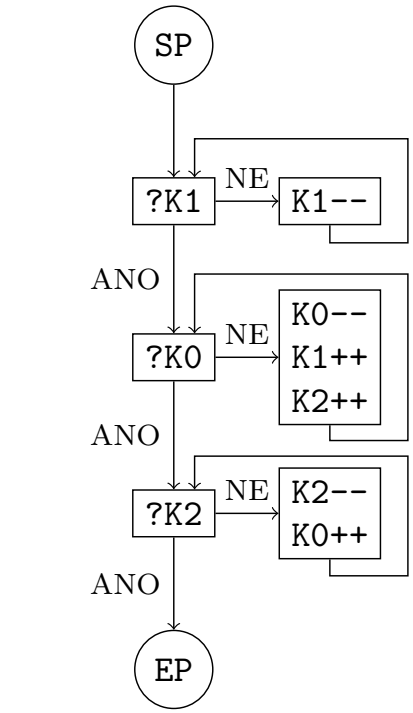
Řešení úlohy č. 5
Kuličkový počítač #2

Ve druhém kole bylo úkolem navrhnout čtyři podprogramy pro kuličkový počítač. Zadání bylo postaveno tak, aby se již hotové podprogramy postupně daly využít v rámci dalších podprogramů. Obecně se dá podotknout, že nově zavedený koncept podprogramů nám velmi zjednodušil návrh programů pro kuličkový počítač.

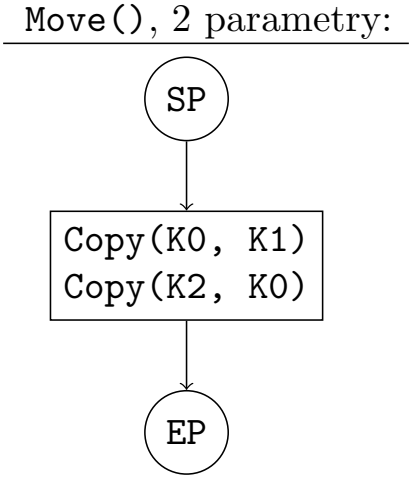
Jako praktický příklad zmiňme operaci „rozkopírovávání“ kyblíčku. Tj. operaci, při které dojde ke zkopírování obsahu nějakého kyblíčku K_i do kyblíčku $K_j, i \neq j$, s těmi požadavky, že obsah kyblíčku K_i zůstane zachován a původní obsah kyblíčku K_j bude přepsán. Podprogram pro vykonání této operace nám výrazně pomůže zkrátit zápis programů pro kuličkový počítač. A navíc si můžeme rozmyslet, že nám tento podprogram může posloužit i mimo jeho původní záměr. Například pro vynulování obsahu kyblíčku stačí přesunout obsah jistě nulového kyblíčku do kyblíčku, který chceme vynulovat. Další užitečnou funkci tohoto podprogramu zmíníme níže, nyní si ukažme, jak takovýto podprogram na „rozkopírovávání“ kyblíčku vypadá. Při jeho implementaci použijeme jednoduchý trik, kdy kuličky z kyblíčku K_0 rozkopírujeme nejen do cílového kyblíčku K_1, ale také do pomocného kyblíčku K_2 (který je na začátku vykonávání podprogramu zaručeně prázdný), z nějž poté kuličky přesuneme zpět do kyblíčku K_0.

`Copy()`, 2 parametry:



Komentář: Podprogram bere jako parametry dva kyblíčky K_0 a K_1. Pokud je K_1 neprázdný, dojde k jeho vyprázdnění. Poté jsou kuličky z K_0 překopírovány do K_1, za zachování původního obsahu kyblíčku K_0.

Automaticky tak díky výše uvedenému popisu dostáváme také podprogram pro přesun obsahu kyblíčku předaného podprogramu jako K_0 do kyblíčku předaného jako K_1. Jednoduše stačí provést překopírování obsahu z K_0 do K_1 a poté překopírovat obsah nějakého prázdného kyblíčku do K_0. Zde opět využijeme toho, že je garantováno, že všechny pomocné kyblíčky v rámci podprogramu jsou na začátku vykonávání podprogramu nulové a použijeme tak např. kyblíček K_2. Podprogram pro přesouvání obsahu kyblíčku tedy vypadá takto:

