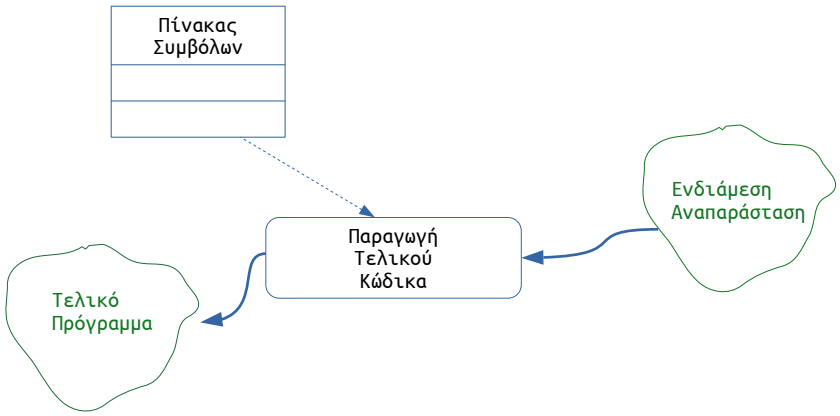
Παραγωγή τελικού κώδικα

Γεώργιος Μανής
Πανεπιστήμιο Ιωαννίνων
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Μάιος 2022

Η τελευταία φάση της παραγωγής κώδικα είναι η παραγωγή του τελικού κώδικα. Ο τελικός κώδικας προκύπτει από τον ενδιάμεσο κώδικα με τη βοήθεια του πίνακα συμβόλων. Συγκεκριμένα, από κάθε εντολή ενδιάμεσου κώδικα προκύπτει μία σειρά εντολών τελικού κώδικα, η οποία για να παραχθεί ανακτά πληροφορίες από τον πίνακα συμβόλων. Η λειτουργία της φάσης της παραγωγής τελικού κώδικα φαίνεται στο σχήμα 1.1.



Σχήμα 1.1: Η παραγωγή τελικού κώδικα στη διαδικασία της μεταγλώττισης.

Η τελική γλώσσα που δημιουργείται από έναν μεταγλωττιστή είναι συνήθως η γλώσσα μηχανής ενός επεξεργαστή. Για τη *𝒞-imple* θα παραγάγουμε τελικό κώδικα σε *συμβολική γλώσσα μηχανής (assembly code)* του επεξεργαστή `RISC-V` [**riscv**].

Ο RISC-V είναι ένας αρκετά αντιπροσωπευτικός επεξεργαστής της τεχνολογίας RISC. Επιλέχθηκε ως επεξεργαστής στόχος για την ανάπτυξη ενός εκπαιδευτικού μεταγλωττιστή, διότι εκτός από αντιπροσωπευτικός, είναι και αρκετά απλός, υποστηρίζοντας όλες τις λειτουργίες που θα ζητούσαμε για την ανάπτυξη του μεταγλωττιστή της *𝒞-imple*. Χρησιμοποιείται σε πολλά Πανεπιστημιακά Τμήματα για μαθήματα κυρίως αρχιτεκτονικής υπολογιστών και δίνει ένα πολύ υποκείμενο σύστημα υλικού για μαθήματα μεταγλωττιστών.

Στο παρόν σύγγραμμα δεν μας ενδιαφέρει να μελετήσουμε σε βάθος τον επεξεργαστή, ούτε και να προσαρμόσουμε την παραγωγή τελικού κώδικα σε αυτόν. Ο τελικός κώδικας θα σχεδιαστεί βασισμένος σε όσο το δυνατόν λιγότερη εξάρτηση από το υλικό και τις ιδιαιτερότητες του RISC-V. Οι εντολές της assembly που θα επιστρατεύσουμε από το διαθέσιμο σύνολο εντολών του RISC-V, υποστηρίζονται από όλους τους επεξεργαστές, με μικρές και μη σημαντικές διαφοροποιήσεις. Η γλώσσα *C-imple* υποστηρίζει μόνο ακέραιους αριθμούς και αυτό έχει το πρόσθετο πλεονέκτημα ότι περιορίζει το σύνολο των εντολών που είναι αναγκαίες.

Παρακάτω θα κάνουμε μία παρουσίαση των εντολών της assembly του RISC-V τις οποίες θα χρησιμοποιήσουμε. Φυσικά, δεν αποτελεί παρουσίαση της γλώσσας μηχανής του RISC-V. Μία αναλυτική παρουσίαση της assembly του RISC-V μπορείτε να βρείτε εδώ [**riscvAss**].

1.1 Η assembly του RISC-V

1.1.1 Οι καταχωρητές του RISC-V

Ο RISC-V διαθέτει 32 καταχωρητές για ακέραιους αριθμούς και 32 καταχωρητές για αριθμούς κινητής υποδιαστολής. Οι καταχωρητές των ακεραίων αριθμών ονομάζονται `x0,x1,...,x31`, αλλά κάθε ένας από αυτούς έχει ένα μνημονικό όνομα, ανάλογα με τη λειτουργία που συνηθίζεται να κάνει. Την αντιστοιχία, αν σας ενδιαφέρει, μπορείτε εύκολα να τη βρείτε, εμείς θα τους παρουσιάσουμε με τα μνημονικά τους ονόματα:

- **zero** (μηδενικός καταχωρητής): καταχωρητής που έχει μόνιμα την τιμή 0. Η συχνή χρήση της σταθεράς 0 οδήγησε στην απόφαση ένας καταχωρητής να αφιερωθεί για το σκοπό αυτόν και να έχει πάντοτε την τιμή 0

- **sp** (stack pointer - δείκτης στοίβας): καταχωρητής που δείχνει στη στοίβα. Σκοπός του είναι να σημειώνει την αρχή του εγγραφήματος δραστηριοποίησης της συνάρτησης ή της διαδικασίας που κάθε στιγμή εκτελείται

- **fp** (frame pointer - δείκτης πλαισίου): καταχωρητής που δείχνει στη στοίβα. Θα τον χρησιμοποιήσουμε για να δείξουμε την αρχή ενός εγγραφήματος δραστηριοποίησης, το οποίο εκείνη τη στιγμή δημιουργείται.

- **t0-t6** (temporary registers - προσωρινοί καταχωρητές): Καταχωρητές που μπορούν να αξιοποιηθούν για οποιονδήποτε σκοπό. Το γράμμα `t` προέρχεται από τη λέξη *temporary*. Ως προσωρινοί καταχωρητές θα χρησιμοποιηθούν και στο σύγγραμμα αυτό, για πολλαπλούς σκοπούς

- **s1-s11** (saved registers - καταχωρητές διατηρούμενων τιμών): Οι τιμές των καταχωρητών αυτών διατηρούνται ανάμεσα σε κλήσεις συναρτήσεων και διαδικασιών

- **a0-a7** (function arguments registers - καταχωρητές ορισμάτων συναρτήσεων): Χρησιμοποιούνται προκειμένου να περαστούν παράμετροι ανάμεσα σε συναρτήσεις ή διαδικασίες. Μπορούν ακόμα να χρησιμοποιηθούν για επιστροφή τιμών

- **ra** (return address - διεύθυνσης επιστροφής): Στον καταχωρητή αυτόν αποθηκεύεται η διεύθυνση στην οποία πρέπει να επιστρέψει ο έλεγχος του προγράμματος, όταν ολοκληρωθεί η εκτέλεση μιας συνάρτησης ή διαδικασίας. Η διεύθυνση τοποθετείται στον καταχωρητή από τον επεξεργαστή με την κλήση της συνάρτησης ή της διαδικασίας

- **pc** (program counter - μετρητής προγράμματος): Περιέχει τη διεύθυνση της εντολής που εκτελείται

- **gp** (global pointer - καθολικός δείκτης): Δείχνει στην αρχή των μεταβλητών που είναι καθολικές σε ένα πρόγραμμα, ώστε να μπορούμε να έχουμε ευκολότερη και τα ταχύτερη πρόσβαση σε αυτές

- **tp** (thread pointer - δείκτης στα δεδομένα ενός νήματος, δεν θα τον χρειαστούμε).

Οι καταχωρητές μπορούν να πάρουν τιμή είτε με απευθείας εκχώρηση αριθμητικής σταθεράς σε αυτούς, είτε με μεταφορά δεδομένων από έναν καταχωρητή σε έναν άλλον, είτε με τέλεση κάποιας αριθμητικής πράξης.

Η απευθείας εκχώρηση αριθμητικής σταθεράς σε έναν καταχωρητή γίνεται με την ψευδοεντολή `li`:

```
li reg,int   # reg = int
   # reg: destination register
   # int: arithmetic integer constant
```

Για παράδειγμα η εντολή:

```
li t0,3
```

εκχωρεί το 3 στον καταχωρητή `t0`. Το γεγονός ότι πρόκειται για ψευδοεντολή και υλοποιείται στην πραγματικότητα μέσω κάποιας άλλης εντολής του RISC-V έχει ενδιαφέρον σε επίπεδο αρχιτεκτονικής υπολογιστών, αλλά δεν μας απασχολεί σε επίπεδο μεταγλωττιστών.

Η μεταφορά τιμής από έναν καταχωρητή σε έναν άλλο γίνεται με την ψευδοεντολή `mv`:

```
mv reg1,reg2   # reg1 = reg2
   # reg1: destination register
   # reg2: source register
```

Στο παρακάτω πρόγραμμα οι καταχωρητές `t1` και `t2` εναλλάσσουν τις τιμές τους, μέσω του καταχωρητή `t0`:

```
mv t0,t1
mv t1,t2
mv t2,t0
```

1.1.2 Πράξεις μεταξύ ακέραιων αριθμών

Για τις τέσσερις αριθμητικές πράξεις ανάμεσα σε καταχωρητές είναι διαθέσιμες οι εξής εντολές:

```
add reg, reg1, reg2    # reg = reg1 + reg2
sub reg, reg1, reg2    # reg = reg1 - reg2
mul reg, reg1, reg2    # reg = reg1 * reg2
div reg, reg1, reg2    # reg = reg1 / reg2
    # reg: destination register
    # reg1,reg2: registers (operands)
```

όπου τα `reg1`,`reg2`,`reg3` είναι καταχωρητές.

Πρόσθεση μίας ακέραιας σταθεράς σε έναν καταχωρητή γίνεται με την `addi`

```
addi target_reg, source_reg, int    # target_reg = source_reg + int
    # target_reg, source_reg: registers
    # int: arithmetic integer constant
```

Η διαίρεση μεταξύ ακεραίων επιστρέφει ακέραιο αριθμό, ενώ υπάρχει και η `rem` η οποία επιστρέφει το υπόλοιπο της διαίρεσης.

Στο παρακάτω πρόγραμμα οι καταχωρητές `t1` και `t2` εναλλάσσουν τις τιμές τους, χωρίς τη χρήση του καταχωρητή `t0`:

```
add t1, t1, t2
sub t2, t1, t2
sub t1, t1, t2
```

1.1.3 Η πρόσβαση στη μνήμη

Για την πρόσβαση στη μνήμη χρησιμοποιούμε τις εντολές `lw` και `sw`. Το `l` συμβολίζει την ανάγνωση, το `s` την εγγραφή, ενώ το `w` ότι πρόκειται να διαβάσουμε ή να γράψουμε δεδομένα μεγέθους μίας λέξης (4 bytes). Υποστηρίζονται εντολές για εγγραφή ή ανάγνωση ενός byte (`lb`,`sb`), μισής λέξης, δηλαδή δύο bytes, (`lh`,`sh`), δύο λέξεων, δηλαδή οκτώ bytes (`ld`,`sd`), τις οποίες, όμως, δεν θα χρειαστούμε.

Η πρόσβαση στη μνήμη γίνεται μέσω ενός καταχωρητή. Για τον σκοπό αυτό χρησιμοποιούμε συνήθως τον καταχωρητή `sp` ή τον `fp`. Πολλές φορές θα χρησιμοποιήσουμε και τον `t0`. Όπως είπαμε, με τις εντολές `lw` και `sw` διαβάζουμε και γράφουμε μία λέξη στη μνήμη. Μας ενδιαφέρει η πρόσβαση μέσω καταχωρητή. Στα σχόλια, παρακάτω, τα σύμβολα `[]` αναπαριστούν έμμεση αναφορά, δηλαδή χ συμβολισμός `[x]` υποδηλώνει ότι θα προσπελάσουμε το περιεχόμενο της μνήμης στη θέση `x`:]

```
lw reg1,offset(reg2)   # reg1 = [reg2 + offset]
    # reg1: destination register
    # reg2: base register
    # offset: distance from reg2
sw reg1,offset(reg2)   # [reg2 + offset] = reg1
    # reg1: source register
    # reg2: base register
    # offset: distance from reg2
```

Η πρόσβαση στη μνήμη γίνεται ως εξής: λαμβάνοντας ως βάση τον καταχωρητή `reg2`, μετακινούμαστε κατά `offset` θέσεις και στο σημείο που μετακινηθήκαμε γράφουμε (αν πρόκειται για `sw`) ή διαβάζουμε (στην περίπτωση της `lw`) την τιμή του `reg1`..

Το παρακάτω πρόγραμμα προσθέτει τους ακεραίους που είναι τοποθετημένοι στις θέσεις 12 και 16 bytes κάτω από τον δείκτη στοίβας `sp` και τοποθετεί το αποτέλεσμα στη θέση 20 bytes κάτω από τον `sp`:

```
lw  t1,-12(sp)
lw  t2,-16(sp)
add t1,t1,t2
sw  t1,-20(sp)
```

Ένας άλλος τρόπος πρόσβασης είναι η πρόσβαση μέσω καταχωρητή, χωρίς να δηλωθεί κάποιο `offset`:

```
lw reg1,(reg2)   # reg1 = [reg2]
    # reg1: destination register
    # reg2: base register
    # offset: distance from reg2
sw reg1,(reg2)   # [reg2] = reg1
    # reg1: source register
    # reg2: base register
    # offset: distance from reg2
```

Ο συμβολισμός `(t0)` είναι ισοδύναμος με τον συμβολισμό `0(t0)`

1.1.4 Εντολές διακλαδώσεων

Στη συνέχεια θα περιγραφούν οι εντολές άλματος που θα χρησιμοποιήσουμε. Η εντολή για άλμα χωρίς συνθήκη είναι η `b` (ή ισοδύναμα η `j`):

```
b label   # reg1 = [reg2]
    # label: an address
```

Το ακόλουθο πρόγραμμα υλοποιεί ατέρμονο βρόχο, αν στο τμήμα με τις τελείες δεν υπάρχει εντολή άλματος ή τερματισμού:

```
mylabel:
  ...
  b mylabel
```

Ενδιαφέρον έχουν οι εντολές αλμάτων υπό συνθήκη, αφού με αυτές υλοποιούμε τις λογικές παραστάσεις.

```
beq reg1,reg2,label    # branch if equal
bne reg1,reg2,label    # branch if not equal
blt reg1,reg2,label    # branch if less than
bgt reg1,reg2,label    # branch if greater than
ble reg1,reg2,label    # branch if less or equal than
bge reg1,reg2,label    # branch if greater or equal than
  # reg1,reg2: the registers to be compared
  # label: the address to jump to
```

Κάθε μία από τις παραπάνω εντολές άλματος υπό συνθήκη, ελέγχει τους δύο καταχωρητές `reg1,reg2` στο πρώτο και στο δεύτερο όρισμά της και ανάλογα με το αποτέλεσμα της σύγκρισης και είδος της συνθήκης που εκφράζει η εντολή, πραγματοποιείται το άλμα στην ετικέτα `label` ή όχι.

