Μεταγλωττιστής από TIL σε UM-32

Μάθημα: Προηγμένα Θέματα Θεωρητικής Πληροφορικής (CST319 / W10)
Διδάσκων: Νικόλαος Καββαδίας
nkavv@uop.gr
20/03/2010

Αντικείμενο της εργασίας

Αντικείμενο αυτής της εργασίας είναι η ανάπτυξη και σχεδιασμός μεταγλωττιστή από την εκπαιδευτική γλώσσα TIL (Tiny Imperative Language) σε κώδικα για την εικονική μηχανή UM-32 (Universal Machine). Ο μεταγλωττιστής θα πρέπει να μπορεί να χρησιμοποιηθεί σε εκτελέσιμη μορφή από τη γραμμή εντολών του κατάλληλου τερματικού σε Cygwin/Windows ή Linux. Ο μεταγλωττιστής θα πρέπει να σχεδιαστεί σύμφωνα με τις αρχές του δομημένου προγραμματισμού για την ευκολότερη συντήρηση και επέκταση του κώδικα.

Η γραμματική της TIL δίνεται στο Σχήμα 1 σε μορφή BNF (Backus-Naur Form), ενώ ένα παράδειγμα εφαρμογής γραμμένο σε TIL δίνεται στο Σχήμα 2.

```
% Grammar for Tiny Imperative Language (TIL)

program -> statement*

statement ->  declaration
           |  assignment_statement
           |  if_statement
           |  while_statement
           |  for_statement
           |  read_statement
           |  write_statement

% Untyped variables
declaration -> "var" identifier ";"

assignment_statement -> identifier ":=" expression ";"

if_statement -> "if" expression "then"
                    statement*
                "end"
             |  "if" expression "then"
                    statement*
                "else"
                    statement*
                "end"

% While loop
while_statement -> "while" expression "do"
                      statement*
                   "end"

% Declaring for
for_statement -> "for"  identifier ":=" expression "to" expression do
                     statement*
                 "end"

read_statement -> "read" identifier ";"
```

```
write_statement -> "write" expression ";"

% Simple
expression -> primary
           |  expression op expression

primary -> identifier
        |  integer
        |  string
        |  "(" expression ")"

op -> "=" | "!="        % lowest priority
   |  "+" | "-"
   |   "*" | "/"          % highest priority
```

Σχήμα 1: Γραμματική για την Tiny Imperative Language (TIL).

```
var n;
read n;
var x;
var fact;
fact := 1;
for x := 1 to n do
    fact := x * fact;
end
write "factorial of ";
write n;
write " is ";
write fact;
write "\n";
```

Σχήμα 2: Υπολογισμός παραγοντικού στην TIL.

Η αρχιτεκτονική UM-32 αποτελεί μία εικονική μηχανή η οποία προτάθηκε το 2006 ως θέμα του προγραμματιστικού διαγωνισμού ICFPContest. Διαθέτει τα εξής χαρακτηριστικά:

- 14 βασικές λειτουργίες (εντολές)
- 8 καταχωρητές
- κάθε εντολή δέχεται μέχρι τρία ορίσματα καταχωρητών (έστω A, B, C). Μόνο η εντολή με τον κωδικό 13 δέχεται ως όρισμα εκτός από καταχωρητή και άμεση αριθμητική τιμή.

Στη συνέχεια ακολουθεί η περιγραφή των υποστηριζόμενων εντολών της UM-32.

Εντολές της UM-32

(0) CONDITIONAL MOVE
Μορφή σε UM-32 assembly:
MVI rA <- rB if rC
Λειτουργία:
Ο καταχωρητής A δέχεται την τιμή του B, εκτός και αν τα περιεχόμενα του καταχωρητή C είναι μηδέν.

(1) ARRAY INDEX (LOAD)
Μορφή σε UM-32 assembly:
LOD rA <- rB[rC]
Λειτουργία:
Ο καταχωρητής φορτώνεται με την τιμή που βρίσκεται στη μνήμη που υποδεικνύει ο B και στη θέση (διεύθυνση) που δείχνει ο C.

