

Mic-1 IDE

- Simulatie van de Mic-1 architectuur (IJVM) -

Gebruikershandleiding

Inleiding

MIC1-Developer is een IDE (Integrated Development Environment) gebaseerd op de MIC1 processor simulatie van Ray Ontko, geïmplementeerd in Java (<http://www.ontko.com/mic1/>). MIC1-Developer maakt een scheiding tussen Ontko's assembler en zijn MIC1 simulator en maakt beide bereikbaar door middel van een grafische user interface.

MIC-1 Developer is als project gebouwd door studenten van de afdeling ICIM van de Hogeschool van Utrecht (Fac. Natuur en Techniek) (voor nadere informatie zie <http://helsloot.dhs.org/mic1/>).

De software is geheel in Java (JDK v1.3) geschreven en zou dus op elke JVM moeten kunnen werken, hoewel het aan te bevelen is de JDK of JRE versie 1.3 te gebruiken.

De broncode voor alle programma's is in de distributie bijgevoegd, zodat het mogelijk is dat gebruikers andere microarchitecturen, assemblers en microassemblers implementeren.

Deze handleiding bevat instructies voor de installatie en het gebruik van de IDE voor de mic1 microarchitectuur simulator. Het is de bedoeling dat deze IDE gebruikt wordt in een practicum bij Andrew A. Tanenbaum's Gestructureerde Computer Organisatie, 4^e editie, in het bijzonder "hoofdstuk 4: Het Microarchitectuur Niveau."

Bestanden

De mic1 distributie bevat een aantal bestand types waarmee wellicht niet iedereen bekend is. Hierna volgt een uitleg van deze bestandstypes.

.bat	Een batch bestand voor Windows 95/98/ME/XP/2000/NT. Dit is een script dat gebruikt wordt om programma's op runnen onder Microsoft Windows operating systems.
.conf	Een configuratie bestand. Dit is een tekst bestand dat configuratie informatie bevat voor het IDE programma.
.ijvm	Een Integer Java Virtual Machine taal bestand. Dit is een binair bestand dat IJVM object code bevat.
.jas	Een Java assembleertaal bestand. Dit is een tekst bestand dat IJVM broncode bevat.
.java	Een Java broncode bestand
.class	Een gecompileerd Java klassenbestand
.mic1	Een microprogramma voor de Mic1 architectuur. Dit is een binair bestand dat object code bevat voor een Mic1 microprogramma

De distributie bevat de volgende bestanden:

Java programma bestanden

De Java klassenbestanden in de classes directoryboom die gecreëerd wordt bij de installatie van de Mic1 IDE.

Hulpbestanden

ijvm.conf Een hulpbestand voor de ijvmasm assembler. Bevat een beschrijving van de assembly taal die de opcode, de mnemonic en de operand types van elke instructie geeft.

Voorbeeld programma's

ijvm.jas	Een programma dat alle mogelijkheden van de mic1 architectuur test.
demo.jas	Print een groeiende rij "=" karakters op de console.
demo2.jas	Print "hello world" op de console.
echo.jas	Een programma dat van het keyboard leest en een ingetoetst karakter op de console weergeeft.
add.jas	Een programma dat twee hexadecimale getallen van het toetsenbord leest en de som op de console weergeeft.
tabel.jas	Een programma dat alle printbare karakters uit de ASCII tabel op de console print.

Documentatie

Readme.txt	Een tekstbestand met distributie en installatiegegevens.
mal.html	Een overzicht van de Microprogramma assembleertaal.
jas.html	Een overzicht van de Java assembleertaal
faq.html	Veel gestelde vragen

Java bronbestanden

De Java broncode van alle java programma's bevindt zich in de source directory die bij de installatie van de IDE gecreëerd wordt. Een listing is in src_mic1.txt te vinden.

MIC1 Integrated Development Environment

De MIC1 IDE software bevat een grafische user interface voor een assembler voor de Integer Java Virtual Machine (IJVM), een geheugen monitor (algemeen- en stackgeheugen) en mic1sim, een simulator voor de Mic1 architectuur met daarbij de grafische weergave van deze architectuur. Studenten kunnen deze IDE en de bijgeleverde voorbeeldprogramma's gebruiken om:

1. De stap-voor-stap interpretatie van een programma in assembleertaal door een microprogramma te bekijken.
2. assembleertaal programma's in IJVM te schrijven en te testen.

