Aritmetično logična enota s carry−lookahead seštevalnikom

V VHDL programirajte arhitekturo n-bitne aritmetične logične enote (ALU) s carry−lookahead seštevalnikom (CLA) po podani entiteti:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity alu_cla is
      generic( n: natural := 8 );
      port( M           :     in    std_logic;  --način delovanja ('0' => aritmetični, '1' => logični)
            F           :     in    std_logic_vector(2 downto 0); -- funkcijski vhod za operacije
            X, Y        :     in    std_logic_vector(n-1 downto 0);
            S           :     out   std_logic_vector(n-1 downto 0);
            Negative,
            Cout,
            Overflow,
            Zero,
            Gout,
            Pout        :     out   std_logic);
end alu_cla;
```

Aritmetično logična enota opravlja funkcije po spodnji tabeli:

M (način)	F (operacija)	Izhod
0	0 0 0	S = X plus Y
0	0 0 1	S = X minus Y
0	0 1 0	S = X plus 1
0	0 1 1	S = X minus 1
0	1 0 0	S = X plus X
0	1 0 1	S = minus 1 (dvojiški komplement)
0	1 1 0	NI V UPORABI
0	1 1 1	NI V UPORABI
1	0 0 0	S = X and Y
1	0 0 1	S = X nand Y
1	0 1 0	S = X or Y
1	0 1 1	S = X nor Y
1	1 0 0	S = X xor Y
1	1 0 1	S = X xnor Y
1	1 1 0	S = X
1	1 1 1	S = Y

Entiteta z imenom **`alu_cla`** ima definirane naslednje vhode/izhode:

- **M** : vhod tipa `std_logic`, ki določa način delovanja ALU:
  - '0' **aritmetične** operacije po zgornji tabeli
  - '1' **logične** operacije po zgornji tabeli
- **F**: 3-bitni vhod strukture tipa `std_logic_vector` - predstavlja funkcijo, ki jo ALU opravlja po zgornji tabeli,
- **X, Y** : n-bitna vhoda strukture tipa `std_logic_vector` - vhodna operanda ALU,
- **S** : n-bitni izhod strukture tipa `std_logic_vector` - rezultat operacije ALU,
- **Negative**: izhod tipa `std_logic`, ki postane '1', ko je rezultat operacije **S** negativen,
- **Cout**: izhod tipa `std_logic`, ki predstavlja izhodni prenos operacije,
- **Overflow**: izhod tipa `std_logic`, ki predstavlja bit preliva (ang. overflow) pri aritmetični operaciji,
- **Zero**: izhod tipa `std_logic`, ki postane '1', ko je rezultat operacije **S** enak nič.
- **Gout**: izhod tipa `std_logic`, predstavlja izhodno vrednost funkcije tvorjenja (ang. generate) n-bitnega CLA v ALU.
- **Pout**: izhod tipa `std_logic`, predstavlja izhodno vrednost funkcije tvorjenja (ang. propagate) n-bitnega CLA v ALU.

Naloge:

1. V arhivu predloge naloge se nahaja datoteka **`cla_gp.vhd`**. V to datoteko kopirajte arhitekturo enote (**`cla_gp`**), ki je podana spodaj. Ime entitete je obvezno: **`cla_gp`**.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY cla_gp IS PORT (
      Cin, x, y : IN STD_LOGIC;
      S, Cout, g, p : OUT STD_LOGIC);
END cla_gp ;

ARCHITECTURE ideal OF cla_gp IS
SIGNAL g_sig, p_sig : STD_LOGIC;
BEGIN
g_sig <= x and y; -- funkcija tvorjenja (generate)
p_sig <= x xor y; -- funkcija sirjenja (propagate)
g <= g_sig;
p <= p_sig;
S <= x xor y xor cin;   -- vsota
Cout <= g_sig or ( p_sig and Cin ); -- izhodni prenos stopnje
END ideal;
```

2. Izdelajte datoteko testnih vrednosti (**`cla_gp_tb.vhd`**) in s simulacijo preverite pravilnost seštevanja za vhoda X in Y in vhodni prenos C_{in}.

3. V arhivu predloge naloge se nahaja datoteka **`cla_add_n_bit.vhd`**. V to datoteko kopirajte arhitekturo enote n-bitnega CLA iz nadaljevanja naloge 26 v Zbirki rešenih nalog pri predmetu NDV na domači strani predmeta. Ime entitete mora biti: **`cla_add_n_bit`**.

