Einführung in die Technische Informatik (ETI)

Beschreibung der mikroprogrammierbaren Maschine

Ausgabe Wintersemester 09/10, Stand: 11. Dezember 2009

Wolfgang Karl, Max Walter, Josef Weidendorfer, Roland Wismüller

Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR-TUM)
Institut für Informatik der Technischen Universität München

4 Bereich II: Mikroprogrammierung

Wolfgang Karl, Roland Wismüller

4.1 Aufbau des mikroprogrammierbaren ETI-Rechners

In der Vorlesung Einführung in die Technische Informatik, in den Übungen hierzu und im Praktikum wird das Konzept der Mikroprogrammierung am Beispiel eines mikroprogrammierbaren Rechners demonstriert. Im folgenden wird dieser mikroprogrammierbare Beispielrechner eingeführt, wobei die Beschreibung nur auf dessen wesentliche Komponenten und deren Funktionsweise eingeht und auf eine detaillierte Darstellung der Einfachheit wegen verzichtet wird.
Beschreibungen von Einzelheiten der Maschine, die nur für das Praktikum, nicht aber für die Übung relevant sind, sind durch blaue Schrift auf grauem Grund (Zeichnungen und Tabellen) bzw. blaue Schrift und Umrahmung (Text) markiert.

Der mikroprogrammierte Rechner besteht entsprechend dem von Neumann'schen Konzept aus einem Leitwerk zur Generierung der Steuersignale für die einzelnen Komponenten des Rechners, einem Rechenwerk zur Verarbeitung von 16-Bit Integer-Daten, einem Speicherwerk mit einem $64K \times 16$ Bit großen Hauptspeicher und einem Ein-/Ausgabewerk, über das die Kommunikation mit dem PC erfolgt. Die einzelnen Werke sind über einen 16 Bit breiten Datenbus und einen 16 Bit breiten Adreßbus miteinander verbunden.

Das **Leitwerk** enthält einen $4K \times 80$ Bit großen Mikroprogrammspeicher (MPS), in dem die Mikroprogramme abgelegt sind. Ein Mikroprogramm besteht aus einer Folge von Mikroinstruktionen. In jedem Taktzyklus wird eine Mikroinstruktion ausgewählt und alle in ihr zusammengefaßten Mikrooperationen werden gleichzeitig zur Ausführung angestoßen. Die Auswahl der im nächsten Taktzyklus auszuführenden Mikroinstruktion erfolgt durch das Mikroleitwerk (MLW), das in jedem Taktzyklus eine 80-Bit breite Speicherzelle des Mikroprogrammspeichers adressiert.

Jedes Mikroprogramm kann einen Maschinenbefehl einer virtuellen Zielmaschine implementieren. Diese virtuelle Zielmaschine ist durch die Menge der durch Mikroprogramme implementierten Maschinenbefehle und das vereinbarte Programmiermodell definiert. Die Programme, die mit den Maschinenbefehlen der Zielmaschine geschrieben werden können, stehen zusammen mit den zu verarbeitenden Daten im Hauptspeicher.

Das **Rechenwerk** besteht aus einer Verarbeitungseinheit (der arithmetischen und logischen Einheit, ALU), die arithmetische und logische Operationen auf 16-Bit breiten Operanden ausführt, die von einer Registerdatei kommen können oder vom Datenbus übernommen werden. Darüberhinaus können Schiebeoperationen ausgeführt werden.

Mit Hilfe der mikroprogrammierbaren Maschine soll zum einen das Konzept bzw. die Technik der Mikroprogrammierung demonstriert werden, zum anderen können die in einem Rechner ablaufenden Vorgänge bei der Abarbeitung eines Maschinenprogramms vermittelt werden.

In Abbildung 4.1 ist die mikroprogrammierbare Maschine dargestellt, wobei zusätzlich angezeigt ist, aus welchen Feldern des Mikroinstruktionsregisters die einzelnen Komponenten des mikroprogrammierbaren Rechners gesteuert werden. (Der Übersichtlichkeit wegen ist das Unterbrechungswerk und das Ein-/Ausgabewerk nicht dargestellt.)

Die Bausteine, mit denen die einzelnen Werke aufgebaut sind, werden im folgenden vereinfacht beschrieben. Für eine genaue Beschreibung der Funktionsweise der Bausteine der Familie Am2900 sei auf die Datenblätter verwiesen.

