Μεταφραστές
Η γλώσσα Ciscal

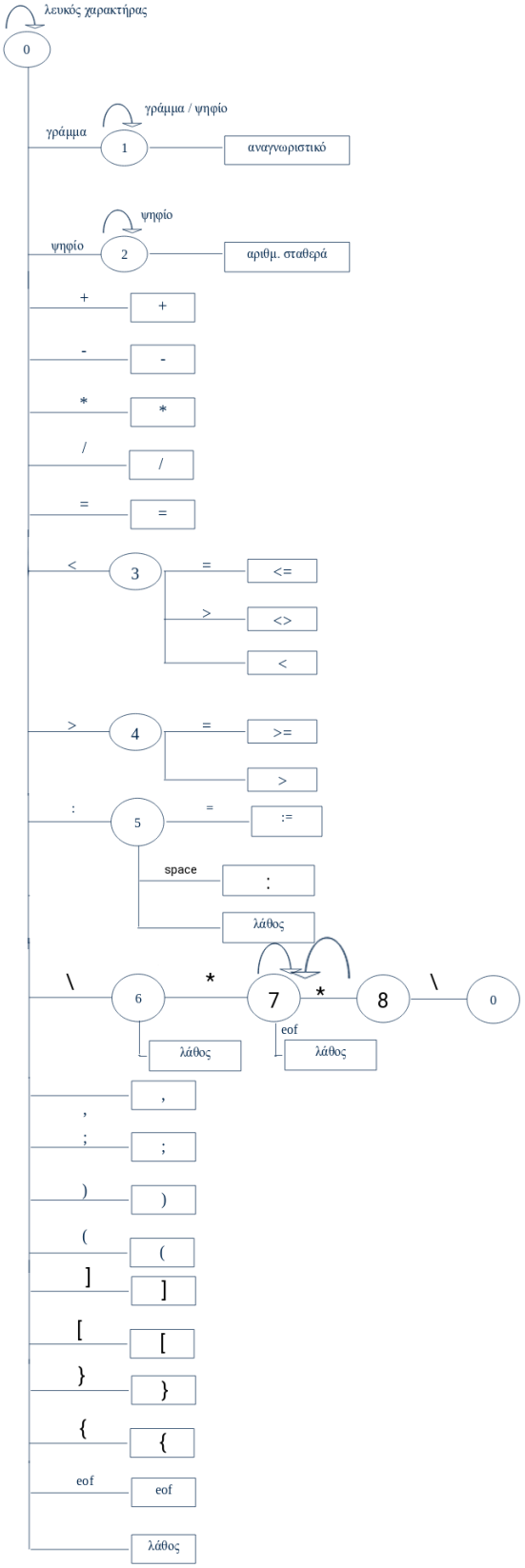
Κλεφτάκης Ηλίας (2461)
Νάκας Κωνσταντίνος Παναγιώτης (2501)
E-mail: {cse32461,cse32501}@cse.uoi.gr

15 Μαΐου 2017

1 Λεκτική ανάλυση

Η λειτουργία του λεκτικού αναλυτή φαίνεται στο διάγραμμα καταστάσεων του Σχήματος 1. Η λεκτική ανάλυση πραγματοποιείται στο αρχείο `lexical.py`. Η βοηθητική συνάρτηση `initialize()` ανοίγει το αρχείο που της ζητείται και το αποθηκεύει σε μία λίστα όπου το κάθε στοιχείο της είναι μία γραμμή του αρχείου. Η συνάρτηση `getNextCharacter()` επιστρέφει τον επόμενο χαρακτήρα του αρχείου και ενημερώνει τις παγκόσμιες μεταβλητές `currentLine` και `currentColumn` για τη θέση του χαρακτήρα μέσα στο αρχείο. Η συνάρτηση `ungetCharacter()` αναγκάζει τη συνάρτηση `getNextCharacter()` να επιστρέψει ξανά τον ίδιο χαρακτήρα στην επόμενη κλήση της.

