

Einführung in die Rechnerarchitektur Praktikum

SPEZIFIKATION

Projekt:

MI: Komplexe Zahlen

Projektleiter	Sandra Grujovic
Dokumentation	Ivan Chimeno
Formaler Vortrag	Martin Zinnecker

1 Aufgabenkurzbeschreibung

Ziel dieses Projekts ist die Verarbeitung komplexer Zahlen in der mikroprogrammierbaren Maschine. Um dies zu ermöglichen sollen die Befehle `cLoad[RA]`, `cStore[RB]`, `cAdd[RA]` und `cSub[addr]` implementiert werden. Um das Projekt zu vereinfachen, ist eine Akkumulator-Architektur bereits vorhanden und besteht aus den Registern `r6` und `r7` und die erste zu verarbeitende komplexe Zahl steht im Akkumulator. Um komplexe Zahlen in der MI-Maschine darstellen und verarbeiten zu können, wird das sogenannte vorzeichenbehaftete (8.8) Fixpunktformat verwendet.

Zahl $z =$	a			$+ i *$	b		
	Vorkomma	.	Nachkomma		Vorkomma	.	Nachkomma
Bit	16				16		

Tabelle 1: Fixpunktformat für komplexe Zahlen

1.1 Befehlsspezifikation

MI-Befehl	Beschreibung
<code>cLoad [RA]</code>	Lädt eine komplexe Zahl aus dem Speicher in den Akkumulator.
<code>cStore [RB]</code>	Kopiert den Akkumulator in den Speicher.
<code>cAdd [RA]</code>	Addiert die komplexe Zahl <code>[RA]</code> zum Akkumulator.
<code>cSub [addr]</code>	Subtrahiert die komplexe Zahl <code>[addr]</code> vom Akkumulator.

Die Akkumulator-Architektur ist bereits vorhanden und besteht aus den Registern `r6` und `r7`.

2 Arbeitsaufträge

2.1 Komplexe Zahlen und die damit verbundenen Rechenregeln

Eine komplexe Zahl z ist eine Zahl, die durch zwei reelle Zahlen und eine imaginäre Zahl ausgedrückt werden kann und folgendes Format besitzt:

$$z = a + i * b \text{ mit } a, b \in \mathbb{R} \text{ und } i^2 = -1$$

Die Besonderheit der komplexen Zahl z ist die Lösbarkeit der Gleichung $x^2 + 1 = 0$ anhand ihrer imaginären Zahl. Anders ausgedrückt: die komplexe Zahl z besteht aus einem reellen Teil a und einem imaginären Teil b .

Man kann mit komplexen Zahlen rechnen. Für dieses Projekt sind nur Addition und Subtraktion zweier Zahlen von Relevanz.

Angenommen zwei komplexe Zahlen $z_1 = a_1 + i * b_1$ und $z_2 = a_2 + i * b_2$ sind definiert.

Die Addition ist also folgend definiert:

$$\begin{aligned} z_1 + z_2 &= a_1 + ib_1 + a_2 + ib_2 = a_1 + a_2 + ib_1 + ib_2 = (a_1 + a_2) + (ib_1 + ib_2) \\ &= (a_1 + a_2) + i(b_1 + b_2) \end{aligned}$$

Analog gilt für die Subtraktion:

$$\begin{aligned} z_1 - z_2 &= a_1 + ib_1 - (a_2 + ib_2) = a_1 + ib_1 - a_2 - ib_2 = (a_1 - a_2) + (ib_1 - ib_2) \\ &= (a_1 - a_2) + i(b_1 - b_2) \end{aligned}$$

Die folgenden Gesetze gelten:

$$z_1 + z_2 = z_2 + z_1 \text{ (Kommutativgesetz)}$$

$$z_1 + (z_2 + z_3) = (z_1 + z_2) + z_3 \text{ (Assoziativgesetz)}$$

2.2 Zahlenbereiche des vorgestellten Zahlenformats

Type	Range
signed	<i>von</i> (-128,99609375 - $i * 128,99609375$) <i>bis</i> (127,99609375 + $i * 127,99609375$)
unsigned	<i>von</i> 0 <i>bis</i> (255,99609375 + $i * 255,99609375$)

2.3 Vor- und Nachteile von Fixpunktformat im Vergleich zu Fließkommazahl

Vorteile Fixpunktformat	Vorteile Fließkommazahl
-Geringerer Rechenaufwand da Stellenwerte einzelner Bits immer gleich	- größer Wertebereich als Fixpunktformat mit selber Bit länge
-gesicherter Exaktheit der Zahl im gesamten Wertebereich	- alle Zahlen lassen sich mit derselben relativen Genauigkeit darstellen sowohl betragsmäßig sehr kleine als auch sehr große Zahlen
-ermöglicht einfachere Implementierung von Operationen z.B. Multiplikation und Divisionen durch Schiebeoperationen	

