

Einführung in die Rechnerarchitektur Praktikum

ANWENDERDOKUMENTATION

Projekt:

MI: Komplexe Zahlen

Projektleiter	Sandra Grujovic
Dokumentation	Ivan Chimeno
Formaler Vortrag	Martin Zinnecker

1 Einleitung

Dieses Mikroprogramm ermöglicht die Verarbeitung komplexer Zahlen in der mikroprogrammierbaren Maschine. Es beschränkt sich hierbei nur auf die Addition und Subtraktion zweier vorzeichenbehafteter Zahlen.

2 Beschreibung des Programms

Das Programm bietet die Möglichkeit an, komplexe Zahlen auf eine einfachere Art und Weise zu addieren und zu subtrahieren. Dabei steht eine Akkumulator-Architektur zur Verfügung, bestehend aus den Registern R6 und R7. In den Akkumulator kann eine 32-Bit Zahl geladen werden, welche dort Rechenoperationen durchführen kann. Es ist möglich, das Ergebnis wieder in dem Hauptspeicher zu speichern oder in dem Akkumulator zu behalten, um darauf weitere Operationen auszuführen.

Die folgende Tabelle beschreibt die Befehle, die vom Mikroprogramm unterstützt werden:

Opcode	Länge	Befehl	Beschreibung
01	16-Bit	cLoad [RA]	Lädt eine komplexe Zahl aus dem Speicher in den Akkumulator.
02	16-Bit	cStore [RB]	Kopiert den Wert des Akkumulators in den Speicher.
03	16-Bit	cAdd [RA]	Addiert die komplexe Zahl [RA] zum Akkumulator.
04	32-Bit	cSub [addr]	Subtrahiert die komplexe Zahl [addr] vom Akkumulator. <i>addr</i> ist eine Immediate.

Tabelle 1: Verfügbare Befehle

Tipp: Da alle Befehle Adressen annehmen, bedeutet [X] die Hauptspeicherzelle mit der Adresse X. Das heißt, dass bei [X], also im Register X, die gewünschte Hauptspeicherzellenadresse eingetragen werden muss. **cSub** verwendet als Parameter eine Immediate und verbraucht dementsprechend zwei 16-Bit lange Speicherzellen.

3 Verwendung

3.1 Installation von JMic

1. Um das Programm verwenden zu können, muss das Programm JMic IDE kostenlos [heruntergeladen werden](#).
2. JMic verwendet das Java Runtime Environment und steht [hier](#) kostenlos zur Verfügung.
3. Nach der Installation von Java kann die JMic.jar Datei direkt geöffnet und das Programm geladen werden.
4. Um das Programm zu laden, müssen man bei: „Datei,, → „Öffnen“ durchnavigieren und einen Doppelklick auf der *Mi.mpr* Datei durchführen.

