mgr inż. Marek ALEKSANDRUK
mgr inż. Lesław KOŁAKOWSKI
Przemysłowy Instytut Automatyki
i Pomiarów MERA–PIAP
W a r s z a w a
mgr inż. Magdalena PORADA
Akademia Muzyczna im. F. Chopina
W a r s z a w a

EARL – JĘZYK PROGRAMOWANIA SEKWENCYJNEGO

1. WSTĘP

Prezentowany w niniejszym opracowaniu język programowania EARL jest nowoczesnym narzędziem do programowania systemów sterowania binarnego, wykorzystującym w pełni naturę i możliwości mikroprocesorowych urządzeń sterujących.

Projektując nowy język programowania systemów sterowania binarnego jego autorzy stają przed dylematem pogodzenia wymogów ugruntowanej wśród użytkowników tradycji z nowoczesnością. Sterowanie procesami binarnymi ma swój długi rodowód. Wielu użytkowników ciągle jeszcze postrzega rozwiązywanie problemów z tej dziedziny w kategoriach konstrukcji urządzeń przekaźnikowych, mimo że stosowane obecnie w systemach sterowania binarnego urządzenia mikroprocesorowe mają zupełnie odmienną naturę. Pomijając nawet fakt, że tradycyjne podejście do problemu zwykle nie pozwala na naturalne i eleganckie zapisanie algorytmu działania sterowanego obiektu, to uniemożliwia ono także pełne wykorzystanie urządzeń mikroprocesorowych, których możliwości są nieporównywalnie większe niż możliwości urządzeń tradycyjnych.

Mając na uwadze powyższy dylemat autorzy prezentowanego języka starali się stworzyć narzędzie proste i wygodne a jednocześnie skuteczne. Mieliśmy świadomość, że nowe technologie, narzucając stosowanie nowych technik rozwiązywania problemów, stwarzają programistom kłopoty polegające na konieczności wytworzenia pewnych, odmiennych od dotychczasowych, nawyków i sposobów podejścia do rozwiązywanych problemów. Dlatego staraliśmy się, aby nasz język dawał użytkownikowi poczucie naturalności opisu obiektu przez program, a jednocześnie pozwalał na eleganckie rozwiązywanie bardziej złożonych zagadnień, dzięki zastosowaniu metod nowoczesnej informatyki, takich jak np. mechanizmy wielozadaniowości.

Podczas analizy działania obiektów binarnych naturalnym podejściem jest podział cyklu ich pracy na pewną liczbę pojedynczych, dość elementarnych operacji. Operacje

te (zwane krokami) mają ustaloną kolejność wykonywania. Można także sprecyzować warunki przejścia od wykonywania jednej operacji do następnej.
Aby sprostać wymaganiom naturalności opisu obiektu przez program, koncepcję języka oparto na pojęciu „kroku". W swej ogólnej postaci program składa się z sekwencji zdarzeń opisanych jako sekwencje „kroków". Każdy z „kroków" określa stan opisywanego obiektu na pewien przedział czasu, co w ogólności obejmuje śledzenie wejść i wysterowywanie wyjść.
Prezentowany tutaj język nosi nazwę EARL od EAsy Robot Language. Nazwa języka pochodzi stąd, że historycznie rzecz biorąc, sterownik, na którym zaimplementowano go po raz pierwszy, był przeznaczony do sterowania robotem prostym.
Propozycja nowego języka została opracowana przy założeniu, że program będzie przygotowywany przy użyciu panelu programowania lub za pomocą dowolnego komputera. W tym drugim przypadku, dzięki oprogramowaniu zainstalowanemu w komputerze, program może być napisany niejako „na boku", tj. bez wykorzystania sterownika, sprawdzony pod względem poprawności formalnej i przetworzony do postaci nadającej się do załadowania do sterownika. Ten drugi sposób przygotowywania programu będzie rozszerzony o możliwości jego przetestowania przy wykorzystaniu specjalnego programu symulacyjnego.

2. OPIS JĘZYKA

2.1. Program i zadania

Język EARL umożliwia dekompozycję złożonego algorytmu sterowania na szereg prostszych jednostek, tzw. zadań. Zapis taki pozwala na łatwe odzwierciedlenie sytuacji, gdy współpracuje ze sobą kilka niezależnych urządzeń kontrolowanych przez jeden sterownik lub gdy mamy do czynienia z jednym urządzeniem składającym się z szeregu podzespołów. Nawet wtedy, gdy sterujemy procesem technologicznym ściśle sekwencyjnym, logiczne wydzielenie osobnych procesów może być bardzo wygodne. Takie typowe, najczęściej występujące zadania, to: praca automatyczna (tu może być zbiór zadań), wykrywanie i obsługa stanów awaryjnych, praca ręczna.
Tekst programu rozpoczyna się od słowa kluczowego PROGRAM, po którym musi wystąpić nazwa programu. Nazwa programu jest dowolnej długości i może się składać zarówno z liter jak i cyfr, przy czym pierwszym znakiem nazwy musi być litera. Koniec programu oznaczany jest napisem EOP.
Program może składać się z wielu oddzielnych zadań, z których każde rozpoczyna się od słowa kluczowego TASK, po którym musi wystąpić nazwa zadania. Reguły tworzenia nazw zadań są takie same jak reguły tworzenia nazw programów. Nazwy zadań w programie nie mogą się powtarzać. Koniec każdego zadania jest oznaczany napisem EOT.
Strukturę programu można przedstawić następująco:

```
PROGRAM <nazwa programu>
  TASK <nazwa zadania>
     treść zadania
  EOT
        ⋮
  TASK <nazwa zadania>
     treść zadania
  EOT
EOP
```

Zadaniem jest sekwencja kroków zawartych pomiędzy słowami kluczowymi TASK i EOT. Liczba kroków nie jest ograniczona. Jedynym ograniczeniem jest rozmiar pamięci sterownika. Definicje zadań nie mogą być zagnieżdżone (tzn. definicja jednego zadania nie może wystąpić pomiędzy napisami TASK i EOT definiującymi inne zadanie).

Program musi zawsze zawierać dokładnie jedno zadanie o nazwie *main*. Zadanie to jest uruchamiane jako pierwsze, po załączeniu sterownika.

Język pozwala na równoległe wykonywanie co najwyżej 16 zadań. Z punktu widzenia użytkownika zadania są wykonywane równolegle, w rzeczywistości oznacza to pracę z podziałem czasu. Zadania mogą się ze sobą komunikować i wpływać wzajemnie na swoje wykonanie.

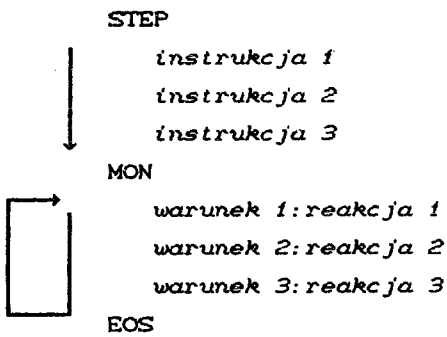
2.2. Krok

Zadanie w języku EARL składa się z ciągu kroków. Każdy krok odpowiada wykonaniu pewnych czynności (np. ustawieniu wyjść binarnych sterownika) oraz poczekaniu na reakcję urządzeń połączonych ze sterownikiem wejściami sygnałowymi. Z chwilą gdy reakcja, na jaką oczekuje się w danym kroku, zostanie wykryta, sterownik przechodzi do wykonywania kolejnego kroku. Początek każdego kroku oznaczany jest przez napis STEP. Kroki mogą być nazywane. Nazwa kroku, podobnie jak nazwa programu i zadania, jest ciągiem liter i cyfr. Pierwszym znakiem nazwy kroku musi być litera. W zadaniu nie mogą wystąpić dwa kroki o takich samych nazwach. Koniec kroku oznaczany jest przez napis EOS. Każdy krok ma strukturę dwuczęściową.

```
STEP
   część wykonawcza
        kroku
MON
   część nadzorująca
        kroku
EOS
```

Część pierwsza (wykonawcza) określa, co należy zrobić w danym kroku. Część druga (nadzorująca) mówi, które elementy (np. wejścia sterownika) mają być kontrolowane i jaki musi być ich stan, aby można było przejść do wykonywania następnego kroku. Początek kroku (napis STEP) jest zarazem początkiem części wykonawczej tego kroku. Początek części nadzorującej określa słowo MON. Końcem części wykonawczej jest (kończący także cały krok) napis EOS.

