ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΚΗ ΑΣΚΗΣΗ ΜΕΤΑΦΡΑΣΤΕΣ

(ΥΛΟΠΟΙΗΣΗ ΜΕΤΑΦΡΑΣΤΗ ΓΙΑ ΤΗΝ ΓΛΩΣΣΑ STARLET)

Ονοματεπώνημα και Α.Μ ομάδας:

- Ηλιάνα Βλάχου 2411
- Σπύρος Τσότζολας 3099

➢ ΣΥΝΟΨΗ:

Στην παρούσα προγραμματιστική άσκηση μας ζητήθηκε να φτιάξουμε έναν μεταγλωττιστή ο οποίος παίρνει σαν είσοδο ένα πρόγραμμα σε γλώσσα Starlet και παράγει το αντίστοιχο πρόγραμμα σε Assembly. Το πρόγραμμα του μεταγλωττιστή είναι γραμμένο σε python και για την δημιουργία του ακολουθήσαμε συγκεκριμένες φάσεις μεταγλώττισης, οι οποίες αναλύονται παρακάτω. Στόχος του μεταγλωττιστή είναι η δημιουργία προγράμματος σε γλώσσα μηχανής, καθώς και η εμφάνιση κατατοπιστικών μηνυμάτων για την διόρθωση συντακτικών λαθών που μπορεί να υπάρχουν σε ένα πρόγραμμα εισόδου.

➢ ΠΕΡΙΕΧΟΜΕΝΑ:

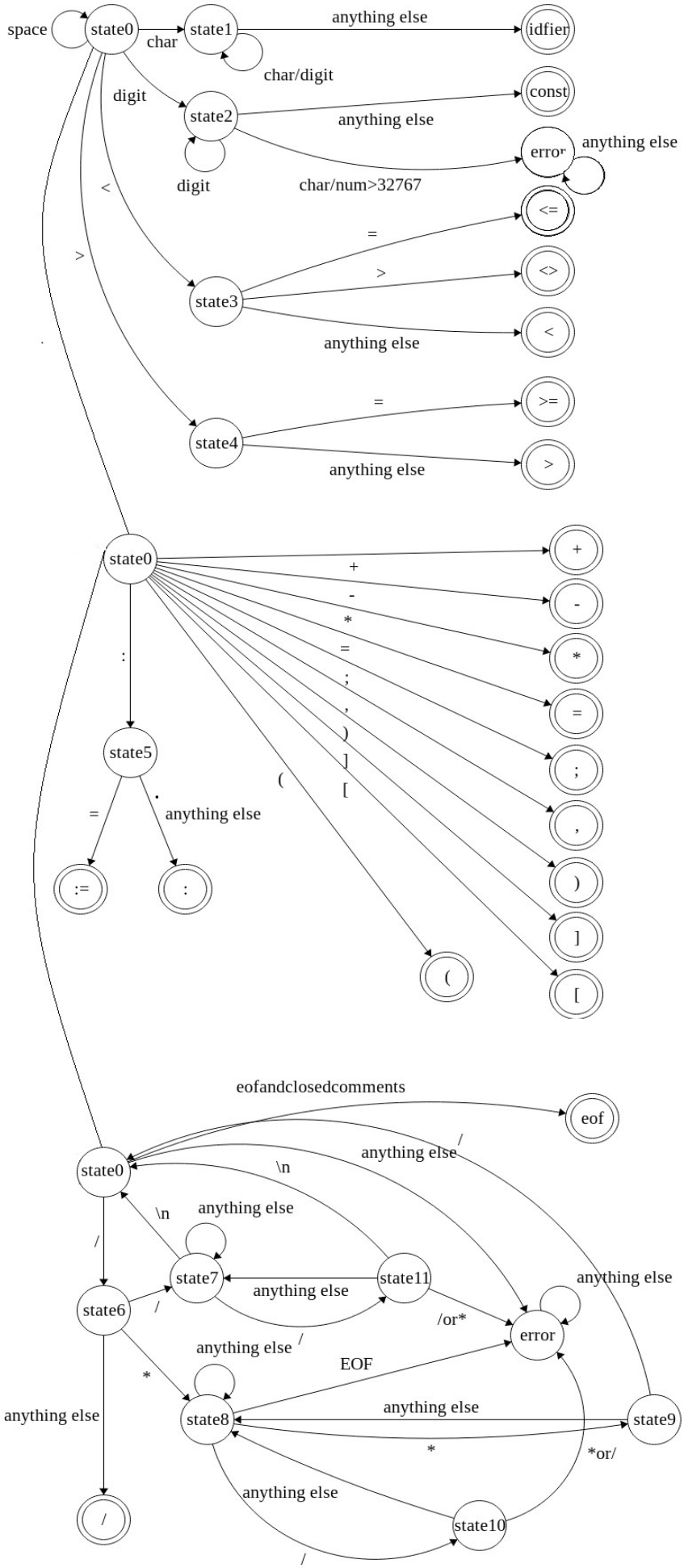
1. ΛΕΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ

- **Λειτουργία λεκτικού αναλυτή:**

Η λειτουργία του Λεκτικού Αναλυτή (ΛΑ) είναι να διαβάζει ένα-ένα τους χαρακτήρες του πηγαίου (Source) προγράμματος και να τους μεταφράζει σε tokens (π.χ. keywords, identifiers, constants). Το στάδιο αυτό πραγματεύεται τον σχεδιασμό και την υλοποίηση του ΛΑ. Επειδή χρειάζεται κάποιος τρόπος περιγραφής των διαφόρων tokens που εμφανίζονται σε κάποια γλώσσα, εισάγονται οι Κανονικές Εκφράσεις (Regular Expressions). Ακόμη χρειάζεται κάποιος μηχανισμός που να αναγνωρίζει τα tokens της γλώσσας. Τέτοιοι μηχανισμοί (token Recognizers) είναι τα Διαγράμματα Μετάβασης Καταστάσεων (Transition Diagrams) και Πεπερασμένα Αυτόματα (Finite Automata). 'Ενα πλεονέκτημα της χρήσης των Κανονικών Εκφράσεων για τον καθορισμό των tokens είναι το γεγονός ότι από μια Κανονική 'Εκφραση μπορούμε να κατασκευάσουμε αυτόματα ένα αναγνωριστή για τα tokens (token Recognizer) που παριστάνονται με Κανονικές Εκφράσεις. Ο Λεκτικός μας Αναλυτής συνεργάζεται με τον Συντακτικό Αναλυτή. Συγκεκριμένα ο Λεκτικός Αναλυτής λειτουργεί σαν υπορουτίνα και καλείται από τον Συντακτικό Αναλυτή όταν αυτός χρειάζεται κάποιο token. Ο σκοπός του διαχωρισμού της Λεκτικής και Συντακτικής Ανάλυσης είναι η απλοποίηση του όλου σχεδιασμού του Μεταγλωττιστή.