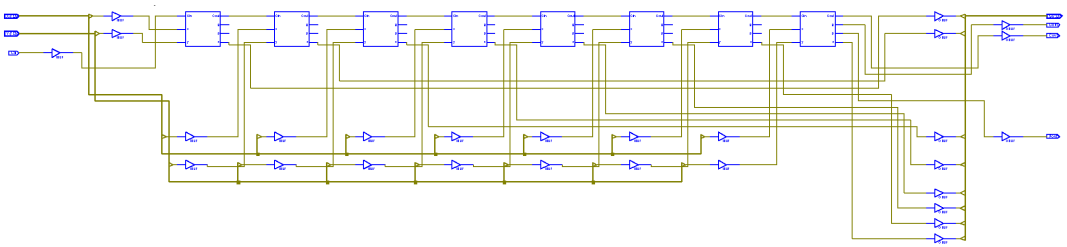
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY cla_add_n_bit IS
generic(n: natural := 8);
PORT (      Cin   :     in    std_logic ;
            X, Y  :     in    std_logic_vector(n-1 downto 0);
            S     :     out   std_logic_vector(n-1 downto 0);
            Gout,
            Pout,
            Cout  :     out   std_logic);
END cla_add_n_bit;
```

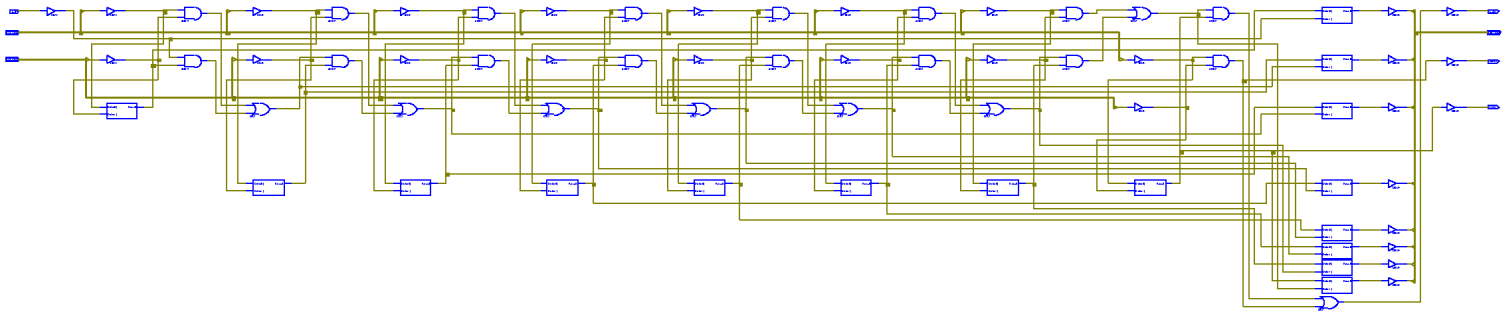
Zanko `FOR ... GENERATE` predelajte tako, da teče indeks od 0 (ne 1 kot je v Zbirki) in realizira funkcije `Gout, Pout, Cout` za splošni n-bitni CLA. Za posplošitev na n-bitno strukturo v deklaraciji entitete uvedite parameter `generic(n: natural :=8)` kot je podano v entiteti n-bitnega CLA.

4. Izdelajte datoteko testnih vrednosti (**`cla_add_n_bit_tb.vhd`**) in s simulacijo preverite pravilnost delovanja za 8-bitni nepredznačeni števili X in Y.

