MOVX @DPTR,A

Strojový kód:



Činnost: ((DPTR)) <- (A)

Délka: 1 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

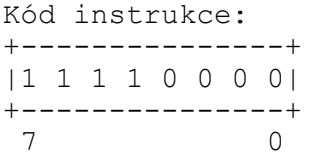
PSW

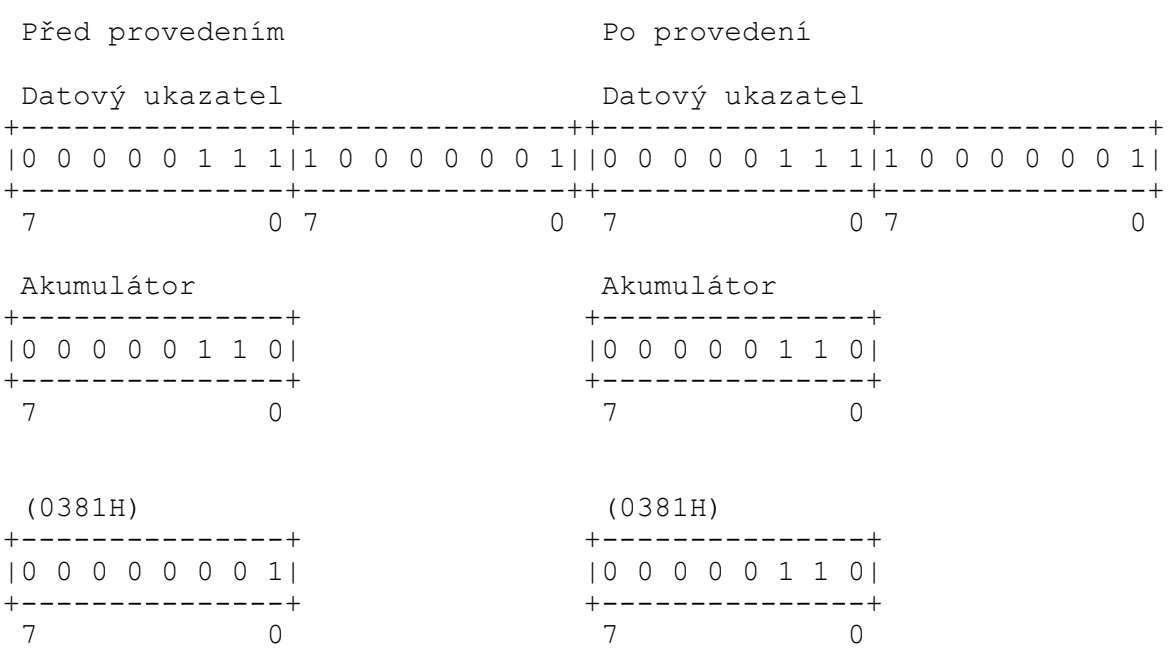
Popis: Tato instrukce použije obsah datového ukazatele jako šestnáctibitovou adresu v externí datové paměti, do níž přesune obsah akumulátoru.

Příklad:

```
MOVX @DPTR,A        ;ulož obsah akumulátoru
```

Kód instrukce:





Poznámky: žádná

MOVX

Přesun obsahu akumulátoru do externí datové paměti adresované registrem

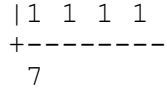
Mnemonika: MOVX

Operandy:

Rr	Registr 0 <= r <= 1
A	Akumulátor

Zápis: MOVX @Rr,A

Strojový kód:

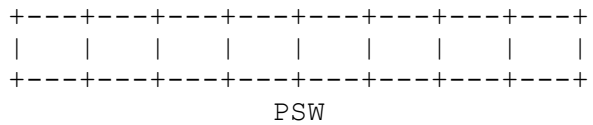


Činnost: ((Rr)) <- (A)

Délka: 1 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

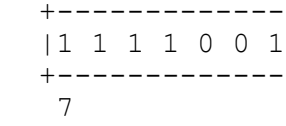


Popis: Tato instrukce použije obsah registru jako nižší byte šestnáctibitové adresy v externí datové paměti, do níž přesune obsah akumulátoru. Jako vyšší byte adresy se použije port P2, který je nutno před použitím instrukce nastavit.

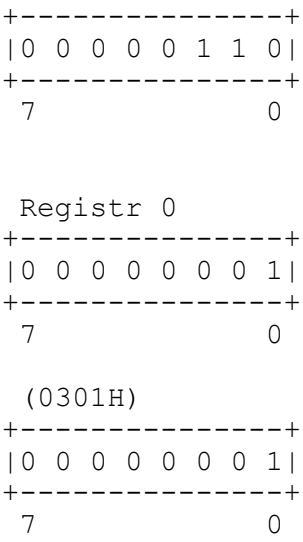
Příklad:

```
MOV  P2,30H
MOVX @R0,A          ;ulož obsah akumulátoru
```

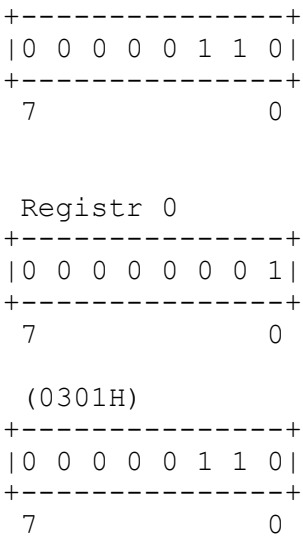
Kód instrukce:



Před provedením



Po provedení



Poznámky: žádná

MOVX

Přesun obsahu externí datové paměti adresované datovým ukazatelem do akumulátoru

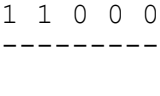
Mnemonika: MOVX

Operandy:

A	Akumulátor
DPTR	Datový ukazatel

Zápis: MOVX A,@DPTR

Strojový kód:



Činnost: (A) <- ((DPTR))

Délka: 1 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

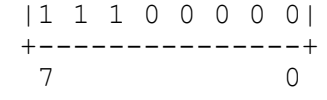
PSW

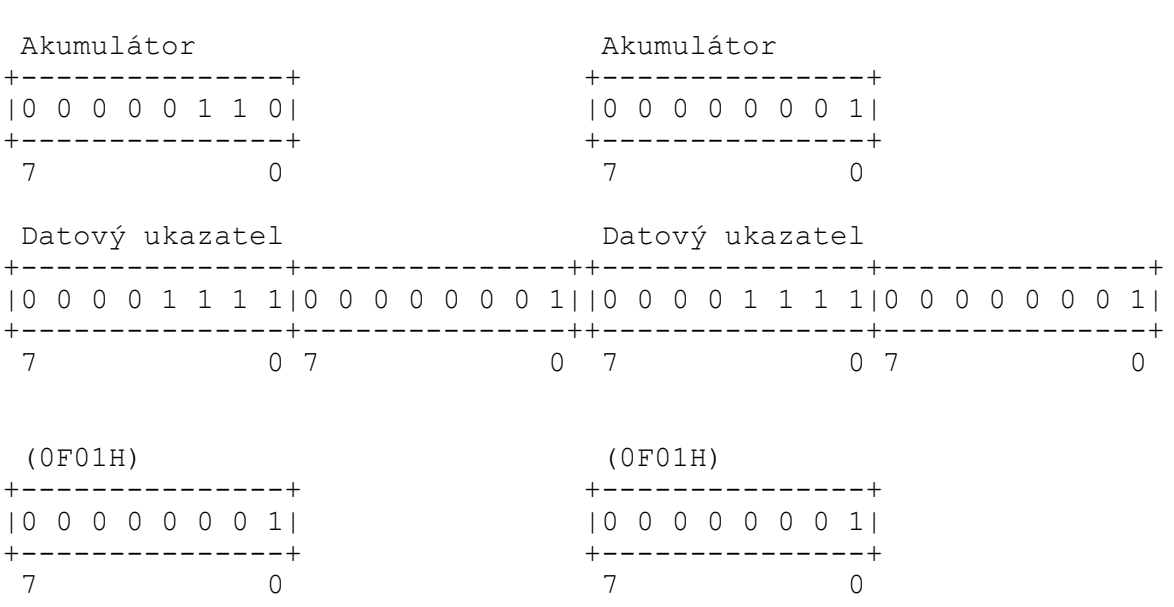
Popis: Tato instrukce použije obsah datového ukazatele jako šestnáctibitovou adresu v externí datové paměti, jejíž obsah se přesune do akumulátoru.

Příklad:

```
MOVX A,@DPTR        ;data do akumulátoru
```

Kód instrukce:




Před provedením
Po provedení
Akumulátor
0 0 0 0 0 1 1 0
Akumulátor
0 0 0 0 0 0 0 1
7 0
7 0
Datový ukazatel
0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 1
Datový ukazatel
0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 1
7 0 7 0
7 0 7 0
(0F01H)
0 0 0 0 0 0 0 1
(0F01H)
0 0 0 0 0 0 0 1
7 0
7 0

Poznámky: 5

MOVX

Přesun obsahu externí datové paměti adresované registrem do akumulátoru

Mnemonika: MOVX

Operandy:

A		Akumulátor
Rr		Registr 0 <= r <= 1

Zápis: MOVX A,@Rr

Strojový kód:

1	1	1	0	0	0	1	r
7							0

Činnost: (A) <- ((Rr))

Délka: 1 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce použije obsah registru jako nižší byte šestnáctibitové adresy v externí datové paměti, jejíž obsah přesune do akumulátoru. Jako vyšší byte adresy se použije port P2, který je nutno před použitím instrukce nastavit.

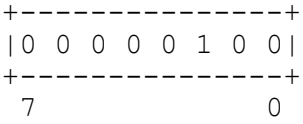
Příklad:

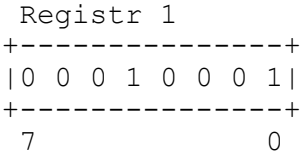
```
MOV  P2,88H
MOVX A,@R1          ;data do akumulátoru
```

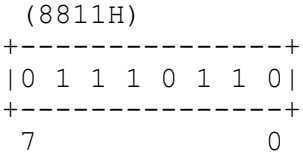
Kód instrukce:

1	1	1	0	0	0	1	1
7							0

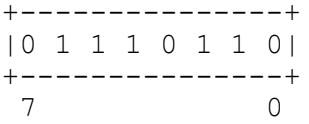
Před provedením

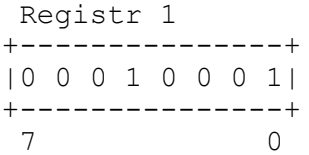






Po provedení





Poznámky: 5

MUL

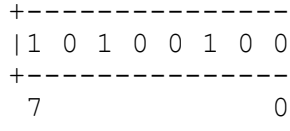
Násobení obsahu akumulátoru obsahem registru B

Mnemonika: MUL

Operandy: AB dvojice registrů

Zápis: MUL AB

Strojový kód:



Činnost: (AB) <- (A) * (B)

Délka: 1 byte

Doba trvání: 4 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x					x		x

PSW

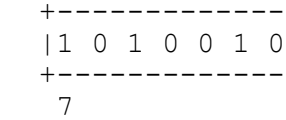
Popis: Tato instrukce násobí obsah akumulátoru obsahem registru B. Oba operandy jsou pokládány za celá čísla bez znaménka. Po násobení obsahují akumulátor nižší a registr B vyšší byte výsledku.

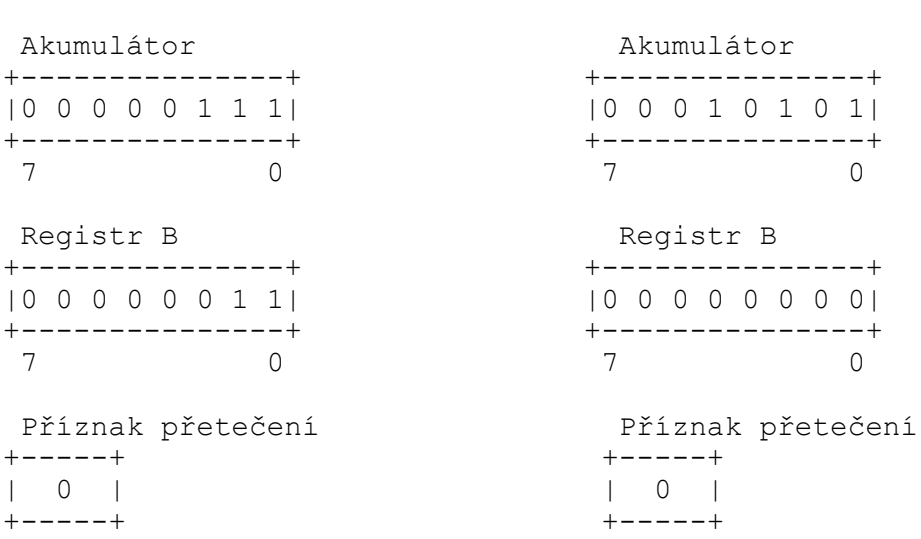
Příznak přenosu se vždy vynuluje. Pokud je vyšší byte výsledku nenulový, příznak přetečení se nastaví na logickou hodnotu 1, jinak má příznak přetečení logickou hodnotu 0.

Příklad:

```
MOV  B,3          ;naplnění registru B hodnotou 3
MUL  AB            ;násobení obsahu akumulátoru
                   ;třemi
```

Kód instrukce:




Před provedením
Akumulátor
0 0 0 0 0 1 1 1
7 0
Registr B
0 0 0 0 0 0 1 1
7 0
Příznak přetečení
0
Po provedení
Akumulátor
0 0 0 1 0 1 0 1
7 0
Registr B
0 0 0 0 0 0 0 0
7 0
Příznak přetečení
0

Poznámky: 5

NOP

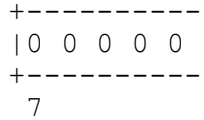
Žádná činnost

Mnemonika: NOP

Operandy: žádné

Zápis: NOP

Strojový kód:

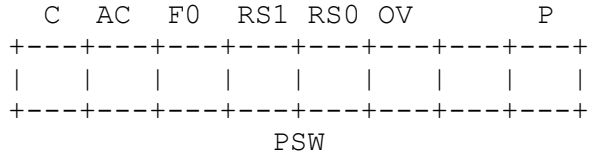


Činnost: žádná

Délka: 1 byte

Doba trvání: 1 strojový cyklus

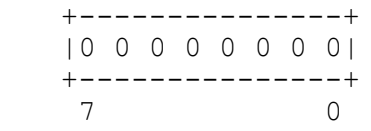
Ovlivňuje:



Popis: Tato instrukce nezpůsobí žádnou činnost.

Příklad:
```
NOP                 ;prodleva jeden strojový cyklus
```

Kód instrukce:



Poznámky: žádná

ORL

Logický součet dat s obsahem akumulátoru

Mnemonika: ORL

Operandy: A akumulátor
data -256 <= data <= +255

Zápis: ORL A,#data

Strojový kód:

0 1 0 0 0 1 0 0	data
7 0	7 0

Činnost: (A) <- (A) OR data

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce provede logický součet hodnoty 8-bitových dat s obsahem akumulátoru. Bit n výsledku bude nastaven na hodnotu 1, bude-li bit n alespoň jednoho operandu roven 1. Jinak je bit n výsledku vynulován. Výsledek je umístěn v akumulátoru.

Příklad: ORL A,#00001000B ;maska nastaví bit 3 na "1"

Kód instrukce:

0 1 0 0 0 1 0 0	0 0 0 0 1 0 0 0
7 0	7 0

Před provedením

Akumulátor

0 1 1 1 0 1 1 1
7 0

Po provedení

Akumulátor

0 1 1 1 1 1 1 1
7 0

Poznámky: 4, 5

ORL

Logický součet obsahu nepřímé adresy s obsahem akumulátoru

Mnemonika: ORL

Operandy:
A	akumulátor
Rr	registr r ={ 1, 2}

Zápis: ORL A,@Rr

Strojový kód:

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

Činnost: (A) <- (A) OR ((Rr))

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

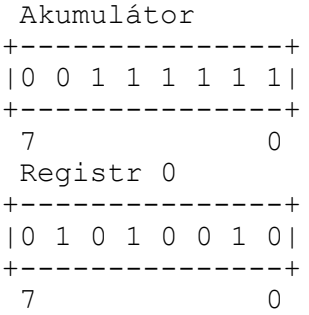
Popis: Tato instrukce provede logický součet obsahu paměťového místa na adrese určené registrem r s obsahem akumulátoru. Bit n výsledku bude nastaven na hodnotu 1, bude-li bit n alespoň jednoho operandu roven 1. Jinak je bit n výsledku vynulován. Výsledek je umístěn v akumulátoru.

Příklad:
```
ORL   A,@R0         ;logický součet  obsahu nepřímé
                    ;adresy a obsahu akumulátoru
```

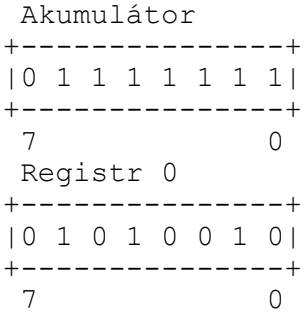
Kód instrukce:

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

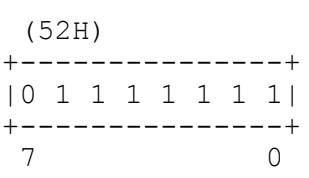
Před provedením



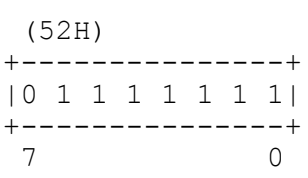
Po provedení



(52H)



(52H)



Poznámky: 5, 15

ORL

Logický součet obsahu registru s obsahem akumulátoru

Mnemonika: ORL

Operandy: A akumulátor
Rr registr 0 <= r <= 7

Zápis: ORL A,Rr

Strojový kód:

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

Činnost: (A) <- (A) OR (Rr)

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce provede logický součet obsahu registru r s obsahem akumulátoru. Bit n výsledku bude nastaven na hodnotu 1, bude-li bit n alespoň jednoho operandu roven 1. Jinak je bit n výsledku roven 0. Výsledek je umístěn v akumulátoru.

Příklad:
```
MOV    R4,10000000B  ;přesun masky do registru 4
ORL    A,R4          ;logický součet R4 a akumulá-
                     ;toru
```

Kód instrukce:

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

Před provedením

Akumulátor

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

Registr 4

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

Po provedení

Akumulátor

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

Registr 4

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

Poznámky: 5

ORL

Logický součet obsahu datové adresy s obsahem akumulátoru

Mnemonika: ORL

Operandy: A akumulátor
datová adresa 0 <= datová adresa <= 255

Zápis: ORL A,datová adresa

Strojový kód:

0 1 0 0 0 1 0 1	datová adresa
7 0	7 0

Činnost: (A) <- (A) OR (datová adresa)

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce provede logický součet obsahu datové adresy s obsahem akumulátoru. Bit n výsledku bude nastaven na hodnotu 1, bude-li bit n alespoň jednoho operandu roven 1. Jinak je bit n výsledku roven 0. Výsledek je umístěn v akumulátoru.

Příklad:
```
ORL A,37H          ;logický součet obsahu na adrese
                   ;37H a akumulátoru
```
Kód instrukce:

0 1 0 0 0 1 0 1	0 0 1 1 0 1 1 1
7 0	7 0

Před provedením

Akumulátor

0 1 1 1 0 0 0 0
7 0

(37H)

0 0 0 0 0 1 1 1
7 0

Po provedení

Akumulátor

0 1 1 1 0 1 1 1
7 0

(37H)

0 0 0 0 0 1 1 1
7 0

Poznámky: 5, 8

ORL

Logický součet obsahu bitové adresy s příznakem přenosu

Mnemonika: ORL

Operandy: C příznak přenosu
adresa bitu 0 <= adresa bitu <= 255

Zápis: ORL C,adresa bitu

Strojový kód:

0 1 1 1 0 0 1 0	adresa bitu
7 0	7 0

Činnost: (C) <- (C) OR (adresa bitu)

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x							

PSW

Popis: Tato instrukce provede logický součet hodnoty příznaku přenosu s hodnotou bitu určeného bitovou adresou. Má-li jeden z bitů logickou hodnotu 1, je výsledkem logická 1. Jinak je výsledkem logická 0. Výsledek je umístěn do bitu příznaku přenosu.

Příklad:
```
ORL C,25H.3       ;logický součet bitu 3 ve slabice
                  ;na adrese 25H a příznaku přenosu
```
Kód instrukce:

0 1 1 1 0 0 1 0	0 0 1 0 1 0 1 1
7 0	7 0

Před provedením

Příznak přenosu

1

(25H)

0 0 1 0 1 1 1 0
7 0

Po provedení

Příznak přenosu

1

(25H)

0 0 1 0 1 1 1 0
7 0

Poznámky: žádné

ORL

Logický součet negace obsahu bitové adresy s příznakem přenosu

Mnemonika: ORL

Operandy: C příznak přenosu
adresa bitu 0 <= adresa bitu <= 255

Zápis: ORL C,/adresa bitu

Strojový kód:

1 0 1 0 0 0 0 0	adresa bitu
7 0	7 0

Činnost: (C) <- (C) OR NOT (adresa bitu)

Délka: 2 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x							

PSW

Popis: Tato instrukce provede logický součet hodnoty příznaku přenosu s negací hodnoty bitu určeného bitovou adresou. Výsledek bude mít logickou hodnotu 1, bude-li mít buď příznak přenosu nebo adresovaný bit logickou hodnotu 1. Jinak je výsledkem logická 0. Výsledek je umístěn do bitu příznaku přenosu. Hodnota bitu adresovaného bitovou adresou se nemění. Výsledek je umístěn v příznaku přenosu.

Příklad:
```
ORL C,/28H.5       ;logický součet negace bitu 5 ve
                   ;slabice na adrese 28H a přízna-
                   ;ku přenosu
```

Kód instrukce:

1 0 1 0 0 0 0 0	0 1 0 0 0 1 0 1
7 0	7 0

Před provedením

Příznak přenosu

1

(25H)

0 1 0 1 1 0 0 0
7 0

Po provedení

Příznak přenosu

1

(25H)

0 1 0 1 1 0 0 0
7 0

Poznámky: žádné

ORL

Logický součet dat s obsahem datové adresy

Mnemonika: ORL

Operandy: datová adresa 0 <= datová adresa <= 255
data -256 <= data <= +255

Zápis: ORL datová adresa,#data

Strojový kód:

0 1 0 0 0 1 1	datová adresa	data
7 0	7 0	7 0

Činnost: (datová adresa) <- (datová adresa) OR data

Délka: 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce provede logický součet osmibitových dat s obsahem určené datové adresy. Bit n výsledku bude nastaven na hodnotu 1, bude-li bit n alespoň jednoho operandu roven 1. Jinak je bit n výsledku roven 0. Výsledek se umístí do paměti na určenou datovou adresu.

Příklad:
```
MOV   57H,PSW       ;přesun PSW na adresu 57H
ORL   57H,#01H      ;maska pro zachování příznaku P
```

Kód instrukce:

0 1 0 0 0 1 1	0 1 0 1 0 1 1 1	0 0 0 0 0 0 0 1
7 0	7 0	7 0

Před provedením

(57H)

0 0 0 0 0 0 0 0
7 0

Po provedení

(57H)

0 0 0 0 0 0 0 1
7 0

Poznámky: 4,7

ORL

Logický součet obsahu datové adresy s obsahem akumulátoru

Mnemonika: ORL

Operandy: datová adresa 0 <= datová adresa <= 255
A akumulátor

Zápis: ORL datová adresa,A

Strojový kód:

0 1 0 0 0 0 1 0	datová adresa
7 0	7 0

Činnost: (datová adresa) <- (datová adresa) OR (A)

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce provede logický součet obsahu určené datové adresy s obsahem akumulátoru. Bit n výsledku bude nastaven na hodnotu 1, bude-li bit n alespoň jednoho operandu roven 1. Jinak je bit n výsledku roven 0. Výsledek se umístí do paměti na určenou datovou adresu.

