# Lab 4 - Modelleren en simuleren van de pipelined 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 mMIPS,
- het modelleren van de pipelined mMIPS hardware in SystemC (RTL-niveau),
- de C compiler van de mMIPS (LCC) .

Daarvoor wordt het (synthetiseerbare) SystemC model gebruikt van de pipelined mini MIPS processor (**mMIPS**) en de software beschreven in "Software mMIPS-lab".

Tijdens dit lab gaan de studenten het SystemC model van de mMIPS uitbreiden met forwarding logica en de structuur en werking van de mMIPS analyseren. De mMIPS 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. Het is de bedoeling, dat de studenten dit boek tijdens dit lab bij zich hebben en het (zo nodig) kunnen raadplegen.

## Opdrachten

Dit lab bestaat uit twee opdrachten. In de eerste opdracht ga je kennismaken met de LCC compiler. Deze compiler kun je gebruiken om een C programma te compileren voor de mMIPS. In de tweede opdracht ga je het mMIPS model uitbreiden met forwarding logica. Alleen de **eerste** opdracht moet worden afgetekend. De **tweede** opdracht hoeft nu dus nog niet afgetekend te worden. Het resultaat van deze opdracht heb je echter wel nodig voor de mMIPS opdracht later in dit blok.

### 1. Kennismaking met de C compiler

In deze eerste opdracht ga je kennismaken met de mMIPS processor en zijn compiler (*LCC*). Hiertoe ga je het onderstaande C programma compileren en vervolgens simuleren op de mMIPS.
*1.c*

```
int *mem = (int*)(0x400080);

void main(void)
{
    int a,b,c,d;

    a = mem[0];
    b = mem[1];
    c = mem[2];

    if ((a & 1) == 0)
    {
        d = a + b;
        d = d + c;
    }
    else
```

```
    {
        d = a + c;
        d = d - b;
    }

    mem[3] = d;
}
```

Voer nu de volgende stappen uit om het bovenstaande programma op de mMIPS te draaien:

1. Download het bestand mmips.zip van www.es.ele.tue.nl/education/5JJ55-65/mmips-lab/labs/4/ en pak het bestand uit in de folder *d:\mmips\mips*.
2. Open in *Visual C++* het project *mmips*. Dit project staat in de folder *d:\mmips\mips\mmips*. Het project bevat een compleet model van de mMIPS.
3. Bestudeer het schema van de mMIPS. Bekijk de verschillen en overeenkomsten met het model van de pipelined mmMIPS waarmee je vorige week hebt gewerkt.
4. Open het bestand *main.cpp* uit het project *mmips* in *Visual C++* en compileer de mMIPS.
5. Open een Cygwin shell en type het volgende commando:
   *cd d:/mmips/mips/mmips*
6. Compileer het programma *1.c* door het volgende commando uit te voeren:

   *lcc -o mips_mem.bin 1.c*

   Als je de bestanden van dit practicum niet op de standaard locatie (*d:\ mmips*) hebt geinstalleerd, dan werkt jouw lcc compiler mogelijk niet goed. In dat geval moet je de stappen uitvoeren die beschreven staan op: www.es.ele.tue.nl/education/5JJ55-65/mmips/lcc/recompile.php. Zorg er ook voor dat de omgevingsvariable *LCCDIR* wijst naar de directory waar jouw lcc compiler staat. Op www.es.ele.tue.nl/education/Computation /mmips-lab/software/environment.php kun je zien hoe je deze omgevingsvariable moet instellen.
7. Disassembleer het bestand *mips_mem.bin* door het volgende commando uit te voeren:

   *disas mips_mem.bin > out.asm*

   Het gedisassembleerde programma wordt nu niet in cygwin weergegeven, maar in plaats daarvan wordt het opgeslagen in het bestand *out.asm*.
8. Open het bestand *out.asm* in een text editor en bekijk het bovenste gedeelte van het gedisassembleerde programma. Het programma begint net als alle vorige keren op adres 0x0. De main routine begint op het adres 0xa0 en eindigt op 0x12c. Zoek de relatie tussen de assembler code van de main routine en de C code van deze routine. Welke assembler instructies horen bij de verschillende C statements? Je zult zien dat de volgorde van assembler instructies soms afwijkt van de volgorde van de C statements. Bij welk statement hoort bijvoorbeeld de instructie op adres 0xd0?
9. Bepaal nu met behulp van de assembler code op welk adres in het geheugen de variabelen *mem[0]* en *mem[1]* staan.
10. Open het bestand *mips_mem.bin* in *HDD HexEditor* en zet zorg ervoor dat in de geheugen locaties van *mem[0]*, *mem[1]* en *mem[2]* respectievelijk de waarden 7, 45 en 13 staan. Sla vervolgens het bestand *mips_mem.bin* op met dezelfde naam en sluit *HDD HexEditor* af.
11. Simuleer nu het programma op het SystemC model van de mMIPS door in cygwin het volgende commando uit te voeren:

    *./mMIPS.exe*
12. Bestudeer met *WinWave* de werking van de mMIPS. Kijk met name naar de werking van de pipeline in de situatie dat er een data hazard optreed.

## 2. Toevoegen van forwarding logica

In de pipelined mMIPS processor kunnen data hazards optreden. 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 nodige gegevens kunnen in de meeste gevallen op tijd direct 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 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). In de huidige mMIPS wordt echter voor alle data hazards gewacht en niet alleen voor de load instructie. In deze opdracht ga je de mMIPS uitbreiden met forwarding logica uitbreiden, zodat er alleen nog op de load instructie gewacht moet worden.

1. Voeg forwarding logica toe om data hazards in the EX stage op te lossen.
2. Schrijf een klein programma in C waarmee je de forwarding logica kan testen. Je kunt natuurlijk ook het programma gebruiken waar je in de eerste opdracht van lab 4 mee gewerkt hebt.
3. Controleer of de forwarding logica correct functioneert. Indien nodig moet je natuurlijk je fouten verbeteren.
4. Voeg nu logica toe om de resterende data hazards (voor zover mogelijk) op te lossen en test daarna deze nieuwe logica.