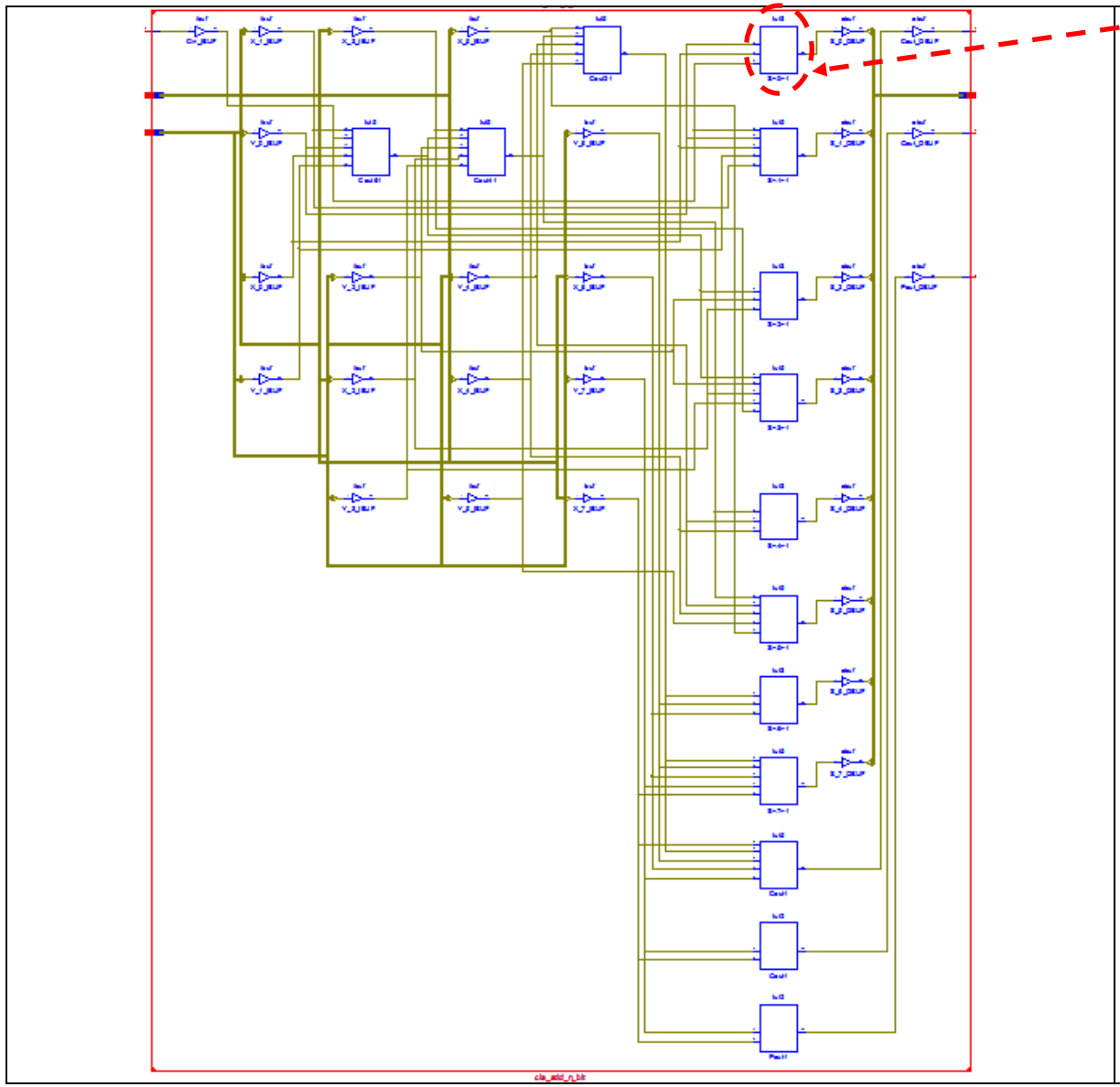
5. Oglejte si **dejansko realizacijo** izdelanega CLA seštevalnika - spodnje velja samo za razvojno okolje ISE10.1 - v ISE 14.7 boste videli samo vzporedno realizacijo : V panelu Sources zamenjajte simulacijo (Behavioral simulation) z implementacijo vezja (Implementation) in v panelu Processes razprite "Implement Design→Synthesize XST→View RTL schematic→Rerun All (desni klik)"). Opisano povezovanje komponente `cla_gp` s povezovalnim stavkom rezultira v zaporedni izvedbi seštevalnika na spodnji sliki:



Dobljena realizacija očitno ni CLA izvedba seštevalnika, saj je orodje za sintezo ("optimize XST") vezje tvorilo z zaporedno vezavo enobitnih seštevalnikov. V izogib izločitvi vmesnih signalov `G, P in C` in posledični zaporedni realizaciji moramo enačbe teh signalov realizirati znotraj `FOR ... GENERATE` zanke **brez** povezovalnega (`PORT MAP`) stavka za mesta od (`0...n-1`), po enačbah iz prve naloge te vaje. Za to moramo izpisati enačbe funkcij `G, P in C` znotraj `FOR ... GENERATE` zanke za vsako mesto po tekočem indeksu (`i`). Pravilno povezana struktura CLA realizacije kaže **vzporedno** povezovanje komponent, kot je prikazano na spodnji sliki:



Če ste za sintezo uporabljali vezje FPGA (npr. Artix 7, xc7a100t, Speed grade: -3, ohišje: csg324) dobite spodnjo realizacijo. Vezja FPGA vsebujejo vpogledne tabele, zato XST nadomesti zgornja vrata z njimi (LUT2, LUT3, LUT5).



