MIPS modulet og registerallokatoren

Værktøjer til oversætterdelen af K1

Dat1E 2001

MIPS modulet

Den vigtigste del af modulet `MipsData.sml` er datatypen `mips`, som beskriver MIPS ordrer, se figur 1. Alle de i K1-opgaven nævnte ordrer (og lidt til) er understøttet af denne datastruktur. Endvidere er de fleste pseudo-ordrer fra `MASM` understøttet.

Alle registre eller konstanter angives i `mips` datatypen som tegnfølger (af typen `string`). Dermed er det muligt at bruge symbolske navne og regneudtryk. I registerfelter vil en tegnfølge, der er rent numerisk blive betragtet som et nummereret register og udskrevet med et `$`-prefix. Ikke-numeriske tegnfølger i registerfelter vil blive betragtet som symbolske navne og udskrevet med et `:`-prefix. Talfelter udskrives uændret, så man kan bruge alle de udtryk, som `MASM` understøtter.

Til at repræsentere en liste af MIPS-intruktioner bruges typen `mips list`. En liste kan omformes til en tekst (`string`) i 'MASM'-kompatibelt format med funktionen `pp_mips_list`.

Der er i `MipsData.sml` defineret tre funktioner, som implementerer pseudo-ordrer `MOVE`, `LI` og `SUBI`. Disse bliver oversat til hhv. `ORI` og `ADDI`. Pseudo-ordrer har to formål:

1) De letter læseligheden af oversættere.

2) Registerallokatoren kan fjerne `MOVE`-ordrer, som flytter mellem to identiske registre. Det kan den ikke hvis `MOVE` er implementeret på en anden måde end den definerede pseudo-ordrer (f.eks. med `ADDI` eller `OR` med register 0).

`JAL` intruktionen har, udover destination, i `mips` datastrukturen et ekstra argument, som er en liste af registre. Denne liste *skal* indeholde de registre, som er brugt til at overføre parametre til den kaldte funktion (hvis registre

```
datatype mips
 = LABEL of string
 | ASSERT of string
 | ORG of string
 | EQU of string*string
 | EQUR of string*string
 | END
 | NOP
 | DC of string list
 | DS of string
 | DSRND of string*string
 | COMMENT of string
 | LUI of string*string
 | ADD of string*string*string
 | ADDI of string*string*string
 | SUB of string*string*string
 | AND of string*string*string
 | ANDI of string*string*string
 | OR of string*string*string
 | ORI of string*string*string
 | SLT of string*string*string
 | SLTI of string*string*string
 | BEQ of string*string*string
 | BNE of string*string*string
 | J of string
 | JAL of string * string list (* label + argumentregistre *)
 | JR of string
 | LW of string*string*string (* lw rd,i(rs) kodes som LW (rd,rs,i) *)
 | SW of string*string*string (* sw rd,i(rs) kodes som SW (rd,rs,i) *)
 | STOP

fun MOVE (rd,rs) = ORI (rd,rs,"0")

fun LI (rd,v) = ADDI (rd,"0",v)

fun SUBI (rd,rs,v) = ADDI (rd,rs,"-(" ^ v ^ ")")
```

Figure 1: MIPS datastrukturen

bruges til dette). Dette er af hensyn til registerallokatoren, som ellers ikke kan se, at disse registre er levende frem til kaldet.

Funktionen `pp_mips_list` udskriver en liste af MIPS instruktioner i et format, der kan læses af MASM assembleren.

MIPS modulet findes i filen `~dat1e/K1/MipsData.sml`.

Registerallokatoren

Registerallokatoren er defineret i `RegAlloc.sig` og `RegAlloc.sml`. Det er en simpel registerallokator uden *spill*. Dette skulle ikke være noget problem med programmer i den størrelse som skal oversættes i K1.

Registerallokatoren kaldes med følgende argumenter:

- En liste af MIPS-ordrer
- Angivelse af de registre, som registerallokatoren må bruge. Dette er givet som et interval af registre, som er opdelt i to grupper: Caller-saves registre og callee-saves registre. Dette angives med tre tal: `rmin`, `callerMax` og `rmax`. Alle registre mellem `rmin` og `callerMax` er caller-saves registre. Registre mellem `callerMax`+1 og `rmax` er callee-saves.

Registerallokatoren returnerer følgende:

- En liste af register-alias definitioner som knytter numeriske registre til symbolske navne. Disse er lavet med `equr` direktivet fra `MASM`.
- En modificeret udgave af den liste af MIPS-ordrer, som blev givet som inddata. `MOVE` ordrer vil være kommenteret ud i den modificerede liste, hvis de to operander er allokeret til samme register.
- En liste af de variabler, der er levende ved indgangen af koden. Denne information er primært nyttig til testudskrifter under aflusning af oversætteren.
- Det højeste allokerede registernummer. Dette kan udnyttes til kun at gemme de callee-saves registre, der rent faktisk bliver brugt.

