Lab 3 - Modelleren en simuleren van de pipelined mini-mini MIPS processor

Het doel van dit lab is kennis te vergaren en ervaring op te doen door studenten over/met:

- het doel van pipelining en de architectuur van een pipelined microprocessor,
- de verschillende deel-concepten, problemen en oplossingen in de pipelined processor,
- de structuur en werking van de pipelined mmMIPS,
- het modelleren van de pipelined mmMIPS hardware in SystemC (RTL-niveau).

Daarvoor wordt het (niet synthetiseerbaar) SystemC model van een pipelined mini-mini MIPS processor (**pipelined mmMIPS**) gebruikt.

Tijdens dit lab gaan de studenten het bijna complete SystemC model van de pipelined mmMIPS afmaken en de structuur en werking van de pipelined mmMIPS analyseren en uitbreiden. De pipelined mmMIPS is behandeld tijdens het college en beschreven in het boek "Computer Organization & Design: The Hardware / Software Interface"van D.A. Patterson en J.L. Hennessy.

Toelichting

In de **single-cycle mmMIPS** is de "clock cycle"(klokcyclus) hetzelfde voor elke instructie en deze klokcyclus is bepaald door de uitvoeringstijd van een instructie met de langste uitvoeringstijd (het langste uitvoeringspad). Dit is de *load word (lw)* instructie. De uitvoeringstijd van de andere instructies (bijvoorbeeld de *jump (j)* of *branch equal (be)*) is veel korter en daarom gaat bij het uitvoeren van deze instructies veel tijd verloren doordat er gewacht moet worden op het einde van de klokcyclus.

In de **multi-cycle mmMIPS** is het uitvoeringspad van elke instructie onderverdeeld in een aantal stappen. Elke stap wordt uitgevoerd in één klokcyclus. Instructies met uitvoeringspadden van verschillende lengte (met een verschillende hoeveelheid stappen) gebruiken verschillende hoeveelheden klokcycli voor hun uitvoering. De klokcyclus is nu veel korter en er gaat nagenoeg geen tijd verloren met het wachten tot het einde van een klokcyclus.

De **single-cycle mmMIPs** en de **multi-cycle mmMIPS** zijn op ieder tijdstip bezig met het uitvoeren van één instructie. De uitvoering van de volgende instructie begint pas nadat de lopende instructie volledig is uitgevoerd.

In de **pipelined mmMIPS** wordt het concept van een lopende band gebruikt. Net als in de multi-cycle mmMIPS is het uitvoeringspad van elke instructie onderverdeeld in een aantal stappen (die pipeline stages worden genoemd). Elk stap wordt ook in één klokcyclus uitgevoerd, maar er wordt niet gewacht op de afronding van de lopende instructie. In elke klokcyclus begint de uitvoering van de volgende instructie. Als gevolg daarvan is de pipelined mmMIPS parallel bezig met de uitvoering van een aantal instructies (zo veel als het aantal pipeline stages). Dit parallellisme verhoogt de doorvoersnelheid ("throughput") van de processor (in de "ideale"pipeline is de verhoging van de doorvoersnelheid evenredig met het aantal stages, maar in de praktijk is het aanzienlijk minder). Net als in de multi-cycle mmMIPS moeten de gegevens die in een bepaalde klokcyclus geproduceerd zijn bewaard worden in additionele registers, zodat ze gebruikt kunnen worden in de volgende klokcyclus. Bovendien moeten bepaalde signalen nu samen met de gegevens naar de volgende pipeline stage getransporteerd worden. Het hergebruik van hardware modules, typisch voor de "multi-cycle" implementatie, is nu in de meeste gevallen onmogelijk. Dit komt doordat vrijwel elk hardware module (functionele eenheid) in vrijwel elke klokcyclus bezig is met de uitvoering van een instructie. Omdat de nodige controle signalen samen met de gegevens naar de volgende pipeline stages getransporteerd worden, kan voor het besturen van de data pad dezelfde combinatorische

controller worden gebruikt als in de single-cycle mmMIPS.