(2) ARRAY AMENDMENT (STORE)
Μορφή σε UM-32 assembly:
STO rA[rB] <- rC
Λειτουργία:
Στον πίνακας που υποδεικνύεται από τον καταχωρητή A, και στη θέση B, αποθηκεύεται η τιμή του καταχωρητή C.

(3) ADDITION
Μορφή σε UM-32 assembly:
ADD rA <- rB, rC
Λειτουργία:
Ο καταχωρητής A δέχεται το άθροισμα των B και C.

(4) MULTIPLICATION
Μορφή σε UM-32 assembly:
MUL rA <- rB, rC
Λειτουργία:
Ο καταχωρητής A δέχεται το γινόμενο των B και C.

(5) DIVISION
Μορφή σε UM-32 assembly:
DIV rA <- rB, rC
Λειτουργία:
Ο καταχωρητής A δέχεται το πηλίκο της διαίρεσης των B και C. Η διαίρεση πραγματοποιείται μόνο όταν ο C δεν είναι 0, και κάθε ποσότητα θεωρείται μη προσημασμένος ακέραιος.

(6) NOT-AND (NAND)
Μορφή σε UM-32 assembly:
NAD rA <- rB, rC
Λειτουργία:
Κάθε bit του καταχωρητή A παίρνει την τιμή 1 εάν είτε το αντίστοιχο bit του B είτε του C είναι 0, αλλιώς παίρνει την τιμή 0.

(7) HALT PROGRAM
Μορφή σε UM-32 assembly:
HLT
Λειτουργία:
Παύση της λειτουργίας της εικονικής μηχανής.

(8) MEMORY ALLOCATION
Μορφή σε UM-32 assembly:
ALC rB <- memory:rC
Λειτουργία:
Καταμερισμός μνήμης με δημιουργία ενός νέου πίνακα με θέσεις αποθήκευσης ίσες με την τιμή του C. Κάθε θέση στο νέο πίνακα αρχικοποιείται στην τιμή 0. Στο νέο πίνακα αντιστοιχείται μία ακέραια τιμή, η οποία δεν χρησιμοποιείται από άλλον πίνακα, και η οποία τοποθετείται στον καταχωρητή B.

(9) ABANDONMENT (MEMORY FREEING)
Μορφή σε UM-32 assembly:
FRE rC
Λειτουργία:

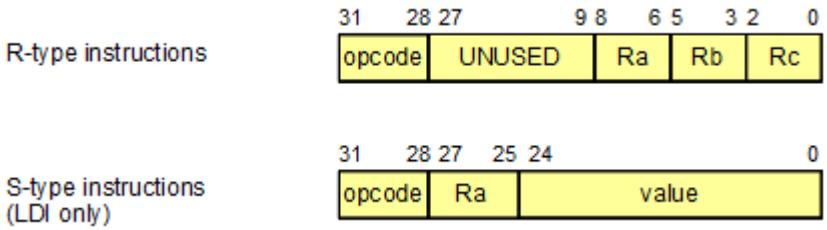
Απελευθέρωση της μνήμης που υποδεικνύεται από τον καταχωρητή C. Επόμενοι καταμερισμοί μνήμης μπορούν πλέον να χρησιμοποιήσουν τη μνήμη αυτή.

(10) DATA OUTPUT
Μορφή σε UM-32 assembly:
OUT rC
Λειτουργία:
Απεικόνιση της τιμής του καταχωρητή C στην κονσόλα. Οι τιμές που μπορούν να απεικονιστούν είναι από 0 ως 255.

(11) DATA INPUT
Μορφή σε UM-32 assembly:
INP rC
Λειτουργία:
Η εικονική μηχανή αναμένει για είσοδο δεδομένων από την κονσόλα (από το χρήστη). Η είσοδος αυτή, η οποία πρέπει να είναι από 0 ως 255 αποθηκεύεται στον καταχωρητή C.