Příklad:
```
MOV   A,10000001B  ;uložení masky do akumulátoru
ORL   10H,A        ;maska pro zachování bitů 0 a 7
```

Kód instrukce:

0 1 0 0 0 0 1 0	0 0 0 1 0 0 0 0
7 0	7 0

Před provedením
Akumulátor

1 0 0 0 0 0 0 1
7 0

(10H)

0 0 1 1 0 0 0 1
7 0

Po provedení
Akumulátor

1 0 0 0 0 0 0 1
7 0

(10H)

1 0 1 1 0 0 0 1
7 0

Poznámky: 9

POP

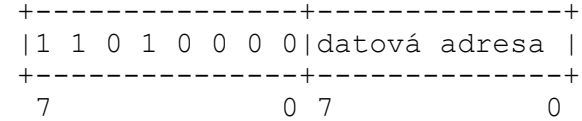
Obnovení obsahu datové adresy ze zásobníku

Mnemonika: POP

Operandy: datová adresa 0 <= datová adresa <= 255

Zápis: POP datová adresa

Strojový kód:

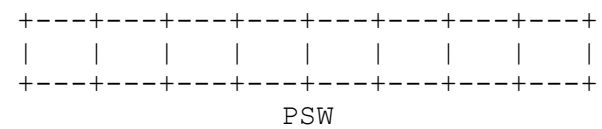


Činnost: (datová adresa) <- ((SP))
(SP) <- (SP) -1

Délka: 2 byte

Doba trvání: 2 strojové cykly

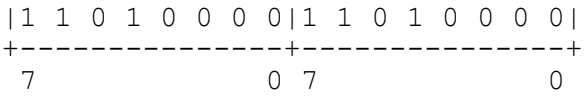
Ovlivňuje:

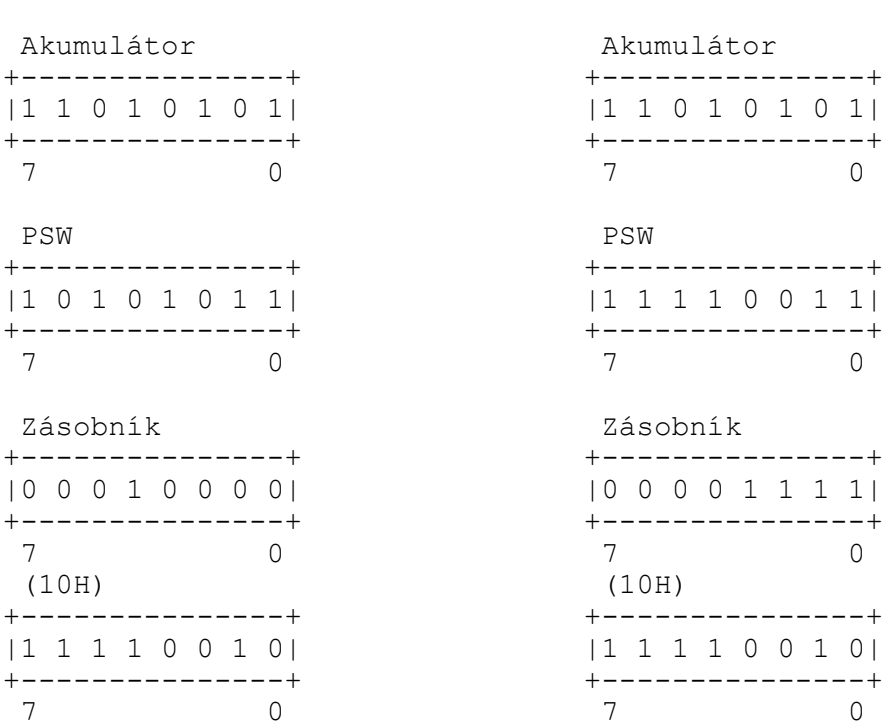


Popis: Tato instrukce provede přesun obsahu slabiky určené ukazatelem zásobníku na určenou datovou adresu. Obsah ukazatele zásobníku se zmenší o 1.

Příklad: POP PSW

Kód instrukce:





Poznámky: 2, 8, 17

PUSH

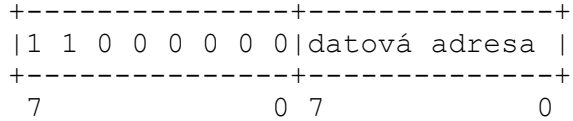
Uložení obsahu datové adresy do zásobníku

Mnemonika: PUSH

Operandy: datová adresa 0 <= datová adresa <= 255

Zápis: PUSH datová adresa

Strojový kód:

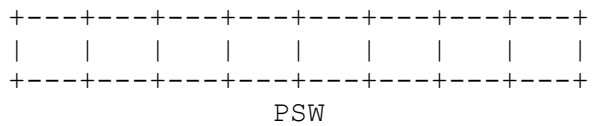


Činnost: (SP) <- (SP) + 1
(SP) <- (datová adresa)

Délka: 2 byte

Doba trvání: 2 strojové cykly

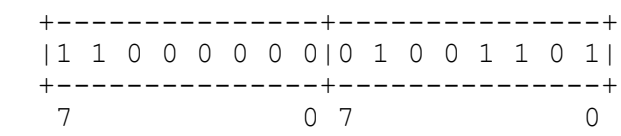
Ovlivňuje:



Popis: Tato instrukce zvýší o 1 obsah ukazatele zásobníku a provede přesun obsahu určené datové adresy na adresu, která je v ukazateli zásobníku.

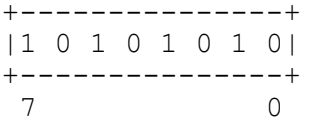
```
Příklad:      PUSH  4DH           ;uloží do zásobníku obsah pamě-
                                  ;ťového místa na adrese 4DH
```

Kód instrukce:

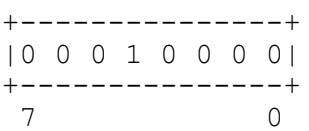


Před provedením

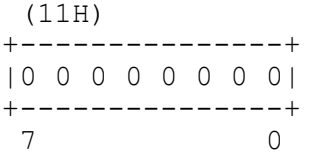
(4DH)



Zásobník

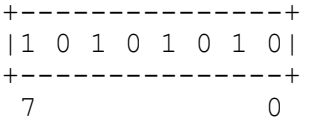


(11H)

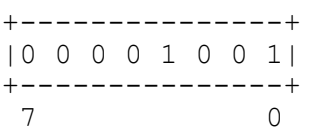


Po provedení

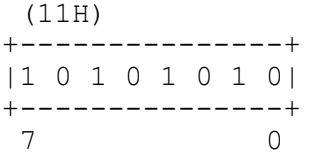
(4DH)



Zásobník



(11H)



Poznámky: 2, 3, 8

RET

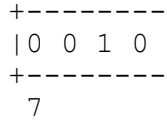
Návrat z podprogramu (ne z obsluhy přerušení)

Mnemonika: RET

Operandy: žádné

Zápis: RET

Strojový kód:



Činnost:
```
(PC HIGH)  <- ((SP))
(SP)  <-  (SP) - 1
(PC LOW)  <-  ((SP))
(SP)  <-  (SP) - 1
```

Délka: 1 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

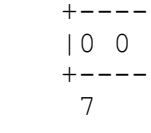
C	AC	F0	RS1	RS0	OV		P

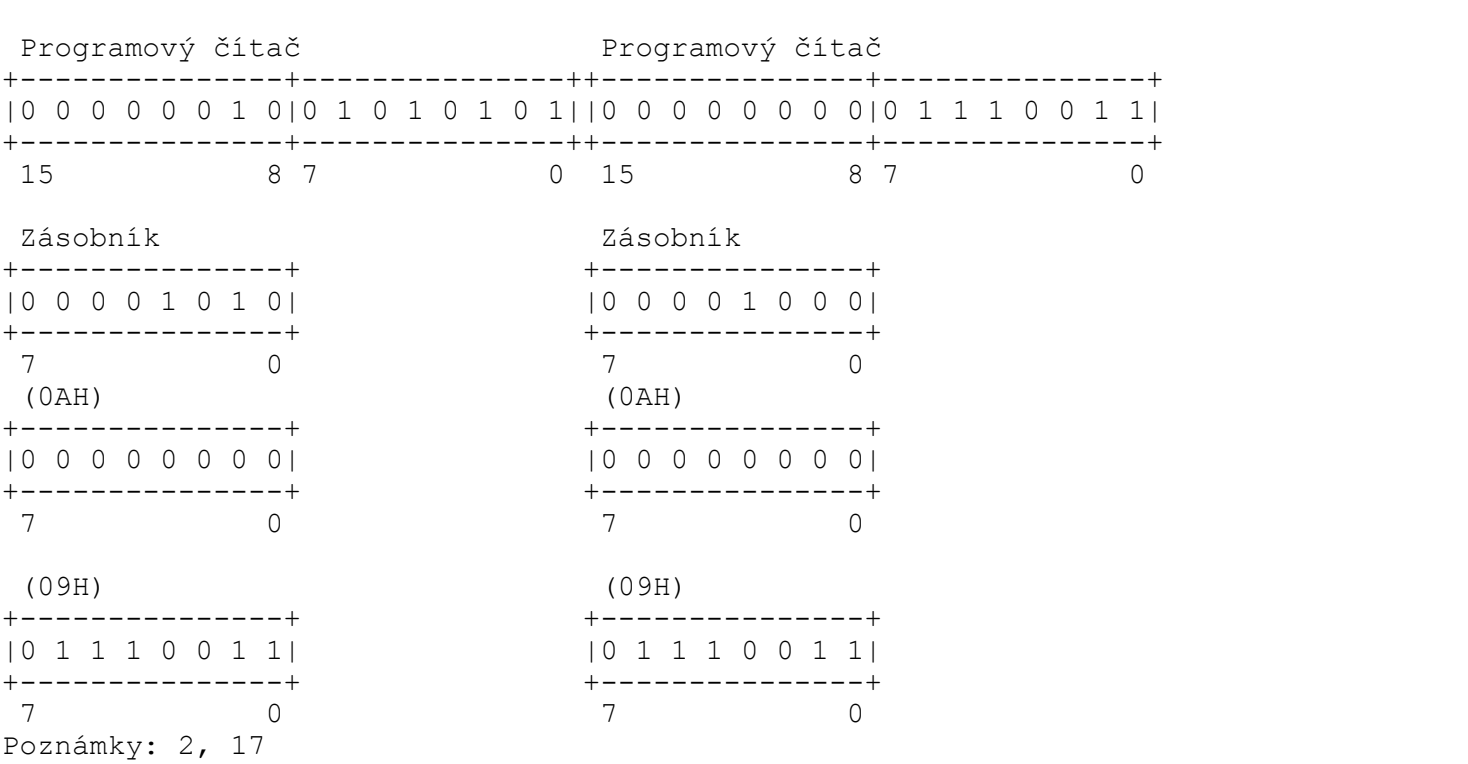
PSW

Popis: Tato instrukce provede návrat z podprogramu. Řízení je převedeno na adresu uloženou v zásobníku. Vyšší řády návratové adresy se vyberou jako první, nižší řády návratové adresy jako druhé. Obsah ukazatele zásobníku se sníží o 2.

Příklad:
```
RET                 ;návrat z podprogramu
```

Kód instrukce:





Poznámky: 2, 17

RETI

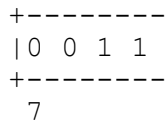
Návrat z podprogramu pro obsluhu přerušení

Mnemonika: RETI

Operandy: žádné

Zápis: RETI

Strojový kód:



Činnost:
```
(PC HIGH)  <- ((SP))
(SP)  <-  (SP) - 1
(PC LOW)  <-  ((SP))
(SP)  <-  (SP) - 1
```

Délka: 1 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

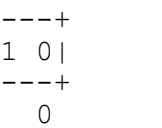
C	AC	F0	RS1	RS0	OV		P

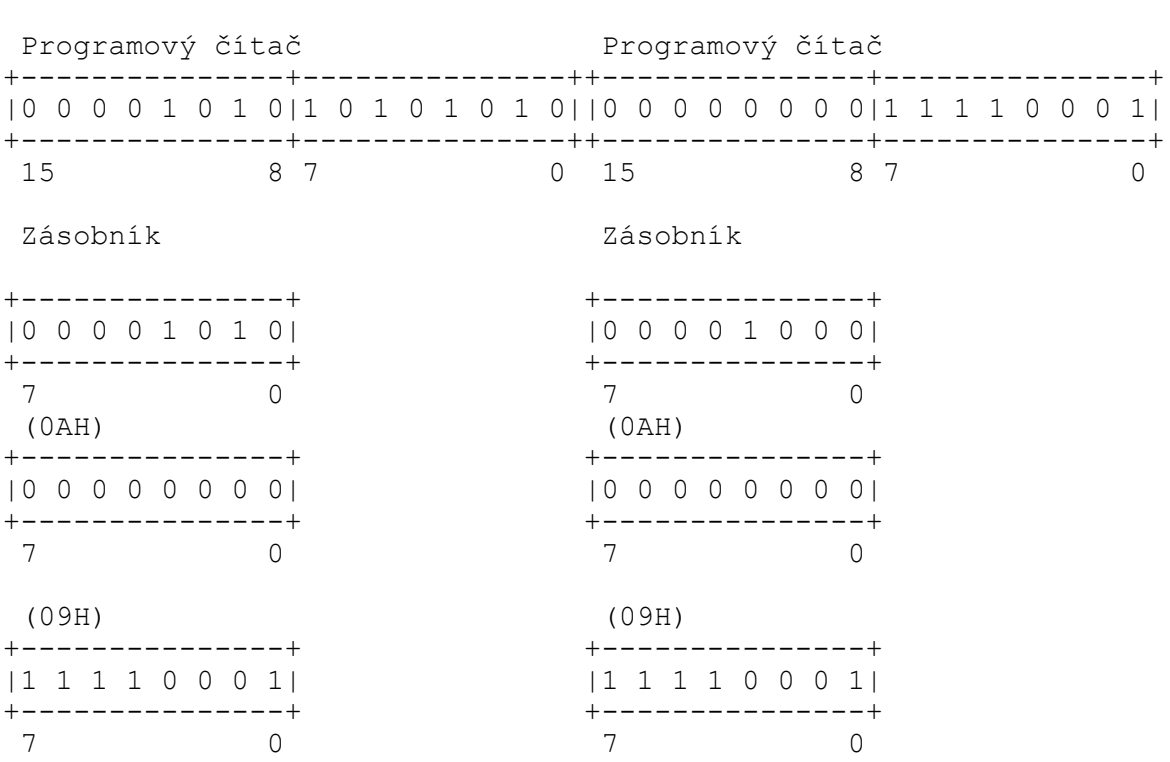
PSW

Popis: Tato instrukce provede návrat z podprogramu pro ošetření přerušení a znovu povolí přerušení se stejnou nebo nižší prioritou. Řízení přechází na adresu, která je uložena v zásobníku. Vyšší řády návratové adresy se vyberou jako první, nižší řády návratové adresy jako druhé. Obsah ukazatele zásobníku se sníží o 2. Stavové slovo programu se automaticky neobnoví.

Příklad:
```
RETI            ;návrat z podprogramu pro obsluhu
                ;přerušení
```

Kód instrukce:





Poznámky: 2, 17

RL

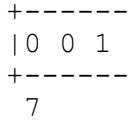
Rotační posuv obsahu akumulátoru vlevo

Mnemonika: RL

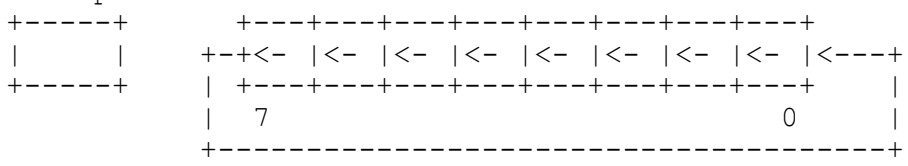
Operandy: A Akumulátor

Zápis: RL A

Strojový kód:



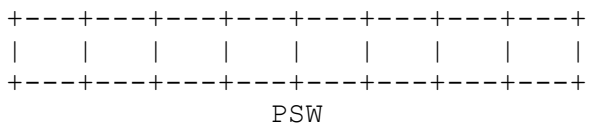
Činnost:



Délka: 1 byte

Doba trvání: 1 strojový cyklus

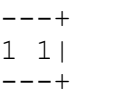
Ovlivňuje:



Popis: Tato instrukce posune každý bit akumulátoru o jednu pozici doleva. Nejvýznamnější bit (bit 7) se přesune do nejméně významného bitu (bit 0).

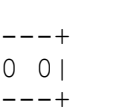
```
Příklad:     RL  A            ;posun obsahu akumulátoru o 1 bit
                              ;doleva
```

Kód instrukce:



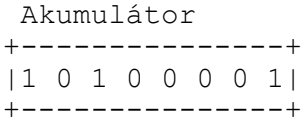
Před provedením

Akumulátor



Po provedení

Akumulátor



Poznámky: žádné

RLC

Rotační posuv obsahu akumulátoru a příznaku přenosu vlevo

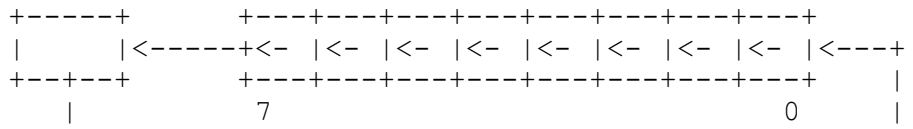
Mnemonika: RLC

Operandy: A Akumulátor

Zápis: RLC A

Strojový kód:

Činnost: Příznak přenosu Akumulátor



Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x							x

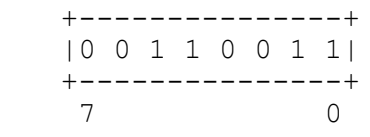
PSW

Popis: Tato instrukce posune každý bit akumulátoru o jednu pozici doleva. Nejvýznamnější bit (bit 7) se přesune do příznaku přenosu a hodnota příznaku přenosu se přesune do nejméně významného bitu (bit 0).

Příklad:

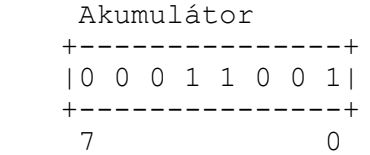
```
RLC     A             ;ACC=ACC*2, CY=přenos
```

Kód instrukce:

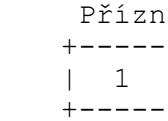


Před provedením

Akumulátor

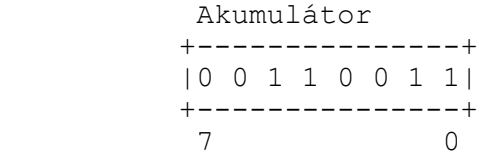


Příznak přenosu



Po provedení

Akumulátor



Příznak přenosu

Poznámky: 5

RR

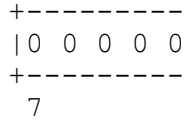
Rotační posuv obsahu akumulátoru vpravo

Mnemonika: RR

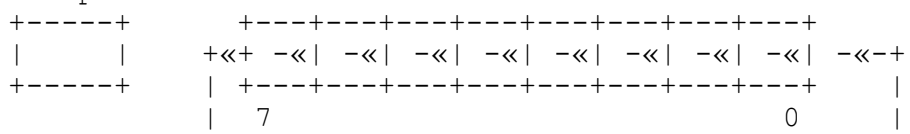
Operandy: A Akumulátor

Zápis: RR A

Strojový kód:



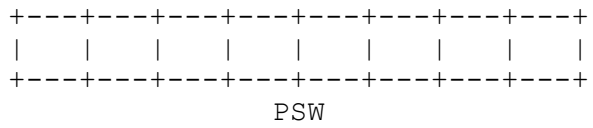
Činnost:



Délka: 1 byte

Doba trvání: 1 strojový cyklus

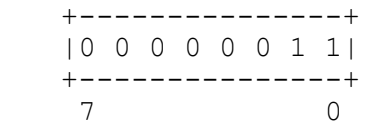
Ovlivňuje:



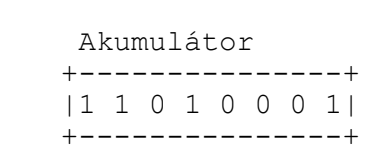
Popis: Tato instrukce posune každý bit akumulátoru o jednu pozici vpravo. Nejméně významný bit (bit 0) se přesune do nejvýznamnějšího bitu (bit 7).

```
Příklad:     RR  A            ;posun obsahu akumulátoru o 1 bit
                              ;doprava
```

Kód instrukce:



Před provedením



Po provedení

Poznámky: žádné

RRC

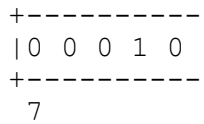
Rotační posuv obsahu akumulátoru a příznaku přenosu vpravo

Mnemonika: RRC

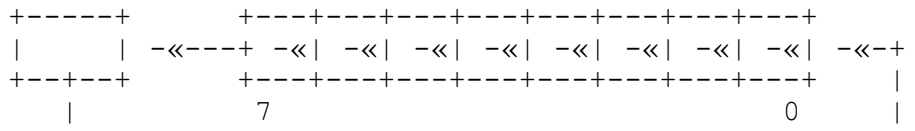
Operandy: A Akumulátor

Zápis: RRC A

Strojový kód:



Činnost: Příznak přenosu Akumulátor



Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

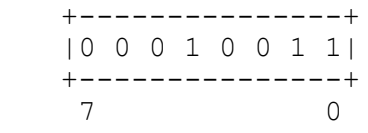
C	AC	F0	RS1	RS0	OV		P
x							x

PSW

Popis: Tato instrukce posune každý bit akumulátoru o jednu pozici vpravo. Nejméně významný bit 0 se přesune do příznaku přenosu a hodnota příznaku přenosu se přesune do nejvýznamnějšího bitu 7.

```
Příklad:      RRC    A        ;posun obsahu akumulátoru a obsahu
                              ;příznaku přenosu o 1 bit vpravo
```

Kód instrukce:



Před provedením		Po provedení	
Akumulátor		Akumulátor	
1 0 0 1 1 0 0 0		1 1 0 0 1 1 0 0	
Příznak přenosu		Příznak přenosu	
1		0	

Poznámky: 5

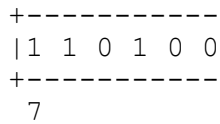
SETB

Nastavení příznaku přenosu

Mnemonika: SETB

Operandy: C Příznak přenosu

Zápis: SETB C
:
Strojový kód:



Činnost: (C) <- 1

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

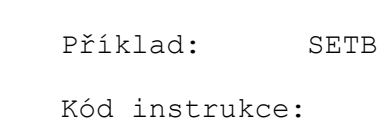
C	AC	F0	RS1	RS0	OV		P
x							

PSW

Popis: Tato instrukce nastavuje příznak přenosu do logické hodnoty 1.

Příklad: SETB C ;nastavení příznaku přenosu do 1

Kód instrukce:



Před provedením	Po provedení
Příznak přenosu	Příznak přenosu
0	1

Poznámky: žádné

SETB

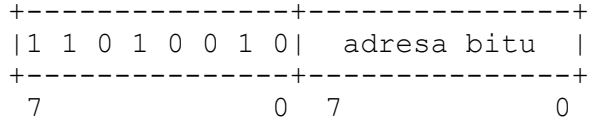
Nastavení bitu

Mnemonika: SETB

Operandy: adresa bitu 0 <= adresa bitu <= 255

Zápis: SETB adresa bitu

Strojový kód:

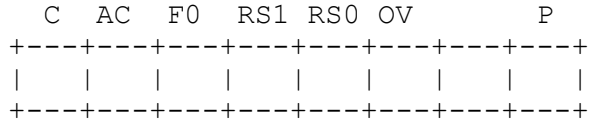


Činnost: (adresa bitu) <- 1

Délka: 2 byte

Doba trvání: 1 strojový cyklus

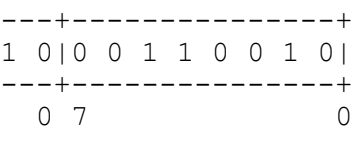
Ovlivňuje:



Popis: Tato instrukce nastavuje bit určený adresou bitu do logické hodnoty 1.

Příklad: SETB 38.2 ;nastavení bitu 2 na adrese 38 do 1

Kód instrukce:



Před provedením (38)

Po provedení (38)

Poznámky: 18

SJMP

Krátký skok

Mnemonika: SJMP

Operandy: adresa v paměti programu

Zápis: SJMP <adr>

Strojový kód:

1 0 0 0 0 0 0 0	rel.offset
7 0	7 0

Činnost:
(PC) <- (PC) + 2
(PC) <- (PC) + rel.offset

Délka: 2 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce předává řízení na určenou adresu v paměti programu. Programový čítač je zvětšen o 2 a ukazuje na následující instrukci. K obsahu zvětšeného programového čítače se přičte relativní posun (rel. offset) a provede se instrukce uložená na takto vzniklé adrese.