Jammer genoeg werkt de elementaire pipeline niet altijd helemaal correct. Niet elke combinatie van instructies kan in een bepaalde klokcyclus correct worden uitgevoerd. Dit wordt veroorzaakt door de zo genoemde **structurele hazards**. Deze structurele hazards zijn weer onder te verdelen in: **data hazards** en **control hazards.**

Een **data hazard** treedt op, als een instructie bepaalde gegevens moet gebruiken, die berekend of opgehaald moeten worden door een andere instructie, die nog in de pipeline is op het moment dat deze gegevens al nodig zijn. Data hazards kunnen gedetecteerd worden en de benodigde gegevens kunnen in de meeste gevallen op tijd doorgestuurd worden naar de ALU inputs - dat heet **forwarding** of **bypassing**. In één geval kan dat niet, namelijk als een instructie de gegevens moet gebruiken van een register, waarvoor de vorige load instructie deze gegevens nog uit het geheugen moet ophalen. In dit geval kan deze data hazard alleen gedetecteerd worden en moet de pipeline wachten (dit heet een **pipeline stall**).

Een **control hazard** treedt op, als een beslissing over het verdere verloop van een programma (instructie sequentie) genomen moet worden, voordat de daarvoor benodigde gegevens beschikbaar zijn. In de pipelined mmMIPS is de control hazard gerelateerd aan de branch instructie, die zijn branch condition pas in de vierde stap (MEM stage) berekent. Tijdens de eerste drie stappen van de branch instructie is het dus niet zeker hoe een programma verder zal verlopen (of de branch wordt genomen of niet), maar in deze tijd moeten drie volgende instructies in de pipeline komen (dit zijn de instructies die door het branch adres zijn aangewezen, of de instructies die direct achter de branch instructie staan). Om branch hazards te voorkomen, kan de compiler een aantal instructies, die voor de branch instructie in het programma staan en van de branch uitvoering onafhankelijk zijn, achter de branch instructie schijven en/of de nodige hoeveelheid van de nop instructies toevoegen. Om minder tijd voor de branch hazards te verliezen kun je ook statische of dynamische branch predictie gebruiken. In dat geval worden de verwachte instructies in de pipeline toelaten en de branch hazard detectie corrigeert deze keuze in het geval dat de predictie fout is.

Opdrachten

Dit lab bestaat uit drie opdrachten. In de eerste opdracht ga je de ontbrekende modules en signalen aansluiten in het pipelined mini-mini MIPS model. In de tweede opdracht ga je de detectie en afhandeling van data hazards implementeren. In de derde opdracht ga je de mmMIPS uitbreiden, zodat ook control hazards goed afgehandeld worden. Alleen de **derde** opdracht moet afgetekend worden.

1. Toevoegen en aansluiten van de modules in het SystemC model

In deze eerste opdracht ga je het SystemC model van de pipelined mmMIPS installeren. Dit model is nog niet compleet, een aantal modules moet nog gedefinieerd en aangesloten worden. In deze opdracht ga je ook deze ontbrekende modules en signalen definieren en aansluiten.

1. Download het bestand pipelined_mmmips.zip van www.es.ele.tue.nl/education/5JJ55-65/mmips-lab/labs/3/ en pak het bestand uit in de folder *d:\mmips\mips*.
2. Open in *Visual C++* het project *pipelined_mmmips*. Dit project staat in de folder *d:\mmips\mips\pipelined_mmmips*. Het project bevat een incompleet model van de pipelined mmMIPS.
3. Bestudeer het schema van de pipelined mmMIPS. De modules en signalen die nog niet gedefinieerd zijn in het SystemC model zijn in **rood** aangegeven. In de rest van deze opdracht ga je deze modules en signalen definieren en de modules aansluiten op de rest van het systeem.
4. Open het bestand *main.cpp* uit het project *pipelined_mmmips* in *Visual C++*.
5. Definieer de ontbrekende modules in *main.cpp*. Geef de modules dezelfde naam als aangegeven op het schema van de pipelined mmMIPS.
6. Definieer de ontbrekende signalen in *main.cpp*. Je moet ervoor zorgen dat ieder signaal een unieke naam heeft. Probeer wel namen te gebruiken die eenvoudig te herleiden zijn tot de modules/poorten waarop de signalen worden aangesloten. De bestaande signalen in het SystemC model pipelined