- **Το πεπερασμένο αυτόματο που υλοποιήσαμε:**

Στην δικιά μας υλοποιήση ΛΑ ο μηχανισμός που χρησιμοποιήσαμε για να αναγνωρίζει τα tokens της γλώσσας είναι ένα πεπερασμένο αυτόματο. Το αυτόματο μας αναγνωρίζει όλες τις δεσμευμένες λέξεις της γλώσσας starlet, τα επιτρεπτά σύμβολα της γλώσσας, αναγνωριστικά, σταθερές, καθώς και πιθανά λάθη(π.χ μη επιτρεπτός χαρακτήρας, κλείσιμο σχολίων χωρίς να έχουν ανοίξει προηγουμένως). Παρακάτω σας το παρουσιάζουμε σχηματικά:


space
state0
char
state1
anything else
idfier
char/digit
digit
state2
const
anything else
digit
char/num>32767
error
anything else
<
>
state3
=
>
anything else
<=
<>
<
state4
=
anything else
>=
>
state0
+
-
*
=
;
,
)
]
[
(
:
state5
=
anything else
:=
:
eofandclosedcomments
eof
state0
anything else
/
\n
\n
anything else
/
state7
anything else
state11
anything else
state6
/
/or*
error
anything else
/
EOF
*
anything else
state8
anything else
state9
*
*or/
anything else
state10
/
/

2. ΣΥΝΤΑΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ

Ο συντακτικός αναλυτής έπεται του λεκτικού αναλυτή. Σε αυτή την φάση ελέγχεται εάν το πρόγραμμα ανήκει στην γλώσσα της οποίας η σύνταξη ορίζεται από μια δεδομένη γραμματική χωρίς συμφραζόμενα, και δημιουργείται το κατάλληλο «περιβάλλον» μέσα στο οποίο έπειτα θα κληθούν οι σημαντικές ρουτίνες. Είσοδος του συντακτικού αναλυτή είναι το αρχικό πρόγραμμα, και έξοδος του είναι μία ένδειξη ότι το πρόγραμμα είναι ή όχι συντακτικά ορθό. Υπάρχουν πολλοί τρόποι να κατασκευαστεί ένας συντακτικός αναλυτής, σε αυτήν την περίπτωση προτιμήθηκε η συντακτική ανάλυση με αναδρομική κατάβαση βασισμένη σε γραμματική LL(1).

Η εσωτερική λειτουργία ενός συντακτικού αναλυτή ορίζεται από τις παρακάτω περιπτώσεις:

- Για κάθε έναν από τους κανόνες της γραμματικής φτιάχνουμε και ένα αντίστοιχο υποπρόγραμμα-ρουτίνα.
- Όταν συναντάμε μη τερματικό σύμβολο καλούμε το αντίστοιχο υποπρόγραμμα.
- Όταν συναντάμε τερματικό σύμβολο τότε:
    1. Εάν και ο λεκτικός αναλυτής επιστρέφει λεκτική μονάδα που αντιστοιχεί στο τερματικό αυτό σύμβολο έχουμε αναγνωρίσει επιτυχώς τη λεκτική μονάδα.
    2. Αντίθετα εάν ο λεκτικός αναλυτής δεν επιστρέψει τη λεκτική μονάδα που περιμένει ο συντακτικός αναλυτής, έχουμε λάθος και καλείται ο διαχειριστής σφαλμάτων
- Όταν αναγνωριστεί και η τελευταία λέξη του πηγαίου προγράμματος, τότε η συντακτική ανάλυση έχει στεφτεί με επιτυχία.

Συγκεκριμένα η συντακτική ανάλυση ξεκινάει έχοντας δημιουργηθεί η πρώτη λεκτική μονάδα από τον λεκτικό αναλυτή και καλώντας την συνάρτηση program(). Εκεί καλείται ξανά ο λεκτικός αναλυτής ο οποίος επιστρέφει την επόμενη λεκτική μονάδα. Σύμφωνα με την γραμματική της Starlet ένα πρόγραμμα είναι αποδεκτό όταν ξεκινάει με την δεσμευμένη λέξη *program* και έπειτα μια λεκτική μονάδα τύπου *id* . Στην συνέχεια ξανακαλείται ο λεκτικός αναλυτής ώστε να δοθεί η επόμενη λεκτική μονάδα και να συνεχιστεί ο συντακτικός έλεγχος. Για αυτήν την λεκτική μονάδα η program() καλεί την συνάρτηση block() η οποία αναφέρεται στον επόμενο κανόνα την γραμματικής. Μέσω της block() καλούνται και άλλοι κανονες της γραμματικής διαδοχικά και καλώντας κάθε φορά τον

λεκτικό αναλυτή για την επόμενη λεκτική μονάδα. Τέλος επιστρέφοντας από την block() η τελευταία επιτρεπόμενη λεκτική μονάδα είναι η *endprogram.* Με τον ίδιο τρόπο λειτουργούν όλες οι συναρτήσεις που αντιστοιχούν σε κανόνες γραμματικής. Μόλις ελεγχθεί και η τελευταία λεκτική μονάδα(το endprogram) ο συντακτικός αναλυτής εμαφανίζει μήνυμα σωστής συντακτικής ανάλυσης:

'*The starlet programm was compiled without problem*'

Εάν ωστόσο κατά την διάρκεια της συντακτικής ανάλυσης υπάρχει λάθος σύμφωνα με την γραμματική σου εμφανίζει 'Syntax error' και σου αναφέρει την λεκτική μονάδα που περίμενε να εμφανιστεί , καθώς και την γραμμή του προγράμματος όπου βρίσκεται το λάθος. Με αυτόν τον τρόπο βοηθάει τον χρήστη να βρεί ευκολότερα το λάθος του και να το αντικαταστήσει με το σωστό.

Παρακάτω αναφέρονται προγράμματα ως είσοδοι στον συντακτικό αναλυτή, καθώς και το μονοπάτι συναρτήσεων-κανόνων της γραμματικής που ακολούθησαν και τέλος το μήνυμα που εμφανίζεται.