Dwuczęściowa struktura kroku odzwierciedla różne sposoby działania sterownika przy wykonywaniu każdej z części tego kroku.



Wykonując pierwszą część kroku sterownik wykonuje jedną po drugiej instrukcje w niej zawarte (w takiej kolejności, w jakiej zostały napisane). Z chwilą, gdy zostanie napotkany symbol MON, sekwencyjne wykonywanie instrukcji zostaje zastąpione cyklicznym sprawdzaniem warunków. Jeżeli został zapisany warunek, na przykład taki, że wejście siódme (IN7) musi być w stanie 1 (musi być ustawione), to sterownik będzie co odpowiedni takt czasu sprawdzał, czy na siódmym wejściu jest jedynka. Tak długo, jak długo na wejściu siódmym będzie zero, sterownik będzie w stanie oczekiwania i przejście do wykonywania innego kroku danego zadania nie będzie możliwe. Na przykład w kroku zdefiniowanym następująco:

```
STEP
   SET OUT5
MON
   IN1: NEXT
EOS
```

najpierw zostanie ustawione wyjście piąte (OUT5) na wartość jeden, a następnie sterownik będzie czekał, aż na wejściu pierwszym pojawi się jedynka. Wtedy dopiero zostanie podjęte wykonywanie następnego kroku.

Jeżeli w danym kroku wymagamy spełnienia jednego z kilku warunków, to wszystkie te warunki muszą zostać wyliczone po słowie MON. Po rozpoczęciu wykonywania

części nadzorującej, warunki te będą sprawdzane cyklicznie. Przejście do wykonywania kolejnego kroku nastąpi w momencie, gdy którykolwiek z podanych warunków zostanie spełniony.

Zarówno część wykonawcza kroku jak i część nadzorująca nie są obowiązkowe. Jeżeli część wykonawcza nie zawiera żadnej instrukcji, to wówczas następuje bezpośrednie przejście do cyklicznego sprawdzania warunków części nadzorującej. Jeżeli w części nadzorującej kroku nie występuje żaden warunek, to sterownik, po wykonaniu wszystkich instrukcji części wykonawczej, przechodzi do kolejnego kroku.

Kroki są wykonywane w zasadzie sekwencyjnie. Zmianę kolejności wykonywania kroków uzyskuje się przez wykonanie instrukcji GOTO lub przez wskazanie w części nadzorującej nazwy kroku, jaki ma być wykonany po spełnieniu danego warunku. Dla określenia, że kolejnym wykonywanym krokiem ma być krok, który w tekście zadania występuje jako następny, służy słowo NEXT. Nie trzeba więc podawać nazwy następnego kroku, co więcej, krok ten nie musi być nazwany.

Wykonanie instrukcji GOTO w części wykonawczej kroku spowoduje, że przejście do wykonywania kolejnego kroku (o nazwie podanej w tej instrukcji lub NEXT) nastąpi bez rozpoczęcia sprawdzania warunków części nadzorującej kroku.

Na przykład:

```
STEP   K7
   IF VAR1 = 12 THEN GOTO K8
   SET OUT3
MON
   IN1 : NEXT
   TOUT 150 : ERR
EOS
```

Z powyżej zdefiniowanego kroku możliwe są następujące wyjścia: skok do kroku K8 (bez wykonania części nadzorującej) w przypadku, gdy zmienna 1 będzie miała wartość 12. Przejście do następnego (według kolejności występowania w zadaniu) kroku, jeżeli przed upływem 15 sekund na wejściu pierwszym pojawi się 1. Przejście do kroku o nazwie ERR w przeciwnym przypadku (tzn. w sytuacji, gdy w czasie 15 sekund na wejściu pierwszym nie pojawi się 1).

Jeżeli krok nie zawiera żadnego warunku w części nadzorującej i nie jest wykonywana instrukcja GOTO, to po wykonaniu części wykonawczej kroku nastąpi przejście do następnego kroku (czyli kroku NEXT).

Po wykonaniu ostatniego kroku zadanie kończy się, tzn. przechodzi w stan ASLEEP. Możliwość cyklicznego wykonywania zadania można uzyskać jawnie używając instrukcji przejścia do pierwszego kroku zadania. Powtórzenie wykonania całego zadania, niezbędne przy pracy cyklicznej sterowanego urządzenia, uzyskuje się poprzez jawne napisanie instrukcji przejścia do pierwszego kroku zadania (GOTO).

2.3. Flagi

W języku EARL przewidziano istnienie tzw. flag, czyli zmiennych przeznaczonych do przechowywania informacji binarnych (0 lub 1). Flagi pozwalają na przykład na przechowywanie pośrednich wyników operacji logicznych. Każdą flagę oznacza się przez napis FLGK <*numer flagi*>, gdzie *numer flagi* jest liczbą z przedziału 0 – 255, np. FLG1, FLG17. Programujący może nadać fladze wartość 1 instrukcją SET lub wartość 0 instrukcją RES. Flaga zachowuje swoją wartość, dopóki nie pojawi się instrukcja zmieniająca tę wartość. Flagi, którym w programie nie nadano jeszcze żadnej wartości, są ustawione na 0.

2.4. Zmienne

W języku przewidziano typ danych zmienna. Zmienne są oznaczane przez VAR <*numer zmiennej*>, gdzie *numer zmiennej* jest liczbą z przedziału 0 – 999. Każda zmienna może zawierać liczbę całkowitą z przedziału –32768 – 32767. Zmienna może być wykorzystywana np. do zliczania części obrabianych przez sterowane urządzenie. Zmienna może być wyzerowana (instrukcją CLR). Jej wartość może być następnie zwiększana (instrukcją INC) lub zmniejszana (instrukcją DEC). Zwiększanie lub zmniejszanie wartości zmiennej polega na dodaniu (odjęciu) jedynki od jego dotychczasowej wartości. Używając pojedynczej instrukcji nie można zmienić wartości zmiennej o więcej niż jeden.

2.5. Liczniki czasu

Liczniki czasu są strukturą danych przeznaczoną do zliczania czasu. Czas jest odmierzany od momentu włączenia (instrukcją START) wyzerowanego (instrukcją CLR) licznika. Licznik czasu może zostać zatrzymany i ponownie uruchomiony (odpowiednio instrukcjami STOP i START), co może być wykorzystane do zliczania czasu wybranych operacji procesu technologicznego. Liczniki czasu są oznaczane w programie symbolami TIM <*numer licznika czasu*>, gdzie *numer licznika czasu* jest liczbą z przedziału 0 – 31. Jednostką czasu zliczaną w tych licznikach jest 10 ms. Wartość licznika czasu wynosząca 200 odpowiada zatem 2 sekundom.

2.6. Wejścia i wyjścia

Obiekt binarny komunikuje się z otoczeniem poprzez sygnały ustawiane na wyjściach i odczytywane z wejść. W języku EARL sygnały te są reprezentowane przez zmienne binarne oznaczane w programie odpowiednio przez IN <*numer*> i OUT <*numer*>, gdzie *numery* wejść i wyjść muszą być liczbami z zakresu 0 – 255.
Na pojedyncze wyjście mogą być wysyłane tylko wartości 0 lub 1, co jest realizowane poprzez instrukcje SET i RES (są to te same instrukcje, za pomocą których nadaje się wartości flagom). Przykładowo, wykonanie instrukcji:

SET OUT3

spowoduje ustawienie wartości 1 na wyjściu o numerze 3. Wartość ta będzie utrzymywana na tym wyjściu tak długo, dopóki nie pojawi się w programie instrukcja:

RES OUT3

wtedy na wyjściu trzecim pojawi się stan 0 i będzie utrzymywany aż do chwili ponownego wykonania instrukcji SET OUT3. Wielokrotne, kolejne wykonywanie instrukcji SET lub RES na tym samym wyjściu nie spowoduje zmiany jego stanu.