Installatie

De mic1 software heeft een Java installatie nodig om te kunnen werken. Dat kan de JDK zijn maar ook een runtime environment (JRE) werkt. Wel is het aan te bevelen om tenminste de versie 1.3 te gebruiken, daar de laatste versie van MIC1-Developer in JDK v1.3 gebouwd en gecompileerd is. Ga naar www.java.sun.com om de nieuwste JDK of JRE op te halen. Bij de volgende instructies wordt er van uit gegaan dat de JDK is geïnstalleerd.

1. Download de mic1 software

Deze is in ZIP formaat te vinden op helsloot.dhs.org/mic1/. Vervolgens moeten de files uit de gecomprimeerde distributie file geïnstalleerd worden:

a Unix of Linux installatie:

- Maak eerst een directory aan voor de IDE software:

```
$ mkdir mic1
```

- Kopieer dan de mic1.zip file naar deze mic1 directory. En pak de bestanden uit met:

```
$ unzip mic1.zip
```

- Klaar! Nu moet er nog voor gezorgd worden dat de software de Java installatie ook nog vinden kan. Om te zien of de Java installatie in het zoekpad staat kun je typen:

```
$ echo $PATH
```

In het antwoord moet dan onder meer iets staan als `/usr/java/jdk1.3.1/bin`. Als dit niet het geval is moet de PATH variabele aangepast worden. Zoek eerst uit wat de directory is waar de Java installatie te vinden is, en pas vervolgens de PATH variabele aan:

```
$ find / -name "jdk" -print (dit kan lang duren, het hele systeem wordt doorzocht)
/usr/java/jdk1.3.1
$ export PATH=/usr/java/jdk1.3.1/bin:$PATH
```

Om ervoor te zorgen dat de PATH variabele bij het inloggen of bij het opstarten van het systeem meteen al goed gezet wordt kun je dit ook aanpassen in respectievelijk je profile bestand (`.bash_profile` bij Linux bash-shell) of (als root) in `/etc/profile`.

- Het programma kan nu gestart worden door:

```
$ java Main
```

b Windows Installatie (95,98,ME,XP,2000,NT4):

- Maak een nieuwe folder met de naam "mic1". Kopieer de mic1.zip file naar de mic1 folder en gebruik een programma als WinZip om de bestanden uit te pakken. Alle programma bestanden, bronbestanden en documentatie zullen uitgepakt worden.
- Voordat de mic1 software gebruikt kan worden moet het bestand "Start.bat", dat te vinden is in de "Classes" folder, aangepast worden. Doe dit door Start.bat te openen in bv. Notepad.
 - Voeg "rem" toe aan het begin van de regels 2 tot 5 van dit bestand, zodat ze er als volgt uitzien:


```
rem echo NOTE: YOU NEED TO EDIT THE FILE START.BAT BEFORE YOUR mic1 SOFTWARE will
rem echo WORK CORRECTLY.
rem pause
rem goto end
```
 - Zorg er vervolgens voor dat op regel 6 het juiste pad naar de bin directory van de Java installatie gezet wordt (NB. Let erop dat achter het path statement `";%path%"` toegevoegd moet worden om ook het oorspronkelijke pad te handhaven!)
 Als het pad naar de Java installatie niet bekend is, kies dan in het Start menu Find→Files or Folders. Zoek naar javac.exe. De directory die aangegeven wordt in de InFolder kolom is de directory die aan het path statement toegevoegd moet worden.
- Om de mic1 software te starten kan nu Start.bat aangeklikt worden. Deze dit batch bestand zal dan de PATH variabele aanpassen en het programma starten.

