Πανεπιστήμιο Ιωάννινων
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Μάθημα: Μεταφραστές

Final Report

Περιεχόμενα

1. Εισαγωγή

Σκοπός της προγραμματιστικής άσκησης του μαθήματος, είναι η δημιουργία ενός Μεταφραστή της γλώσσας Starlet που μας δίνεται. Η γλώσσα Starlet είναι μια μικρή γλώσσα υλοποιημένη για τις ανάγκες του μαθήματος. Παρουσιάζει όμως αρκετό ενδιαφέρον καθώς πολλές εντολές της περιέχονται σε άλλες γλώσσες προγραμματισμού.

Αρχικά στο πρώτο στάδιο ο Λεκτικός αναλύτης, διαβάζει ένα ρεύμα χαρακτήρων και παράγει ένα ρεύμα λέξεων. Σωρεύει χαρακτήρες για να σχηματίσει λέξεις και εφαρμόζει ένα σύνολο κανόνων για να διαπιστώσει αν κάθε λέξη της γλώσσας πηγής είναι έγκυρη ή όχι. Αν η λέξη είναι έγκυρη, ο αναλυτής μας την ταξινομεί σε μια συνακτική κατηγορία, η αλλιώς μέρος του λόγου. [1]

Η συντακτική ανάλυση είναι το δεύτερο στάδιο του μεταγλωττιστή μας. Ο συντακτικός αναλυτής εργάζεται με το πρόγραμμα στη μορφή στην οποία το έχει μετασχηματίσει ο σαρωτής˙ βλέπει ένα ρεύμα λέξεων, καθεμία από τις οποίες είναι επισημειωμένη με μία συντακτική κατηγορία (το ανάλογο του «μέρους του λόγου» της λέξης). Ο συντακτικός αναλυτής παράγει μία συντακτική δομή για το πρόγραμμα, συμμορφώνοντας τις λέξεις με ένα γραμματικό μοντέλο της γλώσσας προγραμματισμού πηγής. Αν ο συντακτικός αναλύτης διαπιστώσει ότι το ρεύμα εισόδου είναι έγκυρο πρόγραμμα, κατασκευάζει ένα πραγματικό μοντέλο του προγράμματος για να χρησιμοποιηθεί στις επόμενες φάσεις της μεταγλώττισης. Αν το ρεύμα εισόδου δεν είναι έγκυρο πρόγραμμα, ο συντακτικός αναλυτής αναφέρει στον χρήστη το πρόβλημα, μαζί με την κατάλληλη διαγνωστική πληροφορία. [2]

Στο τρίτο στάδιο, αυτό του ενδιάμεσου κώδικα και της παραγωγής του κώδικα σε γλώσσα προγραμματισμού C, σκοπός μας είναι να αρχίσουμε να μετατρέπουμε το πρόγραμμα Starlet στην τελική μας μορφή. Πρόκειται δηλαδή για το βήμα το οποίο χρειαζόμαστε στο επόμενο στάδιο, έτσι ώστε να μπορέσουμε να το μετατρέψουμε σε πρόγραμμα μηχανής (assembly).

Στη συνέχεια ο μεταγλωττιστής συνάγει πληροφορία σχετικά με τις διάφορες οντότητες που χειρίζεται το μεταφραζόμενο πρόγραμμα. Πρέπει να ανακαλύψει και να αποθηκεύσει πολλά διαφορετικά είδη πληροφορίας. Ο μεταγλωττιστής πρέπει να καταγράψει αυτές τις πληροφορίες στην ενδιάμεση αναπαράσταση είτε να τη συναγάγει εκ νέου αν και όποτε χρειαστεί. Χάριν αποδοτικότητας, οι περισσότεροι μεταγλωττιστές καταγράφουν τα γεγονότα αντί να τα υπολογίζουν εκ νέου. Έτσι λοιπόν πρέπει να δημιουργηθεί μία κεντρική αποθήκη για αυτά τα γεγονότα και να επιτραπεί η αποδοτική προσπέλασή της. Αυτή η κεντρική αποθήκη, που ονομάζεται πίνακας συμβόλων γίνεται αναπόσπαστο μέρος της ΕΑ του μεταγλωττιστή. Μέσω του πίνακα συμβόλων η πληροφορία που συνάγεται από ενδεχομένως μακρινά μεταξύ τους μέρη του πηγαίου κώδικα γίνεται τοπική.[3]

Στο τελευταίο στάδιο υπάρχει η παραγωγή του τελικού κώδικα για τον επεξεργαστή MIPS. Έτσι λοιπόν, από κάθε εντολή της ενδιάμεσης αναπαράστασης παράγουμε τις αντίστοιχες εντολές του τελικού κώδικα. Κύριες ενέργειες σε αυτή τη φάση είναι η απεικόνιση των μεταβλητών στη μνήμη (στοίβα) καθώς και το πέρασμα παραμέτρων και η κλήση συναρτήσεων. Με αυτόν τον τρόπο δημιουργείται ο τελικός κώδικα σε γλώσσα προγραμματισμού Assembly.

---

[1] Keith D. Cooper, Linda Torczon – Σχεδίαση και κατασκευή μεταγλωττιστών, Μτφρ: Αλέξανδρος Χορταράς, Επιμ: Γεώργιος Μανής, Νικόλαος Παπασπύρου, Αντώνιος Σαββίδης, Πανεπιστημιακές Εκδόσεις Κρήτης, Ηράκλειο 2018, σελ.25

[2] Keith D. Cooper, Linda Torczon – Σχεδίαση και κατασκευή, ο.π. , σελ.85

[3] Keith D. Cooper, Linda Torczon – Σχεδίαση και κατασκευή, ο.π. , σελ.261- 262

2. Η γλώσσα προγραμματισμού Starlet

Η Starlet υποστηρίζει συναρτήσεις, μετάδοση παραμέτρων με αναφορά, τιμή και αντιγραφή, αναδρομικές κλήσεις και άλλες ενδιαφέρουσες δομές. Επίσης, επιτρέπει φώλιασμα στη δήλωση συναρτήσεων κάτι που λίγες γλώσσες υποστηρίζουν.