<u>Παράδειγμα 1:</u>

```
program example3
      declare a,b,c,d,e,x,y,px,py,temp;
       loop
            if (not[a<c and b<d]) then
                  exit
           endif;
           if (a=e) then
                  c:=c+e
            endif
       endloop;
      x:=1;
endprogram
```

Το μονοπάτι συναρτήσεων που ακολουθεί το πρόγραμμα είναι:

1. program()
2. block()
3. declaration()
4. varlist()
5. subprograms()
6. statements()
7. statement()
8. loopStat()
9. statements()
10. statement()
11. ifStat()
12. condition()
13. boolterm()
14. boolfactor()
15. condition()
16. boolterm()
17. boolfactor()
18. expression()
19. optionalSign()
20. term()
21. factor()
22. idtail()
23. relationalOper()
24. expression()
25. optionalSign()
26. term()
27. factor()
28. idtail()
29. boolfactor
30. expression()
31. optionalSign()
32. term()
33. factor()
34. idtail()
35. relationalOper()
36. expression()
37. optionalSign()

38. term()
39. factor()
40. idtail()
41. statement()
42. exitStat()
43. elsepart()
44. ifStat()
45. condition()
46. boolterm()
47. boolfactor()
48. expression()
49. optionalSign()
50. term()
51. factor()
52. idtail()
53. relationOper()..
54. expression()
55. optionalSign()
56. term()
57. factor()
58. idtail()
59. statements()
60. assigmentStat()
61. expression()
62. optionalSign()
63. term()
64. factor()
65. idtail()
66. elsepart()
67. assigmentStat()
68. expression()
69. optionalSign()
70. term()
71. factor()
72. idtail()

Το τελικό μήνυμα που εμφανίζεται είναι : '*The starlet programm was compiled without problem*'

Παράδειγμα 2:

```
program example1
        declare d,i,g,f;
        function two (in g)
                declare k;
                if (k>I then
                        o:=q-1
                endif;
                j:=j*k;
                k:=k+g
                while (k<1);
                        m:=j;
                x:=7
        endfunction
        g:=two(in g)
endprogram
```

Το μονοπάτι που ακολουθείται από τον συντακτικό αναλυτή :

1. program()
2. block()
3. declarations()
4. varlist()
5. subprograms()
6. subprogram()
7. funcbody()
8. formalpars()
9. formalparlist()
10. formalparitem()
11. block()
12. declarations()
13. varlist()
14. subprograms()
15. statements()
16. ifStat()
17. condition()
18. boolterm()
19. boolfactor()
20. expression()

21. optionalSign()
22. term()
23. relationOper()
24. expression()
25. optionalSign()
26. term()

Σε αυτό το σημείο ο συντακτικός αναλυτής τερματίζει την διαδικασία και εμφανίζει το ανάλογο μήνυμα λάθους:
'Syntax error: A ")" was expected in line:5'.

3. ΕΝΔΙΑΜΕΣΟΣ ΚΩΔΙΚΑΣ

Η αμέσως επόμενη φάση έπειτα από την ολοκλήρωση του συντακτικού αναλυτή, είναι η παραγωγή του ενδιάμεσου κώδικα. Σε αυτό το σημείο στόχος μας είναι η δημιουργία ενός ενδιάμεσου προγράμματος σε κάποια «ενδιάμεση γλώσσας», το οποίο στην συνέχεια θα χρησιμοποιηθεί για την παραγωγή του τελικού κώδικα σε assembly. Στην δική μας περίπτωση δημιουργούμε ένα αρχείο-ενδιάμεσο πρόγραμμα με κατάληξη .int, καθώς και ένα αρχείο με πρόγραμμα C γλώσσας, με κατάληξη .c.

Για την παραγωγή των παραπάνω αρχείων χρειάστηκε να κρατήσουμε δύο λίστες από τετράδες (quads για την ενδιάμεση γλώσσα και Cquads για την C), όπου κάθε τετράδα αποτελεί μία απλή εντολή της ενδιάμεσης γλώσσας και περιέχει μία ετικέτα, έναν τελεστή και τα τελούμενα. Οι τετράδες παράγονται μέσα από συγκεκριμένες συναρτήσεις (αντίστοιχες με κανόνες γραμματικής) που δημιουργήθηκαν στην προηγούμενη φάση και με την χρήση απαραίτητων βοηθητικών συναρτήσεων. Εφόσον ολοκληρωθεί η ανάγνωση εντολών του προγράμματος εισόδου και η παραγωγή των τετράδων της λίστας, δημιουργείτε τελικά το αρχείο με τις εντολές ενδιάμεσου κώδικα.

Πιο συγκεκριμένα για την παραγωγή των τετράδων, απαραίτητη είναι η χρήση βοηθητικών συναρτήσεων, οι οποίες παρουσιάζονται παρακάτω:

nextquad(): επιστρέφει το αριθμό – label της επόμενης τετράδας.

genquad(op,x,y,z): δημιουργεί την επόμενη τετράδα με βάση τα ορίσματα.

newTemp(): επιστρέφει την επόμενη προς χρήση προσωρινή μεταβλητή π.χ. t_4.

emptylist(): δημιουργεί και επιστρέφει μια κενή λίστα.

makelist(x): δημιουργεί και επιστρέφει μία λίστα με στοιχείο το όρισμα x.

mergelist(x,y): ενώνει στην λίστα x, τις λίστες x, y.

backpatch(x,z): αναζητάει τις τετράδες των οποίων τα labels περιέχει η λίστα x και συμπληρώνει στην κενή θέση την ετικέτα z.

Για τον ενδιάμεσο κώδικα σε C χρησιμοποιούνται οι παραπάνω συναρτήσεις, εκτός των genquad() και backpatch(), που αντικαθιστούνται από τις Cgenguad() και Cbackpatch().

Στην συνέχεια, για την παραγωγή του κώδικα πρέπει να προστεθούν επιπλέον εντολές στις συναρτήσεις- κανόνες γραμματικής. Οι εντολές αυτές αφορούν είτε το πέρασμα πληροφορίας από έναν κανόνα στον άλλο, είτε την δημιουργία λιστών με labels σε τετράδες που δεν είναι ολοκληρωμένες, είτε τέλος την παραγωγή της τετράδας- εντολής. Παρακάτω αναφέρονται οι κανόνες γραμματικής καθώς και επεξηγηματικά η προσθήκη κώδικα σε αυτούς.

<program> ::= **program** id {p0} <block> **endprogram** {p1}

{p0} : παράγει την πρώτη εντολή του προγραμματος id, «begin_block».

{p1} : παράγει τις δύο τελευταίες εντολές του προγραμματος, το «halt» και το «endblock».

<assignment-stat> ::= id := <expression> {p0}

{p0} : παράγει την επόμενη εντολή εκχώρησης της τιμης expression.place στην μεταβλητή id.

