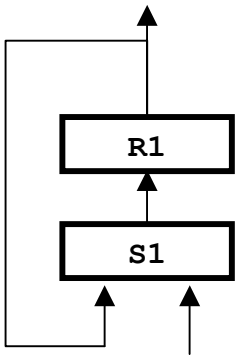
Przygotowanie: Przemysław Sołtan
e-mail: kerk@moskit.ie.tu.koszalin.pl

Przykłady

Konwersja kodów KD i UD

```
//-----------------------------------------------------------------------
// CONV1.GLX - przykład realizacji konwersji
// kodu KD na UD i na odwrót
//-----------------------------------------------------------------------
```




```
//-----------------------------------------------------------------------
dlr R1 : rg(16)           // deklaracja rejestru R1
dlr S1 : sm(16)           // deklaracja sumatora S1
link S1(in1):R1(out)      //
link R1(in):S1(out)       //
accept R1:8001h           // zapisz R1:=8001h
//-----------------------------------------------------------------------
//konwersja KD ----> UD

start   { and  R1,8000h; jz S1,label1 }  //skok, gdy liczba dodatnia
        { and  R1,7FFFh; move R1 }       //zeruj najstarszy bit
        { sub  R1,1,1; move R1 }         //odejmuj R1:=R1-1;
        { nxor R1,0 ; move R1}           //negacja bitów rejestru R1
//-----------------------------------------------------------------------
//konwersja UD ----> KD

label1  { and  R1,8000h; jz S1,end }     //skok, gdy liczba dodatnia
        { and  R1,7FFFh; move R1 }       //zeruj najstarszy bit
        { nxor R1,0; move R1 }           //negacja bitów rejestru R1
        { add  R1,1,0; move R1 }         //dodaj R1:=R1+1;
//-----------------------------------------------------------------------
end     { jmp start }
//-----------------------------------------------------------------------
```

Konwersja kodów dziesiętnych

```
//---------------------------------------------------------------------
// CONV2.GLX - przykład realizacji konwersji kodów dziesiętnych.
//
// GALAXY v1.4 code by Przemysław <KERK> Sołtan
// e-mail: kerk@termit.ie.tu.koszalin.pl
// http://termit.ie.tu.koszalin.pl/~kerk/galaxy.htm
//---------------------------------------------------------------------
dlr R1 : rg(4)          // deklaracja rejestru R1
dlr S1 : sm(4)          // deklaracja sumatora S1
link S1(in1):R1(out)    //
link R1(in):S1(out)     //
accept R1:4h            // dopuszczalny zakres liczby ==> (0-9)
//---------------------------------------------------------------------
// UWAGA !!! Procedury nie uwzględniają reakcji na niedozwolone
// wartości danego kodu.
//---------------------------------------------------------------------
//konwersja z kodu 'BCD' na kod '+3'
start   { add  R1,3,0; move R1 }       // dodaj 3 do R1
//---------------------------------------------------------------------
//konwersja z kodu '+3' na kod 'BCD'
Label1  { sub R1,3,1; move R1 }        // odejmuj 3 od R1
//---------------------------------------------------------------------
//konwersja z kodu 'BCD' na kod '2421'
Label2  { sub R1,4,0; jc S1,Label3 }   // jesli R1 > 4 to C=1 sumatora
        { add R1,6,0; move R1 }        // zwiększ R1 o 5  (R1:=R1+5)
//---------------------------------------------------------------------
//konwersja z kodu '2421' na kod 'BCD'
Label3  { sub R1,4,0; jc S1,Label4 }   // jesli R1 > 4 to C=1 sumatora
        { add R1,6,0; move R1 }        // zwiększ R1 o 6  (R1:=R1+6)
//---------------------------------------------------------------------
//konwersja z kodu '2421' na kod 'BCD'
Label4  { and R1,1000b; jz S1,end }    // testuj najstarszy bit
        { sub R1,6,1; move R1 }        // zmniejsz R1 o 6  (R1:=R1+6)
//---------------------------------------------------------------------
end     { jmp start }
//---------------------------------------------------------------------
```