Από την άλλη όμως πλευρά, η Starlet δεν υποστηρίζει βασικές δομές όπως η for, ή τύπους δεδομένων όπως οι πραγματικοί αριθμοί και οι συμβολοσειρές. Αυτές οι παραλήψεις έχουν γίνει καθαρά για τις ανάγκες τις άσκησης και δεν έχουν να κάνουν με τη δυσκολία κατασκευής του μεταγλωττιστή.

2.1 Λεκτικές Μονάδες

Το αλφάβητο της Starlet αποτελείται από:

- τα μικρά κεφαλαία γράμματα της λατινικής αλφαβήτου («A»,…, «Z» και «α»,…, «z»),
- τα αριθμητικά ψηφία («0»,… «9»),
- τα σύμβολα αριθμητικών πράξεων («+», «-», «*», «/»),
- τους τελεστές συσχέτισης «<», «>», «=», «<=», «>=», «<>»,
- το σύμβολο ανάθεσης «:=»,
- τους διαχωριστές («;», «,», «:»)
- καθώς και τα σύμβολα ομαδοποίησης («(», «)», «[», «]»)
- και διαχωρισμού σχολίων («/*», «*/», «//»).

Τα σύμβολα «[» και «]» χρησιμοποιούνται στις λογικές παραστάσεις όπως τα σύμβολα «(» και «)» στις αριθμητικές παραστάσεις.

Μερικές λέξεις είναι δεσμευμένες:

program, endprogram
declare
if then else endif
while endwhile, dowhile enddowhile
loop, endloop, exit
forcase, endforcase, incase, endincase, when, default, enddefault
function, endfunction, return, in, inout, inandout
and, or, not
input, printp

Οι λέξεις αυτές δεν μπορούν να χρησιμοποιηθούν ως μεταβλητές. Οι σταθερές της γλώσσας είναι ακέραιες σταθερές που αποτελούνται από προαιρετικό πρόσημο και από μία ακολουθία αριθμητικών ψηφίων. Τα αναγνωριστικά της γλώσσας είναι συμβολοσειρές που αποτελούνται από γράμματα και ψηφία, αρχίζοντας όμως από γράμμα. Ο μεταγλωττιστής λαμβάνει υπόψη του μόνο τα τριάντα πρώτα γράμματα. Οι λευκοί χαρακτήρες (tab, space, return)) αγνοούνται και μπορούν να χρησιμοποιηθούν με οποιονδήποτε τρόπο χωρίς να επηρεάζεται η λειτουργία του μεταγλωττιστή, αρκεί βέβαια να μην βρίσκονται μέσα σε δεσμευμένες λέξεις, αναγνωριστικά, σταθερές. Το ίδιο ισχύει και για τα σχόλια, τα οποία πρέπει να βρίσκονται μέσα στα σύμβολα **/*** και ***/** ή να βρίσκονται μετά το σύμβολο **//** και ως το τέλος της γραμμής. Απαγορεύεται να ανοίξουν δύο φορές σχόλια, πριν τα πρώτα κλείσουν. Δεν υποστηρίζονται εμφωλευμένα σχόλια.

2.2 Τύποι και δηλώσεις μεταβλητών

Ο μοναδικός τύπος δεδομένων που υποστηρίζει η Starlet είναι οι ακέραιοι αριθμοί. Οι ακέραιοι αριθμοί πρέπει να έχουν τιμές από –32767 έως 32767. Η δήλωση γίνεται με την εντολή ***declarations***. Ακολουθούν τα ονόματα των αναγνωριστικών χωρίς καμία άλλη δήλωση, αφού γνωρίζουμε ότι πρόκειται για ακέραιες μεταβλητές και χωρίς να είναι αναγκαίο να βρίσκονται στην ίδια γραμμή. Οι μεταβλητές χωρίζονται μεταξύ τους με κόμματα. Το τέλος της δήλωσης αναγνωρίζεται με το ελληνικό ερωτηματικό. Επιτρέπεται να έχουμε περισσότερες των μία συνεχόμενες χρήσεις της ***declarations*.**

2.3 Τελεστές και εκφράσεις

Η προτεραιότητα των τελεστών από τη μεγαλύτερη στη μικρότερη είναι:

(1) Μοναδιαίοι λογικοί: «not»
(2) Πολλαπλασιαστικοί: «*», «/»
(3) Μοναδιαίοι προσθετικοί: «+», «-»
(4) Δυαδικοί προσθετικοί: «+», «-»
(5) Σχεσιακοί «=», «<», «>», «<>», «<=», «>=»
(6) Λογικό «and»,
(7) Λογικό «or»

2.4 Δομές της γλώσσας

Εκχώρηση

Χρησιμοποιείται για την ανάθεση της τιμής μίας μεταβλητής ή μίας σταθεράς, ή μίας έκφρασης σε μία μεταβλητή.

Απόφαση if

Η εντολή απόφασης ***if*** εκτιμάει εάν ισχύει η συνθήκη condition και εάν πράγματι ισχύει, τότε εκτελούνται οι εντολές που ακολουθούν το ***then*** έως ότου συναντηθεί ***else*** ή ***endif***. Το ***else*** δεν αποτελεί υποχρεωτικό τμήμα της εντολής και γι' αυτό βρίσκεται σε αγκύλη. Οι εντολές που το ακολουθούν εκτελούνται εάν η συνθήκη condition δεν ισχύει. Το ***endif*** είναι υποχρεωτικό τμήμα της εντολής.