<if-stat> ::= **if (** <condition> **) then** {p0} <statements> {p1} <elsepart> **endif** {p2}

{p0} : συμπληρώνει τις τετράδες των οποίων τα labels περιέχει η λίστα condTrue, η οποία δημιουργείται μέσα στον κανόνα condition, με το label της επόμενης τετράδας.

{p1} : δημιουργεί μια λίστα iflist η οποία περιέχει το label της επόμενης τετράδας, την οποία παράγει και είναι ένα jump το οποίο θα πρέπει να σε οδηγεί στις εντολές του κώδικα μετά το endif και τέλος συμπληρώνει τις τετράδες των οποίων τα labels περιέχει η λίστα condFalse, η οποία δημιουργείται μέσα στον κανόνα condition, με το label της επόμενης τετράδας.

{p2} : συμπληρώνει τις τετράδες των οποίων τα labels περιέχει η λίστα iflist με το label της επόμενης τετράδας.

<while-stat> ::= **while (** {p0} <condition> **)** {p1} <statements> {p2} **endwhile**

{p0} : κρατάει σε μία μεταβλητή το label της επόμενης τετράδας η οποία θα είναι η συνθήκη έτσι ώστε να μπορεί μετά να κάνει jump στην συνθήκη και να γίνεται η επανάληψη.

{p1} : συμπληρώνει τις τετράδες των οποίων τα labels περιέχει η λίστα condTrue , η οποία δημιουργείται μέσα στον κανόνα condition , με το label της επόμενης τετράδας

{p2} : παράγει την επόμενη τετράδα η οποία είναι το jump στην συνθήκη και συμπληρώνει τις τετράδες των οποίων τα labels περιέχει η λίστα condFalse , η οποία δημιουργείται μέσα στον κανόνα condition , με το label της επόμενης τετράδας έτσι ώστε στην περίπτωση που δεν ισχύει η συνθήκη να οδηγείται στον κώδικα εκτός του while.

<dowhile-stat> ::= **dowhile** {p0} <statements> **endwhile** <condition>{p1}

{p0} : κρατάει σε μία μεταβλητή το label της επόμενης τετράδας η οποία θα είναι η πρώτη εντολή της επανάληψης ,έτσι ώστε να μπορεί μετά να κάνει jump από την συνθήκη στην πρώτη εντολή και να γίνεται η επανάληψη.

{p1} : συμπληρώνει τις τετράδες των οποίων τα labels περιέχει η λίστα condFalse και condTrue, η οποίες δημιουρ{γείται μέσα στον κανόνα condition , με το label της επόμενης τετράδας και το label της πρώτης συνθήκης αντίστοιχα.

<loop-stat> ::= **loop** {p0} <statements> {p1} **endloop**

{p0} : κρατάει σε μία μεταβλητή το label της επόμενης τετράδας η οποία θα είναι η πρώτη εντολή της επανάληψης ,έτσι ώστε να μπορεί μετά να κάνει jump από την εκεί και να γίνεται η επανάληψη.

{p1} : παράγει την τετράδα με το jump στην πρώτη εντολή της επανάληψης.

<exit-stat> ::= **exit** {p0}

{p0} : επεκτίνει την λίστα exitlist με το label της επόμενης τετραδας και παράγει κώδικα jump .

<forcase-stat> ::= **forcase** {p0}
(**when (** <condition> **) :** {p1} **<**statements> {p2})*
default : <statements> **enddefault**
endforcase {p3}

{p0} : δημιουργεί μια λίστα κενή exitlist η οποία θα περιέχει τα labels των τετράδων jump που θα οδηγούν εκτός της forcase.

{p1} : συμπληρώνει τις τετράδες των οποίων τα labels περιέχει η λίστα condTrue, η οποία δημιουργείται μέσα στον κανόνα condition, με το label της επόμενης τετράδας.

{p2} : προσθέτει στην exitlist το label της επόμενης εντολής- τετράδας jump , την οποία και παράγει, και συμπληρώνει τις τετράδες των οποίων τα labels περιέχει η λίστα condFalse, η οποία δημιουργείται μέσα στον κανόνα condition, με το label της επόμενης τετράδας.

{p3} : συμπληρώνει τις τετράδες των οποίων τα labels περιέχει η λίστα exitlist ,με το label της επόμενης τετράδας.

<incase-stat> ::= **incase** {p0}
(**while (** <condition> {p1} **) :** <statements> {p2})*
endincase

{p0} : κρατάει σε μία μεταβλητή το label της επόμενης τετράδας , και δημιουργεί μία καινούρια μεταβλητή την οποία θα χρησιμοποιήσουμε έτσι ώστε να γίνεται έλεγχος αν έχει γίνει κάποιο condition true, και παράγει κώδικα ο οποίος ορίζει την μεταβλητή αυτή στο 0.

{p1} : συμπληρώνει τις τετράδες των οποίων τα labels περιέχει η λίστα condTrue , η οποία δημιουργείται μέσα στον κανόνα condition , με το label της επόμενης τετράδας.

{p2} : παραγει κώδικα που ορίζει την μεταβλητή που δημιουργήσαμε παραπάνω στο 1, και συμπληρώνει τις τετράδες των οποίων τα labels περιέχει

η λίστα condFalse, η οποία δημιουργείται μέσα στον κανόνα condition , με το label της επόμενης τετράδας.

<return-stat> ::= **return** <expression> {p0}

{p0} : παράγει εντολή τύπου retv για το expression.place.

<print-stat> ::= **print** <expression> {p0}

{p0} : παράγει εντολή τύπου out για το expression.place.

<input-stat> ::= **input** id {p0}

{p0} : παράγει εντολή τύπου input για το id.

<actualparitem> ::= **in** <expression> {p0} | **inout** id {p1} |
inandout id {p2}

{p0} : παράγει εντολή τύπου par για το expression.place.

{p1} : παράγει εντολή τύπου par για το id.

{p0} : παράγει εντολή τύπου par για το id.

<condition> ::= <boolterm> {p0} (**or** {p1} **<**boolterm> {p2})*

{p0} : περνάει την πληροφορία από τις λίστες booltermTrue booltermFalse, οι οποίες προέρχονται από τον κανόνα boolterm , στις condTrue και condFalse αντίστοιχα.

{p1} : συμπληρώνει τις τετράδες των οποίων τα labels περιέχει η λίστα condFalse , με το label της επόμενης τετράδας.

{p2} : περνάει στο condFalse την πληροφορία της λίστας booltermFalse από την καινούρια κλήση της συνάρτης.

