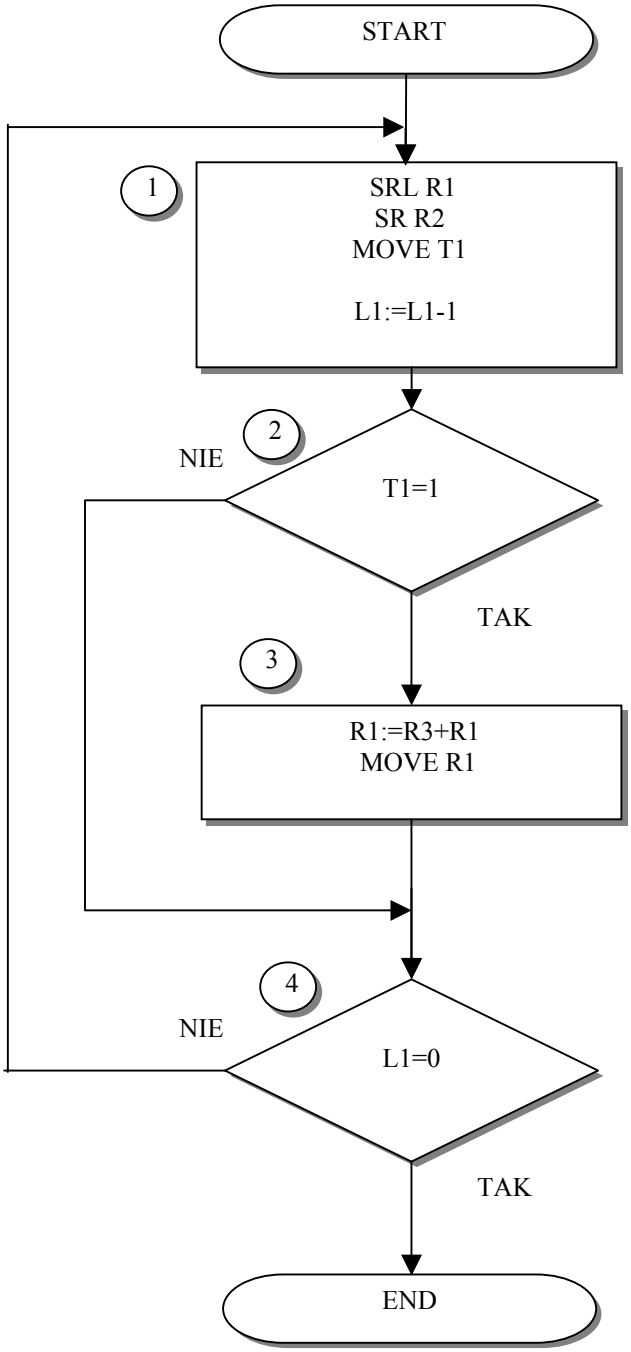
Przygotowanie: Przemysław Sołtan
e-mail: kerk@moskit.ie.tu.koszalin.pl

Algorytm mnożenia sekwencyjnego (wariant 1)

```
//-----------------------------------------------------------------
//Poczynając z mniej znaczących bitów mnożnika, przesuwamy formowany
//rezultat w stronę mniej znaczących bitów.
//R1 - rejestr sum częściowych i bardziej znaczącej części iloczynu
//R2 - rejestr mnożnika X i mniej znaczącej części iloczynu
//R3 - rejestr mnożnej A
//-----------------------------------------------------------------
```



Algorytm mnożenia sekwencyjnego (przykład).

Implementacja algorytmu dokonywana jest w kilku krokach.

Deklaracja bloków funkcyjnych

Po zapoznaniu się z algorytmem określamy rodzaje i liczbę bloków funkcyjnych potrzebnych do implementacji zadanego algorytmu.

Deklarację bloków funkcyjnych dokonujemy dyrektywą DLR.

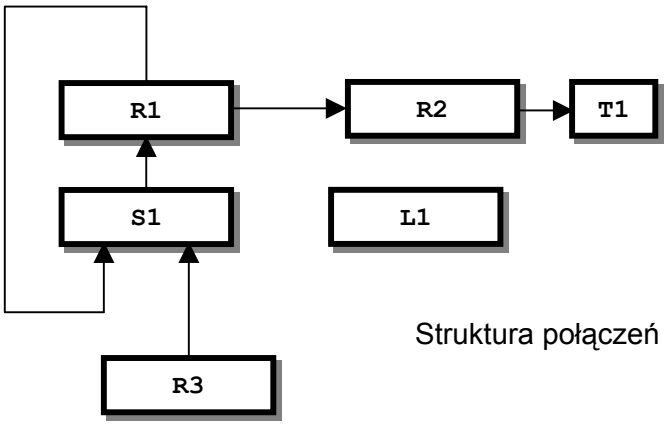
```
//--------------------------------------------------------------------
dlr R1  : rg(16)  //
dlr R2  : rg(16)  //
dlr R3  : rg(16)  //
dlr S1  : sm(16)  //
dlr L1  : l(8)    //
dlr T1  : t       //
//--------------------------------------------------------------------
```

Zadeklarowane bloki funkcyjne algorytmu mnożenia.