Επανάληψη while

Η εντολή επανάληψης ***while*** επαναλαμβάνει συνεχώς τις εντολές statements που βρίσκονται ανάμεσα στο ***while*** και στο ***endwhile*****,** όσο η συνθήκη condition ισχύει. Αν την πρώτη φορά που θα αποτιμηθεί η condition, το αποτέλεσμα της αποτίμησης είναι ψευδές, τότε οι statements δεν εκτελούνται ποτέ.

Επανάληψη dowhile-enddowhile

Η εντολή επανάληψης ***dowhile-enddowhile*** επαναλαμβάνει συνεχώς τις εντολές statements που βρίσκονται ανάμεσα στο ***dowhile*** και στο ***enddowhile*****,** όσο η συνθήκη condition ισχύει. Οι statements εκτελούνται τουλάχιστον μία φορά, πριν αποτιμηθεί η condition.

Επανάληψη loop

Η εντολή επανάληψης **loop** επαναλαμβάνει για πάντα τις εντολές statements που βρίσκονται ανάμεσα στο **loop** και στο **endloop.**Έξοδος από το βρόχο γίνεται όταν κληθεί η εντολή **exit**.

Επανάληψη forcase

Η δομή επανάληψης ***forcase*** ελέγχει τις *condition* που βρίσκονται μετά τα ***when***. Μόλις μία από αυτές βρεθεί αληθής, τότε εκτελούνται οι *statements* που ακολουθούν. Μετά ο έλεγχος μεταβαίνει έξω από την ***forcase.*** Αν καμία από τις ***when*** δεν ισχύει, τότε ο έλεγχος μεταβαίνει στη **default** και εκτελούνται οι αντίστοιχες statements. Στη συνέχεια ο έλεγχος μεταβαίνει στην αρχή της **forcase**.

Επανάληψη incase

Η δομή επανάληψης ***incase*** ελέγχει τις *condition* που βρίσκονται μετά τα ***when***, εξετάζοντας τες κατά σειρά. Για κάθε μία από αυτές που η αντίστοιχη *condition* ισχύει, εκτελούνται οι *statements* που ακολουθούν το σύμβολο "**:**"**.** Θα εξεταστούν όλες οι *condition και θα* εκτελεστούν όλες οι *statements* των οποίων οι *condition* ισχύουν. Αφότου εξετατούν όλες οι ***when*** ο έλεγχος μεταβαίνει έξω από τη δομή ***incase*** εάν καμία από τις *statements* δεν έχει εκτελεστεί ή μεταβαίνει στην αρχή της **incase,** έαν έστω και μία από τις *statements* έχει εκτελεστεί.

Επιστροφή τιμής

Χρησιμοποιείται μέσα σε συναρτήσεις για να επιστραφεί το αποτέλεσμα της συνάρτησης.

Έξοδος

Εμφανίζει στην οθόνη το αποτέλεσμα της αποτίμησης του expression.

Είσοδος

Ζητάει από το χρήστη να δώσεει μία τιμή μέσα από το πληκτρολόγιο.

Η αναλυτική υλοποίηση της κάθε δομής φαίνεται στη γραμματική της γλώσσας παρακάτω και υλοποιείται και ελέγχεται μέσω της Συντακτικής Ανάλυσης.