<boolterm> ::= <boolfactor> {p0} (**and** {p1} <boolfactor> {p2})*

{p0} : περνάει την πληροφορία από τις λίστες boolfactorTrue boolfactorFalse, οι οποίες προέρχονται από τον κανόνα boolterm , στις booltermTrue και booltermFalse αντίστοιχα.

{p1} : συμπληρώνει τις τετράδες των οποίων τα labels περιέχει η λίστα booltermTrue , με το label της επόμενης τετράδας.

{p2} : συνενώνει τις λίστες booltermFalse και boolfactorFalse από την καινούρια κλήση της boolfactor και περνάει την πληροφορία της καινούριας boolfactorTrue στην booltermTrue.

<boolfactor>::= **not [** <condition> {p0} **] | [** <condition> {p1}**] |**
<expression> <relational-oper><expression> {p2}

{p0}: περνάει στις λίστες boolfactorTrue και boolfactorFalse την πληροφορία των condFalse και condTrue αντίστοιχα.

{p1}: περνάει στις λίστες boolfactorTrue και boolfactorFalse την πληροφορία των condTrue και condFalse αντίστοιχα.

{p2}: προσθέτει στην λίστα boofactorTrue το label της επόμενης τετράδας , η οποία αφορά την συνθήκη , την παράγει και περνάει στην λίστα boolfactorFalse το label της επόμενης τετράδας.

<expression> ::= <optional-sign> <term> {p0}
(<add-oper> <term >{p1})*{p2}

{p0} : περνάει στην expression.place την πληροφορία του optional_sign αν αυτό είναι ίσο με «-».

{p1}: δημιουργούμε μία καινούρια μεταβλητή και παράγουμε κώδικα πρόσθεσης των δύο term.place στην μεταβλητή αυτή.
{p2} : περνάμε την πληροφορία της καινούριας μεταβλητής στην expression.place

<term> ::= <factor>{p0} (<mul-oper> <factor>{p1})*

{p0} : περνάμε στην term.place το factor.place.

{p1} : δημιουργούμε μία καινούρια μεταβλητή , και παράγουμε την εντολή πράξης μεταξύ των δύο τελευταίων factor.place , και περνάει στο term.place την πληροφορία της καινούριας μεταβλητής.

<factor> ::= constant {p0}| **(** <expression> {p1}**)** | id <idtail>{p2}

{p0}: περνάει στο factor.place την πληροφορία του ακέραιου.

{p1} : περνάει στο factor.place την πληροφορία του expression.place.

{p2} : παράγει κώδικα για κλήση της συνάρτησης με όνομα id.

<relational-oper> ::= (=| <= | >= | <> | < | >){p0}

{p0} : περνάει την πληροφορία του συμβόλου στο relationaloper.place.

<add-oper> ::= (+ | -) {p0}

{p0} : περνάει την πληροφορία του συμβόλου στο addoper.place.

<mull-oper> ::= (* | /) {p0}

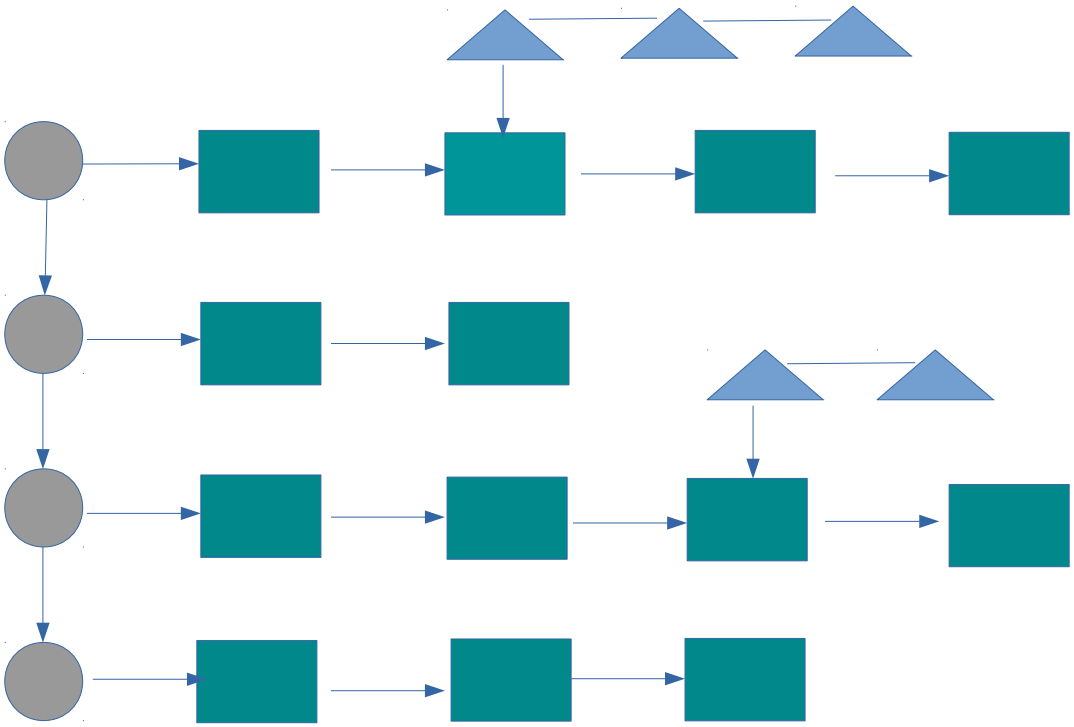
{p0} : περνάει την πληροφορία του συμβόλου στο mulloper.place.

<optional-sign> ::= e | <add-oper> {p0}

{p0} : περνάει την πληροφορία του addoper.place στο optionalsign.place.

4. Πίνακας Συμβόλων

- **Μορφή του πίνακα συμβόλων:**



όπου

scope

entitity

argument

- **Χρησιμότητα πίνακα συμβόλου:**

Ο πίνακας συμβόλων χρησιμοποιείται για να αποθηκεύει χρήσιμες πληροφορίες για διάφορες δομές του πηγαίου κώδικα που πρόκειται να μεταγλωτιστεί.

Συγκεκριμένα εισαγωγή στόν πίνακα γίνεται όταν βρεθεί δήλωση μίας νέας συνάρτησης, δήλωση μεταβλητών, για τις παραμέτρους της συνάρτησης, καθώς και για την αποθήκευση των προσωρηνών μεταβλητών που δημιουργούνται στον ενδιάμεσο κώδικα.

