ΜΥΥ802 – ΜΕΤΑΦΡΑΣΤΕΣ
GreekPlusPlus

Εργασία Εξαμήνου 2025

Παπασπύρος Στυλιανός – 5162, Σταυρόπουλος Παναγιώτης 4990

1η Φάση (Λεκτικός – Συντακτικός Αναλυτής)

Σε αυτήν την φάση μας ζητήθηκε να σχεδιάσουμε τον απαραίτητο λεκτικό αναλυτή (lexer), υπεύθυνο για την ορθή παραγωγή των token από τον πηγαίο κώδικα, και τον απαραίτητο συντακτικό αναλυτή (parser), υπεύθυνο για τον γραμματικό έλεγχο του πηγαίου κώδικα της GreekPlusPlus.

Συγκεκριμένα, ο lexer σχεδιάστηκε έτσι ώστε σε κάθε κλήση του να επιστρέφει το επόμενο token (την επόμενη λεκτική μονάδα δηλαδή που προκύπτει από τον πηγαίο κώδικα), την ομάδα στην οποία ανήκει το token (id, αριθμητικός/σχεσιακός τελεστής, αριθμός, δεσμευμένη λέξη, κλπ), και τον αύξων αριθμό της γραμμής στην οποία «βρέθηκε» το token.

Οι πληροφορίες αυτές είναι εξαιρετικά σημαντικές τόσο για τον parser, στον οποίο και τροφοδοτούνται, όσο και για τον πρώιμο εντοπισμό σφαλμάτων στον κώδικα της GreekPlusPlus και – κατά το δυνατόν – καλύτερη ενημέρωση του χρήστη.

Ο Parser με την σειρά του, είναι υπεύθυνος για τον προαναφερθέν έλεγχο και επιπλέον, σύμφωνα με την γραμματική της GreekPlusPlus, για την συντακτική ορθότητα του προγράμματος.

2η Φάση (Ενδιάμεσος κώδικας)

Σε αυτή την φάση μας ζητήθηκε να σχεδιάσουμε και να υλοποιήσουμε το μέρος που παράγει τον ενδιάμεσο κώδικα. Ο ενδιάμεσος κώδικας λειτουργεί ως γέφυρα ανάμεσα στον συντακτικό αναλυτή (Parser) και τον τελικό κώδικα (RISC-V assembly), ο οποίος όμως θα αναλυθεί παρακάτω.

Πρώτα από όλα, πρέπει να αναφερθεί ότι χρησιμοποιήσαμε την μορφή 4άδων (Quads) για την αναπαράσταση του ενδιάμεσου κώδικα. Στον κώδικα μας η κλάση Quad, περιέχει όλες τις πληροφορίες και τις μεθόδους που θα χρησιμοποιηθούν στον Parser. Παρακάτω θα εξηγήσουμε τις αλλαγές που κάναμε ώστε να παράγεται ο κατάλληλος ενδιάμεσος κώδικας.

- Για την αρχή και το τέλος του κύριου προγράμματος δηλαδή 4-αδες της μορφής begin_block, end_block και halt, χρειάστηκε να τροποποιήσουμε τις μεθόδους program και program_block. Αναλυτικότερα στην program, «κρατάμε» το όνομα του κύριου προγράμματος, ενώ στην program_block δημιουργούμε τις αντίστοιχες 4-αδες. Σημαντικός είναι ο χώρος στον οποίο δημιουργούμε τις 4-αδες αυτές, καθώς όπως θα δείτε η begin_block δημιουργείται αφού έχει κληθεί τόσο η μέθοδος declarations() όσο και η subprograms(), ενώ οι 4-αδες halt και end_block δημιουργούνται αφού το πρόγραμμα έχει επιστρέψει απο την κλήση της sequence() και έχει αναγνωρίσει το «τέλος_προγράμματος»

- Στην συνέχεια, θα αναφερθούμε στην subprograms() και πιο συγκεκριμένα στις μεθόδους func/proc αναλόγως με το ποιά από τις δύο, έχει κληθεί. Ας ξεκινήσουμε με την func(). Στην func(), λειτουργούμε αρχικά όπως και παραπάνω, δηλαδή «κρατάμε» το όνομα της συνάρτησης που κλήθηκε. Επόμενο βήμα όπως αναφέρει και η γραμματική της άσκησης, είναι να κληθεί η μέθοδος func_block

και κατά σειρά οι funcinput, funcoutput, declarations και subprograms. Αφού γίνουν όλα αυτά, τότε δημιουργούμε την 4-αδα begin_block για την συνάρτηση. Όπως και για το κυρίως πρόγραμμα, έτσι και εδώ αφού κληθεί η sequence και αναγνωριστεί το «τέλος_συνάρτησης», τότε δημιουργούμε την 4-αδα end_block.