3. Η γραμματική της γλώσσας Starlet

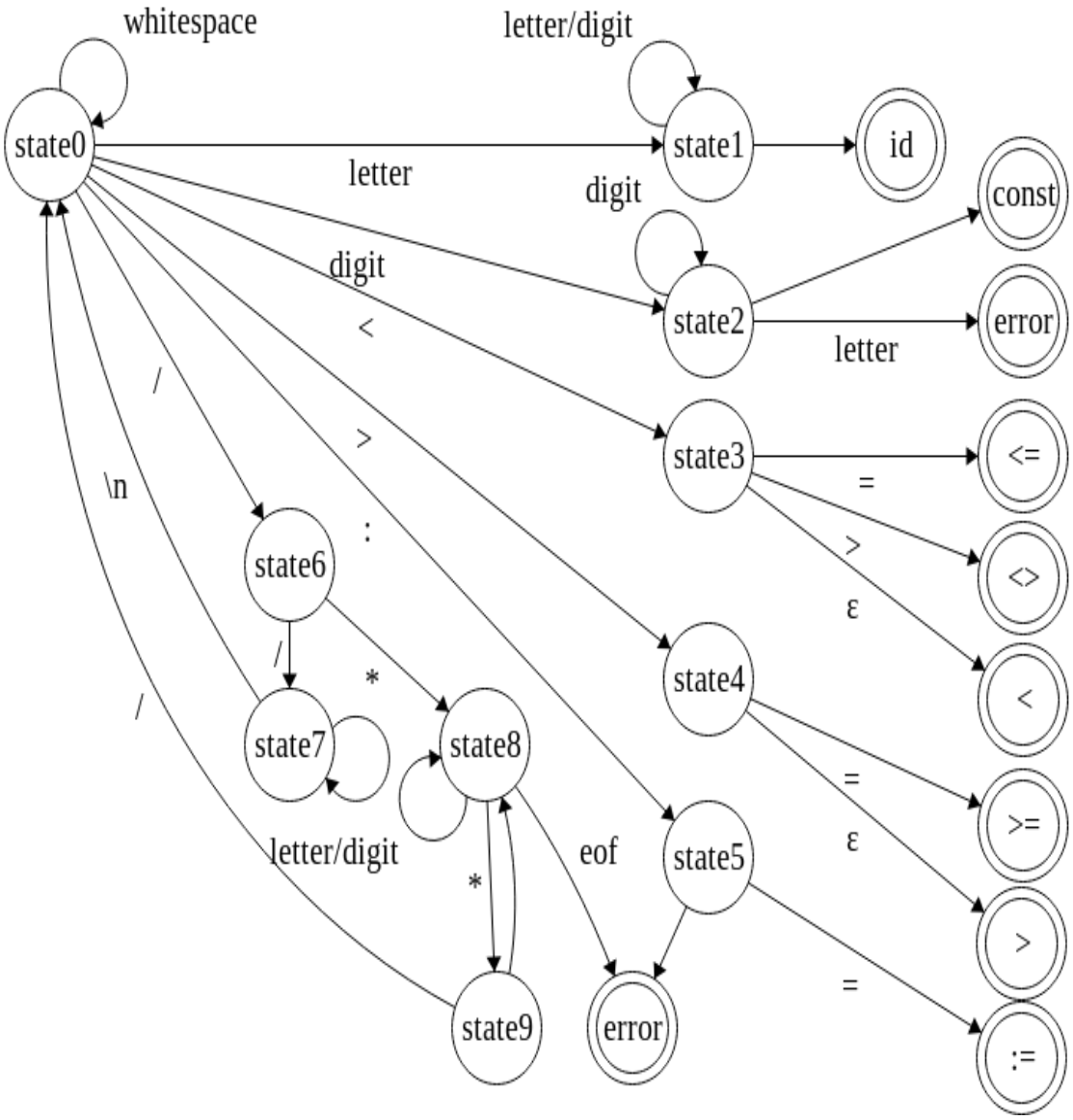
Η γραμματική της γλώσσας μας αποτελείται από ένα σύνολο κανόνων που πρέπει να ακολουθηθούν για τη σωστή γραφή αλλά και υλοποίηση του προγράμματός μας. Παρακάτω (Εικόνα 0) παρατίθεται αυτό το σύνολο κανόνων της γραμματικής μας. Με bold έχουμε αναπαραστίσει τις δεσμευμένες μας λέξεις.

```
<program>            ::= program id <block> endprogram
<block>              ::= <declarations> <subprograms> <statements>
<declarations>       ::= (declare <varlist>;)*
<varlist>            ::= ε | id  ( , id )*
<subprograms>        ::= (<subprogram>)*
<subprogram>         ::= function id <funcbody> endfunction
<funcbody>           ::= <formalpars> <block>
<formalpars>         ::=  ( <formalparlist> )
<formalparlist>      ::= <formalparitem>  ( , <formalparitem> )* | ε
<formalparitem>      ::= in id | inout id | inandout id
<statements>         ::= <statement> ( ; <statement> )*
<statement>          ::= ε |
                         <assignment-stat> |
                         <if-stat> |
                         <while-stat> |
                         <do-while-stat> |
                         <loop-stat> |
                         <exit-stat> |
                         <forcase-stat> |
                         <incase-stat> |
                         <return-stat> |
                         <input-stat> |
                         <print-stat>
<assignment-stat>    ::= id := <expression>
<if-stat>            ::= if (<condition>) then <statements> <elsepart> endif
<elsepart>           ::= ε | else <statements>
<while-stat>         ::= while (<condition>) <statements> endwhile
<do-while-stat>      ::= dowhile <statements>  enddowhile (<condition>)
<loop-stat>          ::= loop <statements> endloop
<exit-stat>          ::= exit
```

```
<forcase-stat>        ::= forcase
                              ( when (<condition>) : <statements> )*
                              default: <statements>  enddefault
                          endforcase
<incase-stat>         ::= incase
                              ( when (<condition>) : <statements> )*
                           endincase
<return-stat>         ::= return <expression>
<print-stat>          ::= print <expression>
<input-stat>          ::= input id
<actualpars>          ::= ( <actualparlist> )
<actualparlist>       ::= <actualparitem> ( , <actualparitem> )* | ε
<actualparitem>       ::= in <expression> | inout id | inandout id
<condition>           ::= <boolterm> (or <boolterm>)*
<boolterm>            ::= <boolfactor> (and <boolfactor>)*
<boolfactor>          ::=not [<condition>] | [<condition>] |
                              <expression> <relational-oper> <expression>
<expression>          ::= <optional-sign> <term> ( <add-oper> <term>)*
<term>                ::= <factor>  (<mul-oper> <factor>)*
<factor>              ::= constant | (<expression>) | id <idtail>
<idtail>              ::= ε | <actualpars>
<relational-oper>     ::= = | <= | >=  | > | < | <>
<add-oper>            ::= + | -
<mul-oper>            ::= * | /
<optional-sign>       ::= ε | <add-oper>
```

4. Λεκτικός Αναλυτής

Ο λεκτικός μας αναλυτής είναι ουσιαστικά μία συνάρτηση η οποία αναπαρίσταται από ένα πεπερασμένο αυτόματο (εικόνα 1). Η συνάρτηση lex() ελέγχει το πρόγραμμα Starlet που δίνει ο χρήστης ώς είσοδο ανα χαρακτήρα για πιθανές τελικές καταστάσεις. Εκτός από τις τελικές καταστάσεις που εμφανίζονται παρακάτω, υπάρχουν και οι καταστάσεις που αντιστοιχούν στα “+”, “-”, “*”, “/”, “=”, “,”, “;”, “(”, “)”, “[”, “]” και τα κενά τα οποία αγνοούνται.



Εικόνα 1: Πεπερασμένο αυτόματο καταστάσεων λεκτικού αναλυτή

4.1 Λεπτομέρειες Μεταβλητών.

Παρακάτω παρατίθεται ο πίνακας με τις μεταβλητές που χρησιμοποιήθηκαν στη συνάρτηση lex() καθώς και μία σύντομη περιγραφή τους.