2.4 Lösungsansatz cLoad [RA]

Ein Befehl zum Laden einer komplexen Zahl aus dem Speicher in dem Akkumulator kann sehr einfach realisiert werden. Wenn wir eine komplexe Zahl $z = a + i * b$ betrachten, steht im Register RA die Adresse der auf die reelle Teil a zeigt. Gleichfalls, wird die imaginäre Teil b von der Adresse $RA + 1$ gezeigt.

Wir können also abschätzen, dass insgesamt zwei Speicherzugriffe benötigt werden da das Parameter des Befehls nur eine Adresse repräsentiert und keinen richtigen Wert. Daraus folgt, dass **cLoad** mindestens vier Takte lang ist, da für jedes Speicherzugriff mindestens zwei benötigt werden.

Folgendes wäre ein intuitiver Lösungsansatz für das Laden einer komplexen Zahl, aus dem Speicher, im Akkumulator:

1. Den Inhalt des Registers RA auf den Adressbus legen, um den Wert der in dieser Adresse liegt im nächsten Takt zu erhalten (die Zahl a).
2. Der Wert liegt jetzt auf dem Datenbus. Diesen in das Register 6 (reeller Teil des Akkumulators) schieben.
3. Die Adresse, die sich im Register RA befindet, anhand das Carry-Flag inkrementieren und auf den Adressbus legen. Der Effekt ist der gleiche wie im Schritt 1, mit dem Unterschied dass RA jetzt auf den imaginären Teil b der Zahl zeigt.
4. Der zweite Schritt wird nochmal ausgeführt mit dem Unterschied, dass der Wert in das Register 7 (imaginärer Teil des Akkumulators) geschoben werden muss.

Die folgende Abbildung stellt eine Skizze des Befehls dar, die später in der Implementierung, mithilfe von jMic, implementiert wird:

79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IR	I3	I2	I1	I0	MMIO	K15	K14	K13	K12	K11	K10	K9	K8	K7	K6	K5	K4	K3	K2	K1	K0	SRC	FUNC	DEST	RA_ADDR	RB_ADDR	Y-MUX	CIN-MUX	Schiebe- steuerung	Statuzregister Test	AM2310- Befehle	Direktdaten	BZ	IRHS																																													
AM2301																										AM2304										AM2310																																											
D15	X	X																						ZA	ADD	NOP	*	IR	*	X	AB	H	CO	X	H	H		X	PS	CONT		*		H	H	H	H	R																															
D13	X	D																					DZ	ADD	RAMF	*	IR	6	MR	H	H	CO	X	H	H		X	PS	CONT		*		H	H	H	H	R																																
D13	X	*																					ZA	ADD	NOP	*	IR	*	X	AB	H	C1	*	H	H		X	PS	CONT		*		H	H	H	H	R																																
D13	X	D																					DZ	ADD	RAMF	*	IR	7	MR	H	H	CO	*	H	H		X	PS	CJP		IFETCH		H	H	H	H	R																																

Abbildung 1: Skizze des cLoad [RA] Befehls als Mikroprogramm

2.5 Lösungsansatz cStore [RB]

Der **cStore** Befehl führt das Gegenteil von **cLoad** aus: Er kopiert den Akkumulator in den Speicher in der Zieladresse, die im Register RB steht. Wie schon erwähnt, besteht der Akkumulator aus zwei Registern: Register 6 und 7. Im Register 6 steht der reelle Teil a und im Register 7 die imaginäre Teil der komplexen Zahl. Um den Akkumulator erfolgreich zu kopieren, müssen die Werte, die sich im Register 6 bzw. 7 befinden zu den Speicherzellen, die durch RB bzw. $RB + 1$ gezeigt werden, kopiert werden. Es werden also zwei Schreibzugriffe benötigt, für welche jeweils mindestens zwei Takte gebraucht werden. Daraus folgt, dass der **cStore** Befehl, genauso wie **cLoad**, mindestens vier Takte lang sein kann.