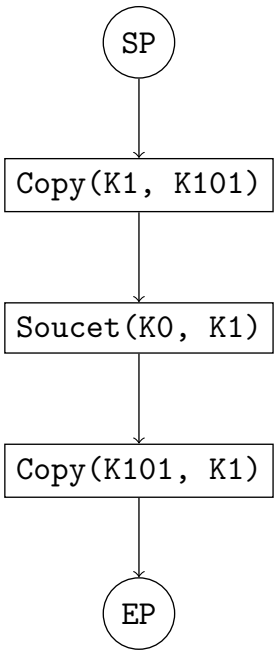


Komentář: Podprogram bere jako parametry dva kyblíčky K_0 a K_1. Pokud je K_1 neprázdný, dojde k jeho vyprázdnění. Poté jsou kuličky z K_0 přemístěny do K_1, tj. K_0 po vykonání podprogramu neobsahuje žádnou kuličku.

Pří návrhu a používání podprogramů však musíme vzít v úvahu důležitý faktor. A to, že parametry se podprogramům v kuličkovém počítači předávají tzv. odkazem. To znamená, že počty kuliček v kyblíčcích předaných podprogramům se mohou při vykonávání podprogramu změnit. Zejména nás tato situace může tížit, pokud doslovně použijeme programy pro aritmetické operace navržené v prvním kole, či podprogramy z tohoto kola, jelikož ty ve své základní podobě garantují pouze to, že bude možno naleznout návratovou hodnotu v kyblíčku K_0 (tj. v prvním kyblíčku, který danému podprogramu předáme). S tímto problémem se můžeme vyrovnat dvěma způsoby. První způsob je neupravovat dané podprogramy, ale vždy si pečlivě ohlídat, co daný podprogram v konkrétním kyblíčku vrátí (a zda nám tak např. neznehodnotí obsah předaného kyblíčku). Druhý způsob využívá výše uvedenou operaci `Copy()` pro zálohu obsahu předaných kyblíčků. Konkrétně upravíme podprogramy tak, aby vždy při začátku vykonávání podprogramu zkopíruje všechny parametry na vstupu podprogramu do nějakých jistě volných kyblíčků. Následně se provede tělo podprogramu a před jeho ukončením se obsah pomocných kyblíčků překopíruje zpět do kyblíčků, které sloužily jako parametry podprogramu; samozřejmě za zachování návratových hodnot, které byly vypočítány v rámci podprogramu.

Tento postup můžeme použít pro libovolný podprogram, kdy jej pouze obalíme do zmíněných bloků operace `Copy()`. Abychom se však nemuseli psát se všemi těmito kopírováními, budeme pro další popis podprogramů uvažovat, že podprogram, jehož název obsahuje příponu `_c`, zachovává obsah všech jemu předaných kyblíčků, až na kyblíčky použité pro návratové hodnoty. Jako příklad uveďme konstrukci podprogramu `Soucet_c()` k podprogramu `Soucet()`. Jako podprogram `Soucet()` uvažujeme program na sečtení dvou čísel z minulého kola. Podprogram `Soucet_c()` bude mít stejný počet parametrů a pouze zachová počet kuliček v nenávratových kyblíčcích. Jelikož má podprogram `Soucet()` dva parametry a v prvním kyblíčku K_0 vrací výsledek, budeme chtít zachovat obsah druhého předaného kyblíčku K_1.

`Soucet_c()`, 2 parametry:



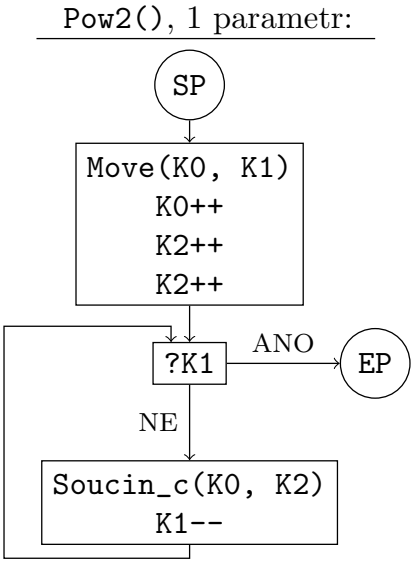
Komentář: Podprogram bere jako parametry dva kyblíčky K_0 a K_1. V kyblíčku K_0 Vrací stejný výsledek jako dříve definovaný podprogram `Soucet()`*, jen garantuje zachování druhého parametru. Pozn.: Konstantu 100 jsme zvolili jako dostatečně vysoké číslo kyblíčku, který již určitě nebude v rámci podprogramu* `Soucet()` *využit. Pokud bychom chtěli zachovat více parametrů, tak jednoduše použijeme pro i-tý parametr (tj. v podprogramu kyblíček K_i) kyblíček K_{100+i}.*

