Oversættelse af Simp samt simulering af MIPS

Karakteropgave 1 (K1)

Dat1E 2001

1 Introduktion

Dette er den karaktergivende rapportopgave K1 på kurset Dat1E, efterår 2001. Opgaven skal løses i grupper på indtil 3 personer. Opgaven bliver stillet fredag d. 16/11 2001 og skal afleveres i 1.dels administrationen i 2 eksemplarer senest tirsdag d. 18/12 2001 kl. 14.00. Alle programmer og testdata brugt i rapporten skal være tilgængelige i et katalog med navnet `~bruger/K1`, hvor `bruger` er den 1. delskonto som er nævnt på forsiden. Der må ikke rettes i disse filer efter afleveringen.

2 Om opgaven

Der skal laves en oversætter fra sproget Simp, som blev brugt i G1, til MIPS assembler. Den genererede kode skal kunne indlæses af assembleren `masm` og afvikles med referencesimulatoren `mips`. Til hjælp for dette udleveres nogle moduler og værktøjer, som bliver beskrevet herunder.

En pipelined mikroarkitektur, der kan udføre udvalgte MIPS maskinkodeordrer skal designes, implementeres og analyseres. Mikroarkitekturen skal implementeres i SimSys.

3 Krav til besvarelsen

Der lægges især vægt på velargumenterede og velbeskrevne metoder og valg, korrekthed af mikroarkitektur og oversætter, inkl. korrekt rapportering af typefejl og generering af korrekt kode. Desuden lægges der i nogen grad vægt på effektivitet, dvs. kvalitet af genereret kode og ydeevne af mikroarkitektur. Udover udskrifter må rapporten maksimalt fylde 40 sider.

3.1 Oversætteren

Det er meningen, at MIPS-kode (dog med symbolske variabelnavne) genereres direkte fra den abstrakte syntaks uden brug af mellemkode. Den udleverede registerallokator skal så bruges til at finde numeriske registre for variablerne.

Der skal laves en analyse af problemet. Specielt skal forskellige mulige kaldkonventioner overvejes, se herunder. Oversætterfunktionerne skal beskrives med diagrammer i stil med dem i kapitel 6 i "Basics of Compiler Design".

De fem testprogrammer `fac.simp, fib.simp, nim.simp, primes.simp` og `qsort.simp` skal oversættes og afvikles med referencesimulatoren. Den genererede kode for disse programmer skal vedlægges rapporten. I fald det ikke lykkes at oversætte disse programmer korrekt, skal der laves små Simp-programmer, der viser hvad der rent faktisk er muligt at oversætte. Desuden skal der gøres overvejelser om hvor problemet sandsynligvis ligger.

Tabeller i Simp er stakallokerede. Det betyder, at man ikke kan bruge en fast (*framesize*) størrelse på aktiveringspost. I stedet skal man holde separat styr på framepointer (FP) og en pointer til toppen af stakken (SP). Ved et funktionskald gemmes den gamle FP i den nye aktiveringspost og den nye FP bliver den aktuelle SP (evt. med et offset svarende til antal parametre e.lign.). SP øges hver gang noget gemmes på stakken (tabeller eller callee-saves registre). Ved retur fra funktion gives SP sin oprindelige værdi tilbage. Denne kan beregnes ud fra FP. Til sidst hentes den gamle FP fra aktiveringsposten, og der returneres.

Man kan lave mange variationer over dette tema. Den valgte kaldkonvention skal beskrives nøje ved angivelse af layout samt beskrivelse af prolog/epilog og kaldsekvens.

Hvis man bruger registre til at overføre parametre, skal man lave et testprogram, der har flere parametre end der kan overføres i registre (de resterende skal så overføres på stakken), således at man kan dokumentere at oversætteren virker for et vilkårligt antal parametre.

3.2 Simulatoren

Der skal implementeres en pipelined mikroarkitektur med genveje (eng: forwarding) som beskrevet i COD afsnit 6.1 til 6.6. Det anbefales at tage udgangspunkt i figur 6.30 side 470 i COD og inkrementelt udvide mikroarkitekturen. Til implementationen skal simulationssystemet SimSys anvendes. Det er et krav at der anvendes SimSys version 2.9 eller senere versioner. Datavejen (excl. arbejdslager og cache) må maximalt bruge 200.000 transistorer; dette tal oplyses af SimSys ved kørsel.

Mikroarkitekturen skal kunne udføre de samme MIPS maskinkodeordrer som krævet i forbindelse med G2 opgaven. Der skal ikke undersøges for overløb (i forbindelse med ADD, SUB eller ADDI) eller genereres *exceptions*. Dog skal SLT og SLTI altid fungere korrekt.