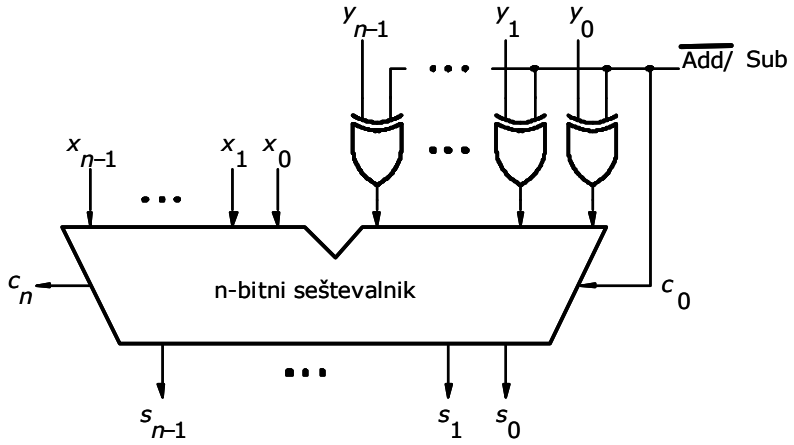
LUT3, ki realizira XOR3 vrata po funkciji
$f^3 = V(1,2,4,7)$

6. Ponovno preverite pravilnost delovanja za 8-bitni nepredznačeni števili X in Y s prej izdelano datoteko testnih vrednosti (**`cla_add_n_bit_tb.vhd`**).

7. V arhivu predloge naloge se nahaja datoteka `alu_cla.vhd`. V njej deklarirajte komponento (`COMPONENT`) prej izdelanega n-bitnega CLA seštevalnika (**`cla_add_n_bit`**).

8. Povežite komponento (**`cla_add_n_bit`**) z entiteto aritmetične enote `alu_cla` z uporabo povezovalnega (`PORT MAP`) stavka.
   Pri povezovanju CLA s `PORT MAP` stavkom uporabite posplošeno povezovanje z nadrejeno komponento **n=>n**. To storite tako:
   `U1: cla_add_n_bit generic map (n => n) port map (...);`

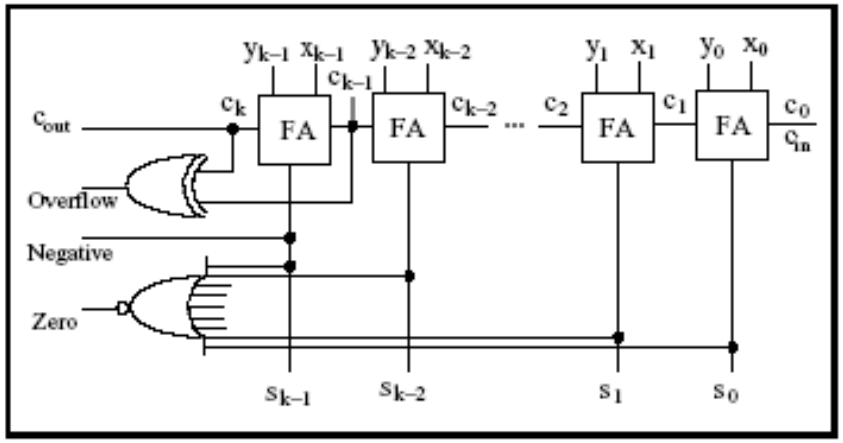
9. Na Y vhod komponente n-bitnega CLA priključite vezje, ki izračuna kontrolirani eniški komplement vhodnega operanda Y aritmetične enote. To vezje sestavlja n XOR vrat kot je prikazano na sliki 1.
   V VHDL namesto polja XOR vrat zapišite isto strukturo z uporabo (`WHEN...ELSE`) stavka.



Slika 1: Realizacija odštevalnika s pomočjo n−bitnega seštevalnika.