Folgendes wäre einen Lösungsansatz für das Kopieren des Akkumulators in den Speicher:

1. Den Inhalt des Registers RB auf den Adressbus legen und MWE auf „Write“ setzen, da ein Schreibzugriff an die Zieladresse die im Register RB enthalten ist, gestartet werden soll. Im nächsten Takt werden die Daten die im Datenbus liegen, in der durch RB angegebenen Speicheradresse, geschrieben.
2. Den Inhalt des Registers 6 in den Datenbus schieben und MWE wieder auf „Read“ setzen. Im nächsten Takt steht der Inhalt im Speicher.
3. Die Adresse, welche sich im Register RB befindet, anhand das Carry-Flag inkrementieren und auf den Adressbus legen. Der Effekt ist der gleiche wie in Schritt 1.
4. Schritt 2 wiederholen mit dem Unterschied, dass nicht Register 6 verwendet wird, sondern Register 7 (zweiter Teil des Akkumulators). Nach diesem Takt wurde der Akkumulator erfolgreich in den Speicher kopiert.

Die folgende Abbildung stellt eine Skizze des Befehls dar, die später in der Implementierung, mithilfe von jMic, implementiert wird:

76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000
13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															

Abbildung 2: Skizze des cStore [RB] Befehls als Mikroprogramm

2.6 Lösungsansatz cAdd [RA]

Der **cAdd** Befehl kommt ins Spiel, wenn man eine komplexe Zahl zum dem Akkumulator addieren möchte. In Absatz 2.1 wurde die Addition zwei solcher Zahlen definiert. Sie kann für diesen Lösungsansatz sehr behilflich sein, vor allem, wenn man die folgende Gleichung anschaut:

$$z_1 + z_2 = (a_1 + a_2) + i(b_1 + b_2)$$

Da der Akkumulator selbst eine komplexe Zahl ist, könnte man die Gleichung für die Implementierung des **cAdd** Befehls verwenden:

$$accu = z + accu = ([RA] + r6) + i([RA + 1] + r7).$$

Das heißt: Um eine komplexe Zahl zum Akkumulator addieren zu können, muss man die reellen Teile bzw. imaginären Teile zusammenaddieren.

2.6.1 Ansatz A

Folgendes wäre ein Lösungsansatz für das Addieren einer komplexen Zahl zum Akkumulator, unter Beachtung, dass RA die Adresse ist, die auf Wert der Zahl zeigt:

1. Den Inhalt des Registers RA auf den Adressbus legen und den lesenden Speicherzugriff starten.
2. Daten die im Datenbus liegen mit dem Wert, die sich im Register 6 befinden zusammenaddieren und in das Register 6 zurückschreiben. Das Maschinenstatusregister auf „Low“ setzen.
3. Adresse im Register RA anhand des Setzens des Carry-Flags um Eins inkrementieren und auf den Adressbus legen.
4. Daten, die im Datenbus liegen mit dem Wert, die sich im Register 7 befinden zusammenaddieren und in das Register 7 zurückschreiben. Hier wird das Maschinenstatusregister auf „High“ gesetzt!

Die folgende Abbildung stellt eine Skizze des Befehls dar, die später in der Implementierung, mithilfe von jMic, implementiert wird:

2.7.1 Ansatz A

Folgende Schritte wären ein Ansatz für das Subtrahieren einer komplexen Zahl vom Akkumulator:

1. Befehlszähler auf den Adressbus laden, um im nächsten Takt die Adresse der komplexen Zahl zu erhalten.
2. Die Adresse liegt auf dem Datenbus. Diese im Q Register speichern, um zu vermeiden, dass die Adresse nicht verloren geht. Gleichzeitig die Adresse auf den Adressbus legen. Im nächsten Takt liegt der erste Teil der Zahl auf dem Datenbus.
3. Die Zahl, die im Register 6 liegt von der Zahl, die im Datenbus liegt subtrahieren. Das Maschinenstatusregister auf „Low“ setzen und das Ergebnis zurück in das Register 6 schreiben.
4. Die Adresse, die im Q Register zwischengespeichert ist um Eins inkrementieren und auf den Adressbus legen.
5. Der zweite Teil der Zahl liegt jetzt auf dem Datenbus. Register 7 mit diesem Wert subtrahieren und in das gleiche Register zurückschreiben.

Die folgende Abbildung stellt eine Skizze des Befehls dar, die später in der Implementierung, mithilfe von jMic, implementiert wird:

