ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Compilers

PaZcal Compiler Documentation

Λώλος Κωνσταντίνος - 03112628

Ποδηματά Χαρίκλεια - 03110004

9ο Εξάμηνο ΣΗΜΜΥ

3 Ιουνίου 2015

Περιεχόμενα

1 PaZcal Compiler User Manual

Ο Compiler μας έχει αναπτυχθεί σε περιβάλλον Linux και παράγει Intel i386 assembly που χρησιμοποιεί το GNU assembler syntax ώστε να μπορεί να εκτελεσθεί άμεσα με χρήση των εργαλείων `as` και `ld`. Το εκτελέσιμο αρχείο του Compiler παράγεται εκτελώντας make στον κεντρικό φάκελο. Πέραν των βασικών απαιτήσεων ο Compiler υποστηρίζει επιπλεόν χειρισμό reals και της δομή switch, καθώς και μια σειρά από optimizations του ενδιάμεσου κώδικα.

Για να μεταφράσουμε ένα πρόγραμμα απλώς εκτελούμε:

```
./pazcal program.pz
```

Τότε παράγεται ενδιάμεσος και τελικός κώδικας χωρίς optimizations και τοποθετούνται σε δύο διαφορετικά αρχεία με κατάληξη: `.imm` και `.asm` αντίστοιχα. Εφόσον η assembly που παράγουμε έχει μήκος λέξης 32-bit, η συμβολομετάφραση από το εργαλείο `as` θα πρέπει να γίνει χρησιμοποιώντας το flag `--32`, πχ:

```
as program.asm -o program.o --32
```

Στη συνέχεια για την παραγωγή του εκτελέσιμου μπορεί να χρησιμοποιηθεί το εργαλείο `ld`, και πάλι με χρήση του flag `-m elf_i386`, δίνοντάς του παράλληλα και τα αρχεία βιβλιοθήκης που βρίσκονται στον φάκελο lib. Για παράδειγμα:

```
ld -m elf_i386 program.o lib/*.o -o program
```

Εναλλακτικά, ο Compiler μας υποστηρίζει 4 flags που τροποποιούν τη λειτουργία του. Συγκεκριμένα:

- `-d`: debugging mode, τυπώνουμε στο standard output ενδιάμεσο και τελικό κώδικα, με επιπλεόν πληροφορία για τον τύπο των μεταβλητών του ενδιάμεσου κώδικα
- `-i` : έξοδος ενδιάμεσου κώδικα στο standard output, το πρόγραμμα διαβάζεται από το standard input
- `-f` : έξοδος τελικού κώδικα στο standard output, το πρόγραμμα διαβάζεται από το standard input
- `-o` : ενεργοποίηση βελτιστοποιήσεων ενδιάμεσου κώδικα

Τα flags `d, i, f` δεν μπορούν να χρησιμοποιηθούν ταυτόχρονα. Το flag `o` μπορεί να χρησιμοποιηθεί σε συνδιασμό με κάποιο από τα υπόλοιπα, πχ:

```
./pazcal -io < program.pz
```

2 Testing

Για τον έλεγχο της ορθότητας του Compiler γράψαμε το script test.py σε Python, το οποίο τρέχει όλα τα testcases που έχουμε και βγάζει μηνύματα λάθους αν έχουμε είτε compiler error, είτε αν έχουμε διαφορετική έξοδο από αυτή που περιμένουμε. Για να εκτελεσθεί ένα test case από το script κάνουμε τα εξής:

- Τοποθετούμε τον κώδικα του προγράμματος στο φάκελο tests. Για να αναγνωρισθεί από το script το όνομά του θα πρέπει να είναι στη μορφή `name.pz`.
- Αν το πρόγραμμά μας αναμένουμε να παράγει κάποια έξοδο, την τοποθετούμε στον ίδιο φάκελο με όνομα `name.out`.
- Αν το πρόγραμμα αναμένει κάποια είσοδο, την τοποθετούμε και αυτή στον ίδιο φάκελο με όνομα `name.in`.

Στη συνέχεια απλώς εκτελούμε:

```
./test.py
```

ή για να ενεργοποιήσουμε τα optimizations:

```
./test.py -o
```

Το script κάνει `make` τον compiler και `assemble` τις βιβλιοθήκες. Στη συνέχεια για κάθε αρχείο της μορφής name.pz στο φάκελο tests το μεταφράζει εξάγοντας τον τελικό κώδικα, τον οποίον κάνει assemble & link μαζί με τις βιβλιοθήκες. Στη συνέχεια εκτελεί το εκτελέσιμο που παράγεται, δίνοντας το κατάλληλο input αν αυτό υπάρχει. Τέλος, ελέγχει ότι το output που παρήγαγε είναι το ίδιο με αυτό που υπάρχει στο αντίστοιχο αρχείο .out, και μας ειδοποιεί αν υπάρχει λάθος σε κάποιο από όλα αυτά τα βήματα. Επιπλέον, στον φάκελο `tests/failtests/` υπάρχουν testcases τα οποία εμφανίζουν compiler errors. Αυτά το script θα προσπαθήσει να τα μεταφράσει και θα μας ειδοποιήσει μόνο αν η μετάφραση επιτύχει. Όλα τα ενδιάμεσα αρχεία που παράγονται τοποθετούνται στον φάκελο tmp/ ο οποίος δημιουργείται από το script αν δεν υπάρχει.