Μεταβλητές	Περιγραφή
buffer	Λίστα από χαρακτήρες.
state	Η τρέχουσα κατάσταση του αυτόματου (αρχικά είναι 0).
ok	Η τελική κατάσταση του αυτόματου
getBack	Boolean μεταβλητή για να δούμε αν χρειάζεται μετακίνηση ο δείκτης.
ret	Η επιστρεφόμενη τιμή του lex() (String).

4.2 Έλεγχος ορίων αριθμού και επιστροφή τιμής.

Οι ακέραιοι αριθμοί ελέγχονται ένα έχουν τιμές από -32767 έως 32767. Σε περίπτωση σφάλματος, ενημερώνουμε το χρήστη ότι ο αριθμός μας βρίσκεται εκτός ορίων με ένα αντίστοιχο μήνυμα. Στο τέλος ενώνουμε τους χαρακτήρες της λίστας buffer στη μεταβλητή ret, αδειάζουμε τον buffer ώστε να μπορεί να χρησιμοποιηθεί ξανά στην επόμενη κλήση και επιστρέφουμε τους πρώτους 30 χαρακτήρες του ret.

5. Συντακτικός Αναλυτής

Ο συντακτικός αναλυτής ενός μεταγλωττιστή έχει την κύρια ευθύνη για την αναγνώριση της σύνταξης – δηλαδή την ευθύνη να διαπιστώνει αν το μεταγλωττιζόμενο πρόγραμμα είναι έγκυρη πρόταση του συντακτικού μοντέλου της γλώσσας προγραμματισμού. Αυτό το μοντέλο διατυπώνεται με τη μορφή μιας τυπικής γραμματικής G αν κάποια σειρά λέξεων s ανήκει σε στη γλώσσα που ορίζει η G, λέμε ότι η G παράγει την s. Για ένα ρεύμα λέξεων s και μία γραμματική G, ο συντακτικός αναλυτής προσπαθεί να αποδείξει κατασκευαστικά ότι το s μπορεί να παραχθεί από την G- η διαδικασία αυτή ονομάζεται συντακτική ανάλυση.

6. Παραγωγή Ενδιάμεσου Κώδικα

Καθώς ο μεταγλωττιστής παράγει γνώση σχετικά με τον κώδικα που μεταγλωττίζει, πρέπει να μεταφέρει αυτή την πληροφορία από τη μία διέλευση στην επόμενη. Έτσι ο μεταγλωττιστής χρειάζεται μία αναπαράσταση για όλες τις πληροφορίες που εξάγει σχετικά με το πρόγραμμα πηγής. Αυτή η αναπαράσταση αποτελεί την παραγωγή του ενδιάμεσου κώδικα.

Η αναπαράσταση που χρησιμοποιείται στον μεταγλωττιστή μας, είναι γνωστή ως κώδικας τριών διευθύνσεων. Αυτοί, υλοποιούνται ως ένα σύνολο τετράδων όπου κάθε τετράδα αναπαρίσταται με τέσσερα πεδία: έναν **τελεστή** (op), δύο **τελεστέους** και έναν **προορισμό**. Ο συνδυασμός αυτός στη συνέχεια θα αναφέρεται ώς **quad**.

Τα quads είναι αριθμημένα, δηλαδή έχουν έναν μοναδικό αριθμό που τα χαρακτηρίζει. Μόλις τελείωσει η εκτέλεση ενός quad, εκτελείται αυτή με τον αμέσως επόμενο μεγαλύτερο αριθμό, εκτός εάν η τετράδα που μόλις εκτελέστηκε υποδείξει κάτι διαφορετικό. Τέλος, ο μεταγλωττιστής μας υποστηρίζει τη δημιουργία ενδιάμεσου κώδικα σε ANSI C.

6.1 Λεπτομέρειες Υλοποίησης

Στον παρακάτω πίνακα φαίνονται οι συναρτήσεις που χρησιμοποιήθηκαν για την παραγωγή του ενδιάμεσου κώδικα, καθώς και μία σύντομη περιγραφή τους.

Συναρτήσεις	Περιγραφή
next_quad()	Επιστρέφει τον αριθμό του επόμενου quad που πρόκειται να παραχθεί.
gen_quad(op=None, x='_', y='_', z='_')	Δημιουργεί το επόμενο quad.
new_temp()	Δημιουργεί και επιστρέφει μία νέα προσωρινή μεταβλητή της μορφής T_1, T_2,…
empty_list()	Δημιουργεί και επιστρέφει μία κενή λίστα.
make_list(x)	Δημιουργεί και επιστρέφει ένα quad που περιέχει μόνο το x.
merge(list1, list2)	Επιστρέφει τη συνένωση των list1 και list2.
backpatch(labellist, z)	Η labellist αποτελείται από quads των οποίων το τελευταίο τελούμενο δεν είναι συμπληρωμένο. Η backpatch επισκέπτεται ένα ένα τα quads και τα συμπληρώνει με την ετικέτα z

6.2 Global Μεταβλητές.

Στην παραγωγή του ενδιάμεσου κώδικα χρησιμοποιήθηκαν οι εξής **global** μεταβλητές:

Μεταβλητές	Περιγραφή
quadDict	Είναι μία δομή λεξικού στην Python. Χρησιμοποιείται για την αποθήκευση των quads. Στο πεδίο key αποθηκεύουμε τον μοναδικό αριθμό και στο πεδίο value το quad.
nextLabel	Είναι ένας μετρητής (αρχικοποιημένος στο 0) που αποθηκεύει τον μοναδικό αριθμό κάθε quad.
tCounter	Είναι ένας μετρητής (αρχικοποιημένος στο 1) που αποθηκεύει την τιμή της τρέχουσας προσωρινής μεταβλητής.
new_exit_list	Είναι μία αρχικά κενή λίστα, η οποία χρησιμοποιήθηκε για τις loop/exit συναρτήσεις.