4.1.1 Das Leitwerk

Das Leitwerk ist aus dem Sequencer–Baustein Am2910 (Mikroleitwerk) und dem Mikroprogrammspeicher mit dem Mikroinstruktionsregister aufgebaut. Weiterhin enthält das Leitwerk einen 16 Bit breiten Befehlszähler (BZ) mit einem Inkrementierer und ein 16 Bit breites Instruktionsregister mit dem in Abbildung 4.1 dargestellten Format. Der Inhalt des Befehlszählers kann um 1 inkrementiert werden und auf den Adreß– und/oder Datenbus ausgegeben werden. Darüberhinaus können Daten vom Datenbus in den Befehlszähler übernommen werden.

4.1.1.1 Beschreibung des Sequencer–Bausteins Am2910

Der Sequencer-Baustein Am2910 hat die Aufgabe, in einem Taktzyklus die Adresse für die nächste auszuführende Mikroinstruktion zu generieren.

Damit auch mit Mikroprogrammen die von Maschinenprogrammen her bekannten Kontrollflußstrukturen (sequentieller Programmfluß, bedingte und unbedingte Sprünge sowie Schleifen) programmiert werden können enthält der Sequencer–Baustein Am2910 alle für eine komplexe Adreßbildung notwendigen Komponenten: den Adreßinkrementierer für die lineare Adreßfortschaltung, den Adreßkeller für Rücksprungadressen von Mikro-Unterprogrammen oder für Schleifenanfangsadressen, einen Schleifenzähler, den Folgeadreßmultiplexer für die Auswahl der verschiedenen Adreßquellen und eine Bedingungslogik für bedingte Aktionen. Darüber hinaus ist der Mikrobefehlszähler auf dem Baustein integriert.

Die nachfolgende Beschreibung der Befehle beschränkt sich auf die zu Verständnis der Funktionsweise des Bausteins notwendigen Informationen. Abbildung 4.2 zeigt das Blockschaltbild des Bausteins und in Tabelle 4.1 sind die möglichen Mikrooperationen zur Steuerung des Bausteins aufgeführt.

Als zentrales Element für die Auswahl der Folgeadresse arbeitet ein **Folgeadreßmultiplexer**, der genau eine von fünf möglichen Adreßquellen auswählen kann: den Mikrobefehlszähler, den Rücksprungadreßkeller, das multifunktionale Zählerregister, den am D–Eingang anliegenden Adreßwert oder die Konstante 0. Der Folgeadreßmultiplexer wird durch die im Mikroinstruktionswort (Feld 18 – 21) stehende Mikrooperation sowie bei bedingten Befehlen durch die Bedingungslogik auf dem Baustein gesteuert.

Der **Mikrobefehlszähler** dient zur linearen Adreßfortschaltung. Die vom Adreßmultiplexer ausgewählte Adresse wird zum einen über den Y–Ausgang an den Mikroprogrammspeicher gegeben und zum anderen in einem Inkrementierer um 1 erhöht, womit eine mögliche Folgeadresse im Mikrobefehlszähler für den nächsten Zyklus bereitsteht. Der Inhalt des Mikrobefehlszählers wird dann ausgewählt, wenn im Mikroinstruktionswort (Feld 18 – 21) die Mikrooperation CONT[1] (für Continue) steht.

Der fünf Stufen tiefe **Rücksprungadreßkeller** wird für das Zwischenspeichern von Rücksprungadressen bei (geschachtelten) Mikro-Unterprogrammen und von Anfangsadressen bei Mikroprogrammschleifen verwendet. Der Rücksprungadreßkeller wird über einen Kellerzeiger verwaltet, welcher immer auf das letzte in den Keller geschriebene Wort verweist. Die zulässigen Operationen auf dem Keller sind PUSH und POP. Bei PUSH wird die im Mikrobefehlszähler bereitstehende Adresse als neues Kellerelement oben auf den Keller geschrieben. Bei POP wird das oberste Kellerelement entfernt und sein Inhalt erscheint am F–Eingang des Folgeadreßmultiplexers.

Das multifunktionale **Zählerregister** dient sowohl zur Zwischenspeicherung von Adressen als auch als Schleifenzähler. Dieses Register wird über den 12 Bit breiten D–Eingang des Bausteins geladen.