Wartości 1 i 0 reprezentują dwa różne stany – ustawiony i wyzerowany, które odpowiadają dwóm różnym poziomom napięć, jakie mogą wystąpić na pojedynczym wyjściu lub wejściu sterownika.

Wartości wejść nie mogą być ustawiane w programie. Wartości te są ustawiane przez urządzenia dołączone do sterownika i mogą być jedynie odczytywane przez program, który na tej podstawie określa czynności, jakie powinny być wykonane.

Znaczenie różnych stanów na wejściach i wyjściach sterownika, zatem poprawność ich interpretacji przez program, pozostaje pod całkowitą kontrolą osoby programującej, dlatego zaleca się stosowanie odpowiednich komentarzy.

2.7. Semafory

Semafory są zmiennymi używanymi do synchronizacji zadań. Są to zmienne wielowartościowe (oznaczane SEM <*numer semafora*>). Można na nich operować jedynie instrukcjami SIGNAL i WAIT. Przewidziano istnienie 32 semaforów numerowanych liczbami z zakresu 0 – 31. Szczegóły na temat semaforów, ich roli w programie i sposobów ich wykorzystywania podano w rozdziale 2.11.

2.8. Warunki

Przy opisie części nadzorowanej kroku powiedzieliśmy, że podczas wykonywania tej części sprawdzane są warunki. Oto kilka przykładów takich warunków:

IN3 – ten warunek będzie spełniony, gdy na trzecim wejściu sterownika pojawi się wartość 1.

IN3 AND IN4 – spełnienie tego warunku wymaga, aby wartość 1 wystąpiła równocześnie na dwóch wejściach sterownika (trzecim i czwartym).

IN5 AND FLG3 – warunek będzie spełniony, gdy stan na wejściu piątym będzie 1 i flaga trzecia będzie ustawiona.

NOT IN3 – ten warunek będzie spełniony, gdy na wejściu trzecim będzie stan 0.

FLG3 OR FLG4 – warunek będzie spełniony, gdy co najmniej jedna z podanych flag będzie ustawiona.

Ogólnie można powiedzieć, że warunek formułowany jest poprzez podanie operacji

logicznych na wartościach dwustanowych (flagach, wejściach, wyjściach). Dopuszczone są 3 operatory logiczne: alternatywa (OR), koniunkcja (AND) i negacja (NOT). Operatory OR i AND są operatorami dwuargumentowymi (muszą mieć podane dwa argumenty znajdujące się po dwóch stronach operatora). Operator NOT jest operatorem jednoargumentowym, a jego argumentem jest wartość zapisana po prawej stronie operatora. Wynik wykonania pojedynczej operacji jest zawsze wartością 0 lub 1. Wynik ten może być argumentem innego operatora, możliwe są zatem wyrażenia wielooperatorowe, np:

IN3 OR IN4 OR IN5
IN4 AND IN5 OR IN3
NOT IN3 AND FLG7

Wyrażenia są zawsze obliczane w kolejności od lewego do prawego operatora z uwzględnieniem priorytetów poszczególnych operatorów. Najwyższy priorytet ma operator NOT a najniższy OR. Zmianę kolejności wykonywania obliczeń uzyskuje się poprzez użycie nawiasów. Na przykład:

IN3 OR IN4 AND IN5 — w tym wyrażeniu najpierw zostanie wykonany operator AND a następnie operator OR (którego prawym argumentem będzie wynik operacji AND dla argumentów IN4 i IN5).

(IN3 OR IN4) AND IN5 — poprzez użycie nawiasu kolejność obliczeń zostaje zmieniona. Najpierw zostanie wykonana operacja OR z argumentami IN3 i IN4, a następnie operacja AND z argumentami: wynik operacji OR i IN5.

Wszystkie powyższe warunki były wyrażeniami logicznymi operującymi na wartościach binarnych. W języku EARL warunek można również określić jako operację porównania dwóch wartości arytmetycznych (zmiennych, liczników czasu, liczb). W jednym warunku można porównać tylko dwie wartości przy użyciu jednego z sześciu operatorów:

=	(równe)
<>	(nierówne)
<	(mniejsze)
>	(większe)
<=	(mniejsze lub równe)
>=	(większe lub równe)

Wynikiem porównania jest zawsze wartość logiczna 1 lub 0. Jeżeli warunek jest spełniony, to wynik ma wartość 1, a jeżeli nie jest spełniony, to wynik ma wartość 0.

Przykłady:

VAR1 <VAR2 — Sprawdzenie, czy zmienna 1 ma wartość mniejszą od wartości zmiennej 2. Jeżeli tak, to warunek jest spełniony.

TIM1 >= 150 Sprawdzenie, czy pierwszy licznik czasu odmierzył już 15 sekund (150/10). Jeżeli tak, to warunek jest spełniony.

2.9. Przekroczenie czasu

Opisując ogólnie krok powiedzieliśmy, że w części nadzorującej kroku występują warunki. Oprócz warunków opisanych w poprzednim punkcie, w części nadzorującej kroku może wystąpić dodatkowo badanie przekroczenia pewnego czasu. Często występuje potrzeba ograniczenia czasu czekania na spełnienie warunków. Na przykład czekanie na wystąpienie zadanego stanu na jakimś konkretnym wejściu sterownika nie powinno trwać dłużej niż pewien ustalony odcinek czasu. Niespełnienie warunku w oczekiwanym czasie stanowi informację, że program powinien wykonać określoną akcję np. wysłać sygnał na odpowiednie wyjście.

Przekroczenie czasu definiuje się przez napisanie słowa TOUT <*liczba jednostek czasu*>. Jednostką czasu w instrukcji TOUT jest, podobnie jak przy licznikach czasu, 10 ms, np.

TOUT 400

oznacza polecenie czekania przez 4 sekundy.

2.10. Instrukcje części wykonawczej kroku

- Instrukcja skoku (przejścia)

GOTO NEXT
GOTO <*nazwa kroku*>

Instrukcja skoku realizuje skok bezwarunkowy. Jeżeli po słowie GOTO występuje napis NEXT, to następuje przejście do wykonywania następnego (zapisanego jako następny) kroku. Jeżeli po słowie GOTO występuje *nazwa kroku*, to następuje przejście do wykonywania kroku o podanej nazwie.

- Instrukcje ustawiania i zerowania flag i wyjść

SET FLG <*numer flagi*>
SET OUT <*numer wyjścia*>
RES FLG <*numer flagi*>
RES OUT <*numer wyjścia*>

Instrukcja SET ustawia wartość podanej flagi lub wyjścia w stan 1. Jeżeli dana flaga lub wyjście było w stanie 1 w chwili wykonania instrukcji SET, to stan ten nie ulega zmianie.

Instrukcja RES (reset) ustawia wartość podanej flagi lub wyjścia w stan 0. Jeżeli dana flaga lub wyjście było w stanie 0 w chwili wykonywania instrukcji RES, to stan ten nie ulegnie zmianie.

- Instrukcja zerowania zmiennych i liczników

CLR VAR <*numer zmiennej*>
CLR TIM <*numer licznika czasu*>

Instrukcja CLR powoduje wyzerowanie zmiennej lub licznika, który został podany jako argument tej instrukcji.

- Instrukcje zwiększenia i zmniejszenia wartości zmiennej

INC VAR <*numer zmiennej*>
DEC VAR <*numer zmiennej*>

Wykonanie instrukcji INC powoduje zwiększenie wartości danej zmiennej o jeden, a DEC zmniejszenie tej wartości (również o jeden).

