Lab 2 - Modelleren en simuleren van de multi-cycle mini-mini MIPS processor

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

- het modificeren van de architectuur van een microprocessor,
- het modelleren van microprocessor hardware in SystemC taal (RTL-niveau),
- het simuleren van SystemC modellen,
- de software die daarvoor gebruikt kan worden.

Daarvoor wordt het (niet synthetiseerbare) SystemC model gebruikt van een multi-cycle mini-mini MIPS processor (**multi-cycle mmMIPS**) en de software beschreven in "Software mMIPS-lab".
Aan het begin van dit lab krijgen de studenten een incompleet SystemC model van de multi-cycle mmMIPS. De studenten moeten tijdens dit lab de ontbrekende onderdelen invullen en daarmee het SystemC model van de multi-cycle mmMIPS compleet maken. De multi-cycle 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 "clock cycle" is bepaald door de uitvoeringstijd van een instructie met de langste uitvoeringstijd/uitvoeringspad, namelijk de *load word (lw)* instructie. De uitvoeringstijd van andere instructies is veel korter (bv. van *jump (j)* of *branch equal (be)*) en daarom gaat bij het uitvoeren van deze instructies veel tijd verloren door het wachten op het einde van de klokcyclus.

In de **multi-cycle mmMIPS** is het uitvoeringspad van elke instructie onderverdeeld in enkele stappen. Elke stap wordt uitgevoerd in één klokcyclus. Instructies met uitvoeringspaden van verschillende lengte (met een verschillende hoeveelheid stappen) gebruiken een verschillende hoeveelheid klokcycli voor hun uitvoering. Een klokcyclus is dus veel korter en er gaat nagenoeg geen tijd verloren met het wachten tot het einde van de klokcyclus. Bovendien maakt de "multi-cycle" implementatie hergebruik van hardware modules mogelijk. Een hardware module (functionele eenheid) kan meerdere keren gebruikt worden tijdens de uitvoering van een bepaalde instructie. Hiervoor is het wel vereist dat deze hardware in verschillende klokcycli wordt gebruikt. Een voorbeeld hiervan zijn de twee adders in de single-cycle mmMIPS. In de multi-cycle mmMIPS zijn deze adders afwezig, hun functie is overgenomen door de ALU. Deze ALU wordt nu meerdere keren ingezet voor verschillende taken tijdens de uitvoering van bepaalde instructies. Om dezelfde reden kunnen ook het instructie geheugen (ROM) en en het data geheugen (RAM) tot één (instruction and data) geheugen (RAM) worden samengevoegd. De gegevens die in een bepaalde klokcyclus van een instructie geproduceerd zijn moeten echter bewaard worden in additionele registers, zodat deze waarden gebruikt kunnen worden in volgende klokcycli. Ook informatie van verschillende bronnen moet nu op verschillende momenten de ALU, memory en register file inputs bereiken en daarvoor zijn additionele multiplexers nodig. Omdat het uitvoeringspad van elke instructie nu in enkele stappen is onderverdeeld moeten de operaties in elke stap en overgangen tussen de stappen op een juiste manier uitgevoerd worden. Hiervoor wordt een sequentiële machine (finite state machine - FSM) gebruikt. De controller is nu dus sequentieel en niet combinatorisch.

Opdrachten

Dit lab bestaat uit vier opdrachten. In de eerste drie opdrachten ga je het multi-cycle mini-mini MIPS model afmaken. Dit model ga je vervolgens in de vierde opdracht testen. Alleen de **vierde** opdracht

moet afgetekend worden.

Het is heel belangrijk dat je de namen van modules en poorten precies overneemt van de opdracht. Wijzig ook geen hoofdletters of kleine letters. Als je dit wel doet zal de mmMIPS niet meer compileren. Let dus goed op dat je alle namen precies zo schrijft zoals ze in de opdrachten staan.

1. SystemC modules ontwerpen

In deze eerste opdracht ga je het SystemC model van de multi-cycle mmMIPS installeren. Een aantal modules ontbreken nog in dit model. Deze modules ga je in deze opdracht zelf ontwerpen.

1. Download het bestand multi_cycle_mmmips.zip van www.es.ele.tue.nl/education/5JJ55-65/mmips-lab/labs/2/ en pak het bestand uit in de folder *d:\mmips\mips*.

2. Open in *Visual C++* het project *multi_cycle_mmmips*. Dit project staat in de folder *d:\mmips\mips\multi_cycle_mmmips*. Het project bevat een incompleet model van de multi-cycle mmMIPS.

3. Bestudeer het schema van de multi-cycle mmMIPS. De modules en signalen die nog niet gedefinieerd zijn in het SystemC model dat je hebt geopend in *Visual C++* zijn in **rood** aangegeven. In de volgende opdracht ga je deze modules en signalen definieren en de modules aansluiten op de rest van het systeem. Voordat je hiermee kan beginnen moet je eerst een drietal ontbrekende module types declareren. Het gaat hier om de typen: MUX3, MUX4 en JUMPADDR.

