16 BIT COMPUTER

IPASS project 2018

Over hoe een computer werkt

Dit project is bedoeld om te laten zien hoe een computer werkt. Dit kan gebruikt worden om inzicht te geven aan mensen die geïnteresseerd zijn in computer engineering Alle resources, software, schema's en broncode zijn vrij te gebruiken voor educatieve doeleinden

Jim van Leeuwen
Jim.vanleeuwen@student.hu.nl
29-6-2018
IPASS-2018 Hogeschool Utrecht

Introductie

Een computer is een dom apparaat dat maar twee dingen kan, bits kopiëren naar een andere plaats en bits vergelijken/bewerken volgens simpele regels. Echter zijn tegenwoordig computers erg complex, snel en krachtig. Het is daardoor lastig om terug naar de basis te gaan om te zien wat er gebeurt. Om dit principe duidelijk te maken is het goed om de werking eenvoudig uit te kunnen leggen. Het doel van het project is om een 16 bits computer te bouwen die gebruikt kan worden om les mee te geven en voor demonstraties. Nieuwe studenten kunnen zo op een speelse en duidelijke manier geïnteresseerd worden in het discipline computer engineering.

Achtergrond

Binnen de moderne samenleving zijn computers volledig geïntegreerd. Bijna iedereen heeft een telefoon bij zich, veel mensen doen hun werk op een werkcomputer en elektronische apparaten worden steeds vaker 'slimmer' door de integratie van computers. Het lijkt bijna alsof de technologie al heel oud is en volledig ontwikkeld.

Echter het tegenovergestelde is waar. In 1944 werd met een lengte van ongeveer 15,5 meter (51 feet) en een hoogte van bijna 2,5 meter (8 feet) de eerst elektronisch programmeerbare computer gemaakt, de Harvard Mark I. Deze computer had een gewicht van 5 ton en simpele berekeningen namen enkele seconden tijd. Toch was deze computer revolutionair doordat hij verschillende programma's kon draaien zonder dat handmatig andere verbindingen gemaakt moesten worden, zoals bij de ENIAC. (Lasewicz, 2003) (Swaine & Freiberger, 2014)

De technologie gaat ook steeds harder. 40 jaar na de Harvard Mark I, kwam in 1985 de Nintendo Entertainment System uit, afgekort de NES. Deze spelcomputer was een 8-bits systeem met 2KB aan RAM. Populaire spellen zoals "Super Mario Bros" namen maar 40 KB aan geheugen in. 40 jaar na de NES zijn 32- en 64-bits systemen de regel en spellen nemen als snel gigabytes aan geheugen in, en hebben een fotorealistische kwaliteit. (McAnlis, 2015)

Anno 2018 zijn alle moderne werkcomputers veelal 64-bits systemen. Wanneer op elektronica webwinkels (mediamarkt, coolblue, etc.) gekeken wordt, is te zien dat alles veel krachtiger is dan ooit te voren, 4GB aan RAM is de standaard, hoewel 8GB ook zeer veel voorkomt. Harde schijven hebben een ongekend volume van tot wel 1TB. Kloksnelheden liggen in de gigahertz, wat betekend dat de klokpulsen sneller dan een nanoseconden elkaar opvolgen. Daar kon vroeger alleen maar gedroomd van worden. Het eind van de ontwikkelingen is nog niet in zicht en blijft zich verder ontwikkelen.

Doel

Met dit project is het de bedoeling om een computer te bouwen die eenvoudig te begrijpen is. De basis van de computer zal gebouwd worden met TTL-technologie. Een van de doelstellingen is dat de registers getallen van 16-bits groot kunnen vasthouden en bewerken. Een andere doelstelling is dat de computer gebruikt kan worden om uitleg te geven over hoe een computer werkt. De computer moet ook programmeerbaar zijn om het verschillende rekentaken uit te laten voeren. Doordat het project over een korte tijd plaats vind, zullen alleen de schema's ontworpen worden, een assembler gebouwd worden en een simulator gebouwd worden.

Ontwerpeisen

Een computer bestaat tegenwoordig uit veel complexe onderdelen, waar niet zomaar aan de binnenkant gekeken kan worden hoe ze werken of waar ze mee bezig zijn. Daarnaast werken de computers enorm snel met klokpulsen die korter dan een nanoseconde duren. Om beter

te kunnen uitleggen hoe een computer in de basis werkt, zal dus visueel gemaakt moeten worden wat de computer aan het doen is.