- Instrukcje startowania i zatrzymywania licznika czasu

START TIM <*numer licznika czasu*>
STOP TIM <*numer licznika czasu*>

Instrukcja START powoduje rozpoczęcie odmierzania czasu i zapisywanie kolejnych jednostek upływu czasu (10 ms) w liczniku TIM o numerze *numer licznika czasu*. Jeżeli licznik nie był uprzednio wyzerowany (instrukcją CLR), to kolejne jednostki czasu są dodawane do poprzedniej wartości licznika czasu. Instrukcja STOP powoduje zatrzymanie odmierzania czasu przez podany licznik czasu. Wykonanie tej instrukcji nie powoduje zerowania licznika.

- Instrukcja warunkowa

IF *warunek* THEN *instrukcja*

Instrukcja warunkowa pozwala na wykonanie *instrukcji* wtedy i tylko wtedy, gdy spełniony jest warunek występujący po słowie IF. Warunek musi mieć postać taką, jak opisano w punkcie WARUNKI (rozdz. 2.6.) Instrukcja występująca po słowie THEN może być dowolną instrukcją za wyjątkiem instrukcji warunkowej. Jeżeli warunek jest spełniony, to wykonywana jest *instrukcja* zapisana po słowie THEN, a następnie kolejna instrukcja po instrukcji warunkowej. Jeżeli warunek nie jest spełniony, to *instrukcja* występująca po słowie THEN nie jest wykonywana i sterownik przechodzi od razu do wykonywania kolejnej instrukcji po instrukcji warunkowej. Na przykład:

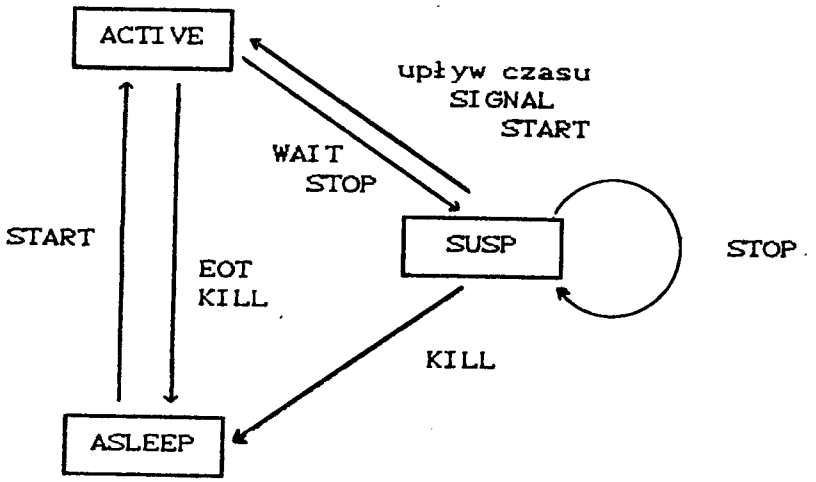
```
IF IN7 AND FLG3 THEN INC VAR3
RES OUT5
```

Instrukcja INC VAR3 zostanie wykonana tylko wtedy, gdy będą jednocześnie ustawione trzecia flaga i siódme wejście. Jeżeli warunek ten nie będzie spełniony, to wyjście piąte zostanie ustawione w stan 0 bez uprzedniego zwiększenia wartości trzeciej zmiennej.

Osobną grupą instrukcji są instrukcje związane ze sterowaniem i synchronizacją pracy zadań. Instrukcje te, podobnie jak opisane wyżej, mogą występować jedynie w części wykonawczej kroku. Ich opis zamieszczono w następnym rozdziale.

2.11. Sterowanie pracą zadań i ich synchronizacja

Zadanie może się znajdować w jednym z następujących stanów: ACTIVE, SUSP, ASLEEP (rys. 1.1.).



Rys. 2.1. Graf stanów zadania

Stan ACTIVE jest stanem aktywnym zadania. System sterowania realizuje jedynie algorytmy zadań aktywnych, tj. znajdujących się w stanie ACTIVE. Zadania aktywne są wykonywane równolegle z punktu widzenia sterowanego obiektu.

Stan ASLEEP jest stanem nieaktywnym zadania. Po zmianie stanu z nieaktywnego na stan ACTIVE zadanie rozpoczyna działanie od swojego pierwszego kroku.

Stan SUSP jest stanem zawieszenia zadania. Zadania zawieszone, podobnie jak zadania nieaktywne, nie są wykonywane przez system sterowania. Zadanie w stanie zawieszenia oczekuje na spełnienie odpowiedniego warunku po czym staje się aktywne. Warunkiem tym może być:

- upływ czasu,
- podniesienie odpowiedniego semafora,
- podniesienie odpowiedniego semafora lub upływ czasu.

W przypadku gdy zadanie zostało zawieszone bezwarunkowo, do stanu ACTIVE może je przenieść jedynie jawna aktywizacja przez inne zadanie. Po uaktywnieniu zawieszone zadanie wznawia pracę od miejsca, w którym zostało zatrzymane.

Równoległe wykonywanie kilku zadań odbywa się asynchronicznie, tj. każde zadanie wykonuje kolejne kroki ze swoją własną prędkością. Po to, aby dane zadania mogły koordynować swoje przebiegi, potrzebne są środki umożliwiające ich wzajemną synchronizację.

W języku EARL przewidziano synchronizację zadań poprzez semafory Dijkstry.

Semaforem jest obiekt danych, oznaczany przez SEM <*numer semafora*>, z którym związana jest wartość (zawsze całkowita, nieujemna) oznaczająca liczbę sygnałów wysłanych i jeszcze nie odebranych oraz kolejka zadań oczekujących na sygnał.

Programista ma do dyspozycji następujące instrukcje służące do sterowania pracą zadań i ich wzajemnej synchronizacji.

- Instrukcja SIGNAL

SIGNAL SEM <*numer semafora*>

Działanie instrukcji polega na podniesieniu semafora. Wykonanie tej operacji powoduje testowanie wartości semafora. Jeśli jest zerem – wtedy jedno zadanie z kolejki oczekujących zadań jest aktywowane. Jeśli wartość semafora jest większa od zera lub jeśli kolejka jest pusta – wtedy wykonanie operacji SIGNAL powoduje zwiększenie wartości semafora o jeden.

- Instrukcja WAIT

WAIT SEM <*numer semafora*>

Działanie instrukcji WAIT polega na opuszczeniu semafora. Wykonanie tej operacji powoduje testowanie wartości semafora. Jeśli jest dodatnia, wtedy zmniejsza się ją o jeden. Jeśli zero (nie ma żadnego oczekującego sygnału) to zadanie jest umieszczane na końcu kolejki zadań związanych z tym semaforem a jego wykonywanie zawiesza się. W chwili rozpoczęcia wykonywania programu wszystkie semafory są podniesione (mają wartość zero).

- Instrukcja LWAIT (ang. Limited WAIT)

LWAIT SEM <*numer semafora*>, <*liczba jednostek czasu*>

Działanie instrukcji LWAIT jest takie same jak WAIT z tą różnicą, że drugi argument instrukcji wprowadza ograniczenie na czas przez jaki wykonywanie zadania może być zawieszone. Czas ten nie może być dłuższy niż określa to argument instrukcji pomnożony przez 10 ms. Po upływie tego czasu zadanie niezależnie od stanu semafora przechodzi do stanu aktywnego.

- Instrukcja START

START <*nazwa zadania*>

Działanie instrukcji START polega na uruchomieniu zadania, tj. na bezwarunkowym przeniesieniu go do stanu aktywnego. W zależności od tego w jakim stanie znajdowało się zadanie dla którego wydano rozkaz START, jego wykonywanie rozpoczyna się od początku (jeśli było w stanie ASLEEP) lub od miejsca, w którym zostało zatrzymane (jeśli było w stanie SUSP).

- Instrukcja STOP

STOP <*nazwa zadania*>

Instrukcja STOP umożliwia bezwarunkowe zawieszenie zadania, czyli przeniesienie go ze stanu ACTIVE do stanu SUSP. Wykonanie operacji STOP dla zadania, które jest już w stanie SUSP i oczekuje na spełnienie jakiegoś warunku, spowoduje przejście tego zadania w stan zawieszenia bezwarunkowego.