Η κύρια συνάρτηση που επιστρέφει το επόμενο λεκτικό στοιχείο είναι η `getNextToken()`. Υλοποιεί ένα αυτόματο πεπερασμένων καταστάσεων στο οποίο η μεταβλητή `state` καθορίζει την τρέχουσα κατάσταση. Ανάλογα με την τρέχουσα κατάσταση και το χαρακτήρα που διαβάστηκε, καθορίζεται η επόμενη κατάσταση. Έτσι αναγνωρίζει αριθμητικές σταθερές, αναγνωριστικά και διάφορα ειδικά σύμβολα της γλώσσας Ciscal. Επίσης φροντίζει να αναγνωρίζει τα σχόλια που είναι έγκυρα ορισμένα, αλλιώς εμφανίζει ειδικό μήνυμα λάθους. Επιπλέον, σε κάθε περίπτωση, αν ο χαρακτήρας δεν είναι επιθυμητός στην τρέχουσα κατάσταση, εμφανίζεται μήνυμα λάθους. Στο τέλος επιστρέφει ένα αντικείμενο `Token`. Αυτό έχει τρία πεδία: το αναγνωριστικό που σχηματίστηκε, τον τύπο του αναγνωριστικού και πληροφορίες σφάλματος, όπως αυτές ορίζονται παρακάτω.


λευκός χαρακτήρας
0
γράμμα / ψηφίο
γράμμα
1
αναγνωριστικό
ψηφίο
ψηφίο
2
αριθμ. σταθερά
+
+
-
-
*
*
/
/
=
=
<
3
=
<=
>
<>
<
>
4
=
>=
>
:
5
=
:=
space
:
λάθος
\
6
*
7
*
8
\
0
λάθος
eof
λάθος
,
,
;
;
)
)
(
(
]
]
[
[
}
}
{
{
eof
eof
λάθος

Σχήμα 1

2 Συντακτική ανάλυση

Η συντακτική ανάλυση πραγματοποιείται στο αρχείο `syntax.py`. Για κάθε κατηγόρημα της γραμματικής της γλώσσας Ciscal υλοποιείται μία αντίστοιχη συνάρτηση. Αρχικά πρέπει να κληθεί η συνάρτηση `program()`, που είναι και η αρχική. Αυτή με τη σειρά της και ανάλογα με το πρόγραμμα προς μετάφραση, φροντίζει να καλέσει τις απαραίτητες συναρτήσεις. Αυτή η διαδικασία συνεχίζεται μέχρι να φτάσουμε στα δομικά στοιχεία της γλώσσας (όπως αριθμητικές σταθερές, ή διάφορα ειδικά σύμβολα). Στη συνέχεια οι συναρτήσεις επιστρέφουν προς τα πάνω, έως όπου να επιστρέψουμε στην αρχική συνάρτηση `program()`, όπου αναμένουμε να βρούμε το τέλος του αρχείου (End-Of-File). Πριν την κλήση κάθε συνάρτησης, είναι υποχρεωτικό να έχουμε τοποθετήσει στην παγκόσμια μεταβλητή token το επόμενο λεκτικό στοιχείο. Επίσης, η ίδια δουλειά γίνεται από κάθε συνάρτηση αμέσως πριν την επιστροφή της (ώστε το επόμενο λεκτικό στοιχείο να είναι έτοιμο για την επόμενη συνάρτηση). Τέλος, σε όλη τη διάρκεια της συντακτικής ανάλυσης ελέγχουμε εξονυχιστικά για όλα τα συντακτικά που ενδέχεται να εντοπιστούν. Ενδεικτικά, κάποια από αυτά είναι: ελλιπής παρενθέσεις, άγκιστρα και αγκύλες, λάθος δεσμευμένες λέξεις της γλώσσας (π.χ. if, inout, while κ.τ.λ) και χρήση αναγνωριστικών σε σημεία που δε χρειάζονται.

Πιο αναλυτικά, για κάθε συνάρτηση έχουμε:

- `program()`: Είναι η συνάρτηση που καλείται όταν βρίσκουμε τη λέξη program κατά τη μεταγλώττιση.
- `block(blockName = None)`: Παίρνει σαν προαιρετικό όρισμα το όνομα του block και φροντίζει να μεταφράσει κάθε block του κώδικα καλώντας τις κατάλληλες συναρτήσεις.
- `declarations()`: Διατρέχει τη λίστα δηλώσεων στο αντίστοιχο block.
- `varlist()`: Διατρέχει μία–μία τις μεταβλητές.
- `subprograms()`: Καλεί τη συνάρτηση `func()` για κάθε υποπρόγραμμα.
- `func(isFunction)`: Καλεί τη συνάρτηση `funcbody()` για συναρτήσεις και διαδικασίες.
- `funcbody(funcName)`: Διατρέχει το κύριο μέρος των συναρτήσεων.
- `formalpars()`: Διατρέχει τη λίστα των τυπικών παραμέτρων.
- `formalparlist()`: Για κάθε παράμετρο που συναντά, καλεί τη `formalparitem()`.
- `formalparitem(allowNothing)`: Διαβάζει μία τυπική παράμετρο.
- `sequence()`: Καλεί τη `statement()` για κάθε εντολή που συναντάει.
- `brack_or_stat()`: Διαβάζει μία εντολή που μπορεί να περικλείεται σε άγκιστρα ή όχι.
- `statement()`: Αποφασίζει για τι είδος εντολής πρόκειται.
- `assignment_stat(identifier)`: Διατρέχει μία εντολή ανάθεσης.