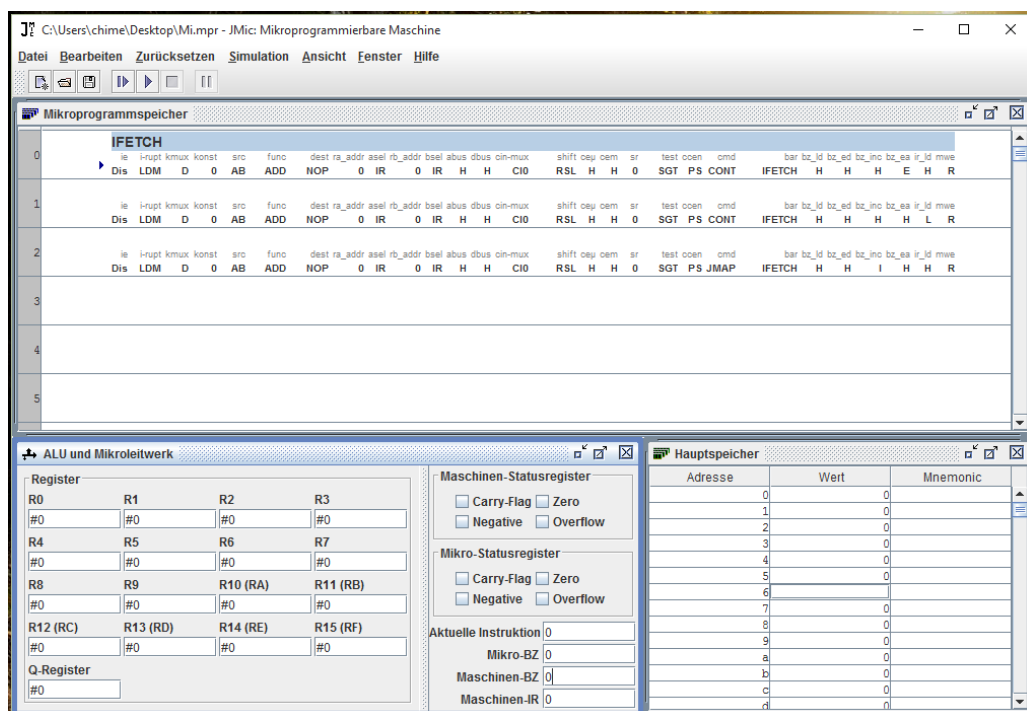


Abbildung 1: JMic IDE mit dem geladenen Programm

3.2 Benutzung der Befehle

Das JMic Programm besteht aus drei Fenstern: Mikroprogrammierspeicher, ALU/Mikroleitwerk und Hauptspeicher. Um die oben definierten Befehle ausführen zu können, muss der

Anwender im ALU/Mikroleitwerk Fenster, die gewünschten Parameter in die Register eintragen (siehe *Tabelle 1* für die genaueren Definitionen der Befehlen) und im Hauptspeicher den Befehlsaufruf speichern. Ein Beispiel für einen Befehlsaufruf wäre `0x0110`. `0x01` steht dabei für den Opcode des Befehls. `1` und `0` sind die Registernummern, die vom Befehl verwendet werden. In diesen Fall wird **cLoad** aufgerufen. **cLoad** nimmt, gemäß *Tabelle 1*, nur das Register RA als Argument an. RB, das zweite Argument, wird vom Programm ignoriert und kann einfach auf 0 gesetzt werden. Mit der F5 Taste kann ein Programm schrittweise ausgeführt werden.

3.3 Belegung der Register

Für den Anwender stehen normalerweise die Register 0 bis 7 zur Verfügung. Die Register 6 und 7 sind jedoch für den Akkumulator reserviert und dürfen nicht überschrieben werden! Eine Überschreibung der beiden Register kann das Verhalten des Programms drastisch ändern und zu falschen Ergebnissen führen! Empfohlen wird demnach die Benutzung der Register 0 bis 5.

3.4 cStore und cLoad Befehle durchführen

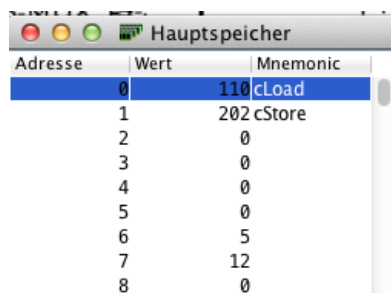
Mit den **cStore** und **cLoad** Befehlen kann eine Zahl in den Akkumulator geladen bzw. im Hauptspeicher gespeichert werden. Dies wird im folgenden Beispiel veranschaulicht.

Beispiel:

Die Werte 5 und 12 liegen bei den Adressen 0x6 und 0x7. Sie sollen, mithilfe des Akkumulators, in die Speicherzellen 0x9 und 0xA kopiert werden.

Das Programm:

Die Werte `0x0110` und `0x0202` werden an die Adressen 0x0 bzw. 0x1 im Hauptspeicher eingetragen.