- Instrukcja KILL

KILL <*nazwa zadania*>

Instrukcja KILL kończy wykonywanie zadania, czyli przeprowadza je w stan ASLEEP.

2.12. Komentarze

Programista może stosować komentarze w pisanym przez siebie programie. Komentarz zaczyna się znakiem ";". Komentarze nie zmieniają w żaden sposób samej treści programu, ani nie wpływają na przebieg jego wykonania.

Wszystkie nadmiarowe spacje są ignorowane, tzn. w miejscach, gdzie spacja jest separatorem ciągi spacji są traktowane jak pojedyncze spacje, natomiast w innych miejscach spacje są ignorowane.

3. PRZYKŁADY

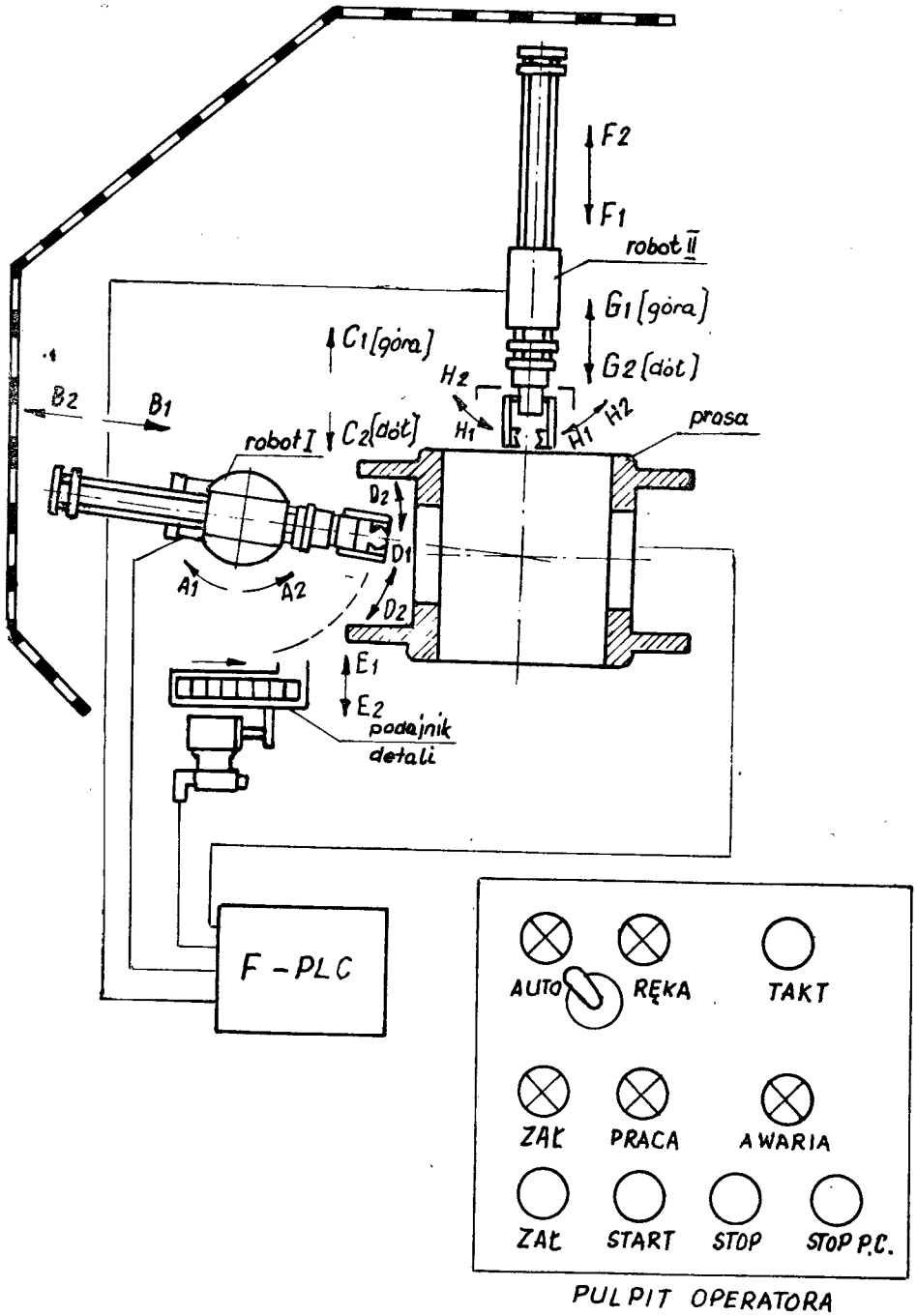
Poniżej przedstawiono zestaw przykładów, w których obiektem sterowanym jest stanowisko składające się z dwu robotów prostych PR–02 i prasy z wahającą matrycą. Schemat stanowiska pokazano na rysunku 3.1. Robot I pobiera detale z podajnika i wkłada je pod prasę. Robot II wyjmuje gotowe detale spod prasy i wrzuca je do pojemnika. Cykl roboczy prasy trwa stosunkowo długo (kilka sekund). W tym czasie robot I zdąży pobrać nowy detal z podajnika i ustawić się w pozycji A2 (gotowy do włożenia detalu pod prasę).

Po włączeniu zasilania roboty znajdują się w położeniu przypadkowym na czujnikach krańcowych a prasa w położeniu z uniesioną matrycą. Przed rozpoczęciem pracy roboty powinny zostać zsynchronizowane, tzn. należy dobrać taką kombinację sygnałów sterujących, która będzie odpowiadała aktualnemu położeniu robotów i w związku z tym umożliwi ich bezuderzeniowe sprowadzenie do pozycji początkowej. Pozycja początkowa stanowiska jest następująca:

- robot I: A2, B2, C2, D2,
- robot II: F2, G2, H2,
- podajnik: E2.

Stanowisko posiada pulpit operatora (rys. 3.1.). Umożliwia on wybór rodzaju pracy za pomocą przełącznika AUTO/RĘKA. Wybrany tryb pracy jest sygnalizowany zapaleniem odpowiedniej lampki:

AUTO – praca automatyczna,
RĘKA – praca ręczna.



Rys. 3.1. Schemat stanowiska roboczego złożonego z prasy i dwóch robotów

W trybie pracy automatycznej stanowisko działa cyklicznie, tak jak opisano to na wstępie, wykonując kolejno następujące operacje: pobranie detalu z magazynka, włożenie go w otwór prasy, tłoczenie, wyjęcie detalu spod prasy i wrzucenie go do pojemnika. Cykl ten jest uruchamiany od bieżącego położenia stanowiska przyciskiem START. Wykonywanie cyklu automatycznego sygnalizuje lampka PRACA. Pracę automatyczną można przerwać natychmiast przyciskiem STOP lub po zakończeniu cyklu przyciskiem STOP P. C. Po naciśnięciu każdego z tych przycisków natychmiast gaśnie lampka PRACA.

W czasie pracy automatycznej mogą zaistnieć dwie sytuacje awaryjne. Pierwsza z nich występuje wtedy, gdy czujnik umieszczony w szczękach robota I sygnalizuje, że robot nie trzyma detalu w momencie, gdy jego ramię znajduje się w otworze prasy. Druga sytuacja awaryjna polega na stwierdzeniu braku detalu w zamkniętych szczękach chwytaka robota II, tuż po tym jak wysunął on swoje ramię z otworu prasy. W pierwszym przypadku program ustawia robota I w pozycji A1, B2, C1, D2, wyłącza zasilanie robotów, podajnika i prasy. W drugim przypadku program tylko wyłącza zasilanie. W obu przypadkach zostaje zapalona lampka AWARIA i zgaszona lampka ZAŁ. Wystąpienie awarii należy potwierdzić przyciskiem STOP, co przeprowadza stanowisko w stan zwykłego zatrzymania gasząc lampkę PRACA. Następnie możliwe jest ponowne włączenie zasilania przyciskiem ZAŁ i uruchomienie stanowiska.