2. Testen van de installatie

a Unix of Linux

Ga naar de directory classes en typ:

```
$ java Main
```

om de IDE te starten

b Windows

Dubbelklik op Start.bat om de IDE te starten

c Beide

Kies in het File menu voor Open, ga naar de Examples directory in de mic1 directory en selecteer ijvmttest.jas om als bytecode programma te laden in de assembler. Kies in het Assembler menu voor Assemble, en kies onder het View menu voor Console om een output console op het scherm te krijgen.

Klik nu de Run button aan om de interpretatie van het macroprogramma door het microprogramma te starten. Na een lange tijd (alle mogelijke instructies en programma constructies worden getest) runnen van de simulator zou het volgende op de console moeten verschijnen: OK.

Indien dit gebeurt (geduld oefenen!) is alles in orde.

MIC1 Registerset

De MIC1 processor bevat de volgende registers:

MAR	MIC1 Adres Register, bevat het adres van de op te halen of weg te schrijven data.
MDR	MIC1 Data Register, bevat het woord dat weggeschreven of opgehaald wordt uit het datageheugen.
PC	Program Counter, bevat het adres van de op te halen instructie.
MBR	MIC1 Bytecode Register, bevat de byte die uit het instructiegeheugen opgehaald is.
MBRU	MIC1 Bytecode Register Unsigned, bedoeld om met databytes om te kunnen gaan die op de opcode volgen.
SP	Stack Pointer, wijst naar de top van de (operand) stack.
LV	Local Variable Pointer, wijst naar het begin van de locale variabelen op de (variabelen)stack.
CPP	Constant Pool Pointer, geeft het begin van de constanten pool aan.
TOS	Top Of Stack, bevat altijd de inhoud van de top van de (operand)stack.
OPC	Old Program Counter, eigenlijk een soort kladregister.
H	Hulpregister, nodig vanwege het MIC1-ontwerp met twee bussen.
MIR1	Micro Instruction Register 1, geeft de uitgevoerde microinstructie weer.
MIR2	Micro Instruction Register 2, geeft de uit te voeren microinstructie weer.
MPC	Micro Program Counter, bevat het adres van de uit te voeren microinstructie

Vlaggen:

Z	1 als het resultaat van een ALU-bewerking nul is.
N	1 als het resultaat van een ALU-bewerking negatief is.

IJVM Assembleertaal Specificatie

Beschrijving van de syntax van de IJVM Assembleertaal zoals die verwacht wordt door de MIC1 IJVM assembler.

1. IJVM Instructieset

Mnemonic	Operanden	Beschrijving
BIPUSH	Byte	Zet een byte op de stack
DUP	N/A	Lees het bovenste woord van de stack en zet het op de stack
ERR	N/A	Print een error bericht en stop de simulator
GOTO	Label naam	Onvoorwaardelijke sprong
HALT	N/A	Stop de simulator
IADD	N/A	Pop twee woorden van de stack; zet hun som op de stack
IAND	N/A	Pop twee woorden van de stack; zet Boolean AND op de stack
IFEQ	Label naam	Haal een woord van de stack en ga verder bij label als het nul is
IFLT	Label naam	Haal een woord van de stack en ga verder bij label als het minder dan nul is
IF_ICMPEQ	Label naam	Haal twee woorden van de stack en ga verder bij label als ze gelijk zijn
IINC	Variabele naam, byte	Tel een constante waarde op bij een lokale variabele
ILOAD	Variabele naam	Zet lokale variabele op de stack
IN	N/A	Leest een karakter uit de keyboard buffer en zet de ASCII waarde daarvan op de stack. Als er geen karakter beschikbaar is wordt er 0 op de stack gezet
INVOKEVIRTUAL	Methode naam	Roep een methode aan
IOR	N/A	Haal twee woorden van de stack; zet Boolean OR op de stack
IRETURN	N/A	Terug uit methode met een integer waarde op de stack
ISTORE	Variabele naam	Haal een woord van de stack and sla het op in een lokale variabele
ISUB	N/A	Haal twee woorden van de stack; zet hun verschil op de stack
LDC_W	Constante naam	Zet constante uit de constante pool op de stack
NOP	N/A	Doe niets
OUT	N/A	Haal een woord van de stack and print het naar standard out (console)
POP	N/A	Verwijder een woord van de top van de stack en plaats het in het MDR
SWAP	N/A	Verwissel de twee bovenste woorden van de stack
WIDE	N/A	Prefix instructie; de volgende instructie heeft een 16-bit index