Příklad:
```
        SJMP   SKOK     ;skok na návěští SKOK
        INC    A
           .
           .
SKOK:   RR     A        ;15 byte od instrukce INC
```

Kód instrukce:

1 0 0 0 0 0 0 0	0 0 0 0 1 1 1 1
7 0	7 0

Před provedením

Programový čítač

1 1 1 0 1 0 0 0	1 1 0 1 1 1 0 0
15 8	7 0

Po provedení

Programový čítač

1 1 1 0 1 0 0 0	1 1 1 0 1 1 0 1
15 8	7 0

Poznámky: 10, 11, 12

SUBB

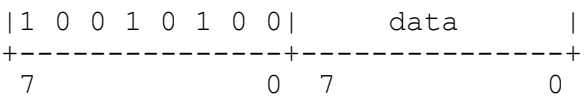
Odečtení dat od obsahu akumulátoru s výpůjčkou

Mnemonika: SUBB

Operandy:
A	Akumulátor
data	-256 <= data <= +255

Zápis: SUBB A,#data

Strojový kód:



Činnost: (A) <- (A) - (C) - data

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

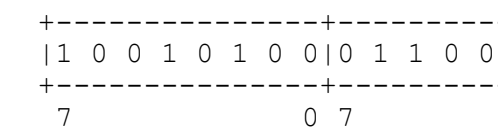
C	AC	F0	RS1	RS0	OV		P
x	x				x		x

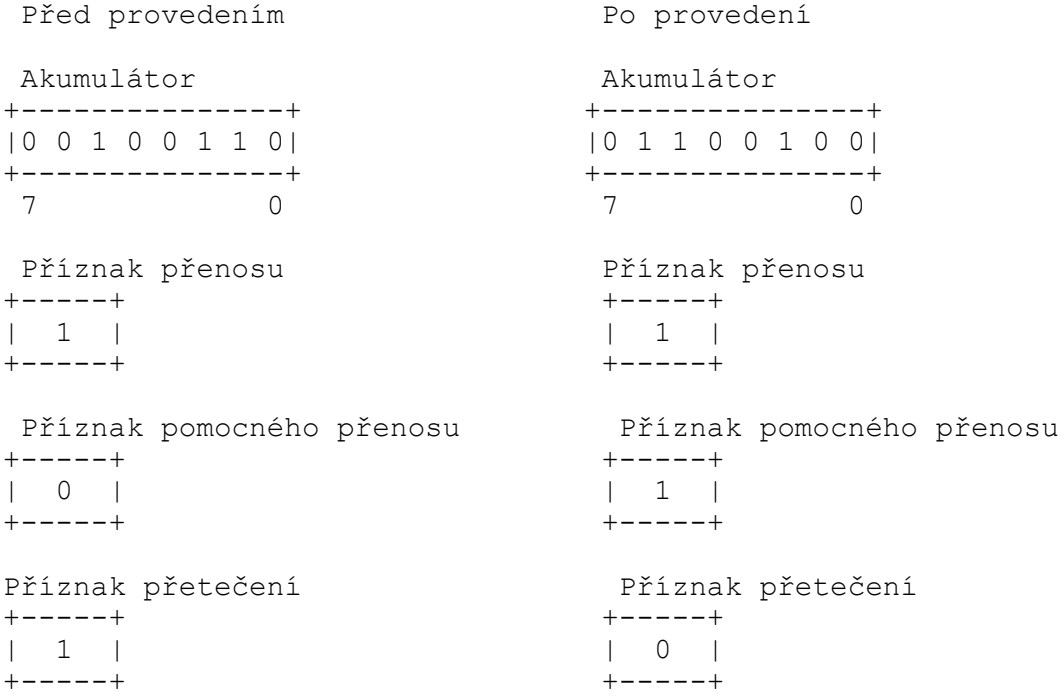
PSW

Popis: Tato instrukce odečte od obsahu akumulátoru hodnotu příznaku přenosu a hodnotu dat. Výsledek je uložen v akumulátoru.

Příklad:
```
SUBB   A,#0C1H    ;odečtení hodnoty 0C1H a příznaku
                  ;přenosu od akumulátoru
```

Kód instrukce:





Poznámky: 4, 5, 6, 13, 14

SUBB

Odečtení obsahu nepřímé adresy od obsahu akumulátoru s výpůjčkou

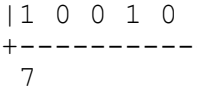
Mnemonika: SUBB

Operandy:

A	Akumulátor
Rr	Registr 0 <= r <= 1

Zápis: SUBB A,@Rr

Strojový kód:



Činnost: (A) <- (A) - (C) - ((Rr))

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

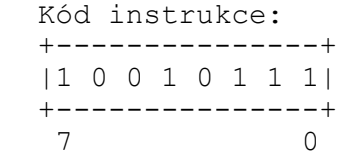
C	AC	F0	RS1	RS0	OV		P
x	x				x		x

PSW

Popis: Tato instrukce odečte od obsahu akumulátoru hodnotu příznaku přenosu a obsah paměťového místa na adrese určené obsahem registru r. Výsledek je uložen v akumulátoru.

```
Příklad:      SUBB   A,@R1      ;odečtení příznaku přenosu a hod-
                                ;noty na  adrese  určené  obsahem
                                ;registru r od obsahu akumulátoru
```

Kód instrukce:



Před provedením

Akumulátor

1	0	0	0	0	1	1	0
7							0

Registr 1

0	0	0	1	1	1	0	0
7							0

(1CH))

0	1	1	0	0	0	1	0
7							0

Příznak přenosu

0

Příznak pomocného přenosu

0

Příznak přetečení

0

Po provedení

Akumulátor

0	0	1	0	0	1	0	0
7							0

Registr 1

0	0	0	1	1	1	0	0
7							0

(1CH)

0	1	1	0	0	0	1	0
7							0

Příznak přenosu

0

Příznak pomocného přenosu

1

Příznak přetečení

1

Poznámky: 5, 6, 13, 14, 15

SUBB

Odečtení obsahu registru od obsahu akumulátoru s výpůjčkou

Mnemonika: SUBB

Operandy:

A	Akumulátor
Rr	Registr 0 <= r <= 7

Zápis: SUBB A,Rr

Strojový kód:

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

Činnost: (A) <- (A) - (C) - (Rr)

Délka: 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
x	x				x		x

PSW

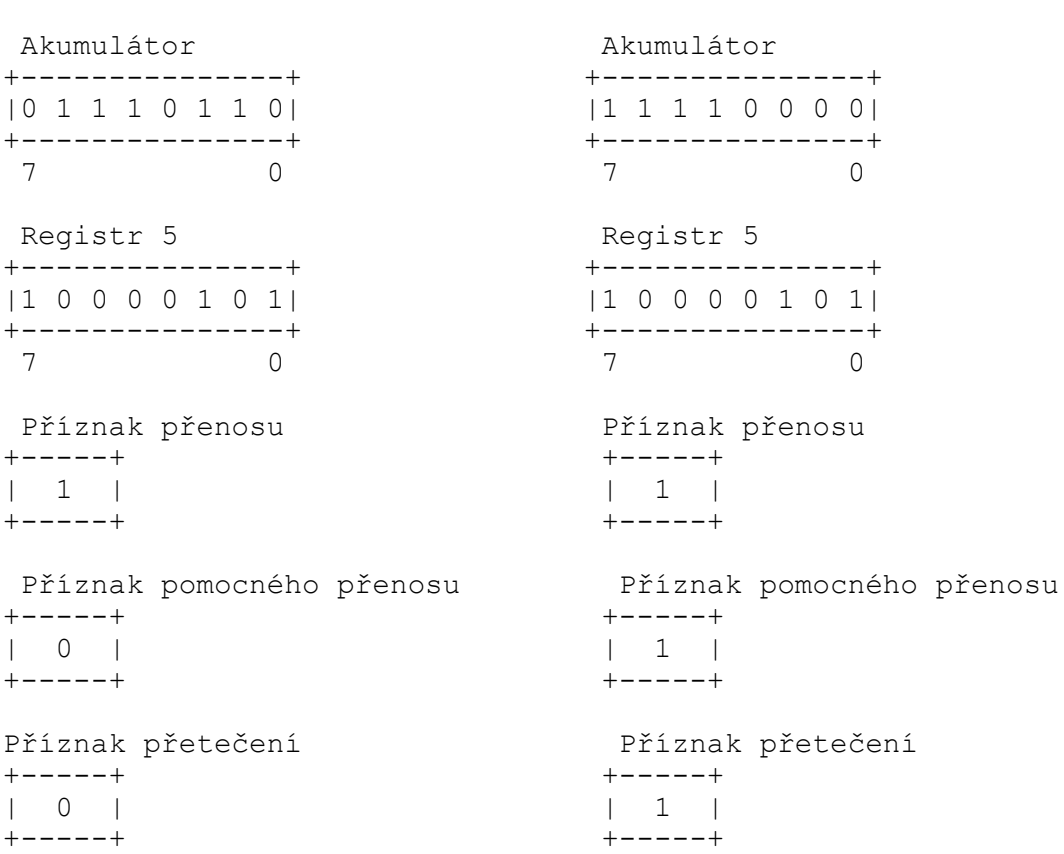
Popis: Tato instrukce odečte od obsahu akumulátoru hodnotu příznaku přenosu a obsah registru r. Výsledek je uložen v akumulátoru.

Příklad:

```
SUBB   A,R5      ;odečtení příznaku přenosu a obsa-
                 ;hu registru 5 od obsahu  akumulá-
                 ;toru
```

Kód instrukce:

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



Poznámky: 5, 6, 13, 14

SUBB

Odečtení obsahu datové adresy od obsahu akumulátoru s výpůjčkou

Mnemonika: SUBB

Operandy:
A	Akumulátor
datová adresa	0 <= datová adresa <= 7

Zápis: SUBB A,datová adresa

Strojový kód:

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	1	datová adresa							

Činnost: (A) <- (A) - (C) - (datová adresa)

Délka: 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

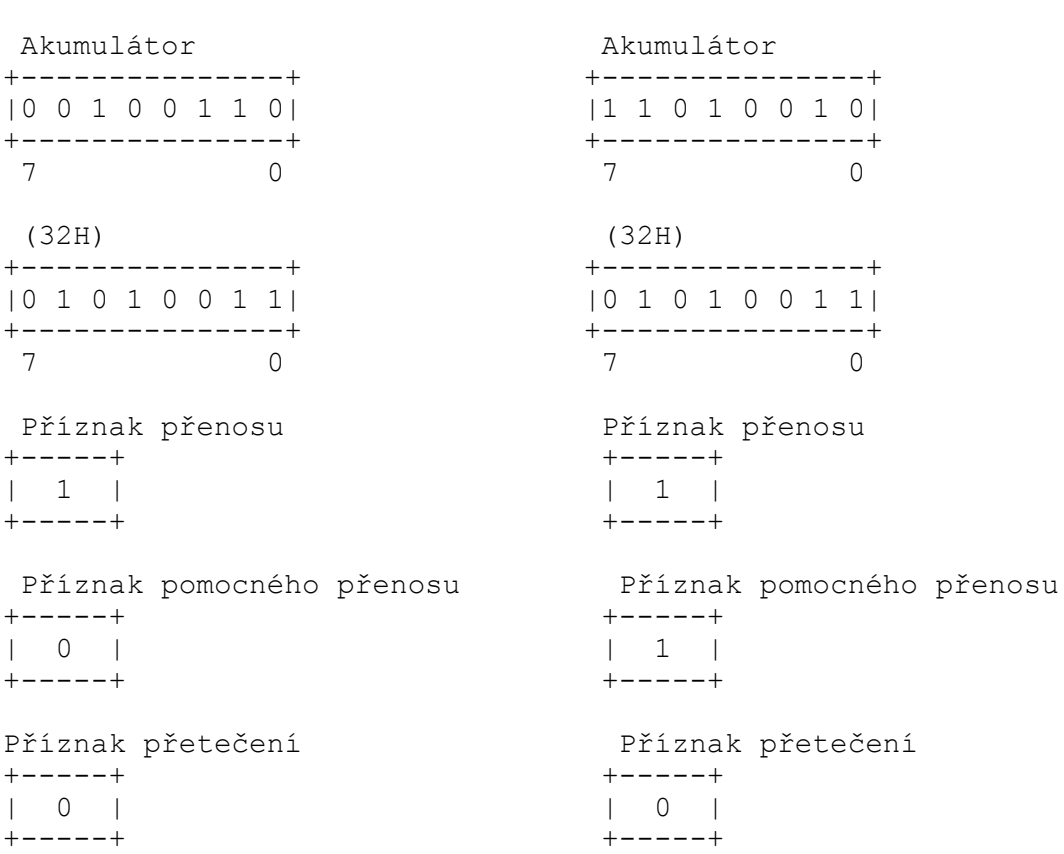
C	AC	F0	RS1	RS0	OV		P
x	x				x		x

PSW

Popis: Tato instrukce odečte od obsahu akumulátoru hodnotu příznaku přenosu a obsah určené datové adresy. Výsledek je uložen v akumulátoru.

Příklad:
```
SUBB   A,32H     ;odečtení příznaku přenosu a obsa-
                 ;hu dat na adrese  32H   od obsahu
                 ;akumulátoru
```
Kód instrukce:

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



Poznámky: 5, 6, 8, 13, 14

SWAP

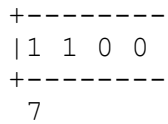
Výměna nižších a vyšších řádů (nibble) v akumulátoru

Mnemonika: SWAP

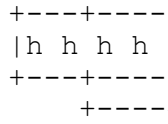
Operandy: A Akumulátor

Zápis: SWAP A

Strojový kód:



Činnost:



Délka : 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

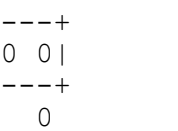
C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce provede výměnu nižších (bity 0 - 3) a vyšších (bity 4 - 7) řádů akumulátoru.

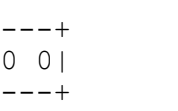
```
Příklad:      SWAP   A         ;výměna  nižších  a  vyšších  řádů
                               ;akumulátoru
```

Kód instrukce:



Před provedením

Akumulátor



Po provedení

Akumulátor

Poznámky: žádné

XCH

Výměna obsahu nepřímé adresy s obsahem akumulátoru

Mnemonika: XCH

Operandy:
A	Akumulátor
Rr	Registr 0 <= r <= 1

Zápis: XCH A,@Rr

Strojový kód:

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

Činnost:
```
(dočasný registr)  <-  ((Rr))
((Rr))   <-  (A)
(A)      <-  (dočasný registr)
```

Délka : 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

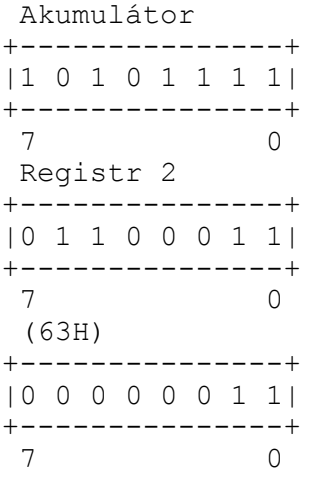
Popis: Tato instrukce provede výměnu obsahu paměťového místa na adrese určené obsahem registru r s obsahem akumulátoru.

Příklad:
```
XCH A,@R2          ;výměna obsahu místa v paměti na
                   ;adrese určené  obsahem registru
                   ;s akumulátorem
```

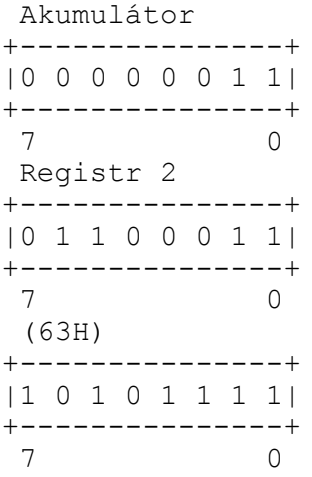
Kód instrukce:

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

Před provedením



Po provedení



Poznámky: 5, 15

XCH

Výměna obsahu registru s obsahem akumulátoru

Mnemonika: XCH

Operandy: A Akumulátor
Rr Registr 0 <= r <= 7

Zápis: XCH A,Rr

Strojový kód:

1	1	0	0	1	r	r	r
7							0

Činnost: (dočasný registr) <- (Rr)
(Rr) <- (A)
(A) <- (dočasný registr)

Délka : 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x
PSW							

Popis: Tato instrukce provede výměnu obsahu registru r s obsahem akumulátoru.

Příklad:
```
XCH A,R1          ;výměna obsahu registru 1
                  ;s akumulátorem
```

Kód instrukce:

1	1	0	0	1	0	0	1
7							0

Před provedením

Akumulátor

1	1	1	1	0	0	0	0
7							0

Registr 1

0	0	0	0	0	0	0	1
7							0

Po provedení

Akumulátor

0	0	0	0	0	0	0	1
7							0

Registr 1

1	1	1	1	0	0	0	0
7							0

Poznámky: 5

XCH

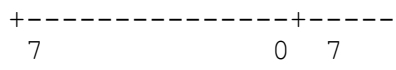
Výměna obsahu datové adresy s obsahem akumulátoru

Mnemonika: XCH

Operandy: A — Akumulátor
datová adresa — 0 <= datová adresa <= 255

Zápis: XCH A,datová adresa

Strojový kód:



Činnost:
(dočasný registr) <- (datová adresa)
(datová adresa) <- (A)
(A) <- (dočasný registr)

Délka : 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

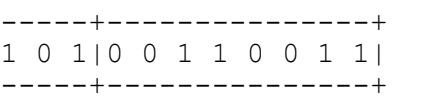
C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce provede výměnu obsahu určené datové adresy s obsahem akumulátoru.

Příklad:
```
XCH A,33H          ;výměna obsahu paměťového místa
                   ;na adrese 33H s akumulátorem
```

Kód instrukce:



Před provedením

Akumulátor

(33H)

Po provedení

Akumulátor

(33H)

Poznámky: 5, 9

XCHD

Výměna nižších řádů (číslic) obsahu nepřímé adresy
s obsahem akumulátoru

Mnemonika: XCHD

Operandy:
A	Akumulátor
Rr	Registr 0 <= r <= 1

Zápis: XCHD A,@Rr

Strojový kód:

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

Činnost:
(dočasný registr) <- ((Rr))0-3
((Rr))0-3 <- (A)0-3
(A)0-3 <- (dočasný registr)

Délka : 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

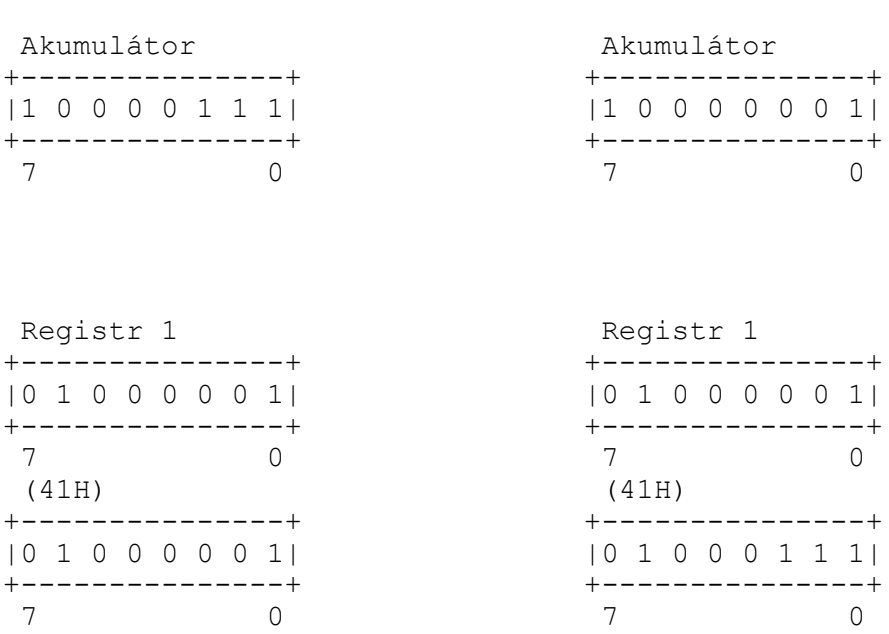
PSW

Popis: Tato instrukce provede výměnu nižších řádů (bity 0 - 3) obsahu paměťového místa na adrese určené obsahem registru r s obsahem nižších řádů akumulátoru. Vyšší řády se nemění.

Příklad:
```
XCHD  A,@R1      ;výměna mezi nižšími řády obsahu
                 ;paměťového místa na adrese určené
                 ;obsahem registru 1 a akumulátoru
```

Kód instrukce:

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



Poznámky: 5, 15

XRL

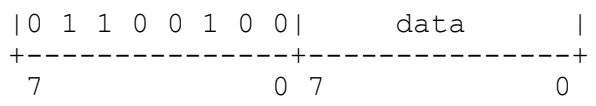
Součet modulo 2 dat s obsahem akumulátoru

Mnemonika: XRL

Operandy:
A		Akumulátor
data		-256 <= data <= +255

Zápis: XRL A,#data

Strojový kód:



Činnost: (A) <- (A) XOR data

Délka : 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

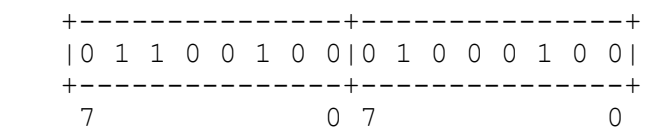
C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce provede součet modulo 2 8-bitových dat s obsahem akumulátoru. Bit n výsledku bude vy - nulován, jestliže bude mít bit n akumulátoru a dat stejnou hodnotu. Jinak má bit n výsledku logickou hodnotu 1. Výsledek je uložen v akumulátoru.

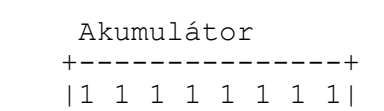
```
Příklad:      XRL A,#44H      ;součet modulo 2 obsahu akumulátoru
                              ;a dat o hodnotě 44H
```

Kód instrukce:



Před provedením

Akumulátor



Po provedení

Akumulátor

Poznámky: 4, 5

XRL

Součet modulo 2 obsahu nepřímé adresy s obsahem akumulátoru

Mnemonika: XRL

Operandy:

A	Akumulátor
Rr	Registr 0 <= r <= 1

Zápis: XRL A,@Rr

Strojový kód:

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

Činnost: (A) <- (A) XOR ((Rr))

Délka : 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

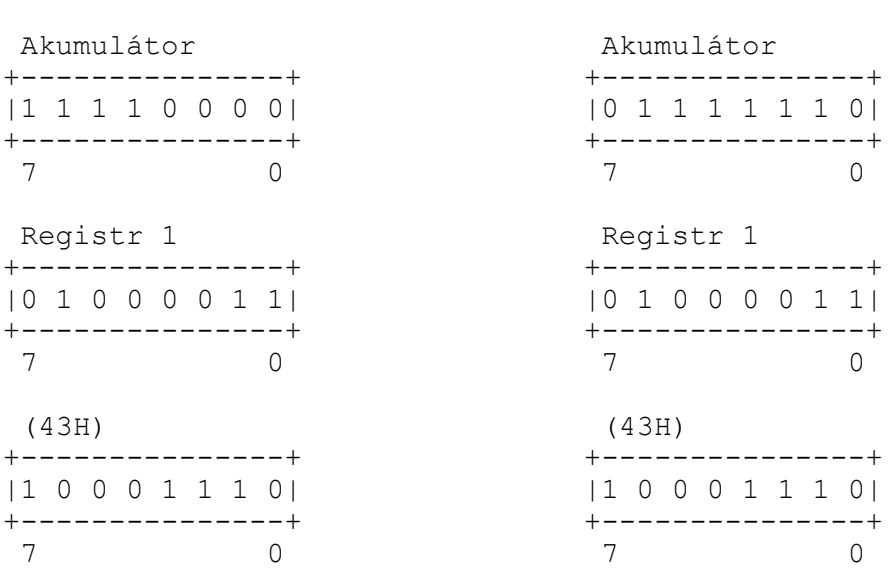
PSW

Popis: Tato instrukce provede součet modulo 2 obsahu paměťového místa na adrese určené obsahem registru r s obsahem akumulátoru. Bit n výsledku bude vynulován, jestliže bude mít bit n akumulátoru a adresovaného paměťového místa stejnou hodnotu. Jinak má bit n výsledku logickou hodnotu 1. Výsledek je uložen v akumulátoru.

Příklad:
```
XRL  A,@R1      ;součet modulo 2 obsahu akumulátoru
                ;a obsahu místa  v paměti na adrese
                ;určené obsahem registru 1
```

Kód instrukce:

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



Poznámky: 5, 15

XRL

Součet modulo 2 obsahu registru s obsahem akumulátoru

Mnemonika: XRL

Operandy: A Akumulátor
Rr Registr 0 <= r <= 7

Zápis: XRL A,Rr

Strojový kód:

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

Činnost: (A) <- (A) XOR (Rr)

Délka : 1 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x

PSW

Popis: Tato instrukce provede součet modulo 2 obsahu registru r s obsahem akumulátoru. Bit n výsledku bude vynulován, jestliže bude mít bit n akumulátoru a registru r stejnou hodnotu. Jinak má bit n výsledku logickou hodnotu 1. Výsledek je uložen v akumulátoru.

