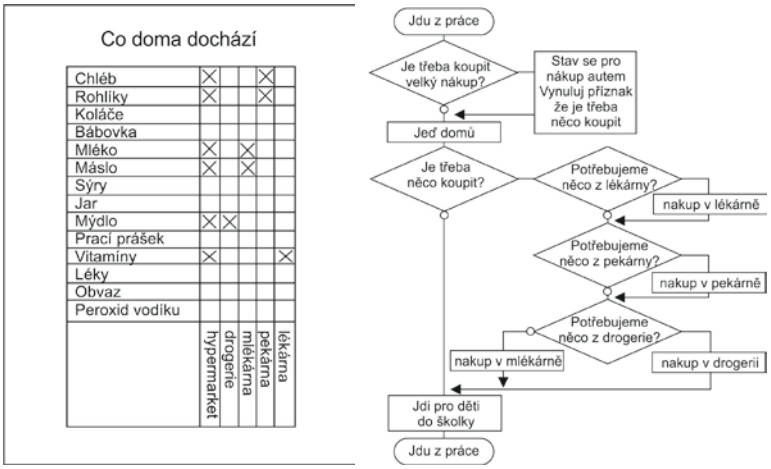
Rošťáci roboťáci a Assembler 1- Flagy aneb od vlajky k programu

Zásadním pojmem v programování, kterému často nerozumíme, je pojem „Flag“.

Definice říká: „V informatice je flag (příznak) hodnotou, která slouží jako jakýsi signál pro funkci nebo proces. Hodnota příznaku se používá k určení dalšího kroku programu. Flagy jsou často binární příznaky, které obsahují booleovskou hodnotu (true nebo false). Nicméně, ne všechny příznaky jsou binární, což znamená, že mohou uložit rozsah hodnot, například volba A = 01, volba B = 10 apod.“ [https://techterms.com/definition/flag]

I když tento pojem neznáte, stejně se příznaky řídíme i my při našem rohodování. Jeden malý příklad za všechny.

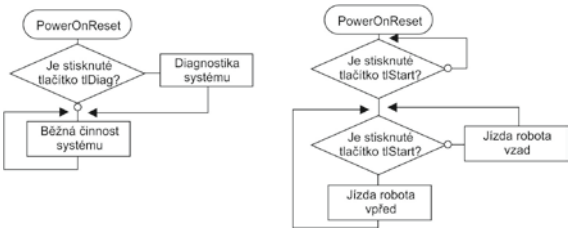
Moje maminka má na ledničce magnetem přichycený seznam s věcmi, co nakupujeme. Do něj si vždy zaznamenává, co doma dochází a je třeba koupit.



Když jde ráno z domu, vždy do něj koukne a podle toho, co je třeba, zajede do hypermarketu, kde mají vše, ale je to daleko a musí se tam jet autem, nebo zajde za roh na náměstí a nakoupí v příslušných obchodech. Než však oběhne všechny obchody, kde bývají fronty, nákupy trvají mnohem déle a musí celý nákup nést v tašce. Do hypermarketu jezdí tedy jen na větší nákupy, drobné nákupy řeší v jednotlivých obchodech. Ovšem i v hypermarketu jí pomáhá seznam při nakupování. Jde z oddělení do oddělení a postupně díky seznamu nakoupí vše bez zbytečného přebíhání. Díky příznakům se tedy můžeme snadno a dynamicky rozhodovat.

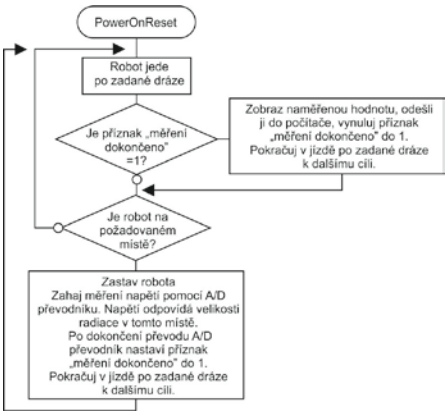
Rozhodování bez použití příznaků

Typickým příkladem rozhodování bez použití příznaků je například čekání na stisik klávesy a po jejím stisku přechod na nějakou činnost. Takto jsme řídili program doposud.