Daarnaast moet niet alleen visueel worden wat de computer doet, ook moet de snelheid (de klok) van de computer ingesteld kunnen worden, of zelfs handmatig bediend kunnen worden. Dit maakt het mogelijk om de stappen die de computer maakt op de voet te kunnen volgen. Hierdoor kunnen studenten ook de tijd nemen om als oefening te voorspellen wat de computer de volgende klokpuls gaat doen.

Een computer wekt door zijn complexiteit vaak ook een onbegrip op over hoe simpel de basis is de computer is. Daarbij komt de gedachte dat het bijna onmogelijk is om een computer te begrijpen. Om dit onbegrip uit de weg te halen zal ook het ontwerp van de computer vereenvoudigd worden. Hierdoor is makkelijk te zien wat de functionaliteit van elk onderdeel is, hoe de onderdelen op zich werken en hoe ze onderling samen werken.

Hierdoor kan de computer ook op demonstraties gebruikt worden. Het moet relatief eenvoudig zijn om computer te programmeren waardoor het interessant wordt om te zien hoe dit gebeurt en het gedrag van de computer veranderd. De functies, taken en toepassingen worden duidelijk inzichtelijk gemaakt

Om kort samenvatten zijn dit de doelen:

- De computer is visueel, er kan op elk moment gezien worden welke data een onderdeel vast houdt.
- De computer moet stap voor stap kunnen werken, en de snelheid moet instelbaar zijn.
- De computer heeft een eenvoudige infrastructuur die snel en makkelijk te begrijpen is.
- De computer moet simpel te programmeren zijn

Hardware architectuur

Om de eenvoudigheid van de computer te waarborgen, zullen alleen de minimale onderdelen gebruikt worden. Elk onderdeel moet eenvoudig in elkaar zitten, en makkelijk na te bouwen zijn.

Registers

De registers slaan tijdelijke waarden op. Elk register heeft ook zijn specifieke doel. Tegenwoordig bezitten computers vaak veel meer registers, maar om de complexiteit laag te houden, zijn er minimaal registers aanwezig. De aanwezige registers zijn:

- 2 rekenkundige registers, op de waarden binnen deze registers zullen de bewerkingen uitgevoerd worden.
- Uitvoer-register, hierin kan uitvoer van de computer gezet worden, bijvoorbeeld waardes die voor de gebruiker belangrijk zijn.
- Geheugenadres-register, deze kan een geheugenadres opslaan voor later. Dit wordt gebruikt om een subroutine aan te roepen.
- Instructie-register, hierin wordt de volgende instructie die de computer moet uitvoeren in opgeslagen zodat de computer er mee kan werken.

- Programmateller, deze houdt bij waar de volgende instructie zich in het geheugen bevind.
- Vlag-register, om eventuele statussen bij te houden is er een vlag register. Die statussen kunnen dat gebruikt worden om beslissingen mee te nemen.

Aritmische en logische eenheid

De ALU (Arithmetic and Logic Unit) is het gedeelde van de computer waar alle bewerkingen in plaats vinden. De ALU kan maar 8 simpele bewerkingen uitvoeren:

- Vergelijken, de twee rekenkundige registers worden vergeleken of register A nul is, of register A gelijk is aan register B en of register A groter is dan register B.
- Verschuiven naar links, alle bits worden 1 positie naar links verschoven en de nieuwe bit wordt een 0. Dit kan vergeleken worden als met vermenigvuldigen met 2.
- Verschuiven naar rechts, alle bits worden 1 positie naar rechts verschoven. De nieuwe bit word een 0, of als de gesigneerde modus geactiveerd is wordt de meest significante bit gekopieerd naar de nieuwe plek (verschil tussen logische en aritmetische verschuiving).
- Logische of, als een van de twee bits op een bepaalde positie hoog is, zal de uitkomst op die positie hoog zijn
- Logische en, als beide bits op een bepaalde positie hoog zijn, zal de uitkomst hoog zijn, en anders laag
- Logische niet, alle bits van register A worden omgedraaid zodat laag hoog wordt en hoog laag.
- Aftrekken, register B wordt van register A afgetrokken volgens het 2's compliment.
- Optellen, register A en B worden bij elkaar opgeteld.