W trybie pracy ręcznej stanowisko również pracuje cyklicznie, jednakże zatrzymuje się samoczynnie po wykonaniu każdej operacji, oczekując na naciśnięcie przycisku TAKT (istotne zbocze a nie poziom sygnału z przycisku). W pracy ręcznej nie są wykrywane sytuacje awaryjne.

Zmiana trybu pracy stanowiska jest możliwa tylko po jego uprzednim zatrzymaniu.

LISTA WYJSC

```
OUT0  - AUTO praca automatyczna
OUT1  - RĘKA praca ręczna
OUT2  - ZAŁ stan zasilania stanowiska
OUT3  - PRACA sygnalizacja wykonywania cyklu automatycznego
OUT4  - AWARIA
OUT5  - sterowanie obrotem podstawy robota I: 0 - A2, 1 - A1
OUT6  - sterowanie ruchem poziomym robota I: 0 - B2, 1 - B1
OUT7  - sterowanie ruchem pionowym chwytaka robota I:
        0 - C2, 1 - C1
OUT24 - sterowanie zamykaniem i otwieraniem szczęk robota I:
        0 - zamykanie, 1 - otwieranie
OUT25 - sterowanie podajnikiem detali: 0 - E2, 1 - E1
OUT26 - sterowanie ruchem poziomym robota II: 0 - F2, 1 - F1
OUT27 - sterowanie ruchem pionowym chwytaka robota II.
```

```
        0 - G2, 1 - G1
OUT28 - sterowanie zamykaniem i otwieraniem szczęk robota II:
        0 - G2, 1 - G1
OUT29 - załączanie prasy: 1 - włącz, 0 - wyłącz
OUT30 - awaryjne wyłączanie zasilania: 0 - wyłącz, 1 - włącz
```

LISTA WEJSC

```
IN8  - AUTO/RĘKA przełącznik rodzaju pracy: 0 - RĘKA, 1 - AUTO
IN9  - TAKT
IN10 - ZAŁ wyłącznik zasilania
IN11 - START pracy automatycznej
IN12 - STOP P.C. stop po cyklu
IN13 - STOP natychmiastowy
IN14 - robot I w położeniu A1 (rys. 2.1)
IN15 - robot I w położeniu A2
IN32 - robot I w położeniu B1
IN33 - robot I w położeniu B2
IN34 - robot I w położeniu C1
IN35 - robot I w polozeniu C2
IN36 - robot II w położeniu F1
IN37 - robot II w położeniu F2
IN38 - robot II w położeniu G1
IN39 - robot II w położeniu G2
IN56 - sygnał informujący o obecności detalu w szczękach
       robota I
IN57 - sygnał informujący o obecności detalu w szczękach
       robota II
IN58 - sygnał informujący o zakonczeniu cyklu pracy prasy
```

PRZYKŁAD 1

Napisz program synchronizacji i sprowadzania do położenia początkowego robota 1. Sprowadzenie do pozycji początkowej ma nastąpić po naciśnięciu przycisku ZAŁ.

```
program synchro
;==============

   task main              ; zadanie synchronizuje i sprowadza
  ;---------              ; do położenia początkowego robota 1
      step A1_A2          ; synchronizacja kolumny robota 1
      mon
```

```
        in14 : next
        not in14 : B1_B2
    eos
    step A1
        set out5
    eos
    step B1_B2          ; synchronizacja ramienia robota 1
    mon
        in32 : next
        not in32 : C1_C2
    eos
    step B1
        set out6
    eos
    step C1_C2          ; synchronizacja chwytaka robota 1
    mon
        in34 : next
        not in34 : ZAL
    eos
    step C1
        set out7
    eos
    step ZAL            ; po synchronizacji - oczekiwanie na
    mon                 ; przycisk ZAŁ
        in10 : next
    eos
    step
        set out2        ; zapal lampkę ZAŁ
        res out6        ; wycofaj ramię robota 1 do położenia B2
    mon
        in33 : next
    eos
    step                ; sprowadzenie do położenia początkowego
        res out5        ; pozostałych osi robota 1
        res out7
        res out24
    mon
        tout 50 and in15 and in35 : next
    eos

eot

eop
```

Przy synchronizacji skorzystano z faktu, że układ sterowania po włączeniu zasilania ma wszystkie wyjścia wyłączone. W związku z tym podczas synchronizacji należy odpowiednie wyjścia tylko załączyć.

PRZYKŁAD 2

Napisz program zawierający zadanie o nazwie „robot2", które realizuje automatyczną pracę robota 2. Nie wykrywaj stanów awaryjnych w pracy robota. Przyjmij, że:

- w chwili początkowej robot znajduje się w położeniu F2, G2, H2,
- wyjmowanie detalu spod prasy należy rozpocząć po otrzymaniu na wejściu IN58 sygnału o zakończeniu przez prasę cyklu pracy i wyłączeniu prasy sygnałem OUT29.

```
program robot2                  program realizujący pracę robota 2 w
;=============                  trybie automatycznym

   task main
   ;--------
      step
         start robot2
         kill main
      eos
   eot

   task robot2                  ; zadanie realizuje cykl pracy robota 2
  ;-----------                  ----------------------------------------
      step                      ; oczekiwanie na koniec cyklu prasy
      mon
         in58 : next
      eos
      step loop                 ; prasa zakończyła tłoczenie
         res out29              ; wyłącz prasę
         set out26              ; wysuń ramię robota do położenia F1
      mon
         in36 : next
      eos
      step                      ; chwytanie detalu
         set out28              ; zamknij szczęki chwytaka
         wait 50                ; czekanie na zamknięcie szczęk
         set out27              ; podnieś chwytak do położenia G1
      mon
         in38 : next
      eos
      step                      ; wycofaj ramię robota do położenia F2
         res out26
      mon
         in37 : next
      eos
      step                      ; wrzuć detal do pojemnika
         res out28              ; otwórz szczęki chwytaka
         res out27              ; opuść chwytak do położenia G2
      mon
         tout 50 and in39 and in58 : loop
      eos
   eot

eop
```

PRZYKŁAD 3

Napisz program realizujący pracę stanowiska w trybie automatycznym. Program powinien składać się z dwu zadań o nazwach „robot1" i „robot2". Zadanie „robot1" realizuje pracę automatyczną robota 1. Podobnie „robot2" realizuje pracę automatyczną robota 2. Zadanie „robot1" uruchamia prasę sygnałem OUT29. Zadania nie wykrywają stanów awaryjnych w pracy robotów. Praca robotów przy wyjmowaniu detalu i wkładaniu następnego powinna tak być zsynchronizowana by ich ramiona nie zderzyły się. W chwili początkowej stanowisko znajduje się w pozycji wyjściowej.

```
program auto                ; program automatycznej pracy robotów
;===========

   task main
   ;--------
      step
         start robot1
         start robot2
         kill main
      eos
   eot

   task robot1              ; zadanie realizuje cykl pracy robota 1
   ;-----------             ----------------------------------------
      step loop             ; obrót robota 1 w stronę podajnika
        set out5            ; - położenie A1'
      mon
        in14 : next
      eos
      step                  ; podanie detalu przez podajnik
        set out25
        wait 50             ; czekanie na podanie detalu
        set out24           ; chwycenie detalu
        wait 50             ; czekanie na zamknięcie szczęk
        res out25           ; wycofanie poajnika
      eos
      step                  ; podnoszenie detalu
         set out7           ; położenie C1
      mon
         in34 : next
      eos
      step                  ; obrót robota 1 w stronę prasy
        res out5              położenie A2
      mon
        in15 : next
      eos
      step                  ; wkładanie detalu pod prasę
        wait sem0           ; ******* oczekiwanie na wyjęcie detalu
        set out6            ; położenie B1
      mon
        in32 : next
      eos
      step                  ; umieszczanie detalu w gnieździe prasy
```

```
      res out7          ; położenie C2
    mon
      in35 : next
    eos
    step                ; puść detal
      res out24         ; otwórz szczęki robota 1
      wait 50           ; czekanie na otwarcie szczęk
      res out6          ; wysuń ramię robota 1 spod prasy
    mon
      in33 : next
    eos
    step
      set out29         ; włącz prasę
      goto loop
    eos
eot

task robot2             ; zadanie realizuje cykl pracy robota 2
;-----------            ----------------------------------------
    step                ; oczekiwanie na koniec cyklu prasy
    mon
      in58 : next
    eos
    step loop           ; prasa zakończyła tłoczenie
      res out29         ; wyłącz prasę
      set out26         ; wysuń ramię robota do położenia F1
    mon
      in36 : next
    eos
    step                ; chwytanie detalu
      set out28         ; zamknij szczęki chwytaka
      wait 50           ; czekanie na zamkniecie szczęk
      set out27         ; podnieś chwytak do położenia G1
    mon
      in38 : next
    eos
    step                ; wycofaj ramię robota do położenia F2
      res out26
    mon
      in37 : next
    eos
    step                ; wrzuć detal do pojemnika
      signal sem0       ; ******* synchronizacja wkładania detalu
      res out28         ; otwórz szczęki chwytaka
      res out27         ; opuść chwytak do położenia G2
    mon
      tout 50 and in39 and in58 : loop
    eos
eot

eop
```