Ας δούμε ένα παράδειγμα. Αν θεωρήσουμε ότι στον καταχωρητή `t0` έχουμε την τιμή της διακρίνουσας (και κάνουμε την παραδοχή ότι είναι ακέραια) ενός τριωνύμου, τότε το παρακάτω πρόγραμμα υπολογίζει τις ρίζες του.

```
  beq t0,zero,equal
  bgt t0,zero,positive
negative:
  ...
  b exit
equal:
  ...
  b exit
positive:
  ...
exit:
  ...
```

Θα αναφέρουμε ακόμα ένα είδος άλματος, αυτό που γίνεται μέσω καταχωρητή και αντιστοιχεί στην εντολή `jr`.

```
jr reg   # jump [reg]
  # reg: a register
```

Ας υποθέσουμε ότι στον καταχωρητή `t0` βρίσκεται αποθηκευμένη μία διεύθυνση, για παράδειγμα η διεύθυνση της πρώτης εντολής μίας συνάρτησης ή διαδικασίας. Τότε με την εκτέλεση της

```
jr t0
```

ο έλεγχος θα μεταφερθεί σε αυτή τη συνάρτηση ή διαδικασία.

1.1.5 Κλήση συνάρτησης ή διαδικασίας

Για την κλήση μίας συνάρτησης ή μίας διαδικασίας υποστηρίζεται η εντολή `jal`. Η `jal` παίρνει ως όρισμα μία διεύθυνση και εκτελεί άλμα στην διεύθυνση αυτή. Ταυτόχρονα τοποθετεί στον καταχωρητή `ra` τη διεύθυνση της εντολής που ακολουθεί την `jal` στον υπό μετάφραση κώδικα.

Η εντολή `jal` διευκολύνει στην κλήση συναρτήσεων και διαδικασιών. Αν καλέσουμε την `jal` με την πρώτη εντολή μιας συνάρτησης ή διαδικασίας, τότε ο έλεγχος θα μεταφερθεί σε αυτήν την συνάρτηση ή τη διαδικασία και στον `ra` θα υπάρχει έτοιμη η διεύθυνση στην οποία θα επιστρέψουμε όταν ολοκληρωθεί η εκτέλεση

της κληθείσας. Υπάρχουν, βέβαια και άλλα βήματα που πρέπει να γίνουν κατά την κλήση συναρτήσεων ή διαδικασιών, αλλά η τοποθέτηση της διεύθυνσης επιστροφής στο `ra` είναι σημαντική διευκόλυνση.

Θα ήταν καλύτερο να περιμένουμε για να δούμε κάποιο παράδειγμα στο αντίστοιχο κεφάλαιο (κεφ. **??**)για την παραγωγή τελικού κώδικα για την κλήση συναρτήσεων και διαδικασιών.

1.1.6 Είσοδος και έξοδος δεδομένων

Η είσοδος δεδομένων από το πληκτρολόγιο γίνεται μέσω των καταχωρητών ορισμάτων `a0` και `a7`. Μόλις τα ορίσματα τοποθετηθούν στους δύο καταχωρητές, τότε καλείται η εντολή `ecall`, ώστε να διαβαστούν τα δεδομένα από το πληκτρολόγιο. Στον `a7`, τοποθετείται η τιμή 5, η οποία ορίζει ότι πρόκειται να διαβαστεί ακέραιος αριθμός. Ο ακέραιος που θα διαβαστεί τοποθετείται στον καταχωρητή `a7`. Δίνεται ένα παράδειγμα ανάγνωσης ενός ακέραιου αριθμού, ο οποίος μετά την `ecall` θα βρίσκεται στον `a0`:

```
li a7,5
ecall
```

Για την εμφάνιση στην οθόνη χρησιμοποιούνται πάλι οι ίδιοι καταχωρητές ορισμάτων με διαφορετικό τρόπο, βέβαια. Αν θέλουμε να εμφανίσουμε στην οθόνη έναν ακέραιο αριθμό, τότε τοποθετούμε στον καταχωρητή `a7` τον αριθμό 1 και στον καταχωρητή `a0` τον ακέραιο που θέλουμε να εμφανίσουμε στην οθόνη. Στη συνέχεια καλούμε την `ecall`. Παράδειγμα κώδικα ακολουθεί, με το οποίο το 44 εμφανίζεται στην οθόνη:

```
li a0,44
li a7,1
ecall
```

Θα παρατηρήσει όμως κανείς ότι μετά τον ακέραιο αριθμό δεν υπάρχει αλλαγή γραμμής, κάτι το οποίο μάλλον είναι λογικό. Αν θέλουμε να αλλάξουμε γραμμή, αυτό θα πρέπει να γίνει με τον χαρακτήρα αλλαγής γραμμής:
`n`. Έτσι, θα ορίσουμε ένα συμβολικό όνομα για τον χαρακτήρα αλλαγής γραμμής. Αυτό γίνεται στο χώρο `.data` στην αρχή του προγράμματος:

```
.data
str_nl: .asciz "\n"
```

Στη συνέχεια τοποθετούμε στον καταχωρητή `a7` τον αριθμό 4 και στον καταχωρητή `a0` το συμβολικό όνομα. Ο παρακάτω κώδικας τυπώνει μόνο μία αλλαγή γραμμής:

```
     .data
str_nl: .asciz "\n"
     .text
    la a0,str_nl
    li a7,4
    ecall
```

Θα περιμένουμε λίγο για κάποιο μικρό παράδειγμα, για να ενσωματώσουμε σε αυτό και τον τερματισμό του προγράμματος.

1.1.7 Τερματισμός προγράμματος

Για τον τερματισμό του προγράμματος χρησιμοποιούμε πάλι τους ίδιους καταχωρητές ορισμάτων `a0` και `a7`. Στον καταχωρητή `a7` τοποθετούμε την τιμή 93, ενώ στον καταχωρητή `a0` αυτό που θέλουμε να επιστρέψουμε στο λειτουργικό σύστημα ως αποτέλεσμα. Ένα παράδειγμα τερματισμού εκτέλεσης το οποίο επιστρέφει στο λειτουργικό σύστημα τον αριθμό 0 φαίνεται στη συνέχεια.

```
li a0,0
```

```
    li a7,93
    ecall
```

Ας δούμε, τώρα ένα ολοκληρωμένο παράδειγμα στο οποίο διαβάζουμε έναν ακέραιο αριθμό και τυπώνουμε το διπλάσιο του αριθμού στην οθόνη.

Στον κώδικα, αντί για σχόλια, έχουν τοποθετηθεί ετικέτες οι οποίες στην πραγματικότητα δεν χρειάζονται, αφού κάνεις δεν κάνει εκεί κάποιο άλμα. Μας φάνηκε αρκετά πιο περιγραφικό να τον οργανώσουμε έτσι. Το πρόγραμμα χωρίζεται στο τμήμα `.data` στο οποίο ορί

```
    li a0,0
    li a7,93
    ecall
```

στηκε το συμβολικό όνομα για την αλλαγή γραμμής και το τμήμα `.text` με το κυρίως πρόγραμμα. Η είσοδος του ακεραίου γίνεται στον χώρο που ορίζεται με την ετικέτα `input`. Στην ετικέτα `double_it` μεταφέρεται από τον καταχωρητή `a0` στον καταχωρητή το `t0` ο αριθμός που διαβάστηκε Εκεί ο αριθμός διπλασιάζεται. Κατά την εκτύπωση, στην ετικέτα `print` ο διπλασιασμένος αριθμός μεταφέρεται πίσω στον καταχωρητή `a0` από όπου και εκτυπώνεται ακολουθούμενος από μία αλλαγή γραμμής. Στην ετικέτα `exit` γίνεται η έξοδος από το πρόγραμμα.

```
main:

input:
    li a7,5
    ecall

double_it:
    mv t0,a0
    addi t0,t0,t0

print:
    mv a0,t0
    li a7,1
    ecall
    la a0,str_nl
    li a7,4
    ecall

exit:
    li a0,0
    li a7,93
    ecall
```

1.1.8 Παράδειγμα ταξινόμησης

Θα παρουσιάσουμε ένα ακόμα πρόγραμμα σε assembly, ως περισσότερο αντιπροσωπευτικό, αλλά επίσης και αρκούντως σύντομο. Το πρόγραμμα αυτό δέχεται σαν είσοδο έναν ακέραιο αριθμό N και ταξινομεί τους N πρώτους ακέραιους που βρίσκονται τοποθετημένοι στις N θέσεις πάνω από τον δείκτη στοίβας. Κάθε ακέραιος καταλαμβάνει 4 bytes.

Πρόκειται για μία έκδοση του αλγόριθμου φυσαλίδας (bubble sort). Ο αλγόριθμος υλοποιείται με δύο βρόχους. Ο πρώτο δείκτης υλοποιεί τον εξωτερικό βρόχο και τοποθετείται στον καταχωρητή `s1`. Ξεκινάει

δείχνοντας το N–2 (προτελευταίο)στοιχείο του πίνακα (ετικέτα `init`) και ο βρόχος ολοκληρώνεται όταν ο δείκτης αυτός γίνει αρνητικός (`bge s1,zero,main_loop` στην ετικέτα `loop_conditions`). Ο δείκτης του εσωτερικού βρόχου τοποθετείται στον καταχωρητή `s2` και παίρνει τιμές από 0 (ετικέτα `main_loop`) μέχρι και `s1` (`ble s2,s1,inner_loop`, στην ετικέτα `loop_conditions`).

Στην ετικέτα `inner_loop` φορτώνονται δύο συνεχόμενες θέσεις μνήμης στους καταχωρητές `t1` και `t2` και ελέγχεται αν ο `t1` είναι μεγαλύτερος ή ίσος με τον `t2`, οπότε και γίνεται ένα άλμα το οποίο αποφεύγει την εναλλαγή των τιμών των δύο αυτών θέσεων μνήμης.

Η εναλλαγή, αν χρειαστεί, θα γίνει στην ετικέτα `swap`. Οι εντολές στην ετικέτα `entry` είναι υπεύθυνες για την είσοδο των δεδομένων και εκείνες στην ετικέτα `exit` υπεύθυνες για τον ομαλό τερματισμό του προγράμματος. Το πλήρες πρόγραμμα ακολουθεί:

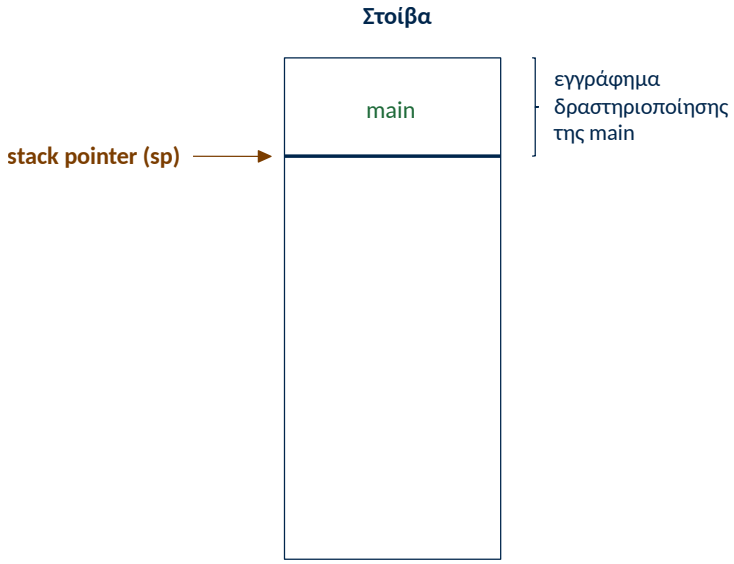
```
entry:

input:
    li a7,5                     # input N
    ecall

init:
    li t0,4                     # initialization of index1(s1)
    mul s1, a0, t0
    addi s1,s1,-8

main_loop:
    li s2,0                     # initialization of index2(s2)

    inner_loop:
        add t0,sp,s2            # check if swap is necessary
        lw t1,(t0)
        lw t2,4(t0)
        bge t1,t2,loop_conditions
    swap:
        sw t1,4(t0)             # swap
        sw t2,(t0)

    loop_conditions:
        addi s2,s2,4            # check loop conditions
        ble s2,s1,inner_loop
        addi s1,s1,-4
        bge s1,zero,main_loop

exit:
    li a0,0                             # the end
    li a7,93
    ecall
```

1.1.9 Το εγγράφημα δραστηριοποίησης

Κάθε συνάρτηση ή διαδικασία που εκτελείται, συμπεριλαμβανομένης και του κυρίως προγράμματος, δεσμεύει έναν χώρο στη στοίβα προκειμένου να τοποθετήσει εκεί δεδομένα ζωτικής σημασίας για τη λειτουργία της, καθώς και τιμές ή διευθύνσεις διαφόρων μεταβλητών. Μία πρώτη συζήτηση για το εγγράφημα δραστηριοποίησης, κυρίως για τις μεταβλητές οι οποίες αποθηκεύονται σε αυτό, είχαμε στο κεφάλαιο του πίνακα

συμβόλων (κεφ. ??). Στην ενότητα αυτή θα δούμε τη δομή ενός εγγραφήματος δραστηριοποίησης, πού τοποθετείται στη στοίβα, ποια η διάρκεια ζωής του και πως συνεισφέρει στη λειτουργία ενός προγράμματος.

Με την εκκίνηση της εκτέλεσης ενός προγράμματος το λειτουργικό σύστημα τού δεσμεύει χώρο ο οποίος θα λειτουργήσει σαν στοίβα. Στο σημείο αυτό τοποθετείται ο *δείκτης στοίβας* `sp` (*stack pointer*). Στη συνέχεια αναλαμβάνει ο κώδικας που παρήχθη από τον μεταγλωττιστή. Ο δείκτης στοίβας μετατοπίζεται τόσες θέσεις όσες θέσεις μνήμης θέλουμε να καταλάβει το εγγράφημα δραστηριοποίησης, δεσμεύοντας, έτσι, χώρο για το κυρίως πρόγραμμα, μέσα στον χώρο που έδωσε για το σκοπό αυτόν στην εφαρμογή το λειτουργικό σύστημα. Θυμίζουμε ότι ο δείκτης στοίβας πρέπει ανά πάσα στιγμή να δείχνει στην αρχή του εγγραφήματος δραστηριοποίησης της συνάρτησης που τη στιγμή αυτή εκτελείται. Η εικόνα της στοίβα είναι τη στιγμή αυτή όπως εικονίζεται στο σχήμα 1.2.