- `if_stat()`: Διατρέχει το πρώτο μέρος μίας εντολής if.
- `elsepart()`: Διατρέχει το δεύτερο μέρος μίας εντολής if.
- `while_stat()`: Διατρέχει μια εντολή while.
- `select_stat()`: Διατρέχει μια εντολή select.
- `do_while_stat()`: Διατρέχει μία εντολή do–while.
- `return_stat()`: Διατρέχει μία εντολή return.
- `print_stat()`: Διατρέχει μία εντολή print.
- `call_stat()`: Διατρέχει μία εντολή call.
- `actualpars(functionItem)`: Διατρέχει τη λίστα των πραγματικών παραμέτρων.
- `actualparlist(functionItem)`: Για κάθε παράμετρο που συναντά καλεί την `actual_par_item`.
- `actualparitem(allowNothing, functionItem, argumentId)`: Διαβάζει μια πραγματική παράμετρο.
- `condition()`: Διατρέχει μια συνθήκη με or.
- `boolterm()`: Διατρέχει μια συνθήκη με and.
- `boolfactor()`: Διατρέχει μία συνθήκη με not, ή μια συνθήκη που βρίσκεται μέσα σε άγκιστρα, ή μία συνθήκη που αποτελείται από δύο εκφράσεις και ένα σχεσιακό τελεστή.
- `expression()`: Διατρέχει μία αριθμητική έκφραση που αποτελείται από προσθέσεις και αφαιρέσεις.
- `term()`: Διατρέχει μία αριθμητική έκφραση που αποτελείται από πολλαπλασιασμούς και διαιρέσεις.
- `factor()`: Διατρέχει μία αριθμητική σταθερά, ή μία έκφραση σε παρενθέσεις, ή μία κλήση συνάρτησης.
- `idtail(identifier)`: Διατρέχει πραγματικές παραμέτρους, εφόσον υπάρχουν.
- `relational_oper()`: Διατρέχει όλους τους σχεσιακούς τελεστές.
- `add_oper()`: Διατρέχει το + και το -.
- `mul_oper()`: Διατρέχει το * και το /.
- `optional_sign()`: Διατρέχει προσημασμένους αριθμούς.

Επίσης, χρησιμοποιούμε τις ακόλουθες βοηθητικές συναρτήσεις:

- `getNextToken()`: Ζητά το επόμενο λεκτικό στοιχείο από το λεκτικό αναλυτή.

- `checkUndeclaredIdentifier()`: Ελέγχει για μη δηλωμένα αναγνωριστικά.
- `checkAlreadyDeclaredIdentifier()`: Ελέγχει για ήδη δηλωμένα αναγνωριστικά στο τρέχον βάθος φωλιάσματος.

3 Ενδιάμεσος κώδικας

Οι βοηθητικές συναρτήσεις που χρειαστήκαμε για τον ενδιάμεσο κώδικα βρίσκονται στο αρχείο `intermediate.py`.

1. Η `nextQuad()` μας δίνει τον αριθμό της επόμενης τετράδας που θα παραχθεί από το σημείο που βρισκόμαστε.
2. Η `newTemp()` μας επιστρέφει την επόμενη διαθέσιμη προσωρινή μεταβλητή που θα παραχθεί και δεν χρησιμοποιείται ήδη. Αυτό επιτυγχάνεται με χρήση παγκόσμιας μεταβλητής που λειτουργεί ως μετρητής.
3. Η `emptyList()` δημιουργεί και επιστρέφει μία κενή λίστα ετικετών τετράδων. Χρησιμοποιείται όταν δεν γνωρίζουμε ακόμη τον προορισμό ενός άλματος.
4. Η `makeList(x)` δημιουργεί και επιστρέφει μία λίστα ετικετών τετράδων με μοναδικό στοιχείο το x. Χρησιμοποιείται για τον ίδιο λόγο με την `emptyList()`.
5. Η `merge(list1, list2)` συνενώνει τις λίστες `list1` και `list2` σε μία λίστα και την επιστρέφει.
6. Η `backPatch(myList, z)` συμπληρώνει το τελευταίο στοιχείο κάθε τετράδας στη `myList` με `z`.
7. Επίσης έχουμε συναρτήσεις που γράφουν τον ενδιάμεσο κώδικα που παράγεται σε ένα αρχείο καθώς και τον ισοδύναμο κώδικα σε C σε ένα διαφορετικό αρχείο. Ο κώδικας C μας βοηθάει στον εύκολο έλεγχο της λειτουργίας του κώδικα.