10. Vhod **nAddSub** priključite na vhodni prenos n-bitnega CLA (`Cin`) kot je prikazano na sliki 1.

11. Realizirajte bite stanja rezultata: Negative (N), Cout (C), Overflow (V), Zero (Z) z realizacijo, prikazano na spodnji sliki.



Slika 2: Realizacija N, C, V, Z bitov seštevalnika.

Pri realizaciji bita preliva - overflow (V) uporabite realizacijo enačbe:

$$OF_{2'K} = \overline{a_{n-1}} \cdot \overline{b_{n-1}} \cdot s_{k-1} + a_{k-1} \cdot b_{k-1} \cdot \overline{s_{k-1}} = c_k \oplus c_{k-1}$$

12. Zgornja enačba zajema primera postavljanja V='1' pri seštevanju. Bit preliva realizirajte tudi za operacijo odštevanja, kot smo razložili na predavanjih. Pri odštevanju obstajata *drugačna* primera, v katerih se postavi V='1'. Dobljeno realizacijo bita preliva opišite kot VHDL logični izraz (`and, or, not, xor, xnor`). Pri realizaciji bita Z uporabite (`WHEN...ELSE`) stavek, pri čemer kodirajte primerjavo n-bitnega vektorja **S** z n bitnim vektorjem samih ničel. Vektor samih ničel realizirate kot n bitno VHDL konstanto (`CONSTANT`) tipa `std_logic_vector`, ki jo postavite na vrednost nič z operacijo `(others =>'0')`. Ta operator postavlja vse neničelne vrednosti v vektorju na '0'.

    Bita N in Z se morata postaviti tudi v primeru operacije S=X in operacije S=Y, glede na stanje vhodov X in Y. V primeru logičnih operacij (`and, nand, or, nor, xor, xnor`) so biti stanja NCVZ brezpredmetni.

13. Izdelajte datoteko testnih vrednosti (**alu_tb.vhd**) in s simulacijo preverite pravilnost delovanja seštevanja in odštevanja za 8-bitni nepredznačeni števili X in Y.

14. Tvorite vmesni vektor **alu_operation**. z uporabo operatorja sestavljanja (&). Signal **alu_operation** naj bo sestavljen iz signala **M** na MSB mestu in vektorja **F**. Tako dobite 4-bitno operacijo iz tabele ALU na vrhu.

15. Izdelajte dva (WITH...SELECT) stavka:
    - Prvi stavek določa kakšen bo izhod ALU (S) za določene primere vektorja **alu_operation**.
    Logične funkcije ALU iz zgornje tabele zapišete z enostavnimi VHDL operatorji, ki jih lahko uporabimo tudi nad vhodnima vektorjema X in Y. V primeru, da vrednost operacije ni definirana (others) naj bo izhod **S** enak 0.

    - Drugi stavek določa kakšen bo vmesni vhod ALU (Y_sig) za določene primere vektorja **alu_operation**.
    V primeru, da vrednost operacije ni definirana (others) naj bo izhod **Y_sig** enak **Y**.

16. Ena operacija ALU zahteva realizacijo prištevanja n-bitne konstante 1. Splošno, n bitno nepredznačeno konstanto 1 deklarirate tako:
    ```
    constant one : std_logic_vector(n-1 downto 0) := (0=>'1', others=>'0');
    ```
    Tako ustvarite splošno n-bitno konstanto z imenom "one", ki ima LSB mesto vektorja '1', vsa ostala mesta '0'.

17. Vhod nAddSub iz deklarirajte kot signal. Kode aritmetičnih operacij so izbrane tako, da je vrednost signala nAddSub enaka LSB mestu vektorja **alu_operation**.

18. Izdelajte datoteko testnih vrednosti (**alu_cla_tb.vhd**) in s simulacijo preverite pravilnost delovanja vseh operacij iz tabele ALU za 8-bitni predznačeni števili X in Y.

Držite se poimenovanja v navodilih. Upoštevajte točno navedbo signalov v podanih entitetah, sicer naloge ne morem popraviti.

Upoštevajte opisano delovanje, ki ustreza opisanim logičnim vrednostim signala (glej opise signalov v entiteti in navodilih).

Pri poimenovanju signalov se držite pravila, da črka "n" pred imenom signala pomeni negativno logiko poimenovanega signala (primer: nCLR je signal, ki je aktiven '0').