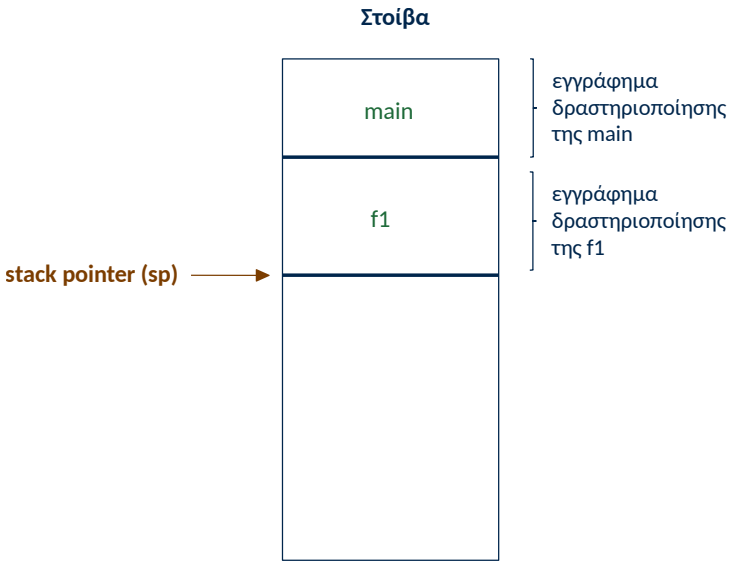
Σχήμα 1.2: Εγγράφημα δραστηριοποίησης του κυρίως προγράμματος στη στοίβα

Αν το κυρίως πρόγραμμα καλέσει τη συνάρτηση `f1`, τότε αυτή θα τοποθετηθεί μετά το εγγραφημα δραστηριοποίησης του κυρίως προγράμματος. Η δέσμευση του χώρου θα γίνει με τη μετατόπιση του `sp`. Το εγγράφημα δραστηριοποίησης του κυρίως προγράμματος εξακολουθεί να υπάρχει στη στοίβα, αφού η εκτέλεσή του δεν έχει ολοκληρωθεί και θα επιστραφεί σε αυτό ο έλεγχος μετά την ολοκλήρωση της εκτέλεσης της `f1`. Η στοίβα έχει τώρα την εικόνα που φαίνεται στο σχήμα 1.3.

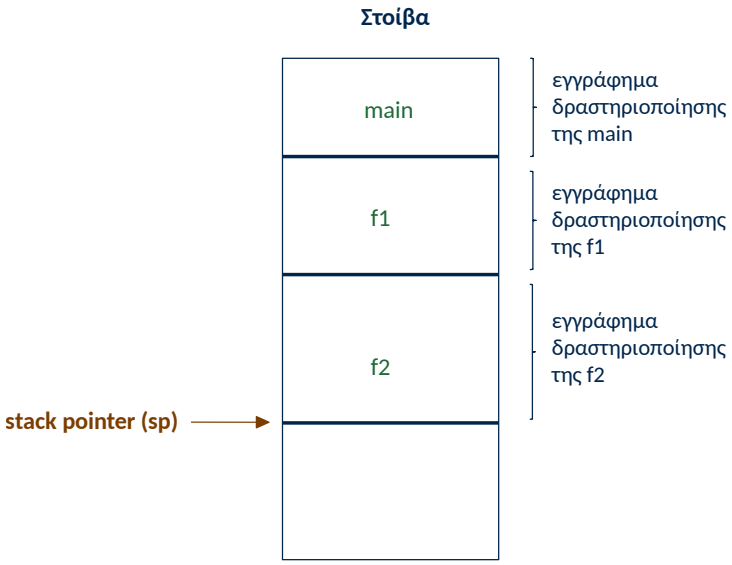
Ας προσθέσουμε ακόμα ένα επίπεδο και ας θεωρήσουμε ότι η συνάρτηση `f1` καλεί τη συνάρτηση `f2`. Με παρόμοιο τρόπο, όπως συζητήσαμε παραπάνω για την `f1`, θα φτάσουμε σε μία είκονα της στοίβας, όπως αυτή εικονίζεται στο σχήμα 1.4.

Ας υποθέσουμε ότι η `f3` δεν καλεί κάποια συνάρτηση ή διαδικασία, οπότε με την ολοκλήρωσή της ο έλεγχος της εκτέλεσης πρέπει να επιστρέψει στην `f2`. Όσον αφορά τη στοίβα, αυτό σημαίνει ότι ο δείκτης στοίβας θα επιστρέψει στην προηγούμενη θέση του και θα δείχνει στην αρχή του εγγραφήματος δραστηριοποίησης της `f2`. Ο χώρος που κατείχε η `f3` θεωρείται πια αποδεσμευμένος. Τα περιεχόμενά του δεν έχουν σβηστεί, απλά θεωρούνται από το σύστημα άχρηστα και επιτρέπεται ο χώρος αυτός να εκχωρηθεί σε άλλη συνάρτηση που πιθανά να κληθεί και να τον χρειαστεί στη συνέχεια, ώστε να τοποθετήσει εκεί δικά της δεδομένα. Χρησιμοποιούμε για τα δεδομένα αυτά τον όρο *σκουπίδια*. Η κατάσταση της στοίβας, όπως είναι μετά την ολοκλήρωση της `f3` φαίνεται στο σχήμα 1.5.

Με παρόμοιο τρόπο θα προστεθούν εγγραφήματα δραστηριοποίησης στη στοίβα κάθε φορά που μία συνάρτηση καλεί μία άλλη και θα επιστρέφεται ο δεσμευμένος χώρος κάθε φορά που μία συνάρτηση ολοκληρώνει την εκτέλεσή της. Όταν ολοκληρωθεί η εκτέλεση του κυρίως προγράμματος, τότε και ο τελευταίος δεσμευμένος χώρος αποδεσμεύεται και στη συνέχεια όλος ο χώρος που καταλάμβανε η εφαρμογή στη στοίβα επιστρέφεται στο λειτουργικό σύστημα για μελλοντική χρήση.



Σχήμα 1.3: Νέο εγγράφημα δραστηριοποίησης στη στοίβα

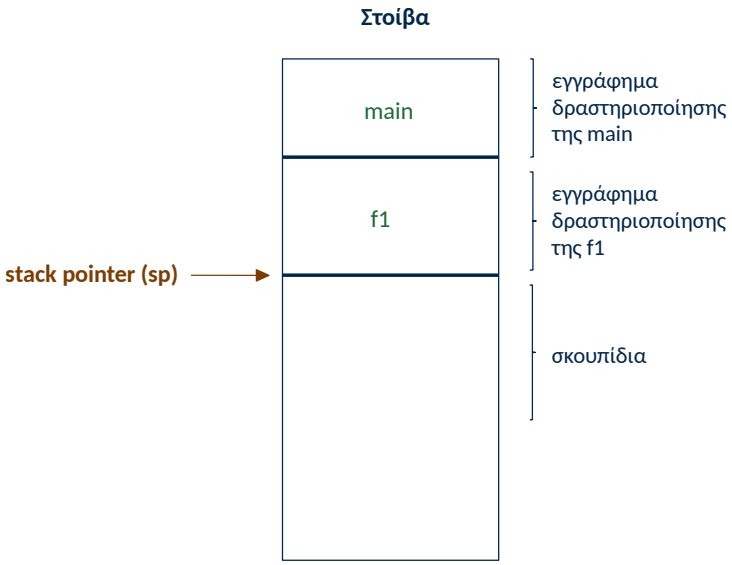


Σχήμα 1.4: Εγγραφήματα δραστηριοποίησης στη στοίβα από φωλιασμένες κλήσεις

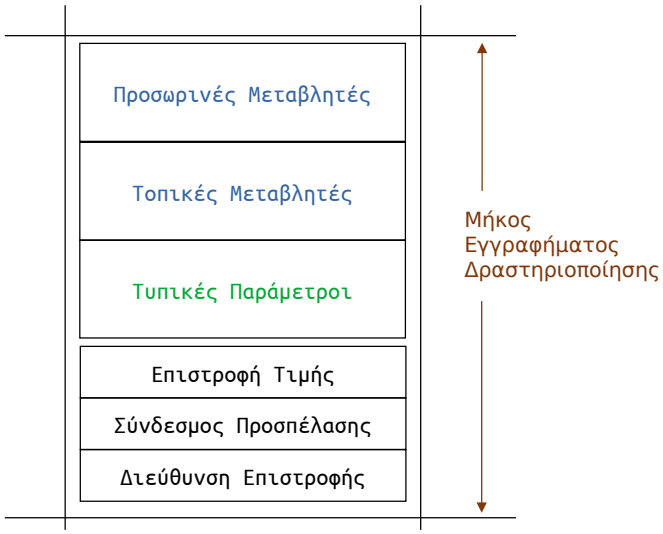
Όλα αυτά δεν γίνονται αυτόματα, αλλά από κώδικα που εμείς θα δημιουργήσουμε στη συνέχεια. Προς το παρόν ας δούμε τι περιέχει μέσα ένα εγγράφημα δραστηριοποίησης.

Δεν είναι η πρώτη φορά που ασχολούμαστε με τα περιεχόμενα ενός εγγραφήματος δραστηριοποίησης. Το έχουμε μελετήσει στο κεφάλαιο του πίνακα συμβόλων (κεφ. **??**). Επαναλαμβάνουμε εδώ το αντίστοιχο σχήμα για διευκόλυνση (1.6). Στο κεφάλαιο εκείνο είχαμε δώσει κύρια βάση στις θέσεις από 12 και επάνω όπου τοποθετούνται μεταβλητές και παράμετροι. Ανάλογα με το είδος της παραμέτρου στο εγγράφημα δραστηριοποίησης μπορεί να είναι τοποθετημένη είτε η τιμή της μεταβλητής που περνάει ως παράμετρος είτε η διεύθυνσή της. Τι περιέχουν, όμως, οι τρεις πρώτες θέσεις του εγγραφήματος δραστηριοποίησης; Περιέχουν πληροφορίες για την επιστροφή της συνάρτησης, μετά την ολοκλήρωσή της, για την σύνδεση με συναρτήσεις προγόνους και για την επιστροφή του αποτελέσματος της συνάρτησης. Κάθε μία τέτοια θέση καταλαμβάνει 4 bytes και για το λόγο αυτόν απαιτούνται συνολικά 12 bytes. Συγκεκριμένα:

- *Διεύθυνση επιστροφής:* η διεύθυνση στην οποία πρέπει να μεταβεί το πρόγραμμα μετά την ολοκλήρωση της εκτέλεσης της συνάρτησης ή της διαδικασίας. Καταλαμβάνει τα 4 πρώτα bytes του εγγραφήματος δραστηριοποίησης



Σχήμα 1.5: Ολοκλήρωση συνάρτησης και επιστροφή δεσμευμένου χώρου στη στοίβα



Σχήμα 1.6: Εγγράφημα δραστηριοποίησης

- *Σύνδεσμος προσπέλασης:* η διεύθυνση του εγγραφήματος δραστηριοποίησης του γονέα της συνάρτησης ή της διαδικασίας. Μέσα από αυτόν τον σύνδεσμο η συνάρτηση ή η διαδικασία μπορεί να προσπελάσει δεδομένα που ανήκουν σε πρόγονούς της. Καταλαμβάνει τα bytes από τη θέση 4 μέχρι και τη θέση 7 του εγγραφήματος δραστηριοποίησης (συνολικά 4 bytes)

- *Επιστροφή τιμής:* η διεύθυνση της μεταβλητής στην οποία επιθυμούμε να γραφεί το αποτέλεσμα της συνάρτησης. Αν πρόκειται για διαδικασία, η θέση αυτή στο εγγράφημα δραστηριοποίησης μένει αχρησιμοποίητη (με σκουπίδια). Καταλαμβάνει τα bytes από τη θέση 8 μέχρι και τη θέση 11 του εγγραφήματος δραστηριοποίησης (συνολικά 4 bytes)

Τα περιεχόμενα των τριών πρώτων θέσεων του εγγραφήματος δραστηριοποίησης συμπληρώνονται κατά την εκτέλεση του προγράμματος από κώδικα τον οποίο παράγει ο μεταγλωττιστής και τον κώδικα αυτόν θα δούμε παρακάτω στο κεφάλαιο αυτό.

1.2 Βοηθητικές συναρτήσεις

Νωρίτερα, στο κεφάλαιο της παραγωγής του ενδιάμεσου κώδικα (κεφ. **??**), ορίσαμε κάποιες βοηθητικές συναρτήσεις οι οποίες μας διευκόλυναν τόσο στον σχεδιασμό του ενδιαμέσου κώδικα, κάνοντάς το σχέδιό περισσότερο σαφές και ευανάγνωστο, όσο και στην υλοποίησή του, μειώνοντας τον όγκο του κώδικα που απαιτείτο να αναπτυχθεί.

Στον τελικό κώδικα θα κάνουμε ακριβώς το ίδιο. Ο σκοπός των βοηθητικών συναρτήσεων παραμένει ο ευκολότερος σχεδιασμός, η ευανάγνωστη περιγραφή και η απλούστευση του τελικού κώδικα. Μόνο που τώρα ο κώδικας που θα γραφεί είναι αρκετά πιο πολύπλοκος, μεγαλύτερος σε όγκο και κρύβει περισσότερη λειτουργικότητα.

Θα αναφέρουμε εδώ τις βοηθητικές συναρτήσεις που χρειαζόμαστε και θα τις δούμε μια προς μία αναλυτικά στη συνέχεια.

- *gnlvcode()*: δημιουργεί τελικό κώδικα για την προσπέλαση πληροφορίας που βρίσκεται αποθηκευμένη στο εγγράφημα δραστηριοποίησης κάποιου προγόνου της συνάρτησης ή της διαδικασίας που αυτή τη στιγμή μεταφράζεται
- *loadvr()*: παράγει τελικό κώδικα ο οποίος διαβάζει μία μεταβλητή που είναι αποθηκευμένη στη μνήμη και την μεταφέρει σε έναν καταχωρητή
- *storerv()*: κάνει την αντίστροφη διαδικασία από το `loadvr`, παράγει τελικό κώδικα ο οποίος αποθηκεύει στη μνήμη την τιμή μιας μεταβλητής η οποία βρίσκεται σε έναν καταχωρητή.

Θα χρησιμοποιήσουμε ακόμα τη βοηθητική συνάρτηση `produce()` η οποία, σε αντιστοιχία με την `genQuad()` που δημιουργούσε μία νέα τετράδα ενδιάμεσου κώδικα, δημιουργεί μία νέα γραμμή (εντολή) τελικού κώδικα. Η δημιουργία του τελικού κώδικα δεν γίνεται σε κάποια δομή στη μνήμη, όπως συνέβαινε με τον ενδιάμεσο κώδικα, αλλά ο κώδικας που παράγεται γράφεται απευθείας σε αρχείο.

1.2.1 Η συνάρτηση `gnlvcode()`

Η πρώτη βοηθητική συνάρτηση είναι η `gnlvcode()`. Η `gnlvcode()` παράγει τελικό κώδικα για την προσπέλαση μεταβλητών ή διευθύνσεων που είναι αποθηκευμένες σε κάποιο εγγράφημα δραστηριοποίησης διαφορετικό από της συνάρτησης που αυτή τη στιγμή μεταφράζεται. Σύμφωνα με τον ορισμό της *C-imple*, κάθε συνάρτηση έχει δικαίωμα να προσπελάσει, πέρα από τις μεταβλητές και διευθύνσεις που είναι αποθηκευμένες στο δικό της εγγράφημα δραστηριοποίησης, μεταβλητές και διευθύνσεις που ανήκουν στο εγγράφημα δραστηριοποίησης κάποιου προγόνου της.