Tímto jsme obsáhli všechny zřejmé výhody i nevýhody při návrhu podprogramů pro kuličkový počítač a můžeme se tak vrhnout na řešení soutěžních podprogramů.

Podprogram pro výpočet určité mocniny čísla 2

Pokud vezmeme v úvahu již vypracované podprogramy, lze tuto úlohu považovat za spíše rozcvičkovou. Jednoduše si totiž uvolníme místo pro výsledek v kyblíčku K_0 tak, že přesuneme danou mocninu z kyblíčku K_0 do kyblíčku K_1. Následně inicializujeme kyblíček K_0 na jednu kuličku (jelikož $2^0 = 1$) a postupně násobíme již dosažený výsledek v kyblíčku K_0 číslem 2, které si připravíme v kyblíčku K_2.

V rámci předchozí diskuze podotkneme, že využijeme upravenou verzi podprogramu pro násobení dvou čísel z předchozího kola tak, aby zachovával druhý jemu předaný parametr, tj. použijeme podprogram `Soucin_c()`. Zároveň si povšimněme, že nemá smysl definovat podprogram `Pow2_c()`, neboť jeho rozhraní je pouze v jednom parametru. Implementace pro kuličkový počítač vypadá následovně.



Komentář: Podprogram bere jako parametr jeden kyblíček K_0, reprezentující číslo a. Jako výsledek vrací také v kyblíčku K_0 číslo 2^a.

Podprogram pro výpočet bitového posunu vlevo/vpravo

Pro jednoduchý návrh podprogramu pro bitový posun je zapotřebí si povšimnou důležitého faktu. A to, že bitový posun čísla vlevo, resp. vpravo, je ekvivalentní násobení, resp. dělení daného čísla číslem 2. Důvod je jednoduchý. Uvažujme dvojkový zápis nějakého čísla x: $(b_n b_{n-1} \ldots b_2 b_1 b_0)_2$, kde $b_i \in \{0, 1\}$. Tento zápis značí, že:

$$x = 2^n b_n + 2^{n-1} b_{n-1} + \ldots + 4b_2 + 2b_1 + b_0$$

Při posunu čísla x vlevo o jednu pozici tak dostaneme číslo x' takové, že jeho binární zápis bude $(b_n b_{n-1} \ldots b_2 b_1 b_0 0)_2$ (nově vzniklou pozici doplníme nulou). Jinými slovy, bude platit, že:

$$x' = 2^{n+1} b_n + 2^n b_{n-1} + \ldots + 8b_2 + 4b_1 + 2b_0$$

Nyní z tohoto zápisu vytkneme číslo 2 a povšimneme si, že výraz v závorce je vlastně číslo x. Tedy:

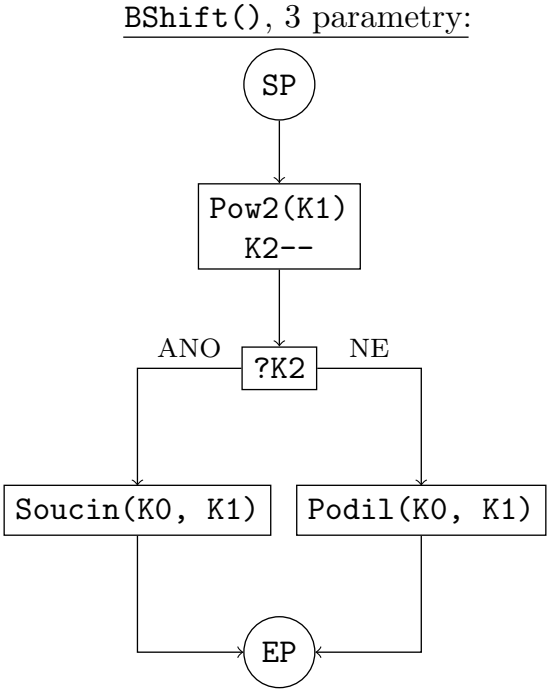
$$x' = 2^{n+1} b_n + 2^n b_{n-1} + \ldots + 8b_2 + 4b_1 + 2b_0 = 2(2^n b_n + 2^{n-1} b_{n-1} + \ldots + 4b_2 + 2b_1 + b_0) = 2x$$