Použití příznaku pro řízení chodu programu

Mějme například A/D převodník. Potřebujeme, aby změřil napětí na svém vstupu, a změřené napětí musíme zobrazit na display robota, nebo naměřenou hodnotu poslat do počítače. Po celou dobu je však třeba řídit robota, aby jel po zadané trase. Když odstartujeme měření, A/D převodník vezme vzorek napětí (jeho okamžitou hodnotu) a zahájí převod. To nějakou dobu trvá. Přitom již robot může jet k dalšímu cíli měření. Naměřená datat odešle hned, jak je získá od A/D převodníku.



Úvod do praktického využití příznaků

Nejčastěji používané příznaky, jsou příznaky jádra mikrokontrolérů, které se nachází v registru nazvaném STATUS regist. Postupně se s nimi seznámíme. První, který si zaslouží naši pozornost je příznak Z. Budeme si zde ukazovat současně dva mikrokontrolery. 16F88 je plně programově kompatibilní s WJECassemblerPICAXE18M2 a budeme je programovat s mladšími žáky pomocí PICAXE Programming editotru 6. Mikrokontroléry PIC16F1847 budeme programovat se staršími žáky pomocí programovadla PICKIT3. Symulovat budeme programa v MPLAB a MPLABX – profesionální vývojové prostředí firmy firmy Microchip.

STATUS regist (address 03h, 83h, 103h a 183h)

IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	PIC16F88
7							0	

bit 2 Z: Zero bit
=1, je-li výsledek aritmetické, nebo logické operace nula
=0, není-li výsledek aritmetické, nebo logické operace nula

STATUS regist (address 03h, 83h, 103h.....F83h)

----	----	----	$\overline{TO}$	$\overline{PD}$	Z	DC	C	PIC16F1847
7							0	

bit 2 Z: Zero bit
=1, je-li výsledek aritmetické, nebo logické operace nula
=0, není-li výsledek aritmetické, nebo logické operace nula

Instrukce, které použijeme:

Move f – přesuň obsah registru

MOVF ***f,d***
provede: (f) do (d)

Instrukce přenese (zkopíruje) obsah registru, jehož adresa je f do cíle d. **Instrukce nastavuje příznak Z bit.** Je-li obsah registru = 00h, pak nastaví Z bit do 1. Není-li 00h, pak Z bit vynuluje. Je-li d=0, pak cílovým registrem je W registr - zkopíruje tedy obsah registru f do W registru. Je-li d=1, pak se obsah registru f uloží zpět do f registru, tedy nezmění obsah žádného z těchto registrů. Tato operace se zdá zbytečná, ale není, instrukce nastavuje příznak Z bit registru SWF podle toho, jaký je obsah registru f.

Move W to f – přesuň obsah W registru do registru

MOVWF ***f***

provede: (W) do (f)

Instrukce přenese (zkopíruje) obsah registru W do registru, jehož adresa je f.
Instrukce nenastavuje žádný příznak.

Move literal to W – přesuň konstantu do W registru

MOVLW ***k***

provede: k do (W)

Instrukce naplní obsah W registru konstantou. **Instrukce nenastavuje žádný příznak.**

Decrement f – odečti jedničku od registru

DECF ***f,d***

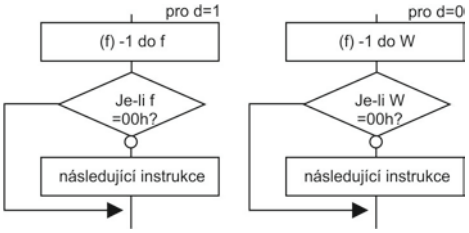
provede: (f)-1 do (d)

Instrukce odečte 1 od obsahu registru f. Je-li d = 0 pak výsledek zapíše do W registru. Je-li d = 1, výsledek zapíše zpět do registru (sníží hodnotu v registru o 1). **Instrukce nastavuje příznak Z bit.** Je-li výsledek po odečtení 1 roven 00h, nastaví Z bit do 1. Není-li 00h, pak Z bit vynuluje.