Κατά τη διάρκεια της συντακτικής ανάλυσης παράγεται ο ενδιάμεσος κώδικας καλώντας τις παραπάνω βοηθητικές συναρτήσεις. Πιο συγκεκριμένα, για τις παρακάτω περιπτώσεις έχουμε:

1. Στην αρχή κάθε μποκ παράγουμε:
   `genquad(" begin_block",name,"_","_")`
2. Στο τέλος κάθε μπλοκ παράγουμε:
   `genquad("end_block",name,"_","_")`
   Αν πρόκειται για το τέλος του προγράμματος, παράγουμε επίσης:
   `genquad("halt","_","_","_")`
3. Στην περίπτωση $E -> T^1(+T^2\{P_1\})^*\{P_2\}$ παράγουμε
   $\{P_1\} : w = newTemp()$
   $genquad(\text{“}+\text{”}, T^1.place, T^2.place, w)$
   $T^1.place = w$
   $\{P_2\} : E.place = T^1.place$

4. Στην περίπτωση $T->F^1(xF^2\{P_1\})^*\{P_2\}$ παράγουμε
   $\{P_1\} : w = newTemp()$
   $genquad(\text{“}x\text{”}, F^1.place, F^2.place, w)$
   $F^1.place = w$
   $\{P_2\} : T.place = F^1.place$

5. Στην περίπτωση $F->(E)\{P_1\}$ παράγουμε
   $\{P_1\} : F.place = E.place$

6. Στην περίπτωση $F->id\{P_1\}$ παράγουμε
   $\{P_1\} : F.place = id.place$

7. Στην περίπτωση $B->Q^1\{P_1\}(or\{P_2\}Q^2\{P3\})^*$ παράγουμε
   $\{P_1\} : B.true = Q^1.true$
   $B.false = Q^1.false$
   $\{P_2\} : backpatch(B.false, nextquad())$
   $\{P_3\} : B.true = merge(B.true, Q^2.true)$
   $B.false = Q^2.false$

8. Στην περίπτωση $Q->R^1\{P_1\}(and\{P_2\}R^2\{P3\})^*$ παράγουμε
   $\{P_1\} : Q.true = R^1.true$
   $Q.false = R^1.false$
   $\{P_2\} : backpatch(Q.true, nextquad())$
   $\{P_3\} : Q.false = merge(Q.false, R^2.false)$
   $Q.true = R^2.true$

9. Στην περίπτωση $R->(B)\{P_1\}$ παράγουμε
   $\{P_1\} : R.true = B.true$
   $R.false = B.false$

10. Στην περίπτωση $R->E^1 relop E^2\{P_1\}$ παράγουμε
    $\{P_1\} : R.true = makelist(nextquad())$
    $genquad(relop, E^1.place, E^2.place, \text{“\_”})$
    $R.false = makelist(nextquad())$
    $genquad(\text{“}jump\text{”}, \text{“\_”}, \text{“\_”}, \text{“\_”})$

11. Στην περίπτωση που έχουμε κλήση διαδικασίας, για παράδειγμα: $call$ $assign\_v(ina, inoutb)$ παράγουμε
    $par, a, CV, \text{”\_”}$
    $par, b, REF, \text{”\_”}$
    $call, assign\_v, \text{”\_”}, \text{”\_”}$

12. Στην περίπτωση που έχουμε κλήση συνάρτησης, για παράδειγμα: $error =$ $assign\_v(ina, inoutb)$ παράγουμε
    $par, a, CV, \text{”\_”}$
    $par, b, REF, \text{”\_”}$
    $w = newTemp()$
    $par, w, RET, \text{”\_”}$
    $call, assign\_v, \text{”\_”}, \text{”\_”}$

