Practicum 5

Gepijplijnde architectuur

Voor dit practicum wordt gebruik gemaakt van de ESCAPE-simulator. Er zijn versies van de simulator beschikbaar voor diverse platformen: Windows, Linux en OS-X. De nodige bestanden om de simulator uit te voeren kun je vinden op in de directory van dit practicum op de Linux Live Image.

Ditmaal zullen we ons toeleggen op een gepijplijnde architectuur. Vergeet niet voor de aanvang van het practicum de handleiding (in het bijzonder het gedeelte over de pijplijnarchitectuur) door te nemen, alsook de cursusslides nog even te bestuderen, zodat je vertrouwd bent met het gebruik van de simulator en met de context van het practicum.

Het programma dat in dit practicum uitgevoerd zal worden ziet er initieel als volgt uit:

adr	instruct	label	symbolische instructie
0000	00000000		NOP
0004	44010080		ADDI R0, 0x0084, R1
0008	440F0004		ADDI R0, 0x0004, R15
000C	44030000		ADDI R0, 0x0000, R3
0010	44020000		ADDI R0, 0x0000, R2
0014	202F0800	loop	SUB R1, R15, R1
0018	0C240000		LDW R4, 0x0000(R1)
001C	18240100		STW R4, 0x0100(R1)
0020	1C441000		ADD R2, R4, R2
0024	24842000		MUL R4, R4, R4
0028	18240200		STW R4, 0x0200(R1)
002C	1C641800		ADD R3, R4, R3
0030	7801FFE0		BRGT R1, loop
0034	00000000		NOP
0038	00000000		NOP
003C	180200FC		STW R2, 0x00FC(R0)
0040	180301FC		STW R3, 0x01FC(R0)
0044	44011234		ADDI R0, 0x1234, R1

Het programma beschikt over een datagebied van adres `0x000` tot en met adres `0x25F`. De inhoud van dit gebied bevat de volgende informatie.

adr	data							
000	00000000	00000001	00000002	00000003	00000004	00000005	00000006	00000007
020	00000008	00000009	0000000A	0000000B	0000000C	0000000D	0000000E	0000000F
040	00000010	00000011	00000012	00000013	00000014	00000015	00000016	00000017
060	00000018	00000019	0000001A	0000001B	0000001C	0000001D	0000001E	0000001F
080	12345678	9ABCDEF0	00000000	00000000	00000000	00000000	00000000	00000000
0A0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0C0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0E0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
100	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
120	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
140	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
160	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
180	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
1A0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
1C0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
1E0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
200	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
220	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
240	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

De instructies in het programma hebben de volgende betekenis. Merk op dat het doeloperand van de ADDI, ADD en SUB hier het derde operand is, en niet het eerste operand!

symbolische instructie		betekenis
ADDI	R1,n,R2	R2 ← R1 + n
ADD	R1,R2,R3	R3 ← R1 + R2
SUB	R1,R2,R3	R3 ← R1 - R2
MUL	R1,R2,R3	R3 ← R1 * R2
LDW	R1,n(R2)	R1 ← woord (32 bit) op adres (R2+n) (load)
STW	R1,n(R2)	woord (32 bit) op adres (R2+n) ← R1 (store)
NOP		no-operation, m.a.w., doe niets
BRGT	R1,label	spring naar label indien R1 groter is dan nul.

Het programma voert de volgende stappen uit: (1) initialiseren van een aantal registers, (2) laden van een een rij bestaande uit 32 woorden (4-bytes per woord)uit het geheugen, (3) kopieren van deze rij, (4) schrijven van het kwadraat van elk woord naar het geheugen, (5) berekenen van de som van de originele rij in register R2 en de som van de gekwadrateerde rij in register R3, en tenslotte (6) wegschrijven van beide sommen op een welbepaald vast adres.

Start de ESCAPE-simulator en kies meteen *pipelined architecture*. Open het project `pijplijn.ppr`. Voer het programma stap voor stap uit en probeer te begrijpen op welke manier het programma door de pijplijn stroomt. Je zet hiervoor in de opties de forwarding af, de delay slots af (no delay slots), en de data memory access time op 1. Op deze manier kun je duidelijk zien op welke plaatsen het programma blokkeert en staat te wachten op gegevens die nog niet in het registerbestand aangekomen zijn.