Příklad:
```
XRL  A,R3       ;součet modulo 2 obsahu akumulátoru
                ;a obsahu registru 3
```
Kód instrukce:

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

Před provedením

Akumulátor

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

Registr 3

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

Po provedení

Akumulátor

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

Registr 3

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

Poznámky: 5

XRL

Součet modulo 2 obsahu datové adresy s obsahem akumulátoru

Mnemonika: XRL

Operandy: A Akumulátor
datová adresa 0 <= datová adresa <= 255

Zápis: XRL A,datová adresa

Strojový kód:

0 1 1 0 0 1 0 1	datová adresa
7 0	7 0

Činnost: (A) <- (A) XOR (datová adresa)

Délka : 2 byte

Doba trvání: 1 strojový cyklus

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P
							x
PSW							

Popis: Tato instrukce provede součet modulo 2 obsahu datové adresy s obsahem akumulátoru. Bit n výsledku bude vynulován, jestliže bude mít bit n akumulátoru a obsahu datové adresy stejnou hodnotu. Jinak má bit n výsledku logickou hodnotu 1. Výsledek je uložen v akumulátoru.

Příklad:
```
XRL  A,42H      ;součet modulo 2 obsahu akumulátoru
                ;a obsahu dat na adrese 42H
```
Kód instrukce:

0 1 1 0 0 1 0 1	0 1 0 0 0 0 1 0
7 0	7 0

Před provedením

Akumulátor

0 1 1 1 0 0 0 1
7 0

(42H)

0 0 1 0 0 0 1 1
7 0

Po provedení

Akumulátor

0 1 0 1 0 0 1 0
7 0

(42H)

0 0 1 0 0 0 1 1
7 0

Poznámky: 5

XRL

Součet modulo 2 dat s obsahem datové adresy

Mnemonika: XRL

Operandy: datová adresa 0 <= datová adresa <= 255
data -256 <= data <= +255

Zápis: XRL datová adresa,#data

Strojový kód:

0 1 1 0 0 0 1 1	datová adresa	data
7 0	7 0	7 0

Činnost: (datová adresa) <- (datová adresa) XOR (data)

Délka : 3 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce provede součet modulo 2 8-bitových dat s obsahem určené datové adresy. Bit n výsledku bude vynulován, jestliže bude mít bit n dat a bit n obsahu datové adresy stejnou hodnotu. Jinak má bit n výsledku logickou hodnotu 1. Výsledek je uložen v paměti dat na určené datové adrese.

Příklad: XRL 42H,#5FH ;součet modulo 2 obsahu paměti na
;adrese 42H a dat o hodnotě 5FH

Kód instrukce:

0 1 1 0 0 0 1 1	0 1 0 0 0 0 1 0	0 1 0 1 1 1 1 1
7 0	7 0	7 0

Před provedením

(42H)

0 0 0 0 0 1 0 1
7 0

Po provedení

(42H)

0 1 0 1 1 0 1 0
7 0

Poznámky: 4, 9

XRL

Součet modulo 2 obsahu akumulátoru s obsahem datové adresy

Mnemonika: XRL

Operandy: datová adresa 0 <= datová adresa <= 255
A Akumulátor

Zápis: XRL datová adresa,A

Strojový kód:

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	1	1	0	0	0	1	0	datová adresa							

Činnost: (datová adresa) <- (datová adresa) XOR (A)

Délka : 2 byte

Doba trvání: 2 strojové cykly

Ovlivňuje:

C	AC	F0	RS1	RS0	OV		P

PSW

Popis: Tato instrukce provede součet modulo 2 obsahu akumulátoru s obsahem určené datové adresy. Bit n výsledku bude vynulován, jestliže bude mít bit n obsahu akumulátoru a obsahu datové adresy stejnou hodnotu. Jinak má bit n výsledku logickou hodnotu 1. Výsledek je uložen v paměti dat na určené datové adrese.

Příklad:
```
XRL  15H,A      ;součet modulo 2 obsahu akumulátoru
                ;a obsahu na adrese 1542H
```

Kód instrukce:

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

Před provedením

Akumulátor

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

(10H)

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

Po provedení

Akumulátor

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

(10H)

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

Poznámky: 9

Poznámky

1. Slabika nižších řádů programového čítače se vždy ukládá do zásobníku jako první, potom se uloží slabika vyšších řádů.

2. Ukazatel zásobníku vždy ukazuje na poslední slabiku uloženou do zásobníkové paměti (tzn. na vrchol zásobníku).

3. U obvodů 8051 nesmí obsah ukazatele zásobníku překročit hodnotu 127. Pokud tuto hodnotu překročí, jsou data uložená v zásobníku ztracena a při jejich obnově se pracuje s nedefinovanými údaji. Obsah ukazatele zásobníku se však normálně inkrementuje i když tato ztracená data už nemůžou být opět získána.

4. Výraz použitý jako datový operand musí být vyhodnocen na 8-bitové číslo. Hodnoty výrazů, které jsou zpracovávány během překladu, musí být v rozsahu -256 až +55.

5. Příznak parity (bit 0 ve stavovém slově procesoru, PSW.0) vždy ukazuje paritu akumulátoru.. Je-li v akumulátoru lichý počet jedniček, nastaví se příznak parity do logické hodnoty 1, v opačném případě se příznak parity vynuluje.

6. Všechny součtové operace ovlivňují příznak přenosu (PSW.7) a příznak pomocného přenosu (PSW.6). Příznak přenosu zachycuje přenos z bitu 7 akumulátoru. Příznak pomocného přenosu zachycuje přenos z bitu 3 akumulátoru. Každé operace ADD nastavuje nebo nuluje každý z obou příznaků.

7. Příznak přetečení (OV) se nastaví tehdy, pokud po operaci vznikne chybný výsledek. (tj. součet dvou kladných čísel je záporný nebo součet dvou záporných čísel je kladný). Příznak přetečení se aktualizuje při každé operaci.

8. Je-li některý z portů V/V určen datovou adresou, potom budou data odebírána ze vstupních špiček portu.

9. Je-li některý z portů V/V určen datovou adresou, potom budou data odebírána nebo vracena do záchytných klopných obvodů portu.

10. Operand <adr> - "adresa v paměti programu" musí být v rozsahu od -128 do +127 a inkrementuje obsah programového čítače.

11. Poslední slabika strojového kódu instrukce je pokládána za číslo ve dvojkovém doplňkovém kódu. Toto číslo se přičte k obsahu programového čítače.

12. Programový čítač je vždy inkrementován před přičtením čísla z poslední slabiky strojového kódu instrukce

13. Příznak pomocného přenosu je vždy nastaven do logické hodnoty 1, dojde-li k výpůjčce ze 3. bitu akumulátoru. Jinak je vynulován.

14. Příznak přetečení (OV) se nastavuje tehdy, pokud vznikne po operaci chybný výsledek (tj při odečtení kladného čísla od záporného vznikne kladný výsledek nebo při odečtení záporného čísla od kladného vznikne záporný výsledek). Příznak přetečení se nuluje každou správnou operací.

15. U obvodů 8051 nesmí obsah registru používaného pro nepřímé adresování překročit hodnotu 127. Je-li obsah registru roven 128 nebo více, potom zdrojové operandy poskytují nedefinovaná data a data ukládaná na místo cílových operandů jsou ztracena. V obou případech program pokračuje beze změny dále. U procesorů RUPI-44 je adresová hranice rovna 192.

16. Je-li port V/V specifikován jako zdrojový operand, jsou data čtena ze špiček portu. Je-li port V/V cílový operand, jsou data přijímána do záchytných klopných obvodů.

17. Je-li obsah ukazatele zásobníku 128 nebo větší (192 u RUPI-44), navracejí se neplatná data při instrukci POP nebo RETURN.

18. Stavové slovo programu je ovlivněno v případě, že operandem je registr PSW nebo některý z jeho bitů.

Kapitola 4 - DIREKTIVY ASEMBLERU

Tato kapitola popisuje direktivy asembleru. Vysvětluje způsob definice symbolů a pravidla používání direktiv ve zdrojovém textu.

Všeobecně

Asembler ASM-52 má celou řadu direktiv, které umožňují přiřazovat symbolům hodnoty, rezervovat paměťový prostor a řídit umístění instrukcí do paměti programu.

Direktivy nelze zaměňovat s instrukcemi. Výsledkem jejich použití není proveditelný kód a s vyjímkou direktiv DB a DW neovlivňují obsah paměti. Ovlivňují činnost asembleru, definují symboly a doplňují informace obsažené v cílovém kódu.

Direktivy lze dělit do následujících skupin:

- Definice symbolů
  - SEGMENT
  - EQU
  - SET
  - BIT
  - CODE
  - DATA
  - IDATA
  - XDATA
- Inicializace a rezervování paměti
  - DS
  - DB
  - DW
  - DBIT
- Spojování programu
  - NAME
  - EXTERNAL
  - PUBLIC
- Řízení asembleru
  - AT
  - ORG
  - END
  - Výběr segmentů
    - BSEG
    - CSEG
    - DSEG
    - ISEG
    - RSEG
    - XSEG
  - USING

Asembler ASM-52 tvoří cílový kód v několika krocích (průchodech). Při prvním průchodu vytvoří tabulku symbolů a symbolům přiřadí hodnoty, ve druhém průchodu vyřeší dopředné odkazy a generuje cílový kód. Tím je dáno určité omezení pro zdrojový text programu, spočívající v nutnosti použít pro direktivy definující symboly a ovlivňující přidělování paměťových adres (ORG, AT, DS, DBIT) pouze takové výrazy, které nepoužívají žádné dopředné odkazy.

Čítač lokací

Čítač lokací je ukazatel do adresního prostoru právě zpracovávaného (aktivního) segmentu. Při první aktivaci (definici) segmentu je tento čítač nastaven na nulu, pokud neobsahuje definice direktivu AT, určující jinou bázovou adresu. Čítač je pak měněn každou instrukcí nebo direktivou ovlivňující přidělování paměti. Je možno jej měnit pomocí direktivy ORG, nelze jej však nastavit na hodnotu nižší, než je bázová adresa. Pokud je aktivován jiný segment a později je již definovaný segment znovu aktivován, nastaví se hodnota čítače na naposledy použitou hodnotu.

Speciální symbol asembleru "$" lze použít jako hodnotu, odpovídající okamžité hodnotě čítače lokací aktivního segmentu. Je však třeba myslet na to, že hodnota tohoto symbolu se dynamicky mění s každou instrukcí nebo direktivou, ovlivňující přidělování paměti. Symbol "$" lze využít např. takto:

```
STRING:  DB       STRLEN,'ASCII STRING'
STRLEN   EQU      $-STRING-1
```

Jména symbolů

Jméno symbolu musí začínat písmenem nebo jedním ze znaků "_" či "?". Následující znaky mohou být písmena, číslice nebo některé speciální znaky. Speciální znaky jsou:

```
+  -  *  /  ,  (  )  '  "  &  :  $  @  ?  =  <  >  %  !  ;  .  _  |  ^
```

a znaky TAB (09H), SPACE (20H), CR (0DH) a LF (0AH). Asembler nečiní rozdílu mezi znaky malé a velké abecedy. Jméno symbolu může být tvořeno až 255 znaky, ale pouze prvních 32 je významných z hlediska rozlišení. To znamená, že následující tři symboly jsou z hlediska asembleru ASM-52 totožné:

```
_THIS__IS__A__LONG__SYMBOL__NAME
_this__is__a__long__symbol__name
_THIS__IS__A__LONG__SYMBOL__NAME__LONGER_THAN_BOTH_ABOVE
```

Jména symbolů se nesmí shodovat s mnemonikou instrukcí, operátory asembleru a ostatními rezervovanými slovy a ani s předdefinovanými symboly, které jsou zvoleny pomocí volby MODE. Kompletní přehled rezervovaných slov asembleru je v příloze 4.

Návěští

Návěští je symbol, jehož hodnota však není přiřazena uživatelem, ale je určena hodnotou čítače lokací v okamžiku jeho definice. Návěští se vyskytuje na řádku buď samotné nebo předchází jednu z direktiv DB, DW, DS a DBIT či instrukci asembleru a za jeho jménem bezprostředně následuje znak ":". Na jednom řádku se smí vyskytovat pouze jediné návěští.

Návěští má příslušnost k segmentu určenou aktivním segmentem, jeho hodnota je dána čítačem lokací v aktivním segmentu v okamžiku definice. Pokud je symbol absolutní, je absolutní i hodnota přiřazená návěští, u přemístitelných segmentů je přemístitelná.

Návěští je možno použít jako libovolný jiný symbol ve výrazech nebo jako adresu paměťového místa. Návěští není možno předefinovat.

Definování symbolů

Direktivy pro definování symbolů umožňují vytvářet symboly, reprezentující registry, segmenty paměti, číselné konstanty a adresy paměti. Žádná z těchto direktiv nesmí být předcházena návěštím.

Direktiva SEGMENT

Všeobecný zápis pro použití direktivy segment je:

```
jméno_přemístitelného_segmentu SEGMENT typ_segmentu [přemístění]
```

Direktiva SEGMENT umožňuje deklarovat přemístitelný segment, přiřadí mu atribut třídy, určí způsob přemístění a nastaví čítač lokací na nulu.

Jméno segmentu se nesmí v modulu opakovat. Pokud se v jiném modulu vyskytne segment se stejným jménem, jsou linkerem oba spojeny do jediného segmentu stejného jména. Je ale nutné, aby takové segmenty měli stejné atributy a stejný způsob přemístění, nebo mohou mít dva typy přemístění, ale jeden z typů musí být UNIT (viz dále).

Atribut typ_segmentu určuje paměťový prostor, pro nějž je segment deklarován. Povolené typy jsou:

```
BIT       - bitově adresovatelný prostor
CODE      - prostor paměti programu
DATA      - přímo adresovatelná data na čipu (0..127)
IDATA     - nepřímo adresovatelný prostor  na čipu (rozsah 0..127
            pro 8051, 0..192 pro 8044, 0..255 pro 8052)
XDATA     - externí datová paměť
```

Atribut typ_přemístění je volitelný, pokud se nepoužije, předpokládá se typ UNIT. Tento atribut má vliv na spojovací činnost linkeru. Typy přemístění jsou:

PAGE - segment, jehož bázi je třeba umístit na počáteční adresu 256-ti bytové stránky. Povolený pouze pro segmenty třídy CODE a XDATA.

INPAGE - segment musí být umístitelný do rozsahu 256-ti bytové stránky. Povolený pouze pro segmenty třídy CODE a XDATA.

INBLOCK - segment musí být umístitelný do rozsahu 2048-ti bytové (2kB) bloku. Povolený pouze pro segmenty třídy CODE.

BITADDRESSABLE - Segment musí být umístěn tak, aby umožňoval současně bitovou adresaci. Rozsah je max. 16 byte, povoleno pouze pro segmenty typu DATA.

UNIT - segment bude umístěn pouze s ohledem na datovou jednotku segmentu. Tou je byte u typů CODE, DATA, IDATA a XDATA a bit u typu BIT.

Zásobník může být umístěn kdekoliv v segmentu DATA nebo IDATA, např.:

```
STACK SEGMENT IDATA
RSEG  STACK
DS    16
      .
      .
MOV   SP,STACK
```

Direktiva EQU

Direktivu EQU lze napsat jedním z těchto tvarů :

```
jméno symbolu   EQU   výraz
```
nebo
```
jméno symbolu   EQU   rezervovaný symbol
```

Direktiva EQU přiřazuje jménu symbolu číselnou hodnotu výrazu nebo rezervovaný symbol. Při přiřazování výrazu k symbolu musí být výraz absolutní nebo jednoduše přemístitelný. Jako jméno symbolu je možné použít datovou nebo programovou adresu, adresu bitu nebo adresu dat v závislosti na typu segmentu výrazu (symbol má stejný typ segmentu jako přiřazovaný výraz).

Direktivou EQU je též možné přiřadit libovolná jména rezervovaným symbolům asembleru A, R0 až R7. Je-li přiřazován symbol k registru, lze ho potom použít na místě registrového operandu v instrukcích.

Jméno symbolu definované direktivou EQU nemůže být předefinováno.

Příklady:

```
     STEP     EQU     A      ;akumulátoru A přiřadí symbol STEP
     RAZ      EQU     $      ;symbolu RAZ přiřadí současnou hodno-
                             ;tu počitadla adres
     M28      EQU     28     ;symbol 28 je ekvivalentní hodnotě 28
     ADR1     EQU     ADR0+1 ;ADR0 je adresa dat, pak ADR1 je také
                             ;adresa dat
```

Direktiva SET

Direktivu SET lze napsat jedním z těchto tvarů :

```
     jméno symbolu   SET   výraz
nebo
     jméno symbolu   SET   rezervovaný symbol
```

Účinek direktivy SET je stejný jako při použití direktivy EQU. Rozdíl je v tom, že jméno symbolu definované direktivou SET lze pomocí další direktivy SET znovu předefinovat.

Direktivou SET nelze měnit jméno symbolu, které bylo přiřazeno direktivou EQU a naopak.

Příklady :

```
   ABC       SET     25     ;přiřazení čísla 25 symbolu ABC
   AHOJ      SET     R2     ;registru R2 je přiřazen symbol AHOJ
   CTVRT     SET     KAP/4  ;vytvoření čtvrtiny z proměnné KAP
   CIT       SET     0      ;nastavení čítače na 0
   CIT       SET     CIT+1  ;inkrementování čítače o 1
```

Direktiva BIT

Direktiva BIT se zapisuje ve tvaru :

```
     jméno symbolu   BIT   adresa bitu
```

Direktiva BIT přiřazuje jménu symbolu adresu bitu. Formát adres bitu je popsán v kapitole 2. Jména symbolů vytvořená touto direktivou mají segment BIT a nesmějí být předefinována.

Příklady :

```
     START:   DS    1
              .
              .
      POL1    BIT   START.0 ;bit 0 ve slabice START
      POL2    BIT   POL1+1  ;bit 1 ve slabice START
      POL3    BIT   035H    ;absolutní bit
```

Direktiva CODE

Direktiva CODE se zapisuje ve tvaru :

```
jméno symbolu   CODE   výraz
```

Direktiva CODE přiřazuje jménu symbolu adresu programu. Použitý výraz musí být buď absolutní nebo jednoduše přemístitelný výraz. Nevyhodnotí-li se výraz jako číslo, musí být typu CODE a jméno symbolu má potom také typ CODE. Symbol vytvořený direktivou CODE nesmí být nikde v programu znovu definován.

Příklady :

```
ZACAT   CODE   00H
POKR1   CODE   08H
KONEC   CODE   2DH
```

Direktiva DATA

Direktiva DATA se zapisuje ve tvaru :

```
jméno symbolu   DATA   výraz
```

Direktiva DATA přiřazuje jménu symbolu adresu datové paměti umístěné na čipu. Použitý výraz musí být buď absolutní nebo jednoduše přemístitelný. Absolutní výrazy, jejichž hodnota bude větší než 127, adresují místa v oblasti datové paměti, kde jsou definovány hardwarové registry. V případě, že výraz není vyhodnocen jako číslo, jeho typ segmentu musí být DATA a definované jméno symbolu bude také typu DATA. Jméno symbolu definované touto direktivou nesmí být v programu předefinováno.

Příklady :

```
ZAC     DATA    50H
KON     DATA    85H
DEF     DATA    ZAC+2  ;definice symbolu  DEF pomocí ZAC
```

Direktiva IDATA

Direktiva IDATA se zapisuje ve tvaru :

```
jméno symbolu   IDATA    výraz
```

Direktiva IDATA přiřazuje jménu symbolu nepřímou vnitřní datovou adresu. Použitý výraz musí být buď absolutní nebo jednoduše přemístitelný. U procesoru 8051 nesmí být hodnota absolutního výrazu větší než 127. V případě, že výraz není vyhodnocen jako číslo, jeho typ segmentu musí být IDATA a definované jméno symbolu bude také typu IDATA. Jméno symbolu definované touto direktivou nesmí být v programu předefinováno.

Příklady :

```
POLE     IDATA    20H
METR     EQU      10H
POLE2    IDATA    POLE+METR
```

Direktiva XDATA

Direktiva XDATA se zapisuje ve tvaru :

jméno symbolu XDATA výraz

Direktiva XDATA přiřazuje jménu symbolu adresu ve vnější datové paměti. Použitý výraz musí být buď absolutní nebo jednoduše přemístitelný. V případě, že výraz není vyhodnocen jako číslo, jeho typ segmentu musí být XDATA a definované jméno symbolu bude také typu XDATA. Jméno symbolu definované touto direktivou nesmí být v programu předefinováno.

Příklady :

```
         RSEG     XSEG1
         ORG      80H
DEN:     DS       10
HOD      XDATA    DEN+12
MIN      XDATA    HOD+15
```

Rezervování a inicializace paměti

Direktivy uvedené v této kapitole slouží k definování nebo k obsazení určeného paměťového prostoru. Definování nebo rezervování začíná od adresy dané aktuální hodnotou počítadla adres v aktivním segmentu.

Před těmito direktivami může být uvedeno návěští.

Direktiva DS

Direktiva DS se zapisuje ve tvaru :

[návěští] DS výraz

Direktiva DS rezervuje určitý počet slabik v paměti. Může se použít ve všech segmentech s výjimkou segmentu BIT. Rezervovaný počet slabik je dán číselnou hodnotou výrazu. Tato hodnota musí být známá v okamžiku zpracování direktivy překladačem, tzn. že musí jít o absolutní výraz a všechna jména použitá ve výrazu musí být definována před touto direktivou DS. Vyhodnocení direktivy se provede tak, že se inkrementuje počitadlo adres segmentu o počet slabik daných číselnou hodnotou výrazu. Výsledná hodnota počitadla přitom nesmí překročit omezení dané pro adresový prostor, ve kterém se direktiva nachází.

Návěští je nepovinné, pokud se uvede, je mu přiřazena okamžitá hodnota čítače lokací. Potom ukazuje na první slabiku rezervovaného bloku paměti.

Direktiva DBIT

Direktiva DBIT se zapisuje ve tvaru :

```
[návěští]    DBIT   výraz
```

Direktiva DBIT rezervuje určitý počet bitů v paměti. Může se použít pouze v segmentu BIT, tedy v adresovém prostoru, kde lze provádět adresování jednotlivých bitů. Rezervovaný počet bitů je dán číselnou hodnotou výrazu. Tato hodnota musí být známá v okamžiku zpracování direktivy překladačem, tzn. že musí jít o absolutní výraz a všechna jména použitá ve výrazu musí být definována před touto direktivou DBIT. Vyhodnocení direktivy se provede tak, že se inkrementuje počitadlo adres segmentu o počet bitů daných číselnou hodnotou výrazu.

Direktiva DB

Direktiva DB se zapisuje ve tvaru :

```
[návěští]    DB   výraz[,výraz,...]
```

Direktiva DB ukládá do programové paměti 8-bitové hodnoty, které jsou dány vyhodnocením položek seznamu výrazů. Direktiva se může použít pro segment typu CODE. Seznam výrazů musí obsahovat položky, které jsou absolutní, jednoduše přemístitelné nebo všeobecně přemístitelné výrazy, případně znakové řetězce. Jednotlivé položky v seznamu výrazů jsou odděleny čárkou. Položky seznamu se ukládají do paměti vzestupně ve stejném pořadí, v jakém byly uvedeny v seznamu. Počet položek seznamu je omezen délkou zdrojového řádku. Pro větší počet položek se použije několik direktiv DB.

Direktiva DB umožňuje zadávání znakových řetězců delších než 2 znaky, takové řetězce však nesmějí být součástí výrazů. Znakový řetězec musí být uzavřen v apostrofech. V seznamu lze zadat i prázdný řetězec. Je-li uvedeno před direktivou návěští, bude mu přiřazena adresa první slabiky v seznamu.

Příklady :

```
ROZVRH:   DB   'PONDELI',10,30,'STREDA',12,40
                  ;v seznamu jsou uvedeny znakové řetězce
                  ;(PONDELI,STREDA) a číselné hodnoty, které
                  ;se zadají do programové paměti
                  ;návěští ROZVRH adresuje písmeno P
SEZNAM:   DB   2,4,6,8,10,12,14,16
                  ;seznam osmi číselných hodnot
                  ;návěští SEZNAM adresuje číslo 2
```

Direktiva DW

Direktiva DW se zapisuje ve tvaru :

[návěští] DW výraz[,výraz,...]

Direktiva DW ukládá do programové paměti 16-bitové hodnoty, které jsou dány vyhodnocením položek seznamu výrazů. Direktiva se může použít jen pro segment typu CODE. Seznam výrazů musí obsahovat položky, které jsou absolutní, jednoduše přemístitelné nebo všeobecně přemístitelné výrazy, případně znakové řetězce. Jednotlivé položky v seznamu výrazů jsou odděleny čárkou. Položky seznamu se ukládají do paměti vzestupně ve stejném pořadí, v jakém byly uvedeny v seznamu. Počet položek seznamu je omezen délkou zdrojového řádku. Pro větší počet položek se použije několik direktiv DW.