Data- og programlager implementeres ved brug af byggeklodsen `MemIOSys`. Denne byggeklods skal instantieres således:

```
D(MemIOSys,mem,7,5,5,20,100,3,fname);
```

Dette betyder bl.a., at arbejdslagerets størrelse er 1Mb og at latenstiden for læsning af et enkelt maskinord fra lageret er 100 ns, mens latenstiden for læsning af efterfølgende ord fra lageret er 3 ns. Simulatoren skal kunne læse heltal fra standard input (*cin*) eller fra en fil og skrive til standard output (*cout*) eller en fil. Dette gøres som beskrevet i G2 opgaveformuleringen og den udleverede skabelon til G2 opgaven.

3.2.1 Signalforsinkelser

Klokcyklustiden bestemmes af de længste signalveje i mikroarkitekturen. Ved at analysere signalforsinkelser kan man få et indtryk af, hvordan ændringer i mikroarkitekturen kan påvirke klokcyklustiden.

SimSys' ressourcemodel skal bruges til at undersøge kvaliteten af den udarbejdede mikroarkitektur. De signalveje og komponenter der er bestemmende for mikroarkitekturens maximale arbejdsfrekvens skal udpeges. Signalveje og komponenter der er *relativt* tæt på at være bestemmende for klokcyklustiden skal også udpeges.

3.2.2 Hopforudsigelse

En svaghed ved den i COD figur 6.30 beskrevne pipelinede mikroarkitektur er, at beslutninger om betinget hop ikke bliver taget før i MEM-fasen. Dette medfører, at det gennemsnitlige CPI for betinget hop bliver relativt højt.

Der skal implementeres en hopforudsiger, der prøver at forudsige udfaldet af hop allerede i IF fasen. Denne hopforudsiger skal udelukkende basere sin beslutning på adressen af den aktuelle ordre (PC). Idet ordretypen ikke er kendt på dette tidspunkt, er hopforudsigeren i realiteten en "næste ordre (adresse)" forudsiger. Diskuter hvordan en sådan hopforudsiger kan konstrueres, og implementer den.

For at kunne evaluere effekten af hopforudsigeren skal denne kunne deaktiveres ved at vælge PC+4 (eller en udregnet hopadresse) som næste ordreadresse.

Den implementerede mikroarkitekturs ydeevne skal herefter kvantificeres ved hjælp af benchmarks placeret i kataloget `~dat1e/K1/benchmarks` (afvikles

uden inddata) samt jeres oversatte Simp-programmer (afvikles med selvvalgte inddata). Alle benchmarks skal afvikles på den udviklede mikroarkitektur (med og uden hopforudsiger) og for hver benchmark skal antallet af udførte ordrer[1], antallet af klokcykler, CPI og simuleret udførelsestid rapporteres. Desuden skal samlet simuleret udførelsestid og CPI for alle benchmarks rapporteres. Forskelle eller mangel på samme imellem mikroarkitektur med og uden hopforudsiger, samt variationer fra benchmark til benchmark skal forklares. Det kan være en fordel at foretage en opdeling af CPI bidragene for bedre at kunne relatere dem til specifikke ordrer eller ordresekvenser.

Som bilag vedlægges udviklet kildetekst og udskrift af kørsler af de udleverede testprogrammer og benchmarks.

4 Værktøjer og moduler til oversætteren

4.1 Abstrakt syntaks for Simp

Filerne `Simp.sig` og `Simp.sml` indeholder datastrukturerklæringer for abstrakt syntaks for Simp programmer.

4.2 En Simp-fortolker og typechecker

En fortolker for Simp findes i filerne `Interpreter.sig` og `Interpreter.sml`. En typechecker findes i `Types.sig` og `Types.sml`. Fortolkeren kan f.eks. kaldes som gjort i programmet `Main.sml`, som kalder lexeren og parseren for at læse et program, typechecker det, læser en liste af tal (på separate linier), kalder fortolkeren og udskriver resultatet (som en liste af tal på hver sin linie).

En oversat udgave findes i filen `RunSimp`. Den køres med filnavnet for et Simp program som kommandolinieparameter. Derefter vil programmet indlæse tal fra standard input (et tal på hver linie) indtil EOF (ctrl-D i Linux/Unix og ctrl-Z i Windows) og køre programmet med disse tal. Fortolkeren skriver ikke noget uddata ud før programmet er kørt færdigt.

Det anbefales at man sammenligner uddata fra sit oversatte program med uddata fra fortolkeren.