Μέσα στους προγόνους της συνάρτησης βρίσκεται και το κυρίως πρόγραμμα. Συνεπώς, η `gnlvcode()` μπορεί να προσπελάσει και τις καθολικές μεταβλητές. Παρόλο που μπορεί να το κάνει, δεν θα της ζητηθεί ποτέ να το κάνει, αφού, όπως θα δούμε αργότερα, υπάρχει γρηγορότερος τρόπος για να γίνει αυτό. Επειδή η ανάγκη προσπέλασης καθολικών μεταβλητών είναι αρκετά συχνή σε ένα πρόγραμμα, θα προτιμήσουμε να μην χρησιμοποιήσουμε την `gnlvcode()` για την προσπέλαση τους, αλλά θα το κάνουμε με διαφορετικό μηχανισμό, που θα δούμε στις δύο επόμενες βοηθητικές συναρτήσεις, αμέσως στη συνέχεια.

Η `gnlvcode()` παίρνει σαν όρισμα μία μεταβλητή, τη μεταβλητή της οποίας την τιμή ή τη διεύθυνση θέλουμε να προσπελάσουμε:

```
def gnlvcode(v):
```

όπου `v` το όνομα της μεταβλητής.

Το αποτέλεσμα της εκτέλεσης της `gnlvcode()` είναι:

- αν αναζητείται η τιμή μιας μεταβλητής, τότε θα μεταφερθεί στον καταχωρητή `t0` η διεύθυνση της μεταβλητής που αναζητείται

- αν αναζητείται η διεύθυνση μιας μεταβλητής, τότε θα μεταφερθεί στον καταχωρητή `t0` η διεύθυνση μνήμης η οποία περιέχει τη διεύθυνση της μεταβλητής που αναζητείται

Η `gnlvcode()` λειτουργεί ως εξής:

- Αναζητεί στον πίνακα συμβόλων το όνομα της μεταβλητής που της δίνεται σαν παράμετρος (αν το όνομα της μεταβλητής δεν βρεθεί στον πίνακα συμβόλων, τότε ο πίνακας συμβόλων θα επιστρέψει στον χρήστη το κατάλληλο μήνυμα σφάλματος και θα τερματίσει τη μεταγλώττιση)

- Από το επίπεδο στο οποίο βρέθηκε η μεταβλητή η `gnlvcode()` θα συμπεράνει πόσα επίπεδα επάνω στο γενεαλογικό δέντρο της συνάρτησης θα πρέπει να ανέβει προκειμένου να φτάσει στο εγγράφημα δραστηριοποίησης που έχει την πληροφορία που αναζητεί

- Το πρώτο βήμα είναι να μεταβεί στον γονέα της συνάρτησης. Η διεύθυνση του εγγραφήματος δραστηριοποίησης του γονέα βρίσκεται αποθηκευμένη στον σύνδεσμο προσπέλασης της συνάρτησης, στη θέση `-8(sp)` δηλαδή, αφού ο `sp` δείχνει την αρχή του εγγραφήματος δραστηριοποίησης της συνάρτησης που κάθε στιγμή εκτελείται. Θα χρησιμοποιηθεί ένας καταχωρητής, ως καταχωρητής που θα βοηθήσει την `gnlvcode()` να ανέβει τα επίπεδα. Έστω ότι αυτός είναι ο `t0`. Η `gnlvcode()` θα παραγάγει για τον σκοπό αυτόν τον κώδικα:

```
lw mv t0,-8(sp)
```

- Αν η `gnlvcode()` από τον πίνακα συμβόλων έχει συμπεράνει ότι πρέπει να ανέβει n επίπεδα για να εντοπίσει το εγγράφημα δραστηριοποίησης που αναζητεί, τότε έχει ακόμα $n-1$ επίπεδα να ανέβει. Με τον ίδιο τρόπο, όπως και προηγούμένως, θα διαβάσει τον σύνδεσμο προσπέλασης από το εγγράφημα δραστηριοποίησης στο οποίο δείχνει ο `t0` και για κάθε επίπεδο που πρέπει να ανέβει θα παραγάγει την εντολή:

```
lw mv t0τ,-8(0)
```

- Μετά την ολοκλήρωση των μεταβάσεων μέσω των συνδέσμων προσπέλασης ο καταχωρητής `t0` δείχνει την αρχή του εγγραφήματος δραστηριοποίησης το οποίο ο πίνακας συμβόλων υπέδειξε

- Το τελευταίο βήμα είναι να κατέβει ο `t0` κατά `offset` θέσεις ώστε να δείξει στη θέση μνήμης που βρίσκεται η πληροφορία που αναζητείται

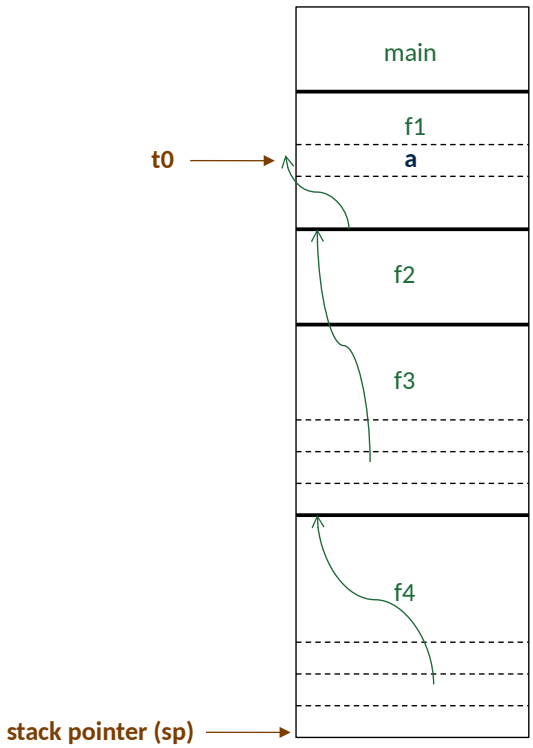
- Το αποτέλεσμα που επιστρέφει η `gnlvcode()` είναι το περιεχόμενο του `t0`

Αν συγκεντρώσουμε τον κώδικα που παράγει η `gnlvcode()` έχουμε:

```
lw t0,-8(sp)
lw t0,-8(t0)
  ...
lw t0,-8(t0)
addi t0,t0,offset
```

Σχηματικά η διαδικασία εικονίζεται στο σχήμα 1.7. Στο σχήμα αυτό η `f4` αναζητεί την τιμή του `a`, η οποία είναι αποθηκευμένη στον παππού της, την `f1`. Μέσω του δικού της συνδέσμου προσπέλασης `4(sp)`, μεταβαίνει στον γονέα της `f3` και τοποθετεί στο εκεί εγγράφημα δραστηριοποίησης τον `t0`. Στη συνέχεια, μέσω του συνδέσμου προσπέλασης του γονέα `4(t0)`, βρίσκει τον παππού `f1` και τοποθετεί τον `t0` στην αρχή του εγγραφήματος δραστηριοποίησης του παππού `f3`. Τέλος, μετατοπίζει τον `t0` κατά `offset` θέσεις, ώστε ο `t0` να λάβει την τελική του θέση δείχνοντας τελικά το `a`.

Στο ίδιο σχήμα θα παρατηρήσετε την ύπαρξη της συνάρτησης `f2` η οποία δεν εμπλέκεται στην αναζήτηση της `a`. Ο λόγος είναι ότι η `f2` δεν αποτελεί πρόγονο της `f4`. Γιατί τότε βρίσκεται στη στοίβα; Μία συνάρτηση



Σχήμα 1.7: Πρόσβαση σε δεδομένα που βρίσκονται σε συναρτήσεις προγόνους, μέσω της `gnvlcode()`

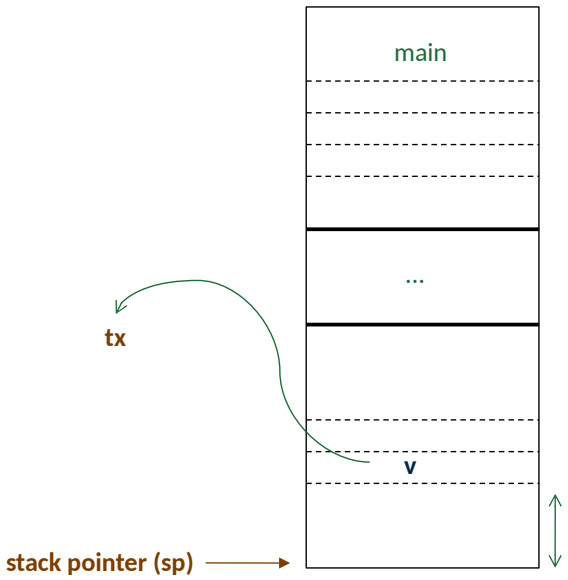
δεν καλείται μόνο από τον γονέα της αλλά μπορεί να κληθεί από αδερφό ή και την ίδια τη συνάρτηση, αναδρομικά. Η κλήση των συναρτήσεων στο παράδειγμά μας έχει γίνει ως εξής: Το κυρίως πρόγραμμα κάλεσε την `f1`, η οποία είναι παιδί της, η `f1` κάλεσε την `f2`, η οποία είναι παιδί της, η `f2` κάλεσε την `f3` η οποία είναι αδελφός της και τέλος η `f3` κάλεσε την `f4`, η οποία είναι παιδί της. Άρα, ενώ η `f2` υπάρχει στην ιεραρχία των κλήσεων, η η `f4` δεν έχει δικαίωμα πρόσβασης στις μεταβλητές της, κάτι που είναι σε συμφωνία και με την περιγραφή της *C-imple*.

Στον παρακάτω κώδικα:

```
function f1()
{
    declare x enddeclare;
    function f2()
    {
        function f3()
        {
                x := 1
        }
    }
}
```

η `gnlvcode(x)` θα τοποθετήσει τον `t0` την διεύθυνση του `x`. Ο πίνακας συμβόλων θα δείξει ότι η `x` βρίσκεται δύο επίπεδα επάνω από το επίπεδο της `f3` και ότι έχει `offset` ίσο με 12. Η `gnlvcode(x)` θα παραγάγει τον ακόλουθο κώδικα:

```
lw t0,-4(sp)
lw t0,-4(t0)
addi t0,t0,-12
```



Σχήμα 1.8: Ανάγνωση μιας τοπικής μεταβλητή ή παραμέτρου που έχει περαστεί με τιμή ή προσωρινής μεταβλητής

1.2.2 Η συνάρτηση `loadvr()`

Η `loadvr()` είναι η συνάρτηση η οποία παράγει τον κώδικα για να διαβαστεί η τιμή μιας μεταβλητής από τη μνήμη, δηλαδή από μία θέση στη στοίβα, και να μεταφερθεί σε έναν καταχωρητή.

Η σύνταξη της `loadvr()` είναι η ακόλουθη:

```
def loadvr(v, reg)
    # v: source variable
    # reg: target register
```

όπου `v` το όνομα της μεταβλητής την τιμή της οποίας που θέλουμε να διαβάσουμε και `reg` το όνομα του καταχωρητή στον οποίο θέλουμε να τοποθετηθεί.

Η `loadvr()` βασίζεται στην πληροφορία που της επιστρέφει ο πίνακας συμβόλων και, ανάλογα με την περίπτωση, παράγει και τον αντίστοιχο κώδικα. Πρόκειται, δηλαδή, για μία μεγάλη δομή πολλαπλής επιλογής (`if...elif...else`). Θα διακρίνουμε τις περιπτώσεις και θα εξετάσουμε τον κώδικα που παράγεται σε κάθε περίπτωση χωριστά.

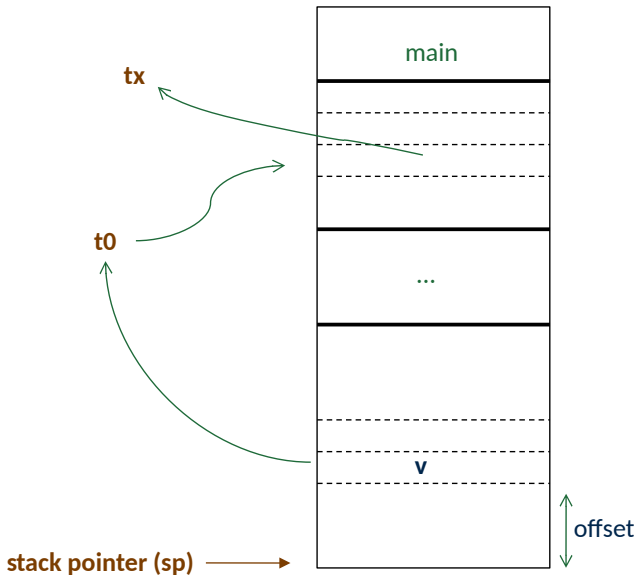
1.2.2.1 Τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή ή προσωρινή μεταβλητή

Πρόκειται για τις περιπτώσεις που η τιμή της μεταβλητής που ζητάμε βρίσκεται αποθηκευμένη στο εγγράφημα δραστηριοποίησης της συνάρτησης που μεταφράζεται. Για να μεταφερθεί μία τέτοια μεταβλητή στον καταχωρητή `reg` παράγεται ο ακόλουθος κώδικας:

```
lw reg,-offset(sp)
```

όπου `offset` το offset της μεταβλητής που επιστρέφει ο πίνακας συμβόλων.

Σχηματικά, η μεταφορά της τιμής μιας τοπικής μεταβλητή ή παραμέτρου που έχει περαστεί με τιμή ή προσωρινής μεταβλητής εικονίζεται στο σχήμα 1.8.



Σχήμα 1.9: Προσπέλαση μιας παραμέτρου που έχει περαστεί με αναφορά

1.2.2.2 Παράμετρος που έχει περαστεί με αναφορά

Όταν μία παράμετρος έχει περαστεί με αναφορά, τότε στο εγγράφημα δραστηριοποίησης της συνάρτησης έχει τοποθετηθεί η διεύθυνσή της. Αυτό δεν γίνεται αυτόματα, θα δούμε πως γίνεται παρακάτω στην κλήση των συναρτήσεων και των διαδικασιών. Για να μεταφερθεί η τιμή μίας τέτοιας μεταβλητής στον καταχωρητή `reg`, απαιτείται ακόμα ένα βήμα. Πρέπει πρώτα να μεταφερθεί η διεύθυνση της μεταβλητής από τη στοίβα σε έναν καταχωρητή, για παράδειγμα τον `t0`, και στη συνέχεια να χρησιμοποιηθεί ο `t0` σαν καταχωρητής δείκτης ώστε να μεταφερθεί στον `reg` η τιμή της μεταβλητής. Για να γίνει αυτό θα πρέπει να παραχθεί ο ακόλουθος κώδικας:

```
lw t0,-offset(sp)
lw reg,(t0)
```

όπου `offset` το offset της μεταβλητής που επιστρέφει ο πίνακας συμβόλων.