Realizacja programowej pętli

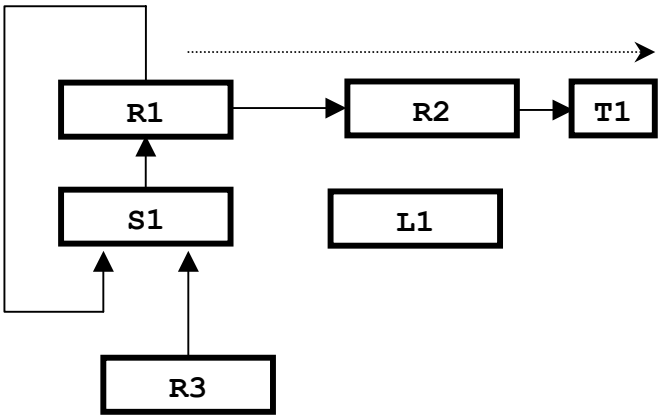
```
//-----------------------------------------------------------------------
// LOOP.GLX - przykład realizacji pętli
//
// GALAXY v1.4 code by Przemysław <KERK> Sołtan
// e-mail: kerk@termit.ie.tu.koszalin.pl
// http://termit.ie.tu.koszalin.pl/~kerk/galaxy.htm
//-----------------------------------------------------------------------
dlr L1     : l(8)              //licznik
accept L1  : 17                //zawartość licznika L1:=17
//-----------------------------------------------------------------------
start      { dec L1 }          //zmniejsz L1:=L1-1
           { jnz L1,start }    //test, czy L1=0
//-----------------------------------------------------------------------
end        { jmp end }
//-----------------------------------------------------------------------
```

Normalizacja

```
//---------------------------------------------------------------------
// NORMAL.GLX - przykład realizowana NORMALIZACJI
//
// GALAXY v1.4 code by Przemysław <KERK> Sołtan
// e-mail: kerk@termit.ie.tu.koszalin.pl
// http://termit.ie.tu.koszalin.pl/~kerk/galaxy.htm
//---------------------------------------------------------------------
dlr R1 : rg(8)       // najstarszy bit określa znak liczby
dlr R2 : l(8)
dlr S1 : sm(8)
link S1(in1):R1(out)
link R1(in):S1(out)
accept R1:1          //  00000001 E+30 ==> 01000000 E+5
accept R2:10         //
//---------------------------------------------------------------------
start   { and  R1,01000000b; jnz S1,end  } //skok, gdy koniec
        { sla  R1; dec R2 } //przesunięcie arytmetyczne (bit znaku bez
                            //zmian)
        { jmp start }
//---------------------------------------------------------------------
end     { jmp end }
//---------------------------------------------------------------------
```

Mnożenie (wariant 1)

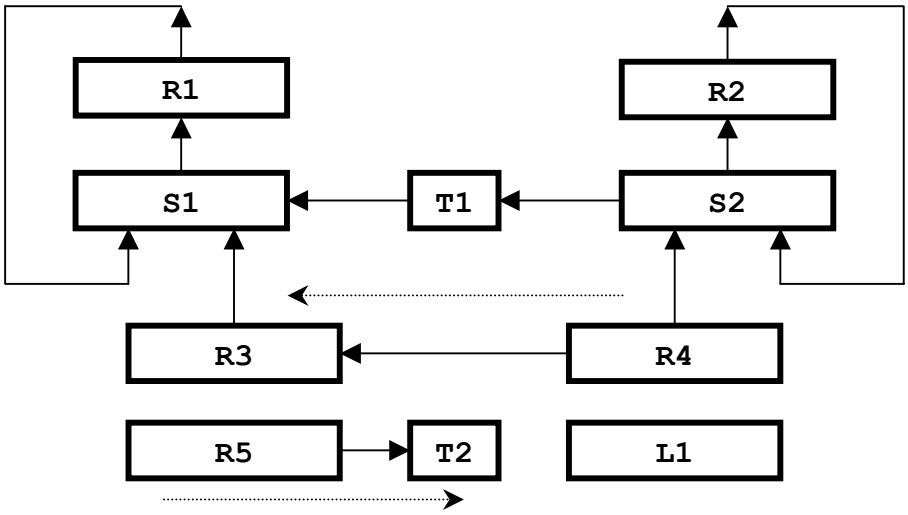
```
//-----------------------------------------------------------------------
// MNOZ1.GLX - przykład mnożenia sekwencyjnego (wariant1)
// Poczynając z mniej znaczących bitów mnożnika, przesuwamy
// formowany rezultat w stronę mniej znaczących bitów.
//-----------------------------------------------------------------------
```




```
//-----------------------------------------------------------------------
dlr R1  : rg(16)
dlr R2  : rg(16)
dlr R3  : rg(16)
dlr S1  : sm(16)
dlr L1  : l(8)
dlr T1  : t
//-----------------------------------------------------------------------
link S1(in1)  : R1(out)          //
link S1(in2)  : R3(out)          //
link R1(in)   : S1(out)          //
link T1(in)   : R2(cr)           //
link R2(cl)   : R1(cr)           //
//-----------------------------------------------------------------------
accept L1  : 17                  //ustawienie licznika
accept T1  : 0                   //zerowanie przerzutnika
accept R1  : 0                   //zerowanie Z.h
//-----------------------------------------------------------------------
accept R3  : 3                  //X
accept R2  : 5                  //Y (Z.l)
//-----------------------------------------------------------------------
start      { srl R1; sr R2; move T1; dec L1 }
           { jz T1,label1 }
           { add R1,R3 ; move R1 }
label1     { jnz L1,start }
//-----------------------------------------------------------------------
end        { jmp end }
//-----------------------------------------------------------------------
```