Podobnou úvahu lze vést také pro posun vpravo. Jediný rozdíl v tom, že při posunu vpravo se z binárního zápisu daného čísla ztrácí koeficient b_0. Ten však upravuje výsledek pouze o jedničku a tedy výsledkem bitového posunu o jednu pozici vpravo nebude přímo $\frac{x}{2}$, ale $\left\lfloor \frac{x}{2} \right\rfloor$. Jinými slovy je bitový posun čísla o jednu pozici vpravo ekvivalentní *celočíselnému* dělení daného čísla číslem 2. To nám však vůbec nevadí, jelikož v minulém kole jsme navrhovali program právě pro celočíselné dělení.

Jediné co nám nyní chybí rozmyslet jsou posuny o více než o jednu pozici. Nabízí se opakovaně zadané číslo násobit/dělit číslem 2, např pro posun o i pozic celkem i-krát. To je ale přeci to samé, jako jednorázově zadané číslo vynásobit/vydělit číslem 2^i. Využijeme tak předchozího podprogramu.

S těmito znalostmi je už samotný návrh podprogramu jednoduchý. V kyblíčku K_1 si vypočítáme příslušnou mocninu čísla 2. Počet kuliček v kyblíčku K_2 snížíme o 1, abychom jednoduše rozhodli o posunu vlevo/vpravo pomocí testování prázdnoty daného kyblíčku. Nakonec pouze číslo v kyblíčku K_0 vynásobíme/vydělíme číslem v kyblíčku K_1 a je hotovo. Pro násobení a dělení využijeme programů z předchozího kola.

Poznamenáme, že zde není zapotřebí využít zálohovací verze podprogramů `Soucin()` a `Podil()`, jelikož po jejich provedení okamžitě vystupujeme z těla podprogramu. Podprogram `Bshift_c()` pro budoucí využití smysl dává, jelikož kromě výsledku v kyblíčku K_0 může být příhodné zachovat původní hodnotu parametrů K_1 a K_2.



Komentář: Podprogram bere jako parametr tři kyblíčky K_0, K_1 a K_2 reprezentující nějaká čísla a, b, c. Musí platit, že $c \in \{1, 2\}$. Jako výsledek vrací podprogram v kyblíčku K_0 číslo, které vznikne posunem čísla a o b bitů směrem daným číslem c: pro $c = 1$ doleva, pro $c = 2$ doprava.

Podprogram pro výpočet bitové operace AND

Pro provedení operace AND budeme potřebovat přístup k bitům zadaných čísel. K tomu je ideální právě operace AND v kombinaci s bitovou maskou co určuje, které bity z daného čísla chceme „vyzobnout“ (viz program pro ověřování palindromicity). Ale co si počít, když máme tuto operaci teprve naimplementovat? Zde se opět vrátíme k tomu, co vlastně binární zápis čísla reprezentuje. Pro zápis nějakého čísla x: $(b_n b_{n-1} \dots b_2 b_1 b_0)_2$ víme, že

$$x = 2^n b_n + 2^{n-1} b_{n-1} + \ldots + 4b_2 + 2b_1 + b_0$$

Pokud se do vztahu pozorně zadíváme tak si můžeme všimnout toho, že jediný člen, který není sudý, je člen uvozený koeficientem b_0. Z toho plyne, že to, zda je číslo x sudé nebo liché, závisí pouze na nenulovosti koeficientu b_0. Pro $b_0 = 0$ je x sudé, pro $b_0 = 1$ je x liché. Rozhodnout o sudosti/lichosti čísla ale už v rámci kuličkového počítače umíme, jelikož jsme v minulém kole implementovali program pro zjištění zbytku po dělení daného čísla jiným číslem. Pro rozhodnutí o sudosti/lichosti čísla x (a v důsledku pro zjištění koeficientu b_0) tedy stačí zkoumat zbytek po dělení čísla x číslem 2.