Der am **D–Eingang** anliegende Bus ist ein Tri–State–Bus, so daß prinzipiell mehrere Quellen auf ihn geschaltet werden können. Die Standardquellen sind das Direktdatenfeld (Branch–Address–Feld, BAR) des Mikroinstruktionsregisters oder ein Abbildungsspeicher (Mapping–PROM). Eine weitere Adreßquelle, mit welcher der D–Eingang beschaltet werden kann, ist typischerweise ein „Vector–Mapping–PROM“, das aus einem Unterbrechungseingangsregister eine Startadresse für ein Mikroprogramm produziert, das die Unterbrechungsbehandlung dann durchführt. (Die Komponenten zur Unterbrechungsbehandlung sind der Einfachheit wegen nicht im Blockschaltbild unseres Beispielrechners aufgeführt). Die Aktivierung einer dieser Quellen übernimmt die bausteininterne Befehlsdekodierung. Wird die Folgeadresse aus dem Direktdatenfeld des Mikroinstruktionsregisters übernommen, so handelt es sich um einen expliziten Sprung (gegebenenfalls in Abhängigkeit einer Bedingung), da

[1] Es werden im nachfolgenden Text anstelle der weniger lesbaren Bitmuster die mnemotechnischen Namen verwendet.

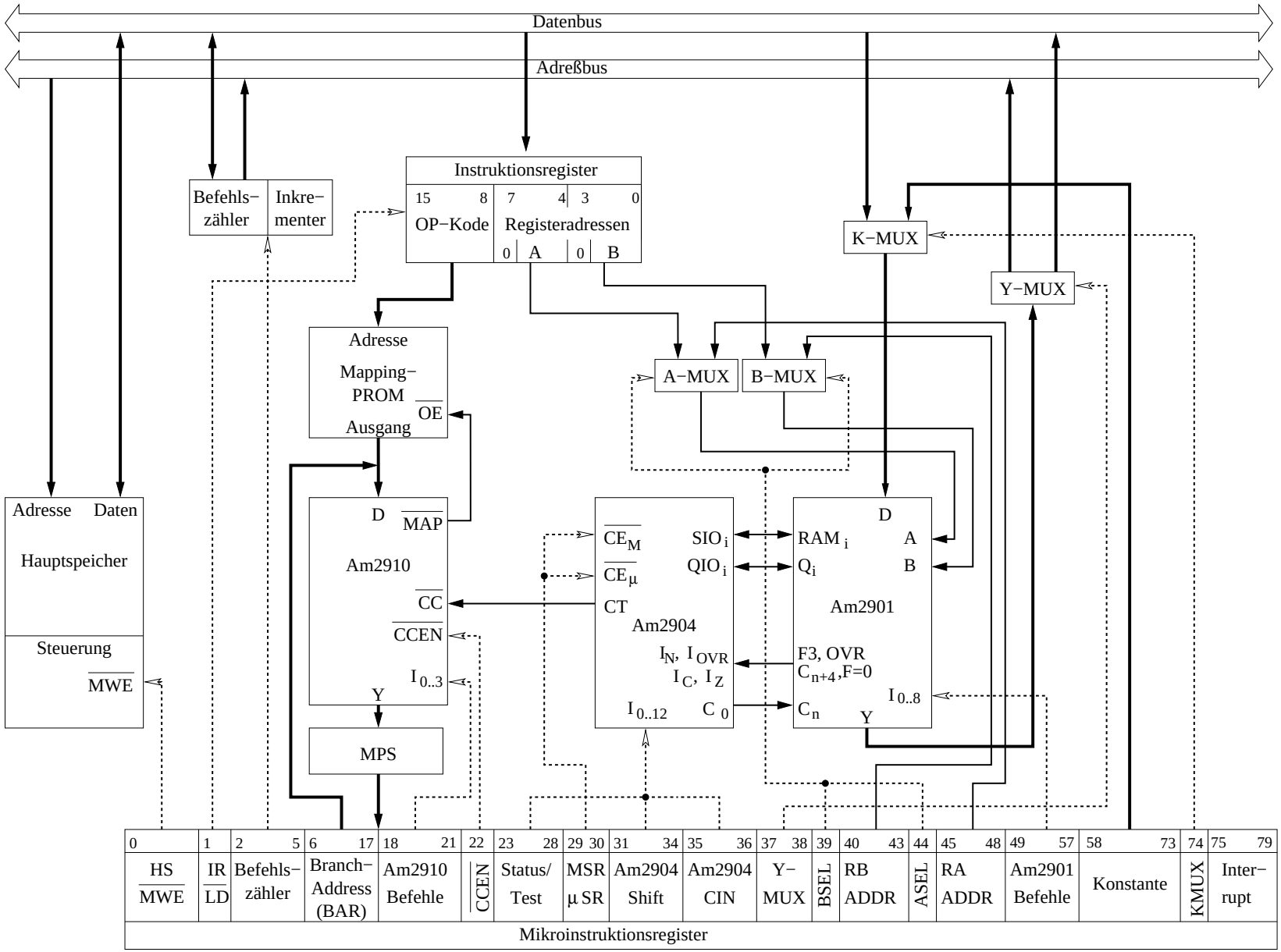


Abbildung 4.1: Blockschaltbild des mikroprogrammierbaren Beispielrechners

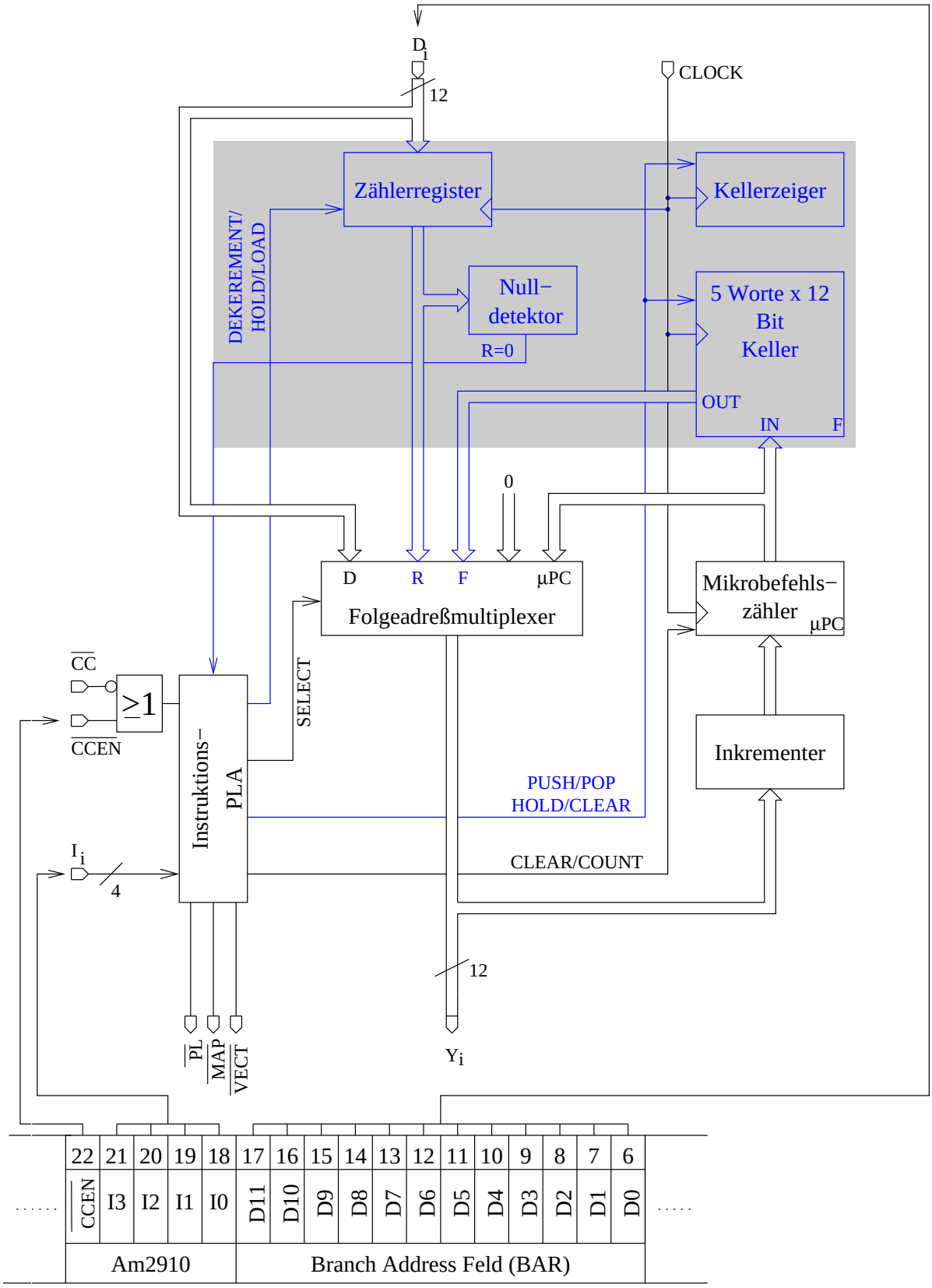


Abbildung 4.2: Funktionsschaltbild des Mikroleitwerks (Sequencer) Am2910

das Sprungziel als Absolutadresse in der laufenden Mikroinstruktion angegeben wird. Wenn die Folgeadresse vom „Mapping–PROM“ geliefert wird, ist es die Einsprungadresse in ein neues Mikroprogramm, dessen Startadresse über die Abbildung im Mapping–PROM aus dem Operationskodeteil des Instruktionsregisters gebildet wird (Dekodierphase des Makroinstruktionszyklus).

