Mem2Reg

- Cilj Mem2Reg pass-a je da se sve promenljive alocirane na steku, sa kojima se radi samo sa load i store instrukcijama, zamene sa registrima.
- Izazov je što registri moraju imati SSA formu.
- Algoritam izgradnje SSA forme od koda koji nije u SSA formi opisan je u radu *Efficiently Computing Static Single Assignment Form and the Control Dependence Graph,* Cytron et al.
- Da bi kod u SSA formi bio ekvivalentan polaznom, potrebno je umetnuti odgovarajuće fi instrukcije.
- Svaku dodelu početnoj promenljivoj menjamo novim registrom.
- Dva glavna pitanja - gde se umeću fi instrukcije, i vrednost kog registra dobijenog od početne promenljive se koristi pri dolasku iz datog bloka u fi instrukciji.

Gde umećemo fi instrukcije?

- Dva puta u grafu p: $X_0 \rightarrow X_k$ i q: $Y_0 \rightarrow Y_l$ konvergiraju u čvoru Z ako važi:
  - $X_0 \neq Y_0$
  - $X_k = Z = Y_l$
  - $X_i = Y_j$ tada i = k ili j = l
- Intuitivno, putevi kojima su jedini zajednički čvorovi krajnji.
- Join skup skupa čvorova A, u oznaci J(A) je skup svih čvorova Z takvih da postoje dva čvora iz A takvih da iz njih kreću putevi koji konvergiraju u Z.
- Iterirani join skup, u oznaci $J^+(A)$ - određuje se na sledeći način:
  - Odrediti J(A) i dodati sve čvorove u $J^+(A)$, ako se $J^+(A)$ nije promenio završava se algoritam
  - Dodati sve čvorove iz J(A) u A
- **Iterirani join skupa čvorova gde se vrše dodele nekoj promenljivoj je baš skup čvorova u koji treba umetnuti fi instrukcije za tu promenljivu**

Granica dominacije

- Granica dominacije (Dominance frontier) za dati čvor X, u oznaci DF(X) je skup čvorova Z za koje važi:
  - X ne dominira Z
  - Postoji čvor Y takav da u CFG postoji grana iz Y do Z, i X dominira Y
- Važi $DF(X) = DF_{local}(X) \cup (\cup_{Z \in Succ(X)} DF_{up}(Z))$, gde su:
  - $DF_{local}(X)$ - Skup svih čvorova do kojih postoji grana u CFG ali ga X ne dominira, tj idom(Y) ≠ X
  - $DF_{up}(Z)$ - Skup svih čvorova koji su u DF(Z) i važi idom(Z) ne dominira Y
  - Succ(X) - Svi čvorovi Y za koje važi da postoji grana od X do Y u CFG
- Granica dominacije određuje se na osnovu prethodnog obilaskom čvorova rastuće po vremenu izlaska DFS obilaska dominatorskog stabla.
- Implementacija DomTree::GetDominanceFrontiers (DomTree.cpp:45)

Iterirana granica dominacije

- Iterirana granica dominacije čvora X, u oznaci $DF^+(X)$, definiše se analogno iteriranom join skupu.
- Ispostavlja se da za proizvoljni skup čvorova A važi $DF^+(X) = J^+(X)$
- Postupak određivanja za svaku od promenljivih koja se obrađuje:
  - Dodati sve čvorove koji sadrže dodele promenljivoj, tj deklaracije registara koji odgovaraju toj promenljivoj u red
  - Sve dok red nije prazan, uzimati čvor sa vrha reda, i obići sve čvorove iz njegovog fronta dominacije. Za svaki od tih čvorova, dodati ga u $DF^+(X)$, i ako nije već dodavan u red, dodati ga u red.
  - U sve čvorove iz $DF^+(X)$ umeću se fi instrukcije.
- Implementacija: InsertPhiInstructions (OurMem2RegPass.cpp:82)

Dodela registara fi instrukcijama

- Kada su fi instrukcije postavljene, potrebno je za svaku dolaznu granu u grafu kontrole toka deklarisati koju vrednost fi instrukcija uzima, tj u ovom slučaju kom registru dobijenom od početne promenljive.
- Algoritam:
  - Za svaku promenljivu deklariše se stek na koji će se stavljati registri koji joj odgovaraju.
  - Započni obradu ulaznog čvora CFG-a.
  - Za svaku naredbu trenutnog bloka:
    - Ako je naredba dodele nekoj promenljivoj, zameni novim registrom i stavi taj registar na vrh steka.
    - Ako je naredba koja koristi neku od promenljivih, zameni upotrebu sa registrom sa vrha steka.
  - Za svako Y takvo da od trenutnog bloka X postoji grana X do Y u CFG dodaj u fi instrukcije koje su u bloku Y vrednost registra za odgovarajuću promenljivu sa vrha steka.
  - Obradi sve Y koji su deca od X u dominatorskom stablu rekurzivno.
  - Ukloni sa steka sve registre koji su kreirani pri obradi trenutnog čvora
- Implementacija: renameVars (OurMem2RegPass.cpp:161)