(12) LOAD PROGRAM
Μορφή σε UM-32 assembly:
GTO memory:rB, offset:rC
Λειτουργία:
Τα περιεχόμενα του πίνακα που δείχνει ο B, φορτώνονται στον πίνακα 0 (μνήμη προγράμματος). Η εκτέλεση του νέου προγράμματος αρχίζει από τη θέση που δείχνει ο καταχωρητής C. Εξ ορισμού, η μνήμη προγράμματος βρίσκεται στον πίνακα '0'.

(13) LOAD IMMEDIATE
Μορφή σε UM-32 assembly:
LDI rA <- #B
Λειτουργία:
Η ειδική αυτή λειτουργία χρησιμοποιείται για τη φόρτωση του καταχωρητή A με μία ακέραια τιμή (B) η οποία είναι κωδικοποιημένη στη διαμόρφωση της εντολής. Ο A καθορίζεται από τα bit 27:25 της εντολής, ενώ η τιμή B από τα bit 24:0.



Σχήμα 3: Κωδικοποίηση των εντολών της εικονικής μηχανής UM-32.

Στην ιστοσελίδα του μαθήματος μία συλλογή από υλικό για την UM-32 όπως προδιαγραφές της αρχιτεκτονικής, διερμηνευτές και παραδείγματα εφαρμογών (με κατάληξη .um).

Η ενδιάμεση αναπαράσταση του μεταγλωττιστή θα πρέπει να είναι αρχικά κάποιας μορφής αφηρημένο συντακτικό δένδρο (AST). Ως ενδεικτική υλοποίηση ενός AST και των βοηθητικών ρουτινών για τη διαχείρισή του αναφέρεται το [treeir]. Στη συνέχεια θα πρέπει να παράγεται γραμμικοποιημένος κώδικας ο οποίος θα χρησιμοποιεί εντολές της αρχιτεκτονικής UM-32.

Στα πλαίσια της εργασίας ζητούνται τα εξής:

- να επεκταθεί η γραμματική της TIL ώστε να υποστηρίζει την φόρτωση και

αποθήκευση από μονοδιάστατους πίνακες, με αρχική διεύθυνση τη θέση 0. Η νέα γλώσσα θα ονομάζεται EXTIL (Extended TIL). Παραδείγματα:

```
var a[10], i;
a[0] = 5;
a[7] = 2 * i + 5;
i = (a[0] + a[1] + a[2]) / 3;
```

- ο μεταγλωττιστής να παράγει κώδικα συμβολομεταφραστή σύμφωνο με την προτεινόμενη διαμόρφωση για την αρχιτεκτονική UM-32, πραγματοποιώντας επιλογή κώδικα και καταμερισμό καταχωρητών.
- προαιρετικά ο μεταγλωττιστής να παράγει και την binary μορφή του κώδικα για την UM-32 (όπως τα παραδείγματα με κατάληξη .um). Η binary αυτή μορφή είναι διερμηνεύσιμη από εργαλεία που περιλαμβάνονται στο βοηθητικό υλικό.

Για την επιλογή κώδικα να χρησιμοποιηθεί αλγοριθμική τεχνική η οποία να καλύπτει με τα δένδρα της ενδιάμεσης αναπαράστασης, πραγματοποιώντας κατάλληλη διαπέραση τύπου DFS (Depth-First Search).

Για τον καταμερισμό καταχωρητών μπορεί να υλοποιηθεί είτε κάποια απλή τεχνική είτε ο αλγόριθμος γραμμικής σάρωσης (linear-scan register allocation) των Poletto και Sarkar, ο οποίος αποτελεί καλό συμβιβασμό ανάμεσα στην ευκολία ανάπτυξης, την ταχύτητα εκτέλεσής του και τις επιδόσεις του τελικού προγράμματος συμβολομεταφραστή. Για παράδειγμα, μία απλή τεχνική θα χρησιμοποιεί πάντα εντολές LOD για την φόρτωση των μεταβλητών σε καταχωρητές, και στη συνέχεια μετά την εκτέλεση των όποιων υπολογιστικών εντολών πάνω στους καταχωρητές, εντολές STO για την αποθήκευσή τους πίσω στη μνήμη δεδομένων.