Direktiva DW neumožňuje zadávání znakových řetězců delších než 2 znaky, řetězce ale mohou být součástí výrazů. Znakový řetězec musí být uzavřen v apostrofech. V seznamu lze zadat i prázdný řetězec. Slova se do paměti ukládají v pořadí vyšší byte napřed, pak nižší byte. Je-li uvedeno před direktivou návěští, bude mu přiřazena adresa vyššího byte prvního slova v seznamu.

Příklady :

```
ROKY: DW  1978,1979,1990
      ;seznam letopočtů
DATA: DW  1007,'AM',0408,'PM'
      ;seznam časových údajů
VECT: DW  FIRST_LABEL, SECOND_LABEL, THIRD_LABEL
      ;tabulka vektorů pro nepřímé skoky
```

Direktivy pro spojování programu

Direktiva PUBLIC

Direktiva PUBLIC se používá takto:

```
PUBLIC seznam_symbolů
```

Direktiva PUBLIC umožňuje použití symbolů definovaných uvnitř jednoho modulu i pro jiné moduly, které budou spojovány společně. Symboly v seznamu se oddělují čárkami. Pokud se některý symbol deklaruje jako PUBLIC vícekrát, nepovažuje to ASM-52 za chybu. Podmínkou ale je, aby symbol, deklarovaný jako PUBLIC byl někde v programu definován, ať již před jeho deklarací jako PUBLIC nebo až po ní. Předem definované symboly (nastavené volbou MODE) a jména segmentů nesmí být deklarovány jako PUBLIC.

Příklady:

```
PUBLIC GET_TIME, SET_TIME
PUBLIC DATE_FLAG,DATE_VALUE
```

Direktiva EXTRN

Zápis direktivy EXTRN vypadá následovně:

```
EXTRN typ_segmentu(seznam_symbolů)
```

Direktiva EXTRN oznamuje asembleru, že symboly s příslušností k segmentu uvedeném v direktivě jsou definovány v jiném modulu. Direktiva EXTRN se může vyskytnout kdekoliv v programu.

Seznam symbolů direktivy EXTRN musí být předcházen typem segmentu, k němuž symboly uvnitř seznamu přísluší. Type segmentu je buď BIT, CODE, DATA, IDATA, XDATA nebo NUMBER. Při spojování modulů musí linker v právě jednom ze spojovaných modulů nalézt definici odpovídajícího symbolu s atributem PUBLIC, v opačném případě ohlásí linker chybu. Oba symboly (v jednom modulu v seznamu EXTRN, ve druhém modulu v seznamu PUBLIC) musí mít stejnou příslušnost k segmentu nebo jeden z nich musí být třídy NUMBER.

Příklady:

```
EXTRN CODE(MAIN,PROC)
EXTRN BIT(STEP_FLAG)
```

Direktiva NAME

Direktiva NAME se používá k definici jména modulu takto:

```
NAME jméno_modulu
```

Pro jméno modulu platí stejná pravidla, jako pro ostatní jména symbolů. Direktiva NAME se musí vyskytovat v programu dříve, než se použije nějaká jiná direktiva nebo instrukce. Může být předcházena pouze příkazy asembleru (viz kapitola 5). Pokud se direktiva NAME nepoužije, jako jméno modulu se použije jméno souboru bez přípony.

Příklad:

```
NAME MAIN
```

Řízení asembleru

Direktiva AT

Direktiva AT se používá ve spojení s direktivou výběru absolutního segmentu (viz str. 4-11, Výběr segmentů) pro nastavení bázové adresy segmentu. Její význam je podobný jako použití direktivy ORG, ale bázová adresa má silnější účinek v tom, že čítač lokací nelze direktivou ORG nastavit na nižší adresu, než je bázová adresa.

Příklad:

```
CSEG AT 100H    ;Nastavení báze programového segmentu
```

Direktiva ORG

Direktiva ORG se používá na změnu obsahu čítače lokací aktivního segmentu, ať absolutního nebo přemístitelného. Použití vypadá takto:

ORG výraz

Výraz musí být absolutní nebo jednoduše přemístitelný, v tom případě se ale musí vztahovat na aktivní segment. Direktiva ORG nesmí být předcházena návěštím.

Použití direktivy ORG mění obsah čítače lokací v aktivním segmentu. Nové instrukce budou zařazovány od nově nastavené adresy. Pokud je segment absolutní, nastavuje direktiva ORG novou absolutní adresu v aktivním segmentu, nesmí však být menší, než je bázová adresa nastavená direktivou AT. V případě přemístitelného segmentu nastaví direktiva ORG hodnotu offsetu od báze aktivního segmentu.

Příklad:

```
          ORG  8000H
          ORG  $ AND 0FFF0H
```

Direktiva END

Direktiva END je povinná a oznamuje asembleru ukončení zdrojového textu. Jakýkoliv text, vyskytující se za direktivou END se považuje za komentář. Direktiva END má povinný tvar:

END

Direktiva END nesmí být předcházena návěštím.

Výběr segmentů

Direktivy pro výběr segmentů umožňují segmentaci částí programu pro různé paměťové oblasti, přepínat aktivní segmenty a vytvářet nové aktivní segmenty.

Formát direktivy pro přemístitelný segment je:

RSEG jméno_segmentu

kde jméno segmentu musí být předtím definováno direktivou SEGMENT.

Pro absolutní segmenty je tvar direktivy tento:

```
+----+
|BSEG|
|CSEG|
|DSEG| [AT absolutní_výraz]
|ISEG|
|XSEG|
+----+
```

BSEG, CSEG, DSEG, ISEG a XSEG aktivuje absolutní segment v bitové, programové, přímo adresovatelné datové, nepřímo adresovatelné datové a externí datové paměti. Pokud je pomocí direktivy AT nastavena i bázová adresa, je nastaven čítač lokací na hodnotu specifikovanou výrazem, který musí být absolutní. Pokud direktiva AT chybí, je bázová adresa nastavena na nulu. Návěští před direktivou nelze použít.

Každý segment má svůj čítač lokací. Ten je při výběru nastaven vždy na nulu, pokud není direktivou AT nastaven na nějakou jinou hodnotu. Po aktivaci asembleru je aktivní absolutní programový segment s bází nula. Při aktivaci přemístitelného segmentu je čítač lokací nastaven na poslední použitou hodnotu, která byla použita při předchozí aktivaci téhož segmentu. Lze použít direktivu ORG na nastavení čítače lokací na jinou hodnotu.

Příklady:

```
PROGRAM     SEGMENT     CODE
STACK       SEGMENT     IDATA

RSEG STACK
DS 10

CSEG AT 0
LJMP  START

RSEG PROGRAM
START:
      .
      .
      .
```

Direktiva USING

Direktiva USING oznamuje asembleru, že následující část programu bude používat sadu novou banku registrů:

```
USING absolutní_výraz
```

Výraz, určující nově používanou sadu registrů, musí být absolutní a musí dát výsledek buď 0, 1, 2 nebo 3. Direktiva má přímý vliv na číselný ekvivalent speciálních symbolů asembleru AR0..AR7.

Příklad:

```
USING 0
CJNE A,AR2,$+3 ;Porovnání obsahu akumulátoru s obsahem
               ;registru 2 banky 0
USING 1
PUSH AR2       ;Uložení obsahu registru 2 banky 1
```

Kapitola 5 - ŘÍZENÍ ASEMBLERU

Tato kapitola popisuje ovládání asembleru. Vysvětluje způsob použití jednotlivých řídících příkazů a jejich použití uživatelem pro dosažení požadovaného průběhu překladu zdrojového textu. Zabývá se rovněž popisem způsobu spuštění asembleru na počítači s operačním systémem DOS.

Jak spustit asembler ASM-52

Všeobecný tvar příkazového řádku má tento tvar:

```
[Disk:][cesta]ASM52 [Disk:][cesta]{jméno_souboru}[řídící_příkazy]
```

Hranaté závorky označují volitelnou část příkazového řádku, složené povinnou. Z toho je vidět, že nejjednodušším způsobem spuštění asembleru může být příkaz

```
ASM52 <soubor>
```

V takovém případě se předpokládá, že asembler ASM52 je buď ve stejném adresáři, jako je zdrojový soubor nebo je na něj vybudovaná cesta příkazem PATH (např. pomocí dávkového souboru AUTOEXEC.BAT). Pro řízení překladu budou použity vnitřně nastavené příkazy (viz. dále nebo příloha 5). Vytvoří se cílový soubor s příponou .OBJ a protokol o překladu (listing) s příponou .LST Žádný další soubor se nevytvoří, asembler ASM52 nepoužívá žádný dočasný soubor.

V příkazovém řádku lze použít libovolný z dále popsaných příkazů vyjma příkazu INCLUDE, který se může vyskytnout pouze ve zdrojovém textu.

Pokud je v příkazovém řádku nalezena chyba, vypíše se chybové hlášení a činnost asembleru skončí, tj. takové chyby se považují za fatální.

Řídící příkazy asembleru

Jak již bylo řečeno, řídící příkazy asembleru se mohou vyskytnout v příkazovém řádku nebo mohou být součástí zdrojového textu. Řádek zdrojového textu může obsahovat jeden nebo více řídících příkazů. Tvar příkazového řádku ve zdrojovém textu má tento tvar:

```
$seznam_řídících_příkazů [;komentář]
```

Znak '$' indikuje, že se jedná o příkazový řádek a musí být prvním znakem řádku. Seznam řídících příkazů může být prázdný (tj. nemusí obsahovat žádný příkaz), může obsahovat jeden příkaz a nebo libovolné množství příkazů.

ASM52 má celou řadu příkazů, které se dělí do dvou skupin, na primární a všeobecné.

Primární příkazy mohou být uvedeny ve spouštěcím příkazovém řádku nebo jako jeden nebo více příkazových řádků ve zdrojovém textu. Jejich nastavení je platné pro celý průběh překladu. Z tohoto důvodu lze primární příkazy specifikovat ve vyvolávacím příkazovém řádku nebo v příkazovém řádku zdrojového textu pouze jednou. Pokud se ve spouštěcím příkazovém řádku použije primární řídící příkaz stejných jako je uveden ve zdrojovém textu, má přednost příkaz použitý ve vyvolávacím příkazovém řádku. Vzhledem k tomu, že primární příkazy mají zásadní vliv na činnost asembleru, musí být primární příkazy použité ve zdrojovém textu uvedeny dříve, než se vyskytne libovolná direktiva nebo instrukce asembleru.

Všeobecné příkazy lze použít kdykoliv na libovolném místě zdrojového textu. Mají pouze částečný vliv na překlad (např. potlačují tvorbu listingu v určité části programu).

Tabulka 5-1 obsahuje seznam všech příkazů asembleru s vyznačením, zda jde o primární nebo všeobecný příkaz, možné zkratky, počáteční nastavení (tj. takové, které je v činnosti, pokud se příkaz nepoužije) a stručný popis. Má-li příkaz argument, musí být ohraničen párem okrouhlých závorek.

Příkaz	Primární/ Všeobecný	Běžné nastavení	Zkratka	Význam
DATE[(datum)]	P	DATE(sytem_date)	DT	Do záhlaví stránky vstoupí datum
NODATE			NODT	
DEBUG	P	NODEBUG	DB	Do cílového kódu formátu Intel-OMF vstoupí ladící informace
NODEBUG			NODB	
ERRORPRINT[(soubor)]	P	NOERRORPRINT	EP	Určuje, zda se má či nemá tvořit soubor chybových výpisů
NOERRORPRINT			NOEP	
HEX[(soubor)]	P	NOHEX	HX	Určuje, zda se bude tvořit cílový soubor ve tvaru Intel-HEX
NOHEX			NOHX	
INCLUDE(soubor)	V	nemá význam	IC	Soubor, vstupující do zdrojového textu
LIST	V	LIST	LI	Do listingu programu vstoupí/nevstoupí řádky se zdrojovým textem
NOLIST			NOLI	

Tab. 5-1 Seznam příkazů asembleru

Příkaz	Primární/ Všeobecný	Běžné nastavení	Zkratka	Význam
MODE(typ_CPU)	P	MODE(51)	MO	Určuje typ použitého procesoru a podle toho se vybere/potlačí sada definovaných symbolů
NOMODE			NOMO	
OBJECT[(soubor)]	P	OBJECT(source.OBJ)	OJ	Určuje jméno a existenci cílového souboru typu Intel-OMF
NOOBJECT			NOOJ	
PAGING	P	PAGING	PI	Určuje, zda protokol o výstupu (listing) bude dělen na stránky
NOPAGING			NOPI	
PAGELENGTH(číslo)	P	PAGELENGTH(62)	PL	Počet řádků na stránce
PAGEWIDTH(číslo)	P	PAGEWIDTH(120)	PW	Počet znaků v řádku
PRINT[(soubor)]	P	PRINT(source.LST)	PR	Určuje jméno a existenci výstupního protokolu (listingu) programu
NOPRINT			NOPR	
REGISTERBANK(číslo)	P	REGISTERBANK(0)	RB	Určuje, které banky registrů budou programem použity
NOREGISTERBANK			NORB	
SYMBOLS	P	SYMBOLS	SB	Určuje, zda výstupní protokol bude/nebude obsahovat tabulku symbolů
NOSYMBOLS			NOSB	
TABS(číslo)	P	TABS(8)	TA	Nastavuje počet mezer, na který se budou expandovat tabelátory
TITLE(řetězec)	V	TITLE(jméno_modulu)	TT	Text, který bude přidán do záhlaví každé stránky výstupního protokolu
XREF	P	NOXREF	XR	Výstupní protokol bude/nebude obsahovat křížové odkazy na symboly
NOXREF			NOXR	

Tab. 5-1 Seznam příkazů asembleru (pokračování)

Příkaz: DATE/NODATE

Zkratka: DA/NODA

Argument: nepovinný řetězec max.9 znaků

Primární/všeobecný: Primární

Počáteční nastavení: DATE(systémový_datum)

Popis: Pokud nebude tvorba listingu potlačena příkazem NOPRINT, bude při použití příkazu DATE záhlaví každé stránky obsahovat údaj o datu překladu. Pokud se nezadá parametr, použije se systémový údaj. Parametr má obsahovat 9 znaků. Pokud jich bude méně, doplní se mezerami, pokud více, použije se prvních devět znaků.

Příkaz: DEBUG/NODEBUG

Zkratka: DB/NODB

Argument: Žádný

Primární/všeobecný: Primární

Počáteční nastavení: NODEBUG

Popis: Určuje, zda bude cílový soubor typu .OBJ obsahovat symbolické informace pro ladění programu pomocí ladících prostředků (např. emulátor ICE-5100 nebo symbolický ladící prostředek DEMON-52)

Příkaz: EJECT

Zkratka: EJ

Argument: Žádný

Primární/všeobecný: Všeobecný

Počáteční nastavení: Nemá význam

Popis: Způsobí odstránkování listingu, tj. asembler vloží do listingu znak form_feed a vypíše záhlaví stránky. Příkaz je neúčinný, pokud je v činnosti příkaz NOPRINT, NOPAGING nebo NOLIST.

Příkaz: ERRORPRINT/NOERRORPRINT

Zkratka: EP/NOEP

Argument: Volitelný, jméno souboru nebo zařízení

Primární/všeobecný: Primární

Počáteční nastavení: NOERRORPRINT

Popis: Použití příkazu ERRORPRINT způsobí tvorbu výstupního chybového protokolu. Pokud se nepoužije argument, budou chybová hlášení vypisována na systémovou konzolu.

Příkaz: HEX/NOHEX

Zkratka: HX/NOHX

Argument: Volitelný - jméno souboru

Primární/všeobecný: Primární

Počáteční nastavení: NOHEX

Popis: Příkaz HEX způsobí tvorbu cílového souboru ve tvaru Intel-HEX (současně se souborem .OBJ). Není-li použit parametr, je jméno cílového souboru stejné, jako jméno zdrojového souboru, ale má příponu .HEX.

Příkaz: INCLUDE

Zkratka: IC

Argument: Jméno souboru

Primární/všeobecný: Všeobecný

Počáteční nastavení: Nemá význam

Popis: Vloží do zdrojového textu obsah souboru, jehož jméno je parametrem příkazu. Na řádku smí být jediný příkaz INCLUDE, vložení je možně do šestnácti úrovní.

Příkaz: LIST/NOLIST

Zkratka: LI/NOLI

Argument: Žádný

Primární/všeobecný: Všeobecný

Počáteční nastavení: LIST

Popis: Příkaz LIST povoluje tvorbu listingu, NOLIST zakazuje. Bez ohledu na nastavení příkazu do listingu vstupují chybová hlášení. Účinnost přikazu LIST je potlačena příkazem NOPRINT.

Příkaz: MODE/NOMODE

Zkratka: MO/NOMO

Argument: Označení typu procesoru (51, 52, 44, ..)

Primární/všeobecný: Primární

Počáteční nastavení: MODE(51)

Popis: Příkazem lze vybrat sadu předdefinovaných symbolů v závislosti na typu procesoru, pro nějž je zdrojový program napsán. Pokud si programátor nepřeje existenci předdefinovaných symbolů, lze příkazem NOMODE potlačit jejich existenci.

Příkaz: OBJECT/NOOBBJECT

Zkratka: OJ/NOOJ

Argument: Volitelný - jméno souboru

Primární/všeobecný: Primární

Počáteční nastavení: OBJECT(zdrojový_soubor.OBJ)

Popis: Indikuje, zda se bude tvořit cílový soubor ve tvaru Intel-OMF nebo ne. Pokud se v příkazu OBJECT nepoužije parametr, je tvořen soubor stejného jména, jako je jméno zdrojového souboru, ale s příponou .OBJ.

Příkaz: PAGING/NOPAGING

Zkratka: PI/NOPI

Argument: Žádný

Primární/všeobecný: Primární

Počáteční nastavení: PAGING

Popis: Příkazem NOPAGING lze potlačit stránkování listingu. Příkazy EJECT a PAGELNGTH jsou ignorovány. Je-li v činnosti příkaz PAGING, je po každém příkazu EJECT nebo po dosažení počtu řádků nastaveném příkazem PAGELENGTH listing odstránkován vložením znaku form_feed a záhlaví.

Příkaz: PAGELENGTH

Zkratka: PL

Argument: Dekadické číslo - počet řádek na stránce

Primární/všeobecný: Primární

Počáteční nastavení: PAGELENGTH(60)

Popis: Nastavuje počet řádků na jednu stránku. Minimální možný počet řádků na stránce je 10. Řádky záhlaví se započítávají. Maximálně nastavitelný počet je 65 535.

Příkaz: PAGEWIDTH

Zkratka: PW

Argument: Dekadické číslo - počet znaků na řádku

Primární/všeobecný: Primární

Počáteční nastavení: PAGEWIDTH(120)

Popis: Nastavuje počet znaků na jeden řádek. Pokud má řádek více znaků, než je nastaveno, přeruší se výpis řadku po dosažení maximálního počtu znaků vložením dvojice znaků cr-lf (0DH-0AH) a výpis pokračuje na novém řádku.

Příkaz: PRINT/NOPRINT

Zkratka: PR/NOPR

Argument: Volitelný - jméno souboru nebo zařízení

Primární/všeobecný: Primární

Počáteční nastavení: PRINT(zdrojový_soubor.LST)

Popis: Příkaz PRINT určuje, zda bude tvořen výstupní soubor (listing), případně lze parametrem určit jméno souboru, který bude obsahovat listing nebo výstupní zařízení, na němž se listing objeví. Pokud se nepoužije parametr, použije se jméno zdrojového souboru s příponou .LST.

Příkaz: REGISTERBANK/NOREGISTERBANK

Zkratka: RB/NORB

Argument: čísla použitých bank registrů (0,1,2,3)

Primární/všeobecný: Primární

Počáteční nastavení: REGISTERBANK(0)

Popis: Indikuje, které banky registrů program používá. Tato informace má význam pro činnost linkeru, který na základě toho rezervuje paměť pro registry. Informaci lze dodatečně ovlivnit direktivou asembleru USING (viz kap. 4). Příkaz NOREGISTERBANK způsobí, že nebude rezervována žádná paměť.

Příkaz: SYMBOLS/NOSYMBOLS

Zkratka: SB/NOSB

Argument: Žádný

Primární/všeobecný: Primární

Počáteční nastavení: SYMBOLS

Popis: Příkaz určuje, zda listing bude či nebude obsahovat tabulku symbolů definovaných programátorem. Příkaz NOSYMBOLS je potlačen příkazem XREF. Příkaz NOPRINT potlačí příkaz SYMBOLS.

Příkaz:	TABS
Zkratka:	TA
Argument:	Celé číslo (1..80)
Primární/všeobecný:	Primární
Počáteční nastavení:	TABS(8)
Popis:	Nastavuje počet mezer, na který se budou expandovat tabelátory

Příkaz:	TITLE
Zkratka:	TT
Argument:	Řetězec
Primární/všeobecný:	Všeobecný
Počáteční nastavení:	TITLE()
Popis:	Použitý parametr se objeví v záhlaví každé stránky listingu.

Příkaz:	XREF/NOXREF
Zkratka:	XR/NOXR
Argument:	Žádný
Primární/všeobecný:	Primární
Počáteční nastavení:	NOXREF
Popis:	Použití příkazu XREF připojí k tabulce symbolů křížové reference jejich použití. Popis křížových odkazů je v kap. 6. Příkaz NOPRINT potlačí příkaz XREF.

Kapitola 6 - CHYBOVÁ HLÁŠENÍ A FORMÁT VÝSTUPNÍHO SOUBORU

Tato kapitola popisuje chybová hlášení asembleru ASM-52 a formát výstupního souboru (listingu).

Chybové zprávy a oprava chyb

Všechny chyby, které asembler ASM-52 detekuje, se buď zobrazí na systémovou konzolu (obvykle displej počítače) nebo se zapíší do cílového souboru na zvláštní řádek. Fatální chyby (např. chyby při spuštění asembleru) se vypíší na displej a způsobí předčasné ukončení činnosti asembleru. Chyby ve zdrojovém textu programu se zapisují do listingu a nezpůsobí předčasný konec překladu s vyjímkou chyb v primárních řídících příkazech, které rovněž způsobí předčasné ukončení činnosti asembleru.

Chybová hlášení na konzolu

Při spuštění asembleru mohou být detekovány některé chyby. Tyto chyby lze dělit na chyby příkazového řádku, chyby vstupně/výstupní a chyby vnitřní. Následující popis se týká právě těchto chyb.

Chyby vstupně/výstupní

Všechny vstupně/výstupní chyby mají jednotný formát, který vypadá takto:

```
***** ERROR 1,
ASM-52 I/O ERROR ON <typ_souboru> FILE <jméno_souboru>
<chyba_DOS>
ASM-52 terminated
```

kde typ_souboru může být:
- 'SOURCE'
- 'LISTING'
- 'OBJECT'
- 'INTEL-HEX'
- 'INCLUDE'
- 'ERRORPRINT'

jméno_souboru je jméno souboru, který chybu způsobil a

chyba_DOS je hlášení, které produkuje operační systém. Jejich přehled je uveden v každé příručce, která se distribuuje spolu s operačním systémem, zde je uveden pouze jejich telegrafický přehled:

- 'All Is Well'
- 'File not found'
- 'Path not found'
- 'Too many opens'
- 'File Acces denied'
- 'Invalid File handle'

```
'Invalid file acces'
'Invalid drive number'
'Attempt to read past End of File'
'Disk or directory full'
'File not asigned'
'File not opened'
'File not opened for Input'
'File not opened for Output'
'Write protected disk'
'Drive not ready'
'Data CRC error'
'Bad Drive request structure length'
'Disk seek error'
'Unknown media type'
'Sector not found'
'Printer out of Paper'
'Device write fault'
'Device read fault'
'Hardware failure'
'Illegal argument value'
'Stack overflow'
'Heap overfolw'
'Illegal pointer operation'
'Floating point overflow'
'Floating point underflow'
'Illegal Floating point operation'
'Overlay manager not instaled'
'Overlay file read error'
```

Chyby při vyvolání

```
Chyba číslo: 0
Text:        ***** ERROR 0,
             ASM-52 FATAL ERROR - NOT ENOUGH MEMORY
```

Tato chyba vznikne, pokud asembler ASM-52 v okamžiku spuštění nemůže uspokojit své požadavky na přidělení paměti. Může k ní dojít, pokud je asembler spuštěn na pozadí některého jiného programu (např. XTREE) nebo pokud je paměť obsazena příliš velkým množstvím rezidentních programů. V takovém případě je třeba paměť uvolnit.

```
Chyba číslo: 2
Text:        ***** ERROR 2,
             ASM-52 INVOCATION ERROR
             <příkazový řádek>
             <doplňkové hlášení>
```

kde příkazový řádek je část příkazového řádku a znak "" označuje místo, které způsobilo chybu. Doplňkové hlášení pak je bližší určení chyby a může to být jedno z následujících hlášení:

```
'LINE TOO LONG'
```

Délka vyvolávacího řádku je příliš dlouhá - je třeba jej zkrátit např. přesunutím části příkazů do zdrojového textu.