Soll beispielsweise eine Mikroprogrammspeicheradresse, die als Sprungziel im BAR-Feld des Mikroinstruktionsregisters steht, ausgewählt werden, muß im Mikroinstruktionswort (Feld 18 – 21) die Operation CJP (für Conditional Jump Pipeline) stehen.

Die für die **bedingte Adreßfortschaltung** notwendigen Statussignale werden vom Bedingungsmultiplexer des Bausteins Am2904 (siehe Abschnitt 4.1.2.2) ausgewählt und liegen am $\overline{CC}$–Eingang des Bausteins Am2910 an. Die Bedingungslogik prüft das am $\overline{CC}$–Eingang anliegende Signal nur, wenn der Mikroprogrammierer im Mikroinstruktionswort (Feld 22) $\overline{CCEN}$ aktiv (d. h. =0) setzt. Ansonsten werden die bedingten Operationen unbedingt ausgeführt.

Die 16 möglichen **Adreßfortschaltbefehle** des Bausteins, die durch die Instruktionsbit I_0 bis I_3 kodiert werden, sind in der Tabelle 4.1 beschrieben. Die in der Spalte „Mnemo“ angegebenen Namen sind für die Operationen sinnvollerweise vereinbart worden.

			Z/	FAIL 1)		PASS 2)		Z/	OE
I_3–I_0	MNEMO	Name	Reg. Inh.	Y	Keller	Y	Keller	Reg.	En.
0	JZ	Jump Zero	X	0	clear	0	clear	hold	PL
1	CJS	Cond JSB PL	X	μPC	hold	D	push	hold	PL
2	JMAP	Jump Map	X	D	hold	D	hold	hold	MAP
3	CJP	Cond Jump PL	X	μPC	hold	D	hold	hold	PL
4	PUSH	Push/Cond Ld Cntr	X	μPC	push	μPC	push	3)	PL
5	JSRP	Cond JSB R/PL	X	R	push	D	push	hold	PL
6	CJV	Cond Jump Vector	X	μPC	hold	D	hold	hold	VEC
7	JRP	Cond Jump R/PL	X	R	hold	D	hold	hold	PL
8	RFCT	Repeat Loop	$\neq 0$	F	hold	F	hold	dec	PL
		CNTR $\neq 0$	$= 0$	μPC	pop	μPC	pop	hold	PL
9	RPCT	Repeat PL	$\neq 0$	D	hold	D	hold	dec	PL
		CNTR $\neq 0$	$= 0$	μPC	hold	μPC	hold	hold	PL
10	CRTN	Cond Rtn	X	μPC	hold	F	pop	hold	PL
11	CJPP	CJP & Pop	X	μPC	hold	D	pop	hold	PL
12	LDCT	Ld Cntr & Cont	X	μPC	hold	μPC	hold	load	PL
13	LOOP	Test End Loop	X	F	hold	μPC	pop	hold	PL
14	CONT	Continue	X	μPC	hold	μPC	hold	hold	PL
15	TWB	3–Way–Branch	$\neq 0$	F	hold	μPC	pop	dec	PL
			$= 0$	D	pop	μPC	pop	hold	PL

1): $\overline{CCEN} = 0$ (Mnemo: C) **and** $\overline{CC} = 1$ X: unerheblich, “Don’t Care”

2): $\overline{CCEN} = 1$ (Mnemo: PS) **or** $\overline{CC} = 0$

3): **if** $\overline{CCEN} = 0$ **and** $\overline{CC} = 1$ **then** hold; **else** load.

Tabelle 4.1: Adreßfortschaltbefehle des Mikroleitwerks AM2910

Drei der hier aufgeführten Fortschaltbefehle hängen vom Inhalt des Zählerregisters ab, so daß für diese Befehle entsprechend den zwei Alternativen „$= 0$“ und „$\neq 0$“ jeweils die unterschiedlichen Aktionen beschrieben sind.

Für zehn Sequencer–Befehle bestimmt das am $\overline{CC}$–Eingang anliegende Signal die auszuführenden Aktionen. Wenn die Bedingung erfüllt ist (PASS–Bedingung), werden die in der Doppelspalte „PASS“ angegebenen Aktionen ausgeführt. Ist die Bedingung nicht erfüllt (FAIL–Bedingung), gelten die in der Doppelspalte „FAIL“ angeführten Aktionen. In der Y–Spalte wird jeweils die Quelle der vom Folgeadreßmultiplexer auszuwählenden und am Y–Ausgang anliegenden Adresse angegeben.
In der Spalte „Keller“ sind die jeweiligen Operationen auf dem Keller angegeben (CLEAR für zurücksetzen, PUSH, POP und HOLD für „keine Aktion“).

