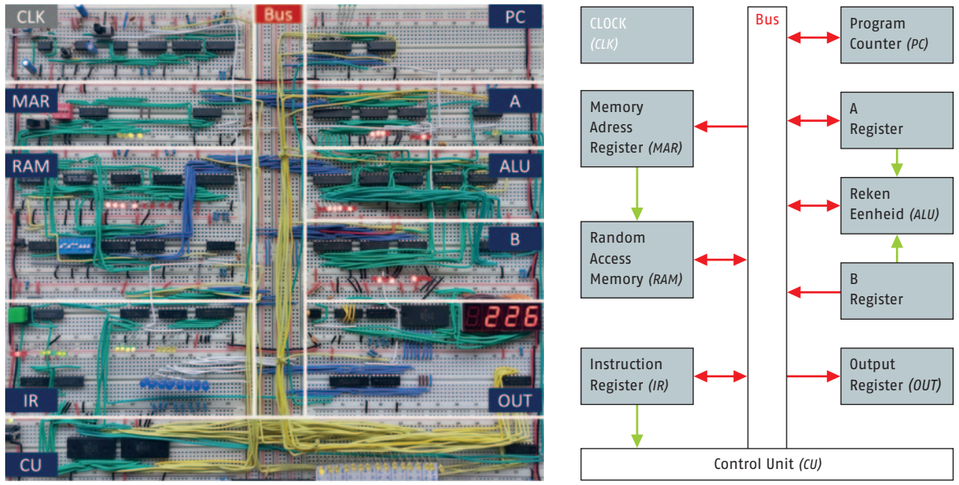
‘Even’ iets opnieuw uitrekenen?

Actuarissen houden van rekenen. Computers stellen hen in staat complexe berekeningen te maken, zoals bij de transitie naar het nieuwe pensioenstelsel. Een beter begrip van hoe de computer de berekeningen uitvoert, stelt de actuaris in staat om dezelfde berekeningen sneller uit te voeren, of om in dezelfde tijd meer berekeningen te maken.

De wens om actuariële berekeningen sneller uit te voeren was de aanleiding om een kijkje te nemen onder de motorkap van de computer. De laptop die wij gebruiken voor onze dagelijkse werkzaamheden staat een dergelijk inwendig onderzoek helaas niet toe. Dat komt doordat de instructies die onze laptop (hierna: C64) op laag niveau uitvoert, door het besturingssysteem worden afgeschermd. Daarom maken we in dit artikel kennis met de C8, een spartaanse maar volledig ‘open’ computer, samengesteld uit onderdelen uit de jaren 70 van de vorige eeuw. Het inzicht dat de C8 verschaft, helpt om actuariële berekeningen te versnellen.

Figuur 1 toont de C8. Elke module heeft hierin zijn eigen verantwoordelijkheid. De Program Counter (PC) houdt het adres bij van de volgende instructie. In het geheugen (RAM) staat het uit te voeren programma.



Figuur 1: de C8 (links) en de globale bouwtekening daarvan (rechts)

De *Arithmetic Logical Unit* (ALU) voert rekenkundige bewerkingen uit. De C8 bevat vijf registers waarvan de inhoud per (klok)cyclus – waarover verderop meer – wijzigt. Het *Memory Address Register* (MAR) houdt een pointer bij naar de actuele instructie in RAM, het *Instruction Register* (IR) bevat de taak die op dat moment wordt verwerkt, het *Output Register* (OR) bevat de gegevens die getoond moeten worden op het display en in de algemene registers (A en B) worden de actuele gegevens klaargezet ter verwerking door de ALU. Op basis van de actuele instructie bepaalt de Control Unit (CU) als een soort manager van een telefooncentrale welke twee modules per cyclus via de bus met elkaar worden verbonden. De rode verbindingen in figuur 1 worden daarbij selectief, in één richting, open gezet. De groene verbindingen staan altijd open. De C8 is een 8-bits computer. Per cyclus stroomt 1 byte (=8 bits) aan data via de bus van de ene naar de andere module.

Drs. P.J.T. Marres AAG is als zelfstandig data actuaris werkzaam bij QuantSense BV.

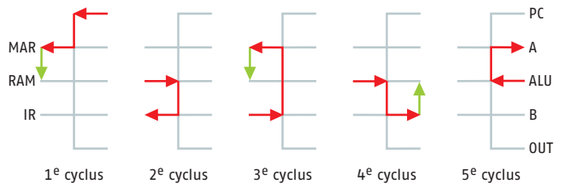


Adres	Broncode	Objectcode	Aantal cycli
0	LDA 14	0001 1110	4
1	ADD 15	0010 1111	5
2	OUT	1110 0000	3
3	HLT	1111 0000	3
14	88	0101 1000	--
15	138	1000 1010	--
		totaal	**15**