Umíme tedy zjistit koeficient b_0 ve binárním zápise daného čísla. Jelikož ale máme k dispozici také bitový posun vpravo, není nic jednoduššího, než opakovaně zadané číslo (či v případě operace AND zadaná čísla) posouvat o jeden bit vpravo a zkoumat nově-vzniknuvší koeficienty b_0.

Poslední věc, kterou musíme rozmyslet jak tento postup aplikovat pro dvě čísla a jak zkonstruovat výsledek operace AND mezi nimi. Jednoduše budeme iterovat tak dlouho, dokud jedno z čísel v kyblíčcích K_0 nebo K_1 nebude nulové. Poté si v pomocných kyblíčcích K_3 a K_4 spočítáme zbytek po dělení daných čísel číslem 2. Pokud budou obě dvě tato čísla nenulová,

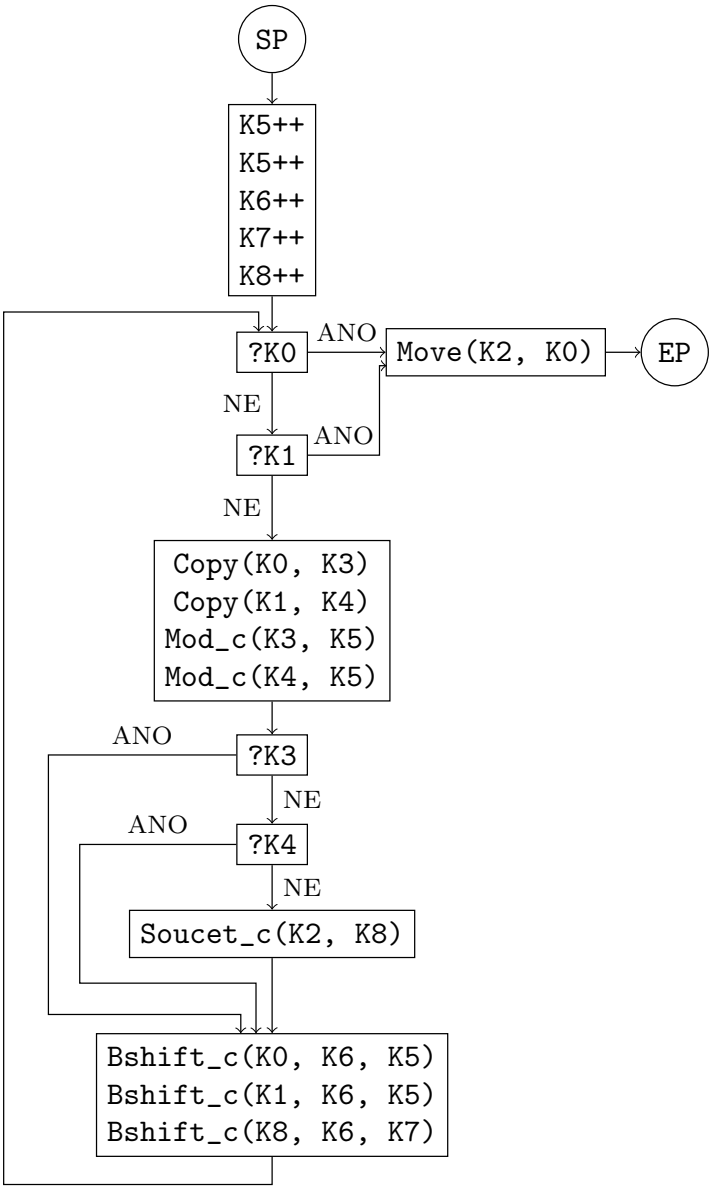
musíme k výsledku přidat jedničku na správnou pozici. K tomu nám stačí si během výpočtu v kyblíčku K_8 držet příslušnou mocninu čísla 2. Proč? Pokud je totiž číslo i-tou mocninou dvojky, je ve tvaru 2^i a ve dvojkovém zápise má tak nenulový právě jeden bit b_i! To protože jedině takto bude platit, že:

$$2^n b_n + 2^{n-1} b_{n-1} + \ldots + 2^i b_i + \ldots + 4b_2 + 2b_1 + b_0 = 2^i b_i = 2^i$$

Pokud tedy chceme přidat k výsledku bit na i-tou pozici (číslujeme od nuly), přičteme k němu číslo 2^i. Pozor, toto funguje jen tehdy, není-li již na i-té pozici ve výsledku jedničkový bit – což v tomto případě je zaručeno, že není.