6.3 Πρόσθετες λεπτομέρειες

Στον ενδιάμεσο κώδικά, μπορούμε να έχουμε πολλές διαφορετικές μορφές τετράδων. Αναλυτικά:

- **op, x, y, z**
  Όπου το **op** είναι ένα εκ των «+», «-», «*», «/». Τα τελούμενα x,y μπορεί να είναι ονόματα μεταβλητών ή αριθμητικές σταθερές. Το τελούμενο z μπορεί να είναι όνομα μεταβλητής. Στην ουσία εφαρμόζεται ο τελεστής op στα τελούμενα x και y , και το αποτέλεσμα αποθηκεύεται στο τελούμενο z.

- **:=, x, _, z**
  Το τελούμενο x μπορεί να είναι όνομα μεταβλητής ή αριθμητική σταθερά, ενώ το τελούμενο z μόνο όνομα μεταβλητής. Στην ουσία η τιμή του x εκχωρείται στη μεταβλητή z.

- **jump,_,_,z**
  Χωρίς να υπάρχει κάποιος άλλος έλεγχος οποιασδήποτε μεταβλητής, γίνεται μεταπήδηση στη θέση z.

- rel**op, x, y, z**
  Όπου το rel**op** είναι ένα εκ των «=», «<», «>», «<>», «<=», «>=». Γίνεται μεταπήδηση στην θέση z, αν ισχύει η συνθήκη relop μεταξύ των x και y. Συνήθως, αυτή η περίπτωση ακολουθείται από μία μορφή quad τύπου jump και εκτελείται όταν δεν ισχύει η συνθήκη relop.

- **begin_block, name, _, _**
  Συμβολίζει την αρχή προγράμματος ή υποπρογράμματος με το όνομα name.

- **end_block, name, _, _**
  Συμβολίζει το τέλος προγράμματος ή υποπρογράμματος με το όνομα name.

- **halt, _, _, _**
  Συμβολίζει τον τερματισμό του προγράμματος.

- **par, x, m, _**
  Όπου το **x** είναι παράμετρος συνάρτησης και το **m** είναι ένας τρόπος μετάδοσης της τιμής.
  CV-> μετάδοση με τιμή,
  REF-> μετάδοση με αναφορά,
  RET-> επιστροφή τιμής συνάρτησης.

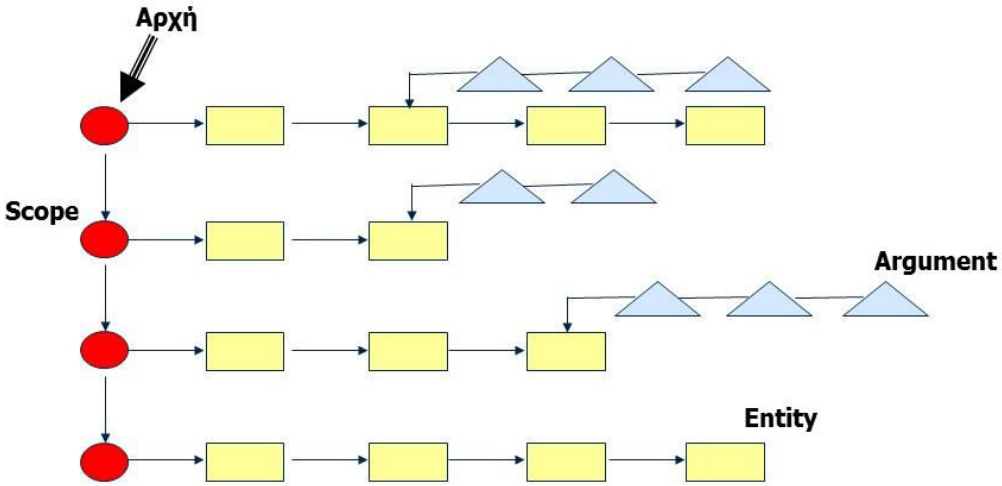
- **call, name, _, _**
  Συμβολίζει την κλήση συνάρτησης name

- **ret, x, _, _**
  Συμβολίζει την επιστροφή τιμής συναρτησης.

7. Πίνακας Συμβόλων

Η τεχνολογία του πίνακα συμβόλων μας προσφέρει έναν αποτελεσματικό και αποδοτικό τρόπο υλοποίησης απεικονίσεων μη ακέραιων δεδομένων σε ένα συνεπτυγμένο σύνολο ακεραίων. Χαρακτηριστικό παράδειγμα η απεικόνιση μίας κειμενικής συμβολοσειράς, όπως του ονόματος μίας μεταβλητής ή μίας προσωρινής τιμής, σε έναν ακέραιο κ.α. [4]

7.1 Μορφή πίνακα συμβόλων και εγγραφές



- **Scope**
  Είναι μία δομή (περιγράφεται στη συνέχεια) η οποία δημιουργείται όταν ξεκινάμε τη μετάφραση μίας νέας συνάρτησης. Στο τέλος με τη διαγραφή διαγράφεται τόσο το ίδιο το Scope όσο και τα Entities αλλα και τα Arguments που εξαρτώνται από αυτήν.

- **Entity**
  Είναι και αυτά μία δομή που κρατάνε πληροφορία σχετική με τις μεταβλητές αλλά και τις συναρτήσεις του προγράμματός μας. Όταν συναντάμε δήλωση μεταβλητης, δημιουργία νέας προσωρινής μεταβλητής, δήλωση νέας συνάρτησης και δήλωση τυπικής παραμέτρου συνάρτησης , δημουργείται ένα νέο Entity κατά περίπτωση.

- **Argument**
  Είναι μία δομή που κρατάει την πληροφορία για τον τρόπο περάσματος των παραμέτρων μίας συνάρτησης και δημιουργείται όταν συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης.

