A1: Y86-assembler, Digitallogik og Pipelining

Department of Computer Science (DIKU)
Compuster Systems 2016

September 28, 2016.

Dette er den 2. aflevering af 7 på DIKU kurset Computersystemer 2016. Størrelsen af opgaven er lavet til at blive løst af to personer og da programmering og opgaveløsning i par generelt giver bedre forståelse, anbefaler vi at I arbejder i grupper af to (ikke flere end to).

Afleveringen vil give fra 0 til 3 point. For at opnå indstilling til eksamen kræves at du opnår mindst 50 % af de mulige point i de 7 opgaver. Desuden skal opnåes mindst to point i hver område (ARC, OS, CN) på kurset. Dette er *første* opgave inden for *arkitektur (ARC)*.

Afleveringsfristen er Tirsdag d. 11 oktober kl. 22:00.
Hvis du arbejder alene skal du afleverer:

- `report.pdf` - med din report
- `src.zip` - med udviklet kode og alle ændringer til de udleverede filer.

Hvis I arbejder i par skal én af jer afleverer:

- `report.pdf` - med din report
- `src.zip` - med udviklet kode og alle ændringer til de udleverede filer
- `group.txt` - som indeholder jeres KU-numre.

Formål

Formålet med denne opgave er at forøge forståelsen af C, Y86-assembler, Digitallogik og Pipelining. Dette realiseres ved gennem forskellige mindre udfordringer at optimere Y86 og den pipelinede implementation.

Baggrund

Ydeevnen for en moderne processor er ofte begrænset af dens energiomsætning. Enten fordi højere omsætning giver ekstra varme som skal ledes væk, eller fordi højere omsætning giver mindre batterilevetid. Eller begge dele. I denne opgave skal I optimere Y86 med henblik på lavere energiomsætning. Dette tænkes opnået ved forskellige forbedringer der reducerer energiforbruget for instruktionshentning. Til slut skal i estimere effekten af jeres optimeringer.

U86-64 anvender 64-bit indlejrede konstanter. Almindeligvis er det tilstrækkeligt med langt mindre indlejrede konstanter. Vi forventer at vi kan mindske energiomsætningen ved instruktionshentning betydeligt ved at bruge mindre indlejrede konstanter hvor det er muligt.

1 Implementation i assembler (15%)

Formålet med denne del af opgaven er at vise kendskab til programmering på assembler niveau. Målet er at I har en kørende implementation af bubble-sort implementeret i Y86, som beskrevet i BOH kapitel 4. Gør det med følgende

1.1. **Implementation af bubble-sort i C**
Implementer bubble-sort (`https://en.wikipedia.org/wiki/Bubble_sort`) i C og test at den fungerer korrekt. Der er udleveret C-filer og en Makefile som kan benyttes.

1.2. **Oversættelse til X86_64**
Når du er sikker på at din koden virker kan du oversætte den til X86_64; du kan benytte den udleverede Makefile. Forstå den genererede assemble kode før du går videre. Det kan betale sig at bruge de tricks der blev introduceret til forelæsningen og tænkte over hvordan den enkelte X86 instruktion kan oversættes til Y86.

1.3. **Manuel oversættelse til Y86**
Den endelige del er at oversætte fra X86 til Y86. Vær opmærksom på at dette ikke nødvendigvis er en direkte oversættelse.

2 Optimering af irmovq, mrmovq og rmmovq (20 %)

For disse tre instruktioner ønsker vi at ændre størrelsen af den indlejrede konstant fra 64 til 16 bit.

2.1. **Ændring af indkodningen**
Definitionen af Y86-64 indkodningen findes i `sim/misc/isa.h`, og `sim/misc/isa.c`. Assembleren Yas findes i `sim/mis/yas.c`. Dan dig et overblik over disse filer, og lav de nødvendige ændringer. (Det er kun nødvendigt at ændre få linier). Efter ændring af indkodningen kan du bruge den genoversatte yas til at oversætte Y86-kode. Du kan verificere fra de genererede objektfiler (`.yo`) at indkodningen er ændret.

2.2. **Opdatering af reference simulatoren**
Efter ændring af indkodningen er det nødvendigt at tilrette referencesimulatoren Yis så den kan håndtere den nye indkodning korrekt. Du finder kildeteksten til yis i `sim/misc/yis.c` og `sim/misc/isa.cpp`. Ret her så programmet i den nye indkodning simuleres korrekt.

2.3. **Opdatering a pipeline simulatoren** Pipeline simulatoren skal opdateres så den kan udføre de nye instruktioner korrekt. Det kræver lidt analyse, som vi anbefaler tages med udgangspunkt i Figur 4.57, p. 484 i BOH. Med ændringen er det ikke længere tilstrækkeligt at genere den næste værdi for PC på basis af den gamle PC og signalerne `Need_regids` og `Need_valC`. Tegn et nyt diagram med de nødvendige ændringer. Implementer derpå ændringerne, dels i kontrollogikken `sim/pipe/pipe-std.hcl` og dels i det C-program der implementerer datavejen og driver simulationen: `sim/pipe/psim.c` .

2.4. **Verificer at dine ændringer virker**
Brug test-suiten i `sim/ptest` til at sikre dig at dine ændringer virker. Bemærk at verifikationen i `sim/ptest` virker ved at holde pipeline simulatorens resultater op imod reference-simulatorens. Hvis man begår den samme fejl i begge simulatorer vil test-suiten ikke fange fejlen.

3 Optimering af hop (20 %)

Hop-instruktionerne bruger ligeledes 64-bit store indlejrede konstanter. Antag at vi kan nøjes med 16-bit på samme vis som for `XXmovq` instruktionerne.

3.1. Optimer de 7 `jXX` instruktioner på samme vis som allerede gjort for XXmovq-instruktionerne.

4 Effekt estimering (10 %)

Gør følgende antagelser:

- Processorens maximale arbejdsfrekvens er begrænset af energiomsætningen i pipeline, primær instruktions-cache og primær data-cache
- Før ændringerne i denne opgave var den maximale arbejdsfrekvens for vores tænkte processor 3GHz og energiomsætningen i ovennævnte elementer var 4 W ved denne frekvens.
- Energiomsætningen fordeler sig med 30 % til instruktions-cachen, 20 % til data-cachen og 50 % til resten af pipelinen.
- Energiomsætningen i en instructions-cache er direkte proportional med mængden af hentede data.
- Energiomsætningen i processoren er proportional med kvadratet på arbejdsfrekvensen.

4.1. Hvad er den maximale arbejdsfrekvens for processoren efter implementeringen af ændringerne?

5 Rapport (35%)

Sammen med hele simulator koden som du har opdateret (ikke kun filerne du har ændret), skal du afleverer en kort rapport (up til 5 sider). Rapporten skal gøre status over besvarelsen:

5.1. Kort forklare de nødvendige ændringer i oversættelsen fra X86 til Y86.

5.2. Beskrive hvad der er lykkedes at implementere og hvorfor/hvorfor ikke.

5.3. Forklarer løsningen på spørgsmålet i del 4 og hvordan er I nået frem til det.

5.4. En beskrivelse af ændringerne til pipelinen i forhold til figur 4.57 med et diagram der viser disse.

Tilføj også dine valg ved eventuelle uklarheder fra opgaveteksten.