'ILLEGAL OR UNRECOGNIZABLE SOURCE FILE NAME'

Jméno zdrojového souboru neodpovídá konvencím operačního systému DOS.

'DUPLICATE USE OF THE SAME FILE NAME FOR
<typ_souboru> AND <typ_souboru>'
kde typ_souboru může být: 'SOURCE'
'LISTING'
'OBJECT'
'INTEL-HEX'
'INCLUDE'
'ERRORPRINT'

Pro dva různé typy souborů bylo použito stejné jméno. Je třeba, aby různé typy souborů měly jména odlišná.

'ILLEGAL OR UNRECOGNIZABLE FILE NAME'

Jméno souboru specifikované v některém řídícím příkaze neodpovídá konvencím operačního systému DOS.

'ILLEGAL CHARACTER IN CONTROL LINE'

V příkazovém řádku se objevil nepřípustný znak.

'BAD OR MISSING PARAMETER'

Příkaz vyžaduje parametr, avšak ten nebyl zadán a nebo je neplatný.

'MISSING RIGHT PARENTHESIS',

V příkaze je použit parametr, avšak chybí koncová uzavírací závorka.

'THIS CONTROL DOESN'T REQUIRE PARAMETER'

Je použit příkaz, který nemá mít žádný parametr, avšak parametr je přesto použit.

'BAD CONTROL'

Řídící příkaz použitý na nesprávném místě nebo v chybném kontextu (např. INCLUDE ve vyvolávacím příkazovém řádku).

'UNRECOGNIZABLE CONTROL',

Neznámý řídící příkaz.

'DUPLICATE USE OF THE SAME CONTROL'

Vícenásobné použití primárního příkazu.

Chyby zdrojového textu

Asembler ASM-52 obsahuje účinný mechanismus detekce a ohlašování chyb. Chybová hlášení obsahují informace, které umožní jednoznačně určit původce chyby. Chyby objevené ve zdrojovém textu se zapisují do listingu před chybný zdrojový řádek. Vyjímku tvoří syntaktická chyba, která se vypisuje bezprostředně za chybným řádkem.

Chybová hlášení v listingu mají jednotný formát:

```
***** ERROR nnn, LINE llll, <hlášení>
```

kde nnn je číslo chyby, zpřehledňující její vyhledání v tomto popisu,
llll je číslo řádku, na němž byla chyba nalezena a
hlášení je vysvětlující text.

Chyba syntaxe je navíc doplňena ukazatelem na chybné místo.

Chyby v listingu se dělí na:
- chyby příkazových řádků (0..99)
- chyby zdrojového textu programu (100..199)
- chyby průběhu překladu (200..299)
- chyby interní (800..999)

```
Chyba číslo: 3
Text:        ASM-52 SEMANTIC ERROR,
             <doplňkové hlášení>
```

Tato chyba je chybou zdrojového textu, je však také považována za fatální v případě, že se týká použití primárního příkazu. Následuje přehled doplňkových hlášení.

'DUPLICATE USE OF THE SAME FILE NAME FOR <typ_souboru> AND <typ_souboru>'

kde typ_souboru může být:
'SOURCE'
'LISTING'
'OBJECT'
'INTEL-HEX'
'INCLUDE'
'ERRORPRINT'

Pro dva různé typy souborů bylo použito stejné jméno. Je třeba, aby různé typy soubor měly jména odlišná.

'ILLEGAL OR UNRECOGNIZABLE FILE NAME'

Jméno souboru specifikované v některém řídícím příkaze neodpovídá konvencím operačního systému DOS.

'ILLEGAL CHARACTER IN CONTROL LINE'

V příkazovém řádku se objevil nepřípustný znak.

'BAD OR MISSING PARAMETER'

Příkaz vyžaduje parametr, avšak ten nebyl zadán a nebo je neplatný.

'MISSING RIGHT PARENTHESIS',

V příkaze je použit parametr, avšak chybí koncová uzavírací závorka.

'THIS CONTROL DOESN'T REQUIRE PARAMETER'

Je použit příkaz, který nemá mít žádný parametr, avšak parametr je přesto použit.

'BAD CONTROL'

Řídící příkaz použitý na nesprávném místě nebo v chybném kontextu (např. INCLUDE ve vyvolávacím příkazovém řádku).

'UNRECOGNIZABLE CONTROL',

Neznámý řídící příkaz.

'DUPLICATE USE OF THE SAME CONTROL'

Vícenásobné použití primárního příkazu.

Chyba číslo: 5
Text: ATTEMPT TO INCLUDE MORE THAN IN 16 LEVELS

Použití příkazu INCLUDE bylo vnořeno do již vložených INCLUDE souborů ve více než šestnácti úrovních, což je nepřípustné. Je třeba zredukovat počet vnoření INCLUDE.

Chyba číslo: 6
Text: MORE THAN ONE INCLUDE CONTROL ON SINGLE LINE

Na jednom řádku smí být pouze jeden příkaz INCLUDE. To bylo však porušeno.

Chyba číslo: 7
Text: PRIMARY CONTROL "<příkaz>" FOLLOWS NON-CONTROL LINE

Primární příkazy se smějí vyskytovat ve zdrojovém textu pouze na začátku programu před libovolnou direktivou asembleru nebo instrukcí. Je třeba všechny primární řídící příkazy soustředit do této oblasti zdrojového textu.

Chyba číslo: 8
Text: PRIMARY CONTROL "<příkaz>" FOLLOWS GENERAL CONTROL

Primární příkaz byl uveden ve zdrojovém textu až po všeobecném, což vedlo ke špatné interpretaci příkazu. Je třeba změnit pořadí příkazů.

Chyba číslo: 9
Text: MISPLACED OR UNRECOGNIZABLE CONTROL

Příkaz ve zdrojovém textu je neznámý nebo je použit v nesmyslném kontextu.

Chyba číslo: 12
Text: PAGEWIDTH BELOW MINIMUM, SET TO 80

Parametr příkazu PAGEWIDTH je menší než minimálně přípustná hranice, bylo provedeno nastavení na 80 znaků na řádek.

Chyba číslo: 13
Text: PAGEWIDTH ABOVE MAXIMUM, SET TO 255

Parametr příkazu PAGEWIDTH je větší než maximálně přípustná hranice, bylo provedeno nastavení na 255 znaků na řádek.

Chyba číslo: 14
Text: PAGELENGTH BELOW MINIMUM, SET TO 10

Parametr příkazu PAGELENGTH je menší než minimálně přípustná hranice, bylo provedeno nastavení na 10 řádků na stránku.

Chyba číslo: 15
Text: ILLEGAL USE OF CONTROL IN INVOCATION LINE

Použití nepovoleného příkazu ve vyvolávacím řádku (týká se příkazu INCLUDE).

Chyba číslo: 100
Text: SYNTAX ERROR

Syntaktická chyba. Překladač objevil na zdrojovém řádku text, který v daném kontextu není povolen. Vadný text je označen ukazatelem (____^), např.:

```
                     3      prog  segmnet code
***** _________________________________^
***** ERROR 100, Line 3, SYNTAX ERROR
```

Chyba číslo: 101
Text: ILLEGAL CHARACTER (<znak>)

V textu se vyskytl znak, který nepatří do sady povolených znaků asembleru ASM-52. Povolené znaky jsou číslice 0..9 , písmena malé i velké abecedy a..z , A..Z a znaky _ ? \ . , ; ' * / = - + & ! | ^ ( ) @ $: < > , dále znak mezera (20H), tabelace (09H) a dvojice znaků cr-lf (0DH-0AH). Znaky použité jako argument direktivy DB v apostrofech mohou být libovolné znaky nepatřící do výše uvedené množiny znaků. To platí i pro komentáře, kde se rovněž mohou vyskytovat i jiné znaky.

Chyba číslo: 102
Text: SOURCE LINE LISTING TERMINATED AT 255 CHARACTERS

Délka řádku se zdrojovým textem je v listingu programu delší než 255 znaků i když zdrojový řádek ve zdrojovém souboru má délku menší. To je většinou způsobeno nahrazením tabelačních znaků mezerami. Řádek v listingu nebude tedy celý, ale generovaný kód nebude ovlivněn.

Chyba číslo: 103
Text: ARITHMETIC OVERFLOW IN NUMERIC CONSTANT EVALUATION

Během vyčíslování výrazu došlo k aritmetickému přetečení. Výraz je třeba upravit nebo ho není možné vyčíslit.

Chyba číslo: 104
Text: ATTEMPT TO DIVIDE BY ZERO

Pokus o dělení nulou. Výraz je třeba upravit.

Chyba číslo: 105
Text: FORWARD REFERENCE OF SYMBOL "<symbol>"
ILLEGAL IN THIS EXPRESSION

Bylo použito dopředného odkazu ve výrazu, kde to není povolené (např. v definici symbolu pomocí direktivy EQU).

Chyba číslo: 106
Text: SIMPLE RELOCATABLE EXPRESSION EXPECTED
ABSOLLUTE EXPRESSION EXPECTED

Výraz v daném kontextu smí být jen jednoduše přemístitelný (např. direktiva ORG) nebo absolutní (např. direktiva AT).

Chyba číslo: 107
Text: REDEFINITION OF SYMBOL "<symbol>" NOT ALLOWED

Pokus předefinovat již existující symbol pomocí direktivy SET, ačkoliv symbol byl definován jiným způsobem. Předefinovat lze jedině symboly, které byly definovány direktivou SET.

Chyba číslo: 108
Text: SYMBOL "<symbol>" ALREADY DEFINED

Pokus od o opětné definování již existujícího symbolu (např. direktivou EQU).

Chyba číslo: 109
Text: TEXT FOUND AFTER END STATEMENT - IGNORED

Po direktivě END byl nalezen další text. Tento text asembler považuje za komentář, který nemá vliv na generovaný kód.

Chyba číslo: 110
Text: MISSING END STATEMENT

Asembler zjistil konec souboru, aniž by předtím našel direktivu END. Chyba nemá vliv na generovaný kód.

Chyba číslo: 111
Text: ILLEGAL CHARRACTER "<znak>"
IN NUMERIC CONSTANT "<konstanta>", ZERO USED

Při vyhodnocování výrazu asembler našel znak, který nepatří do sady znaku pro použitou číselnou soustavu. Ve vyčíslení výrazu se pokračuje, konstanta má hodnotu 0. Povolené znaky jsou:

"0" "1" a přípona "B" nebo "b" pro dvojkovou soustavu
"0".."7" a přípony "O", "Q", "o" a "q" pro osmičkovou soustavu
"0".."9" bez přípony nebo s příponou "D" či "d" pro desítkovou a "0".."9", "A".."F", "a".."f" s příponami "H", "h" pro šestnáctkovou soustavu.

Chyba číslo: 112
Text: LABEL NAME "<symbol>" ALREADY USED

Použité návěští již existuje, je třeba použít pro návěští jiný symbol.

Chyba číslo: 114
Text: UNDEFINED SYMBOL "<symbol>"

Symbol použitý ve výrazu nebo jako argument direktivy není definován.

Chyba číslo: 115
Text: VALUE OF EXPRESSION NEAR OPERAND "<operand>"
TRUNCATED TO BYTE

Hodnota výrazu je vyšší než 255, ačkoliv daný kontext vyžaduje hodnotu v rozsahu 0..255 (byte). Vyšší byte hodnoty byl doplněn nulami a takto upravená hodnota byla použita.

Chyba číslo: 116
Text: DIRECTIVE "<příkaz>" ILLEGAL IN THIS SEGMENT

Použití direktivy, která je v aktivním segmentu nepřípustná (např. direktiva DW v segmentu třídy BIT atp.).

Chyba číslo: 117
Text: STRING TERMINATED BY END-OF-LINE

Argument direktivy DB, DW nebo instrukce je znakový řetězec, avšak není ukončen apostrofem.

Chyba číslo: 118
Text: STRING LONGER THAN 2 CHARACTERS ILLEGAL IN THIS CONTEXT

Jako součást výrazu je použit znakový řetězec s délkou větší, než dva znaky, ačkoliv to v daném kontextu není možné. Znakové řetězce s délkou větší než dva znaky jsou možné pouze jako argument direktivy DB.

Chyba číslo: 119
Text: SOURCE LINE CANNOT EXCEED 255 CHARACTERS

Délka řádku zdrojového textu je větší než 255 znaků, což je nepřípustné. Asembler ignoroval všechny znaky následující po 255-tém znaku až do konce řádku.

Chyba číslo: 120
Text: DESTINATION ADDRESS OUT OF RANGE FOR RELATIVE REFERENCE
DESTINATION ADDRESS OUT OF RANGE FOR INBLOCK REFERENCE

Cílová adresa skoku nebo volání není dosažitelná pomocí relativního nebo blokového adresování, tuto část programu je třeba přepsat.

Chyba číslo: 121
Text: SEGMENT NAME EXPECTED

Argumentem direktivy RSEG musí být symbol, který byl definován direktivou SEGMENT.

Chyba číslo: 122
Text: REFERENCE TO OTHER SEGMENT

Výsledný výraz má příslušnost k jinému segmentu, než je aktivní segment, což je nepřípustné (např. v direktivě ORG).

Chyba číslo: 123
Text: BIT SEGMENT ADDRESS EXPECTED
CODE SEGMENT ADDRESS EXPECTED
DATA SEGMENT ADDRESS EXPECTED
IDATA SEGMENT ADDRESS EXPECTED
XDATA SEGMENT ADDRESS EXPECTED

Výraz se měl vyčíslit jako adresa v očekávaném segmentu oblasti paměti, ale ve skutečnosti je výsledkem výrazu adresa jiné paměťové oblasti.

Chyba číslo: 125
Text: LOCATION IS NOT BIT ADDRESSABLE

V daném kontextu se požaduje výraz, který je adresou bitu, avšak po vyčíslení výrazu se zjistilo, že odkazuje na adresu paměti dat, která bitové adresování neumožňuje.

Chyba číslo: 126
Text: ILLEGAL BIT OFFSET

Výraz použitý pro offset bitu má hodnotu větší, než sedm (např. v instrukci CLR ACC.9).

Chyba číslo: 127
Text: PUBLIC ATTRIBUTE ILLEGAL FOR SYMBOL "<symbol>"

V seznamu PUBLIC se vyskytl symbol, který nemůže být public (např. posloupnost direktiv REG_7 EQU R7
PUBLIC REG_7)
Jako PUBLIC mohou být deklarovány ty symboly, které vyjadřují adresu v segmentu paměti, případně číselné konstanty.

Chyba číslo: 128
Text: EXTERNAL REFERENCE ILLEGAL IN THIS CONTEXT
SEGMENT REFERENCE ILLEGAL IN THIS CONTEXT

Použitý odkaz na segment paměti nebo externí symbol není v daném kontextu povolen.

Chyba číslo: 129
Text: ATTEMPT TO SET LOCATION COUNTER BELOW SEGMENT BASE

Pokus nastavit čítač lokací na nižší adresu, než je nastavena báze segmentu (např. direktivy CSEG AT 23H
ORG 0BH).

Chyba číslo: 130
Text: LOCATION COUNTER OVERFLOW

Pokus o nastavení čítače lokací na vyšší hodnotu, než je v daném segmentu (případně ještě podmíněno i typem přemístění) možné. Limitní hodnoty jsou:

65535 (0FFFFH) pro segmenty typu CODE a XDATA (absolutní nebo přemístitelné s relokací typu UNIT či PAGE)

2047 (7FFH) pro přemístitelné segmenty typu CODE s relokací typu INBLOCK

255 (0FFH) pro přemístitelné segmenty typu CODE a XDATA s relokací typu INPAGE

255 (0FFH) pro přemístitelné i absolutní segmenty typu DATA s relokací typu UNIT

15 (0FH) pro přemístitelné segmenty typu DATA s relokací typu BITADDRESSABLE

127 (7FH) pro přemístitelné i absolutní segmenty typu IDATA pro procesory řady 8051

191 (0BFH) pro přemístitelné i absolutní segmenty typu IDATA pro procesory řady 8044

255 (0FFH) pro přemístitelné i absolutní segmenty typu IDATA pro procesory řady 8052

127 (7FH) pro přemístitelné i absolutní segmenty typu BIT

Chyba číslo: 131
Text: INPAGE RELOCATED SEGMENT OVERFLOW
INBLOCK RELOCATED SEGMENT OVERFLOW
BITADDRESSABLE RELOCATED SEGMENT OVERFLOW

Přetečení čítače lokací přemístitelného segmentu s typem relokace uvedeným na začátku chybového hlášení. Limity pro čítače lokací v jednotlivých přemístitelných segmentech jsou uvedeny ve vysvětlivce k chybě 130 .

Chyba číslo: 132
Text: TOO MANY RELOCATABLE SEGMENTS
TOO MANY EXTERNAL SYMBOLS

Uživatelský program obsahuje příliš mnoho externích deklarací nebo deklarací segmentů. V obou případech je povolené maximum 255.

Chyba číslo: 133
Text: ILLEGAL RELOCATABLE EXPRESSION

Přemístitelný výraz je v rozporu s pravidly, uvedenými v kapitole 2 této příručky.

Chyba číslo: 134
Text: ILLEGAL RELOCATION TYPE FOR CURRENT SEGMENT

Přemístitelný segment deklarovaný s nepřípustným typem přemístění. Povolené kombinace popisuje kapitola 4.

Chyba číslo: 135
Text: ILLEGAL REGISTER BANK NUMBER

Nepovolené číslo banky registrů. Možné jsou pouze výrazy, jejichž výsledek dá číslo 0, 1, 2 nebo 3 v direktivě USING a nebo konstanty 0, 1, 2 či 3 v příkaze REGISTERBANK.

Chyba číslo: 136
Text: SYMBOL "<symbol>" ILLEGAL IN MODE <mode>

Symbol použitý ve zdrojovém programu sice existuje jako předdefinovaný symbol, ale vybraný typ procesoru (příkazem MODE) tento symbol nezná.

Chyba číslo: 137
Text: SYMBOL "<symbol>" IS PREDEFINED IN MODE <mode>, REDEFINITION NOT ALLOWED

Pokus o předefinování jednoho z předdefinovaných symbolů pro speciální funkční registry nebo bitu těchto registrů.

Chyba číslo: 139
Text: REDEFINITION OF RESERVED SYMBOL NOT ALLOWED

Pokus o předefinování symbolu, který je rezervovaný pro použití asemblerem ASM-52.

Chyba číslo: 140
Text: BYTE OF BIT ADDRESS NOT IN BIT-ADDRESSABLE SEGMENT

Byte použitý jako báze bitové adresy není v bitově adresovatelném datovém segmentu.

Vnitřní chyby asembleru

Chyba číslo: 201
Text: PHASE ERROR

Chyba průchodu. Hodnota výrazu nebo návěští v prvním průchodu byla jiná než ve druhém průchodu. Tato chyba by se neměla vůbec vyskytnout. Objeví-li se, spojte se s Vaším dealerem nebo přímo s firmou PROMIS.

Chyba číslo: 202
Text: EXPRESSION TOO COMPLEX

Příliš složitý výraz k vyčíslení. Výskyt chyby je málo pravděpodobný, může ale nastat, použije-li se v programu např. více než šestnáct úrovní uzávorkování výrazu. V takovém případě je třeba výraz zjednodušit.

Chyba číslo: 800
Text: ASM-52 INTERNAL ERROR (CROSS-REFERENCE TABLE GENERATION)
Chyba číslo: 801
Text: ASM-52 INTERNAL ERROR (CROSS-REFERENCE TABLE PARSING)
Chyba číslo: 900
Text: ASM-52 INTERNAL ERROR (UNKNOWN CLASS)
Chyba číslo: 901
Text: ASM-52 INTERNAL ERROR (SYMBOL TABLE GENERATION)
Chyba číslo: 902
Text: ASM-52 INTERNAL ERROR (SYMBOL TABLE PARSING)

Výše uvedených pět chyb se nesmí při překladu objevit. Pokud se tak stane, není cosi v pořádku s Vaší kopií asembleru ASM-52. Zkuste si asembler znovu nahrát z distribučního média. Pokud problém přetrvává, spojte se s Vaším dealerem nebo přímo s firmou PROMIS.

Chyba číslo: 903
Text: ASM-52 INTERNAL ERROR (SYMBOL TABLE FULL)

Výskyt této chyby je rovněž nepravděpodobný, ale může být způsoben příliš velkým počtem symbolů, definovaných ve zdrojovém textu. V takovém případě je třeba počet symbolů omezit nebo program rozdělit na více modulů a ty později spojit linkerem. Vzhledem k tomu, že kapacita tabulky symbolů při průměrné délce symbolu 10 znaků je více než 3.000 symbolů, je výskyt této chyby prakticky vyloučen.

Formát výstupního souboru

Výstupní soubor (listing) je produkován asemblerem tehdy, není-li jeho tvorba potlačena příkazem NOPRINT. Výstupní soubor obsahuje záhlaví s informacemi o asembleru, kopii textu zdrojového souboru (a případně i textu souborů vnořených příkazem INCLUDE) s očíslovanými řádky a s generovaným kódem, hlášení o chybách (pokud se vyskytly) a nakonec tabulku symbolů (nebyla-li její tvorba potlačena příkazem NOSYMBOLS) a výpis křížových odkazů (byl-li požadován příkazem XREF).

Strukturu listingu lze demonstrovat pomocí následujícího krátkého listingu programu.

```
DOS 5.0 MCS-51/52/44 Assembler Vers. 1.1
OBJECT MODULE PLACED IN C:\UTIL51\TIMER0.OBJ
ASEMBLER INVOKED BY: C:\ASM52\ASM52.EXE C:\UTIL51\TIMER0.ASM XREF
PL(60) PW(64) TABS(4)

LOC  OBJ            LINE     SOURCE

                       1
                       2     NAME TIMER_0
                       3
----                   4     CSEG AT 0BH
                       5
000B 020000   R        6             LJMP    TIMER_0_BEGIN
                       7
                       8
                       9     TIMER SEGMENT CODE
                      10
----                  11     RSEG TIMER
                      12
                      13     EXTRN DATA(TIME_COUNT)
                      14
    0000              15     INIT_TIMER:
                      16     PUBLIC  INIT_TIMER
0000 85F08A           17             MOV     TL0,LOW(-10000)
0003 85D88C           18             MOV     TH0,HIGH(-10000)
0006 22               19             RET
                      20
    0007              21     TIMER_0_BEGIN:
0007 C0E0             22             PUSH    ACC
0009 120000           23             CALL    INIT_TIMER
000C 0500     E       24             INC     TIME_COUNT
000E E500     E       25             MOV     A,TIME_COUNT
0010 B46403           26             CJNE    A,100,TIMER_0_END
0013 E4               27             CLR     A
0014 F500     E       28             MOV     TIME_COUNT,A
    0016              29     TIMER_0_END:
0016 D0E0             30             POP     ACC
0018 32               31             RETI
                      32     END
```



```
S Y M B O L   T A B L E   L I S T I N G
---------------------------------------

N A M E           TYPE   V A L U E     ATTRIBUTES AND REFERENCES

INIT_TIMER . . . C ADDR   0000H R PUB   SEG=TIMER  15  23
TIMER. . . . . . C SEG    0019H         REL=UNIT  9  11
TIMER_0. . . . .  ----    ----          2
TIMER_0_BEGIN. . C ADDR   0007H R       SEG=TIMER  21  6
TIMER_0_END. . . C ADDR   0016H R       SEG=TIMER  29  26
TIME_COUNT . . . D ADDR   ----    EXT   13  24  25  28

REGISTER BANK USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Záhlaví listingu

Každá stránka listingu má stejné záhlaví tvořené identifikací asembleru, titulkem, datumem a číslem stránky (titulek i datum lze potlačit, viz příkazy kap. 5). Pokud není specifikován žádný titulek, použije se jméno modulu (dané direktivou NAME; chybí-li, jménem zdrojového textu bez přípony).

Vyjímku tvoří první stránka, obsahující údaje o verzi operačního systému a asembleru, dále o umístění cílového souboru a příkazový řádek, kterým byl asembler spuštěn (viz. obr 6-1).