- **Υλοποίηση πίνακα συμβόλων:**

Συναρτήσεις που υλοποιήσαμε και η χρησιμότητά τους:

- newEntity(name,type): Αναλαμβάνει να εισάγει τις πληροφορίες που θέλει να κρατάει ο πίνακας συμβόλων στην λίστα entity. Πέρνει σαν όρισμα το όνομα της δομής, καθώς και το είδος της δομής (συνάρτηση, μεταβλήτη, είδος παραμέτρου, προσωρινή μεταβλητή) και ελέγχει με βάση το είδος(type) για το τι δομή πρόκειται, ωστε να εισάγει στην λίστα entity την λίστα με τις αντίστοιχες πληροφορίες. Συγκεκριμένα αν πρόκειται για μεταβλητητή η αντίστοιχη λίστα περιέχει το όνομα της, το offset της και την χαρακτηριστική συμβολοσείρα <var>. Τα ίδια ακριβώς στοιχεία εισάγονται και στις περιπτώσεις που πρόκειται για προσωρνή μεταβλητή ή παράμετρος με την μόνη διαφορά να έιναι στην χαρακτηριστική συμβολοσειρά. Στην περίπτωση που πρόκειται για συνάρτηση η λίστα που εισάγεται στην λίστα entity περίεχει την χαρακτηριστική συμβολοσείρα <func>, το όνομα της, την ετικέτα της πρώτης τετράδας του κώδικα της, μια κενή λίστα και το 0(είναι το framelength της το οποίο θα ενημερωθεί στην διαγραφή της). Προσθήκη νέας λίστας στην λίστα entity γίνεται όταν συναντάμε δήλωση μεταβλήτης, όταν δημιουργείται νέα προσωρινή μεταβλητή, όταν συναντάμε δήλωση νέας συνάρτησης και όταν συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης.

- newScope(): Αναλαμβάνει να μεταφέρει τις πληροφορίες που έχει κρατήσει η λίστα entity στην λίστα scope προσθέτοντας επιπλέον και το βάθος σφωλιάσματος(nestingLevel). Προσθήκη νέου scope γίνεται όταν ξεκινάμε την μετάφραση μίας νέας συνάρτησης. Συνεπώς η newScope καλείται από την newEntity όταν αυτή δεχθεί σαν όρισμα συνάρτηση. Η πληροφορίες κρατούνται σε μια λίστα scope στην οποία κάθε φορά εισάγεται μία λίστα που περιέχει σαν στοιχεία την λίστα entity και το nestingLevel. Στην συνέχεια μηδενίζει την λίστα entity, ώστε την επόμενη φορά που κληθεί να μν προσθεθούν στην λίστα scope στοιχεία που ήδη έχουν προσθεθεί.

- DeleteScope(): Όταν τελειώνουμε την μετάφραση μίας συνάρτησης καλείται η συγκεκριμένη συνάρτηση, ώστε να διαγράψει όλες τις εγγραφές στο πίνακα συμβόλων που εξαρτώνται από αυτή. Ελέγχει δύο περιπτώσεις. Αν η λίστα entity είναι άδεια πηγαίνει και κάνει μία αναζήτηση στην λίστα scope και βρίσκει την εγγραφή που πρέπει να διαγραφεί και την διαγράφει. Στην συνέχεια μειώνει κατά μία μονάδα το nestingLevel και βρίσκει το μέγιστο offset που είναι αποθηκευμένο στην scope, ωστε αφού το βρει και του προσθέσει επιπλέον τέσσερις μονάδες να αναθέσει την τιμή του στο framelength της συνάρτησης. Αντίθετα, η δεύτερη περίπτωση είναι, αν η λίστα entity δεν είναι άδεια. Στην περίπτωση αυτή κάνει ακριβώς τις ίδιες λειτουργίες με την μόνη διαφορά ότι διγράφει επιπλέον και την λίστα entity.

- newArgument(nameFunc, paritem): Η συνάρτηση αυτή καλείται όταν συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης, ώστε να εισάγει στην κενή λίστα(αναφέρθηκε στην περιγραφή της newEntity) της λίστας(αυτή που περιέχει στοιχεία σχετικά με τις συναρτήσεις) που είναι είναι αποθηκεύμενη στην λίστα entity το αντίστοιχο είδος παραμέτρου.

- searchEntity(name): Για κάθε όνομα πού βρίσκει στο κώδικα που πρόκειται να μεταγλωτιστεί καλείται η συνάρτηση αυτή, ώστε να ελέγξει αν το συγκεκριμένο όνομα έιναι αποθηκευμένο είτε σε κάποια λίστα που είναι αποθηκεύμενη στην λίστα entity, είτε σε κάποιο entity που είναι αποθηκευμένο σε κάποια λιστα, που είναι αποθηκευμένη στην λίστα scope. Αν βρεθεί το συγκεκριμένο όνομα ο μεταγλωτιστής συνεχίζει, αντίθετα αν δεν βρεθεί το πρόγραμμα τερματίζει και εμφανίζει μήνυμα λάθους: πχ. the two in line 35 is not declared

Καθολικές δομές που χρησιμοποιήσαμε και η χρησιμότητά τους:

- nestingLevel: Χρησιμοποιείται για να κρατάμε το βάθος σφωλιάσματος του πίνακα συμβόλων. Αρχικοποιείται στο 1 και αυξάνεται κάθε φορά που καλείται η newScope, ενώ μειώνεται κάθε φορά που καλείται η deleteScope. Επίσης χρησιμοποιείται και σε πολλούς ελέγχους στο κώδικά μας, ώστε να καθορίζεται σε ποιό βάθος σφωλιάσματος βρισκόμαστε και να εκτελεστούν οι αντίστοιχες λειτουργίες

- offset: Χρησιμοποιείται για να κρατάμε το offset των μεταβλητών, προσωρινών μεταβλητών και τυπικών παραμέτρων που αποθηκεύονται στον πίνακα συμβόλων. Αρχικοποιείται στην τιμή 12 και αυξάνεται κατά

4 κάθε φορά που καλείται η newEntity και η οποία δεν έχει δεχθεί σαν όρισμα συνάρτηση. Όταν κληθεί η newScope το offset ξαναγίνεται 12, ώστε οι νέες εγγραφές στον πίνακα συμβόλων να ξεκινούν με offset 12.

- firstquaad: Κρατάμε την ετικέτα της πρώτης τετράδας του κώδικα της συνάρτησης που έχει αποθηκευτεί στον πίνακα συμβόλων.