Adresse	Wert	Mnemonic
0	110	cLoad
1	202	cStore
2	0	
3	0	
4	0	
5	0	
6	5	
7	12	
8	0	

Abbildung 2: Hauptspeicher mit gewünschten Werten und Befehlsaufrufen

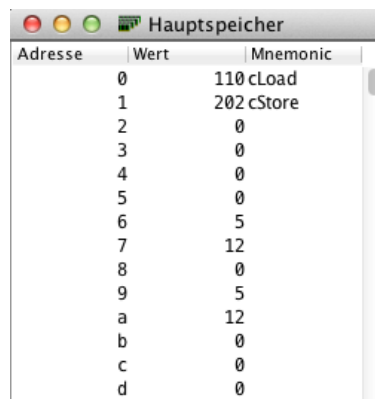
Die Start- und Zieladressen werden im ALU/Mikroleitwerk, in die Register R1 und R2, eingetragen:

Register			
R0	R1	R2	R3
#0	0x6	0x9	#0
R4	R5	R6	R7
#0	#0	#0	#0
R8	R9	R10 (RA)	R11 (RB)
#0	#0	#0	#0
R12 (RC)	R13 (RD)	R14 (RE)	R15 (RF)
#0	#0	#0	#0

Abbildung 3: ALU/Mikroleitwerk mit Start- und Zieladressen

Erklärung:

1. Im Hauptspeicher müssen in die Adressen 0x6 bzw. 0x7, die Werte 05 und 12, eingegeben werden. Diese sollen schließlich kopiert werden.
2. Die Zahlen können mithilfe des **cLoad** Befehls in den Akkumulator geladen werden. Da der Befehl mit 32-Bit Zahlen arbeitet, reicht es aus, im Register 1 die Adresse des Wertes 05 einzugeben d.h. 0x6.
3. Analog, um die Zahlen von dem Akkumulator in den Speicher kopieren zu können, muss im Register 2 die Zieladresse eingegeben werden d.h. 0x9. Der Inhalt des Registers wird als Argument für den **cStore** Befehl verwendet.
4. Die beiden Befehle können jetzt in dem Hauptspeicher aufgerufen werden. **cLoad** verwendet als Opcode die 0x01 und **cStore** die 0x02. Entsprechend müssen also im Hauptspeicher, bei den Adressen 0x0 und 0x1, die Werte 0110 bzw. 0202 eingetragen werden. Dabei steht die 1 für Register R1 und das 2 für Register R2.
5. Nach der Ausführung sollen die Werte 05 und 12 bei den Adressen 0x9 und 0xA, liegen.

Ausgabe:


Adresse	Wert	Mnemonic
0	110	cLoad
1	202	cStore
2	0	
3	0	
4	0	
5	0	
6	5	
7	12	
8	0	
9	5	
a	12	
b	0	
c	0	
d	0	

Abbildung 4: Die kopierten Werte stehen in 0x9 und 0xA

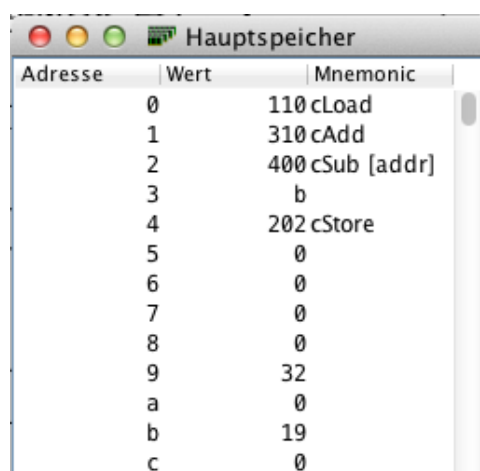
3.5 cAdd und cSub Befehle durchführen

Mit den **cAdd** und **cSub** Befehlen und dem Akkumulator, können Additions- und Subtraktionsoperationen zwischen zwei komplexen Zahlen durchgeführt werden. Das folgende Programm demonstriert die Verwendung der beiden Befehle.

Beispiel: Durchführung von $50 + 50 - 25$. Das Ergebnis soll bei der Adresse 0xD stehen. Die Zahlen 50 bzw. 25 liegen bei den Adressen 0x9 bzw. 0xB.