```
MCS-51/52/44 Assembler    TIMER_0          02/23/1992     Page 1

DOS 5.0 MCS-51/52/44 Assembler Vers. 1.1
OBJECT MODULE PLACED IN C:\UTIL51\TIMER0.OBJ
ASEMBLER INVOKED BY: C:\ASM52\ASM52.EXE C:\UTIL51\TIMER0.ASM XREF
PL(60) PW(64) TABS(4)
```

Obr. 6-1 Záhlaví první stránky programu

```
LOC  OBJ            LINE     SOURCE
                       1
                       2     NAME TIMER_0
                       3
----                   4     CSEG AT 0BH
                       5
000B 020000   R        6             LJMP    TIMER_0_BEGIN
                       7
                       8
                       9     TIMER SEGMENT CODE
                      10
----                  11     RSEG TIMER
                      12
                      13     EXTRN DATA(TIME_COUNT)
                      14
    0000              15     INIT_TIMER:
                      16     PUBLIC  INIT_TIMER
0000 85F08A           17             MOV     TL0,LOW(-10000)
0003 85D88C           18             MOV     TH0,HIGH(-10000)
0006 22               19             RET
                      20
    0007              21     TIMER_0_BEGIN:
0007 C0E0             22             PUSH    ACC
0009 120000           23             CALL    INIT_TIMER
000C 0500     E       24             INC     TIME_COUNT
000E E500     E       25             MOV     A,TIME_COUNT
0010 B46403           26             CJNE    A,100,TIMER_0_END
0013 E4               27             CLR     A
0014 F500     E       28             MOV     TIME_COUNT,A
    0016              29     TIMER_0_END:
0016 D0E0             30             POP     ACC
0018 32               31             RETI
                      32     END
```

Obr. 6-2 Listing zdrojového textu

Listing zdrojového textu

Hlavní část výstupního souboru je tvořena upraveným listingem zdrojového textu podle obr. 6-2.

Formulář je předtištěn záhlavím, jehož položky mají následující význam:

LOC - je momentální stav čítače lokací uvnitř přemístitelného nebo absolutního segmentu (zobrazení v šestnáctkové soustavě).
OBJ - je produkovaný cílový kód (zobrazení v šestnáctkové soustavě).
LINE - obsahuje údaj o úrovni vnoření pomocí INCLUDE a číslo zdrojového řádku od počátku souboru. Čísla řádků jsou zobrazována v desítkové soustavě.
SOURCE - je kopie textu zdrojového řádku. Může zabírat více řádků podle toho, jak je listing pomocí příkazů formátován.

Direktivy DB a DW mají pole OBJ uspořádáno v závislosti na délce produkovaného kódu.

Direktivy ORG, DS a DBIT neinicializují paměť, nepoužívají tedy pole OBJ. Ovlivňují čítač lokací, zobrazující se v poli OBJ.

Direktivy definující symboly (EQU, SET, BIT, CODE, DATA, IDATA, XDATA a návěští nepoužívají pole LOC ani OBJ, jimi definované hodnoty se zobrazí uprostřed polí LOC a OBJ. Pokud direktiva definuje alternativní jméno registru, vypíše se místo hodnoty řetězec 'REG'.

Tabulka symbolů a závěrečné hlášení

Listing programu je ukončen tabulkou symbolů (pokud se generovala) a závěrečným hlášením o použitých bankách registrů a počtu detekovaných chyb.

Tabulka symbolů začíná vždy na nové stránce záhlavím tabulky symbolů dle obr. 6-3.

```
S Y M B O L    T A B L E    L I S T I N G
-----------------------------------------

N A M E             TYPE   V A L U E     ATTRIBUTES AND REFERENCES

INIT_TIMER . . . C ADDR   0000H R PUB   SEG=TIMER  15  23
TIMER. . . . . . C SEG    0019H         REL=UNIT  9  11
TIMER_0. . . . .  ----    ----          2
TIMER_0_BEGIN. . C ADDR   0007H R       SEG=TIMER  21  6
TIMER_0_END. . . C ADDR   0016H R       SEG=TIMER  29  26
TIME_COUNT . . . D ADDR   ----    EXT   13  24  25  28

REGISTER BANK USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Obr. 6-3 Tabulka symbolů a závěrečné hlášení

Pole NAME obsahuje jména použitých symbolů. Její šířka je závislá na počtu znaků nejdelšího symbolu.

Pole TYPE obsahuje údaje o typu symbolu. ADDR značí adresu v paměti, NUMB je číslo bez segmentové příslušnosti, REG je symbolický název registru a SEG je symbol, označující paměťový segment. Symboly typu ADR a SEG jsou předcházeny údajem o třídě segmentu. Tento údaj je tvořen jedním písmenem:

B - segmenty třídy BIT
C - segmenty třídy CODE
D - segmenty třídy DATA
I - segmenty třídy IDATA
X - segmenty třídy XDATA

Pole VALUE obsahuje číselný ekvivalent symbolu (absolutní symbol) ukončený znakem A, offset symbolu v segmentu (přemístitelný symbol) ukončený znakem R a nebo délku segmentu pro symboly type SEG bez následného označení.

Bitové adresy jsou vypsány pomocí složené notace báze bitu (datová adresa bitově adresovatelného byte), následuje oddělovací tečka a po ní offset bitu (číslo 0..7).

Pokud je symbol deklarován jako PUBLIC, následuje ještě identifikátor PUB a naopak pro externí symbol identifikátor EXT.

Pole ATTRIBUTES obsahuje ještě dodatečné informace o symbolu. Tyto informace jsou údaje o typu přemístění a příslušnosti k segmentu (u přemístitelných symbolů).

Pokud byly vyžádány křížové reference, obsahuje toto pole ještě údaje o výskytu symbolu v programu. První je vždy číslo řádku na němž byl symbol definován (je předcházeno navíc znakem '') a za ním následují čísla řádků, které obsahují instrukce používající symbol jako operand.

Listing je ukončen hlášením o počtu použitých bank registrů a počtu zjištěných chyb.

PŘÍLOHA 1
ABECEDNÍ SEZNAM INSTRUKCÍ

Tato příloha obsahuje seznam všech instrukcí procesoru řady MCS-51 v abecedním pořadí.

Každá instrukce je popsána pomocí mnemokódu, schematického znázornění činnosti, seznamu operandů a délky instrukce vyjádřené jednak délkou instrukce v paměti programu (byte) a jednak dobou provádění instrukce (cycle).

Následující seznam osvětluje využití zkratek a symbolů, použitých v popisu instrukcí.

Symbol	Význam
A	Akumulátor
AB	Dvojice registrů pro násobení a dělení
bit_adr	Adresa bitu u obvodů 8051
b b b b b b b b	Adresa bitu vyjádřená jako operand instrukce
adresa bloku	11-bitová adresa uvnitř 2KB bloku
aaa aaaaaaaa	11-bitová adresa uvnitř 2KB bloku vyjádřená jako operační kód instrukce a operandu
B	Registr pro násobení a dělení
rel.offset	8-bitové relativní posunutí ve tvaru čísla ve dvojkovém doplňkovém kódu
o o o o o o o o	relativní posunutí vyjádřené jako operační kód operandu
C	Příznak přenosu
<adr>	Adresa v paměti programu
l l l l l l l l	Adresa v paměti programu vyjádřená jako operand instrukce LJMP nebo LCALL
data	Přímý datový operand (konstanta)
d d d d d d d d	Přímý datový operand jako operand instrukce
data_adr	8-bitová adresa v paměti RAM na čipu
m m m m m m m m	8-bitová adresa v paměti RAM na čipu jako operand instrukce
DPTR	Ukazatel dat
PC	Programový čítač (čítač instrukcí)
Rr	Registr, r = 0 - 7 (celé číslo), při nepřímém adresování r = 0 nebo 1
r nebo r r r	Binární vyjádření čísla registru jako součást operandu instrukce
SP	Ukazatel zásobníku
HIGH	Slabika vyšších řádů
LOW	Slabika nižších řádů
AND	Logický součin
NOT	Negace
OR	Logický součet
XOR	Součet modulo 2 (exkluzívní součet)
+	Plus
-	Mínus
*	Násobení
/	Dělení
=	Rovná se

Symbol	Význam
(X)	Obsah prvku X
((X))	Obsah paměťového místa adresovaného obsahem prvku X
<	Menší než
>	Větší než
<>	Nerovná se
<-	Přesun hodnoty ne směru šipky
0 1	Číslice nula a jedna
#	Označení přímých dat v instrukci
@	Označení nepřímé adresy v instrukci

Následující tabulkaobsahuje abecední seznam instrukcí.

Mnemonika a činnost instrukce	Operační kód (binárně)	Délka byt	Délka cyc	Příznaky
ACALL <adr> (PC) <- (PC)+2 (SP) <- (SP)+1 ((SP)) <- LOW(PC) (SP) <- (SP)+1 ((SP)) <- HIGH(PC) (PC)0-10 <- adresa bloku	a a a 1 0 0 0 1 a a a a a a a a	2	2	
ADD A,#data (A) <- (A) + data	0 0 1 0 0 1 0 0 d d d d d d d d	2	1	C AC OV P
ADD A,@Rr (A) <- (A) + ((Rr))	0 0 1 0 0 1 1 r	1	1	C AC OV P
ADD A,Rr (A) <- (A) + (Rr)	0 0 1 0 1 r r r	1	1	C AC OV P
ADD A,data_adr (A) <- (A) + (data_adr)	0 0 1 0 0 1 0 1	2	1	C AC OV P
ADDC A,#data (A) <- (A) + (C) + data	0 0 1 1 0 1 0 0	2	1	C AC OV P
ADDC A,@Rr (A) <- (A) + (C) +((Rr))	0 0 1 1 0 1 1 r	1	1	C AC OV P
ADDC A,Rr (A) <- (A) + (C) + (Rr)	0 0 1 1 1 r r r	1	1	C AC OV P
ADDC A,data_adr (A)<-(A)+(C)+(data_adr)	0 0 1 1 0 1 0 1 d d d d d d d d	2	1	C AC OV P
AJMP <adr> (PC) <- (PC) + 2 (PC)0-10 <- adresa bloku	a a a 0 0 0 0 1 a a a a a a a a	2	2	

Mnemonika a činnost instrukce	Operační kód (binárně)	Délka byt	Délka cyc	Příznaky
ANL A,#data (A) <- (A) AND data	0 1 0 1 0 1 0 0 d d d d d d d d	2	1	P
ANL A,@Rr (A) <- (A) AND ((Rr))	0 1 0 1 0 1 1 r	1	1	P
ANL A,Rr (A) <- (A) AND (Rr)	0 1 0 1 1 r r r	1	1	P
ANL A,data_adr (A) <- (A) OR (data_adr)	0 1 0 1 0 1 0 1 d d d d d d d d	2	1	P
ANL C,bit_adr (C) <- (C) AND (bit_adr)	1 0 0 0 0 0 1 0 b b b b b b b b	2	1	C
ANL C,/bit_adr (C)<-(C) AND NOT(bit_adr)	1 0 1 1 0 0 0 0 b b b b b b b b	2	2	C
ANL data_adr,#data (data_adr)<-(data_adr) AND data	0 1 0 1 0 0 1 1 m m m m m m m m d d d d d d d d	3	2	
ANL data_adr,A (data_adr)<-(data_adr) AND A	0 1 0 1 0 0 1 0 m m m m m m m m	2	1	

CJNE @Rr,#data,<adr>
(PC) <- (PC) + 3
IF ((Rr))<>data
THEN (PC) <- (PC)+rel.offset
IF ((Rr)) < data THEN (C)<-1
ELSE (C) <- 0
	1 0 1 1 0 1 1 r
	d d d d d d d d
	o o o o o o o o
		3	2	C

CJNE A,#data,<adr>
(PC) <- (PC) + 3
IF (A)<>data
THEN (PC) <- (PC)+rel.offset
IF (A) < data THEN (C) <- 1
ELSE (C) <- 0
	1 0 1 1 0 1 0 0
	d d d d d d d d
	o o o o o o o o
		3	2	C

CJNE A,data_adr,<adr>
(PC) <- (PC) + 3
IF (A) <> (data_adr)
THEN (PC) <- (PC)+rel.offset
IF (A) < (data_adr)
THEN (C) <- 1
ELSE (C) <- 0
	1 0 1 1 0 1 0 1
	m m m m m m m m
	o o o o o o o o
		3	2	C

CJNE Rr,#data,<adr>
(PC) <- (PC) + 3
IF (Rr) <> data
THEN (PC)<-(PC)+rel.offset
IF (Rr) < data THEN (C)<-1
ELSE (C) <- 0
	1 0 1 1 1 r r r
	d d d d d d d d
	o o o o o o o o
		3	2	C

Mnemonika a činnost instrukce	Operační kód (binárně)	Délka byt	Délka cyc	Příznaky
CLR A (A) <- 0	1 1 1 0 0 1 0 0	1	1	P
CLR C (C) <- 0	1 1 0 0 0 0 1 1	1	1	C
CLR bit_adr (bit_adr) <- 0	1 1 0 0 0 0 1 0	2	1	
CPL A (A) <- NOT (A)	1 1 1 1 0 1 0 0	1	1	
CPL C (C) <- NOT (C)	1 0 1 1 0 0 1 1	1	1	C
CPL bit_adr (bit_adr) <- NOT (bit_adr)	1 0 1 1 0 0 1 0	2	1	
DA A Upraví obsah akumulátoru na desítkový tvar.	1 1 0 1 0 1 0 0	1	1	C P
DEC @Rr ((Rr)) <-((Rr)) - 1	0 0 0 1 0 1 1 r	1	1	
DEC A (A) <- (A) - 1	0 0 0 1 0 1 0 0	1	1	P
DEC Rr (Rr) <- (Rr) - 1	0 0 0 1 1 r r r	1	1	
DEC data_adr (data_adr)<-(data_adr)-1	0 0 0 1 0 1 0 1 m m m m m m m m	2	1	
DIV AB (AB) <- (A) / (B)	1 0 0 0 0 1 0 0	1	4	C OV P
DJNZ Rr,<adr> (PC) <- (PC)+2 (Rr) <- (Rr) -1 IF (Rr)<>0 THEN (PC) <- (PC)+rel.offset	1 1 0 1 1 r r r o o o o o o o o	2	2	
DJNZ data_adr,<adr> (PC) <- (PC)+3 (data_adr)<-(data_adr)-1 IF (data_adr)<>0 THEN (PC) <- (PC)+rel.offset	1 1 0 1 0 1 0 1 m m m m m m m m o o o o o o o o	3	2	
INC @Rr ((Rr)) <- ((Rr)) +1	0 0 0 0 0 1 1 r	1	1	
INC A (A) <- (A) +1	0 0 0 0 0 1 0 0	1	1	P

Mnemonika a činnost instrukce	Operační kód (binárně)	Délka byt	Délka cyc	Příznaky
INC DPTR (DPTR) <- (DPTR) +1	1 0 1 0 0 0 1 1	1	2	
INC Rr (Rr) <- (Rr) +1	0 0 0 0 1 r r r	1	1	
INC data_adr (data_adr) <- (data_adr) +1	0 0 0 0 0 1 0 1 m m m m m m m m	2	1	
JB bit_adr,<adr> (PC) <- (PC) + 3 IF (bit_adr) = 1 THEN (PC) <- (PC)+rel.offset	0 0 1 0 0 0 0 0 b b b b b b b b o o o o o o o o	3	2	
JBC bit_adr,<adr> (PC) <- (PC) + 3 IF (bit_adr) = 1 THEN (bit_adr) <- 0 (PC) <- (PC) + rel.offset	0 0 0 1 0 0 0 0 b b b b b b b b o o o o o o o o	3	2	
JC <adr> (PC) <- (PC) + 2 IF (C) = 1 THEN (PC) <- (PC)+rel.offset	0 1 0 0 0 0 0 0 o o o o o o o o	2	2	
JMP @A+DPTR (PC) <- (A) + (DPTR)	0 1 1 1 0 0 1 1	1	2	
JNB bit_adr,<adr> PC <- (PC) + 3 IF NOT (bit_adr) THEN (PC) <- (PC)+rel.offset	0 0 1 1 0 0 0 0 b b b b b b b b o o o o o o o o	3	2	
JNC <adr> PC <- (PC) + 2 IF (C) = 0 THEN (PC) <- (PC)+rel.offset	0 1 0 1 0 0 0 0 o o o o o o o o	2	2	
JNZ <adr> PC <- (PC) + 2 IF (A) <> 0 THEN (PC) <- (PC)+rel.offset	0 1 1 1 0 0 0 0 o o o o o o o o	2	2	
JZ <adr> PC <- (PC) + 2 IF (A) = 0 THEN (PC) <- (PC)+rel.offset	0 1 1 0 0 0 0 0 o o o o o o o o	2	2	
LCALL <adr> (PC) <- (PC)+3 (SP) <- (SP)+1 ((SP)) <- LOW(PC) (SP) <- (SP)+1 ((SP)) <- HIGH(PC) (PC) <- <adr>	0 0 0 1 0 0 1 0 l l l l l l l l l l l l l l l l	3	2	

Mnemonika a činnost instrukce	Operační kód (binárně)	Délka byt	Délka cyc	Příznaky
LJMP <adr> (PC) <- <adr>	0 0 0 0 0 0 1 0 l l l l l l l l l l l l l l l l	3	2	
MOV @Rr,#data ((Rr)) <- data	0 1 1 1 0 1 1 r d d d d d d d d	2	1	
MOV @Rr,A ((Rr)) <- (A)	1 1 1 1 0 1 1 r	1	1	
MOV @Rr,data_adr ((Rr)) <- (data_adr)	1 0 1 0 0 1 1 r m m m m m m m m	2	2	
MOV A,#data (A) <- data	0 1 1 1 0 1 0 0 d d d d d d d d	2	1	P
MOV A,@Rr (A) <- ((Rr))	1 1 1 0 0 1 1 r	1	1	P
MOV A,Rr (A) <- (Rr)	1 1 1 0 1 r r r	1	1	P
MOV A,data_adr (A) <- (data_adr)	1 1 1 0 0 1 0 1 m m m m m m m m	2	1	P
MOV C,bit_adr (C) <- (bit_adr)	1 0 1 0 0 0 1 0 b b b b b b b b	2	2	C
MOV DPTR,#data (DPTR) <- data	1 0 0 1 0 0 0 0 d d d d d d d d d d d d d d d d	3	2	
MOV Rr,#data (Rr) <- data	0 1 1 1 1 r r r d d d d d d d d	2	1	
MOV Rr,A (Rr) <- (A)	1 1 1 1 1 r r r	1	1	
MOV Rr,data_adr (Rr) <- (data_adr)	1 0 1 0 1 r r r m m m m m m m m	2	2	
MOV bit_adr,C (bit_adr) <- (C)	1 0 0 1 0 0 1 0 b b b b b b b b	2	2	
MOV data_adr,#data (data_adr) <- data	0 1 1 1 0 1 0 1 m m m m m m m m d d d d d d d d	3	2	
MOV data_adr,@Rr (data_adr) <- ((Rr))	1 0 0 0 0 1 1 r m m m m m m m m	2	2	
MOV data_adr,A (data_adr) <- (A)	1 1 1 1 0 1 0 1 m m m m m m m m	2	1	

Mnemonika a činnost instrukce	Operační kód (binárně)	Délka byt	Délka cyc	Příznaky
MOV data_adr,Rr (data_adr) <- ((Rr))	1 0 0 0 1 r r r m m m m m m m m	2	2	
MOV data_adr1,data_adr2 (data_adr1) <- (data_adr2)	1 0 0 0 0 1 0 1 m m m m m m m m m m m m m m m m	3	2	
MOVC A,@A+DPTR (A) <- ((A) + (DPTR))	1 0 0 1 0 0 1 1	1	2	P
MOVC A,@A+PC (PC) <- (PC) + 1 (A) <- ((A) + (PC))	1 0 0 0 0 0 1 1	1	2	P
MOVX @DPTR,A ((DPTR)) <- (A)	1 1 1 1 0 0 0 0	1	2	
MOVX @Rr,A ((Rr)) <- (A)	1 1 1 1 0 0 1 r	1	2	
MOVX A,@DPTR (A) <- ((DPTR))	1 1 1 0 0 0 0 0	1	2	P
MOVX A,@Rr (A) <- ((Rr))	1 1 1 0 0 0 1 r	1	2	P
MUL AB (AB) <- (A) * (B)	1 0 1 0 0 1 0 0	1	4	C OV P
NOP žádná	0 0 0 0 0 0 0 0	1	1	
ORL A,#data (A) <- (A) OR data	0 1 0 0 0 1 0 0 d d d d d d d d	2	1	P
ORL A,@Rr (A) <- (A) OR ((Rr))	0 1 0 0 0 1 1 r	1	1	P
ORL A,Rr (A) <- (A) OR (Rr)	0 1 0 0 1 r r r	1	1	P
ORL A,data_adr (A) <- (A) OR (data_adr)	0 1 0 0 0 1 0 1 m m m m m m m m	2	1	P
ORL C,bit_adr (C) <- (C) OR (bit_adr)	0 1 1 1 0 0 1 0 b b b b b b b b	2	1	C
ORL C,bit_adr (C)<-(C) OR NOT(bit_adr)	1 0 1 0 0 0 0 0 b b b b b b b b	2	2	C
ORL data_adr,#data (data_adr)<-(data_adr) OR data	0 1 0 0 0 0 1 1 m m m m m m m m d d d d d d d d	3	2	

Mnemonika a činnost instrukce	Operační kód (binárně)	Délka byt	Délka cyc	Příznaky
ORL data_adr,A (data_adr)<-(data_adr) OR (A)	0 1 0 0 0 0 1 0 m m m m m m m m	2	1	
POP data_adr (data_adr) <- ((SP)) (SP) <- (SP) -1	1 1 0 1 0 0 0 0 m m m m m m m m	2	2	
PUSH data_adr (SP) <- (SP) + 1 (SP) <- (data_adr)	1 1 0 0 0 0 0 0 m m m m m m m m	2	2	
RET (PC HIGH) <- ((SP)) (SP) <- (SP) - 1 (PC LOW) <- ((SP)) (SP) <- (SP) - 1	0 0 1 0 0 0 1 0	1	2	
RETI (PC HIGH) <- ((SP)) (SP) <- (SP) - 1 (PC LOW) <- ((SP)) (SP) <- (SP) - 1	0 0 1 1 0 0 1 0	1	2	
RL A rotace akumulátoru vlevo	0 0 1 0 0 0 1 1	1	1	
RLC A rotace akumulátoru a příznaku přenosu vlevo	0 0 1 1 0 0 1 1	1	1	C P
RR A rotace akumulátoru vpravo	0 0 0 0 0 0 1 1	1	1	
RRC A rotace akumulátoru a příznaku přenosu vpravo	0 0 0 1 0 0 1 1	1	1	C P
SETB C (C) <- 1	1 1 0 1 0 0 1 1	1	1	C
SETB bit_adr (bit_adr) <- 1	1 1 0 1 0 0 1 0 b b b b b b b b	2	1	
SJMP <adr> (PC) <- (PC) + 2 (PC) <- (PC) + rel.offset	1 0 0 0 0 0 0 0 o o o o o o o o	2	2	
SUBB A,#data (A) <- (A) - (C) - data	1 0 0 1 0 1 0 0 d d d d d d d d	2	1	C AC OV P
SUBB A,@Rr (A) <- (A) - (C) - ((Rr))	1 0 0 1 0 1 1 r	1	1	C AC OV P

Mnemonika a činnost instrukce	Operační kód (binárně)	Délka byt	Délka cyc	Příznaky
SUBB A,Rr (A) <- (A) - (C) - (Rr)	1 0 0 1 1 r r r	1	1	C AC OV P
SUBB A,data_adr (A)<-(A)-(C)-(data_adr)	1 0 0 1 0 1 0 1 m m m m m m m m	2	1	C AC OV P
SWAP A záměna horního a dolního nibble akumulátoru	1 1 0 0 0 1 0 0	1	1	
XCH A,@Rr (dočasný registr) <- ((Rr)) ((Rr))<- (A) (A) <- (dočasný registr)	1 1 0 0 0 1 1 r	1	1	P
XCH A,Rr (dočasný registr) <- (Rr) (Rr) <- (A) (A) <- (dočasný registr)	1 1 0 0 1 r r r	1	1	P
XCH A,data_adr (dočasný registr)<-(data_adr) (data_adr) <- (A) (A) <- (dočasný registr)	1 1 0 0 0 1 0 1 m m m m m m m m	2	1	P
XCHD A,@Rr (dočasný registr)<-((Rr))0-3 ((Rr))0-3 <- (A)0-3 (A)0-3 <- (dočasný registr)	1 1 0 1 0 1 1 r	1	1	P
XRL A,#data (A) <- (A) XOR data	0 1 1 0 0 1 0 0 d d d d d d d d	2	1	P
XRL A,@Rr (A) <- (A) XOR ((Rr))	0 1 1 0 0 1 1 r	1	1	P
XRL A,Rr (A) <- (A) XOR (Rr)	0 1 1 0 1 r r r	1	1	P
XRL A,data_adr (A) <- (A) XOR (data_adr)	0 1 1 0 0 1 0 1 m m m m m m m m	2	1	P
XRL data_adr,#data (data_adr)<-(data_adr) XOR data	0 1 1 0 0 0 1 1 m m m m m m m m d d d d d d d d	3	2	
XRL data_adr,A (data_adr)<-(data_adr) XOR (A)	0 1 1 0 0 0 1 0 m m m m m m m m	2	2	

PŘÍLOHA 2
SEZNAM INSTRUKCÍ PODLE OPERAČNÍHO KÓDU

Tato příloha obsahuje seznam všech instrukcí procesoru řady MCS-51. Instrukce jsou seřazeny podle operačního kódu.