Deklaracja połączeń bloków funkcyjnych

Po zadeklarowaniu bloków funkcyjnych należy dokonać ich połączeń zgodnie z wymaganą strukturą. Dokonuje się tego przy pomocy dyrektywy LINK.

```
//--------------------------------------------------------------------
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)          //
//--------------------------------------------------------------------
```



Struktura połączeń algorytmu mnożenia.

Deklaracja wartości początkowych

Aby ustawić wartości początkowe bloków funkcyjnych należy wykorzystać dyrektywę ACCEPT. Przy braku deklaracji wartość przechowywana przez dany blok wynosi zero.

```
//--------------------------------------------------------------------
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)
//--------------------------------------------------------------------
```

Implementacja algorytmu

Po zadeklarowaniu całej struktury algorytmu należy zabrać się za projektowanie głównego programu sterującego współpracą bloków funkcyjnych.

```
//--------------------------------------------------------------------
start      { srl R1; sr R2; move T1; dec L1 }  // faza1
           { jz T1,label1 }                    // faza2
           { add R1,R3 ; move R1 }             // faza3
label1     { jnz L1,start }                    // faza4
end        { jmp end }                         //
//--------------------------------------------------------------------
```

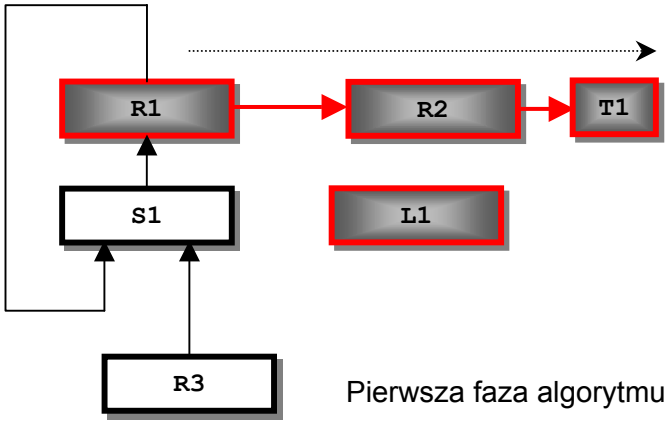
W projektowanym programie wyróżniamy cztery fazy pracy algorytmu.

Przesunięcie rejestrów i dekrementacja licznika

W pierwszej fazie dokonywane jest przesunięcie bitów w prawo rejestrów R1 i R2 i przerzutnika T1 połączonych ze sobą dyrektywą LINK. Przesunięcie odbywa się w ten sposób, że na rejestrze R1 dokonujemy przesunięcia logicznego ***SRL R1*** *(najstarszy bit rejestru przybiera wartość 0, a najmłodszy bit jest wypychany w prawo),* na rejestrze R2 dokonujemy przesunięcia prostego ***SR R2*** *(na najstarszy bit rejestru pobrany jest z rejestru R1, a najmłodszy bit jest wypychany w prawo),* a na przerzutniku dokonujemy *(zapis bitu z rejestru R2)* ***MOVE T1***.

W tej samej fazie dokonujemy także zmniejszenia licznika ***DEC L1*** realizującego programową pętlę.

```
//--------------------------------------------------------------------
start      { srl R1; sr R2; move T1; dec L1 }  // faza1
//--------------------------------------------------------------------
```

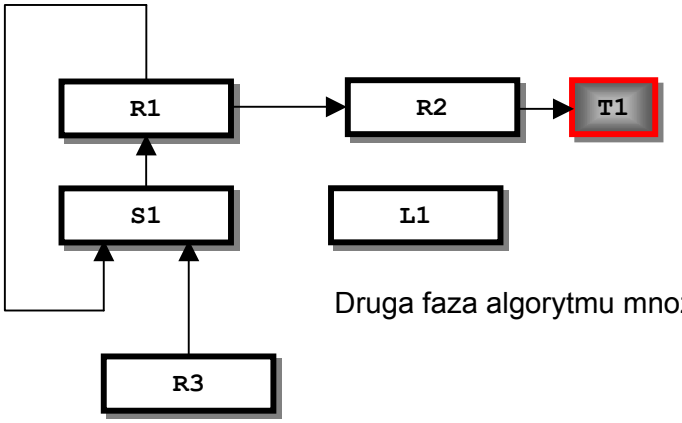


Pierwsza faza algorytmu mnożenia.