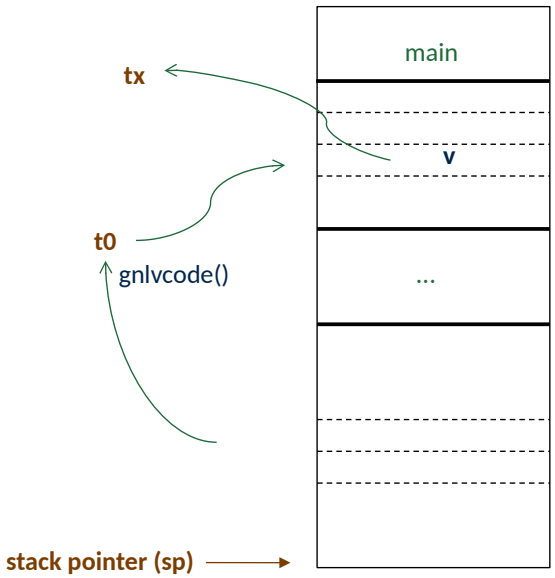
Σχηματικά, η μεταφορά της τιμής μιας παραμέτρου που έχει περαστεί με αναφορά εικονίζεται στο σχήμα 1.9.

1.2.2.3 Τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή η οποία ανήκει σε πρόγονο

Πρόκειται για περιπτώσεις που η τιμή της μεταβλητής που ζητάμε βρίσκεται αποθηκευμένη στο εγγράφημα δραστηριοποίησης κάποιας συνάρτησης πρόγονο της συνάρτησης που μεταφράζεται. Αυτό σημαίνει ότι η πληροφορία που αναζητούμε βρίσκεται σε κάποιο εγγράφημα δραστηριοποίησης στο οποίο μπορούμε να έχουμε πρόσβαση μέσα από μία σειρά ανακτήσεων συνδέσμων προσπέλασης.

Να σημειώσουμε εδώ, ότι όλα τα εγγραφήματα δραστηριοποίησης των προγόνων της συνάρτησης βρίσκονται στη στοίβα, αφού η ολοκλήρωση της εκτέλεσης τους δεν έχει ολοκληρωθεί, δεδομένου ότι κάθε συνάρτηση μπορεί να κληθεί μόνο από γονέα ή από αδελφό. Ή με άλλα λόγια, για να κληθεί μία συνάρτηση, πρέπει να έχει κληθεί ο γονέας της. Ολοκλήρωση της εκτέλεσης του γονέα σημαίνει ότι όλα τα παιδιά του θα έχουν ολοκληρώσει την δική τους εκτέλεση.

Στο σημείο αυτό μπορούμε να εκμεταλλευτούμε την συνάρτηση `gnlvcode()` που φτιάξαμε νωρίτερα. Η `gnlvcode()` θα πάρει σαν παράμετρο τη μεταβλητή που θέλουμε να διαβάσουμε και θα τοποθετήσει στον `t0` την διεύθυνση στην οποία βρίσκεται αποθηκευμένη η μεταβλητή. Στη συνέχεια, δεν έχουμε παρά να διαβάσουμε το περιεχόμενο της θέσης μνήμης:



Σχήμα 1.10: Προσπέλαση τιμής μιας τοπικής μεταβλητή ή παραμέτρου με τιμή και βρίσκεται σε κάποιον πρόγονο

```
gnlvcode()
produce('lw reg,(t0)')
```

Στον παραπάνω συμβολισμό εννοούμε ότι γίνεται κλήση της `gnlvcode()` και στη συνέχεια παράγεται η εντολή `lw reg,(t0)`

Ας δούμε ένα παράδειγμα. Στον παρακάτω κώδικα:

```
function f1(in x)
{
    function f2()
    {
        function f3()
        {
            declare a enddeclare;
            a := x
        }
    }
}
```

με την κλήση της `loadvr(x,t1)` η `gnlvcode()` θα τοποθετήσει τον `t0` την διεύθυνση του `x`. Ο πίνακας συμβόλων θα δείξει ότι η `x` βρίσκεται δύο επίπεδα επάνω από το επίπεδο της `f3` και ότι έχει `offset` ίσο με 12. Στη συνέχεια θα μεταφερθεί το περιεχόμενο της θέσης μνήμης που δείχνει ο `t0` στον καταχωρητή `t1`

```
lw t0,-8(sp)
lw t0,-8(t0)
addi t0,t0,-12
lw t1,(t0)
```

Σχηματικά, η μεταφορά της τιμής μιας τοπικής μεταβλητή ή παραμέτρου με τιμή η οποία βρίσκεται σε πρόγονο της υπό μετάφρασης συνάρτησης ή διαδικασίας εικονίζεται στο σχήμα 1.10.

1.2.2.4 Παράμετρος που έχει περαστεί με αναφορά, η οποία ανήκει σε πρόγονο

Πρόκειται για περιπτώσεις που ζητάμε την τιμή μιας μεταβλητής η οποία όμως έχει περαστεί με αναφορά σε κάποιον πρόγονο της συνάρτησης. Άρα, στο εγγράφημα δραστηριοποίησης του προγόνου βρίσκεται αποθηκευμένη η διεύθυνση της μεταβλητής. Σε αυτό το εγγράφημα δραστηριοποίησης έχουμε πρόσβαση μέσα από μία σειρά ανακτήσεων συνδέσμων προσπέλασης.

Όπως και στην περίπτωση της τοπική μεταβλητής ή της παραμέτρου που έχει περαστεί με τιμή η οποία ανήκει σε κάποιον πρόγονο και την συζητήσαμε νωρίτερα, έτσι και εδώ μπορούμε να εκμεταλλευτούμε την συνάρτηση `gnlvcode()`. Η `gnlvcode()` θα πάρει σαν παράμετρο τη μεταβλητή που θέλουμε να διαβάσουμε και θα τοποθετήσει στον `t0` την διεύθυνση στην οποία βρίσκεται αποθηκευμένη η διεύθυνση της μεταβλητής (και όχι η τιμή της όπως στην προηγούμενη περίπτωση). Χρειαζόμαστε δηλαδή ακόμα ένα βήμα, αφού διαβάσουμε το περιεχόμενο της θέσης μνήμης που τοποθέτησε τον `t0` η `gnlvcode()`, να τη χρησιμοποιήσουμε για να φτάσουμε στην τιμή της μεταβλητής που ζητούμε:

```
gnlvcode()
produce('lw t0,(t0)')
produce('lw reg,(t0)')
```

Ας δούμε ένα παράδειγμα. Στον παρακάτω κώδικα:

```
function f1(inout x)
{
    function f2()
    {
        function f3()
        {
            declare a enddeclare;
            a := x
        }
    }
}
```

Το `x` έχει περαστεί στην `f1` με αναφορά. Το εγγόνι της, η `f3` θέλει να τη διαβάσει. Ο πίνακας συμβόλων δίνει την πληροφορία ότι πρόκειται για παράμετρο με αναφορά, δύο επίπεδα επάνω από το τρέχον βάθος φωλιάσματος και έχει `offset`=12. Άρα μέσω της `gnlvcode()` θα ανέβουμε δύο επίπεδα ώστε μετά τον γονέα να φτάσουμε στον παππού και στη συνέχεια θα μετατοπίσουμε τον `t0` κατά 12 θέσεις:

```
lw t0,-8(sp)
lw t0,-8(t0)
addi t0,t0,-12
```

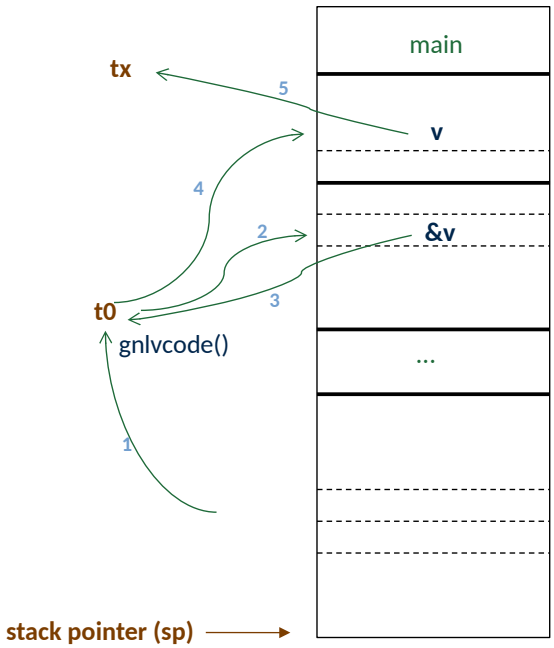
Μας μένει να διαβάσουμε τα περιεχόμενα της θέσης μνήμης που μας υπέδειξε η `gnlvcode()` και, επειδή εκεί είναι αποθηκευμένη η διεύθυνση της μεταβλητής που ζητάμε, να κάνουμε ακόμα ένα βήμα και μέσω της διεύθυνσης να φτάσουμε στη ζητούμενη τιμή:

```
lw t0,(t0)
lw t1,(t0)
```

Ο ολοκληρωμένος κώδικας που θα παραχθεί είναι ο ακόλουθος:

```
lw t0,-8(sp)
lw t0,-8(t0)
addi t0,t0,-12
lw t0,(t0)
lw t1,(t0)
```

Σχηματικά τα παραπάνω εικονίζονται στο σχήμα 1.11.



Σχήμα 1.11: Προσπέλαση τυπικής παραμέτρου που έχει περαστεί με αναφορά και βρίσκεται σε κάποιον πρόγονο

1.2.2.5 Καθολική μεταβλητή

Οι καθολικές μεταβλητές είναι μεταβλητές που έχουν δηλωθεί στο κυρίως πρόγραμμα. Τουλάχιστον στη γλώσσα *C-imple*. Έτσι, μέσω της `gnlvCode()`, μπορούμε να ανέβουμε όσα επίπεδα χρειαστεί και να φτάσουμε στο κυρίως πρόγραμμα, το οποίο είναι πρόγονος κάθε άλλης συνάρτησης στο πρόγραμμα.

Επειδή, όμως, η πρόσβαση στις καθολικές μεταβλητές είναι συνήθως συχνή, ενώ η διαδικασία με τα επίπεδα έχει κάποιο κόστος, ιδιαίτερα αν πρέπει να ανέβουμε περισσότερα επίπεδα. Μια λύση, η οποία επιταχύνει τον χρόνο πρόσβασης είναι να αφιερώσουμε έναν καταχωρητή ο οποίος θα σημειώνει την αρχή του εγγραφήματος δραστηριοποίησης του κυρίως προγράμματος και δεν θα χρησιμοποιηθεί για κάποιο άλλο σκοπό, όσο εκτελείται το πρόγραμμα. Θα επιλέξουμε τον καταχωρητή `gp` (global pointer). Έτσι, όταν σε μία ερώτηση προς τον πίνακα συμβόλων επιστραφεί η απάντηση ότι μία μεταβλητή είναι καθολική ότι έχει ένα συγκεκριμένο `offset`, τότε η πρόσβαση σε αυτήν γίνεται ως εξής:

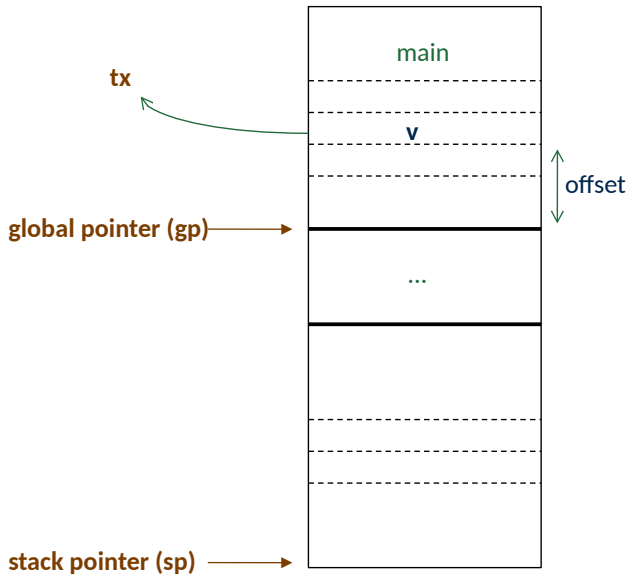
```
lw reg,-offset(gp)
        # reg: register
```

Σχηματικά, ο τρόπος πρόσβασης στις καθολικές μεταβλητές φαίνεται στο σχήμα 1.12. Η μεταβλητή v χαρακτηρίζεται ως καθολική από τον πίνακα συμβόλων και προσπελάζεται μέσω του καταχωρητή `gp` και βρίσκεται `offset` θέσεις πάνω από αυτόν.

Στην περίπτωση που έχουμε μία γλώσσα η οποία διαχωρίζει τις μεταβλητές του κυρίως προγράμματος από τις καθολικές μεταβλητές, η παραπάνω λύση μπορεί να εφαρμοστεί για την πρόσβαση στις καθολικές μεταβλητές, ενώ για την πρόσβαση στις μεταβλητές του κυρίως προγράμματος θα χρησιμοποιηθεί ο μηχανισμός πρόσβασης μέσω των συνδέσμων προσπέλασης.

1.2.2.6 Εκχώρηση αριθμητικής σταθεράς

Μία απλή περίπτωση, η οποία ίσως και να μην ανήκει εδώ, αφού δεν πρόκειται για μετακίνηση από τη μνήμη, αλλά θα τη συζητήσουμε καταχρηστικά και θα την εντάξουμε στην `loadvr()`, είναι η φόρτωση μιας αριθμητικής σταθεράς σε έναν καταχωρητή. Η *C-imple* υποστηρίζει μόνο ακέραιες σταθερές, άρα η μόνη



Σχήμα 1.12: Προσπέλαση καθολικής μεταβλητής

εντολή που χρειαζόμαστε είναι η `li`:

```
li reg, integer
        # reg: register
        # integer: integet arithmetic value
```

Κάπου εδώ ολοκληρώνονται όλες οι περιπτώσεις που πρέπει να φροντίσει η `gnvlCode()`. Ας δοκιμάσουμε να γράψουμε έναν υποτυπώδη ψευδοκώδικα για την `loadvr()`, σε υψηλό επίπεδο:

```
def loadvr(v,reg):
        if v is integer_constant:
        else:
                retrive information for v from symbol table
                if v is global_variable:
                        ... # using gp
                elif v is local_variable or parameter_by_value
                                or temporary_variable:
                        ... # using sp
                elif v is parameter_by_reference:
                        ... # using sp
                elif v is (local_variable or parameter_by_value)
                                in ancestor_function
                        ... # using gnlvCode()
                elif v is parameter_by_reference in ancestor_function
                        ... # using gnlvCode()
```

1.2.3 Η συνάρτηση `storerv()`

Η συνάρτηση `storerv()` δεν διαφέρει πολύ στην υλοποίησή της από την `loadvr()`. Κάθε μεταβλητή θα βρεθεί στη μνήμη με παρόμοιους μηχανισμούς με αυτούς που χρησιμοποιήθηκαν στην `loadvr()`. Η διαφοροποίηση βρίσκεται στις τελευταίες εντολές που παράγει η κάθε κλήση της `storerv()` όπου αντί για εντολή ανάγνωσης (`lw`), έχουμε εντολή αποθήκευσης `sw`.