18	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Interrupt		Konstante												SRC	FUNC	RA_ADDR		RB_ADDR		T-MUX	CIR-MUX	Schiebe steuerung	Statusregister Test		AM2910- Befehle		Direktdaten				BZ	IRHS																																													
														AM2901												AM2904												AM2910				BZ	IRHS																																				
D15	D13	X	x	x												ZA	ADD	NOP	x	IR	x	IR			X	H	H	x	PS	CONT	x			H	H	H	H	H	R																																								
D15	D13	X	D	x												DZ	ADD	QREG		IR	x	IR			X	H	H	x	PS	CONT	x			H	H	H	H	H	R																																								
D15	D13	X	D													DA	SUBR	RAMF	6	MR	6	MR			X	H	L	x	PS	CONT	x			H	H	H	H	H	R																																								
D15	D13	X	x	x												ZQ	ADD	NOP	x	IR	x	IR			X	H	H	x	PS	CONT	x			H	H	H	H	H	R																																								
D15	D13	X	D	x												DA	SUBR	RAMF	7	MR	7	MR			X	H	H	x	PS	CJP	IFETCH			H	H	H	H	H	R																																								

Abbildung 4: Skizze des cSub [addr] Befehls als Mikroprogramm

2.7.2 Ansatz B

Folgende Schritte wären ein Alternativer Ansatz für das Subtrahieren einer komplexen Zahl vom Akkumulator:

1. Befehlszähler auf den Adressbus laden, um im nächsten Takt die Adresse der komplexen Zahl zu erhalten.
2. Die Adresse liegt auf dem Datenbus. Diese im Q Register speichern, um zu vermeiden, dass die Adresse verloren geht. Gleichzeitig die Adresse auf den Adressbus legen. Im nächsten Takt liegt der erste Teil der Zahl auf dem Datenbus.
3. Den Datenbus und 0 mit EXNOR in r0 zwischenspeichern
4. R0 mit 1 inkrementieren und auf r6 Addieren
5. Die Adresse, die im Q Register zwischengespeichert ist um Eins inkrementieren und auf den Adressbus legen.
6. Den Datenbus und 0 mit EXNOR in r0 zwischenspeichern
7. R0 mit 1 inkrementieren und auf r7 Addieren

3 Bewertung der Lösungsansätze

3.1 cAdd [RA]

Lösungsansatz A		Lösungsansatz B	
Vorteile	Nachteile	Vorteile	Nachteile
<ul style="list-style-type: none"> - Verwendung des Carry-Flags ermöglicht die Vermeidung unnötiger Zwischenspeicherungen - Die Ausführung der zur Verfügung stehender Addition Operation ermöglicht intuitiver und lesbarer „Code“. 	<ul style="list-style-type: none"> - Carry-Flag nicht für das Inkrementieren Speicheradressen gedacht 	<ul style="list-style-type: none"> - Anhand der Zwischenspeicherung der Speicheradresse im Q Register, wird das Carry-Flag nicht mehr „misshandelt“. - „Sauberer“ Code statt abgetrennter - Vermeidung von Fehler. 	<ul style="list-style-type: none"> - Durch „Sauberen“ Code leidet die Effizienz des Mikroprogramms. - Benötigt mehr Takte!

3.2 cSub [addr]

Lösungsansatz A		Lösungsansatz B	
Vorteile	Nachteile	Vorteile	Nachteile
<ul style="list-style-type: none"> - Intuitivere Lösungsansatz - Nachvollziehbar da jede Mikroinstruktion selbsterklärend ist. - Da der Subtraktionsbefehl verwendet wird, werden alle benötigten Register ausgenutzt. - Keine Register mit redundante Daten. 	<ul style="list-style-type: none"> - Wie bei der Addition, das Subtraktion kann effizienter realisiert werden. - Takte können gespart werden. 	<ul style="list-style-type: none"> - Kann verwendet werden falls die Subtraktion zweier Zahlen, in der Maschine, für den Entwickler nicht zur Verfügung steht bzw. nicht implementiert ist. 	<ul style="list-style-type: none"> - Benötigt sehr viele Takte. - Sub Befehl in Mi Maschine vorhanden deswegen unrentabel

4 Entscheidungswahl

Bei cLoad [RA] und cStore [RB] wird keine zweiten Lösungsansätze angeboten, da diese befehle zu trivial sind, um einen zweiten sinnvollen Lösungsansatz aufzuzeigen.

4.1 cAdd [RA]

Für cAdd [RA] wird Lösungsansatz A ausgewählt, da es sich bei diesem Ansatz um den effizienteren Ansatz handelt und dieser ohne zusätzliche Register auskommt. Auch die Implementierung in der mikroprogrammierbaren Maschine sollte diesbezüglich leichter und verständlicher vonstattengehen.

4.2 cSub [addr]

Für cSub [addr] wird Lösungsansatz A ausgewählt, da es sich bei diesem Ansatz um den Intuitiveren, effizienteren Ansatz handelt und dieser mit weniger zusätzlichen Register auskommt. Auch die Implementierung in der mikroprogrammierbaren Maschine sollte diesbezüglich leichter und verständlicher vonstattengehen.