- Για τις procedures (proc) ακολουθούμε την ίδια ακριβώς διαδικασία όπως και στις συναρτήσεις, δηλαδή ο χώρος στον οποίο δημιουργούνται οι begin_block και end_block είναι και εδώ συγκεκριμένος και εξαρτάται απολύτως απο την γραμματική της άσκησης.

- Επόμενη μέθοδος στην οποία κάναμε αλλαγές είναι η assignment_stat. Πιο συγκεκριμένα, η μέθοδος αυτή αφορά γραμμές της μορφής -> ID ':=' expression . Στο πρώτο μέρος της μεθόδου αυτής γίνεται η παραγωγή τον 4-αδων επιστροφής συνάρητησης. Ξέρουμε ότι μια συνάρτηση (για παράδειγμα αύξηση) επιστρέφει έτσι: αύξηση := α + 1, συνεπώς αρχικά πρέπει να δούμε αν το ID αφορά συνάρτηση, ή ένα απλό variable. Αν αφορά συνάρτηση, τότε μετά την κλήση και αποτίμηση της expression(), δημιουργούμε την 4αδα retv. Αν τώρα δούμε ότι το ID αφορά ενα απλό variable, τότε αφού και πάλι κληθεί η expression δημιουργούμε την 4αδα του assignment, με μια εξαίρεση η οποία θα αναλυθεί παρακάτω.

- Ένα πολύ σημαντικό μέρος της παραγωγής του ενδιάμεσου κώδικα αφορά τη δημιουργία τετραδών (quads) για τη if_stat. Σε αυτό το πλαίσιο, αρχικά γίνεται αποτίμηση της συνθήκης καλώντας τη condition(), και έτσι αποκτούμε δύο λίστες με τετράδες: μία για το true branch και μία για το false branch, δηλαδή τις τετράδες που πρέπει να μεταπηδήσουν σε κάποια άλλη θέση, ανάλογα με το αν η

συνθήκη είναι true ή false. Στη συνέχεια, κάνουμε backpatching στη λίστα που περιέχει τις true-τετράδες, δίνοντάς της ως προορισμό την πρώτη τετράδα του then μπλοκ (δηλαδή, πού πρέπει να μεταβεί η εκτέλεση αν το if ισχύει). Για παράδειγμα έστω η 4αδα
4 : <= , χ , 1 , 6 η οποία μας λέει ότι αν x <= 1 τότε πήγαινε στην 6η 4αδα (backpatching με το 6). Έπειτα, διαχειριζόμαστε το then και το else μέρος. Πριν ολοκληρώσουμε το then, δημιουργούμε μια τετράδα jump με άγνωστο προορισμό (π.χ. jump _, _, _), που θα μας μεταφέρει στο τέλος του if χωρίς να ενδιαφερθούμε για το else. Αυτή η 4αδα προστίθεται σε λίστα ώστε να γίνει αργότερο backpatching με την κατάλληλη διεύθυνση. Για το else, βρίσκουμε τη θέση (quad label) από την οποία ξεκινά, και κάνουμε backpatching στη λίστα των false τετραδών με αυτό το label, δηλαδή, ορίζουμε πού θα πάει η εκτέλεση αν δεν ισχύει η συνθήκη. Τέλος γίνεται backpatching της 4αδας jump, ώστε να μεταφέρει την εκτέλεση στο αμέσως επόμενο σημείο μετά το if.

- Το επόμενο επίσης σημαντικό σημείο της παραγωγής του ενδιάμεσου κώδικα είναι το while και do stat. Ας ξεκινήσουμε απο το while_stat. Η εκτέλεση του while ξεκινάει με την δημιουργεία ενός label το οποίο «κρατάει» την αρχή του βρόχου και θα χρησιμοποιηθεί για να μπορέσουμε να κάνουμε άλμα πίσω στην αρχή μετά το τέλος του. Όπως και πριν αποτιμάται η συνθήκη καλώντας την condition() και έτσι παίρνουμε 2 λίστες απο τετράδες , μία για το true branch και μία για το false branch. Κάνουμε backpatching τις true τετράδες ώστε να δείχνουν στην επόμενη τετράδα δηλαδή της αρχής του while loop body. Η επεξεργασία του σώματος του βρόχου γίνεται με την sequence(). Στην συνέχεια δημιουργούμε μια τετράδα jump που μας πηγαίνει στην αρχή του while loop, ενώ στο τέλος κάνουμε backpatching στις false τετράδες ώστε αυτές να δείχνουν στο αμέσως επόμενο σημείο μετά το while. Στο do_stat τώρα, εκτελείται πρώτα η sequence και μετά γίνεται η

αποτίμηση της συνθήκης. ΄Ετσι, πρώτα δημιουργούμε την start_label και εκτελούμε το loop body και έπειται κάνουμε αποτίμηση της συνθήκης και κατα σειρά backpatching στην αρχή του βρόχου αν η συνθήκη είναι ψευδής. Όλες οι true τετράδες δείχνουν στο αμέσως απόμενο σημείο μετά το do, δηλαδή backpatching με το end_label.