Můžeme tedy přejít k samotné implementaci. Ta opět umožňuje variantu `And_c()`, jelikož může dávat smysl zachovat další parametr. Zároveň v rámci operace využijeme spoustu již hotových podprogramů ve verzích se zachováním parametrů. Kyblíčky K_5, K_6 a K_7 využijeme pro uložení konstant potřebných pro průběh výpočtu. V kyblíčcích K_6 a K_7 uchováme číslo 1 jako parametr bitových posunů a v kyblíčku K_5 uchováme číslo 2 pro zjišťování zbytku po dělení číslem 2 a také jako parametr bitových posunů.

`And()`, 2 parametry:



Komentář: Podprogram bere jako parametr dva kyblíčky K_0 a K_1, reprezentující čísla a a b. Nad binárními zápisy těchto čísel v těchto kyblíčcích aplikuje operaci AND, tj. výsledek v kyblíčku K_0 bude a&b.

Podprogram pro ověření palindromicity binárního zápisu čísla

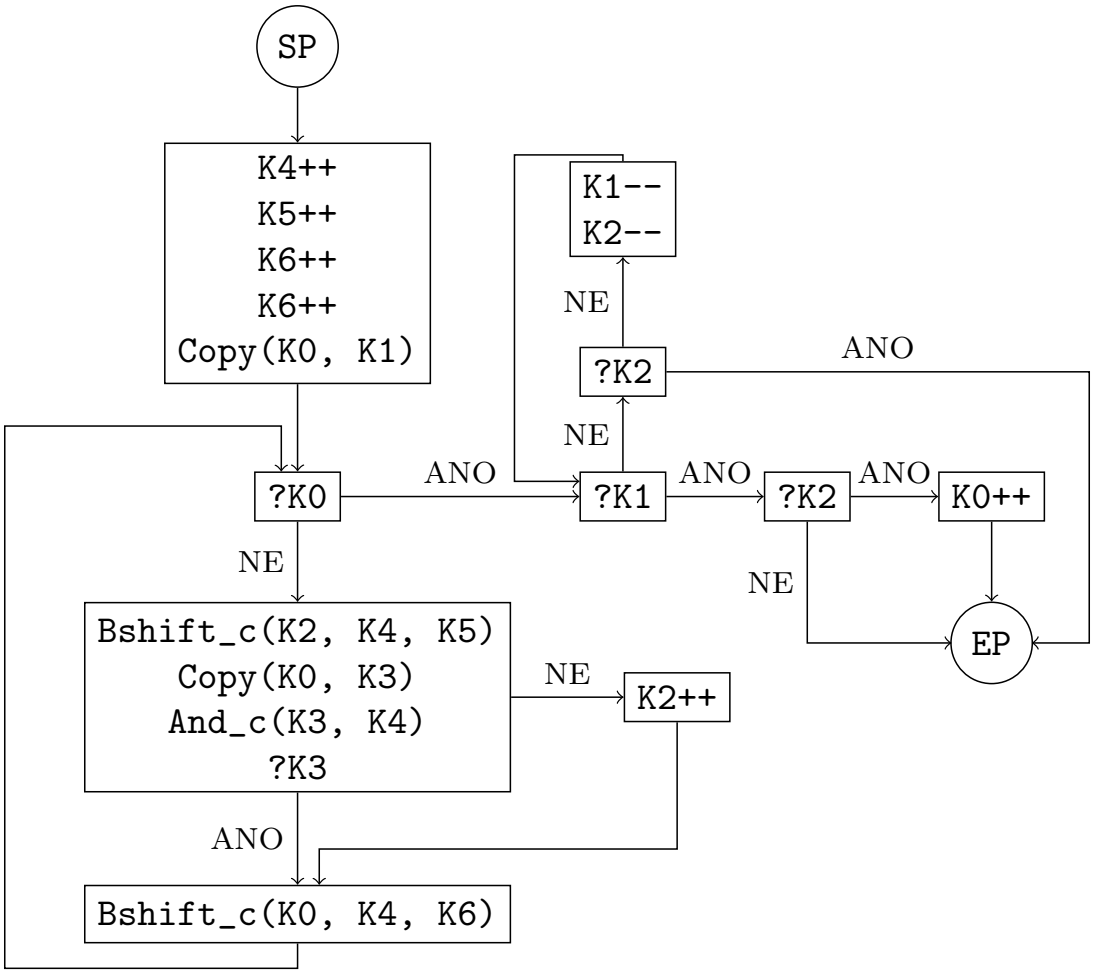
Pro ověření palindromicity binárního zápisu zadaného čísla nám poslouží následující úvaha – je-li binární zápis čísla palindromický, čte se stejně zepředu i zezadu. V tom případě musí platit, že pokud binární zápis daného čísla otočíme (tj. zapíšeme jeho koeficienty v opačném pořadí), neměl by se od neotočeného zápisu čísla nijak lišit. Jelikož jsme ale popsali přímý vztah mezi binárním zápisem čísla a jeho numerickou hodnotou tak platí, že binární zápis čísla jednoznačně určuje jeho hodnotu. Pro ověření palindromicity nám tedy stačí porovnat hodnoty čísla na vstupu a čísla, jehož binární zápis je otočeným zápisem binárního zápisu vstupního čísla.