Mnożenie (wariant 2)

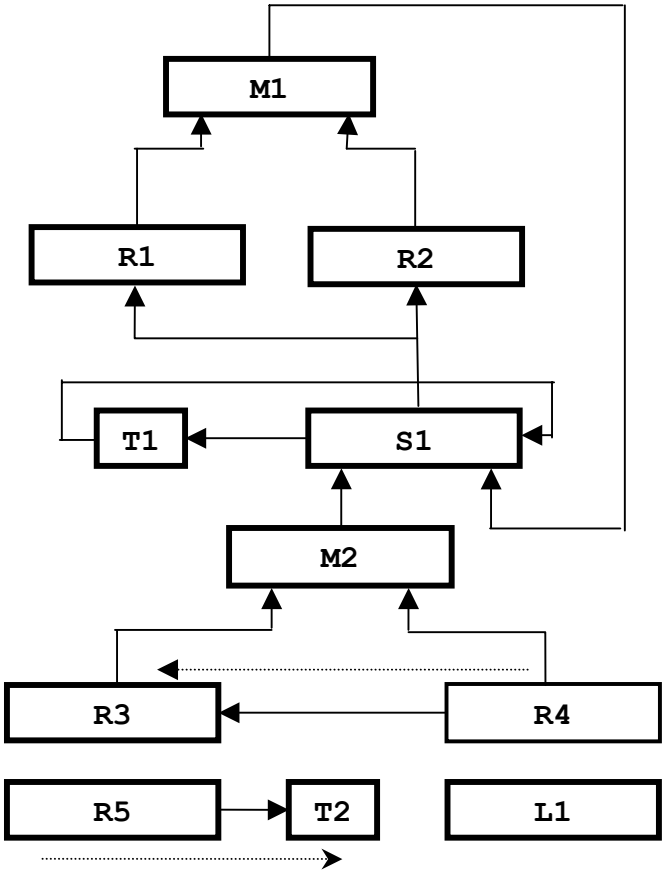
```
//-----------------------------------------------------------------------
// MNOZ2.GLX - przykład mnożenia sekwencyjnego (wariant2)
// Poczynając z mniej znaczących bitów mnożnika, przesuwamy
// mnożną w stronę starszych znaczących bitów.
//-----------------------------------------------------------------------
```




```
//-----------------------------------------------------------------------
dlr R1 : rg(16)
dlr R2 : rg(16)
dlr R3 : rg(16)
dlr R4 : rg(16)
dlr R5 : rg(16)
dlr S1 : sm(16)
dlr S2 : sm(16)
dlr L1  : l(8)
dlr T1  : t
dlr T2  : t
//-----------------------------------------------------------------------
link T1(in)  : R5(cr)
link S2(in1) : R2(out)
link S2(in2) : R4(out)
link R2(in)  : S2(out)
link T2(in)  : S2(c)
link S1(ci)  : T2(out)
link S1(in1) : R1(out)
link S1(in2) : R3(out)
link R1(in)  : S1(out)
link R3(cr)  : R4(cl)
//-----------------------------------------------------------------------
accept R1 : 0             //Z.h
accept R2 : 0             //Z.l
accept R4 : 3             //X
accept R5 : 5             //Y
accept L1 : 17            //licznik
//-----------------------------------------------------------------------
start      { srl R5 ; move T1 }
           { jz T1, label1 }
           { add R2, R4 ; move R2 ; move T2 }
           { add R3, R1 ; move R1 }
label1     { sll R4 ; sl R3; dec L1 }
           { jnz L1, start }
//-----------------------------------------------------------------------
end        { jmp end }
```