Za n-bitne strukture teče indeks elementov tipa std_logic_vector od 0 (LSB mesto) do n-1 (MSB mesto).

Če naloga zahteva uporabo že izdelanih datotek, zaradi skladnosti uporabljajte podane predloge in ne lastnih.

Rezultati simulacij:

Tabela 1: Simulacija delovanja CLA.

Cin	X	Y	S	Gout	Pout	Cout	Komentar k primeru
0	0	0	0	0	0	0	Seštevanje v obsegu 8 bitnih števil
0	5	4	9	0	0	0	Seštevanje v obsegu 8 bitnih števil
0	127	1	-128	0	0	0	Seštevanje s prelivom v pozitivno smer
0	127	-128	-1	0	1	0	Seštevanje v obsegu 8 bitnih števil
0	1	-128	-127	0	1	0	Seštevanje v obsegu 8 bitnih števil
0	-128	-128	0	1	0	1	Seštevanje s prelivom v negativno smer
0	127	127	-2	0	0	0	Seštevanje s prelivom v pozitivno smer
0	-128	126	-2	0	1	0	Seštevanje v obsegu 8 bitnih števil

Tabela 2: Simulacija delovanja ALU.

M	F	Operacija	X	Y	S	N	C	V	Z	Gout	Pout	Opomba
0	0	S = X plus Y	0	0	0	0	0	0	1	0	0	test Z=1
1	1	S = X nand Y	0	0	-1	0	0	0	0	0	0	0 ↑ 0 = 1; N, Z, V, C, G, P [1]
1	0	S = X and Y	0	0	0	0	0	0	1	0	0	0 • 0 = 0; N, Z, V, C, G, P [1]
1	2	S = X or Y	0	0	0	0	0	0	1	0	0	0 + 0 = 0; N, Z, V, C, G, P [1]
1	3	S = X nor Y	-86	85	0	0	1	1	1	0	1	0 ↓ 0 = 0; N, Z, V, C, G, P [1]
1	4	S = X xor Y	-86	85	-1	1	0	0	0	0	1	0 ⊕ 1 = 1; N, Z, V, C, G, P [1]
1	5	S = X xnor Y	-86	85	0	0	1	1	1	0	1	!(0 ⊕ 1) = 0; N, Z, V, C, G, P [1]
1	6	S = X	-86	85	-86	1	0	0	0	0	1	N, Z, V, C so veljavni
1	7	S = Y	-86	85	85	0	1	1	0	0	1	N, Z, V, C so veljavni
0	0	S = X plus Y	-81	87	6	0	1	0	0	0	1	C=1
0	0	S = X plus Y	-86	81	-5	1	0	0	0	0	1	N=1
0	1	S = X minus Y	-65	-48	-17	1	0	0	0	1	0	N=1
0	1	S = X minus Y	-127	-48	-79	1	0	0	0	1	0	N=1
0	2	S = X plus 1	-127	-48	-126	1	0	0	0	1	0	N=1
0	3	S = X minus 1	-127	-48	-128	1	1	0	0	1	0	N=1, C=1
0	4	S = X plus X	-127	-48	2	0	1	1	0	1	0	C=1, V=1
0	5	S = minus 1	-127	-48	-1	1	1	0	0	1	0	N=1
0	0	S = X plus Y	14	127	-115	1	0	1	0	0	0	V=1 test seštevanja
0	0	S = X plus Y	-14	-127	115	0	1	1	0	1	0	V=1 test seštevanja
0	1	S = X minus Y	14	-127	-115	1	0	1	0	0	0	V=1 test odštevanja
0	1	S = X minus Y	-14	127	115	0	1	1	0	1	0	V=1 test odštevanja
0	2	S = X plus 1	127	127	-128	1	0	1	0	0	0	V=1 test povečevanja
0	3	S = X minus 1	-128	127	127	0	1	1	0	1	0	V=1 test zmanjševanja
0	4	S = X plus X	127	127	-2	1	0	1	0	0	0	V=1 test podvojevanja
0	4	S = X plus X	-128	127	0	0	1	1	1	1	0	V=1 test podvojevanja
0	6	Ni v uporabi										N, Z, V, C, G, P [1]
0	7	Ni v uporabi										N, Z, V, C, G, P [1]

[1] niso bistveni.