Η σύνταξη της `storerv()` είναι η ακόλουθη:

```
def storerv(reg,v)
    # reg: source register
    # v: target variable
```

όπου v το όνομα της μεταβλητής την τιμή της οποίας που θέλουμε να διαβάσουμε και reg το όνομα του καταχωρητή στον οποίο θέλουμε να τοποθετηθεί.

Θα δούμε πάλι μία μία τις περιπτώσεις, μόνο που αυτή τη φορά θα είμαστε λιγότερο περιγραφικοί και για κάθε μία περίπτωση θα παραθέσουμε μόνο τον κώδικα που θα παραχθεί, συνοδευόμενο από σχόλια μέσα στον κώδικα.

1.2.3.1 Τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή ή προσωρινή μεταβλητή

```
sw reg,-offset(sp)
    # reg: source register
    # offset (from symbol table - distance from stack pointer)
```

Στο σημείο αυτό, ίσως αναρωτηθεί κανείς για ποιο λόγο συμπεριλαμβάνουμε στην κατηγορία αυτή την παράμετρο που έχει περαστεί με τιμή. Το πέρασμα παραμέτρου με τιμή χρησιμοποιείται για να περαστούν δεδομένα προς την συνάρτηση, ενώ μέσα από αυτή την παράμετρο δεν είναι δυνατόν να επιστραφεί οποιαδήποτε πληροφορία στην καλούσα. Η τιμή της παραμέτρου αυτής βρίσκεται στη στοίβα της κληθείσας συνάρτησης και μετατρέπεται σε *σκουπίδια* μόλις ολοκληρωθεί η εκτέλεσή της και ο δείκτης στοίβας sp μεταφερθεί στο εγγράφημα δραστηριοποίησης της καλούσας. Για όσο καιρό όμως εκτελείται η συνάρτηση, η θέση μνήμης της τυπικής παραμέτρου εξακολουθεί να βρίσκεται δεσμευμένη στη στοίβα και όποιος έχει δικαίωμα πρόσβασης σε αυτή μπορεί να την διαβάσει ή να την τροποποιήσει. Κατά τη διάρκεια, δηλαδή, της εκτέλεσης μιας συνάρτησης ή διαδικασίας η τυπική παράμετρος που έχει περαστεί με αναφορά έχει τη συμπεριφορά μιας τοπικής μεταβλητής η οποία είναι αρχικοποιημένη.

1.2.3.2 Παράμετρος που έχει περαστεί με αναφορά

```
lw t0,-offset(sp)
    # t0 used as temporary register
    # offset (from symbol table - distance from stack pointer)
sw reg,(t0)
    # reg: source register
    # t0 used as index register
```

1.2.3.3 Τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή η οποία ανήκει σε πρόγονο

```
gnlvcode()
    # t0 has the address of the target variable
produce('sw reg,(t0)')
    # reg: source register
    # t0 used as index register
```

Το ίδιο σχόλιο που έγινε για την εγγραφή μίας τιμής σε μία τυπική παραμέτρου που έχει περαστεί με τιμή στην κληθείσα συνάρτηση, ισχύει και για την περίπτωση που η τυπική παράμετρος έχει περαστεί με τιμή σε κάποιο πρόγονο.

1.2.3.4 Παράμετρος που έχει περαστεί με αναφορά, η οποία ανήκει σε πρόγονο

```
call gnlvcode()
    # t0 has the address of the address of the target variable
produce('lw t0,(t0)')
    # t0 has the address pf the target variable
produce('sw reg,(t0)')
produce('sw reg,(t0)')
    # reg: source register
    # t0 used as index register
```

1.2.3.5 Καθολική μεταβλητή

```
    sw reg,-offset(gp)
    # reg: source register
    # offset (from symbol table - distance from global pointer)
```

1.2.3.6 Αριθμητική σταθερά

Εδώ υπάρχει μια μικρή διαφοροποίηση με την αντίστοιχη περίπτωση της `loadvr()`, αφού η αριθμητική σταθερά πρέπει πρώτα να φορτωθεί σε έναν καταχωρητή (πχ. τον `t0`) και στη συνέχεια από εκεί να τοποθετηθεί στην μεταβλητή στόχο:

```
loadvr(1,t0)
storerv(t0,z)
```

Πίνακας 1.1: Μονάδες υλικού/λογισμικού που χρησιμοποιούνται από τις `loadvr()` και `storerv()`

	χρήση βασικού δείκτη	χρήση συνδέσμου προσπέλασης	χρήση συνάρτησης `gnlvcode()`	χρήση καταχωρητή δείκτη `t0`
τοπική μεταβλητή	**sp**	όχι	όχι	όχι
παράμετρος με τιμή	**sp**	όχι	όχι	όχι
προσωρινή μεταβλητή	**sp**	όχι	όχι	όχι
παράμετρος με αναφορά	**sp**	όχι	όχι	ναι
τοπική μεταβλητή σε πρόγονο	**sp**	ναι	ναι	ναι
παράμετρος με τιμή σε πρόγονο	**sp**	ναι	ναι	ναι
παράμετρος με αναφορά σε πρόγονο	**sp**	ναι	ναι	ναι
καθολική μεταβλητή	**gp**	όχι	όχι	όχι
αριθμητική σταθερά	όχι	όχι	όχι	όχι

Στον πίνακα 1.1 έχουν συγκεντρωθεί για κάθε περίπτωση τα αγαθά χρησιμοποιεί η κάθε μεταβλητή ώστε να προσπελαστεί από τις `loadvr()` και `storerv()`. Χρησιμοποιήστε τον πίνακα για να βεβαιωθείτε ότι έχετε κατανοήσει την υλοποίηση της κάθε περίπτωσης και χωρίς να γυρίσετε πίσω στο διάβασμα, προσπαθήστε μέσα από αυτόν να επαναφέρετε τις υλοποιήσεις στη μνήμη σας.

Να σημειώσουμε ότι στην `storerv()` ο καταχωρητή `t0` χρησιμοποιείται για την προσωρινή αποθήκευση της αριθμητικής σταθεράς, αλλά όχι σαν καταχωρητής δείκτης. Για τον λόγο αυτό δεν σημειώνεται στην αντίστοιχη στήλη του πίνακα 1.1.

1.3 Εκχώρηση και αριθμητικές πράξεις

Στην ενότητα αυτή θα χρησιμοποιήσουμε τις βοηθητικές συναρτήσεις που φτιάξαμε παραπάνω για να παράγουμε τελικό κώδικα για τις αριθμητικές πράξεις και τις εκχωρήσεις.

Η εντολή του ενδιάμεσου κώδικα που εκχωρεί μία μεταβλητή σε μία άλλη θυμίζουμε ότι είναι η εξής:

```
:=, x, _, z     # z:=x
```

Έχοντας διαθέσιμες τις `loadvr()` και `storerv()` είναι πολύ εύκολο να δημιουργήσουμε τον κώδικα που απαιτείται. Αρκεί να φορτώσουμε την μεταβλητή `x` σε έναν καταχωρητή (πχ τον `t0`) και στη συνέχεια από εκεί να τον μεταφέρουμε στη θέση μνήμης που η βρίσκεται η μεταβλητή `z`. Τόσο η `loadvr()` όσο και η `storerv()` είναι φτιαγμένες έτσι ώστε να καλύπτουν όλες τις περιπτώσεις και δεν υπάρχει λόγος να προβληματιζόμαστε πια για την ορθή μεταφοράς της τιμής μιας μεταβλητής σε έναν καταχωρητή ή το αντίστροφο. Έτσι για να παραχθεί ο ζητούμενος κώδικας πρέπει να καλέσουμε αρχικά την `loadvr()` και μετά την `storerv()` με τις κατάλληλες παραμέτρους:

```
loadvr(x,t0)
storerv(t0,z)
```

Στην περίπτωση που έχουμε εκχώρηση αριθμητικής σταθεράς σε μεταβλητή, τότε απλά αντικαθιστούμε την μεταβλητή `x` με την αριθμητική σταθερά. Έχουμε φροντίσει στην `storerv()` ώστε να λειτουργεί αυτό σωστά.

```
loadvr(integer,t0)
storerv(t0,z)
```

Η *C-imple* υποστηρίζει τις τέσσερις αριθμητικές πράξεις.

```
op, x, y, z     # z = x op y
# op: +, -, * , /
# x, y, z: variables
```

Για κάθε μία από αυτές, οι δύο μεταβλητές `x` και `y` μεταφέρονται αντίστοιχα στους καταχωρητές `t1` και `t2`, μέσω της `loadvr()`. Στη συνέχεια γίνεται η πράξη add,textttsub,**mul**,textttdiv, ανάλογα την πράξη μου θέλουμε να εκτελεστεί. Σαν παράμετροι στις εντολές αυτές θα χρησιμοποιηθούν οι `t1` και `t2` σαν καταχωρητές πάνω στους οποίους θα εκτελεστεί η πράξη και ο `t1` σαν καταχώρητης στον οποίο θα τοποθετηθεί το αποτέλεσμα. Απομένει μόνο η μεταφορά του αποτελέσματος από τον καταχωρητή `t1` στην μεταβλητή `z`:

Για την εντολή ενδιάμεσου κώδικα:

```
+, x, y, z     # z = x + y
```

θα γίνουν οι εξής κλήσεις:

```
loadvr(x,t1)
loadvr(y,t2)
produce('add t1,t2,t1')
storerv(t1,z)
```

1.4 Διακλαδώσεις

Η αποσύνθεση ενός προγράμματος σε τμήματα κώδικα και διακλαδώσεις που μεταφέρουν τον κώδικα από τμήμα σε τμήμα έγινε κατά την παραγωγή του ενδιάμεσου κώδικα. Στη φάση της παραγωγής του τελικού κώδικα κάθε μία από τις εντολές αυτές πρέπει να παραγάγει έναν σχετικά από κώδικα ο οποίος θα εκτελεί ένα απλό άλμα στην περίπτωση της εντολής `jump` του ενδιάμεσου κώδικα, ενώ ένα λογικό άλμα θα ακουλουθεί τη σύγκριση δύο μεταβλητών στις περιπτώσεις των λογικών αλμάτων.

Έτσι, για την εντολή ενδιάμεσου κώδικα:

```
jump, _, _, label
```

θα παραχθεί η εντολή:

```
j label
```

Εάν πρόκειται για το λογικό άλμα:

```
cond_jump_int_code, x, y, label
```

θα εκτελεστούν οι παρακάτω βοηθητικές συναρτήσεις:

```
loadvr(x,t1)
loadvr(y,t2)
produce('cond_jump_fin_code t1,t2,label')
```

όπου η προφανής αντιστοιχία ανάμεσα στα `cond_jump_int_code` και `cond_jump_fin_code` φαίνεται στον πίνακα 1.2

Πίνακας 1.2: Αντιστοίχιση λογικών αλμάτων στον ενδιάμεσο και στον τελικό κώδικα

`cond_jump_int_code`	`cond_jump_fin_code`
==	beq
<>	bne
<	blt
>	bgt
<=	ble
>=	bge

1.5 Αρχή προγράμματος, κυρίως πρόγραμμα και τέλος προγράμματος

Εκκινώντας ένα πρόγραμμα, αυτό που πρέπει να εκτελεστεί είναι η πρώτη εντολή του κυρίως προγράμματος. Αυτή όμως δεν συμπίπτει με την πρώτη εντολή του τελικού κώδικα που έχει παραχθεί, αφού η μετάφραση ενός προγράμματος ακολουθεί τη σειρά εμφάνισης των εντολών στο αρχικό πρόγραμμα. Έτσι, οι εντολές του κυρίως προγράμματος θα μεταφραστούν τελευταίες, μετά την μετάφραση όλων των συναρτήσεων και διαδικασιών.

Με την εκκίνηση του προγράμματος ο κώδικας πρέπει να εκτελέσει ένα άλμα στην πρώτη εντολή του κυρίως προγράμματος. Άρα η πρώτη εντολή που θα παραχθεί θα είναι μία `jump`. Η ετικέτα της πρώτης εντολής του κυρίως προγράμματος δεν είναι ακόμα γνωστή, μπορούμε όμως να τοποθετήσουμε μία ετικέτα πριν από αυτήν και να κάνουμε το άλμα στην ετικέτα αυτή. Έτσι, αν ονομάσουμε την ετικέτα `main` η πρώτη εντολή κάθε προγράμματος θα είναι

```
L0:
    j main
```

Όταν μεταβούμε στην ετικέτα `main` τότε θα πρέπει να κάνουμε δύο ενέργειες για να αρχικοποιήσουμε δύο καταχωρητές, τον δείκτη στοίβας `sp` και τον καταχωρητή για τις καθολικές μεταβλητές `gp`.

Ο καταχωρητής `sp` δείχνει στην αρχή του χώρου που μας παραχωρήθηκε για τη στοίβα από το λειτουργικό σύστημα. Πρέπει να τον τοποθετήσουμε στην αρχή του εγγραφήματος δραστηριοποίησης του κυρίως προγράμματος. Άρα πρέπει να μεταφερθεί προς τα πάνω, τόσα bytes, όσο το εγγράφημα δραστηριοποίησης του κυρίως προγράμματος. Εκεί θα τοποθετηθούν οι καθολικές μεταβλητές, άρα εκεί τα τοποθετηθεί και ο `gp`

Άρα στην ετικέτα `main` θα υπάρχουν οι εξής δύο εντολές:

```
main:
    addi sp,sp,framelength_main
    mv gp,sp
L… :
```

Μετά ακολουθεί η ετικέτα της πρώτης εκτελέσιμης εντολής του κυρίως προγράμματος `L…` , όποια κι αν είναι αυτή.

Όταν ολοκληρωθεί η παραγωγή κώδικα για όλες τις εντολές του προγράμματος, θα φτάσουμε και στην τελευταία εντολή, η οποία είναι η `halt`. Με την `halt` θα επιστραφεί ο έλεγχος στο λειτουργικό σύστημα. Είδαμε στην ενότητα 1.1.7 ότι ο τερματισμός προγράμματος, επιστρέφοντας στο λειτουργικό σύστημα τον αριθμό 0, γίνεται με τις εντολές:

```
    li a0,0
    li a7,93
    ecall
```

Συνεπώς, το παράδειγμα `fibonacci.ci` του κεφαλαίου **??** θα παραγάγει τον εξής κώδικα:

```
L0:
    j main
L1:
    # code for fibonacci function
    …
main:
    addi sp,sp,framelength_main
    mv gp,sp
L… :
    # code for main program
    …
    #  code for halt,_,_,_
    li a0,0
    li a7,93
    ecall
```

1.6 Πέρασμα παραμέτρων

Οι παράμετροι μιας συνάρτησης θα τοποθετηθούν στο εγγράφημα δραστηριοποίησης αμέσως επάνω από τα 12 δεσμευμένα bytes για την διεύθυνση επιστροφής, τον σύνδεσμο προσπέλασης και την διεύθυνση επιστροφής τιμής της συνάρτησης. Η σειρά τοποθέτησης είναι αυτή της εμφάνισής τους.

