Practicum 6

Microcode

6.1 Inleiding

Dit practicum zal gebeuren aan de hand van de ESCAPE-simulator. Het is nuttig om de handleiding van deze simulator (gedeelte over microprogrammering) grondig te bestuderen voor aanvang van het practicum. 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 Minerva.

Je moet zowel je code als een verslag indienen voor (**zondag 5 mei 2013 om 22:00:00**) via `https://indianio.ugent.be`. Let op! We verwachten per groep één enkel zip bestand waarin de volgende bestanden zijn opgenomen.

- `practicum6_verslag.pdf`
- `practicum6_vraag1.cod`
- `practicum6_vraag1.mco`
- `practicum6_vraag2.cod`
- `practicum6_vraag2.mco`
- `practicum6_vraag3.cod`
- `practicum6_vraag3.mco`
- `practicum6_vraag4.cod`
- `practicum6_vraag4.mco`

Zoals steeds gelden de gebruikelijke afspraken voor het indienen.

6.2 Werken met ESCAPE

Om vertrouwd te geraken met de simulator kan je de volgende werkwijze volgen.

1. In de directory `pract06` vind je het uitvoerbaar bestand `Escape`. Je kunt dit programma opstarten met het script `start-escape.sh`. Voor de geïnteresseerde student: dit script zet het pad goed voor dynamische geladen bibiotheken, zodat de bibliotheek `libQt4Pas.so.5` gevonden wordt.

2. Ga meteen naar *microprogrammed architecture* en open het project `microcode.mpr` via *File – Open Project*.

3. Bekijk het assemblerprogramma met *View Instruction Memory*. Je krijgt het volgende te zien:

adr	instruct	label	symbolische instructie
0000	10020010		ADDI R2,0x0010,R0
0004	10010000		ADDI R1,0x0000,R0
0008	10060004		ADDI R6,0x0004,R0
000C	18000018		JUMP TEST
0010	04440100	LOOP	LD R4,0x0100(R2)
0014	04450200		LD R5,0x0200(R2)
0018	1CA41800		MULT R3,R4,R5
001C	1C631800		MULT R3,R3,R3
0020	0C610800		ADD R1,R1,R3
0024	28461000		SUB R2,R2,R6
0028	1440FFE4	TEST	BRGE R2,LOOP
002C	080100FC		ST 0x00FC(R0),R1
0030	00000000		NOP

   Dit programma laadt de constante 16 (hexadecimaal 10) in register R2, de constante 0 in R1, en de constante 4 in register R6, en springt dan naar het einde van de lus, gemarkeerd door het label TEST, op adres 0x24. Merk op: register R0 bevat steeds de waarde 0. Het doeloperand is steeds het eerste operand.

   Indien R2 groter of gelijk is aan nul, wordt er naar het label LOOP gesprongen op adres 0x10 en wordt de lus hernomen. In deze lus worden registers R4 en R5 geladen uit het geheugen op adressen 0x100+R2 en 0x200+R2. De eerste lusiteratie zijn dit dus de adressen 0x110 en 0x210. De daar gelezen waarden worden vermenigvuldigd en het kwadraat hiervan wordt opgeteld bij R1. Vervolgens wordt R2 verminderd met R6 (in dit voorbeeld bevat R6 de waarde 4). Dit gaat zo door totdat R2 kleiner dan 0 is (vijf iteraties dus). Uiteindelijk wordt de som van de producten geschreven op adres 0xFC(R0), dit is het adres 0xFC want R0 heeft steeds de waarde 0.

4. Open nu het microcodevenster (*View Microcode*) en voer het het programma uit door steeds op clock te klikken en kijk hoe het microprogramma uitgevoerd wordt. Probeer per stap te begrijpen op welke manier de verschillende controlepunten aangestuurd worden. In het microcodescherm kan je ook de sprongtabellen zien.

5. Ga na hoeveel cycli het duurt alvorens de laatste instructie van het programma (op adres 0x2C afgewerkt is). *Oplossing: 184*.