- Το επόμενο και ίσως πιο σημαντικό σημείο είναι το for_stat. Πρώτα από όλα κάνουμε αρχικοποίηση του «μετρητή», αφού αποτιμήσουμε με την κλήση της expression(), την αρχική του τιμή και έτσι δημιουργούμε την πρώτη τετράδα της μορφής -> πχ (:=, 1, _, i) . Επόμενο βήμα είναι να αποτιμήσουμε την τελική τιμή του μετρητή. Στην συνέχεια καταχωρούμε την start_label έτσι ώστε να μπορούμε να κάνουμε άλμα στην αρχή του βρόχου μετά απο κάθε επανάληψη. Έπειτα, φτιάχνουμε μια τετράδα, η οποία θα συγκρίνει σε κάθε επανάληψη αν η συνθήκη του βρόχου (μετρητή) ισχύει ή όχι, ενώ ακολουθεί η 4αδα jump, σε περίπτωση που η συνθήκη δεν ισχύει (η 4αδα αυτή στην συνέχεια κάνει backpatching με το exit_label. Το επόμενο βήμα είναι να φτιαχτούν οι 4αδες που αυξάνουν τον μετρητή ανάλογα με το βήμα που έχει αποτιμηθεί, ενώ ακολουθεί ένα jump στην αρχή του βρόχου καθώς και το backpatching που αναφέρθηκε νωρίτερα.

- Ακολουθούν τα input_stat/print_stat και call_stat τα οποία δημιουργούν τετράδες για την είσοδο/έξοδο δεδομένων καθώς και για την κλήση μιας συνάρτησης ή μιας διαδικασίας. Τα βάλαμε μαζί, λόγω του ότι δεν έχουν κάποιο ιδιαίτερο βαθμό δυσκολίας ή κατανόησης.

- Στην συνέχεια, έχουμε το assignment_stat το οποίο δημιουργεί τετράδες ανάθεσης της μορφής -> 15 : par , t@2 , RET , _ . Ξέρουμε ότι μια συνάρτηση επιστρέφει τιμή έτσι: αύξηση := α + 1 επομένως,

θα χριεαστεί να δημιουργηθεί μια τετράδα η οποία θα αποθηκεύει την τιμή που επιστρέφει η συνάρτηση σε ένα temp variable, αλλά και άλλη μια 4αδα η οποία θα καλεί αυτή την συνάρτηση. Αυτός είναι και ο σκοπός του assign_call_stat.

- Για τις παραμέτρους των συναρτήσεων υπάρχει η actualparitem, στην οποία ανάλογα με το αν η μεταβλητή περνάει με τιμή (CV) ή με αναφορά (REF) δημιουργούνται και οι αντίστοιχες τετράδες. Μια μεταβλητή περνάει με αναφορά σε μια συνάρτηση αν έχει μπροστά της % (και τότε δημιουργούνται 4αδες της μορφής πχ 13 : par , b , REF , _) ενώ αλλιώς περνάει με τιμή και δημιουργούνται 4αδες πχ 14 : par , c , CV , _

- Η μέθοδος condition, αποτιμά εκφράσεις που περιέχουν το λογικό ή. Πιο συγκεκριμένα αποτιμά αρχικά τον πρώτο όρο, αν είναι false τότε κάνει backpatch την false list και συνεχίζει στην αποτίμηση του 2ου όρου. Αν έστω και ένα απο τα παραπάνω είναι true τότε ενώνει τα jumps τόσο απο την αποτίμηση του 1ου όσου και απο αυτή του 2ου όρου καθώς έχουμε λογικό ή. Αν κανένα δεν είναι true, τότε η τελική false_list είναι αυτή του τελικού όρου καθώς μόνο τότε η συνολική συνθήκη αποτιμάται ως false. Στο τέλος επιστρέφεται η true_list και η false_list της συνθήκης.