Mnożenie (zmodyfikowany wariant 2)

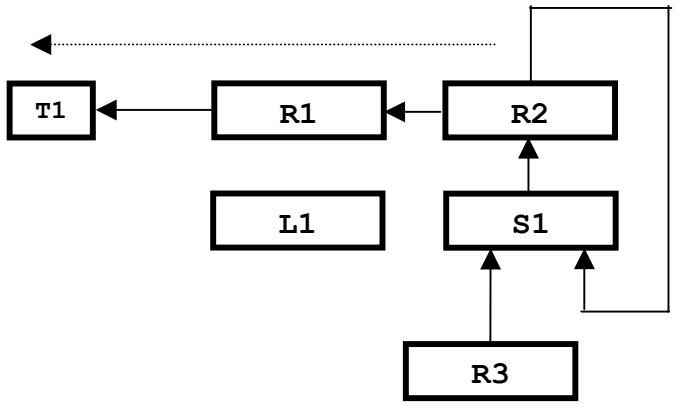
```
//-----------------------------------------------------------------------
// MNOZ2a.GLX - przykład mnożenia sekwencyjnego (wariant2a)
// Poczynając z mniej znaczących bitów mnożnika, przesuwamy
// mnożną w stronę starszych znaczcych bitów.
// W programie dokonano modyfikacji polegajcej na wykorzystaniu
// tylko jednego sumatora i przełączających go multiplekserów.
//-----------------------------------------------------------------------
```




```
//---------------
dlr R1 : rg(16)
dlr R2 : rg(16)
dlr R3 : rg(16)
dlr R4 : rg(16)
dlr R5 : rg(16)
dlr S1 : sm(16)
dlr M1 : mx(16)
dlr M2 : mx(16)
dlr L1  : l(8)
dlr T1  : t
dlr T2  : t
//---------------
link T1(in)  : R5(cr)
link S1(in1) : M1(out)
link S1(in2) : M2(out)
link R1(in)  : S1(out)
link R2(in)  : S1(out)
link T2(in)  : S1(c)
link S1(ci)  : T2(out)
link M1(in1) : R1(out)
link M1(in2) : R2(out)
link M2(in1) : R3(out)
link M2(in2) : R4(out)
link R3(cr)  : R4(cl)
//-----------------------------------------------------------------------
accept R1 : 0             //Z.h
accept R2 : 0             //Z.l
accept R3 : 0             //
accept R4 : 3             //X
accept R5 : 5             //Y
accept L1 : 17            //licznik
//-----------------------------------------------------------------------
start      { srl R5 ; move T1 }
           { jz T1, label1 }
           { add R2, R4 ; move R2 ; move T2 }
           { add R3, R1 ; move R1 }
label1     { sll R4 ; sl R3; dec L1 }
           { jnz L1, start }
//-----------------------------------------------------------------------
end        { jmp end }
//-----------------------------------------------------------------------
```

Mnożenie (wariant 3)