13. Στην περίπτωση $S->return(E)\{P_1\}$ παράγουμε
    $\{P_1\} : genquad(\text{“}retv\text{”}, E.place, \text{”\_”}, \text{”\_”})$

14. Στην περίπτωση $S->id := E\{P_1\}$ παράγουμε
    $\{P_1\} : genquad(\text{“ := “}, E.place, \text{”\_”}, id)$

15. Στην περίπτωση $S->while\{P_1\}Bdo\{P_2\}S^1\{P_3\}$ παράγουμε
    $\{P_1\} : Bquad := nextquad()$
    $\{P_2\} : backpatch(B.true, nextquad())$
    $\{P_3\} : genquad(\text{“}jump\text{”}, \text{”\_”}, \text{”\_”}, Bquad)$
    $backpatch(B.false, nextquad())$

16. Στην περίπτωση $S->repeat\{P_1\}S^1\ until(cond)\{P_2\}$ παράγουμε
    $\{P_1\} : sQuad := nextquad()$
    $\{P_2\} : backpatch(cond.False, sQuad)$
    $backpatch(cond.True, nextquad())$

17. Στην περίπτωση $S->ifBthen\{P_1\}S^1\{P_2\}TAIL\{P_3\}$,'οπου: $TAIL->elseS^2 \mid TAIL-> e$ παράγουμε
    $\{P_1\} : backpatch(B.true, nextquad())$
    $\{P_2\} : ifList = makelist(nextquad())$
    $genquad(\text{“}jump\text{”}, \text{”\_”}, \text{”\_”}, \text{”\_”})$
    $backpatch(B.false, nextquad())$
    $\{P_3\} : backpatch(ifList, nextquad())$

18. Στην περίπτωση $S->input(id)\{P_1\}$ παράγουμε
    $\{P_1\} : genquad(\text{“}inp\text{”}, id.place, \text{”\_”}, \text{”\_”})$

19. Στην περίπτωση $S->print(E)\{P_1\}$ παράγουμε
    $\{P_1\} : genquad(\text{“}out\text{”}, E.place, \text{”\_”}, \text{”\_”})$

4 Πίνακας Συμβόλων

Για τον πίνακα συμβόλων έχουμε υλοποιήσει τρεις κλάσεις καθώς και τις απαραίτητες βοηθητικές συναρτήσεις μέσα στο αρχείο `symbolTable.py`.

Στην κλάση EntityItem δημιουργούμε το κατάλληλο αντικείμενο ανάλογα με τον τύπο της οντότητας.

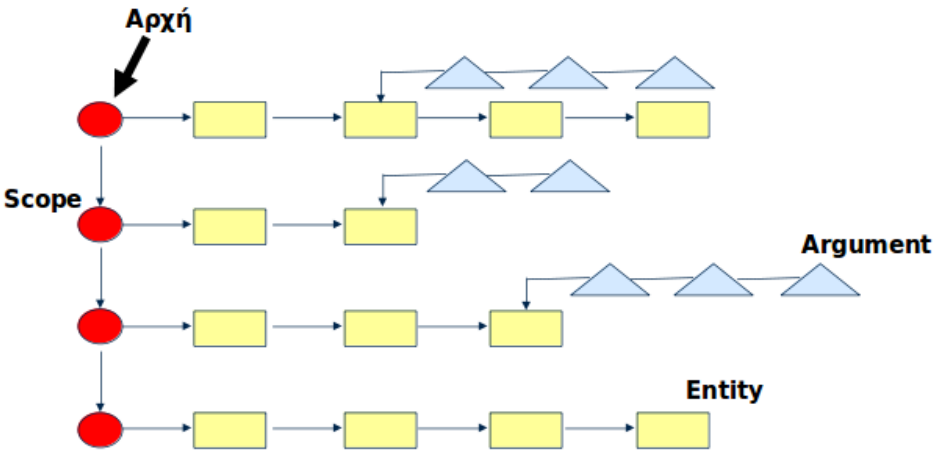
1. Αν είναι *παράμετρος* κρατάμε τις πληροφορίες **`itemType, itemName, isIn`** και **`offset`**.

2. Αν είναι *συνάρτηση*, κρατάμε τις πληροφορίες **`itemType, itemName, isFunction, startQuad, argumentList`** και **`frameLength`**.

3. Αν είναι *μεταβλητή*, κρατάμε τις πληροφορίες **`itemType, itemName`** και **`offset`**.