- **byte:** Een numerieke waarde, in octaal (032 – een nul ervoor), decimaal (26 – niets ervoor), of hexadecimaal (0x1A – nul-x ervoor) formaat. Ook karakter waardes zijn toegestaan (‘M – een enkele quote ervoor). Wordt naar een 1-byte constante vertaald.
- **label naam:** De string naam van een label, wordt vertaald naar een 2-byte offset (t.o.v. de PC)
- **variabele naam:** De string naam van een lokale variabele, wordt vertaald naar een 1-byte waarde die de offset aangeeft in het lokale variabelen frame
- **methode naam:** De string naam van een methode. Bij de vertaling wordt het adres van de methode berekend en in de constanten pool gezet. Deze operand wordt dan vervangen door de 2-byte index (in de constantenpool) van dat adres.
- **Constante naam:** De string naam van een constante. Wordt vertaald naar een 2-byte index (in de constanten pool).
- **N/A:** Deze instructie gebruikt geen operands.

2. Constanten

Globale constanten worden gedeclareerd in de `.constant` sectie aan het begin van het bronbestand. De waarde van de constante kan gedeclareerd worden als hexadecimaal (voorvoegsel "0x"), octaal (voorvoegsel "0") of als decimaal (geen voorvoegsel). De aldus gedeclareerde constanten kunnen dan bij hun naam benaderd worden door een instructie die een constante als parameter verwacht (bv `LDC_W constante_naam`)

Syntax:

```
.constant
constante1      waarde1
constante2      waarde2
.end-constant
```

Voorbeeld:

```
// Dit programma drukt alle printbare ASCII karakters (waarde 32 - 126)
// af op de console
.constant
een      1
start    32
stop     126
.end-constant

.main
      LDC_W start
next:  DUP
      OUT
      LDC_W stop
      ISUB
      IFEQ done
      LDC_W one
      IADD
      GOTO next
done:  POP
      HALT
.end-main
```

De waardes van de constanten worden in de constanten pool gezet, naar dit deel van het geheugen wordt verwezen door de CPP. Behalve de door de gebruiker gedefinieerde constanten worden door de assembler ook de absolute adressen van de methoden in de constanten pool gezet. Zie de paragraaf over methoden voor nadere informatie hierover.

3. Lokale Variabelen

Lokale variabelen worden binnen een `.main` of `.method` sectie gedeclareerd en kunnen uitsluitend door die `main` of `method` benaderd worden. Er bestaan geen globale variabelen. Lokale variabelen kunnen via een naam benaderd worden (bv `ISTORE var_name`).

Syntax:

```
// binnen een .main of .method declaratie:  
  
.var  
var1  
var2  
.end-var
```

4. Labels

Labels worden gebruikt om naar programmaregels te wijzen ten behoeve van `jump` of `goto` instructies. Labels zijn alleen maar bereikbaar vanuit de methode waarin ze gedeclareerd zijn. Met andere woorden, sprongen kunnen alleen voorkomen binnen een methode, en dus niet van de ene naar de ander methode.

Labels bestaan uit een willekeurig woord afgesloten met een dubbele punt. Ze worden gedeclareerd aan het begin van de regel en slaan op de daarachter volgende instructie. Hieronder zie je een programma fragment om een en ander te illustreren:

```
.  
.  
.  
ILOAD total  
LDC_W max_value  
ISUB  
IFLT lt_max      // If total < max_value, goto lt_max  
GOTO gte_max      // else (total >= max_value), goto gte_max  
lt_max: HALT  
gte_max:           // Labels kunnen op een aparte regel staan  
                  // vóór de instructie waarop ze slaan  
ERR
```

5. Hoofdprogramma (Main)

Elk JVM programma moet een `.main` methode bevatten. `.main` moet gedeclareerd worden vóór elke andere methode.