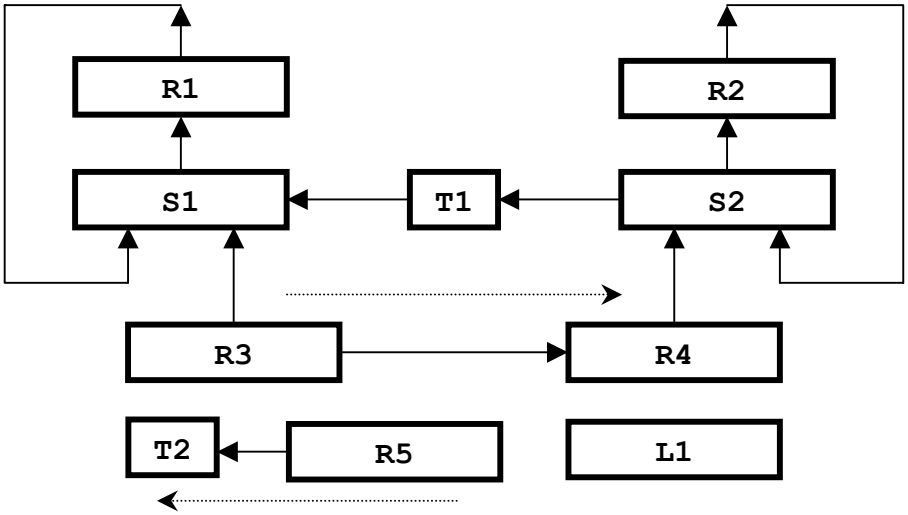
```
//-----------------------------------------------------------------------
// MNOZ3.GLX - przykład mnożenia sekwencyjnego (wariant3)
// Poczynając ze starszych znaczących bitów mnożnika przesuwamy
// formowany rezultat w stronę starszych znaczących bitów.
//-----------------------------------------------------------------------
```




```
//-----------------------------------------------------------------------
dlr R1 : rg(16)
dlr R2 : rg(16)
dlr R3 : rg(16)
dlr S1 : sm(16)
dlr L1 : l(8)
dlr T1 : t
//-----------------------------------------------------------------------
link S1(in1) : R2(out)
link S1(in2) : R3(out)
link R2(in)  : S1(out)
link T1(in)  : R1(cl)
link R1(cr)  : R2(cl)
//-----------------------------------------------------------------------
accept L1  : 16
accept R3  : 5
accept R1  : 3
//-----------------------------------------------------------------------
start      { sll R2 ; sl R1 ; move t1; dec l1 }
           { jz T1,label1 }
           { add R2,R3 ; move R2 }
label1     { jnz l1,start }
//-----------------------------------------------------------------------
end        { jmp end }
//-----------------------------------------------------------------------
```

Mnożenie (wariant 4)

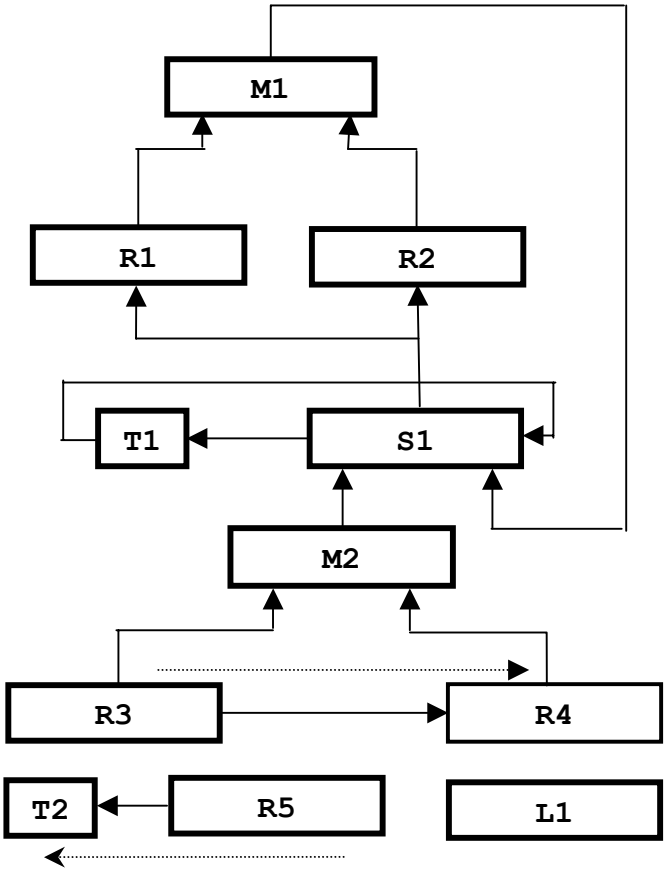
```
//---------------------------------------------------------------------
// MNOZ4.GLX - przykład mnożenia sekwencyjnego (wariant4)
// Poczynając ze starszych znaczących bitów mnożnika przesuwamy
// mnożną w stronę mniej znaczących bitów.
//---------------------------------------------------------------------
```




```
//--------------
dlr R1 : rg(16)
dlr R2 : rg(16)
dlr R3 : rg(16)
dlr R4 : rg(16)
dlr R5 : rg(16)
dlr S1 : sm(16)
dlr S2 : sm(16)
dlr L1 : l(8)
dlr T1 : t
dlr T2 : t
//-------------
link T2(in)  : R5(cl)
link S2(in1) : R2(out)
link S2(in2) : R4(out)
link R2(in)  : S2(out)
link T1(in)  : S2(c)
link S1(ci)  : T1(out)
link S1(in1) : R1(out)
link S1(in2) : R3(out)
link R1(in)  : S1(out)
link R4(cl)  : R3(cr)
//---------------------------------------------------------------------
accept R1    : 0            //Z.h
accept R2    : 0            //Z.l
accept R3    : 3            //X
accept R4    : 0            //
accept R5    : 5            //Y
accept l1    : 17           //licznik
//---------------------------------------------------------------------
start      { srl R3 ; sr R4 ; dec L1 }
           { sll R5 ; move T2 }
           { jz T2, label1 }
           { add R2, R4 ; move R2 ; move T1 }
           { add R3, R1 ; move R1 }
label1     { jnz L1, start }
//---------------------------------------------------------------------
end        { jmp end }
//---------------------------------------------------------------------
```