Die beiden letzten Spalten erklären die Aktionen auf dem Zählerregister und das vom internen Steuerblock (Instruktions–PLA) generierte Enable–Signal für eine der möglichen Quellen für den D–Eingang (MAP für das Mapping–PROM, PL für das Direktdatenfeld des Mikroinstruktionsregisters und VEC für das Vector–Mapping–PROM).

Der Befehl JZ produziert die **Mikroprogrammspeicheradresse 0**. An dieser Adresse kann sinnvollerweise ein Mikroprogramm zum Initialisieren des Systems stehen.

Die Befehle JMAP und CJV erlauben den **Einsprung in Mikroprogramme**, deren Startadressen von externen Adreßquellen (Mapping–PROM, Vector–Mapping–PROM) über den D–Eingang geliefert und an den Mikroprogrammspeicher weitergegeben werden.

Die **sequentielle Adreßfortschaltung** im laufenden Mikroprogramm geschieht mit dem Befehl CONT. Sprungbefehle sind CJP (Sprungziel steht im Direktdatenfeld des Mikroinstruktionsregister) und JRP (Sprungziele stehen im Direktdatenfeld des Mikroinstruktionsregister oder im Zählerregister, das vorher allerdings mit der Zieladresse geladen werden muß).

Mikro-Unterprogramme werden mit den Befehlen CJS (Anfangsadresse des Unterprogramms aus dem Mikroinstruktionsregister) und JSRP (Anfangsadresse alternativ aus dem Mikroinstruktionsregister oder aus dem Zählerregister) aufgerufen, wobei automatisch die Rücksprungadresse auf das oberste Kellerelement geschrieben wird. Bedingte Rücksprünge sind dann mit dem Befehl CRTN möglich, der die Rücksprungadresse vom Keller holt.

Eine Reihe von Befehlen dient der **Schleifenprogrammierung**. Der Befehl PUSH lädt in Abhängigkeit einer externen Bedingung das Zählerregister und legt die Schleifenanfangsadresse im Keller ab. Der Befehl LDCT lädt nur den Zähler. Die Befehle RPCT, RFCT und TWB sind Schleifenendebefehle, die zunächst prüfen, ob der Inhalt des Zählerregisters 0 ist. Wenn der Inhalt des Zählerregisters ungleich 0 ist, wird an die im Direktdatenfeld des Mikroinstruktionsregisters angegebene bzw. an die auf dem Keller hinterlegte Schleifenanfangsadresse verzweigt und der Inhalt des Zählerregisters um 1 erniedrigt. Bei dem Befehl TWB ist das Abfragen des Zählerregisters kombiniert mit dem Test einer externen Bedingung, so daß eine Dreiwegeverzweigung vorliegt. Der Befehl LOOP springt in Abhängigkeit einer externen Bedingung an die auf dem obersten Kellerelement liegende Schleifenanfangsadresse. Das Herunterzählen des Zählerregisters und die Verwaltung des Kellers erfolgt automatisch.

4.1.2 Das mikroprogrammierbare Rechenwerk

Zur Verarbeitung der Daten ist ein 16 Bit breites Rechenwerk vorgesehen. Es ist auf der Basis von Rechenwerkbausteinen, den kaskadierbaren Bitslice–Komponenten Am2901, aufgebaut. Vier dieser Module, ergänzt durch den Wortrandlogikbaustein Am2904, bilden das 16 Bit breite Rechenwerk. Seine Daten erhält das Rechenwerk vom Hauptspeicher, mit dem es über den Datenbus verbunden ist. Ergebnisse können auf den Datenbus und auf den Adreßbus, falls es sich um Adressen handelt, ausgegeben werden.

4.1.2.1 Beschreibung des Rechenwerkbausteins Am2901

Abbildung 4.3 zeigt die Grundstruktur des Rechenwerkbausteins Am2901. Da es sich um einen kaskadierbaren 4–Bit–Baustein handelt, seine Datenpfade und internen Komponenten also 4 Bit breit sind, können durch Aneinanderreihung solcher Module Rechenwerke mit einem um ein Vielfaches von vier Bit breiten Datenwort aufgebaut werden.