Syntax:

```
.main  
  
-- variabelen declaratie --  
  
-- programma --  
  
.end-main
```


6. Methodes

Als een methode gedeclareerd wordt wordt zijn beginadres toegevoegd aan de globale constanten daarna kan de methode met behulp van zijn naam aangeroepen worden (bv `INVOKEVIRTUAL method_name`). De parameters van een methode worden gedeclareerd door middel van een tussen haakjes gezette en door komma's gescheiden lijst van de parameters. Deze parameters worden dan toegevoegd aan de lokale variabelen van de methode en kunnen dus alleen binnen de methode benaderd worden met behulp van hun naam (bv `LOAD param1`). Als de methode geen parameters heeft zet dan niets tussen de haakjes. Het aanroepen van een methode verdient bijzondere aandacht. Het Mic1 microprogramma vereist een specifiek protocol bij het aanroepen van een methode, waarvan de te volgen stappen hierna volgen:

- Push OBJREF. Dit is een verwijzing naar het object dat de methode aanroept. In JVM wordt OBJREF eigenlijk alleen maar gebruikt om een plaats op de stack te reserveren en wordt overschreven met het adres van de plaats waar de oude PC is opgeslagen. De oorspronkelijke waarde die door de aanroepende methode op de stack is gezet wordt verder niet gebruikt. Dus in JVM kan elke waarde voor OBJREF worden gebruikt, maar er moet een waarde worden opgegeven.
- Push over te dragen parameters. Parameters worden op de stack gezet in de volgorde waarin zij in de lijst van de methode declaratie staan.
- Roep de methode aan met `INVOKEVIRTUAL method_name`.
- De return waarde staat op de top van de stack. Als `IRETURN` in een methode aangeroepen wordt, wordt de waarde op de top van de stack van de huidige methode gekopieerd naar de top van de stack van de aanroepende methode, om op die manier als return waarde te dienen.

Zie de in de distributie meegeleverde FAQ voor een meer gedetailleerde uitleg met een compleet voorbeeld.

Syntax

```
.method method_name(param1, param2, ...)  
  
-- variabelen declaratie --  
  
-- methode statements --  
  
.end-method
```

7. Input en output

Input en output worden afgehandeld door de hoofdgeheugen module in de mic1 simulator. Het hoofdgeheugen behandelt de hoogste geheugenadressen als een I/O apparaat. Als in dit gebied een woord geschreven wordt schrijft het hoofdgeheugen de karaktercode van dit woord naar het standard out tekstgebied van de simulator. Als een woord vanuit dit gebied gelezen wordt zal het hoofdgeheugen de eerstvolgende beschikbare waarde uit een interne keyboardbuffer teruggeven, die de karaktercode bevat de door de gebruiker indrukte toets. Als deze buffer leeg is (geen toets ingedrukt) wordt een nulwaarde teruggegeven.

Het `OUT` commando haalt een woord van de stack en print het naar het standard out tekstgebied.

```
BIPUSH      0x41  // Zet "A" op de stack  
OUT          // Print "A" op standard out
```

Het `IN` commando leest een karakter uit de keyboard buffer en zet deze waarde op de stack. Als er geen toets beschikbaar is om uit te lezen (geen toets ingedrukt of alle toetsen zijn al uitgelezen), zal een nulwaarde op de stack gezet worden. Daarom is het verstandig om in een programma het `IN` commando in een lus in te bouwen die doorloopt totdat `IN` een geldige waarde teruggeeft.

```

Getch:   IN           // Lees een karakter uit keyboardbuffer
         DUP          // Verdubbel de waarde om te vergelijken
         IFEQ reread  // als karakter=0 (geen toets), ga naar reread
         GOTO done    // anders, ga naar done (karakter is een
                     // geldige toets)
reread:  POP          // haal ongeldig karakter van de stack af
         GOTO getch   // probeer opnieuw een karakter op te halen
done:    // karaktercode staat op de stack

```

8. Commentaar

In de JVM assembler wordt commentaar aangegeven op dezelfde wijze als bij C++ of als bij Java gebruikelijk is: // (een dubbele slash) geeft het begin van een commentaar regel aan. Alles wat hierna volgt is commentaar tot aan het einde van de regel.