De uitkomst van de berekening word in register A opgeslagen en de vlaggen worden tijdens elke berekening gezet. Wanneer de bit die eruit geschoven hoog is, zal de overdracht-vlag ook hoog zijn naast de vergelijkingen van de vergelijk functie. De overdracht-vlag kan ook hoog worden bij optellen en aftrekken als het resultaat groter is dan in 16-bits kan worden opgeslagen.

Geheugen

Het geheugen van de computer is een module dat veel waardes kan opslaan. Echter kan er maar 1 waarde tegelijkertijd geschreven of gelezen worden. Toch is het geheugen erg handig om de instructies voor de computer in op te slaan. De computer gebruikt een adresbus van 10-bit breed om de locaties in het geheugen aan te wijzen. Elke geheugenplek bevat 16 bits die naar de databus worden geschreven. Zo is er effectief 1024 adressen met elk 16 bits aan opslag wat tot 128 megabytes aan geheugen leidt.

Aanstuureenheid

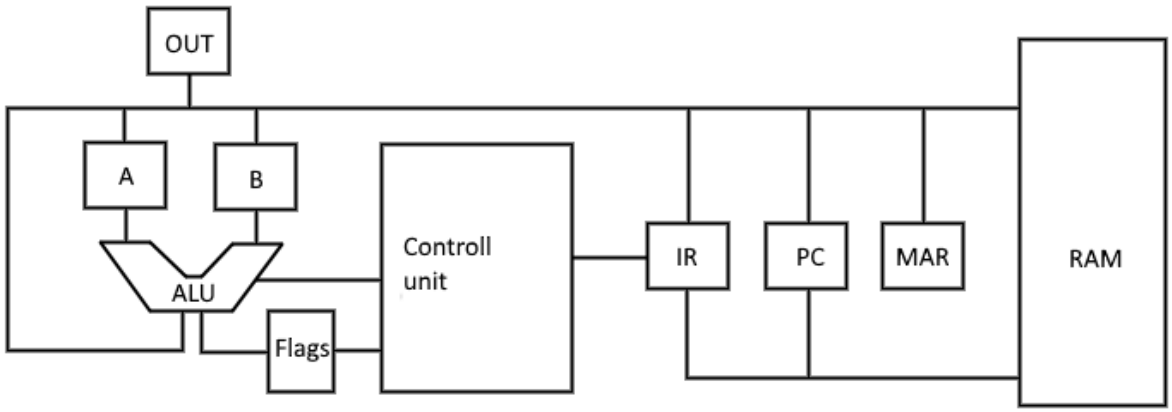
De aanstuureenheid is een "fixed state machine". Dit houd in dat er een beperkt aantal toestanden zijn die de aanstuureenheid kan hebben. De eenheid werkt elke instructie in 3 stappen af, de ophaal-, voorbereid-, en uitvoer-fase. Tijdens de ophaalfase wordt de volgende instructie naar het instructieregister geladen vanuit het geheugen. In de voorbereidfase wordt de programmateller met 1 verhoogd. En de uitvoerfase is het interessantst, hier vind alle uitvoering van de instructie plaats. En dan blijft het bij één van twee mogelijke uitvoeringen, of de computer kopieert bits naar een andere plaats, of de ALU veranderd de waarde in register A.

Overzicht

Om een schematisch overzicht te krijgen van hoe alle registers en andere onderdelen met elkaar verbonden zijn, is er een overzicht gemaakt. In het overzicht is ook goed te zien dat er 2 hoofdbussen zijn. De databus, die alle data vervoerd, en de adresbus, die de juiste geheugenlocatie bij het

geheugen selecteert. Alle onderdelen zijn met afkortingen weergegeven. De betekenissen van de afkortingen zijn:

- A, register A
- B, register B
- OUT, uitvoer-register
- Flags, vlagregister
- Controll unit, aanstuureenheid
- IR, instructieregister
- PC, programmateller
- MAR, geheugenadres-register
- RAM, geheugen



Assembly overzicht

De assembly waarin geprogrammeerd word is eenvoudig van opzet en ligt dicht tegen de hardware aan. Registers worden direct aangewezen en bestaan geen variabelen. De enige vorm van verlichting is dat geheugenadressen om data op te slaan relatief zijn, dus '0' is de eerste geheugenlocatie na alle instructies en '1' is de tweede en zo door. Hierdoor hoeft er niet nagedacht te worden over de absolute locaties of over hoeveel geheugen het programma zelf inneemt. Een andere laag van abstractie zijn labels. Wanneer er een sprong gemaakt moet worden of een subroutine aangeroepen word, kan naar het label verwezen worden voor waar de volgende code staat in plaats van het fysieke adres van de volgende instructie.