Registerallokatoren findes i filerne `RegAlloc.sml` og `RegAlloc.sig` i kataloget `~dat1e/K1`.

Kaldkonventioner

Registerallokatoren understøtter rene callee-saves kaldkonventioner eller blandede caller-saves/callee-saves konventioner, men ikke rene caller-saves konventioner. Det skyldes at registerallokatoren ikke har adgang til aktiverinsgposten og derfor ikke har et sted at gemme caller-saves registre ved kald. Det betyder også at gemning/hentning af callee-saves registre ikke gøres af registerallokatoren. Kode for dette kan tilføjes efter registerallokatoren er kaldt, når man ved, hvilke callee-saves registre er brugt.

Herunder er en kort vejledning om hvordan registerallokatorer kan bruges ved forskellige kaldkonventioner.

Ren stakbaseret callee-saves.

Denne konvention svarer til figurerne 9.4-9.6 i "Basics of Compiler Design". Registerallokatoren kaldes med koden svarende til den del af figur 9.5 som starter med $parameter_1 := M[FP + 4]$ og slutter med $M[FP + 4] := result$.

Da `JAL` bruger et register til at gemme returadressen, skal gemningen af denne flyttes fra kaldstedet til prologen af den kaldte funktion, altså fra figur 9.6 til figur 9.5. Se endvidere herunder.

Da der ikke er nogen caller-saves registre, sættes `callerMax` parameteren til registerallokatoren til 0. `rmin` sættes til 1 og `rmax` til det højeste register, som ikke bliver brugt til specielle formål, såsom FP eller link-register. I eksemplet herunder er `rmax` sat til 25.

Hvis `body` er kroppen af funktionen, inklusive hentning af parametre fra stakken og gemning af resultatet i stakken (som beskrevet herover), så kan man kalde registerallokatoren som følger:

```
let val (allocs, newbody, liveAtEntry, maxUsed)
          = RegAlloc.registerAlloc body 1 0 25
```

Efter kaldet indeholder `allocs` registerdefinitioner (en liste af `EQUR` pseudoordrer), `newbody` en ny version af kroppen (hvor nogle `MOVE` ordrer er kommenteret ud). `liveAtEntry` skal ikke bruges i denne sammenhæng, men `maxUsed` indeholder nummeret på det højeste brugte register.

Man skal nu foran `newbody` tilføje kode til at gemme registrene $1 \ldots$ `maxr` på stakken og efter `newbody` tilføje kode til at hente dem fra stakken igen.

For at gøre funktionen komplet skal yderligere tilføjes:

- En label for funktionen.
- De registerdefinitioner, som er givet i `allocs`.

```
let
  val body = kode for kroppen af funktionen
  val loadcode = kode til at hente parametre fra stakken
  val resultcode = kode til at lægge resultat på stakken
  val body2 = loadcode @ body @ resultcode
  val (allocs,body3,liveAtEntry,maxUsed) =
         RegAlloc.registerAlloc body2 1 0 25
  val frameSize = 4*(antal parametre+1+maxUsed)
  val savecode = kode til at gemme reg. 1..maxUsed i aktiveringspost
  val restorecode = kode til at hente reg. 1..maxUsed i aktiveringspost
in
    [MipsData.LABEL f, MipsData.SW ("31",FP,"0"),
     MipsData.EQU (f ^ "_framesize", Int.toString frameSize)]
  @ allocs @ savecode @ body3 @ restorecode
  @ [MipsData.LW ("31",FP,"0"), MipsData.JR "31"]
end
```

Figure 2: Kodegenerering for funktionen f med ren callee-saves strategi

- Kode til at gemme register 31 i aktiveringskroppen.
- (Efter kroppen) kode til at hente register 31 fra aktiveringsposten.
- En `JR` ordre til at hoppe til returadressen.

I figur 2 er skitseret et stykke SML, der genererer kode for en funktion jævnfør ovenstående beskrivelse. Visse detaljer er udeladt (markeret med *kursiv*). `FP` er en SML variabel, der indeholder nummeret (som tegnfølge) på det register, derf bruges som frame-pointer.

