**Instrukční soubor**kap. 31.0 datasheetu, strana 359

|  |  |  |  |
| --- | --- | --- | --- |
| **mnemonika** | **popis** | **T** | mění bity  STATUS reg. |
| **Instrukce přesunů** | | | |
| MOVF f , d | okopíruje číslo z  f do W nebo opět do f – podle d | 1 | Z |
| MOVWF f | okopíruje číslo z W do f | 1 |  |
| MOVLW k | uloží osmibitové číslo k do W | 1 |  |
| CLRF f | vynuluje registr f | 1 | Z |
| CLRW | vynuluje registr W | 1 | Z |
| MOVLB k | Uloží číslo k do registru BSR – přepne stránku RAM | 1 |  |
| MOVLP k | Uloží číslo k do registru PCLATH – pro skoky | 1 |  |
| TRIS f | Uloží číslo z W do registru TRISx nezávisle na stránce | 1 |  |
| OPTION | Uloží hodnotu z W do registru OPTION\_REG | 1 |  |
| **Aritmetické instrukce** | | | |
| ADDWF f , d | sečte reg. W a f , výsledek uloží do W nebo f – dle d | 1 | C, DC, Z |
| ADDWFC f , d | sečte reg. W a f a bit C , výsledek uloží podle d | 1 | C, DC, Z |
| SUBWF f , d | odečte W od f (dělá rozdíl f – W), výsledek podle d | 1 | C, DC, Z |
| SUBWFB f , d | dělá rozdíl f – W – (bit C) , výsledek uloží podle d | 1 | C, DC, Z |
| ADDLW k | sečte W s číslem k ( W + k → W ) | 1 | C, DC, Z |
| SUBLW k | odečte od konstanty W ( k – W → W ) | 1 | C, DC, Z |
| DECF f , d | odečte od reg. f 1 , výsledek do W nebo f – dle d | 1 | Z |
| INCF f , d | přičte k reg. f 1 , výsledek do W nebo f – dle d | 1 | Z |
| **Logické instrukce** | | | |
| COMF f , d | negace všech bitů reg. f | 1 | Z |
| ANDWF f , d | logický součin W a f (W and f) každý bit zvlášť | 1 | Z |
| ANDLW k | logický součin W a konstanty ( W and k → W ) | 1 | Z |
| IORWF f , d | log. součet (OR) W a f ( W or f → d ) | 1 | Z |
| IORLW k | log. součet W a čísla k ( W or k → W ) | 1 | Z |
| XORWF f , d | log. EX – OR registrů W a f ( W xor f → d ) | 1 | Z |
| XORLW k | log. EX-OR W a čísla k ( W xor k → W ) | 1 | Z |
| **Bitové instrukce a rotace** | | | |
| BCF f , b | nastavit bit b registru f do 0 (bit clear) | 1 |  |
| BSF f , b | nastavit bit b registru f do 1 (bit set ) | 1 |  |
| RLF f , d | rotace bitů registru f doleva přes bit C | 1 | C |
| RRF f , d | rotace bitů registru f doprava přes bit C | 1 | C |
| LSLF f , d | Logický posuv vlevo (Logical Left Shift) | 1 | C |
| LSRF f , d | Logický posuv vpravo (Logical Right Shift) | 1 | C |
| ASRF f , d | Artimetický posuv vpravo (Arithmetic Right Shift) | 1 | C |
| SWAPF f , d | přehodí dolní a horní polovinu registru | 1 |  |
| **Skoky** | | | |
| GOTO aa | nepodmíněný skok na adresu aa (provede se vždy) | 2 |  |
| BRA k | Relativní skok o k buněk (2.doplněk) < -256 ; +255> | 2 |  |
| BRW | Relativní skok o číslo ve W < 0 ; 255> | 2 |  |
| BTFSC f , b | bit test, skip if clear | 1,2 |  |
| BTFSS f , b | bit test, skip if set | 1,2 |  |
| DECFSZ f , d | decrement and skip if zero | 1,2 |  |
| INCFSZ f , d | increment and skip if zero | 1,2 |  |
| **Podprogramy a přerušení** | | | |
| CALL aa | volání podprogramu na adrese aa | 2 |  |
| CALLW | volání podprogramu podle obsahu reg. W | 2 |  |
| RETURN | návrat z podprogramu | 2 |  |
| RETLW k | návrat z podprogramu s číslem k v registru W | 2 |  |
| RETFIE | návrat z přerušení ( navíc 1 → GIE, zpět shadow reg. ) | 2 |  |
| **Ostatní** | | | |
| NOP | nedělej nic | 1 |  |
| CLRWDT | nuluje čítač u watchdog | 1 | TO, PD |
| SLEEP | uspí procesor (standby mode) | 1 | TO, PD |
| RESET | softwarový RESET procesoru |  |  |
| **C - compiler optimized** | | | |
| ADDFSR n , k | přičti číslo k  k registru FSRn | 1 |  |
| MOVIW n, mm | nepřímé adresování, do W, pre-post/inc-dec | 1 |  |
| MOVWI n, mm | nepřímé adresování, z W, pre-post/inc-dec | 1 |  |