Je kan het programma uit de inleiding veranderen door instructies te editeren. Het invoegen van een instructie gebeurt door in insert mode te gaan, en dan op return te drukken. Er wordt dan een nop-instructie toegevoegd die vervolgens kan overschreven worden door de gewenste instructie.

6.3 Opgave

1. Voeg de kwadratering en de optelling samen tot 1 instructie: multiply-square-and-accumulate (MSAC). Probeer deze instructie te definiëren in het microcodevenster (het label is reeds aanwezig), en vervang nadien de twee instructies in het instructievenster door MSAC R1,R4,R5. Vergewis jezelf van het feit dat het programma nog steeds hetzelfde doet. Hoeveel cycli vereist het programma nu?

2. De tweede oefening bestaat erin om de SUB en de BRGE te vervangen door een DBRGE-instructie (decrement and branch if greater than or equal), met als syntaxis: DBRGE R2,R6,getal, waarbij het getal de offset is die bij de nieuwe PC moet opgeteld worden (b.v. -10 of 0xfff0 in hexadecimaal) om terug naar het begin van de lus te springen. Het is nu wellicht beter om de jump bij het begin van de lus gewoon weg te laten om te vermijden dat de lus 1 keer te weinig uitgevoerd wordt. Een andere mogelijkheid is om de initiële waarde van R2 op 0x14 te zetten. Hoeveel cycli zijn nu nodig?

3. Open nu het project `microcode_memcmp.mpr` met assemblercode die R3 geheugenwoorden vergelijkt, en het resultaat van deze vergelijking terugschrijft in register R3. Deze vergelijking gebeurt als volgt: de overeenkomende geheugenwoorden in beide geheugenbereiken worden van elkaar afgetrokken. Indien dit resultaat verschillend is van 0, wordt dit resultaat teruggegeven als eindresultaat. Enkel indien beide geheugenbereiken volledig gelijk zijn, zal het eindresultaat 0 zijn. De twee te vergelijken geheugenbereiken starten op de adressen bewaard in registers R1 en R2:

adr	instruct	label	symbolische instructie
0000	10030008		ADDI R3,0x0008,R0
0004	10010100		ADDI R1,0x0100,R0
0008	10020120		ADDI R2,0x0120,R0
000C	2C600028	BEGIN	BRLE R3,END
0010	04250000		LD R5,0x0000(R1)
0014	04460000		LD R6,0x0000(R2)
0018	28C52800		SUB R5,R6,R5
001C	38A00010		BRNE R5,SETRESULT
0020	10210004		ADDI R1,0x0004,R1
0024	10420004		ADDI R2,0x0004,R2
0028	1063FFFF		ADDI R3,0xFFFF,R3
002C	1800FFDC		JUMP BEGIN
0030	0C051800	SETRESULT	ADD R3,R5,R0
0034	18000004		JUMP DONE
0038	10030000	END	ADDI R3,0x0000,R0
003C	34000000	DONE	HALT

Deze code bevat echter een instructie die nog niet in de microcode gedefinieerd is: de BRNE instructie, die een sprong neemt als het meegegeven register verschillend is van 0. Implementeer deze instructie in microcode. Verifieer dat de resulterende code doet wat er van verwacht wordt door enkele geheugenbereiken met elkaar te vergelijken, en de waarden in R1, R2, en R3 te bestuderen. We hebben voor jullie reeds enkele waarden ingevuld in het geheugen van elk 8 woorden lang vanaf adres 0x100.

Hoeveel cycli doet je code er over als deze twee identieke geheugenbereiken van 8 woorden elk moet vergelijken?

4. Definiëer in microcode de instructie MEMCMP R3,R1,R2 die R3 woorden vergelijkt van adres R2 naar adres R3. Vervang de overeenkomstige assemblercode door MEMCMP R3,R1,R2. Zorg ervoor dat je code ook correct werkt als de toegangstijd tot het geheugen verschillend is van 1, dit kan je instellen via Options-Memory Access Time.

Hoeveel cycli heeft jouw implementatie nu nodig om twee identieke geheugenbereiken van 8 woorden elk te vergelijken?