PRZYKŁAD 4

Każde wyjście układu sterowania jest wyposażone w indywidualną sygnalizację za pomocą diody LED stanu sygnału. Załączenie wyjścia zapala jego diodę. Wyłączenie gasi ją. Mając do dyspozycji cztery wyjścia OUT24–27, wejście IN32 oraz wcześniej opisany pulpit operatora napisz program, który realizuje cyklicznie przemieszczający się, czasowo uzależniony, punkt świetlny sterowany z pulpitu operatora. Operacją jest w tej sytuacji przemieszczenie punktu świetlnego na następną pozycję. Pojawienie się stanu 1 na wejściu IN32 symuluje awarię. W programie powinno być wyróżnione jedno zadanie o nazwie „punkt", realizujące ruch punktu świetlnego. Pozostała część programu poprzez odpowiednie oddziaływanie na to zadanie wykonuje wszystkie funkcje pulpitu operatora.

```
program pulpit              ; program sterowania ruchem punktu za pomocą
;=============              ; pulpitu operatora

   task main
   ;--------
      step
         start pulpit
         kill main
      eos
   eot

   task pulpit              ; zadanie obsługi pulpitu operatora
   ;----------
      step reka             ; stan pracy ręcznej
         set out1           ; zapal RĘKA
         res out0           ; zgaś AUTO
         res out3           ; zgaś PRACA
         set flg1           ; odblokuj instrukcje wait sem1
         start punkt
         start takt         ; uruchom przycisk taktujący pracę ręczną
      mon
         in8 :next          ; testowanie położenia AUTO przełącznika
      eos                     AUTO/REKA
      step auto_stp         ; stan stopu po pracy ręcznej
         res out1           ; zgaś RĘKA
         set out0           ; zapal AUTO
         res flg1           ; zablokuj instrukcje wait sem1
         kill takt          ; zablokuj przycisk taktujący pracę ręczną
      mon
         not in8 :reka      ; testowanie położenia RĘKA w AUTO/STOP
         in11 : next        ; testowanie przycisku START
      eos
      step
         signal sem1
      eos
      step auto             ; stan pracy automatycznej
         set out3           ; zapal PRACA
      mon
         in12 : next        ; testowanie przycisku STOP P.C.
         in13 : stop_r      ; testowanie przycisku STOP
      eos
      step stop_pc          ; stan stopu po cyklu
```

```
        res out3          ; zgaś PRACA
        set flg2          ; ustaw flagę blokująca pracę automatyczną
                          ; po końcu cyklu
     mon
        in11 : next       ; testowanie przycisku START
        in8 : reka        ; testowanie położenia REKA w AUTO/REKA
     eos
     step                 ; uruchom pracę automatyczną
        res flg2
        goto auto
     eos
     step stop_r          ; stan stopu
        res out3          ; zgaś PRACA
        stop punkt
     mon
        in11 : next       ; testowanie przycisku START
        not in8 : reka    ; testowanie położenia RĘKA w AUTO/REKA
     eos
     step                 ; uruchom pracę automatyczną
        start punkt
        goto auto
     eos
eot

task takt                 ; zadanie obsługi klawisza taktującego
;----------               ; w pracy ręcznej
   step test              ; oczekiwanie na klawisz taktujący
   mon
      in9 : next
   eos
   step                   ; podniesienie semafora
      signal sem1
   mon                    ; oczekiwanie na puszczenie klawisza
      not in9 : test
   eos
eot

task punkt                ; zadanie realizujące ruch punktu
;---------               ----------------------------------
   step loop
      if flg1 then wait  sem1
      res out27
      set out24
   mon
      tout 100 : next
   eos
   step
      if flg1 then wait  sem1
      res out24
      set out25
   mon
      tout 100 : next
   eos
   step
      if flg1 then wait  sem1
      res out25
      set out26
   mon
      tout 100 : next
   eos
```

```
            step
               if flg1 then wait  sem1
               res out28
               set out27
            mon
               tout 100 and not flg2 : loop
            eos
            eot

      eop
```

