Københavns Universitet
Maskinarkitektur

G2

Gruppe:
Asger Lund Hansen (CMG881) og Mads Ynddal (SJT402)
og Troels Ynddal (QWV828)

15. oktober 2014

1 Instruktioner

Efter at en instruktion er blevet indlæst fra 'Program Registeret', bliver instruktionens `opcode` dekodet af 'Control' PLA'en. 'Control' understøtter følende instruktioner:
`addu, addiu, slt, slti, subu, and, andi, or, ori, lw, sw, beq, jal, jr`

R-type instruktionerne har ikke nogen unik 'opcode' (opcode'en er altid 00000), så de bliver sendt videre, til dekodning i `EX-stage`. Det vil sige, at instruktionerne `addiu, slti, andi, ori, lw, sw, beq` og `jal` dekodes af 'Control'. Dekodning består i at sætte de 9 kontrol-flag. De 9 flag bruges af hvert pipeline stage til at justere ALU, MUX og lignende.

Foruden 'Control' bliver instruktionen også splittet op og sendt til `rd`, register, Hazard Detection og Immediate. `rd` bruges til Hazard Detection, så den kan holde øje med hvilke registeradresser der bliver skrevet til og hvornår.

I `EX` bliver R-type instruktionerne dekodet. Da de ikke har en 'opcode', skal vi dekode dem ved at læse deres 'funct'. Til dette har vi lavet kredsløbet 'ALUOp', som bl.a. kan læse 'funct' og indstille ALU'en korrekt. Det er desuden 'ALUOp', som sørger for jumps og branches bliver udregnet korrekt.

Branch-on-equal (`beq`) bliver tjekket ved at trække to registre fra hinanden og se op ALU'ens `Zero`-flag bliver sat. Hvis de to registre ellers er ens, tjekker den, om 'Branch'-flaget fra 'Control' er sat. Hvis begge disse betingelser er sande, vil den aktivere en branch.

2 Stall

Det er lykkedes at lave et kredsløb, som kun stall'er, hvis det er nødvendigt. Det sker, hvis man ønsker at bruge `lw` efter fulgt af en instruktion, som skal bruge den indlæste værdi. I dette tilfælde, bliver der tvunget en 'nop' operation ind efter `lw`. Grunden til at vi bliver nødt til at vente er, at vi normalt finder resultatet på en instruktion i `EX`, men dette glæder ikke for `lw`, da den først bliver udført i `MEM`.

3 Hazard

Vi har lavet vores Data Hazard ved at sammenligne instruktionen i `ID` med instruktionerne i de følgende 'pipeline stages'. Hvis registeret i `ID` er ved at blive ændret på, i en anden instruktion, vil den sætte "Hazard"flaget 'højt'.

Dette kan vi bruge i vores 'Forwarding', til at finde frem til den nyeste instans af registeret.

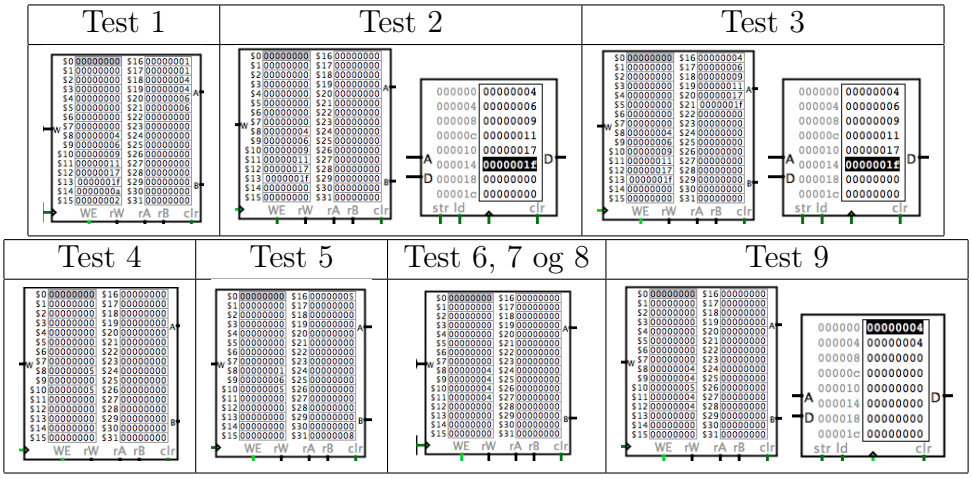
4 Forwarding

Forwarding sker ved, at vi har en 'tunnel' forbundet til slutningen af hvert 'pipeline stage', så vi kan hente et resultat, som endnu ikke er blevet skrevet tilbage i registeret. Vores forwarding sker umiddelbart efter, at vi har læst fra registeret. Vi kan vha. to multipleksere, vælge om vi vil bruge værdien fra registeret, eller om vi vi forwarde en værdi fra `EX`, `MEM` eller `WB`.

5 Branch delay slot

Branch delay slot er supporteret ved at `beq`,`jal` og `jr` først bliver afgjort i `EX`-stage og derfor vil den næste instruktion nå at blive læst ind i `ID`-stage. Instruktion i `ID` bliver ikke afbrudt, men fortsætter med at blive udført, hvilket giver os ét Branch delay slot.

6 Test



Vi har udført de tests der blev udleveret med opgaven. Vi har håndkørt alle testene og sammenlignet dette med resultatet fra både vores Single Cycle og Pipeline. Alle testene giver det resultat, vi havde forventet. Billederne ovenfor illustrere *kun* pipeline.