Για διευκόλυνσή μας θα χρησιμοποιήσουμε τον καταχωρητή `fp` σαν δείκτη στο νέο εγγράφημα δραστηριοποίησης. Θα τοποθετήσουμε, δηλαδή, τον `fp` στην αρχή του εγγραφήματος δραστηριοποίησης της νέας συνάρτησης ή διαδικασίας, εκεί που θα τοποθετηθεί ο `sp` όταν ξεκινήσει η εκτέλεση της. Έτσι θα δημιουργήσουμε εύκολη πρόσβαση στο υπό δημιουργία εγγράφημα δραστηριοποίησης.

Πριν κάνουμε, οποιεσδήποτε ενέργειες για το πέρασμα της πρώτης παραμέτρου, τοποθετούμε τον `fp` στη θέση του. Προσθέσουμε στον `sp` τόσα bytes από όσα αποτελείται το εγγράφημα δραστηριοποίησης της καλούσας. Έτσι υπολογίζουμε τη διεύθυνση του τέλους του εγγραφήματος δραστηριοποίησης της καλούσας, το σημείο δηλαδή στο οποίο θα τοποθετηθεί το εγγράφημα δραστηριοποίησης της κληθείσας. Για το σκοπό αυτόν παράγουμε την εντολή:

```
addi fp,sp,framelength
# framelength is the size of the activation record
# of the calling procedure/function
```

Η παραπάνω εντολή θα δημιουργηθεί όταν συναντάται η πρώτη `par` στον ενδιάμεσο κώδικα. Αν η κληθείσα δεν έχει παραμέτρους, τότε δεν υπάρχει κάποια `par` που να προηγείται της `call` και η εντολή `addi` θα δημιουργηθεί όταν εμφανιστεί η `call`.

Μετά την τοποθέτηση του `fp` στην αρχή του υπό δημιουργία εγγραφήματος δραστηριοποίησης, για κάθε παράμετρο που συναντάμε (`par`), και ανάλογα με το αν αυτή περνά με τιμή ή αναφορά, κάνουμε τις ανάλογες ενέργειες οι οποίες αναλύονται στις ακόλουθες ενότητες.

1.6.1 Πέρασμα παραμέτρων με τιμή

Κατά το πέρασμα μιας παραμέτρου με τιμή, η τιμής της παραμέτρου αντιγράφεται στο εγγράφημα δραστηριοποίησης της υπό δημιουργίας συνάρτησης, στη θέση που έχει δεσμευτεί για την παράμετρο αυτή.

Αρχικά, η τιμή της παραμέτρου θα αναζητηθεί και θα τοποθετηθεί προσωρινά σε έναν καταχωρητή. Η αναζήτηση και η τοποθέτηση της παραμέτρου σε καταχωρητή μπορεί να γίνει χρησιμοποιώντας τη βοηθητική συνάρτηση `loadvr()`. Η `loadvr()` θα αντλήσει την απαιτούμενη πληροφορία από τον πίνακα συμβόλων και θα παραγάγει τον κώδικα που θα αναζητήσει την τιμή της παραμέτρου, είτε τοπικά, είτε στο επίπεδο των καθολικών μεταβλητών, είτε σε επίπεδα προγόνων της υπό δημιουργίας συνάρτησης. Με την ολοκλήρωση της `loadvr()` ο κώδικας που θα έχει παραχθεί θα τοποθετεί, όταν εκτελεστεί, την τιμή της ζητούμενης παραμέτρου σε κάποιον καταχωρητή. Έστω ότι ο καταχωρητής που επιλέξαμε ως ενδιάμεσος είναι ο καταχωρητής `t0`.

Το επόμενο βήμα είναι η αντιγραφή της τιμής της παραμέτρου από τον `t0` στην κατάλληλη θέση στη στοίβα. Για τον υπολογισμό της θέσης της στη στοίβα υπολογίζεται ο χώρος που έχουν καταλάβει οι παράμετροι που έχουν ήδη τοποθετηθεί εκεί. Η νέα παράμετρος θα καταλάβει τις επόμενες διαθέσιμες θέσεις.

Επειδή στην *C-imple* έχουμε μόνο ακέραιες μεταβλητές, κάθε παράμετρος καταλαμβάνει 4 bytes. Άρα η i-οστή παράμετρος μίας συνάρτησης θα έχει offset:

$$d = 12 + (i - 1) * 4bytes \tag{1.1}$$

Θυμίζουμε ότι στα 12 πρώτα bytes του εγγραφήματος δραστηριοποίησης αποθηκεύεται πληροφορία απαραίτητη για τη λειτουργία της συνάρτησης. Θα δούμε πιο αναλυτικά παρακάτω, στο κεφάλαιο αυτό, τι ακριβώς αποθηκεύεται εκεί και για πιο λόγο.

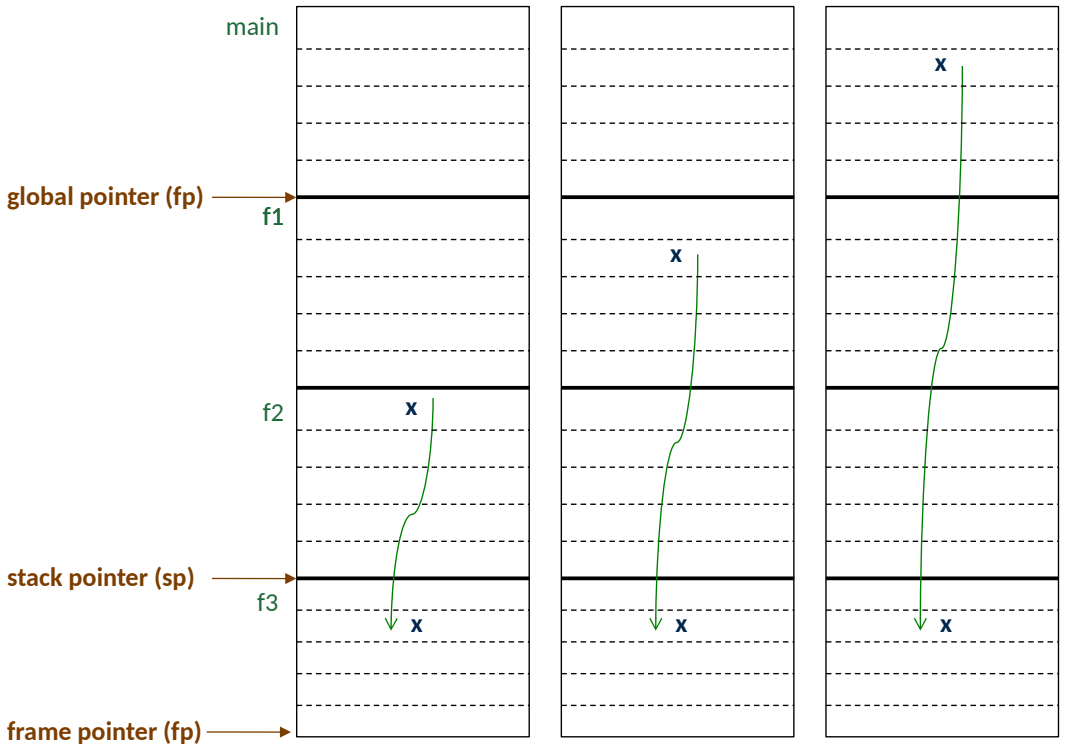
Η εντολή που θα αντιγράψει την τιμή της παραμέτρου από τον καταχωρητή `t0` στην κατάλληλη θέση στη στοίβα είναι η εξής:

```
sw t0, -d(fp)
```

όπου το d δίνεται από εξίσωση 1.2 και είναι ακέραιος αριθμός. Θυμίζουμε ότι έχουμε ήδη μεταφέρει τον `fp` στην αρχή του εγγραφήματος δραστηριοποίησης της καλούσας συνάρτησης.

Στο σχήμα 1.13 φαίνονται τρεις περιπτώσεις περάσματος παραμέτρου με τιμή, ανάλογα με το που βρίσκεται η παράμετρος που θα περαστεί. Η συνάρτηση `f2` είναι αυτή που τη στιγμή αυτή εκτελείται. Γι' αυτό και ο stack pointer δείχνει στο εγγράφημα δραστηριοποίησης της `f2`. Η `f2` έχει κληθεί από την `f1`, η οποία με τη σειρά της έχει κληθεί από την `main`. Η `f2` δημιουργεί την `f3`, ώστε να της περάσει τον έλεγχο μόλις ολοκληρωθεί η δημιουργία του εγγραφήματος δραστηριοποίησης. Για το λόγο αυτό ο `fp` έχει τοποθετηθεί στην αρχή του εγγραφήματος δραστηριοποίησης της `f3` και μέσω του `fp` έχουμε εύκολη πρόσβαση σε αυτό. Υποθέτουμε ότι θέλουμε να περάσουμε σαν παράμετρο την τιμή της `x` και ότι η `x` είναι η πρώτη παράμετρος της `f3`. Άρα η `x` θα τοποθετηθεί στη θέση $d = 12$ του εγγραφήματος δραστηριοποίησης της `f3`.

Όπως φαίνεται στις τρεις περιπτώσεις του σχήματος **??** η παράμετρος μπορεί να βρίσκεται στο εγγράφημα δραστηριοποίησης της καλούσας συνάρτησης (αριστερά στο σχήμα), σε κάποιον πρόγονο (στη μέση) ή μπορεί να αποτελεί καθολική μεταβλητή (δεξιά). Μία μεταβλητή μπορεί να βρεθεί στο εγγράφημα δραστηριοποίησης της συνάρτησης που εκτελείται αν πρόκειται για τοπική μεταβλητή, παράμετρο που έχει περαστεί με τιμή στην συνάρτηση που εκτελείται ή αποτελεί προσωρινή μεταβλητή. Μία μεταβλητή μπορεί να βρεθεί στο εγγράφημα δραστηριοποίησης μίας συνάρτησης προγόνου αν αποτελεί τοπική μεταβλητή ή παράμετρο που



Σχήμα 1.13: Πέρασμα παραμέτρων με τιμή. Αριστερά όταν η παράμετρος βρίσκεται στο εγγράφημα δραστηριοποίησης της καλούσας. Στη μέση όταν βρίσκεται σε κάποιον πρόγονο. Δεξιά, όταν αποτελεί καθολική μεταβλητή

έχει περαστεί με τιμή στην συνάρτηση πρόγονο. Μία μεταβλητή είναι καθολική όταν έχει δηλωθεί στο κυρίως πρόγραμμα.

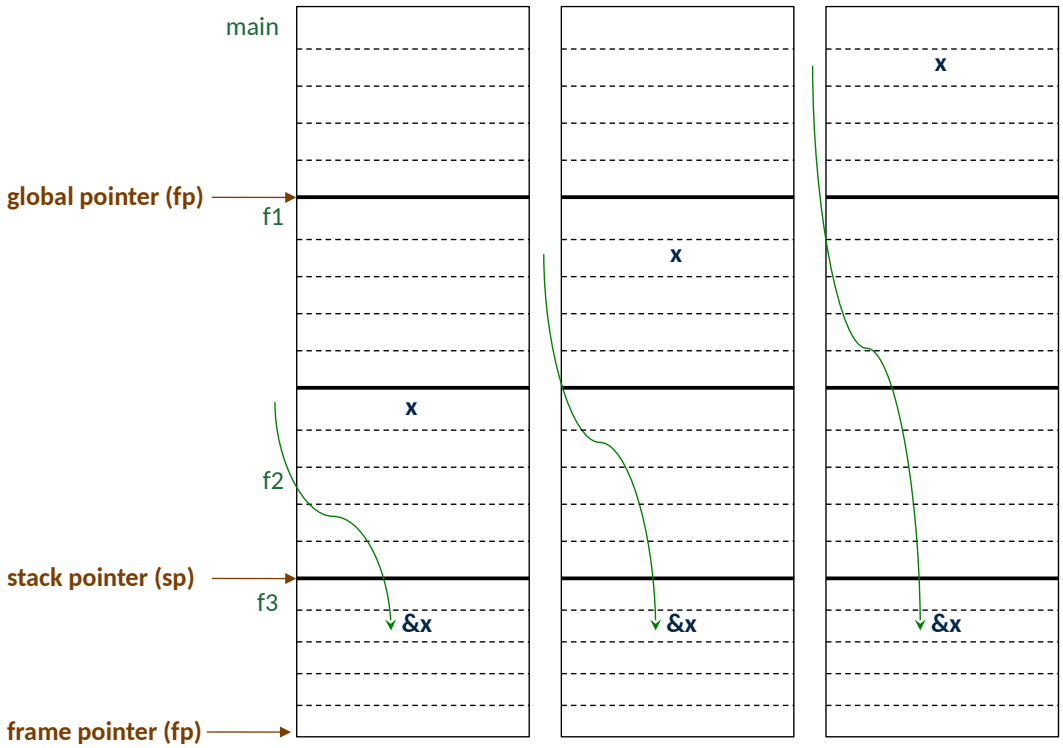
1.6.2 Πέρασμα παραμέτρων με αναφορά

Κατά το πέρασμα παραμέτρων με τιμή αντιγράφεται στο εγγράφημα δραστηριοποίησης της υπό δημιουργίας συνάρτησης η τιμή της μεταβλητής που περνάμε ως παράμετρο. Κατά το πέρασμα παραμέτρων με αναφορά αντιγράφεται στο εγγράφημα δραστηριοποίησης της υπό δημιουργίας συνάρτησης η διεύθυνση της μεταβλητής αυτής. Η πρόσβαση στη μεταβλητή που περάστηκε ως παράμετρος γίνεται μέσω αυτής της διεύθυνσης.

Ο μηχανισμός περάσματος είναι πρακτικά αυτός που ακολουθούμε για να περάσουμε μία παράμετρο με αναφορά στη γλώσσα προγραμματισμού C. Το πέρασμα με αναφορά γίνεται με τον ίδιο τρόπο σε όλες τις γλώσσες προγραμματισμού, με την μόνη διαφορά ότι στις υπόλοιπες γλώσσες προγραμματισμού οι λεπτομέρειες υλοποίησης κρύβονται από τον προγραμματιστή, ενώ στη C η διαχείριση/υλοποίηση είναι ευθύνη του προγραμματιστή.

Η παραγωγή κώδικα για το πέρασμα παραμέτρου με τιμή είναι κάτι σχετικά απλό όσον αφορά την υλοποίησή της, δεδομένου ότι είχε ήδη υλοποιηθεί η συνάρτηση `loadvr()`, η οποία παρήγαγε μέρος του κώδικα και απλοποίησε σημαντικά τη διαδικασία. Στο πέρασμα παραμέτρου με αναφορά, η υλοποίηση απαιτεί περισσότερη προσπάθεια, αφού δεν έχουμε υλοποιήσει κάποια αντίστοιχη συνάρτηση που να μας τοποθετεί τη διεύθυνση μιας μεταβλητής σε κάποιον καταχωρητή. Έτσι, θα πρέπει να υλοποιήσουμε ολόκληρη την παραγωγή του κώδικα και να διαχωρίσουμε περιπτώσεις ανάλογα με αυτό που μας επιστρέφει ο πίνακας συμβόλων.

Θα διαχωρίσουμε δύο βασικές περιπτώσεις. Στην πρώτη, στη θέση της στοίβας που μας παραπέμπει ο πίνακας συμβόλων βρίσκεται η τιμή της μεταβλητής που θέλουμε να περάσουμε σαν παράμετρο. Στη δεύτερη, στη θέση αυτή, βρίσκεται η διεύθυνσή της.