**W** – pracovní registr procesoru

**f -** buňka paměti RAM registr je buňka paměti RAM

**d -** určení místa, kam se má uložit výsledek: d = 1 – výsledek do registru f

d = 0 – výsledek do registru W

**b –** číslo bitu, se kterým se má pracovat

**aa –** adresa v programové paměti EEPROM

**Poznámky k instrukčnímu souboru**

Procesor má, podobně jako ostatní procesory, pracovní registr W. Je to analogie akumulátoru u jiných procesorů. Výsledky jakékoli operace (a to i obyčejného přesunu) se ukládají buď do registru W, pokud je parametr d=0, nebo do registru f, pokud je parametr d=1.

Jako registr f lze použít buňku o jakékoli adrese z paměti RAM, ať už znamená Core registers, SFR, GPR, Common RAM (adresy 00 – 7F).

Program budeme psát v assembleru. Překladač pak takto napsaný program přeloží do strojového kódu (to jsou 14bitová číslíčka, která se potom naprogramují do programové EEPROM procesoru).

Trochu problémy bude dělat parametr d – destination – určení místa, kam se má uložit výsledek.

d=1 znamená, že se výsledek ukládá do buňky paměti RAM – registr f.

Pokud je d=0, je výsledek uložen do registru W. Pokud parametr d neudáte, překladač dosadí implicitní hodnotu. Ta je 1, tedy pokud se d neudá , ukládá výsledek do registru f. Pokud chceme uložit výsledek do W, **musíme** udat parametr d = 0

Překladač „umí“ velmi mnoho užitečných věcí. Jednou z nich je příkaz EQU, který přiřadí skupině písmenek číslo (podobně jako const v Pascalu), takže v programu je potom možno používat takto vzniklý identifikátor.

reg equ 0Eh ; zde samozřejmě může být jakékoli číslo , skupina písmenek reg znamená 0Eh

nadále budeme předpokládat, že identifikátoru reg je přiřazeno číslo v rozsahu adres RAM, tedy 0 – 7Fh

**MOVF reg , 0**  ; tato instrukce okopíruje obsah buňky reg do registru W. Parametr d nutno udat !

**MOVF reg , 1**  ; tato instrukce okopíruje obsah buňky reg opět do buňky reg. Zdánlivě nesmyslné, ale

MOVF nastavuje bit Z  STATUS registru, takže tímto způsobem lze udělat test na hodnotu 0

**MOVF reg**  ; totéž - instrukce okopíruje obsah buňky reg do buňky reg. Protože d není udáno, dosadil

; překladač d = 1

**MOVWF reg** ; tato instrukce okopíruje registr W do buňky reg . Pozor ! rádo se plete s MOVF! Ale lze si

to zapamatovat: poslední dvě písmenka instrukce říkají odkud kam (W → F, f adresa buňky)