PRZYKŁAD 5

Opierając się na poprzednich przykładach napisz program realizujący pełne sterowanie opisanym na wstępie stanowiskiem, składającym się z dwu robotów, prasy i pulpitu operatora

```
program prasa                ; program pełnego sterowania stanowiskiem
;============                   składającym się z dwóch rbotów i prasy

   task main
   ;--------
      step A1_A2             ; synchronizacja kolumny robota 1
      mon
         in14 : next
         not in14 : B1_B2
      eos
      step A1
         set out5
      eos
      step B1_B2             ; synchronizacja ramienia robota 1
      mon
         in32 : next
         not in32 : C1_C2
      eos
      step B1
         set out6
      eos
      step C1_C2             ; synchronizacja chwytaka robota 1
      mon
         in34 : next
         not in34 : F1_F2
      eos
      step C1
         set out7
      eos
      step F1_F2             ; synchronizacja ramienia robota 2
      mon
         in36 : next
         not in36 : G1_G2
      eos
      step F1
         set out26
      eos
      step G1_G2             ; synchronizacja chwytaka robota 2
```

```
      mon
         in38 : next
         not in38 : ZAL
      eos
      step G1
         set out27
      eos
      step ZAL              ; po synchronizacji - oczekiwanie na
      mon                   ; przycisk ZAŁ
         in10 : next
      eos
      step
         set out2           ; zapal lampkę ZAŁ
         res out6           ; wycofaj ramie robota 1 do położenia B2
         res out26          ; wycofaj ramię robota 2 do położenia F2
      mon
         in33 : next
      eos
      step                  ; sprowadzenie do położenia początkowego:
         res out5           ; pozostałych osi robota 1
         res out7
         res out24
         res out27          ; pozostałych osi robota 2
         res out28
      mon
         tout 50 and in15 and in35 and in39 : next
      eos
      step                  ; próbny cykl pracy prasy
         set out29
      mon
         in58 : next
      eos
      step
         res out29          ; zatrzymaj prasę
         start pulpit
         kill main
      eos
eot

task pulpit                 ; zadanie obsługi pulpitu operatora
;----------
   step reka                ; stan pracy ręcznej
      set out1              ; zapal RĘKA
      res out0              ; zgaś AUTO
      res out3              ; zgaś PRACA
      set flg1              ; odblokuj instrukcje wait sem1 w zadaniach
                            ; robot1 i robot2
      start robot1
      start robot2
      start takt            ; uruchom przycisk taktujący pracę ręczną
   mon
      in8 :next             ; testowanie położenia AUTO w AUTO/RĘKA
   eos
   step auto_stp            ; stan stopu po pracy ręcznej
      res out1              ; zgaś RĘKA
      set out0              ; zapal AUTO
      res flg1              ; zablokuj instrukcje wait sem1 w zadaniach
                            ; robot1 i robot2
      kill takt             ; zablokuj przycisk taktujący pracę ręczna
   mon
```

```
      not in8 :reka     ; testowanie położenia RĘKA w AUTO/STOP
      in11 : next       ; testowanie przycisku START
   eos
   step
      signal sem1
      signal sem1
   eos
   step auto            ; stan pracy automatycznej
      set out3          ; zapal PRACA
   mon
      flg0 : awaria     ; testowanie stanów awaryjnych
      in12 : next       ; testowanie przycisku STOP P.C.
      in13 : stop_r     ; testowanie przycisku STOP
   eos
   step stop_pc         ; stan stopu po cyklu
      res out3          ; zgaś PRACA
      set flg2          ; ustaw flagę blokującą pracę automatyczną
                        ; po końcu cyklu
   mon
      in11 : next       ; testowanie przycisku START
      in8 : reka        ; testowanie położenia RĘKA w AUTO/RĘKA
   eos
   step                 ; uruchom pracę automatyczną
      res flg2
      goto auto
   eos
   step stop_r          ; stan stopu
      res out4          ; zgaś AWARIA w trybie potwierdzenia stopu
                        ; awaryjnego
      res out3          ; zgaś PRACA
      stop robot1
      stop robot2
   mon
      in11 : next       ; testowanie przycisku START
      not in8 : reka    ; testowanie położenia RĘKA w AUTO/RĘKA
   eos
   step                 ; uruchom pracę automatyczną
      start robot1
      start robot2
      goto auto
   eos
   step awaria          ; reakcja na stan awaryjny
      set out4          ; zapal lampkę AWARIA
   mon
      not in13 : stop_r ; oczekiwanie na potwierdzenie awarii
   eos                  ; przyciskiem STOP

eot

task takt               ; zadanie obsługi klawisza taktującego
;----------             ; w pracy ręcznej
   step test            ; oczekiwanie na klawisz taktujący
   mon
      in9 : next
   eos
   step                 ; podniesienie semafora
      signal sem1
   mon                  ; oczekiwanie na puszczenie klawisza
      not in9 : test
   eos
```

```
eot

task robot1              ; zadanie realizuje cykl pracy robota 1
;-----------             ----------------------------------------
    step loop            ; obrót robota 1 w stronę podajnika
       if flg1 then wait  sem1 ; synchronizacja w pracy ręcznej
       set out5          ; położenie A1
    mon
       in14 : next
    eos
    step                 ; podanie detalu przez podajnik
       set out26
       wait 50           ; czekanie na podanie detalu
       set out24         ; chwycenie detalu
       wait 50           ; czekanie na zamknięcie szczęk
       res out25         ; wycofanie poajnika
    eos
    step                 ; podnoszenie detalu
       set out7          ; położenie C1
    mon
       in34 : next
    eos
    step                 ; obrót robota 1 w stronę prasy
       res out5            położenie A2
    mon
       in15 : next
    eos
    step                 ; wkładanie detalu pod prasę
       if flg1 then wait  sem1 ; synchronizacja w pracy ręcznej
       wait sem0         ; ****** oczekiwanie na wyjęcie detalu
       set out6          ; położenie B1
    mon
       not in56 : next ; sprawdzanie obecności detalu w szczękach
       in32 : OK
    eos
    step                 ; awaria - zgubiono detal
       set flg0          ; ustaw sygnał awarii
       stop robot1       ; zatrzymaj robota
    eos
    step OK              ; umieszczanie detalu w gnieździe prasy
       res out7          ; położenie C2
    mon
       in35 : next
    eos
    step                 ; puść detal
       res out24         ; otwórz szczęki robota 1
       wait 50           ; czekanie na otwarcie szczęk
       res out6          ; wysuń ramię robota 1 spod prasy
    mon
       in33 : next
    eos
    step
       if flg1 then wait  sem1 ; synchronizacja w pracy ręcznej
       set out29         ; włącz prasę
    mon
       not flg2 : loop
    eos
eot

task robot2              ; zadanie realizuje cykl pracy robota 2
```

```
;-----------                ----------------------------------------
    step                   ; oczekiwanie na koniec cyklu prasy
    mon
       in58 : next
    eos
    step loop              ; prasa zakończyła tłoczenie
       res out29           ; wyłącz prasę
       if flg1 then wait  sem1 ; synchronizacja w pracy ręcznej
       set out26           ; wysuń ramię robota do położenia F1
    mon
       in36 : next
    eos
    step                   ; chwytanie detalu
       set out28           ; zamknij szczęki chwytaka
       wait 50             ; czekanie na zamknięcie szczęk
       set out27           ; podnieś chwytak do położenia G1
    mon
       in38 : next
    eos
    step                   ; wycofaj ramię robota do położenia F2
       res out26
    mon
       not in57 : next     ; sprawdzanie obecności detalu w szczękach
       in37 : OK
    eos
    step                   ; awaria - zgubiono detal
       set flg0            ; ustaw sygnał awarii
       stop robot2         ; zatrzymaj robota
    eos
    step OK                ; wrzuć detal do pojemnika
       signal sem0         ; ******* synchronizacja wkładania detalu
       if flg1 then wait  sem1 ; synchronizacja w pracy ręcznej
       res out28           ; otwórz szczęki chwytaka
       res out27           ; opuść chwytak do położenia G2
    mon
       tout 50 and in39 and in58 and not flg2 : loop
    eos
eot

eop
```

4. SKŁADNIA JĘZYKA EARL

<program> ::=

PROGRAM<nazwa "programu"><treść programu>EOP

<treść programu> ::=

{<zadanie>}*

<zadanie> ::=

TASK<nazwa "zadania"><treść zadania>EOT

```
<treść zadania> ::=
    {<krok>}*

<krok> ::=
    STEP[<nazwa "kroku">]<treść kroku>EOS

<treść kroku> ::=
    [<część wykonawcza>][MON<część nadzorująca>]

<część wykonawcza> ::=
    {<instrukcja>}+

<część nadzorująca> ::=
    [TOUT<liczba dziesiętna>:<przejście>]
                {<warunek>:<przejście>}+

<przejście> ::=
    <nazwa "kroku">
    !NEXT

<instrukcja> ::=
    <instrukcja prosta>!<instrukcja złożona>

<instrukcja złożona> ::=
    IF<warunek>THEN<instrukcja prosta>

<instrukcja prosta> ::=
    !GOTO <przejście>
    !SET(<flaga>!<wyjście>)
    !RES(<flaga>!<wyjście>)
    !CLR(<zmienna>!<licznik czasu>)
    !INC <zmienna>
    !DEC <zmienna>
    !START (<licznik czasu>!<nazwa "zadania">)
    !STOP (<licznik czasu>!<nazwa "zadania">)
    !WAIT (<semafor>[,<liczba jedn. czasu>]!<liczba jedn. czasu>)
    !KILL<nazwa "zadania">

<flaga> ::=
    FLAG<numer "flagi">
```

```
<zmienna> ::=
    VAR<numer "zmiennej">

<licznik czasu> ::=
    TIM<numer "licznika czasu">

<wejscie> ::=
    IN<numer "wejścia">

<wyjscie> ::=
    OUT<numer "wyjścia">

<semafor> ::=
    SEM<numer "semafora">

<warunek> ::=
    {<wart. arytmet.>}<operator relacji>{<wart. arytmet>}:
                                        <wyrażenie logiczne>

<wyrażenie logiczne> ::=
    <składnik logiczny>{OR<składnik logiczny>}*

<składnik logiczny> ::=
    <czynnik logiczny>{AND<czynnik logiczny>}*

<czynnik logiczny> ::=
     <wejście>
    |<wyjście>
    |<flaga>
    |(<wyrażenie logiczne>)
    |NOT<czynnik logiczny>

<nazwa> ::=
    <litera>{<litera>|<cyfra>}*

<numer> ::=
    <liczba dziesiętna>

<wartość arytmetyczna> ::=
```

```
    <licznik>
    |<licznik czasu>
    |<liczba dziesiętna>

<liczba dziesiętna> ::=
    <<cyfra>>+

<cyfra> ::=
    0|1|2|3|4|5|6|7|8|9

<operator relacji> ::=
    =|<>|<|>|<=|>=
```