Αν όλα πάνε καλά περιμένουμε μια έξοδο της μορφής:

```
./test.py
186/186 test cases passed!
```

3 Optimizations

Ο compiler μας υλοποιεί τα παρακάτω optimizations ενδιάμεσου κώδικα:

- Αποτίμηση σταθερών εκφράσεων
- Αντίστροφη διάδοση αντιγράφων: απαλοιφή ενδιάμεσων αποτελεσμάτων που γίνονται assigned σε κάποια άλλη μεταβλητή στην επόμενη εντολή.
- Απλοποίηση συνθηκών και αλμάτων: αντικατάσταση conditional jump που απλώς προσπερνά ένα jump με conditional jump αντίστροφης συνθήκης που οδηγεί εκεί που οδηγούσε το jump.
- Απλοποίηση jumps που οδηγούν σε jumps ώστε να οδηγούν στον τελικό στόχο.
- Απαλοιφή jumps στην επόμενη εντολή
- Μετατροπή conditional jump με συνθήκη που μπορεί να αποτιμηθεί στο compile time σε απλό jump.
- Απαλοιφή απροσπέλαστου κώδικα: Απαλοιφή τετράδων στις οποίες η εκτέλεση δεν μπορεί ποτέ να φτάσει, πχ. εντολές μετά από κάποιο return ή break, ή εντολές μέσα σε κάποιο if (false).
- Απαλοιφή κώδικα που υπολογίζει τιμές που δεν χρησιμοποιούνται
- Απαλοιφή αναθέσεων μιας μεταβλητής στον εαυτό της
- Αλγεβρικοί μετασχηματισμοί

Συγκεκριμένα, σε ό, τι αφορά τους αλγεβρικούς μετασχηματισμούς, οι μετασχηματισμοί που υλοποιήθηκαν ήταν οι ακόλουθοι:

(α΄) $+$, α, 0, β $\rightarrow$:=, α, -, β

(β΄) $+$, 0, α, β $\rightarrow$:=, α, -, β

(γ΄) $-$, α, 0, β $\rightarrow$:=, α, -, β

(δ΄) $*$, α, 0, β $\rightarrow$:=, 0, -, β

(ε′) $*$, 0, α, β → :=, 0, -, β

(ϛ′) $*$, α, 1, β → :=, α, -, β

(ζ′) $*$, 1, α, β → :=, α, -, β

(η′) $/$, α, 1, β → :=, α, -, β

(ϑ′) $\%$, α, 1, β → :=, α, -, β

4 Βιβλιοθήκες

Η γλώσσα assembly που βγάζει ο compiler μας είναι η assembly `i386`. Για τις συναρτήσεις βιβλιοθήκης χρησιμοποιήσαμε τις βιβλιοθήκες που πήραμε από το Δημήτρη Τσίπρα, τις οποίες τροποποιήσαμε ελαφρώς λόγω του ότι επιλέξαμε διαφορετική σύμβαση για τη δομή του activation record. Για την εκτύπωση των reals γράψαμε μια δικιά μας συνάρτηση βιβλιοθήκης σε PaZcal, την οποία μεταφράσαμε με τον compiler μας. Όλες οι βιβλιοθήκες βρίσκονται στο φάκελο `lib/`, και ο κώδικας PazCal της συνάρτησης εκτύπωσης reals στο αρχείο `lib/src/write_real.pz`.

Για την απλοποίση της διαχείρισης μνήμης χρησιμοποιήσαμε το συλλέκτη σκουπιδιών του Hans Boehm που προτείνεται στη σελίδα του μαθήματος, ο οποίος βρίσκεται στο φάκελο gc/.

5 Κώδικας

Ο κώδικάς του Compiler είναι μοιρασμένος σε διαφορετικά αρχεία ως ακολούθως:

- `lexer.l`: Αρχείο που υλοποιεί τη λεκτική ανάλυση.
- `parser.y`: Αρχείο που υλοποιεί τη συντακτική και σημασιολογική ανάλυση, και παράγει τον ενδιάμεσο κώδικα σε μορφή τετράδων.
- `symbol.cpp`: Αρχείο που υλοποιεί όλες τις λειτουργίες του symbol table. Στις αρχικές, δοσμένες συναρτήσεις προστέθηκαν και άλλες που υλοποιήσαμε εμείς.
- `error.cpp`: Αρχείο που υλοποιεί τις συναρτήσεις για τα errors.
- `general.cpp`: Αρχείο για όλες τις γενικές συναρτήσεις του προγράμματος, που χρειάζονται σε διάφορες φάσεις της μεταγλώττισης.
- `final.cpp`: Αρχείο που υλοποιεί όλες τις συναρτήσεις που χρειαζόμαστε για την παραγωγή του κώδικα assembly.
- `opt.cpp`: Αρχείο που υλοποιεί όλες τις συναρτήσεις που χρειαζόμαστε για τα optimizations του compiler, στα οποία έχουμε αναφερθεί σε προηγούμενο κεφάλαιο.