[4] Keith D. Cooper, Linda Torczon – Σχεδίαση και κατασκευή μεταγλωττιστών, Μτφρ: Αλέξανδρος Χορταράς, Επιμ: Γεώργιος Μανής, Νικόλαος Παπασπύρου, Αντώνιος Σαββίδης, Πανεπιστημιακές Εκδόσεις Κρήτης, Ηράκλειο 2018, σελ.25

7.2 Λεπτομέρειες Κλάσεων

- **Scope**
  - nesting_level: Βάθος φωλιάσματος του Scope (αρχικά 0).
  - enclosing_scope: Αναφορά στο αντικείμενο Scope του πατέρα.
  - entities: Μία λίστα από αντικέιμενα τύπου Entity.
  - sp: Αριθμός offset που λαμβάνει υπ όψιν όλα τα Entity και ενημέρωνεται κατάλληλα. Χρησιμοποιείται για ενημέρωση του framelength και είναι αρχικοποιημένο στην τιμή 12.

- **Argument**
  - par_mode: Ο τύπος περάσματος παραμέτρου σε συνάρτηση (cv,ref,ret).
  - next_argument: Αναφέρεται στο επόμενο argument.

- **Entity**
  - name: Όνομα του αντικειμένου
  - entity_type: Τύπος του αντικειμένου (VAR, FUNC, PAR, TMPVAR).

- **Variable** (Entity) δηλαδή κληρονομεί από το Entity
  - offset: Απόσταση από την κορυφή της στοίβας.

- **Function** (Entity)
  - ret_val: Τύπος επιστροφής.
  - start_quad: Ετικέτα της πρώτης τετράδας του κώδικα της συνάρτησης.
  - arguments: Λίστα παραμέτρων.
  - framelength: Μήκος εγγραφήματος δραστηριοποίησης.

- **Parameter** (Entity)
  - par_mode: Ο τύπος περάσματος (in-> cv, inout-> ref, inandout-> ret).
  - offset: Απόσταση από την κορυφή της στοίβας.

- **TempVariable** (Entity)
  - offset: Απόσταση από την κορυφή της στοίβας.

7.3 Λεπτομέρειες Συναρτήσεων

- **Συναρτήσεις της κλάσης Scope**

Συνάρτηση	Περιγραφή
get_sp(self)	Αυξάνει το offset κατά 4. Ενημερώνει το πεδίο της κλάσης του και επιστρέφει την τιμή.
add_entity(self, ent)	Προσθέτει ένα πεδίο ent τύπου Entity στη λίστα entities.
set_enclosing_scope(self, x)	Αναθέτει το πεδίο x τύπου Scope στο enclosing scope.

- **Συναρτήσεις της κλάσης Function(Entity)**

Συνάρτηση	Περιγραφή
set_framelength(self, x)	Αναθέτει την τιμή x στο τοπικό framelength
set_start_quad(self, x)	Αναθέτει την τιμή x στο τοπικό start_quad
set_ret_val(self, x)	Αναθέτει την τιμή x στο τοπικό ret_val

- **Υπόλοιπες συναρτήσεις που χρειάστηκαν για την υλοποίηση**

Συνάρτηση	Περιγραφή
search_entity(name, entity_type)	Η συνάρτηση αυτή μπορεί να αναζητήσει κάποιο Entity με βάση το όνομα και τον τύπο του. Η αναζήτηση ενός entity γίνεται ξεκινώντας από την αρχή του πίνακα (scope_list) και την πρώτη του γραμμή. Αν δε βρεθεί πηγαίνουμε στην επόμενη γραμμή έως ότου βρεθεί το entity ή τελειώσουν όλα τα entities οπότε επιστρέφουμε και μήνυμα λάθους. Αν με το ζητούμενο όνομα υπάρχει πάνω από ένα entity τότε επιστρέφουμε το πρώτο που θα συναντήσουμε.
add_arg_to_func(par_mode, f_name)	Η συνάρτηση αυτή μπορεί να προσθέσει μία παράμετρο σε μία συνάρτηση εφοσον υπάρχει, και επιστρέφει μήνυμα λάθους αν όχι.
add_var_entity(var_name)	Η συνάρτηση αυτή δημιουργεί και προσθέτει ένα Variable(Entity) στο

	τρέχον Scope καλώντας τη συνάρτηση add_entity της κλάσης Scope εφόσον η μεταβλητή δεν έχει ξαναδηλωθεί προηγουμένως και δεν υπάρχει σαν παράμετρος συνάρτησης. Σε αντιθετη περίπτωση επιστρέφει μήνυμα λάθους.
add_param_entity(par_name,par_mode)	Η συνάρτηση αυτή δημιουργεί και προσθέτει ένα Parameter(Entity) στο τρέχον Scope καλώντας τη συνάρτηση add_entity της κλάσης Scope εφόσον είναι μοναδική. Σε αντίθετη περίπτωση επιστρέφει μήνυμα λάθους.
exists_as_param(name, level)	Η συνάρτηση αυτή τσεκάρει αν το Entity με βάση το πεδίο name και το level υπάρχει σαν παράμετρος συνάρτησης στο ίδιο βάθος φωλιάσματος.
unique(name,entity_type,nesting_level)	Η συνάρτηση αυτή υλοποιεί μία αναζήτηση στον scope_list ενός entity με βάση το όνομα, τον τύπο και το βάθος φωλιάσματος. Στην ουσία είναι ένας έλεγχος μοναδικότητας με βάση τα παραπάνω πεδία.
exists(name)	Η συνάρτηση αυτή υλοποιεί έναν απλό έλεγχο με βάση το ονόμα.

7.4 Global Μεταβλητές

Για το κομμάτι της υλοποίησης του πίνακα συμβόλων χρείαστηκε μόνο μία καθολική μεταβλητή η οποία είναι η scope_list. Πρόκειται για μία λίστα που συγκρατεί πεδία αντικειμένων τύπου Scope.