mmMIPS heten bijvoorbeeld *bus_<module naam>* of *bus_<module naam>_<output poort naam>* als een module meerdere output poorten heeft.

7. Sluit alle nieuwe signalen aan op de juiste poorten van de verschillende modules in het pipelined mmMIPS model.

8. Voeg alle nieuwe signalen toe aan het gedeelte *Tracing* van het bestand *main.cpp*.

2. Data hazards detecteren en afhandelen

De module *hazard* is verantwoordelijk voor het detecteren van de data en control hazards. Zodra een hazard gedetecteerd wordt zet de module een '1' op de *Hazard* output poort. Dit signaal wordt door de module *hazard_ctrl* gedetecteerd. Deze module vervangt in dat geval de instructie in de ID-stage van de pipeline door een NOP-instructie. Als de uitgang *Hazard* gelijk is aan '0', dan kopieert de *hazard_ctrl* de signalen uit de module *ctrl* naar zijn uitgangspoorten. In het SystemC model van de pipelined mmMIPS processor, dat je in vorige opdracht hebt geïnstalleerd, is de *hazard* module nog niet geïmplementeerd. In deze opdracht ga je de detectie van data hazards in de module *hazard* implementeren en testen.

1. Open in *Visual C++* de bestanden *hazard.cpp* en *hazard.h* uit het project *pipelined_mmmips*.

2. Pas de functie *HAZARD::hazard()* in het bestand *hazard.cpp* aan zodat data hazards correct gedetecteerd worden. Hiervoor moet je het statement op regel 51 vervangen door een aantal if/else statements die de juiste situatie (EX hazard, MEM hazard, WB hazard, no hazard) detecteert en de variabele *hazard* de juiste waarde geeft (hazard=1 als er een hazard is en hazard=0 als er geen hazard is).

   De pipelined mmMIPS die je in dit practicum gebruikt wijkt af van de pipelined mmMIPS die in de vierde versie van het boek "Computer Organization & Design: The Hardware / Software Interface"van D.A. Patterson en J.L. Hennessy wordt beschreven. In de pipelined mmMIPS die gebruikt wordt in dit practicum is geen forwarding aanwezig en de branch detectie vind plaats in de MEM-stage. Vanwege deze verschillen kun je de voorwaarden voor het detecteren van (data) hazards die in het boek worden gegeven niet direct overnemen in de functie HAZARD::hazard(). Je moet deze voorwaarden aanpassen op de pipelined mmMIPS die in dit practicum wordt gebruikt.

3. Als volgende stap moet je nu het model van de pipelined mmMIPS testen. Het doel is om eventuele problemen met de detectie van data hazards te vinden. Hiervoor gebruik je het onderstaande programma. De instellingen van de register waarden worden in de functie *sc_main* gemaakt. Het programma staat in *1.asm* en de corresponderende hexadecimale programma code is te vinden in *1.bin*.

   1.asm

```
.set noat
.align 2
.set noreorder
# registers set in sc_main:
# $1 = 52
# $3 = 20
# $5 = 32
# $6 = 8
# $7 = 10
sub $2, $1, $3 # $2 = 32
and $4, $2, $5 # $4 = 32
or  $8, $2, $6 # $8 = 40
add $9, $4, $2 # $9 = 64
slt $1, $6, $7 # $1 = 1
```

   Om het pipelined mmMIPS model te testen moet je onderstaande stappen uitvoeren:

   (a) Compileer het project *pipelined_mmMIPS* met *Visual C++* met behulp van *Build* en *Build Solution* van het *Visual C++* menu.