- Αντίστοιχα με την condition, η μέθοδος boolterm, αποτιμά εκφράσεις που περιέχουν το λογικό και. Αρχικά αποτιμάται ο πρώτος Boolean όρος. Αν αυτός είναι true, τότε γίνεται backpatch της true list ώστε η ροή να συνεχίσει στον επόμενο όρο (αφού έχουμε λογικό και πρέπει να είναι όλα true ώστε η έκφραση να είναι αληθής. Στην συνέχεια αποτιμάται ο επόμενος όρος, αν οποιοσδήποτε από τους όρους είναι false τότε η συνολική έκφραση αποτιμάται ως false και οι false lists συγχωνεύονται. Η τελική true list προκύπτει από τον τελικό όρο καθώς, μόνο αν είναι και αυτός true είναι η έκφραση

αληθής. Στο τέλος επιστρέφεται η true_list και η false_list της συνθήκης.

- Για την αποτίμηση εκφράσεων μια επίσης σημαντική μέθοδος είναι η boolfactor η οποία μπορεί να αποτιμά είτε σχέσεις της μορφής α < β, είτε σχέσεις μέσα σε '[]', είτε και σχέσεις με λογικό όχι πριν από αυτές. Αν βρούμε λογικό όχι, τότε οι true_list και false_list που επιστρέφει η condition αντιστρέφονται. Αν εντοπιστεί απλώς μια έκφραση εντός αγκυλών, τότε καλείται η condition για να αποτιμήσει την εσωτερική συνθήκη. Τέλος, αν δεν εντοπιστεί κανένα από τα παραπάνω, έχουμε έκφραση της μορφής α < β και έτσι δημιουργούμε την αντίστοιχη τετράδα (π.χ. <, α, β, _). Πιο συγκεκριμένα, πρώτα γίνεται αποτίμηση των δύο εκφράσεων αριστερά και δεξιά του τελεστή (left_expr και right_expr) και αποθηκεύεται ο τελεστής. Στη συνέχεια, παράγεται μια τετράδα για τη σχεσιακή πράξη, με προορισμό που θα συμπληρωθεί αργότερα μέσω backpatching. Η θέση αυτής της τετράδας προστίθεται στη λίστα trueList, δηλαδή στην περίπτωση όπου η σχέση ισχύει. Αμέσως μετά παράγεται και μια τετράδα jump, η οποία αντιστοιχεί στο άλμα που θα εκτελεστεί όταν η συνθήκη δεν ισχύει – αυτή η θέση προστίθεται στη falseList.

- Για την παραγωγή τετράδων πρόσθεσης και αφαίρεσης (πχ +(-), α, β, t@1) έχουμε την μέθοδο expression η οποία αποτιμά αρχικά τον πρώτο όρο, συνεχίζει στην αποτίμηση του τελεστή (+ ή -) ενώ αποτιμά και τον δεύτερο όρο και τελικά δημιουργεί την αντίστοιχη τετράδα. Για την αποθήκευση του αποτελέσματος της πρόσθεσης ή της αφαίρεσης χρησιμοποιούμε μια προσωρινή μεταβλητή καλώντας την μέθοδο newTemp.

- Με τον ίδιο τρόπο λειτουργεί και η μέθοδος term, με την μόνη διαφορά να είναι στο ότι τώρα διαχειριζόμαστε πράξεις πολλαπλασιασμού και διαίρεσης, δηλαδή ο τελεστής είναι * ή /

- Τέλος υπάρχει η μέθοδος factor, η οποία σε ότι αφορά τον ενδιάμεσο κώδικα, διαχειρίζεται το αν ένας αριθμός είναι θετικός ή αρνητικός. Πιο συγκεκριμένα για τους αρνητικούς, παράγει τετράδες της μορφής (-, 0, α, t@2) καθώς θεωρεί πως -α είναι (0 – α)

3η Φάση (Πίνακας Συμβόλων)

Σε αυτήν την φάση μας ζητήθηκε να υλοποιήσουμε τον πίνακα συμβόλων. Ο πίνακας συμβόλων είναι η δομή στην οποία κρατάμε πληροφορίες σχετικά με τις μεταβλητές, σταθερές, συναρτήσεις και διαδικασίες του προγράμματος.

Συγκεκριμένα, ο πίνακας αναπαρίσταται από μία λίστα, κάθε θέση της οποίας αποτελεί και μία εμβέλεια (scope) του προγράμματος. Για παράδειγμα, ξεκινώντας πάντα από την πρώτη θέση και λαμβάνοντάς την υπ' όψην ως την καθολική (global scope), για κάθε ορισμό συνάρτησης ή διαδικασίας που συναντούμε στον κώδικα, προσθέτουμε από μία θέση (εμβέλεια) στην λίστα (local scope).