Στην κλάση **`ScopeItem`** κρατάμε μία λίστα από **`Entities`** και έχουμε 2 βοηθητικές μεθόδους ώστε να προσθέτουμε **`Entity`** ή να προσθέτουμε **`Argument`**.

Στην κλάση **`ArgumentItem`** κρατάμε αν η παράμετρος περνιέται με τιμή ή με αναφορά χρησιμοποιώντας μία boolean μεταβλητή.



Σχήμα 2: Ο πίνακας συμβόλων εν ώρα εκτέλεσης

Στο Σχήμα 2 φαίνεται διαγραμματικά ένα παράδειγμα του πίνακα συμβόλων εν ώρα εκτέλεσης. Οι κόκκινοι κύκλοι αναπαριστούν αντικείμενα τύπου `ScopeItem`, τα κίτρινα ορθογώνια αναπαριστούν αντικείμενα τύπου `EntityItem`, ενώ τα γαλάζια τρίγωνα αναπαριστούν αντικείμενα τύπου `ArgumentItem`.

Τέλος, παρέχουμε βοηθητικές συναρτήσεις που προσθέτουν, αφαιρούν και τροποποιούν αντικείμενα των παραπάνω κλάσεων και αναζητούν πληροφορίες μέσα στον πίνακα συμβόλων για δοθέντα elements.

5 Τελικός κώδικας

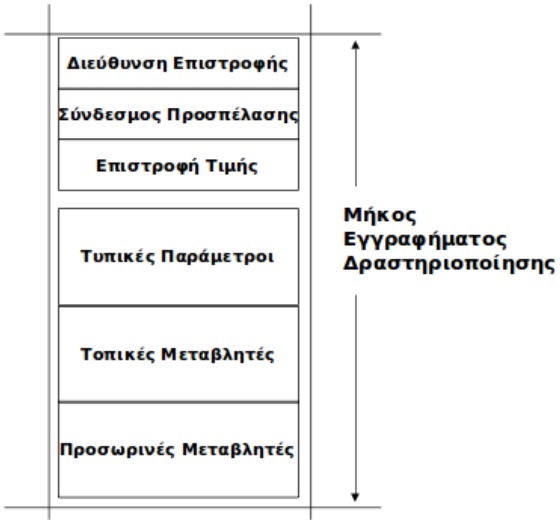
Ο τελικός κώδικας και οι βοηθητικές συναρτήσεις που χρησιμοποιεί υλοποιούνται στο αρχείο `final.py`.

Η συνάρτηση `gnvlcode()` μεταφέρει στον καταχωρητή `$t0` τη διεύθυνση μιας μη τοπικής μεταβλητής την οποία παίρνει ως παράμετρο. Για να το επιτύχει αυτό αρχικά βρίσκει τη μεταβλητή στον πίνακα συμβόλων. Στη συνέχεια, παράγουμε εντολές τελικού κώδικα ώστε να βρεθούμε στη στοίβα[1] του γονέα και μετά, για όσες φορές χρειαστεί μέχρι να φτάσουμε στη στοίβα του προγόνου που περιέχει τη ζητούμενη μεταβλητή. Τέλος, με χρήση του offset της μεταβλητής βρίσκουμε ακριβώς το σημείο που βρίσκεται η μεταβλητή.

Η συνάρτηση `loadvr(v, r)` μεταφέρει τη διεύθυνση μιας μεταβλητής v στον καταχωρητή r. Για να το καταφέρουμε αυτό, διακρίνουμε τις εξής περιπτώσεις:

1. Αν είναι σταθερά, απλά φορτώνουμε την τιμή της στον καταχωρητή.
2. Αν είναι καθολική μεταβλητή, φορτώνουμε την τιμή της χρησιμοποιώντας το offset της, αλλά και τον καταχωρητή `$s0`, στον οποίο έχουμε αποθηκεύσει ένα δείκτη στη στοίβα της main.

[1] μια εικόνα της στοίβας για μία συνάρτηση σε χρόνο εκτέλεσης φαίνεται στο Σχήμα 3



Σχήμα 3: Η στοίβα μιας συνάρτησης σε χρόνο εκτέλεσης

3. Αν είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον, ή προσωρινή μεταβλητή, τότε τη φορτώνουμε χρησιμοποιώντας το offset της από την τρέχουσα στοίβα.