Mnożenie (zmodyfikowany wariant 4)

```
//-----------------------------------------------------------------------
// MNOZ4a.GLX - przykład mnożenia sekwencyjnego (wariant4a)
// Poczynając ze starszych znaczących bitów mnożnika przesuwamy
// mnożną w stronę mniej znaczących bitów.
// W programie dokonano modyfikacji polegajcej na wykorzystaniu
// tylko jednego sumatora i przełączających go multiplekserów.
//-----------------------------------------------------------------------
```




```
//--------------
dlr R1 : rg(16)
dlr R2 : rg(16)
dlr R3 : rg(16)
dlr R4 : rg(16)
dlr R5 : rg(16)
dlr S1 : sm(16)
dlr M1 : mx(16)
dlr M2 : mx(16)
dlr L1 : l(8)
dlr T1 : t
dlr T2 : t
//--------------
link T2(in)  : R5(cl)
link M1(in1) : R1(out)
link M1(in2) : R2(out)
link M2(in1) : R3(out)
link M2(in2) : R4(out)
link R1(in)  : S1(out)
link R2(in)  : S1(out)
link T1(in)  : S1(c)
link S1(ci)  : T1(out)
link S1(in1) : M1(out)
link S1(in2) : M2(out)
link R4(cl)  : R3(cr)
//-----------------------------------------------------------------------
accept R1    : 0            //Z.h
accept R2    : 0            //Z.l
accept R3    : 3            //X
accept R4    : 0            //
accept R5    : 5            //Y
accept l1    : 17           //licznik
//-----------------------------------------------------------------------
start      { srl R3 ; sr R4 ; dec L1 }
           { sll R5 ; move T2 }
           { jz T2, label1 }
           { add R2, R4 ; move R2 ; move T1 }
           { add R3, R1 ; move R1 }
label1     { jnz L1, start }
//-----------------------------------------------------------------------
end        { jmp end }
//-----------------------------------------------------------------------
```

Mnożenie w kodzie UD (z konwersją kodów)

```
//-------------------------------------------------------------------
// MNOZ_u2.GLX - przykład mnożenia sekwencyjnego (wariant1)
// Poczynając z mniej znaczących bitów mnożnika, przesuwamy
// formowany rezultat w stronę mniej znaczących bitów.
//
// X(UD)*Y(UD)=Z(KD)
//
// GALAXY v1.4 code by Przemysław <KERK> Sołtan
// e-mail: kerk@termit.ie.tu.koszalin.pl
// http://termit.ie.tu.koszalin.pl/~kerk/galaxy.htm
//-------------------------------------------------------------------
dlr R1  : rg(16)
dlr R2  : rg(16)
dlr R3  : rg(16)
dlr S1  : sm(16)
dlr S2  : sm(16)
dlr L1  : l(8)
dlr L2  : l(1)
dlr T1  : t
//-------------------------------------------------------------------
link S1(in1)  : R2(out)         //
link S1(in2)  : R3(out)         //
link R2(in)   : S1(out)         //
link T1(in)   : R1(cr)          //
link R1(cl)   : R2(cr)          //
link R3(in)   : S1(out)
link R1(in)   : S2(out)
link S2(in1)  : R1(out)
//-------------------------------------------------------------------
accept L1  : 17                 //ustawienie licznika
accept T1  : 0                  //zerowanie przerzutnika
accept R2  : 0                  //zerowanie Z.h
//-------------------------------------------------------------------
accept R1  : 1111111111111110b //Y=-2(UD)
accept R3  : 1111111111111101b //X=-3(UD)
//-------------------------------------------------------------------
// X (R3) UD
lab1       { and  R3,100000000b; jz S1,lab2 }
           { nxor R3,0; move R3 }
           { add  R3,1,0; move R3 }
           { inc L2 }
//-------------------------------------------------------------------
// Y (R1) UD
lab2       { and  R1,100000000b; jz S2,lab3 }
           { nxor R1,0; move R1 }
           { add  R1,1,0; move R1 }
           { inc L2 }
//-------------------------------------------------------------------
lab3       { srl R2; sr R1; move T1; dec L1 }
           { jz T1,lab4 }
           { add R2,R3 ; move R2 }
lab4       { jnz L1,lab3 }
//-------------------------------------------------------------------
// Z (r13,r12) KD  jesli Z ujemne to ustaw najstarszy bit rejestru r13
           { jz l2,lab5 }
           { or R2,100000000b; move r2 }
//-------------------------------------------------------------------
lab5       { jmp lab5 }
//-------------------------------------------------------------------
```