Σε κάθε scope τώρα, αποθηκεύουμε κάποια στοιχεία ανάλογα με τον τύπο της εγγραφής που πρόκειται να κάνουμε:

- Μεταβλητή
    - Όνομα
    - Offset (απόσταση από την αρχή του εγγραφήματος δραστηριοποίησης)*

- Συνάρτηση
    - Όνομα
    - startQuad (ετικέτα της πρώτης τετράδας του κώδικα της συνάρτησης)
    - arguments (λίστα παραμέτρων)
    - framelength (μήκος εγγραφήματος δραστηριοποίησης)

- Σταθερά
    - Όνομα
    - Τιμή

- Παράμετρος
    - Όνομα
    - parMode (τρόπος περάσματος)
    - Offset (απόσταση από την κορυφή της στοίβας)

- Προσωρινή Μεταβλητή
    - Όνομα
    - Offset (απόσταση από την κορυφή της στοίβας)

*Το εγγράφημα δραστηριοποίησης (activation record), δημιουργείται για κάθε συνάρτηση ή διαδικασία που πρόκειται να κληθεί και καταστρέφεται με την ολοκλήρωση της εκτέλεσής της. Σε αυτό τοποθετούνται όλες οι πληροφορίες για την εκτέλεσή της διαδικασίας/συνάρτησης, συμπεριλαμβανομένων των πραγματικών παραμέτρων (actual parameters), των τοπικών μεταβλητών και των προσωρινών μεταβλητών.

4η Φάση (Τελικός Κώδικας)

Σε αυτή την φάση μας ζητήθηκε να σχεδιάσουμε και να υλοποιήσουμε το μέρος του που παράγει τον τελικό κώδικα. Πιο συγκεκριμένα το πως θα μεταβούμε από μια ενδιάμεση αναπαραγωγή της γλώσσας, σε γλώσσα μηχανής (RISC-V assembly).

Πρώτα από όλα πρέπει να αναφερθεί πως για την παραγωγή του τελικού κώδικα χρησιμοποιήσαμε τόσο τον ενδιάμεσο κώδικα, όσο και τον πίνακα συμβόλων, ενώ η βασική κλάση που περιέχει την υλοποίηση του τελικού κώδικα είναι η CodeGenerator.

Θα χωρίσουμε την εξήγηση της υλοποίησης μας σε δύο μέρη, το πρώτο θα αφορά τις μεθόδους της κλάσης CodeGenerator, ενώ το δεύτερο θα αναφέρει τον λόγο και τον τρόπο με τον οποίο χρησιμοποιήθηκαν οι μέθοδοι αυτοί στον Parser.

1. Η κλάση CodeGenerator:

- Η πρώτη μέθοδος της κλάσης, είναι η gnlvcode() η οποία αφορά μη τοπικές μεταβλητές. Αρχικά ψάχνει την μεταβλητή στον πίνακα συμβόλων, καθώς και το πόσα επίπεδα πάνω βρίσκεται αυτή από το τρέχον. Στην συνέχεια πηγαίνουμε στην στοίβα του γονέα, ενώ ανεβαίνουμε σταδιακά επίπεδα μέχρι να φτάσουμε στο σωστό scope της μεταβλητής. Αφού φτάσαμε στο σωστό scope, τότε μετακινούμαστε μέσα σε αυτό με το κατάλληλο offset ώστε να βρούμε την διεύθυνση της μεταβλητής. Πρόκειται για μια βοηθητική μέθοδο η οποία θα χρησιμοποιηθεί από άλλες μεθόδους παρακάτω.

- Ακολουθεί η μέθοδος loadvr() της οποίας σκοπός είναι η μεταφορά δεδομένων (ή μιας σταθεράς) από την μνήμη, σε ένα καταχωρητή (έστω destination_register). Η μέθοδος αυτή διακρίνει περιπτώσεις

ανάλογα με το αν μια μεταβλητή είναι σταθερά, ή ανάλογα με το αν ανήκει σε διαφορετικό επίπεδο (scope) από το τρέχον.

Στην πιο απλή περίπτωση, δηλαδή αν η μεταβλητή είναι σταθερά, παράγεται ο αντίστοιχος τελικός κώδικας χωρίς να χρειαστεί να αλλάξουμε κάτι στους fp, sp, gp.

Αν η μεταβλητή είναι global, τότε μεταβαίνουμε τόσες θέσεις από τον gp, όσες και το offset της μεταβλητής και την φορτώνουμε στον destination_register. Ο gp δείχνει στον χώρο μνήμης που αφορά τις μεταβλητές του κυρίως προγράμματος

Αν η μεταβλητή είναι τοπική μεταβλητή στην συνάρτηση που εκτελείται τώρα, ή είναι τυπική παράμετρος που περνάει με τιμή, ή είναι προσωρινή μεταβλητή, αυτό σημαίνει ότι βρίσκεται στο εγγράφημα δραστηριοποίησης της τρέχουσας συνάρτησης (δηλαδή εκεί που δείχνει ο sp). Επομένως αρκεί να κινηθούμε όσες θέσεις χρειάζεται (με βάση το offset) της μεταβλητής και να την φορτώσουμε στον destination_register.