Decrement f, Skilp if 0 – sniž obsah registru o 1, je-li výsledek 00h, pak přeskoč následující instrukci

DECFSZ ***f,d***

provede: (f)-1 do (d)



Instrukce odečte 1 od obsahu registru f. Je-li d = 0 pak výsledek zapíše do W registru. Je-li d = 1, výsledek zapíše zpět do registru (sníží hodnotu v registru o 1). **Instrukce nenastavuje příznak Z bit.** Je-li výsledek po odečtení 1 roven 00h, pak program přeskočí následující instrukci. Není-li 00h, pak se následující instrukce provede.

Return from Subroutine – návrat z podprogramu

RETURN

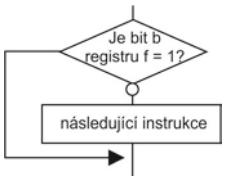
provede: Návratová adresa ze stacku do PC

Začne se provádět program od adresy, která byla jako poslední uložená do stacku, tedy adresy, následující za instrukcí volání posledního podprogramu.

Bit test f, Skip if Set – Testuj bit registru, přeskoč následující instrukci, je-li bit = 1

BTFSS ***f,b***

provede: přeskoč následující instrukci je-li (f<b>)=1

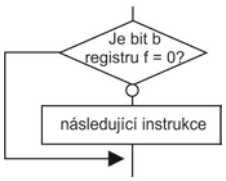


Instrukce testuje bit b registru f. Je-li jeho hodnota 1, přeskočí se následující instrukce.

Bit test f, Skip if Clear – Testuj bit registru, přeskoč následující instrukci, je-li bit = 0

BTFSC ***f,b***

provede: přeskoč následující instrukci je-li (f<b>)=0

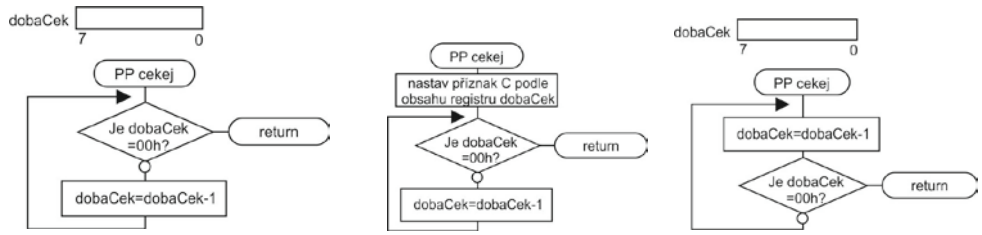


Instrukce testuje bit b registru f. Je-li jeho hodnota 0, přeskočí se následující instrukce.

Podprogramy čekání

Tento způsob čekání není v systémech řízených v reálném čase běžně používaný, protože po celou dobu čekání procesor nic jiného nedělá, než počítá, a nereaguje na žádné vnější podmínky. Doposud jsme tento typ čekání používali nejčastěji. Podívejme se, jak se takovéto čekání může realizovat v assembleru.

Na následujícím obrázku jsou dva způsoby dekrementace obsahu registru při počítání průchodů. První způsob je založen na tom, že nejdřív otestuje, je-li registr doby čekání již 00h. Pojkud není, edečte se jednička a opět se testuje jeho obsah. Druhý pak praguje tak, že nejdříve se dekrementuje obsah čítače doby čekání a až pak se testuje, je-li roven 00h.



Úkol 1: Popište rozdíl mezi těmito dvěma algoritmy čekání – v čem s liší?

Úkol2: Upravte vývojové diagramy tak, aby bylo možné čekání realizovat pomocí výše uvedených instrukcí.

Úkol 3: Odsymulujeme činnost obou způsobů řešení této čekací smyčky a při symulaci zjistíme, jaká je závislost doby čekání na konstantě uložené do registru dobaCek pře zavoláním podprogramu.

Úkol 4: Jak je tato doba závislá na kmitočtu generátoru hodin mikrokontroléru? Jaká maximální bude při použití generátoru hodin fosc = 4 MHz?