Σχήμα 1.14: Πέρασμα παραμέτρων με αναφορά, όταν στο εγγράφημα δραστηριοποίησης είναι αποθηκευμένη η τιμή της μεταβλητής. Αριστερά όταν η παράμετρος βρίσκεται στο εγγράφημα δραστηριοποίησης της καλούσας. Στη μέση όταν βρίσκεται σε κάποιον πρόγονο. Δεξιά, όταν αποτελεί καθολική μεταβλητή

Όταν περνάμε μία παράμετρο, στη στοίβα της νέας συνάρτησης ή διαδικασίας τοποθετείται είτε η τιμή της παραμέτρου, είτε η διεύθυνσή της. Αν πρόκειται για την τιμή, επειδή στην *C-imple* έχουμε μόνο ακέραιες μεταβλητές, κάθε παράμετρος καταλαμβάνει στη στοίβα 4 bytes. Αν πρόκειται για διεύθυνση χρειαζόμαστε πάλι 4 bytes, αφού και μία διεύθυνση χρειάζεται 4 bytes για να αποθηκευτεί. Έτσι, η i-οστή παράμετρος μίας συνάρτησης ή διαδικασίας θα απέχει από την αρχή του εγγραφήματος δραστηριοποίησης (offset):

$$d = 12 + (i - 1) * 4bytes \tag{1.2}$$

1.6.2.1 Παράμετρος με αναφορά, όταν στη στοίβα υπάρχει αποθηκευμένη η τιμή της παραμέτρου

Πρόκειται για τις περιπτώσεις που η μεταβλητή που περνιέται ως παράμετρος είναι α) τοπική μεταβλητή ή προσωρινή μεταβλητή ή παράμετρος που έχει περαστεί με τιμή στη συνάρτηση που εκτελείται, β) τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή σε συνάρτηση πρόγονο ή γ) καθολική μεταβλητή. Για κάθε μία από τις περιπτώσεις αυτές δημιουργούμε διαφορετικό κώδικα, αλλά το ζητούμενο είναι πάντα το ίδιο: η διεύθυνση της μεταβλητής που περνιέται ως παράμετρος θα πρέπει να αντιγραφεί στην κατάλληλη θέση του εγγραφήματος δραστηριοποίησης που δημιουργείται. Ας δούμε τις περιπτώσεις αυτές περισσότερο αναλυτικά.

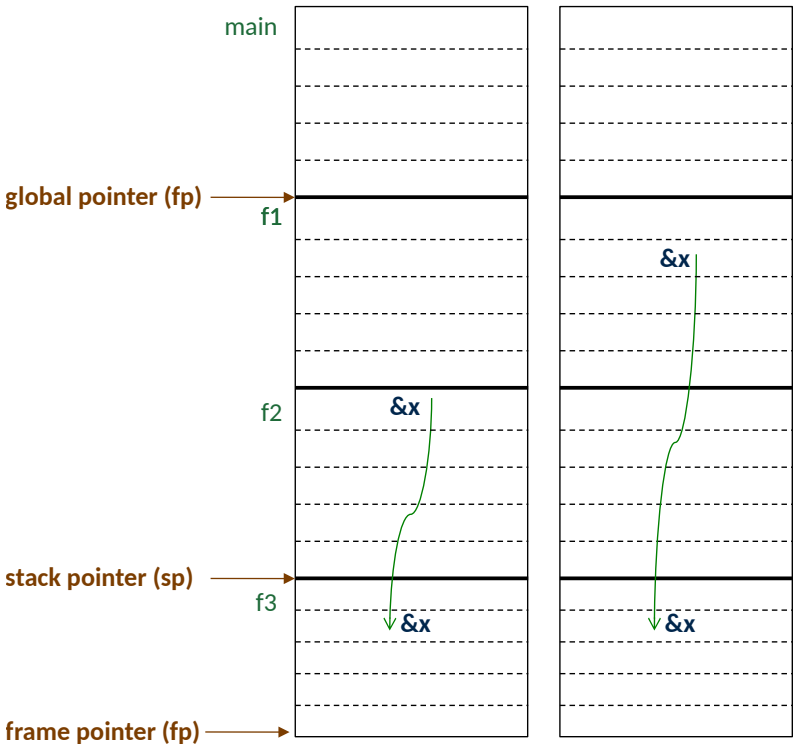
α) τοπική μεταβλητή ή προσωρινή μεταβλητή ή παράμετρος που έχει περαστεί με τιμή στη συνάρτηση που εκτελείται:

Σε όποια από τις τρεις αυτές περιπτώσεις ανήκει η μεταβλητή, αυτή βρίσκεται `offset` bytes πάνω από τον `sp`. Άρα στη θέση d bytes πάνω από τον `fp` θα τοποθετήσουμε το `offset(sp)`

```
addi t0,sp,-offset
sw t0,-d(fp)
```

β) τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή σε συνάρτηση πρόγονο στην οποία ανήκει:

Η κλήση της `gnlvcode()` θα μεταφέρει τη διεύθυνση της μεταβλητής που θέλουμε να περάσουμε ως παράμετρο στον καταχωρητή `t0`. Από τον `t0` θα αντιγραφεί στη θέση d bytes πάνω από τον `fp`, η οποία είναι δεσμευμένη για την παράμετρο αυτή.



Σχήμα 1.15: Πέρασμα παραμέτρων με αναφορά, όταν στο εγγράφημα δραστηριοποίησης είναι αποθηκευμένη η διεύθυνση της μεταβλητής. Αριστερά όταν η παράμετρος βρίσκεται στο εγγράφημα δραστηριοποίησης της καλούσας. Δεξιά όταν βρίσκεται σε κάποιον πρόγονο

```
call gnlvcode(x)
sw t0,-d(fp)
```

γ) καθολική μεταβλητή:

Αφού πρόκειται για καθολική μεταβλητή, αυτή βρίσκεται τοποθετημένη `offset` bytes πάνω από τον `gp`. Άρα στη θέση *d* bytes πάνω από τον `fp` θα τοποθετήσουμε το `offset(gp)`.

```
addi t0,gp,-offset
sw t0,-d(gp)
```

Οι τρεις αυτές περιπτώσεις συνοψίζονται στο σχήμα 1.14.

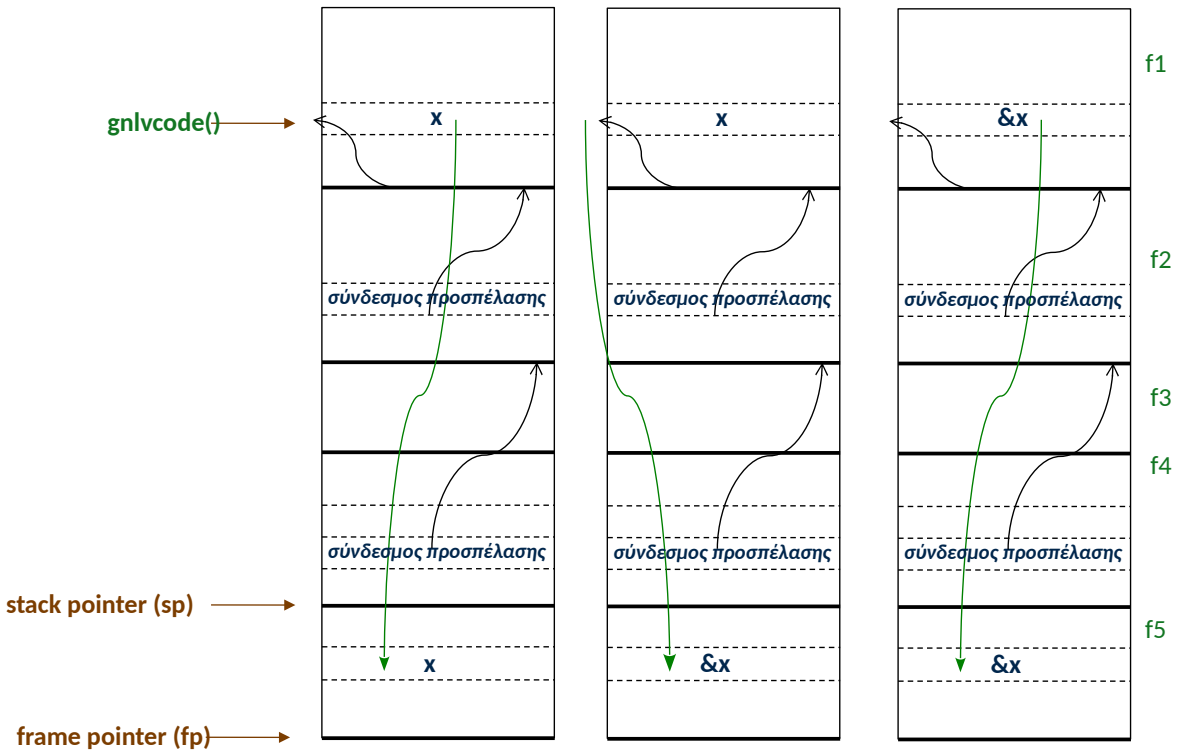
1.6.2.2 Παράμετρος με αναφορά, όταν στη στοίβα υπάρχει αποθηκευμένη η διεύθυνσή της παραμέτρου

Για να υπάρχει στη στοίβα αποθηκευμένη η διεύθυνση της μεταβλητής που θέλουμε να περάσουμε ως παράμετρος και όχι η τιμή της, σημαίνει ότι α) η μεταβλητή που θέλουμε να περάσουμε να περάσουμε ως παράμετρο με αναφορά είναι παράμετρος που έχει περαστεί με αναφορά στην καλούσα συνάρτηση ή διαδικασία β) παράμετρος που έχει περαστεί με αναφορά σε κάποιον πρόγονο. Θα εξετάσουμε τις δύο περιπτώσεις χωριστά τις οποίες μπορείτε να δείτε στο σχήμα 1.15.

α) παράμετρος που έχει περαστεί με αναφορά στην καλούσα συνάρτηση ή διαδικασία:

Στην περίπτωση αυτή, στη θέση offset πάνω από τον `sp` υπάρχει τοποθετημένη η διεύθυνση της μεταβλητής που έχει περαστεί ως παράμετρος με αναφορά στην καλούσα. Τοποθετήθηκε εκεί κατά την δημιουργία του εγγραφήματος δραστηριοποίησης της καλούσας. Από εκεί πρέπει να αντιγραφεί στη θέση *d* που έχουμε δεσμεύσει για την παράμετρο αυτή στην κληθείσα, την οποία προσπελάζουμε μέσω του `fp`:

```
lw t0,-offset(sp)
sw t0,-d(fp)
```



Σχήμα 1.16: Κατά το πέρασμα παραμέτρου με αναφορά, η τιμή ή η διεύθυνση της οποίας είναι αποθηκευμένη σε κάποιον πρόγονο, εκμεταλλευόμαστε τους συνδέσμους προσπέλασης για να εντοπίσουμε τη διεύθυνση της μεταβλητής που θα περαστεί ως παράμετρος

α) παράμετρος που έχει περαστεί με αναφορά σε συνάρτηση ή διαδικασία πρόγονο:
Η πρόσβαση σε θέση της στοίβας ενός προγόνου γίνεται με την `gnlvcode()`. Η `gnlvcode()` τοποθετεί στον καταχωρητή `t0` τη διεύθυνση της πληροφορίας που επιστρέφει ο πίνακας συμβόλων. Εκεί βρίσκεται η διεύθυνση της μεταβλητής που μας ενδιαφέρει. Δεν έχουμε, λοιπόν, παρά να την αντιγράψουμε στη θέση μνήμης που έχουμε δεσμεύσει στο νέο εγγράφημα δραστηριοποίησης. Σημειώστε ότι χρειαζόμαστε ένα βήμα περισσότερο από την προηγούμενη περίπτωση αφού η `gnlvcode()` θα τοποθετήσει στον καταχωρητή `t0` τη διεύθυνση στη στοίβας στην οποία βρίσκεται η διεύθυνση που αναζητούμε. Ο κώδικας που αντιστοιχεί στα παραπάνω ακολουθεί:

```
call gnlvcode(x)
lw t0,(t0)
sw t0,-d(fp)
```

1.6.2.3 Παράδειγμα περάσματος παραμέτρου με τη χρήση της gnlvcode()

Η `gnlvcode()` έχει χρησιμοποιηθεί σε τρεις από τις περιπτώσεις που εξετάσαμε παραπάνω και οι οποίες συγκεντρώνονται στο σχήμα 1.16. Και στις τρεις περιπτώσεις η πληροφορία που χρειαζόμαστε βρίσκεται αποθηκευμένη σε κάποιον πρόγονο. Αριστερά στο σχήμα έχουμε πέρασμα με τιμή, στη μέση πέρασμα με αναφορά όταν περνάμε τοπική μεταβλητή ή παράμετρο που έχει περαστεί με αναφορά στη συνάρτηση πρόγονο και δεξιά όταν περνάμε παράμετρο που έχει περαστεί με αναφορά στη συνάρτηση πρόγονο.

Και στις τρεις περιπτώσεις θα ανεβάσουμε με την `gnlvcode()` τον `t0` να δείχνει στην πληροφορία που μας ενδιαφέρει. Στο παράδειγμα έχουμε τέσσερις συναρτήσεις που έχουν εκκινήσει την εκτέλεσή τους (`f1,f2,f3,f4`), με την `f4` να είναι αυτή που αυτή τη στιγμή εκτελείται και η οποία δημιουργεί την `f5` για να της περάσει αργότερα τον έλεγχο. Η `f1` έχει καλέσει την `f2`, η οποία έχει καλέσει την `f3` και με τη σειρά της την `f4`. Γονέας της `f4` είναι η `f2` και παππούς η `f1`, όπως μπορεί να συμπεράνει κανείς από τους συνδέσμους προσπέλασης.

Έτσι, η αναζήτηση εκκινεί από την `f4`, από τον σύνδεσμο προσπέλασης της οποίας μεταβαίνουμε στην `f2`. Επαναλαμβάνοντας το ίδιο βήμα, και χρησιμοποιώντας τον σύνδεσμο προσπέλασης της `f2`, φτάνουμε στο εγγράφημα δραστηριοποίησης της `f1`. Αφαιρώντας το offset φτάνουμε στην πληροφορία που αναζητούμε.

Αν πρόκειται για την πρώτη ή την τρίτη περίπτωση του σχήματος 1.16 πρέπει να αντιγράψουμε στο νέο εγγράφημα δραστηριοποίησης την πληροφορία που υπάρχει στη θέση μνήμης που δείχνει ο `t0`. Απλά, στην μία περίπτωση αυτή θα είναι τιμή, ενώ στη άλλη θα είναι διεύθυνση.

Μας μένει ακόμα η μεσαία περίπτωση του σχήματος 1.16, όπου εκεί αυτό που θέλουμε να αντιγράψουμε είναι η διεύθυνση του `x`. Αυτή βρίσκεται τοποθετημένη από την `gnvlcode()` στον `t0` και από εκεί αντιγράφεται στην `f5`.

1.7 Κλήση συνάρτησης ή διαδικασίας