Αν η μεταβλητή έχει δηλωθεί στην συνάρτηση που εκτελείται τώρα και είναι τυπική παράμετρος που περνάει με αναφορά, αυτό σημαίνει ότι περνάμε την διεύθυνση μνήμης, όπου είναι αποθηκευμένη η πραγματική μεταβλητή και όχι την τιμή της. Επομένως αρχικά φορτώνουμε σε έναν βοηθητικό καταχωρητή (έστω t0) την τιμή που βρίσκεται offset θέσεις από τον sp (δηλαδή την διεύθυνση μνήμης της μεταβλητής) και τελικά φορτώνουμε στον destination_register την τιμή που βρίσκεται στη διεύθυνση που τώρα δείχνει ο t0. Δηλαδή κάνουμε dereference της διεύθυνσης για να πάρουμε εν τέλει την πραγματική τιμή.

Αν η μεταβλητή έχει δηλωθεί σε κάποιο πρόγονο και εκεί είναι τοπική μεταβλητή ή τυπική παράμετρος που περνάει με τιμή αυτό σημαίνει ότι βρίσκεται στο εγγράφημα δραστηριοποίησης του προγόνου. Άρα καλούμε αρχικά την gnlvcode() η οποία επιστρέφει στον t0 έναν δείκτη στην κατάλληλη θέση της μεταβλητής, στο εγγράφημα δραστηριοποίησης του προγόνου και τελικά φορτώνουμε την τιμή που βρίσκεται στον t0 στον destination_register

Τέλος αν η μεταβλητή ανήκει σε κάποιο πρόγονο και εκεί είναι τυπική παράμετρος που περνάει με αναφορά, τότε ανεβαίνουμε με την gnlvcode() στο εγγράφημα δραστηριοποίησης του προγόνου και έτσι ο καταχωρητής t0 δείχνει πλέον στην θέση της διεύθυνσης της παραμέτρου. Στην συνέχεια κάνουμε dereference της διεύθυνσης και έτσι ο t0 έχει την πραγματική τιμή της παραμέτρου, την οποία στην συνέχεια μεταφέρουμε στον destination_register.

- Η επόμενη μέθοδος είναι η storevr(), της οποίας σκοπός είναι η μεταφορά δεδομένων από έναν καταχωρητή (έστω from_register) στην μνήμη. Όπως και στην loadvr(), έτσι και εδώ διακρίνουμε περιπτώσεις ανάλογα με το είδος της μεταβλητής.

  Αν η μεταβλητή είναι καθολική τότε πηγαίνουμε στον που δείχνει ο gp, δηλαδή εκεί που αποθηκεύονται οι καθολικές μεταβλητές και βάζουμε στην κατάλληλη θέση (με βάση το offset της μεταβλητής) την τιμή που βρίσκεται στον from_register.

Αν η μεταβλητή είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον, ή προσωρινή μεταβλητή, τότε η μεταβλητή βρίσκεται στο τρέχον εγγράφημα δραστηριοποίησης. Επομένως κινούμαστε offset θέσεις από την αρχή ενεργού εγγραφήματος δραστηριοποίησης (από τον sp) και αποθηκεύουμε την τιμή που βρίσκεται στον from_register.

Αν η μεταβλητή είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον, τότε φορτώνουμε στον καταχωρητή t0 την διεύθυνση μνήμης όπου είναι αποθηκευμένη η μεταβλητή (η οποία βρίσκεται offset θέσεις από τον sp) και έπειτα αποθηκεύουμε την τιμή του from_register στην διεύθυνση που δείχνει ο t0, δηλαδή στο πραγματικό σημείο που βρίσκεται η μεταβλητή.
Αν η μεταβλητή είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και έχει βάθος φωλιάσματος μικρότερο από το τρέχον, τότε αυτό σημαίνει ότι βρίσκεται στο εγγράφημα δραστηριοποίησης κάποιου προγόνου. Επομένως με την χρήση της gnlvcode() ο t0 θα περιέχει την διεύθυνση της μεταβλητής και έτσι αποθηκεύουμε την τιμή του from_register στην διεύθυνση που δείχνει ο t0, δηλαδή στην διεύθυνση μνήμης της μεταβλητής.

Τέλος, αν η μεταβλητή είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον, τότε καλούμε αρχικά την gnlvcode() και έτσι ο t0 δείχνει σε μια θέση μνήμης που περιέχει την διεύθυνση της μεταβλητής. Στην συνέχεια κάνουμε dereference και εν τέλει ο t0 περιέχει την θέση στην οποία είναι αποθηκευμένη η πραγματική τιμή της μεταβλητής και τέλος γράφουμε το περιεχόμενο του from_register στην θέση αυτή.