**MOVLW číslo8bit** ; instrukce uloží číslo do registru W (L – literál – číslo. Tedy L → W)

Konstantu do registru – buňky RAM lze „narvat“ pouze pomocí registru W, tedy posloupností:

MOVLW konstanta

MOVWF reg

**ADDWF reg, 0** ; instrukce sečte obsah W a buňky reg a výsledek uloží do W

**ADDWF reg, 1** ; instrukce sečte obsah W a buňky reg a výsledek uloží do reg

**ADDWF reg**  ; instrukce sečte obsah W a buňky reg a výsledek uloží do reg – překladač dosadil d = 1

**ADDLW číslo8bit** ; instrukce sečte obsah W s osmibitovým číslem a výsledek uloží do W

instrukce součtu nastavují bity registru STATUS – C,DC,Z

**ADDWFC reg, 0** ; instrukce sečte obsah W a buňky reg a jeśtě k tomu přičte hodnotu bitu C v registru STATUS. výsledek uloží do W. Tohle se nazývá „sčítání s přenosem“ – hodí se pro sčítání vícebytových čísel. Funguje samozřejmě i pro d=1

**SUBWF reg, 0** ; udělá rozdíl (reg - W) a výsledek uloží do W

Pozor ! Tento procesor má instrukci rozdílu uspořádanou netradičně – od registru se

odečítá pracovní registr W ! není to tisková chyba, opravdu reg – W → W

**SUBWF reg, 1**  ; rozdíl stejně jako předchozí instrukce, výsledek do reg ………. reg – W → reg

**SUBWF reg** ; totéž …………. reg – W → reg ……. překladač dosadí d = 1

**SUBLW číslo8bit** ; odečte **od čísla** W, výsledek do W (L – W → W …. L ..literál … číslo)

Při provedení instrukce SUB se do bitu C ukládá **negovaná** hodnota přetečení z nejvyššího bitu výsledku

**SUBWFB reg, 0** ; udělá rozdíl (reg - W – C ) a výsledek uloží do W . Tedy oproti instrukci SUBWF ještě dále odečte hodnotu bitu C ve STATUS registru. Hodí se pro odčítání vícebytových čísel.

**DECF reg , 0** ; od reg odečte 1, výsledek uloží do W. **Obsah reg se tedy nezmění !!!!!**

**DECF reg , 1** ; od reg odečte 1, výsledek uloží do reg . **Obsah reg se tedy sníží o 1 !**

**DECF reg** ; od reg odečte 1, výsledek uloží do reg. **Obsah reg se tedy sníží o 1!** dosazeno d = 1

**INCF reg** ; jako DECF, přičítá se 1; registr samozřejmě klidně přetéká – po 255 následuje 0

**COMF reg , 0** ; provede logickou negaci každého bitu reg zvlášť, výsledek do W, reg nezměněn

**COMF reg , 1** ; provede logickou negaci každého bitu reg zvlášť, výsledek do reg, reg se tedy mění

**COMF reg** ; default pro d je 1, v dalším textu již nebudeme připomínat

**AND** ; logický součin, každý bit zvlášť, operandy a možnosti jako u ADD

**IOR**  ; logický součet – OR , každý bit zvlášť, operandy a možnosti jako u ADD (inclusive OR)

**XOR**  ; logická funkce „výhradně nebo“ – XOR , EXCLUSIVE-OR, EX-OR

**BCF reg , b** ; nastaví b-tý bit registru reg do 0 (bit clear)

**BSF reg , b** ; nastaví b-tý bit registru reg do 1 (bit set)

**RLF reg , 0** ; rotace bitů registru reg spolu s c doleva, reg se nemění, výsledek ve W

**RLF reg , 1** ; rotace bitů registru reg spolu s c doleva, výsledek v reg

příklad: nechť C = 1, obsah reg = 1Ch

po provedení RLF reg , 1 bude: reg = 39h, C = 0



**RRF reg , 0** ; rotace bitů registru reg spolu s c doprava, reg se nemění, výsledek ve W

**RRF reg , 1** ; rotace bitů registru reg spolu s c doprava, výsledek v reg

příklad: nechť C = 1, obsah reg = 1Ch

po provedení RLF reg , 1 bude: reg = 8Eh, C = 0



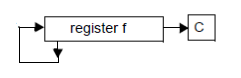
**LSRF reg, d** ; logický posuv bitů registru reg doleva (logical shift right) . Nejnižší bit vyleze do bitu C, zleva se do nejvyššího bitu nasune 0, bity se posouvají doprava