Tabel 1: programma dat de getallen 88 en 138 bij elkaar optelt

OPTELLEN IN 15 CYCLI

Bij wijze van voorbeeld vragen we de C8 om de getallen 88 en 138 bij elkaar op te tellen. Tabel 1 toont de broncode: de instructie LDA 14 kopieert het getal in geheugenlocatie 14 (=88) naar het A-register, ADD 15 kopieert het getal naar geheugenlocatie 15 (=138) in het B-register. De ALU telt nu beide getallen direct bij elkaar op. Met OUT wordt het resultaat (=226) getoond op het display en HLT zorgt er voor dat het programma stopt. Deze broncode is handig als geheugensteuntje voor de programmeur. De processor begrijpt echter alleen nullen en enen. Daarom wordt de broncode omgezet naar objectcode. We zitten nu bijna op het laagste niveau van de machine. We kunnen nog één spade dieper. Elke instructie bestaat namelijk uit meerdere cycli. De instructie ADD 15 bestaat bijvoorbeeld uit vijf cycli, zie figuur 2. Hierin zien we dat er bij elke cyclus steeds een andere verbinding wordt gelegd tussen twee modules. Voor een simpele optelling (waar de ADD-instructie een onderdeel van is) heeft de C8 dus 15 cycli nodig (tabel 1).



Figuur 2: de vijf cycli van de ADD-instructie

KORTOM: MINDER CYCLI = SNELLER REKENEN

In algemene zin wordt de doorlooptijd T (in seconden) van een programma benaderd door de volgende formule:

$$T = \text{aantal_cycli} / \text{kloksnelheid}$$

Een programma dat bijvoorbeeld 8 biljoen cycli nodig heeft, draait op een machine met een 3,5 GHz processor ca. $8 \times 10^{12} / (3{,}5 \times 10^{9}) = 38$ minuten. Voor het gemak gaan we bij deze benadering voorbij aan enkele geavanceerde ‘trucs’ die moderne processoren in huis hebben om cycli sneller te verwerken, zoals pipelining. Dit laat onverlet dat we een programma in algemene zin kunnen versnellen door het in minder cycli te laten uitvoeren.

MAAR ONZE C64 WERKT TOCH TOTAAL ANDERS DAN DE C8?

Hoe kunnen we de inzichten uit de C8 benutten om actuariële berekeningen sneller uit te voeren op de C64 die we gebruiken voor onze dagelijkse werkzaamheden? Op het eerste gezicht lijkt de C64 namelijk in geen velden en wegen op de C8, die zelfs geen toetsenbord, muis of scherm heeft! Ook hebben we gezien dat de C8 op zeer laag niveau moet worden aangestuurd, namelijk door machinecode met dipswitches[1] in RAM in te voeren. De C64 is veel handiger: de gebruiker geeft opdrachten via toetsenbord en muis in een hoge programmeertaal of grafische (spreadsheet) interface; de C64 zorgt daarna zelf voor de doorvertaling naar machinecode en daarna weer terug naar de gebruiker. Daarnaast zit het geheugen van de C64 geavanceerder in elkaar. Waar het geheugen bij de C8 nog een homogeen geheel is, bestaat het bij de C64 uit een hiërarchie van verschillende lagen. Hoe hoger de data in de geheugenhiërarchie verblijven, hoe sneller toegankelijk voor de processor. Waar het ‘hoge’ geheugen sneller toegankelijk is, heeft het ‘lage’ geheugen het voordeel van een hogere opslagcapaciteit en een lagere prijs. Een typische design trade-off dus. Ondanks al deze verschillen tussen de C8 en de C64 blijft de basis hetzelfde: minder cycli leidt meestal tot snellere berekeningen. Dit principe gaan we nu toepassen op een actuariële berekening.

INVAARBEREKENINGEN 3X VERSNELLEN

Invaarberekeningen zijn gebaseerd op somproducten van meerdere vectoren, denk aan overlevingskansen, stapfuncties en disconteringsfactoren. Een voor de hand liggende (en al redelijk goede) implementatie is om de vectoren eerst elementsgewijs met elkaar te vermenigvuldigen en het tussenresultaat daarna te sommeren. Deze strategie in twee stappen voldoet prima bij het maken van een beperkt aantal invaarberekeningen. Bij de transitie van een pensioenfonds naar Wtp willen we echter invaarberekeningen maken voor een groot aantal deelnemers. De vermenigvuldiging resulteert dan in een array die niet meer past in het ‘hoge’ geheugen. Het tussenresultaat van de vermenigvuldiging wordt tijdelijk opgeslagen in het ‘lage’ geheugen, dat veel trager benaderbaar is. Door de vermenigvuldiging en de optelling te combineren, hoeft de processor niet meer te zoeken in het lage geheugen (jargon: ‘cache misses’) en wordt het aantal cycli met ongeveer een factor 3 gereduceerd. Conform de hiervoor beschreven formule zijn we daardoor 3x sneller klaar.

Zowel Python als R bieden met de Einstein sommatie een goede implementatie van deze cycli-besparende strategie.

CONCLUSIE

De C8, een spartaanse maar geheel open computer, laat zien hoe elke opdracht die we onze computer geven, intern wordt vertaald naar hele basale opdrachten, waar de processor mee uit de voeten kan. Met dit inzicht kunnen we actuariële berekeningen significant versnellen, of we kunnen meer berekeningen uitvoeren in dezelfde tijd. ■

1 – De dipswitches waarmee de objectcode wordt geprogrammeerd, zijn in figuur 1 zichtbaar in de MAR- en RAM-module.