- Στην συνέχεια ακολουθεί η μέθοδος generateAssignment() η οποία αφορά εντολές εκχώρησης της μορφής (:=, x, “_”, z). Στην μέθοδο αυτή αρχικά φορτώνουμε την τιμή της μεταβλητής x στον προσωρινό καταχωρητή t1 με την βοήθεια της loadvr() ενώ στην συνέχεια αποθηκεύουμε την τιμή από τον t1 στην μεταβλητή z

- Για εντολές αριθμητικών πράξεων έχουμε δημιουργήσει την μέθοδο generateArithmetic(), η οποία αφορά εντολές της μορφής (op, x, y ,z) αρχικά φορτώνει την τιμή της μεταβλητής x στον καταχωρητή t1, φορτώνει την τιμή της μεταβλητής y στον t2, ανάλογα με το είδος του τελεστή (+, -, /, *) παράγει τον αντίστοιχο τελικό κώδικα και τέλος αποθηκεύεται το αποτέλεσμα στην μεταβλητή z.

- Για τις παραμέτρους μια συνάρτησης υπάρχει η μέθοδος generateParameters(), η οποία διακρίνει περιπτώσεις ανάλογα με το είδος της παραμέτρου (CV, REF, RET)

  Αν η παράμετρος είναι CV δηλαδή περνάει με τιμή και έστω ότι έχει όνομα χ, τότε αρχικά φορτώνουμε την τιμή της παραμέτρου στον καταχωρητή t0 και έπειτα αποθηκεύουμε την τιμή του t0 στην θέση

  -(12 + 4i)(fp) με i τον αύξων αριθμό της παραμέτρου και fp τον καταχωρητή που δείχνει σταθερά στο εγγράφημα δραστηριοποίησης μιας συνάρτησης.

  Αν η παράμετρος είναι REF και η καλούσα συνάρτηση και η μεταβλητή χ έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος χ είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή τότε η χ αποθηκεύεται στο τρέχον εγγράφημα δραστηριοποίησης. Άρα αρχικά υπολογίζουμε την διεύθυνση της χ (αφού έχουμε REF) και την βάζουμε στον καταχωρητή t0 και έπειτα αποθηκεύουμε αυτή την διεύθυνση στο εγγράφημα δραστηριοποίησης της καλούμενης συνάρτησης.

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, αυτό σημαίνει ότι η χ είναι REF στην καλούσα, δηλαδή στο εγγράφημα δραστηριοποίησης της καλούσας υπάρχει η διεύθυνση της μεταβλητής. Άρα τώρα απλά διαβάζουμε την διεύθυνση που δείχνει στην μεταβλητή, την βάζουμε στον t0 και έπειτα αποθηκεύουμε την διεύθυνση αυτή στο εγγράφημα δραστηριοποίησης της καλούμενης συνάρτησης.

Αν η καλούσα συνάρτηση και η μεταβλητή χ έχουν διαφορετικό βάθος φωλιάσματος, η παράμετρος χ είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή, τότε καλούμε την συνάρτηση gnlvcode() ώστε ο t0 να έχει πλέον την σωστή διεύθυνση της χ (στο βάθος φωλιάσματος της) και έπειτα αποθηκεύουμε την διεύθυνση που βρήκαμε στο εγγράφημα δραστηριοποίησης της καλούμενης συνάρτησης.

Αν η καλούσα συνάρτηση και η μεταβλητή χ έχουν διαφορετικό βάθος φωλιάσματος, η παράμετρος χ είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με αναφορά, τότε πάλι όπως και πριν καλούμε την gnlvcode() και αποθηκεύουμε την διεύθυνση της θέσης της χ στον t0, στην συνέχεια παίρνουμε την διεύθυνση που δείχνει η χ και τέλος αποθηκεύουμε την διεύθυνση της μεταβλητής στο εγγράφημα δραστηριοποίησης της καλούμενης συνάρτησης.

Αν η παράμετρος είναι RET, τότε η καλούσα συνάρτηση ετοιμάζει μια προσωρινή μεταβλητή στο δικό της εγγράφημα δραστηριοποίησης στην οποία η καλούμενη θα επιστρέψει το

αποτέλεσμα της. Άρα αρχικά υπολογίζουμε την διεύθυνση της μεταβλητής χ στο εγγράφημα δραστηριοποίησης της καλούσας συνάρτησης και στην συνέχεια αποθηκεύουμε την διεύθυνση αυτή στην $3^{η}$ θέση του εγγραφήματος δραστηριοποίησης της καλούμενης, δηλαδή στην θέση -8(fp).