Een regels bestaat uit een standaard opbouw:

Label: <whitespace> **commando** *<whitespace> p1, p2* <whitespace> #opmerkingen

Legenda:
Dikgedrukt is verplicht.
Onderstreept is optioneel
Schuingedrukt is bij sommige commando's verplicht, bij andere niet

De commando's met bijbehorende parameters vind u in de volgende tabel:

Commando	**Betekenis**
NOOP	Geen operatie

CALL	Roept subroutine aan
RTN	Keert terug na subroutine
CLR p1	Leegt register aangegeven in p1
MOV p1, p2	Verplaatst de data van p1 naar p2
JMP c, label	Springt onder conditie c naar het aangegeven label. Wanneer altijd de sprong genomen moet worden moet c 'null' zijn. Anders moet c de letter 'C' voor de overdraagvlag, 'Z' voor nul-vlag, 'E' voor gelijk-vlag of 'G' voor groter dan vlag. Een combinatie van de letters 'C', 'Z', 'E' en 'G' is mogelijk als meerdere condities goed mogen zijn.
STR p1,loc	Slaat register aangegeven in p1 op in de relatieve locatie aangegeven in loc
LD p1, loc	Laad de waarde van de locatie aangegeven in loc naar het register aangegeven in p1
LDD p1, v	Laad een directe waarde in register p1. De waarde v mag maximaal 1023 zijn
TGSN	Veranderd de gesigneerde modes van aan naar uit of van uit naar aan
HALT	Stopt de computer
COMP	Vergelijkt de waarden in register A en register B
SFTL	Verschuif naar links
SFTR	Verschuif naar rechts
OR	Logische OR functie op register A en register B
AND	Logische AND functie op register A en register B
INV	Logische NOT functie op register A
SUB	Trekt register B van register A af
ADD	Telt register A en register B op

De registers die in de parameters genoemd kunnen worden zijn register A met $a, register B met $b en het uitvoerregister met $o.

Machinetaal overzicht

De instructies in voor de computer zijn opgebouwd uit 2 delen. Het eerste deel bestaat uit 6 bits die de instructie vormen. Vervolgens komen 10 bits die extra informatie kunnen bevatten zoals de geheugenlocatie waarnaartoe gesprongen moet worden of de waarde die direct in een register geladen moet worden, hierdoor is ook de begrenzing van maximaal 1023 om een directe waarde te laden naar een register en 1024 geheugenlocaties in het geheugen.

Code	**Betekenis**
000000	Geen operatie
00001X	Verplaats, X=0 verplaats $B naar $A, X=1 verplaats $A naar $B
0001RR	Leegt het register aangegeven met RR
001WRR	Laad of slaat gegevens op van/naar register RR. Als W=0 wordt het register opgeslagen, als W=1 wordt data naar het register schreven.
01CZEG	Sprong naar nieuwe locatie. Sprong wordt genomen als C hoog is en de overdracht vlag, of als Z hoog is en de nul-vlag, of als E hoog is en de gelijk-vlag of als de G hoog is en de groter dan-vlag is hoog. Als C, Z, E en G allemaal laag zijn wordt de sprong altijd genomen
10001X	Roep subroutine aan of keert terug. Als X=0 dan wordt er terug gekeerd van een subroutine, als X=1 wordt er een subroutine aangeroepen
1001XX	Laad direct een waarde in het register RR
101XXX	Voert een ALU-functie uit. De functie XXX kan in de tabel hieronder gevonden worden

11001X	Verplaatst register naar de uitvoer. Als X = 0 wordt register A verplaats, als X=1 wordt register B verplaatst
110001	Wisselt van gesigneerde modus
111111	Stopt de machine.

Register	**Code**
Register A	00
Register B	01
Uitvoerregister	10

Functie	**Code**
Vergelijk	000
Verschuif naar links	001
Verschuif naar rechts	010
Logische OR	011
Logische AND	100
Logische NOT	101
Aftrekken	110
Optellen	111

Assembler

Het doel van de assembler is dat tekst die goed leesbaar is voor de mens, wordt omgezet naar numerieke waarden waar de computer goed mee kan werken. Om de assembler makkelijker te laten werken is er een batch bestand naast de applicatie aanwezig. Wanneer het pad dat naar de assembler en batch bestand leidt wordt toegevoegd aan de “PATH” variabele van een machine, is het mogelijk om via de command line interface bestanden te compilen. Hiervoor zorgt met dat de map waar de CLI zich bevind gelijk is aan waar het bestand staat. Vervolgens wordt het commando: “assemble16 [source] [destination]” uitgevoerd. Hierbij is de “[source]” het bestand met de assembly-code en “[destination]” het bestand dat aangemaakt moet worden waar de binaire waarden naartoe geschreven worden.