(b) Simuleer het bovenstaande programma *1.asm* met het gecompileerde SystemC model van de pipelined mmMIPS om de werking van de pipelined mmMIPS en data hazards te observeren en analyseren. Kopieer hiertoe *1.bin* naar *mips_rom.bin* en voer het commando *./pipelined_mmmips.exe 400* uit.

(c) Bekijk de resultaten met *HDD Hex Editor* en *WinWave* en controleer of het SystemC model van de pipelined mmMIPS correct werkt. De meest interessante signalen kun je eenvoudig bekijken door in *WinWave* naar het menu *File* te gaan en vervolgens de optie *Read Save File* te kiezen. Open vervolgens het bestand *MIPS.w*. Hierna kun je natuurlijk altijd nog extra signalen toevoegen.

(d) Bestudeer met *WinWave* de werking van de pipelined mmMIPS. Kijk met name naar de werking van de pipeline in de situatie dat er een data hazard optreed.

3. Control hazards detecteren en afhandelen

In de vorige opdracht heb je de *hazard* module aangepast, zodat data hazards correct gedetecteerd worden. Control hazards worden echter nog niet gedetecteerd. In deze opdracht ga je de functie *HAZARD::hazard()* verder aanpassen zodat ook control hazards gedetecteerd worden.

In de mmMIPS is de aanname gemaakt, dat de compiler maximaal één instructie die voor de branch instructie in het programma staat en die onafhankelijk is van de branch instructie achter de branch instructie mag zetten. Als de compiler geen instructie kan vinden die onafhankelijk is van de branch, voegt de compiler een extra NOP instructie toe.

Een branch instructie kan pas ontdekt worden na de twee eerste pipeline stages (Instruction Fetch (IF) en Instruction Decoding (ID)). Dat is dus in de Execuction (EX) pipeline stage (uitgang *id_ctrl_mem_branch* register). De waarde van de branch conditie wordt berekend in de derde stap, dus in de Execuction (EX) pipeline stage en kan in de vierde stap op de uitgang van de module *and1* gevonden worden. Op het moment dat de branch instructie wordt gedetecteerd zijn de drie volgende instructies al in de pipeline (IF, ID en EX stage). Afhankelijk van de branch conditie mogen de instructies in de IF en ID stage niet uitgevoerd worden. Om dit probleem op te lossen moet je de functie *HAZARD::hazard()* zo aanpassen, dat zodra er een branch in de pipeline komt de eerste helft van de pipeline (vanaf PC tot ID/EX register) bevroren worden voor één klokcyclus en er een NOP instructie in de pipeline geplaatst wordt.

1. Pas de functie *HAZARD::hazard()* in het bestand *hazard.cpp* aan zodat control hazards correct gedetecteerd worden. Hiervoor moet je voor het statement eerste **if** statement dat je in opdracht 2 hebt toegevoegd een extra **if/else** statement toevoegen. Met dit statement detecteert je de branch operaties. Afhankelijk van hoe je precies de branch afhandelt moet je ook nog extra hardware (registers en signalen) toevoegen en deze verbinden met de hazard module of juist hardware verwijderen. Je moet zelf bepalen hoe je het ontwerp van de mmMIPS moet aanpassen om de control hazards goed af te handelen.

2. Kopieer het programma *2.bin* naar *mips_rom.bin*. Dit programma kun je gebruiken om branch hazards te observeren. In *2.asm* kun je zien wat dit programma precies doet.

3. Simuleer het aangepaste programma met de pipelined mmMIPS. Controleer vervolgens de werking van de pipelined mmMIPS met *WinWave*.