Βιβλιογραφικές αναφορές

[bison] Bison homepage. http://www.gnu.org/software/bison/bison.html

[flex] Flex (The Fast Lexical Analyzer) homepage. http://flex.sourceforge.net

[GCC] The GNU Compiler Collection homepage. http://gcc.gnu.org

[IBURG] IBURG, a tree parser generator. http://www.cs.princeton.edu/software/iburg/

[Ker90] Brian Kernighan and Dennis Ritchie, The C Programming Language, 2nd edition, Prentice Hall, 1990.

[Lev09] John Levine, Flex & Bison, O'Reilly Publishing, 2009.

[Nie10] Tom Niemann, "A Compact Guide to Lex & Yacc."

http://www.epaperpress.com/lexandyacc/

[Pat04] D.A. Patterson and J.L. Hennessy, Computer Architecture: the Hardware-Software Interface Approach, 3rd edition, Morgan Kaufmann Publishers, San Fransisco, CA, 2004.

[TIL] Tiny Imperative Language. http://www.program-transformation.org/Sts/TinyImperativeLanguage

[treeir] Intermediate representation trees. http://code.google.com/p/drhanson/

[UM1] Προδιαγραφές της αρχιτεκτονικής UM-32. http://www.boundvariable.org/um-spec.txt (αρχικό, όχι ευανάγνωστο κείμενο για λόγους προγραμματιστικού χιούμορ).

[UM2] Ιστοσελίδα στο μάθημα των Μεταγλωττιστών του Παπασπύρου με υλικό για την UM-32. http://courses.softlab.ntua.gr/compilers/2009a/examples/um

[UM3] Συλλογή καλών διερμηνευτών για την UM-32, πολλοί από τους οποίους γραμμένοι σε C. http://www.cse.unsw.edu.au/~dons/um.html

[UM4] Απλή εξήγηση των λειτουργιών της UM-32. http://esoteric.voxelperfect.net/wiki/UM-32

Παράδοση και βαθμολόγηση της εργασίας

Στην εργασία του μαθήματος, ο φοιτητής καλείται να παραδώσει

- τον πηγαίο κώδικα του μεταγλωττιστή
- παραδείγματα δοκιμής της λειτουργίας του μεταγλωττιστή
- τεχνική αναφορά η οποία θα τεκμηριώνει τόσο τη διαδικασία της ανάπτυξης όσο και του σχεδιασμού του μεταγλωττιστή
- συνιστώμενες προδιαγραφές συστήματος του τελικού χρήστη για τη δημιουργία του μεταγλωττιστή με κτίσιμο (make) από τον πηγαίο κώδικα

Η εργασία παραδίδεται σε ηλεκτρονική μορφή (PDF της εργασίας + αρχεία κώδικα) στο email του διδάσκοντα. Οι φοιτητές μπορούν να παραδώσουν τις εργασίες τους το αργότερο μέχρι και την ημερομηνία διεξαγωγής των εξετάσεων περιόδου Ιουνίου-Ιουλίου 2010 για το μάθημα. Εργασία η οποία θα παραδοθεί μετά το πέρας αυτής της ημερομηνίας, θα βαθμολογηθεί ώστε να ληφθεί υπόψη για τις εξετάσεις της επόμενης περιόδου.

Μια εργασία βαθμολογείται με άριστα το δέκα (10), ο οποίος βαθμός και προστίθεται στο βαθμό της γραπτής εξέτασης πολλαπλασιασμένος με τον παράγοντα 0.3.

Η συγκεκριμένη προγραμματιστική εργασία είναι ατομική.