4. Αν είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον, τότε αρχικά φορτώνουμε τη διεύθυνσή της χρησιμοποιώντας το offset της από την τρέχουσα στοίβα και στη συνέχεια φορτώνουμε την τιμή στην οποία δείχνει αυτή η διεύθυνση.

5. Αν είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο από το τρέχον, τότε αρχικά φορτώνουμε τη διεύθυνσή της χρησιμοποιώντας τη βοηθητική συνάρτηση `gnlvcode()` και στη συνέχεια φορτώνουμε την τιμή στην οποία δείχνει αυτή η διεύθυνση.

6. Αν είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον, τότε αρχικά φορτώνουμε τη διεύθυνσή της χρησιμοποιώντας τη βοηθητική συνάρτηση `gnlvcode()`, και στη συνέχεια φορτώνουμε την τιμή στην οποία δείχνει αυτή η διεύθυνση στον καταχωρητή r.

Η συνάρτηση `storerv(r, v)` μεταφέρει τη διεύθυνση μιας μεταβλητής v στον καταχωρητή r. Για να το καταφέρουμε αυτό, διακρίνουμε τις εξής περιπτώσεις:

1. Αν είναι σταθερά, απλά αποθηκεύουμε την τιμή του στη μεταβλητή.

2. Αν είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον, ή προσωρινή μεταβλητή, τότε αποθηκεύουμε την τιμή του χρησιμοποιώντας το offset του από την τρέχουσα στοίβα.

3. Αν είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον, τότε αρχικά φορτώνουμε τη διεύθυνσή της χρησιμοποιώντας το offset της από την τρέχουσα στοίβα και στη συνέχεια αποθηκεύουμε την τιμή του στη διεύθυνση που δείχνει.

4. Αν είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο από το τρέχον, τότε αρχικά φορτώνουμε τη διεύθυνσή της χρησιμοποιώντας τη βοηθητική συνάρτηση `gnlvcode()` και στη συνέχεια αποθηκεύουμε την τιμή του στη διεύθυνση που δείχνει.

5. Αν είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον, τότε αρχικά φορτώνουμε τη διεύθυνσή της χρησιμοποιώντας τη βοηθητική συνάρτηση `gnlvcode()`, και στη συνέχεια φορτώνουμε τη διεύθυνση που είναι αποθηκευμένη η τιμή που χρειαζόμαστε και στη συνέχεια αποθηκεύουμε τα περιεχόμενα του καταχωρητή r σε αυτή την τιμή.

Η συνάρτηση `generateFinalCode()` παράγει τελικό κώδικα για τις διάφορες εντολές ενδιάμεσου κώδικα που μπορεί να συναντήσει. Διακρίνουμε τις παρακάτω περιπτώσεις:

1. `jump, "_", "_", label`
   Παράγουμε την εντολή
   `j label`

2. `relop (?), x,y,z`
   Παράγουμε τις εντολές
   `loadvr(x,1)`
   `loadvr(y,2)`
   `branch(?),$t1,$t2,z`, όπου branch(?) είναι μία από τις εντολές `beq, bne, bgt, blt, bge, ble`

3. `:=, x, "_", z`
   Παράγουμε τις εντολές
   `loadvr(x,1)`
   `storerv(1,z)`

4. `op x,y,z`
   Παράγουμε τις εντολές
   `loadvr(x,1)`
   `loadvr(y,2)`
   `op $t1,$t1,$t2`
   `storerv(1,z)`, όπου op είναι μία από τις εντολές add,sub,mul,div

5. `out "_", "_", x`
   Παράγουμε τις εντολές
   `li $v0,1`
   `li $a0, x`
   `syscall`

6. `in "_", "_", x`
   Παράγουμε τις εντολές

```
li $v0,5
syscall
```

7. `retv "_", "_", x`
   Παράγουμε τις εντολές
   `loadvr(x,1)`
   `lw $t0, -8($sp)`
   `sw $t1,($t0)`

8. Πριν από την πρώτη παράμετρο, τοποθετούμε τον `$fp` να δείχνει στην στοίβα της συνάρτησης που θα δημιουργηθεί `add $fp,$sp,framelength`. Στη συνέχεια διακρίνουμε υποπεριπτώσεις:

   (α′) `par,x,CV , _`
   Παράγουμε τις εντολές
   `loadvr(x,0)`
   `sw $t0, -(12+4i)($fp)`, όπου i ο αύξων αριθμός της παραμέτρου