`Main.sml` kan bruges som skabelon for oversætteren, idet man i stedet for at kalde fortolkeren kalder en oversætterfunktion og skriver det genererede program ud.

[1] Antallet af udførte ordrer for et givet program findes nemmest ved brug af referencesimulatoren *mips*.

Medmindre man er sikker på at ens lexer og parser fra G1 er 100% korrekt, bør man bruge de udleverede versioner af disse.

4.3 MIPS datastruktur og registerallokator

Disse værktøjer findes i kataloget `~dat1e/Simp`. De er beskrevet i dokumentet "MIPS modulet og registerallokatoren", som er udleveret ved forelæsningerne. Det kan også findes i filen `register.pdf` i ovennævnte katalog eller fra hjemmesiden (under "ugesedler").

Registerallokatoren laver *ikke* spill. Der skulle være rigeligt med registre til de udleverede testprogrammer, så hvis man får en exception "not colourable" fra registerallokatoren, bør man revidere sin oversætter. En mulige årsag kan være at man har afsat for mange registre til caller-saves.

4.4 Eksempelprogrammer

Dokumentet "Sproget Simp" viser fem eksempelprogrammer i Simp: `fac.simp`, `fib.simp`, `nim.simp`, `primes.simp` og `qsort.simp`. De skal bruges til at teste indlæsningen. De kan allesammen findes i kataloget `~dat1e/Simp` samt via kursets hjemmeside. Bemærk at programmet `qsort.simp` er ændret i forhold til det ved G1's start udleverede program, da dette var fejlbehæftet.

5 Vink

- **KISS** ("Keep It Simple, Stupid"). Når den simple løsning virker, kan man prøve at være smart (hvis man har tid nok).

- Typereglerne for Simp garanterer at en `return` sætning altid vil efterfølges umiddelbart af funktionens afslutning. Derfor behøver koden for `return` e kun at lægge værdien af e det rigtige sted (på stakken eller i returregistret). Returhop fra funktioner sker da via den sædvanlige epilog, som vil følge umiddelbart efter koden for `return`. Epilogen kan derfor laves identisk for funktioner med eller uden returværdi.

- `nim.simp` allokerer tabeller i en rekursiv funktion, så hvis det er det eneste program, der ikke kan køre, kan man se efter om tabeller bliver deallokeret ved retur fra funktioner og iøvrigt om SP og FP bliver gendannet korrekt.

- Lav en tegning af jeres mikroarkitektur, og overbevis jer om at den virker, herunder find mulige hazards i designet og løs dem. Lav små

testprogrammer (i MIPS assembler) som afprøver alle de hazards I kan finde. Hvis I finder fejl i jeres design under afprøvningen, gå tilbage til jeres tegning af arkitekturen og se/beskriv for hinanden hvad I havde overset og ret så tegningen til. Lav et testprogram som viser fejlen og tilføj det til jeres "testsuite".

- Vi anbefaler, at mikroarkitekturen udviklet i G2 udvides inkrementelt i følgende rækkefølge:
  - Tag udgangspunkt i løsningen af G2, men brug `MemIOSys` byggeklodsen i stedet for `IdealMemIOSys`. Tilføj styrelogik nødvendig for at stoppe pipelinen, når `holdAndWait` signalet er højt. Implementer pipeline registre med byggeklodser af typen `WrEnFlipFlop`, eller ved at bruge "Boundary" faciliteten beskrevet i SimSys manualen.
  - Test funktionaliteten ved at tilføje NOP ordrer til nogle af de udleverede testprogrammer, således at I undgår hazards.
  - Tilføj forwarding og hazard detektion. Bemærk at kravet om at register 0 er konstant nul stiller særlige krav. Det kan betale sig at overveje, hvordan dette krav opfyldes på elegant vis.
  - Færdiggør styrelogik til håndtering af hop. Det er nu muligt at afvikle testprogrammer og benchmarks på maskinen.
- Det anbefales at man anvender spor-faciliteten i SimSys, som er beskrevet i appendix A i SimSys manualen. Dette gør det langt nemmere for jer at overbevise læseren om, at jeres arkitektur fungerer korrekt.
- Udskriv kildetekst og data dobbeltspaltet på tværs, se beskrivelsen af `a2ps` i "Vejledning i brug af DIKUs EDB-system". Brug fortløbende sidenummerering i rapport og bilag.
- Kurset har sin egen nyhedsgruppe, `diku.dat1e`, der er beregnet som elektronisk diskussionsforum for problemstillinger knyttet til kurset. Instruktorer og lærere læser indlæg i denne gruppe regelmæssigt, og bidrager gerne med opklaring af spørgsmål m.v. Brug nyhedsgruppen.