Mnożenie w kodzie UD (z dodatnim mnożnikiem)

```
//---------------------------------------------------------------------
// MNOZ_u2a.GLX - przykład mnożenia sekwencyjnego (wariant1)
// Poczynając z mniej znaczących bitów mnożnika, przesuwamy
// formowany rezultat w stronę mniej znaczących bitów.
//
// GALAXY v1.4 code by Przemysław <KERK> Sołtan
// e-mail: kerk@termit.ie.tu.koszalin.pl
// http://termit.ie.tu.koszalin.pl/~kerk/galaxy.htm
//---------------------------------------------------------------------
dlr A   : rg(4)                 // A [1|011]
dlr X   : rg(4)                 // X [0|011]
dlr P   : rg(4)                 // P [X|XXX]
dlr S1  : sm(4)
dlr L1  : l(4)
dlr T1  : t
dlr T2  : t
//---------------------------------------------------------------------
link S1(in1)  : P(out)          //
link S1(in2)  : A(out)          //
link P(in)    : S1(out)         //
link T1(in)   : X(cr)           //
link X(cl)    : P(cr)           //
link P(cl)    : T2(out)         //
//---------------------------------------------------------------------
accept L1  : 4                  //ustawienie licznika
accept T1  : 0                  //zerowanie przerzutnika
accept T2  : 0                  //
accept P   : 0                  //zerowanie Z.h
//---------------------------------------------------------------------
accept A   : 1011b              // A = -5     A[1011]
accept X   : 0011b              // X =  3     X[0011]
accept P   : 0                  // P =  0     P[0000]
//---------------------------------------------------------------------
start      { sra P; sra X; move T1; move T2,1 }
           { jz T1,label1 }
           { add P,A ; move P }
label1     { dec L1 }
           { jnz L1,start }
//---------------------------------------------------------------------
end        { jmp end }      // P[1110] X[X001] ==> 1110001b (-15)
//---------------------------------------------------------------------
```

Mnożenie w kodzie UD (z ujemnym mnożnikiem i mnożną)

```
//-----------------------------------------------------------------
// MNOZ_U2b.GLX - przykład mnożenia sekwencyjnego liczb UD.
// Poczynając z mniej znaczących bitów mnożnika, przesuwamy
// formowany rezultat w stronę mniej znaczących bitów.
//
// A = -5
// X = -3  mnożnik jest liczbą ujemną
//         (niezbędna jest korekta wyniku)
//
// GALAXY v1.4 code by Przemysław <KERK> Sołtan
// e-mail: kerk@termit.ie.tu.koszalin.pl
// http://termit.ie.tu.koszalin.pl/~kerk/galaxy.htm
//-----------------------------------------------------------------
dlr A   : rg(4)
dlr X   : rg(4)
dlr P   : rg(4)
dlr S1  : sm(4)
dlr L1  : l(4)
dlr T1  : t
dlr T2  : t
//-----------------------------------------------------------------
link S1(in1)  : P(out)         //
link S1(in2)  : A(out)         //
link P(in)    : S1(out)        //
link T1(in)   : X(cr)          //
link X(cl)    : P(cr)          //
link P(cl)    : T2(out)        //
//-----------------------------------------------------------------
accept L1  : 4                  //ustawienie licznika
accept T1  : 0                  //zerowanie przerzutnika
accept T2  : 0
accept P   : 0
//-----------------------------------------------------------------
accept A   : 1011b              // -5
accept X   : 1101b              // -3
//-----------------------------------------------------------------
start      { sr P; sra X; move T1; move T2,1 }        // SHIFT
           { dec L1; jz T1,label1 }                   // DEC(L)
           { add P,A ; move P }                       // ADD A
label1     { jnz L1,start }                           //
           { sub P, A,1; move P }                     // KOREKTA (-A)
//-----------------------------------------------------------------
end        { jmp end }    // P(0001) X(X111) ==> 0001111b (+15)
//-----------------------------------------------------------------
```

