Není čekání, jako čekání

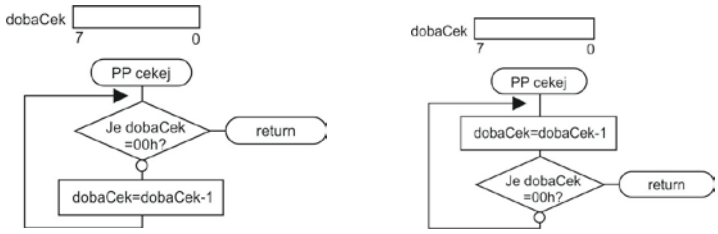
Čekání ve smyčce

Tento způsob čekání je podobný, jako když hrajeme na schovku. Jeden piká a ostatní se schovávají. Kdo piká počítá například do deseti. Jedna, dva, tři ... až deset. Lépe však obráceně od desíti do nuly. Obojí čekání je stejně dlouhé. Počítá-li však deset, devět, osm ... dva, jedna nula-už jdu, ti co se schovávají, ví stále, kolik času ještě mají. Po dobu počítání ten kdo piká, nic jiného nedělá, jen počítá. Nereaguje na to, co se kolem něho děje.

Tento způsob čekání realizuje například instrukce ***PAUSE***, kterou jsme pro čekání dosud v Basicu používali.

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, odečte se jednička a opět se testuje jeho obsah. Druhý pak pracuje tak, že nejdříve se dekrementuje (sníží o 1) obsah čítače doby čekání a pak se testuje, je-li roven 00h.

Před voláním podprogramu musí být naplněn registr ***dobaCek*** naplněn počáteční hodnotou.



Je-li ***dobaCek*** osmibitová proměnná (jeden byte – čte se bajt), pak její hodnota se bude pohybovat v rozsahu 0 až 255. Při přičítání jedničky se pohybuje hodnota ...254,255,0,1,2..., při odečítání jedničky se mění hodnota následovně ... ...3,2,1,0,255,254,253... Když potřebuji pochopit, jak algoritmus pracuje, nemusíme počítat s těmito čísly.

Můžeme pro názornost pracovat s jednou desítkovou číslicí, tedy s čísly 0 až 9. Když přičteme 1 k 9, přejde se na 10. Jednička je v tomto případě už druhá desítková číslice. Ta nás už nezajímá. Tak jsme poznali, že jedna desítková číslice při inkrementaci, nebo-li přičítání jedničky, mění hodnoty následovně ... 8,9,0,1,2,3,4,5,6,7,8,9,0,1,2, ... Podobně při dekrementaci (odečítání jedničky) .....2,1,0,9,8,7,6,5,4,3,2,1,0,9,8,....

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

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

MOVF	*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.

MOVWF	*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.

MOVLW	*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.**

DECF	*Decrement f*	*odečti jedničku od registru*

DECF ***f,d***

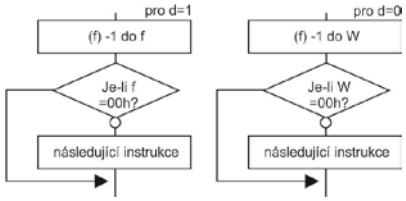
provede: (f)-1 do (d)

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.

DECFSZ	*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	*Return from Subroutine*	*návrat z podprogramu*

RETURN

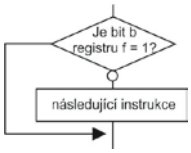
provede: Návratová adresa ze stacku do PC

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

BTFSS	*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

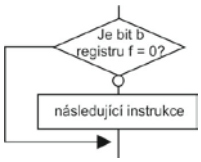


Instrukce testuje bit b registru f. Je-li jeho hodnota 1, přeskočí se následující instrukce.

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

BTFSC ***f,b***

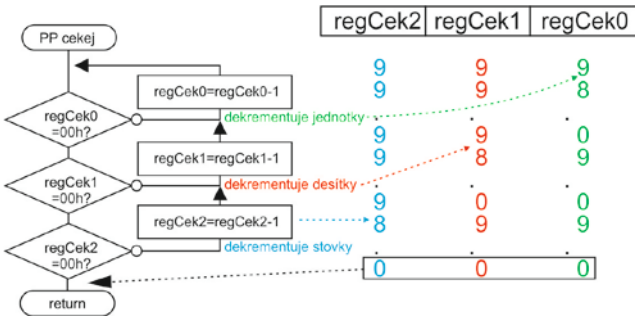
provede: přeskoč následující instrukci je-li (f<b>)=0



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

Úkol 3: Odsymulujeme činnost obou způsobů řešení této čekací smyčky.

Prodloužení čekání ve smyčce

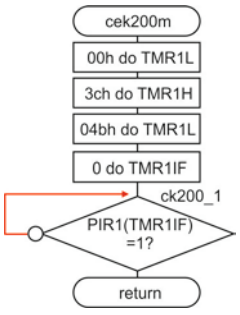


Pro ověření činnosti algoritmu použijeme desítkové číslo 999. Pokud poběží algoritmus správně pro desítkové číslo, poběží správně i pro 24 bitové binární číslo.

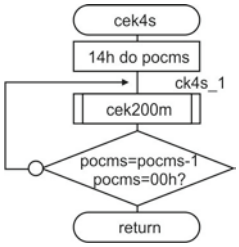
Potřebujeme-li zvětšit počet cyklů, kolik se má čekat musíme použít víc bytů doby čekání. Jeden byte zajistí hodnotu 0 až 255, 2 byty pak 0 až 65535 a 3 byty 0 až 16777215.

! POZOR ! ***Tento způsob čekání v systémech řízených v reálném čase musíme používat velmi opatrně. Po celou dobu čekání procesor nic jiného nedělá, než počítá, a nereaguje na žádné vnější podmínky!***

Proto delší doby čekání raději realizujeme s použitím čítačů a využitím toho, že signalizují své přetečení (přechod z maximální hodnoty do nuly). Například čítač/časovač TMR1 v mikrokontrolérech PIC firmy Microchip. Při přetečení nastaví příznak TMR1IF v registru PIR1. Procesor může normálně pracovat a cyklicky kontroluje stav tohoto příznaku. V okamžiku, kdy zjistí, že je příznak v jedničce, uplynul čas čekání. Činnosti, které má dělat v době čekání umístíme do červeně označené smyčky.



Pro zvětšení doby čekání můžeme program upravit následujícím způsobem.



Poznámka: *Pro zjištění, že byl příznak nastaven do 1, můžeme využívat přerušení, tím se ale budeme zabývat později.*