Laboratorul 8
Generarea de cod

Introducere

În laboratoarele precedente s-au tratat etapele de analiză lexicală, sintactică și semantică. În acest mod, am obținut o reprezentare abstractă a codului de CPLang sub formă de AST, care a fost validată în ultima etapă.

În acest laborator vom trata generarea directă a codului de asamblare, fără a ne preocupa expres considerentele de optimizare/performanță, cum ar fi folosirea registrelor în loc de adrese pe stivă pentru a reduce operațiile cu memoria, aritmetică in-place, sau altele. De asemenea, nu vom trata în acest laborator aspecte legate de generarea de cod orientat obiect. Obiectivul este generarea unui cod corect, ce respectă convențiile, și care poate fi rulat în QTSpim.

Ideea generală

La bază vor exista trei categorii de template-uri principale (`StringTemplate`), fiecare reprezentând una din următoarele secțiuni:

- **`.data`** — aici se vor depune toate declarațiile de variabile globale. Inițializarea lor (dacă este cazul) se va face în rutina `main`.
- **`.text` (funcții)** — aici se vor depune toate declarațiile de funcții. Acestea vor fi constituite dintr-o etichetă, urmată de prologul, corpul și epilogul funcției.
- **`.text` (`main`)** — reprezintă corpul principal al programului. Va fi plasată la final pentru a nu se intercala cu declarațiile de funcții.

Convențiile folosite sunt cele detaliate la curs între slide-urile 282–294 (conform numărului de slide din dreapta jos).

Observații

Fiecărui nod din AST i-a fost adăugat un câmp **`debugStr`**. Dacă nu este `null`, acesta conține linia de cod originală. Este menită a fi introdusă în codul generat, pentru a fi mai ușor de citit. Vedeți exemplul pentru adunare.

Task-uri

Task 1

Definiți template-urile în fișierul `cgen.stg`, și instanțiați-le în visitor-ul `CodeGenVisitor` pentru tipurile de valori **imediate** și operatorul **minus unar**. Urmăriți comentariile `TODO 1`.

Task 2

Completați template-urile și metodele `visit` pentru **operațiile binare** (aritmetice și relaționale). Tratați cazurile atât pentru numere întregi, cât și floating point (detalii aici[1]). Urmăriți comentariile `TODO 2`.

Definițiile de template-uri pot deveni rapid lungi și repetitive. Căutați o soluție aici [2] pentru a reduce numărul de linii de cod din `cgen.stg` pentru aceste template-uri.

Task 3

Completați template-urile și metodele `visit` pentru construcțiile **`If`** și **`Call`**. Urmăriți comentariile cu `TODO 3` și **respectați convențiile prezentate la curs**.

Adăugați un mecanism prin care mai multe construcții `If` să genereze **etichete diferite**. Pentru construcția `Call`, definiți template-ul astfel încât să se poată genera apeluri cu **oricâți parametri**.

Task 4

Completați construcțiile aferente pentru **`Assign`** și **`VarDef`**. Urmăriți comentariile cu `TODO 4`.

[1] https://www.doc.ic.ac.uk/lab/secondyear/spim/node20.html
[2] https://theantlrguy.atlassian.net/wiki/spaces/ST/pages/1409038/StringTemplate+cheat+sheet

În cazul definițiilor de variabile va trebui înregistrat numele variabilei în secțiunea `.data`; pentru aceasta, utilizați `ST`-ul `dataSection`. Dacă declarația conține și **expresia de inițializare**, se va adăuga codul aferent în `main`, altfel rămânând doar eticheta. Pentru definirea valorilor cu tipul `Bool` sau `Int` folosiți `.word 0`, iar pentru cele cu tipul `Float`, `.float 0.0`.

Task 5

Completați construcțiile aferente pentru **definițiile de funcții** și **referirile la variabile** (globale sau parametri formali). Urmăriți convenția prezentată la curs, și comentariile cu `TODO 5` din `CodeGenVisitor` și `IdSymbol`.

Variabilele globale vor fi accesate pe baza **etichetei** omonime din zona de date, iar parametrii formali, relativ la ***frame pointer*** (`$fp`). Pentru a distinge între cele două situații, îmbogățiți definiția simbolurilor cu informația necesară (de exemplu, numele etichetei sau offset-ul față de *frame pointer*).

Bonus

Încercați să **îmbunătățiți** codul generat în următoarele două moduri:

1. Pentru **apelurile de funcție**, calculați anticipat **deplasamentul *stack pointer*-ului** în funcție de numărul de parametri actuali, astfel încât să nu îl deplasați după fiecare depunere pe stivă a unui parametru, ca în convențiile de la curs. Astfel, va trebui să determinați offset-ul față de *stack pointer* unde veți depune direct valoarea unui parametru actual.
2. Pentru **definițiile de funcție**, determinați anticipat **necesarul de locații temporare pe stivă**, pentru a evita deplasarea *stack pointer*-ului la fiecare depunere a unui rezultat temporar (slide-urile 297–300). Astfel, locațiile temporare vor fi rezervate în înregistrarea de activare și accesate la anumite offset-uri față de ***frame pointer*** (nu *stack pointer*). Acest lucru va duce și la **recalcularea offset-urilor parametrilor formali** față de *frame pointer*.
3. În cazul punctului anterior, **reduceți** necesarul de locații temporare, exploatând **comutatitivitatea** operatorilor de adunare și înmulțire. Din cauza asimetriei ecuațiilor de forma $NT(e_1 + e_2) = \max(NT(e_1), 1 + NT(e_2))$, putem reduce necesarul de locații temporare dacă generăm mai întâi cod pentru subexpresia cu NT mai mare. Astfel, la fiecare nod de AST corespunzător unui operator comutativ, putem alege dacă prelucrăm mai întâi subexpresia din stânga sau pe cea din dreapta.