Godkendelsesopgave 2
Enkeltcyklusarkitektur og assemblerprogrammering

Maskinarkitektur, blok 1 2011

Introduktion

Dette er den anden godkendelsesopgave på DIKUs bachelorkursus *Maskinarkitektur*. Den består af 2 delopgaver. Opgaven skal udarbejdes i grupper af 2-3 personer, afvigelser herfra kræver tilladelse fra jeres instruktor.

Aflevering

Opgaven skal afleveres af en repræsentant fra gruppen på Absalon senest søndag d. 2. oktober kl 23:55. Der skal afleveres Logisim-filer og kildekode som løser opgaven samt en mindre rapport (i PDF-format), der dokumenterer jeres arbejde. Rapporten kan være på enten dansk eller engelsk.

Programmel

Til at løse G2.1 skal I bruge Logisim. I bedes opdatere jeres komponentbibliotek ved at benytte `http://github.com/downloads/andersbll/logisim-diku/logisim-diku-1.1.jar`.

Udover *Logisim* skal I bruge *Mars* til at løse G2.2. MARS er en MIPS-simulator, der letter fejlfindingsarbejdet ved at vise indholdet af registrene og hukommelsen efterhånden som programmet afvikles trin for trin. Mars kan hentes fra `http://courses.missouristate.edu/KenVollmar/MARS/`.

Hjælp til opgaven

Alle spørgsmål vedrørende opgaven bedes stilles på kursets forum på Absalon for at undgå forfordeling mellem kursusdeltagerne. Husk at benytte jer af øvelsestimerne, hvor opgaven vil blive gennemgået.

G2.1 Enkeltcyklusarkitektur

Konstruer en fortolker som kan udføre følgende MIPS-instruktioner.

R-type: `addu`, `subu`, `and`, `or`, `nor`, `slt`, `sll`, `srl`, `sra`, `jr`

I-type: `addiu`, `andi`, `ori`, `slti`, `beq`, `sw`, `lw`

J-type: `j`, `jal`

Instruktionerne er kort beskrevet i figur 3.13, s. 243. Fortolkeren skal implementeres som en digitallogisk model i Logisim og tage udgangspunkt i figur 4.17, s. 322 (I bliver dog nødt til at udvide kredsløbet, f.eks. som i figur 4.24 på side 329). Bemærk at bogen er inkonsekvent mht. semantikken for instruktionen `jal`. Instruktionen skal fortolkes således: `R[31]=PC+4; PC=JumpAddr;`

I forhold til G1 er ALU-komponenten blevet udvidet til at understøtte skifteoperationer. Dens opførsel er defineret ud fra følgende tabel.

ALUOp	Funktion	C
0000	AND	`A & B`
0001	OR	`A | B`
0010	add	`A + B`
0110	sub	`A - B`
0111	slt	`A < B ? 1 : 0`
1100	NOR	`~(A | B)`
1000	sll	`B << A`
1001	srl	`(unsigned int)B >> A`
1010	sra	`(int)B >> A`

Udleveret skelet

For at hjælpe jer i gang medfølger et mikroarkitekturskelet. Det står jer frit at ændre det efter behag. Bemærk at programhukommelseskomponenten indeholder et program, der afprøver korrektheden af jeres løsning. Afprøvningen er dog ingenlunde udtømmende og det kan derfor være en god idé at I selv foretager yderligere kontrol.

Råd og vink

- Implementer instruktionerne i denne rækkefølge:
  1. `addu`, `subu`, `and`, `or`, `nor`, `slt`
  2. `addiu`, `andi`, `ori`, `slti`
  3. `sll`, `srl`, `sra`
  4. `beq`, `sw`, `lw`
  5. `j`, `jal`, `jr`
- Benyt jer af det grønne referencekort, *MIPS Reference Data*, i [1]. Her er alle instruktionerne listet og deres funktionalitet beskrevet.
- Bemærk at I-instruktionformatets *immediate*-felt skal nuludvides for nogle instruktioner (`andi`, `ori`) og fortegnsudvides for andre (`addiu`, `slti`, `beq`, `sw`, `lw`).
- Implementer kontrollogikken vha. *PLA*-komponenten fra biblioteket *MIPS*.

- Der er nødvendigt at udvide bitbredden på *ALUOp*-wiren mellem *Control*- og *ALUControl*-komponenterne for at kunne understøtte alle instruktionerne.
- Det indbyggede datahukommelsesmodul kan af praktiske årsager ikke rumme 2^{32} bytes. Når I sender adresser til hukommelsesmodulet skal I derfor skære de mest betydende bits fra så bitbredden svarer til størrelsen på hukommelsen.
- Kontrolsignalerne for skifteoperationerne og `jr` bestemmes af *funct*-feltet (og ikke *opcode*-feltet). Derfor kan disse kontrolsignaler med fordel genereres fra *ALUControl*-PLAen.

G2.2 Assemblerprogrammering

Oversæt følgende C-program til MIPS assembler. I må kun bruge den delmængde af MIPS-instruktionssættet som understøttes af jeres enkeltcyklusarkitektur fra G2.1. Dermed skal det færdige assemblerprogram kunne køre på jeres Logisim-kredsløb.

Det er vigtigt at I sørger for at det afleverede assemblerprogram er veldokumenteret af hensyn til dem der skal læse jeres kode. Derudover må I ikke ændre programstrukturen så I omgår de rekursive funktionskald.

Collatz-formodningen

Betragt funktionen

$$f(n) = \begin{cases} n/2 & \text{hvis } n \text{ er lige} \\ 3n+1 & \text{hvis } n \text{ er ulige} \end{cases}$$

Collatz-formodningen siger at for et vilkårligt $n \in \mathbb{N}^*$ vil rekursive anvendelser af f resultere i værdien 1 før eller siden.

C-funktionen `longest_path` nedenfor finder den længste Collatz-sti blandt alle $n <$ `limit`. Med den længste Collatz-sti menes antal iterationer før $f(f(\ldots f(n)\ldots))$ første gang returnerer 1.

```c
#include <stdio.h>

int collatz(int n) {
  if(n == 1) {
    return 0;
  } else if(n%2 == 0) {
    n = n/2;
  } else {
    n = 3*n+1;
  }
  return 1 + collatz(n);
}

int longest_path(int limit) {
  int max_length = 0;
  int n;
  for(n=1; n<limit; n++) {
```

```
18      int length = collatz(n);
19      if(length > max_length) {
20        max_length = length;
21      }
22    }
23    return max_length;
24  }
25
26  int main(){
27    longest_path(10)
28    longest_path(50)
29  }
```

Udleveret skelet

For at ensrette jeres assemblerkode, så den er nemmere for instruktorerne at rette, skal I arbejde videre på det udleverede assemblerprogram.

Råd og Vink

- For at lave operationen `n%0` kan instruktionen `andi $X, $X, 1` benyttes.
- Husk at følge MIPS-kaldkonventionerne ved funktionskald.

Litteratur

[1] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design, Fourth Edition, Fourth Edition: The Hardware/Software Interface*. Morgan Kaufmann Publishers, 4th edition, 2008.