Testowanie zawartości przerzutnika T1

Po dokonaniu przesunięć testujemy zawartość przerzutnika T1 mikrooperacją ***JZ T1,label1*** realizującą operację **if(...) then(...)**. W tej fazie dokonuje się sprawdzania poszczególnych bitów rejestru R2 będącego mnożną **X**. W przypadku, gdy zawartość przerzutnika jest zerowa następuje skok do etykiety ***label1***.

```
//-----------------------------------------------------------------
          { jz T1,label1 }                    // faza2
//-----------------------------------------------------------------
```



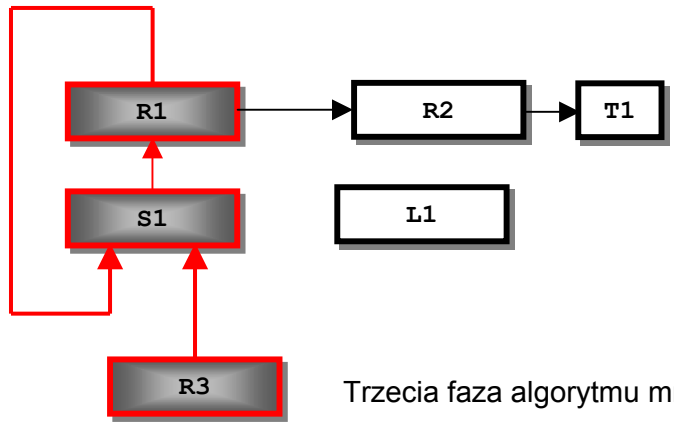
Druga faza algorytmu mnożenia.

Dodawanie zawartości rejestrów

W fazie trzeciej następuje dodawanie zawartości rejestru R1 i R3

ADD R1,R3 z zapisem wyniku sumowania do rejestru R1 **MOVE R1**. W ten sposób dokonuje się sumowanie mnożnika A z sumą częściową przechowywaną w rejestrze R1.

```
//-----------------------------------------------------------------
          { add R1,R3 ; move R1 }             // faza3
//-----------------------------------------------------------------
```

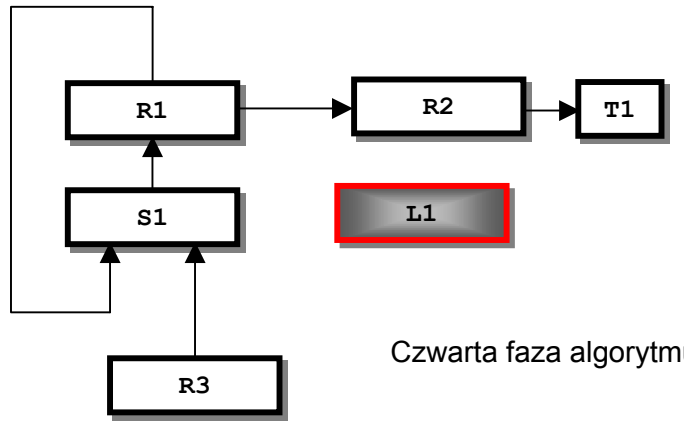


Trzecia faza algorytmu mnożenia.

Testowanie zawartości licznika

W fazie czwartej następuje testowanie stanu znacznika Z licznika L1 **JNZ L1,start**. Znacznik ten jest ustawiony na 1, gdy zawartość licznika jest zerem. Przy inicjalizacji programu licznik zawierał określoną wartość, a w trakcie działania programu, była ona sukcesywnie zmniejszana o jeden. W ten oto sposób dokonuje się realizacji pętli programowej.

```
//-------------------------------------------------------------------
label1     { jnz L1,start }                    // faza4
//-------------------------------------------------------------------
```



Czwarta faza algorytmu mnożenia.

Zakończenie programu

Program można zakończyć zapętlającą się instrukcją skoku. W przypadku jej braku debugger zatrzymuje się na ostatnim wierszu programu.

```
//-------------------------------------------------------------------
end          { jmp end }                    // koniec
//-------------------------------------------------------------------
```

Listing programu

Ostatecznie otrzymujemy, w postaci listingu programu, kompletną realizację algorytmu mnożenia sekwencyjnego zgodnego ze składnią programu GALAXY.

```
//-----------------------------------------------------------------------
// MNOZ1.glx – algorytm mnożenia sekwencyjnego (wariant1)
//-----------------------------------------------------------------------
//Poczynając z mniej znaczących bitów mnożnika, przesuwamy formowany
//rezultat w stronę mniej znaczących bitów.
//R1 – rejestr sum częściowych i bardziej znaczącej części iloczynu
//R2 – rejestr mnożnika X i mniej znaczącej części iloczynu
//R3 – rejestr mnożnej A
//-----------------------------------------------------------------------
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 }  // faza1
           { jz T1,label1 }                    // faza2
           { add R1,R3 ; move R1 }             // faza3
label1     { jnz L1,start }                    // faza4
end        { jmp end }                         // koniec
//-----------------------------------------------------------------------
```