Als zentrale Komponenten enthält der Baustein eine Zweitor–Registerdatei (RAM) mit 16 Registern und eine arithmetisch/logische Einheit (ALU), die acht Mikrooperationen ausführen kann. Die Operanden liegen an den ALU–Eingängen R und S an und können fünf verschiedenen Quellen entstammen. Weiterhin verfügt der Baustein Am2901 über ein multifunktionales Q–Register sowie zwei Schiebeeinheiten (RAM– und Q–Schiebeeinheit).

Durch Anlegen von A– und B–Registeradressen an die **Registerdatei** können jeweils zwei beliebige – auch identische – Register über die beiden Ausgabewege A und B gelesen werden. Das Ergebnis einer ALU–Operation wird in das Register geschrieben, welches im Register–B–Adreßfeld des Mikroinstruktions– bzw. des Instruktionsregisters spezifiziert ist. Der Weg vom ALU–F–Ausgang zur Registerdatei führt über die **RAM–Schiebeeinheit**, wo

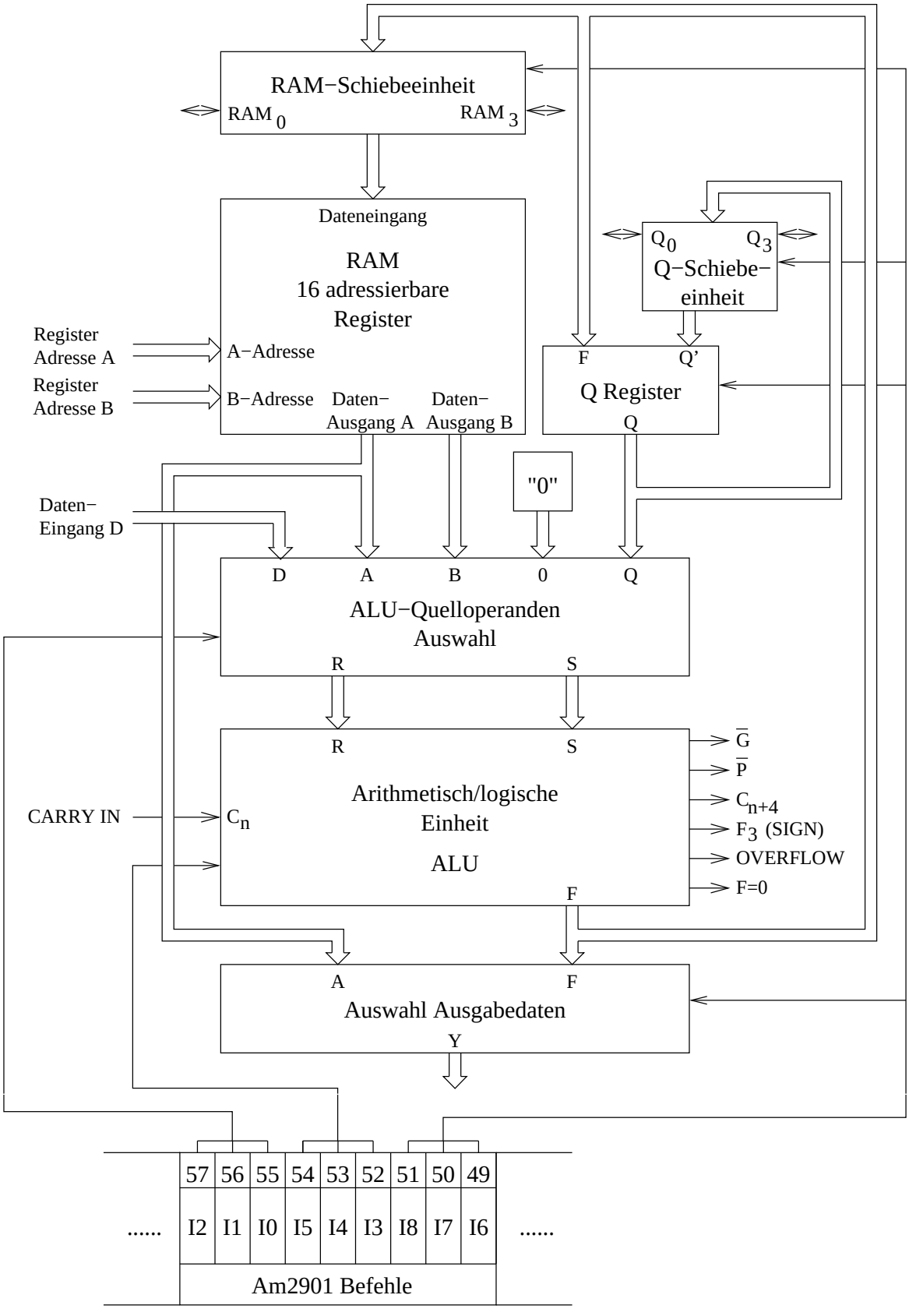


Abbildung 4.3: Funktionsschaltbild des Rechenwerkbausteins Am2901

Mnemo	Mikrokode				ALU–Quell–operanden	
	I_2	I_1	I_0	Oktal–wert	R	S
AQ	0	0	0	0	A	Q
AB	0	0	1	1	A	B
ZQ	0	1	0	2	0	Q
ZB	0	1	1	3	0	B
ZA	1	0	0	4	0	A
DA	1	0	1	5	D	A
DQ	1	1	0	6	D	Q
DZ	1	1	1	7	D	0

Tabelle 4.2: ALU Quelloperandensteuerung

Mnemo	Mikrokode				ALU Funktion	Symbol
	I_5	I_4	I_3	Oktal–wert		
ADD	0	0	0	0	R Plus S	R+S
SUBR	0	0	1	1	S Minus R	S−R
SUBS	0	1	0	2	R Minus S	R−S
OR	0	1	1	3	R OR S	R∨S
AND	1	0	0	4	R AND S	R∧S
NOTRS	1	0	1	5	$\overline{\text{R}}$ AND S	$\overline{\text{R}}$∧ S
EXOR	1	1	0	6	R XOR S	R⊕S