8. Τελικός Κώδικας

Στη φάση αυτή, η παραγωγή τελικού κώδικα σε γλώσσα μηχανής είναι έτοιμη να πραγματοποιηθεί. Έπειτα μπορεί να τρέξει χρησιμοποιώντας τον MARS για τον επεξεργαστή MIPS.

8.1 Αρχιτεκτονική MIPS

Καταχωρητές	Περιγραφή
$t0-$t7	Χρησιμοποιούνται για την καταχώρηση προσωρινών τιμών.
$s0-$s7	Χρησιμοποιούνται για την καταχώρηση τιμών που διατηρούνται ανάμεσα σε κλήσεις συναρτήσεων.
$a0-$a3	Χρησιμοποιόνται για την καταχώρηση ορισμάτων.
$v0-$v1	Χρησιμοποιούνται για την καταχώρηση τιμών.
$sp	Χρησιμοποιείται για δείκτη στη στοίβα του προγράμματος.
$fp	framepointer
$ra	Χρησιμοποιείται για την επιστροφή διεύθυνσης.

8.2 Περιγραφή Εντολών

Εντολές	Περιγραφή
add, sub, mul, div	Αριθμητικές Παραστάσεις
move $t0, $t1	Μεταφορά ανάμεσα σε καταχωρητές. (t0=t1)
lw/li $t1, mem	Περιεχόμενο μνήμης σε καταχωρητή. Το i χρησιμοποείται για σταθερά.
sw $t1, mem	Περιεχόμενο καταχωρητή σε μνήμη.
beq/blt/bgt/ble/bge/bne $t1, $t2, label	Τελεστές σύγκρισης. Αν ικανοποιείται η συνθήκη κάνουν jump στο label.
j label	Jump στο label
jal label	Κλήση συνάρτησης
jr $ra	Άλμα στη διεύθυνση επιστροφής συνάρτησης.

8.3 Λεπτομέρειες Συναρτήσεων

Συνάρτηση	Περιγραφή
gnvlcode(v)	Μεταφέρει στον $t0 την διεύθυνση μιας μη τοπικής μεταβλητής. Από τον πίνακα συμβόλων βρίσκει πόσα επίπεδα επάνω βρίσκεται η μη τοπική μεταβλητή και μέσα από τον σύνδεσμο προσπέλασης την εντοπίζει.
loadvr(v, r)	Μεταφορά δεδομένων στον καταχωρητή r. Η μεταφορά μπορεί να γίνει από τη στοίβα ή να εκχωρηθεί στο r μία σταθερά. Ανάλογα την περίπτωση διαφέρουν και οι εντολές assembly
storerv(r, v)	Μεταφορά δεδομένων από τον καταχωρητή r στη μνήμη (μεταβλητή v). Και εδώ ανάλογα την περίπτωση διαφέρουν και οι εντολές assembly.
write_to_asm(quad, name, labelno)	Για κάθε quad, ανάλογα την περίπτωση της τετράδας, παράγονται με βάση τον πίνακα συμβόλων οι αντίστοιχες εντολές assembly.
testing (name)	Αναζήτηση ενός Entity στον πίνακα συμβόλων με βάση το όνομά του. Η αναζήτηση ξεκινάει από το μεγαλύτερο προς το μικρότερο βάθος φωλιάσματος και επιστρέφει το αντικείμενο τύπου Entity, καθώς και το βάθος φωλιάσματος του Scope.

8.4 Global Μεταβλητές

Μεταβλητή	Περιγραφή
parlist	Είναι μία λίστα από quads των παραμέτρων.
lmain_flag	Είναι μία Boolean μεταβλητή που χρησιμοποιήθηκε στη write_to_asm()
scopes_list	Πίνακας Συμβόλων που παράχθηκε στο προηγούμενο κεφάλαιο.
quadDict	Ενδιάμεσος κώδικας που παράχθηκε σε προηγούμενο κεφάλαιο.
program_name	Όνομα προγράμματος

9. Βασικές Συναρτήσεις Υλοποίησης

Παρακάτω παρατίθονται οι συναρτήσεις που χρησιμοποίηθηκαν στη main αλλά και σε διάφορα άλλα σημεία του προγράμματός μας για τη σωστή υλοποίηση του μεταγλωττιστή μας.

Συνάρτηση	Περιγραφή
is_valid_id(tk)	Είναι μία συνάρτηση που χρησιμοποιήθηκε στη συντακτική ανάλυση και ελέγχει αν ένα token είναι έγκυρο για τη χρήση που προορίζεται.
error(x)	Είναι μία συνάρτηση που χρησιμοποιήθηκε για κλήσεις λαθών. Εκτυπώνει ανάλογο μήνυμα στο χρήστη καθώς και τη γραμμή που έχει γίνει το λάθος. Τέλος, κλείνει το σύστημα.
open_files(filenames*)	Είναι μία συνάρτηση που χρησιμοποιείται για το άνοιγμα των αρχείων. Το filenames είναι τα ονόματα των αρχείων σε μορφή .c, .int, .asm. Επίσης ανοίγει και το αρχείο που δίνει ο χρήστης σαν όρισμα.
write_int_to_file ()	Είναι μία συνάρτηση που χρησιμοποιείται για την εγγραφή του ενδιάμεσου κώδικα στο αρχείο.
write_to_c ()	Είναι μία συνάρτηση που χρησιμοποιείται για την εγγραφή του ενδιάμεσου κώδικα στο αρχείο C.
to_c(key)	Είναι μία συνάρτηση που χρησιμοποιείται για τη μετατροπή του ενδιάμεσου κώδικα, σε κώδικα γλώσσας προγραμματισμού C.
main ()	Η κύρια συνάρτηση του προγράμματός μας.