Test: bepaal de totale uitvoeringstijd van het programma. Hiervoor vink je *multiple cycles* aan en zet je de waarde op 600. Bovendien kies je via *View - Breakpoints* een breakpoint op R1=0x1234. Na het aanklikken van reset, en clock loopt het programma totdat de pijplijn de waarde 1234 in register 1 probeert te schrijven (op adres 40). Op dat ogenblik kun je in *time* het aantal verstreken klokcycli aflezen. Hoeveel cycli lees je af? Oplossing: 540.

Voor deze opgave is het de bedoeling – tenzij anders expliciet vermeld wordt – dat je voor elke vraag telkens vertrekt van de oplossing uit de vorige vraag. Daarom is het dus best dat je bij elke vraag de resulterende code ook eens kopieert in een afzonderlijk bestand, zodat je hier later kan naar verwijzen en op terugvallen indien nodig.

5.1 Opgave

1. Ga in het datageheugen (via `F6`) na welke waarde weggeschreven wordt door de instructies die zich bevinden op de volgende twee instructieadressen:

   i) 0x3C

   ii) 0x40

2. De pijplijn blokkeert verschillende keren omdat gegevens niet tijdig in het registerbestand geschreven worden. Bekijk deze blokkeringen stapsgewijs met de pipeline activity en usage vensters. Door NOP-instructies in te voegen kunnen we het gebruik van een register uitstellen. Voeg NOP-instructies toe totdat er geen blokkeringen meer voorkomen (`F5`, invoegen van een NOP-instructie doe je met enter in insert-mode). Het totale aantal uitvoeringscycli mag hierbij uiteraard niet toenemen. Is deze operatie zinnig, of onzinnig?

3. We gaan nu terug naar het originele programma. Zet de forwarding aan en voer het programma opnieuw uit. Probeer te begrijpen op welke manier waarden nu via forwarding doorgestuurd worden. Hoeveel cycli heeft het programma nu nodig om uit te voeren?

4. Zelfs met forwarding blijft er nog 1 blokkering over. Je kunt deze blokkering vermijden door het programma aan te passen. Maak deze aanpassing. Hoeveel cycli bekom je dan?

5. i) Zet nu de delay slots aan (double delay slot). Hoeveel uitvoeringscycli krijg je nu?

   ii) Kun je het programma wijzigen zodat je nuttig gebruik maakt van de instructies in het delay slot? Hoeveel cycli bekom je dan?

6. Zet nu de memory access time op 3 en voer het programma opnieuw uit. Hoeveel cycli bekom je nu? Kun je deze blokkeringen wegwerken of niet? Waarom (niet)?

7. Vanaf nu laat je de memory access time op 3 staan. Een mogelijke statische optimalisatiemethode is lusuitvouwing, waarbij een aantal iteraties van een lus samengenomen worden om uiteindelijk de instructies efficiënter door de pijplijn te laten stromen. Gelet op het feit dat het origineel programma 33 iteraties uitvoert, wat geen veelvoud is van 4, zullen we eerst een enkele iteratie buiten de lus brengen. Dit werd reeds gedaan in `pijplijn7.cod`. Vertrek van dit programma en breng 4 instanties van de instructies binnen de lus in 1 iteratie van de lus (dit kan omdat we nog 32 iteraties moeten uitvoeren, wat een viervoud is). Hoeveel cycli win je door deze veronderstelling?

8. Nu kun je het groter stuk sequentiële code in de lus uit opgave 7 beter optimaliseren dan de kleine lus. Herschik en herschrijf de code zodat je een zo snel mogelijk programma bekomt. Teken de afhankelijkheidsgraaf van de instructies en hou rekening met de timing informatie om gemakkelijk in te zien hoe de instructies herschikt kunnen worden. Stel de afhankelijkheidsgraaf van je geoptimaliseerde code op. Stel deze graaf zó voor dat het duidelijk is in welke volgorde de instructies worden uitgevoerd.

   Je mag zoveel registers gebruiken als je zelf nodig acht. Verder mag je ook het double delay slot benutten en moet je gebruik maken van forwarding. Zorg er wel voor dat het programma correct blijft werken! Dat betekent ook dat de juiste data naar de juiste lokatie in het geheugen geschreven wordt.

   Hoeveel cycli bekom je bij deze code ?

   Waarom is het belangrijk dat we hier forwarding gebruiken? Tracht eventueel dezelfde oefening te herhalen zonder forwarding om op deze vraag te antwoorden.