Mnożenie w kodzie UD (z ujemnym mnożnikiem)

```
//-----------------------------------------------------------------
// MNOZ_U2c.GLX - przykład mnożenia sekwencyjnego liczb UD.
// Poczynając z mniej znaczących bitów mnożnika, przesuwamy
// formowany rezultat w stronę mniej znaczących bitów.
//
// A =  5
// X = -3 // mnożnik jest liczbą ujemną (niezbędna jest korekta
//           wyniku)
//
// GALAXY v1.4 code by Przemysław <KERK> Sołtan
// e-mail: kerk@termit.ie.tu.koszalin.pl
// http://termit.ie.tu.koszalin.pl/~kerk/galaxy.htm
//-----------------------------------------------------------------
dlr A   : rg(4)
dlr X   : rg(4)
dlr P   : rg(4)
dlr S1  : sm(4)
dlr L1  : l(4)
dlr T1  : t
dlr T2  : t
//-----------------------------------------------------------------
link S1(in1)  : P(out)         //
link S1(in2)  : A(out)         //
link P(in)    : S1(out)        //
link T1(in)   : X(cr)          //
link X(cl)    : P(cr)          //
link P(cl)    : T2(out)        //
//-----------------------------------------------------------------
accept L1  : 4                  //ustawienie licznika
accept T1  : 0                  //zerowanie przerzutnika
accept T2  : 0
accept P   : 0
//-----------------------------------------------------------------
accept A   : 0101b              //  5
accept X   : 1101b              // -3
//-----------------------------------------------------------------
start      { sr P; sra X; move T1 }                  // SHIFT
           { dec L1; jz T1,label1 }                  // DEC(L)
           { add P,A ; move P }                      // ADD A
label1     { jnz L1,start }                          //
           { sub P, A,1; move P }                    // KOREKTA (-A)
//-----------------------------------------------------------------
end        { jmp end }    // P(1110) X(X001) ==> 0001111b (+15)
//-----------------------------------------------------------------
```

Dzielenie restytucyjne (odtwarzające)

```
\-----------------------------------------
\Dzielenie restytucyjne X=QD+R  |R|<|D|
\-----------------------------------------
dlr X  :rg(5)
dlr Q  :rg(5)
dlr D  :rg(5)
dlr S  :sm(5)
dlr L1 :l(4)
dlr T1 :t
\-----------------------------------------
link S(in2):D(out)
link S(in1):X(out)
link X(in) :S(out)
link D(in) :S(out)
link T1(in):S(c)
link Q(cr):T1(outn)
\-----------------------------------------
accept L1:4
accept X:00111b //  X:=7/16
accept D:01101b //  D:=13/16
accept Q:0
\-----------------------------------------
label1  { sll X }
        { sub X,D,1; move X; move T1; jnc S,label3 } //skok, gdy c=0
        { add X,D,0; move X }
label3  { dec L1; sl Q }
        { jnz L1,label1}
\-----------------------------------------
```

Dzielenie nierestytucyjne (nieodtwarzające)

```
\-------------------------------------------
\Dzielenie nierestytucyjne X=QD+R  |R|<|D|
\-------------------------------------------
dlr X  :rg(5)
dlr Q  :rg(5)
dlr D  :rg(5)
dlr S  :sm(5)
dlr L1 :l(4)
dlr T1 :t
\-------------------------------------------
link S(in2):D(out)
link S(in1):X(out)
link X(in) :S(out)
link D(in) :S(out)
link T1(in):S(c)
link Q(cr):T1(outn)
\-------------------------------------------
accept L1:4
accept X:00111b //  X:=7/16
accept D:01101b //  D:=13/16
accept Q:0
\-------------------------------------------
label1  { sll X; jnz T1,label2 }
        { sub X,D,1; move X; move T1; jmp label3 }
label2  { add X,D,0; move X }
label3  { dec L1; sl Q }
        { jnz L1,label1}
        { jz T1,label3 }  // korekta, gdy reszta jest ujemna.
        { add X,D,0; move X }
label4  { jmp label4 }
\-------------------------------------------
```