De werking van de assembler is dat de eerst er een bestandsobject gemaakt wordt. Hierin zijn alle lijnen aanwezig en kan elke lijn aangeroepen worden.

De tweede stap is dat elke lijn in het bestand gesplitst wordt in het label, commando en parameter 1 en 2 wanneer aanwezig.

De derde stap is dat van elke lijn gekeken wordt of er een label aanwezig is. Zo ja, wordt deze opgeslagen inclusief het corresponderende machinetaal lijnnummer. Wanneer dit gedaan is, is ook bekend hoeveel instructies het programma heeft.

Voor de vierde stap is dit handig om te weten. Tijdens deze stap worden alle lijnen gecompileerd naar machinetaal. Eventuele regelnummers van labels, waarden of geheugenlocaties om naar op te slaan worden omgezet naar een binaire lijn en achter de opcode aangeplakt.

Als laatst worden de lijnen naar het uiteindelijke bestand geschreven. Elke instructie krijgt zijn eigen lijn. Hierdoor zijn de individuele instructies makkelijk van elkaar te onderscheiden en kunnen eventuele fouten (bijvoorbeeld als ergens een bit mist) snel gevonden worden en opgelost worden.

Simulator

De simulator is gebouwd om ervoor te zorgen dat de code die in assembly getypt is en vervolgens gecompileerd met de assembler ook goed werkt. Daarnaast is het een voorbeeld dat dan meerdere mensen tegelijkertijd hun code kunnen proberen en de code is sneller in de simulator geladen, dan in de fysieke computer.

De simulator bestaat uit twee delen. De eerste is de userinterface. Dit gedeelde van de code maakt alle tekstvelden en andere visuele onderdelen aan. Het tweede gedeelte is de feitelijke simulator. De code hiervoor is zo geschreven dat het zo dicht mogelijk bij de hardware ligt, zo wordt aftrekken alleen met bitsgewijze operatoren uitgevoerd. Aan elk onderdeel van de simulator worden de visuele tekstvelden en knoppen gegeven. Hierdoor is het mogelijk dat als ergens een waarde veranderd, de bijbehorende tekst ook veranderd. Ook eventuele acties worden direct simulator opgevangen.

Het grafische gedeelte van de simulator is zo ontworpen dat direct visueel zichtbaar is welke onderdelen met elkaar verbonden zijn. Ook is geprobeerd de uiteindelijke layout van de fysieke computer na te bootsen. Alle knoppen hebben een vrij duidelijke naam die hun functie vrij direct weergeven. Een programma kan met behulp van de “Load program” knop in het geheugen geladen worden. Wanneer deze knop wordt ingedrukt, komt er een scherm waar het juiste bestand geselecteerd kan worden. De reset-knop is zet alles op nul, behalve de data die het programma in het geheugen bevat. Eventuele gegevens die het programma had opgeslagen, zullen verloren gaan.

Omdat niet iedereen zeer goed is in binair getallen lezen, is het mogelijk om alle getallen in decimaal weer te geven. Echter wanneer iemand beter de mechaniek achter de simulator wil doorgronden, kan het handiger zijn om in binair de waardes te bekijken. Omdat van te voren niet duidelijk is wel niveau de gebruiker heeft, en wat de functie van het gebruik van de simulator is, kunnen alle waarden in decimaal, hexadecimaal en binair weergegeven worden.

Geciteerde werken

Lasewicz, P. (2003, 01 23). *IBM*. Opgehaald van IBM's ASCC introduction: https://www-03.ibm.com/ibm/history/exhibits/markI/markI_intro.html

McAnlis, C. (2015, 12 5). *Where do all the bytes come from*. Opgehaald van FreeCodeCamp: https://medium.freecodecamp.org/where-do-all-the-bytes-come-from-f51586690fd0

Swaine, M. R., & Freiberger, P. A. (2014, 12 18). *Harvard Mark I*. Opgehaald van Encyclopedia britannica: https://www.britannica.com/technology/Harvard-Mark-I