Programm:

Die Werte 0x0110, 0x0310, 0x0400, 0x0b und 0x0202 werden an die Adressen 0x0 bis 0x4 im Hauptspeicher eingetragen:



Adresse	Wert	Mnemonic
0	110	cLoad
1	310	cAdd
2	400	cSub [addr]
3	b	
4	202	cStore
5	0	
6	0	
7	0	
8	0	
9	32	
a	0	
b	19	
c	0	

Abbildung 5: Hauptspeicher mit gewünschten Werte und Befehlsaufrufen

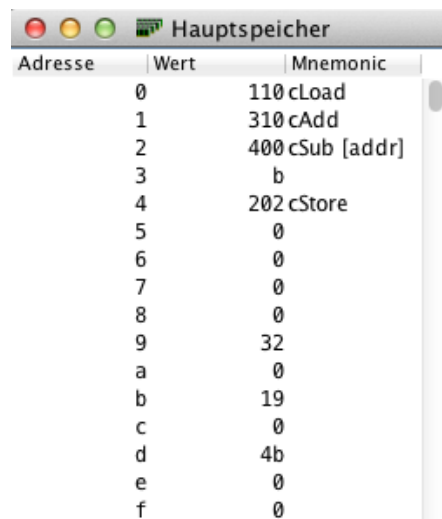
Die Start- und Zieladressen werden in dem ALU/Mikroleitwerk, in die Register R1 und R2, eingetragen:

Register			
R0	R1	R2	R3
#0	0x9	0xd	#0
R4	R5	R6	R7
#0	#0	#0	#0
R8	R9	R10 (RA)	R11 (RB)
#0	#0	#0	#0
R12 (RC)	R13 (RD)	R14 (RE)	R15 (RF)
#0	#0	#0	#0
Q-Register			
#0			

Abbildung 6: ALU/Mikroleitwerk mit Start- und Zieladressen

Erklärung:

1. Um die Operation $50 + 50 - 25$ durchführen zu können, muss diese in vier Schritte aufgeteilt werden: 50 muss in den Akkumulator geladen werden. Dann muss der Akkumulator mit 50 aufaddiert werden. Das Ergebnis wird von 25 subtrahiert und schließlich bei der Adresse *0xD* gespeichert.
2. Bei der Adresse *0x0* wird **cLoad** ausgeführt (**cLoad** und **cStore** wurden im *Abschnitt 3.4* detailliert durchgenommen) mit *0x9* als Startadresse (enthält den Wert 50 in Hexadezimalschreibweise) die im Register R1 enthalten ist. Der Akkumulator enthält jetzt den Wert 50.
3. Um den Akkumulator mit 50 aufzuaddieren, wird dieselbe Startadresse für den **cAdd** Befehl genommen. Er steht bei der Adresse *0x1*, mit dem Opcode 03. Der Akkumulator enthält jetzt den Wert 100.
4. Um 25 von dem Akkumulator zu subtrahieren wird der **cSub** Befehl (mit Opcode 04) bei der Adresse *0x2* verwendet. Da **cSub** eine Immediate annimmt, folgen nach dem Opcode zwei Nullen (RA und RB werden ignoriert). Der eigentliche Parameter steht in der nächsten Speicherzelle. Da die Zahl 25, bei der Adresse *0xB*, liegt, wird sie bei der Adresse *0x3* eingetragen.
5. Nach der Ausführung der Subtraktion wird das Ergebnis, mithilfe des **cStore** Befehls, vom Akkumulator, zu der Hauptspeicheradresse *0xD* kopiert.

Ausgabe:

Adresse	Wert	Mnemonic
0	110	cLoad
1	310	cAdd
2	400	cSub [addr]
3	b	
4	202	cStore
5	0	
6	0	
7	0	
8	0	
9	32	
a	0	
b	19	
c	0	
d	4b	
e	0	
f	0	

Abbildung 7: Ausgabe in Dezimalschreibweise

Die Ausgabe zeigt, dass 75 das Ergebnis von $50 + 50 - 25$ ist, und wie im Register R2 angegeben, bei den Adressen 0xD bzw. 0xE liegt.