- Για την κλήση συναρτήσεων έχουμε την μέθοδο generateCall(), η οποία διακρίνει περιπτώσεις ανάλογα με το βάθος φωλιάσματος της καλούσας και της κληθείσας.

  Αν η καλούσα και η κληθείσα έχουν το ίδιο βάθος φωλιάσματος, τότε αυτό σημαίνει πως έχουν τον ίδιο γονέα και επομένως αρχικά φορτώνουμε τον σύνδεσμο προσπέλασης της καλούσας στον t0 και έπειτα τον αποθηκεύουμε στον σύνδεσμο προσπέλασης της κληθείσας.

  Αν η καλούσα και η κληθείσα έχουν διαφορετικό βάθος φωλιάσματος, τότε αυτό σημαίνει πως η καλούσα είναι γονιός της κληθείσας και επομένως ο σύνδεσμος προσπέλασης της κληθείσας πρέπει να δείχνει στο εγγράφημα δραστηριοποίησης της καλούσας.

  Τέλος για αυτή την μέθοδο μεταφέρουμε τον δείκτη στοίβας στην κληθείσα, καλούμε την jal και τον μεταφέρουμε πάλι πίσω.

- Στην συνέχεια υπάρχουν οι μέθοδοι generatePrint(), generateHalt(), generateScan() οι οποίες βοηθούν για την παραγωγή τελικού κώδικα για τετράδες της μορφής (7 : out , α , _ , _), (24 : halt , _ , _ , _) και (46 : in , β , _ , _)

- Για την επιστροφή τιμών συνάρτησης, δηλαδή τετράδες της μορφής (retv, _, _, χ) έχουμε την μέθοδο generateReturn(), η οποία φορτώνει την τιμή της χ σε έναν προσωρινό καταχωρητή (έστω t1), στην συνέχεια παίρνει την διεύθυνση της $3^{η}$ς θέσης του εγγραφήματος

δραστηριοποίησης και τελικά αποθηκεύει την τιμή του χ στην διεύθυνση αυτή.

- Για τις εντολές αλμάτων έχουμε τις generateJump() και generateRelation(). Η 1η δημιουργεί απλά μια εντολή άλματος της μορφής (j L3), ενώ η 2η ανάλογα με το τι είναι το τελεστής (<=, >=, <, >, <>, =) δημιουργεί τελικό κώδικα της μορφής (πχ bgt t1, t2, L3).

- Τέλος υπάρχουν οι beginBlock(), endBlock(), beginMain() οι οποίες σηματοδοτούν την αρχή/τέλος ενός block, αλλά και την αρχή του κυρίως προγράμματος

2. <u>Πως χρησιμοποιούνται στον Parser:</u>

- Αρχικά στην program_block() πριν την κλήση της subprograms() καλούμε την beginBlock(), ενώ μετά την κλήση της καλούμε και την beginMain() για να σηματοδοτήσουμε την έναρξη του κυρίως προγράμματος. Στο τέλος καλούμε την generateHalt() για να σηματοδοτήσουμε την λήξη του.

- Έπειτα στις μεθόδους funcblock() και procblock() καλούμε πάλι τις beginBlock() και endBlock() για να ξεκινήσουμε και να τερματίσουμε ένα block (είτε συνάρτηση είτε διαδικασία)

- Στο assignment_stat() γίνεται η διαχείριση των εντολών εκχώρησης αλλά και της επιστροφής συνάρτησης αφού όπως είχε αναφερθεί και στον ενδιάμεσο κώδικα η επιστροφή μιας συνάρτησης (πχ αύξηση) γίνεται ως εξής -> (πχ αύξηση := α + 3). Επομένως εδώ

χρησιμοποιούνται οι μέθοδοι generateReturn(), generateAssignment() της κλάσης CodeGenerator.

- Για την υλοποίηση του τελικού κώδικα του ifstat() έχουμε χρησιμοποιήσει τις generateRelation() και generateJump()

- Για το print_stat έχουμε την μέθοδο generatePrint()

- Για το call_stat δηλαδή για την κλήση συναρτήσεων έχουμε χρησιμοποιήσει την μέθοδο generateCall()

- Στην assign_call_stat() η οποία διαχειρίζεται εντολές της μορφής χ := αύξηση(α, β) χρησιμοποιούνται οι generateParameters(), generateCall() και generateAssignment().

- Για την διαχείριση των παραμέτρων στην μέθοδο actualparitem() χρησιμοποιείται η generateParameters()

- Ενώ τέλος για την διαχείριση αριθμητικών εντολών στην expression() καλείται η generateArithmetic()