Získání otočeného binárního zápisu není složité – opět budeme postupně zkoumat koeficienty b_0 při postupném bitovém posouvání vstupního čísla o jednu pozici vpravo. Pokud bude daný koeficient nenulový, přidáme do výsledku na určenou pozici jedničkový bit. Jaká bude tato pozice? Uvědomíme si, že nejnižší bit v zadaném čísle (tj. první koeficient b_0) musí být nejvyšším bitem v čísle s otočeným zápisem. Bity tedy budeme přidávat sice vždy na konec otočeného čísla (přičtením jedničky, tj. 2^0), ale otočené číslo v rámci výpočtu budeme posouvat vždy o jeden bit vlevo. Na konci výpočtu tak skutečně jako poslední bit k otočenému číslu přidáme nejvyšší bit původního čísla.

Jak již bylo nastíněno výše, nejelegantnější způsob pro přístup k bitům zadaného čísla je použít tzv. bitovou masku. Bitová maska je číslo, jehož binární reprezentace má jedničky na těch pozicích, ze kterých chceme získat bitovou informaci. Provedeme-li totiž pro zadané číslo x a masku m operaci $x\&m$ (tj. mezi čísly x a m provedeme operaci AND), ve výsledku budou jedničkové právě ty bity, které byly jedničkové v čísle x, ale pouze na pozicích určených binárním zápisem čísla m. Speciálně, pokud chceme z čísla x získat i-tý bit, stačí jako bitovou masku zvolit číslo 2^i a provést operaci $x\&2^i$. Poté stačí ověřit, zda je výsledek této operace nenulový – pokud ano, byl na pozici i v čísle x jedničkový bit (tj. $b_i = 1$); pokud ne, byl na pozici i nulový bit (tj. $b_i = 0$). Ještě konkrétnější aplikace je zjišťování koeficientu b_0. Poté provádíme operaci $x\&2^0 = x\&1$, což využijeme v návrhu tohoto podprogramu.

V rámci implementace budeme otáčet binární zápis čísla v kyblíčku K_0 do kyblíčku K_2. V kyblíčku K_3 budeme v rámci výpočtu provádět operaci AND a kyblíčky K_4, K_5, K_6 opět použijeme pro uložení konstant pro průběh výpočtu. Konstanty nainicializujeme pouze jednou, neboť v podprogramu použijeme zachovávací varianty podprogramů. Jedinou věc, kterou budeme muset provádět v každé iteraci algoritmu je přesun čísla v kyblíčku K_0 do kyblíčku K_3, jelikož vždy dojde přemazání výsledkem operace AND. Nakonec porovnáme, zda se hodnoty v kyblíčcích K_1 (ve kterém si zachováme původní hodnotu čísla na vstupu) a K_2 rovnají. To ověříme postupným odečítáním jedničky od obou čísel.

`BinPalindrom()`, 1 parametr:



Komentář: Podprogram bere jako parametr kyblíček K_0, charakterizující číslo. Jako výsledek bude v kyblíčku K_0 jedna kulička právě tehdy, je-li binární zápis čísla K_0 palindromický. V opačném případě nebude v kyblíčku K_0 na konci vykonávání podprogramu žádná kulička.