****

**LSLF reg, d** ; logický posuv bitů registru reg doleva (logical shift left) . Do nejnižšího bit o se zleva nasune 0, nejvyšší bit vyleze do bitu C, bity se posouvají doleva

****

**ASRF reg, d** ; aritmetický posuv bitů registru reg doleva (logical shift right) . Nejnižší bit vyleze do bitu C, bity se posunou doprava, nejvyšší bit – znaménkový – se okopíruje ( hodnota, která byla na 7. bitu, zůstává stále stejná, a tento bit se ještě navíc okopíruje doprava ) .

****

**SWAPF ;** přehodí dolní a horní polovinu (4 bity) registru

Příklad: reg = 5Ch, po provedení SWAPF je reg = C5h

Určení místa pro uložení výsledku považujeme již za samozřejmost, proto se o něm nezmiňujeme

**BTFSC reg , b**  ; bit test and skip if clear

podívá se na b-tý bit registru reg. Pokud tento bit je 0, **jedna** následující instrukce se přeskočí. Pokud bit je 1, následující instrukce se provede.

**BTFSS reg , b**  ; bit test and skip if set

podívá se na b-tý bit registru reg. Pokud tento bit je 1, **jedna** následující instrukce se přeskočí. Pokud bit je 0, následující instrukce se provede.

K často používanému testu registru na 0 se používá trik:

MOVF reg, 1

BTFSS STATUS,2 ... (nebo BTFSC.....)

GOTO někam

další instrukce

Tento trik využívá skutečnosti, že instrukce přesunů **nastavují** stavový registr. MOVF registr,1 znamená přesunout hodnotu registru sám do sebe. Instrukce ovšem přitom nastaví stavový registr, takže pak lze testovat jeho jednotlivé bity. Se stavovým registrem se zachází stejně jako s kteroukoli jinou buňkou paměti, tedy testovat bit Z lze instrukcí BTFS STATUS,2 (bit Z je 2. bit STATUS registru). Nezapomeňte, že bit Z se nastaví do 1, pokud výsledek předcházející operace byl 0.

Při určení funkce programu je třeba myslet hlavou.

předpokládejme, že obsah reg = 0

po provedení MOVF se nastaví bit Z do 1 (protože hodnota registru byla 0 )

BTFSS přeskočí GOTO někam (přeskoč, pokud je bit nastaven)

tedy pro reg = 0 se GOTO neprovede, pro reg různé od 0 se GOTO provede

pokud tedy nebyla hodnota buňky 0 , skáčeme na někam, pokud byla 0, pokračujeme další instrukcí

Podmíněné skoky lze udělat pouze kombinací

BTFSS (nebo BTFSC)

GOTO

další instrukce

První instrukce testuje bit (lhostejno, zda portu, registru) a podle jeho hodnoty buď přeskočí nebo nepřeskočí následující GOTO.

**DECFSZ reg , 0**  ; instrukce odečte 1 od obsahu registru reg. Výsledek uloží do W, tedy obsah reg se nemění

Potom testuje výsledek – pokud je roven 0, přeskočí se jedna následující instrukce

(decrement and skip if zero )

**DECFSZ reg , 1** ; instrukce odečte 1 od obsahu registru reg. Výsledek uloží do reg, tedy obsah reg se o 1 sníží.

Potom testuje výsledek – pokud je roven 0, přeskočí se jedna následující instrukce

(decrement and skip if zero )

**INCFSZ reg , d ;**  podobně jako DEC, ale k registru se 1 přičte (increment and skip if zero )

**Důležité !** vždy se nejdříve udělá DEC (INC), a teprve potom se testuje výsledek na nulovou hodnotu !

Ke konstrukci cyklů se hodí posloupnost instrukcí:

MOVLW počet opakování

MOVWF počítátko

...............