- entity: Είναι η λίστα που κρατάει τις πληροφορίες που θέλουμε κάθε φορά που καλείται η newEntity.(περιγράφηκε και παραπάνω με περισσοτερες λεπτομέριες)

- scope: Είναι η λίστα που κρατάει τις πληροφορίες που θέλουμε κάθε φορά που καλείται η newScope.(περιγράφηκε και παραπάνω με περισσοτερες λεπτομέριες)

- nameFunc: Κρατάμε το όνομα της συνάρτησης, στην οποία αντιστοιχούν οι τυπικές παράμετροι που θα εισαγχούν στον πίνακα συμβόλων ως argument. Περνιέται σαν όισμα στην newArgument, ωστε αυτή να καθορίσει με βάση αυτό το όνομα και το nestingLevel σε ποιό σημείο στην λίστα scope να αποθηκευτεί το αντίστοιχο είδος παραμέτρου.

- entertoScopeNow: Αρχικοποιείται στην τιμή False και τροποποιείται κάθε φορά που καλείται η deleteScope. Συγκεκριμένα, όπως αναφέρθηκε και παραπάνω στην deleteScope διακρίνουμε 2 περιπτώσεις. Όταν βρεθούμε στην περίπτωση που η λίστα entity είναι κενή διατειρεί την τιμή του entertoScopeNow στο False. Αντίθετα, αν βρεθούμε στην περίπτωση που δεν είναι κενή η λίστα entity κάνει το entertoScopeNow true. Η τροποποιήση αυτή συματοδοτεί ότι στην επόμενη κλήση της newEntity, αφού ενημερωθεί η λίστα entity στην συνέχεια θα την εισάγει και στην κατάλληλη λίστα που είναι ήδη αποθηκευμένη στην λίστα scope. Η συγκεκριμένη ενέργεια γίνεται, ωστε οι επόμενες εισαγωγές στον πίνακα συμβόλων μετά από μια διαγραφή να γίνουν στο σωστό βάθος σφωλιάσματος στην λίστα scope.

5. Τελικός Κώδικας

- **περιγραφή σταδίου**
  - Στόχος στη σεγκεκριμένη φάση είναι να παράγουμε το αντίστοιχο κώδικα στη συμβολική γλώσσα assembly του πηγαίου κώδικα starlet. Από κάθε εντολή ενδιάμεσου κώδικα θέλουμε να παράγουμε τις αντίστοιχες εντολές τελικού κώδικα(εντολές assembly).
  - Κύριες ενέργειες στην φάση αυτή:
    - οι μεταβλήτες απεικονίζονται στην μνήμη(στοίβα)
    - το πέρασμα παραμέτρων και η κλήση συναρτήσεων
  - Θα δημιουργήσουμε κώδικα για τον επεξεργαστή mix

- **Βοηθητικές συναρτήσεις:**
  - gnvlcode(var): Η συνάρτηση αυτή μεταφέρει στον καταχωρητή $t0 την διεύθυνση μίας μη τοπικής μεταβλητής. Απο τον πίνακα συμβόλων βρίσκει πόσα επίπεδα επάνω βρίσκεται η μη τοπική μεταβλητή και με βάση το offset της που είναι αποθηκεύμο είτε στην λίστα entity, είτε στην λίστα scope την εντοπίζει.
  - loadvr(v,r): Η συνάρτηση αυτή μεταφέρει δεδομένα στον καταχωρητή r. Η μεταφορά μπορεί να γίνει είτε από την μνήμη(στοίβα), είτε να εκχωρηθεί στο r μια σταθερά. Διακρίνουμε τις εξής περιπτώσεις: 1) αν v είναι σταθερά, 2) αν v είναι καθολική μεταβλητή –δηλαδή ανήκει στο κυρίως πρόγραμμα, 3) αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον, ή προσωρινή μεταβλητή, 4) αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον, 5) αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο από το τρέχον, 6) αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον. Για κάθε μία περίπτωση παράγουμε και την αντίστοιχη ακολουθεία εντολών assembly.
  - storevr(r,v): Η συνάρτηση αυτή μεταφέρει δεδομένα από τον καταχωρητή r στη μνήμη (μεταβλητή v). Διακρίνουμε τις εξής περιπτώσεις: 1) αν v είναι καθολική μεταβλητή –δηλαδή ανήκει στο κυρίως πρόγραμμα, 2) αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος

φωλιάσματος ίσο με το τρέχον, ή προσωρινή μεταβλητή, 3) αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον, 4) αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο από το τρέχον, 5) αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον. Για κάθε μία περίπτωση παράγουμε και την αντίστοιχη ακολουθεία εντολών assembly.

Σημείωση: Για την υλοποιήση των παραπάνω συναρτήσεων(gnvlcode,loadvr,storevr) χρειάζεται να καλούμε κάθε φορά την searchEntity που περιγράφηκε παραπάνω. Η συνάρτηση αυτή εκτός από το να ελέγχει αν ένα συγκεκριμένο όνομα είναι αποθηκευμένο στο πίνακα συμβόλων, στην περίπτωση που το βρεί αναθέτει σε κάποιες global μεταβλητές που θα περιγραφούν παρακάτω το offset, το nestingLevel και το type του ονόματος. Μετά τις αναθέσεις αυτές μπορούν στην συνέχεια να εξεταστούν οι περιπτώσεις που αναφέρθηκαν παραπάνω.

- FrCalleeAndFindCaller(funcname): Η συνάρτηση αυτή βρίσκει και αναθέτει σε global μεταβλητές το framelength και την ετικέτα της πρώτης τετράδας του κώδικα της συνάρτησης που καλείται, καθώς και το όνομα της συνάρτησης που την καλεί.

- **Σημεία που προσθέθηκε τελικός κώδικας:**

Παράγαμε τελικό κώδικα για εντολές εκχώρησης, αλμάτων, αριθμητικών πράξεων, εισόδου-εξόδου, επιστροφής τιμής συνάρτησης, κλήσης συνάρτησης, καθώς και για τις παραμέτρους συνάρτησης. Συγκεκριμένα μέσα στις συναρτήσεις:

assignmentStat()--για εντολές εκχώρησης

expression()--για αριθμητικές πράξεις με +/-

term()--για αριθμητικές πράξεις με *|/

inputStat()--για εντολές εισόδου

printStat()--για εντολές εξόδου

returnStat()--για την επιστροφή τιμής συνάρτησης

idtail()--για την κλήση συνάρτησης

boolfactor()--για εντολές αλμάτων

actualpariterm()--για τις παραμέτρους συνάρτησης

και κάτω ακριβώς από τις αντίστοιχες genquad() που παράγουν τις αντίστοιχες εντολές σε μορφή ενδιάμεσου κώδικα. Εκτός από τις παραπάνω συναρτήσεις επίσης προσθέσαμε τελικό κώδικα και μέσα σε συναρτήσεις, στις οποιές υπήρχε κάποιο genquad() που παρήγαγε ενδιάμεσο κώδικα είτε για μια εντολή εκχώρησης, είτε για μια εντολή άλματος.