   (β′) `par,x,REF , _`
   Διακρίνουμε τις υποπεριπτώσεις:

   - Αν η καλούσα συνάρτηση και η μεταβλητή x έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή, παράγουμε τις εντολές
     `add $t0,$sp, -offset`
     `sw $t0, -(12+4i)($fp)`
   - Αν η καλούσα συνάρτηση και η μεταβλητή x έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με αναφορά παράγουμε τις εντολές
     `lw $t0, -offset($sp)`
     `sw $t0, -(12+4i)($fp)`
   - Αν η καλούσα συνάρτηση και η μεταβλητή x έχουν διαφορετικό βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή, παράγουμε τις εντολές
     `gnlvcode(x)`
     `sw $t0, -(12+4i)($fp)`
   - Αν η καλούσα συνάρτηση και η μεταβλητή x έχουν διαφορετικό βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με αναφορά, παράγουμε τις εντολές
     `gnlvcode(x)`
     `lw $t0,($t0)`
     `sw $t0, -(12+4i)($fp)`

   (γ′) `par,x,RET , _`
   Παράγουμε τις εντολές
   `add $t0,$sp, -offset`
   `sw $t0, -8($fp)`

9. `call, _, _, f`
   Διακρίνουμε τις υποπεριπτώσεις:

- Αν καλούσα και κληθείσα έχουν το ίδιο βάθος φωλιάσματος, τότε έχουν τον ίδιο γονέα και παράγουμε τις εντολές
  `lw $t0, -4($sp)`
  `sw $t0, -4($fp)`
- Αν καλούσα και κληθείσα έχουν διαφορετικό βάθος φωλιάσματος, τότε η καλούσα είναι ο γονέας της κληθείσας και παράγουμε τις εντολές
  `sw $sp, -4($fp)`

Στη συνέχεια μεταφέρουμε τον δείκτη στοίβας στην κληθείσα με την εντολή
`add $sp,$sp,framelength`
Έπειτα καλούμε τη συνάρτηση f με την εντολή `jal f` και όταν επιστρέψουμε παίρνουμε πίσω τον δείκτη στοίβας στην καλούσα συνάρτηση με την εντολή
`add $sp,$sp ,-framelength`

Μέσα στην κληθείσα στην αρχή κάθε συνάρτησης αποθηκεύουμε στην πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής της την οποία έχει τοποθετήσει στον $ra η jal με την εντολή
`sw $ra ,($sp)`
Στο τέλος κάθε συνάρτησης 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 καλούσα με τις εντολές
`lw $ra ,($sp)`
`jr $ra`

10. Το κυρίως πρόγραμμα δεν είναι το πρώτο πράγμα που μεταφράζεται, οπότε στην αρχή του προγράμματος χρειάζεται ένα άλμα που να οδηγεί στην πρώτη ετικέτα του κυρίως προγράμματος με την εντολή
    `j Lmain`
    Στη συνέχεια πρέπει να κατεβάσουμε τον $sp κατά framelength της main με την εντολή
    `add $sp,$sp,framelength`
    και να σημειώσουμε στον $s0 το εγγράφημα δραστηριοποίησης της main ώστε να έχουμε εύκολη πρόσβαση στις global μεταβλητές με την εντολή
    `move $s0,$sp`

6 Παρουσίαση μηνυμάτων σφάλματος

Στο αρχείο `error.py` ορίζεται η δομή `ErrorData` καθώς και η συνάρτηση `error()` που χρησιμοποιούνται στα υπόλοιπα μέρη του μεταφραστή για την παρουσίαση σφαλμάτων στο χρήστη. Η δομή `ErrorData` αποτελείται από τρία πεδία: το αρχείο πηγαίου κώδικα Ciscal, τη γραμμή και τη στήλη μέσα σε αυτό το αρχείο όπου συνέβη το σφάλμα. Η συνάρτηση `error()` δέχεται ως παραμέτρους την περιγραφή του σφάλματος και ένα αντικείμενο `ErrorData` και τυπώνει το αρχείο, τη γραμμή και τη στήλη που έγινε το σφάλμα, καθώς και την περιγραφή του σφάλματος. Επίσης ενημερώνει το

χρήστη ότι η διαδικασία της μετάφρασης δεν ήταν επιτυχής και τον παροτρύνει να διορθώσει το πρόβλημα και να προσπαθήσει ξανά. Η μετάφραση τερματίζεται στο πρώτο σφάλμα.