ZAC: ...........

..........

...instrukce, které se mají opakovat .........

............

DECFSZ počítátko

GOTO ZAC

.............

**CALL aa** ; volání podprogramu na adrese aa

Při volání podprogramu se provedou následující kroky:

1. uloží se návratová adresa do zásobníku . Návratová adresa je číslo, které je právě v registru PC. Při provádění instrukce PC již inkrementoval, takže obsahuje adresu následující instrukce. Tato hodnota se uloží do zásobníku – paměť 16x15. Buňka, do které se má návratová adresa uložit, je označena čtyřbitovým ukazatelem zásobníku SP.
2. hodnota ukazatele zásobníku se zvýší, takže SP ukazuje na další volnou buňku v zásobníku. Protože zásobník má pouze 16 buněk, lze použít pouze 16 úrovní volání podprogramů a přerušení. Pokud bude úrovní více, SP přeteče a bude ukazovat na nesprávnou návratovou adresu, takže při RETURN se program vrátí někam jinam.

1. provede se skok na adresu aa (tedy GOTO aa). To ovšem platí jen přibližně: registr PC je naplněn 15bitovým číslem, jehož bity 0 – 10 jsou dány adresou aa (tedy aa je v intervalu 0 – 2047) a bity 11, 12, 13,14 jsou okopírovány z bitů 3,4,5,6 registru PCLATH. Tento mechanismus umožňuje stránkování paměti (viz kap. 3.5 strana 41 datasheetu ). Náš procesor má paměť programu 4k, uvedený mechanismus je tedy nutno brát v úvahu při přechodu mezi dolními a horními 2k paměti (mezi blokem adres 0x000-0x3ff a 0x400-0x7ff) !!!Pozor ! Platí i pro GOTO !!!!

**RETURN** ; návrat z podprogramu

Při návratu se provedou následující akce:

1. Do registru PC se uloží číslo ze zásobníku, z buňky, na kterou ukazuje SP.
2. SP se vrátí na předcházející úroveň (o jenno sleze, tedy decf SP)
3. protože v PC je hodnota návratové adresy, bude program pokračovat na místě, odkud byl podprogram zavolán

Je jasné, že každému CALL musí odpovídat RETURN, jinak bude program dělat nesmysly.

Zásobník ani SP nejsou programátorovi přístupné, nelze do nich nijak zasahovat.

**RETLW k ;** návrat s číslem v registru W. totéž jako RETURN, navíc se číslo k zapíše do W. Této instrukce

lze použít při práci s tabulkami.

**RETFIE** ; návrat z přerušení. Totéž jako RETURN, navíc se povolí přerušení, tedy GIE = 1. Samozřejmě

lze této instrukce použít i pro návrat z podprogramu, pokud přitom potřebujeme povolit přerušení.

**Stavový registr – STATUS – adresa 03h v kterékoli bance**

Kap. 3.4 , stana 23

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  |  |  | R | R | R/W | R/W | R/W |
|  |  |  | **\*TO** | **\*PD** | **Z** | **DC** | **C** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | **C** – Carry – přetečení – z nejvyššího bitu výsledku u aritmetických instrukcí |
| **DC** – poloviční přenos – mezi dolní a horní polovinou výsledku | |
| **Z** – Zero – nastaví se do 1, pokud výsledek aritmetické nebo logické  operace je roven 0. Obyčejný přesun – MOVF - také nastavuje Z bit !  U odčítání se do C uloží negovaná hodnota přenosu | | |
| **\*PD** – power Down bit – 1 po power-up nebo po CLRWDT 0 po SLEEP | | | |
| **\*TO** – Time-Out bit ----- 0 – WDT přetekl a udělal RESET  1 jinak | | | | |
| Nic | | | | | | |
| Nic | | | | | | | |