   Open de bestanden *mux.cpp* en *mux.h* en declareer naar analogie van het MUX2 type de SC_MODULE types MUX3 en MUX4. Open vervolgens de bestanden *shift.cpp* en *shift.h* en declareer een SC_MODULE van het type JUMPADDR. De input van deze module zijn de huidige waarde van de program counter (uitgang van het register PC) en de huidige instructie (uitgang van het register IR). JUMPADDR combineert deze twee inputs tot het doel adres van de jump instructie. In het boek "Computer Organization & Design: The Hardware / Software Interface"van D.A. Patterson en J.L. Hennessy kun je opzoeken hoe je de ingangssignalen precies moet combineren.

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

Het multi-cycle mmMIPS model dat je geïnstalleerd hebt in de vorige opdracht is nog niet klaar. Een aantal modules en signalen zijn nog niet gedefinieerd. Deze modules en signalen zijn in **rood** aangegeven in het schema van de multi-cycle mmMIPS. In deze opdracht ga je deze modules en signalen definiëren. Daarnaast ga je alle modules op elkaar aansluiten.

1. Open het bestand *main.cpp* uit het project *multi_cycle_mmmips* in *Visual C++*.

2. Definieer de ontbrekende modules in *main.cpp*. Geef de modules dezelfde naam als aangegeven in het schema van de multi-cycle mmMIPS.

3. Definiëer 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 multi-cycle mmMIPS heten bijvoorbeeld *bus_<module naam>* of *bus_<module naam>_<output poort naam>* als een module meerdere output poorten heeft.

4. Sluit alle nieuwe signalen aan op de juiste poorten van de verschillende modules in het multi-cycle mmMIPS model.

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

3. Afronden implementatie state-machine in de controler

De controller die gedeclareerd wordt in de bestanden *ctrl.cpp* en *ctrl.h* is nog niet compleet. Alleen de eerste toestand, *INSTR_FETCH*, van de finite state machine is geïmplementeerd. In deze opdracht ga je de ontbrekende delen van de finite state machine invullen.

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

2. Bestudeer de sensitivity lijst van de module CTRL. Bekijk op basis van welk signaal de toestand van de finite state machine verandert. Bestudeer ook hoe de *INSTR_FETCH* is geïmplementeerd in de functie *CTRL::next_ state_output()*.

3. Implementeer nu de ontbrekende states van de controller. Hierbij kun je gebruik maken van de afbeelding van de controller zoals deze staat in het boek "Computer Organization & Design: The Hardware / Software Interface"van D.A. Patterson en J.L. Hennessy. De volgende toestand die vanuit de toestanden *INSTR_DECODE* en *MEM_ADDRESS* bereikt wordt hangt af van de opcode (variable **op** in de functie *CTRL::next_state_output*). Gebruik de variable **op** en **if/else** statements om de juiste volgende toestand te bepalen.

4. Testen van het multi-cycle mmMIPS model

In deze opdracht ga je de werking van het SystemC model van de multi-cycle mmMIPS testen. Dit doe je door het programma dat de berekening a=(b+c)*d uitvoert simuleren op het door jou gemaakte multi-cycle mmMIPS model. De assembler code van dit programma is hieronder weergegeven. Je kunt deze code ook vinden in het bestand *mult.asm*. De corresponderende hexadecimale programma code voor dit programma staat in *mult.bin*. Dit bestand bevat ook de input gegevens (b=30, c=70 en d=5) voor de berekening. De programma code begint in *mult.asm* op adres 0x00 en de invoer gegevens beginnen op 0x40. Aan het einde van de simulatie wordt de inhoud van het geheugen opgeslagen in het bestand *mips_mem.dump*.

mult.asm

```
.set noat
.set noreorder
.align 2

lw $16, 0x40($0)         # constant 1 (loop increment)
lw $17, 0x44($0)         # b = memory[0x44]
lw $18, 0x48($0)         # c = memory[0x48]
lw $19, 0x4C($0)         # d = memory[0x4C]
add $17, $17, $18        # b += c
sub $20, $20, $20        # result = 0
sub $21, $21, $21        # count = 0
LOOP:
beq $21, $19, LOOPEND    # if count == d goto END
add $20, $20, $17        # result += (b+c)
add $21, $21, $16        # count += 1
beq $1, $2, LOOP         # goto LOOP
LOOPEND:
sw $20, 0x50($0)         # memory[0x50] = result
END:
beq $0,$0,END            # infinite loop to stop operation
```

Om het bovenstaande programma te simuleren op de multi-cycle mmMIPS moet je de volgende stappen uitvoeren:

1. Compileer het project *multi_cycle_mmmips* met *Visual C++* met behulp van **Build** en **Build Solution** van het *Visual C++* menu.

2. Kopieer *mult.bin* naar *mips_mem.bin*.

3. Voer het commando *./multi_cycle_mmmips.exe 3000* uit.

4. Bekijk de resultaten met *HDD Hex Editor* en *WinWave* en controleer of jouw SystemC model van de multi-cycle mmMIPS correct werkt. Corrigeer zo nodig de fouten die je ontdekt.