- **Τελικός κώδικας ανά εντολή**:
  - Εντολές αλματών(πχ μία relop x,y,z)

Για την συγκεκριμένη εντολή χριάστηκε να μεταφέρουμε σε ένα καταχωρητή την τιμή του x, σε ένα άλλο καταχωρητή την τιμή του y και στην συνέχεια να συγκρίνουμε τις δύο τιμές και αναλόγως το relop να μεταβούμε στο κατάλληλο label(z ή ένα άλλο).

  - Εντολές εκχώρησης(πχ :=, x, _, z)

Για την συγκεκριμένη εντολή χρειάστηκε να μεταφέρουμε σε ένα καταχωρητή την τιμή x και στην συνέχεια να μεταφέρουμε τον καταχωρητή αυτό στην μνήμη(μεταβλητή z).

  - Εντολές αριθμητικών πράξεων(πχ op x,y,z)

Για το συγκεκριμένο είδος εντολών χρείαστηκε να μεταφέρουμε σε ένα καταχωρητή την τιμή x, σε ένα άλλο την τιμή του y. Στην συνέχεια εκτελεί την πράξη, βάζει το αποτέλεσμα σε ένα καταχωρητή και μεταφέρουμε τον καταχωρητή αυτό στην μνήμη(μεταβλητή z).

  - Επιστροφή τιμής συνάρτησης(πχ retv _,_,x)

Για το συγκεκριμένο είδος εντολής χρειάστηκε να αποθηκεύεσουμε τον x στη διεύθυνση που είναι αποθηκευμένη στην 3η θέση του εγγραφήματοςδραστηριοποίησης.

- Παράμετροι συνάρτησης

  Πριν από την πρώτη παράμετρο, τοποθετούμε τον $fp να δείχνει στην στοίβα της συνάρτησης που θα δημιουργηθεί. Στην συνέχεια ανάλογα το είδος της παραμέτρου παράγονται και οι αντίστοιχες εντολές assembly. Συγκεκριμένα για πέρασμα με τιμή μεταφέρει την παράμετρος σε ένα καταχωρητή και στην συνέχεια αποθηκεύει το καταχωρητή αύτο καταλληλες θέσεις πάνω από εκεί που δίχνει ο $fp.

  Για πέρασμα με αναφορά διακρίνουμε τις εξής περιπτώσεις: 1) αν η καλούσα συνάρτηση και η παράμετρος έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή τότε μεταφέρουμε σε ένα καταχωρητή την διεύθυνση της παραμέτρου και στην συνέχεια αποθηκεύουμε τον καταχωρητή αυτό κατάλληλες θέσεις πάνω από εκεί που δίχνει ο $fp. 2) αν η καλούσα συνάρτηση και η παράμετρος έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με αναφορά τότε μεταφέρουμε σε ένα καταχωρητή την τιμή της παραμέτρου και στην συνέχεια αποθηκεύουμε τον καταχωρητή αυτό κατάλληλες θέσεις πάνω από εκεί που δίχνει ο $fp. 3) αν η καλούσα συνάρτηση και η παράμετρος έχουν διαφορετικό βάθος φωλιάσματος, η παράμετρος είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή τότε αφου προκείται για μη τοπική μεταβλητή, καλείται η gnvlcode, ώστε να μεταφέρει σε ένα καταχωρητή την διεύθυνσή της και στην συνέχεια αποθηκεύουμε τον καταχωρητή αυτό κατάλληλες θέσεις πάνω από εκεί που δίχνει ο $fp. 4) αν η καλούσα συνάρτηση και η παράμετρος έχουν διαφορετικό βάθος φωλιάσματος, η παράμετρος είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με αναφορά τότε καλέιται η gnvlcode ώστε να μεταφέρουμε σε ένα καταχωρητή την διεύθυνση της, στην συνέχεια μεταφέρουμε σε ένα καταχωρητή την τιμή που είναι αποθηκευμένη σε αυτή την διεύθυνση και τέλος αποθηκεύουμε τον καταχωρητή αυτό κατάλληλες θέσεις πάνω από εκεί που δίχνει ο $fp.

  Για το πέρασμα με αντιγραφή χρειάστηκε να μεταφέρουμε την παράμετρο σε ένα καταχωρητή, στην συνέχεια να αποθηκεύσουμε το καταχωρητή αύτο καταλληλες θέσεις πάνω από εκεί που δίχνει ο $fp. Στην συνέχεια για να πάρουμε την αλαγμένη τιμή της παραμέτρου, μετά το τέλος της συνάρτησης πηγαίνουμε κατάλληλες θέσεις πάνω απο εκεί που δίχνει ο $sp και μεταφέρουμε σε ένα καταχωρητή την

αντίστοιχη τιμή που είναι αποθηκευμένη σε αυτή την διεύθυνση. Τελος καλούμε την storevr ώστε να αποθηκεύσει τον καταχωρητή αυτό στην παράμετρό μας.

- Αποθήκευση στην προσωρινή μεταβλητή το αποτέλεσμα της συνάρτησης(par,x,RET)

  Για το συγκεκρίμενο είδος εντολής ενδιάμεσου κώδικα χρειάστηκε να γεμίσουμε το 3ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης με τη διεύθυνση της προσωρινής μεταβλητής στην οποία θα επιστραφεί το αποτέλεσμα.

- Κλήση συνάρτησης(call,_,_,f)

  Για το συγκεκρίμενο είδος εντολής ενδιάμεσου κώδικα χρειάστηκε αρχικά να γεμίσουμε το 2ο πεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης με την διεύθυνση του εγγραφήματος δραστηριοποίησης του γονέα της, ώστε η κληθείσα να γνωρίζει που να κοιτάξει αν χρειαστεί να προσπελάσει μία μεταβλητή την οποία έχει δικαίωμα να προσπελάσει, αλλά δεν της ανήκει. Στη συνέχεια χρειάστηκε να μεταφέρουμε τον δείκτη στοίβας στην κληθείσα, να καλέσουμε τη συνάρτηση και τέλος όταν επιστρέφουμε να παίρνουμε πίσω τον δείκτη στοίβας στην καλούσα. Στην αρχή κάθε συνάρτησης χρειάστηκε να αποθηκεύουμε στην πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής της την οποία έχει τοποθετήσει στον $ra η jal. Στο τέλος κάθε συνάρτησης χρειάστηκε να κάνουμε το αντίστροφο, παίρνουμε από την πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής της συνάρτησης και την βάζουμε πάλι στον $ra. Μέσω του $ra επιστρέφουμε στην καλουσα.