Hexadec.kód	Mnemonika	Operandy	Délka instrukce (byte)	Délka instrukce (cyklů)
00	NOP		1	1
01	AJMP	<adr>	2	2
02	LJMP	<adr>	3	2
03	RR	A	1	1
04	INC	A	1	1
05	INC	data_adr	2	1
06	INC	@R0	1	1
07	INC	@R1	1	1
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
10	JBC	bit_adr	3	2
11	ACALL	<adr>	2	2
12	LCALL	<adr>	3	2
13	RRC	A	1	1
14	DEC	A	1	1
15	DEC	data_adr	2	1
16	DEC	@R0	1	1
17	DEC	@R1	1	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
20	JB	bit_adr	3	2
21	AJMP	<adr>	2	2
22	RET		1	2
23	RL	A	1	1
24	ADD	A,#data	2	1
25	ADD	A,data_adr	2	1
26	ADD	A,@R0	1	1
27	ADD	A,@R1	1	1
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

Hexadec.kód	Mnemonika	Operandy	Délka instrukce (byte)	Délka instrukce (cyklů)
2D	ADD	A,R5	1	1
2E	ADD	A,R6	1	1
2F	ADD	A,R7	1	1
30	JNB	bit_adr,<adr>	3	2
31	ACALL	<adr>	2	2
32	RETI		1	2
33	RLC	A	1	1
34	ADDC	A,#data	2	1
35	ADDC	A,data_adr	2	1
36	ADDC	A,@R0	1	1
37	ADDC	A,@R1	1	1
38	ADDC	A,R0	1	1
39	ADDC	A,R1	1	1
3A	ADDC	A,R2	1	1
3B	ADDC	A,R3	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
40	JC	<adr>	2	2
41	AJMP	<adr>	2	2
42	ORL	data_adr,A	2	1
43	ORL	data_adr,#data	3	2
44	ORL	A,#data	2	1
45	ORL	A,data_adr	2	1
46	ORL	A,@R0	1	1
47	ORL	A,@R1	1	1
48	ORL	A,R0	1	1
49	ORL	A,R1	1	1
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
50	JNC	<adr>	2	2
51	ACALL	<adr>	2	2
52	ANL	data_adr,A	2	1
53	ANL	data_adr,#data	3	2
54	ANL	A,#data	2	1
55	ANL	A,data_adr	2	1
56	ANL	A,@R0	1	1
57	ANL	A,@R1	1	1
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
60	JZ	<adr>	2	2
61	AJMP	<adr>	2	2
62	XRL	data_adr,A	2	2

Hexadec.kód	Mnemonika	Operandy	Délka instrukce (byte)	Délka instrukce (cyklů)
63	XRL	data_adr,#data	3	2
64	XRL	A,#data	2	1
65	XRL	A,data_adr	2	1
66	XRL	A,@R0	1	1
67	XRL	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
70	JNZ	<adr>	2	2
71	ACALL	<adr>	2	2
72	ORL	C,bit_adr	2	1
73	JMP	@A+DPTR	1	2
74	MOV	A,#data	2	1
75	MOV	data_adr,#data	3	2
76	MOV	@R0,#data	2	1
77	MOV	@R1,#data	2	1
78	MOV	R0,#data	2	1
79	MOV	R1,#data	2	1
7A	MOV	R2,#data	2	1
7B	MOV	R3,#data	2	1
7C	MOV	R4,#data	2	1
7D	MOV	R5,#data	2	1
7E	MOV	R6,#data	2	1
7F	MOV	R7,#data	2	1
80	SJMP	<adr>	2	2
81	AJMP	<adr>	2	2
82	ANL	C,bit_adr	2	1
83	MOVC	A,@A+PC	1	2
84	DIV	AB	1	4
85	MOV	data_adr1,data_adr2	3	2
86	MOV	data_adr,@R0	2	2
87	MOV	data_adr,@R1	2	2
88	MOV	data_adr,R0	2	2
89	MOV	data_adr,R1	2	2
8A	MOV	data_adr,R2	2	2
8B	MOV	data_adr,R3	2	2
8C	MOV	data_adr,R4	2	2
8D	MOV	data_adr,R5	2	2
8E	MOV	data_adr,R6	2	2
8F	MOV	data_adr,R7	2	2
90	MOV	DPTR,#data	3	2
91	ACALL	<adr>	2	2
92	SUBB	A,R2	1	1
93	MOVC	A,@A+DPTR	1	2
94	SUBB	A,#data	2	1
95	SUBB	A,data_adr	2	1
96	SUBB	A,@R0	1	1
97	SUBB	A,@R1	1	1
98	SUBB	A,R0	1	1

Hexadec.kód	Mnemonika	Operandy	Délka instrukce (byte)	Délka instrukce (cyklů)
99	SUBB	A,R1	1	1
9A	MOV	bit_adr	2	2
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
A0	ORL	C,bit_adr	2	2
A1	AJMP	<adr>	2	2
A2	MOV	C,bit_adr	2	2
A3	INC	DPTR	1	2
A4	MUL	AB	1	4
A6	MOV	@R0,data_adr	2	2
A7	MOV	@R1,data_adr	2	2
A8	MOV	R0,data_adr	2	2
A9	MOV	R1,data_adr	2	2
AA	MOV	R2,data_adr	2	2
AB	MOV	R3,data_adr	2	2
AC	MOV	R4,data_adr	2	2
AD	MOV	R5,data_adr	2	2
AE	MOV	R6,data_adr	2	2
AF	MOV	R7,data_adr	2	2
B0	ANL	C,/bit_adr	2	2
B1	ACALL	<adr>	2	2
B2	CPL	bit_adr	2	1
B3	CPL	C	1	1
B4	CJNE	A,#data,<adr>	3	2
B5	CJNE	A,data_adr,<adr>	3	2
B6	CJNE	@R0,#data,<adr>	3	2
B7	CJNE	@R1,#data,<adr>	3	2
B8	CJNE	R0,#data,<adr>	3	2
B9	CJNE	R1,#data,<adr>	3	2
BA	CJNE	R2,#data,<adr>	3	2
BB	CJNE	R3,#data,<adr>	3	2
BC	CJNE	R4,#data,<adr>	3	2
BD	CJNE	R5,#data,<adr>	3	2
BE	CJNE	R6,#data,<adr>	3	2
BF	CJNE	R7,#data,<adr>	3	2
C0	PUSH	data_adr	2	2
C1	AJMP	<adr>	2	2
C2	CLR	bit_adr	2	1
C3	CLR	C	1	1
C4	SWAP	A	1	1
C5	XCH	A,data_adr	2	1
C6	XCH	A,@R0	1	1
C7	XCH	A,@R1	1	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

Hexadec.kód	Mnemonika	Operandy	Délka instrukce (byte)	Délka instrukce (cyklů)
D0	POP	data_adr	2	2
D1	ACALL	<adr>	2	2
D2	SETB	bit_adr	2	1
D3	SETB	C	1	1
D4	DA	A	1	1
D5	DJNZ	data_adr,<adr>	3	2
D6	XCHD	A,@R0	1	1
D7	XCHD	A,@R1	1	1
D8	DJNZ	R0,<adr>	2	2
D9	DJNZ	R1,<adr>	2	2
DA	DJNZ	R2,<adr>	2	2
DB	DJNZ	R3,<adr>	2	2
DC	DJNZ	R4,<adr>	2	2
DD	DJNZ	R5,<adr>	2	2
DE	DJNZ	R6,<adr>	2	2
DF	DJNZ	R7,<adr>	2	2
E0	MOVX	A,@DPTR	1	2
E1	AJMP	<adr>	2	2
E2	MOVX	A,@R0	1	2
E3	MOVX	A,@R1	1	2
E4	CLR	A	1	1
E5	MOV	A,data_adr	2	1
E6	MOV	A,@R0	1	1
E7	MOV	A,@R1	1	1
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
EE	MOV	A,R6	1	1
EF	MOV	A,R7	1	1
F0	MOVX	@DPTR,A	1	2
F1	ACALL	<adr>	2	2
F2	MOVX	@R0,A	1	2
F3	MOVX	@R1,A	1	2
F4	CPL	A	1	1
F5	MOV	data_adr,A	2	1
F6	MOV	@R0,A	1	1
F7	MOV	@R1,A	1	1
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

PŘÍLOHA 3
SPECIÁLNÍ FUNKČNÍ REGISTRY

Tato příloha obsahuje přehled obsazení prostoru speciálních funkčních registrů pro nejčastěji používané typy procesorů řady MCS-51 a jejich derivátů od firem Intel, Siemens a Philips.

První sloupec tabulky obsahuje hexadecimální adresu datové pamětiv oblasti SFR, další sloupce pak obsahují symbolické názvy speciálních funkčních registrů pro typ procesoru, uvedený v záhlaví sloupce. Pokud není příslušná adresa obsazena žádným speciálním funkčním regitrem, není položka vyplněna.

	8051	8052	80521	8051FA	8051GA	8051GB	80152
080H	P0	P0	P0	P0	P0	P0	P0
081H	SP	SP	SP	SP	SP	SP	SP
082H	DPL	DPL	DPL	DPL	DPL	DPL	DPL
083H	DPH	DPH	DPH	DPH	DPH	DPH	DPH
084H			DPL1		ADRES	ADRES0	GMOD
085H			DPH1				TFIFO
086H			DPS		WDTLB		
087H	PCON	PCON	PCON	PCON	PCON	PCON	PCON
088H	TCON	TCON	TCON	TCON	TCON	TCON	TCON
089H	TMOD	TMOD	TMOD	TMOD	TMOD	TMOD	TMOD
08AH	TL0	TL0	TL0	TL0	TL0	TL0	TL0
08BH	TL1	TL1	TL1	TL1	TL1	TL1	TL1
08CH	TH0	TH0	TH0	TH0	TH0	TH0	TH0
08DH	TH1	TH1	TH1	TH1	TH1	TH1	TH1
08EH						AUXR	
08FH							
090H	P1	P1	P1	P1	P1	P1	P1
091H							P5
092H							DCON0
093H							DCON1
094H						ADRES1	BAUD
095H					WDTDIS		ADR0
096H					WDTUB		
097H					ADCON	ADCON	
098H	SCON	SCON	SCON	SCON	SCON	SCON	SCON
099H	SBUF	SBUF	SBUF	SBUF	SBUF	SBUF	SBUF
09AH						C1CAPM0	
09BH						C1CAPM1	
09CH						C1CAPM2	
09DH						C1CAPM3	
09EH						C1CAPM4	
09FH						C1MOD	
0A0H	P2	P2	P2	P2	P2	P2	P2
0A1H							P6
0A2H							SARL0
0A3H							SARH0
0A4H						ADRES2	IFS
0A5H					OFDCON	OFDCON	ADR1
0A6H					WDTCON	WDTCON	
0A7H						IEA	

	8051	8052	80521	8051FA	8051GA	8051GB	80152
0A8H	IE	IE	IE	IE	IE	IE	IE
0A9H				SADDR		SADDR	
0AAH						C1CAP0L	
0ABH						C1CAP1L	
0ACH						C1CAP2L	
0ADH						C1CAP3L	
0AEH						C1CAP4L	
0AFH						CL1	
0B0H	P3	P3	P3	P3	P3	P3	P3
0B1H							
0B2H							SARL1
0B3H							SARH1
0B4H						ADRES3	SLOTTM
0B5H						IPA1	ADR2
0B6H						IPA	
0B7H						IP1	
0B8H	IP	IP	IP	IP	IP	IP	IP
0B9H				SADEN		SADEN	
0BAH						C1CAP0H	
0BBH						C1CAP1H	
0BCH						C1CAP2H	
0BDH						C1CAP3H	
0BEH						C1CAP4H	
0BFH						CH1	
0C0H						P4	P4
0C1H							
0C2H							DARL0
0C3H							DARH0
0C4H						ADRES4	BKOFF
0C5H							ADR3
0C6H						EXICON	
0C7H						ACMP	
0C8H		T2CON		T2CON		T2CON	IEN1
0C9H				T2MOD		T2MOD	
0CAH		RCAP2L		RCAP2L		RCAP2L	
0CBH		RCAP2H		RCAP2H		RCAP2H	
0CCH		TL2		TL2		TL2	
0CDH		TH2		TH2		TH2	
0CEH							
0CFH							
0D0H	PSW	PSW	PSW	PSW	PSW	PSW	PSW
0D1H							
0D2H							DARL1
0D3H							DARH1
0D4H						ADRES5	TCDCNT
0D5H							AMSK0
0D6H							
0D7H					SEPCON	SEPCON	
0D8H				CCON		CCON	TSTAT
0D9H				CMOD		CMOD	
0DAH				CCAPM0		CCAPM0	
0DBH				CCAPM1		CCAPM1	
0DCH				CCAPM2		CCAPM2	
0DDH				CCAPM3		CCAPM3	
0DEH				CCAPM4		CCAPM4	

	8051	8052	80521	8051FA	8051GA	8051GB	80152
0DFH							
0E0H	ACC	ACC	ACC	ACC	ACC	ACC	ACC
0E1H							
0E2H							BCRL0
0E3H							BCRH0
0E4H						ADRES6	PRBS
0E5H							AMSK1
0E6H							
0E7H					SEPDAT	SEPDAT	
0E8H						C1CON	RSTAT
0E9H				CL		CL	
0EAH				CCAP0L		CCAP0L	
0EBH				CCAP1L		CCAP1L	
0ECH				CCAP2L		CCAP2L	
0EDH				CCAP3L		CCAP3L	
0EEH				CCAP4L		CCAP4L	
0EFH							
0F0H	B	B	B	B	B	B	B
0F1H							
0F2H							BCRL1
0F3H							BCRH1
0F4H						ADRES7	RFIFO
0F5H							MYSLOT
0F6H							
0F7H					SEPSTA	SEPSTA	
0F8H						P5	IPN1
0F9H				CH		CH	
0FAH				CCAP0H		CCAP0H	
0FBH				CCAP1H		CCAP1H	
0FCH				CCAP2H		CCAP2H	
0FDH				CCAP3H		CCAP3H	
0FEH				CCAP4H		CCAP4H	
0FFH							

	80C515	80C517	8044	80451	80452	80C552	80C652
080H	P0	P0	P0	P0	P0	P0	P0
081H	SP	SP	SP	SP	SP	SP	SP
082H	DPL	DPL	DPL	DPL	DPL	DPL	DPL
083H	DPH	DPH	DPH	DPH	DPH	DPH	DPH
084H		WDTL					
085H		WDTH					
086H		WDTREL					
087H	PCON	PCON	PCON	PCON	PCON	PCON	PCON
088H	TCON	TCON	TCON	TCON	TCON	TCON	TCON
089H	TMOD	TMOD	TMOD	TMOD	TMOD	TMOD	TMOD
08AH	TL0	TL0	TL0	TL0	TL0	TL0	TL0
08BH	TL1	TL1	TL1	TL1	TL1	TL1	TL1
08CH	TH0	TH0	TH0	TH0	TH0	TH0	TH0
08DH	TH1	TH1	TH1	TH1	TH1	TH1	TH1
08EH							
08FH							
090H	P1	P1	P1	P1	P1	P1	P1
091H							
092H		DPSEL			DCON0		
093H					DCON1		
094H							
095H							
096H							
097H							
098H	SCON	S0CON	SCON		SCON	S0CON	S0CON
099H	SBUF	S0BUF	SBUF		SBUF	S0BUF	S0BUF
09AH		IEN2					
09BH		S1CON					
09CH		S1BUF					
09DH		S1REL					
09EH			EINT				
09FH			EBUF				
0A0H	P2	P2	P2	P2	P2	P2	P2
0A1H							
0A2H					SARL0		
0A3H					SARH0		
0A4H							
0A5H							
0A6H							
0A7H							
0A8H	IEN0	IEN0	IE	IE	IE	IEN0	IE
0A9H	IP0	IP0				CML0	
0AAH						CML1	
0ABH						CML2	
0ACH						CTL0	
0ADH						CTL1	
0AEH						CTL2	
0AFH						CTL3	
0B0H	P3	P3	P3	P3	P3	P3	P3
0B1H							
0B2H					SARL1		
0B3H					SARH1		
0B4H							
0B5H							
0B6H							

	80C515	80C517	8044	80451	80452	80C552	80C652
0B7H							
0B8H	IEN1	IEN1	IP	IP	IP	IP0	IP
0B9H	IP1	IP1					
0BAH							
0BBH							
0BCH							
0BDH							
0BEH							
0BFH							
0C0H	IRCON	IRCON		P4	P4	P4	
0C1H	CCEN	CCEN					
0C2H	CCL1	CCL1			DARL0		
0C3H	CCH1	CCH1			DARH0		
0C4H	CCL2	CCL2				P5	
0C5H	CCH2	CCH2				ADCON	
0C6H	CCL3	CCL3				ADCH	
0C7H	CCH3	CCH3					
0C8H	T2CON	T2CON	STS	P5		TM2IR	
0C9H		CC4EN	SMD			CMH0	
0CAH	CRCL	CRCL	RCB			CMH1	
0CBH	CRCH	CRCH	RBL			CMH2	
0CCH	TL2	TL2	RBS			CTH0	
0CDH	TH2	TH2	RFL			CTH1	
0CEH		CCL4	STAD			CTH2	
0CFH		CCH4	DMA			CTH3	
0D0H	PSW	PSW	PSW	PSW	PSW	PSW	PSW
0D1H							
0D2H		CML0			DARL1		
0D3H		CMH0			DARH1		
0D4H		CML1					
0D5H		CMH1					
0D6H		CML2					
0D7H		CMH2					
0D8H	ADCON	ADCON0	NSNR	P6		S1CON	S1CON
0D9H	ADDAT	ADDAT	SIUST			S1STA	S1STA
0DAH	DAPR	DAPR	TCB			S1DAT	S1DAT
0DBH		P7	TBL			S1ADR	S1ADR
0DCH		ADCON1	TBS				
0DDH		P8	FIFO1				
0DEH		CTRELL	FIFO2				
0DFH		CTRELH	FIFO3				
0E0H	ACC	ACC	ACC	ACC	ACC	ACC	ACC
0E1H		CTCON					
0E2H		CML3			BCRL0		
0E3H		CMH3			BCRH0		
0E4H		CML4					
0E5H		CMH4					
0E6H		CML5			HSTAT		
0E7H		CMH5			HCON		
0E8H	P4	P4		CSR	SLCON	IEN1	
0E9H		MD0			SSTAT		
0EAH		MD1			IWPR	TM2CON	
0EBH		MD2			IRPR	CTCON	
0ECH		MD3			CBP	TML2	
0EDH		MD4				TMH2	

	80C515	80C517	8044	80451	80452	80C552	80C652
0EEH		MD5			FIN	STE	
0EFH		ARCON			CIN	RTE	
0F0H	B	B	B	B	B	B	B
0F1H							
0F2H		CML6			BCRL1		
0F3H		CMH6			BCRH1		
0F4H		CML7					
0F5H		CMH7					
0F6H		CMEN			ITHR		
0F7H		CMSEL			OTHR		
0F8H	P5	P5			IEP	IP1	
0F9H					MODE		
0FAH		P6			ORPR		
0FBH					OWPR		
0FCH					IMIN	PWM0	
0FDH					IMOUT	PWM1	
0FEH					FOUT	PWMP	
0FFH					COUT	T3	

PŘÍLOHA 4
REZERVOVANÁ SLOVA ASEMBLERU

Tato příloha obsahuje seznam všech klíčových slov rezervovaných pro použití asemblerem. Žádné z těchto slov nemůže být použito jako identidikátor v uživatelském programu.

Je třeba ještě výslovně upozornit na to, že příkazy asembleru (viz příloha 5) nejsou považovány za klíčová slova a lze je tedy bez omezení použít jako symboly v programu.

Rovněž předdefinované symboly, vybrané příkazem MODE(device) nelze během programu předefinovat, tyto symboly lze použít jako operandy výrazů a instrukcí.

Operátory				
AND	GT	LOW	NE	SHL
EQ	HIGH	LT	NOT	SHR
GE	LE	MOD	OR	XOR

Mnemonika instrukcí				
ACALL	DA	JNB	MUL	RR
ADD	DEC	JNC	NOP	RRC
ADDC	DIV	JNZ	ORL	SETB
AJMP	DJNZ	JZ	POP	SJMP
ANL	INC	LCALL	PUSH	SUBB
CALL	JB	LJMP	RET	SWAP
CJNE	JBC	MOV	RETI	XCH
CLR	JC	MOVC	RL	XCHD
CPL	JMP	MOVX	RLC	XRL

Symbolické adresy registrů			
AR0	AR2	AR4	AR6
AR1	AR3	AR5	AR7

Operandy a speciální symboly asembleru				
A	DPTR	R1	R4	R7
AB	PC	R2	R5	
C	R0	R3	R6	

Direktivy				
AT	DB	EQU	ORG	XDATA
BIT	DBIT	EXTRN	PUBLIC	XSEG
BSEG	DS	IDATA	RSEG	
CODE	DSEG	ISEG	SEGMENT	
CSEG	DW	NAME	SET	
DATA	END	NUMBER	USING	

PŘÍLOHA 5
SEZNAM PŘÍKAZŮ ASEMBLERU

Následující tabulka obsahuje seznam příkazů asembleru ASM-52 se stručným popisem jejich funkce a s údaji o počátečním nastavení jednotlivých příkazů. Tabulka obsahuje rovněž přehled možných zkratek příkazů.

Příkaz	Primární/ Všeobecný	Běžné nastavení	Zkratka	Význam
DATE[(datum)]	P	DATE(sytem_date)	DT	Do záhlaví stránky vstoupí datum
NODATE			NODT	
DEBUG	P	NODEBUG	DB	Do cílového kódu formátu Intel-OMF vstoupí ladící informace
NODEBUG			NODB	
ERRORPRINT[(soubor)]	P	NOERRORPRINT	EP	Určuje, zda se má či nemá tvořit soubor chybových výpisů
NOERRORPRINT			NOEP	
HEX[(soubor)]	P	NOHEX	HX	Určuje, zda se bude tvořit cílový soubor ve tvaru Intel-HEX
NOHEX			NOHX	
INCLUDE(soubor)	V	nemá význam	IC	Soubor, vstupující do zdrojového textu
LIST	V	LIST	LI	Do listingu programu vstoupí/nevstoupí řádky se zdrojovým textem
NOLIST			NOLI	
MODE(typ_CPU)	P	MODE(51)	MO	Určuje typ použitého procesoru a podle toho se vybere/potlačí sada definovaných symbolů
NOMODE			NOMO	
OBJECT[(soubor)]	P	OBJECT(source.OBJ)	OJ	Určuje jméno a existenci cílového souboru typu Intel-OMF
NOOBJECT			NOOJ	
PAGING	P	PAGING	PI	Určuje, zda protokol o výstupu (listing) bude dělen na stránky
NOPAGING			NOPI	
PAGELENGTH(číslo)	P	PAGELENGTH(62)	PL	Počet řádků na stránce
PAGEWIDTH(číslo)	P	PAGEWIDTH(120)	PW	Počet znaků v řádku
PRINT[(soubor)]	P	PRINT(source.LST)	PR	Určuje jméno a existenci výstupního protokolu (listingu) programu
NOPRINT			NOPR	

Příkaz	Primární/ Všeobecný	Běžné nastavení	Zkratka	Význam
REGISTERBANK(číslo)	P	REGISTERBANK(0)	RB	Určuje, které banky registrů budou programem použity
NOREGISTERBANK			NORB	
SYMBOLS	P	SYMBOLS	SB	Určuje, zda výstupní protokol bude/nebude obsahovat tabulku symbolů
NOSYMBOLS			NOSB	
TABS(číslo)	P	TABS(8)	TA	Nastavuje počet mezer, na který se budou expandovat tabelátory
TITLE(řetězec)	V	TITLE(jméno_modulu)	TT	Text, který bude přidán do záhlaví každé stránky výstupního protokolu
XREF	P	NOXREF	XR	Výstupní protokol bude/nebude obsahovat křížové odkazy na symboly
NOXREF			NOXR	

INDEX