Når man inde i kroppen af funktionen skal oversætte et funktionskald, skal man (jvf. fig 9.6 i "Basics of Compiler Design") lægge $framesize$ til FP. Men $framesize$ kendes først efter at registerallokatoern er kaldt og (med `maxUsed`) angiver hvor mange registre, der skal gemmes. Derfor er det en god ide, inde i kroppen at bruge en symbolsk konstant for $framesize$ og efter kaldet til registerallokatoren tilføje en definition af den symbolske konstant til koden (med en `EQU` pseudo-ordre). Husk at bruge forskellige symbolske navne for $framesize$ i forskellige funktioner. Figur 2 viser hvordan det gøres.

Når man skal oversætte et funktionskald, beregnes parametrene først i mellemkodevariabler. Lige før kaldet lægges den aktuelle *framesize* til FP og paramterene flyttes fra mellemkodevariablerne til den nye aktiveringspost (som vist i figur 9.6 i "Basics of Compiler Design"). Derefter laves en `JAL`

instruktion til den kaldte funktions label. Da der ikke bruges registre til parameteroverførsel, er registerlisten til `MipsData.JAL` tom. Bagefter `JAL` instruktionen sættes kode til at flytte resultatet fra aktiveringsposten og trække *framsize* fra FP igen. Som nevnt ovenover, er *framesize* en symbolsk variable, der hedder noget forskelligt fra funktion til funktion. For at bruge den rigtige symbolske variabel, skal oversætterfunktionen kende navnet på denne. Dette kan gøres med en ekstra nedarvet attribut til oversætterfunktionen eller via en global variabel.

Hvis man har separat FP og SP, som beskrevet i afsnit 9.8.1 i "Basics of Compiler Design", behøves $framesize$ ikke. Til gengæld skal prolog/epilog og kaldsekvens modificeres så både FP og SP opdateres. Se afsnit 9.8.1 for flere detaljer.

Stakbaseret blandet caller-saves og callee-saves.

Registerallokatoren er lavet sådan at variabler, der er levende henover funktionskald, *ikke* bliver allokeret i caller-saves registre. Dermed er det ikke nødvendigt at gemme levende variabler, som ligger i caller-saves registre (der vil nemlig ikke være levende variabler i caller-saves registre), så koden for et kald i den blandede kaldkonvention ser ud præcis som i den rene callee-saves strategi ovenfor.

Man skal sørge for, at der er callee-saves registre nok til at holde alle variabler, der er levende over funktionskald. Derfor bør højest 1/3 af de allokerbare registre være caller-saves.

I forhold til den rene callee-saves strategi beskrevet i figur 2, skal der kun laves to ændringer:

1) I kaldet til registerallokatoren skal `callerMax` sættes til det højeste caller-saves register (f.eks. 8), så man skriver "8" i stedet for "0" i kaldet.

2) Kun registre mellem `callerMax`+1 (f.eks. 9) og `maxUsed` skal gemmes og hentes i aktiveringsposten.

3) Beregningen af *framesize* ændres så man i stedet for `maxUsed` bruger `(maxUsed-callerMax)`.

Med meget lidt ekstra indsats kan man altså spare et betydeligt antal gemninger og hentninger af registre.

Brug af registre til parameteroverførsel/retur.

Denne konvention svarer til figur 9.7-9.10 i "Basics of Compiler design".

I forhold til den ovenfor beskrevne blandende konvention skal der laves følgende ændringer:

I kaldsekvensen:

1) De første parametre lægges i caller-saves registre (`rmin` ... `callerMax`) i stedet for på stakken. De resterende parametre (hvis der er flere) skal stadig lægges på stakken.

2) I koden for funktionskaldet skal `JAL` ordren markeres med de variabler, der bruges til overførsel af parametre til kaldet.

3) Efter kaldet laves et flyt fra det første caller-saves register til den symbolske variabel, der skal indeholde resultatet af kaldet. Registerallokatoren vil fjerne dette flyt, hvis det ikke er nødvendigt, dvs, hvis den symbolske variabel kan allokeres i det første caller-saves register.

I prolog og epilog:

1) I stedet for at hente alle parametrene fra stakken, flyttes de første af dem fra caller-saves registrene til symbolske registre[1]. Registerallokatoren vil eliminere de indsatte `MOVE`, hvis der alligevel ikke er brug for dem (dvs., hvis det symbolske register kan allokeres i samme register, som parameteren blev overført i).

2) Efter beregning af kroppen lægges værdien af denne i det første caller-saves register i stedet for på stakken.

3) Beregningen af *framesize* ændres, så parametre, der er overført i registre ikke tæller med til *framesize*.

[1]Det er *ikke* en god ide bare at lade dem blive liggende i de nummererede registre, da de så vil blive overskrevet ved funktionskald.