EXNOR	1	1	1	7	R XNOR S	$\overline{\text{R} \oplus \text{S}}$

Tabelle 4.3: ALU Funktionssteuerung

Mnemo	Mikrokode				RAM Funktion		Q-Reg. Funktion		Y Ausgang
	I_8	I_7	I_6	Oktal–wert	Schieben	Laden	Schieben	Laden	
QREG	0	0	0	0	X	NONE	NONE	F→Q	F
NOP	0	0	1	1	X	NONE	X	NONE	F
RAMA	0	1	0	2	NONE	F→B	X	NONE	A
RAMF	0	1	1	3	NONE	F→B	X	NONE	F
RAMQD	1	0	0	4	DOWN	F/2 →B	DOWN	Q/2 →Q	F
RAMD	1	0	1	5	DOWN	F/2 →B	X	NONE	F
RAMQU	1	1	0	6	UP	2F→B	UP	2Q→Q	F
RAMU	1	1	1	7	UP	2F→B	X	NONE	F

Tabelle 4.4: ALU Zielsteuerung (DOWN = Rechtsschieben, UP = Linksschieben)

Mnemo	I_{543}	ALU Funktion	I_{210} AQ 0	AB 1	ZQ 2	ZB 3	ZA 4	DA 5	DQ 6	DZ 7
			ALU Quelloperanden							
			A,Q	A,B	0,Q	0,B	0,A	D,A	D,Q	D,0
ADD	0	$C_n = 0$	A+Q	A+B	Q	B	A	D+A	D+Q	D
		R+S								
		$C_n = 1$	A+Q+1	A+B+1	Q+1	B+1	A+1	D+A+1	D+Q+1	D+1
SUBR	1	$C_n = 0$	Q−A−1	B−A−1	Q−1	B−1	A−1	A−D−1	Q−D−1	−D−1
		S−R								
		$C_n = 1$	Q−A	B−A	Q	B	A	A−D	Q−D	−D
SUBS	2	$C_n = 0$	A−Q−1	A−B−1	−Q−1	−B−1	−A−1	D−A−1	D−Q−1	D−1
		R−S								
		$C_n = 1$	A−Q	A−B	−Q	−B	−A	D−A	D−Q	D
OR	3	R∨S	A∨Q	A∨B	Q	B	A	D∨A	D∨Q	D
AND	4	R∧S	A∧Q	A∧B	0	0	0	D∧A	D∧Q	0
NOTRS	5	$\overline{R} \wedge S$	$\overline{A} \wedge Q$	$\overline{A} \wedge B$	Q	B	A	$\overline{D} \wedge A$	$\overline{D} \wedge Q$	0
EXOR	6	R⊕S	A⊕Q	A⊕B	Q	B	A	D⊕A	D⊕Q	D
EXNOR	7	$\overline{R \oplus S}$	$\overline{A \oplus Q}$	$\overline{A \oplus B}$	$\overline{Q}$	$\overline{B}$	$\overline{A}$	$\overline{D \oplus A}$	$\overline{D \oplus Q}$	$\overline{D}$

Die Instruktionsbit I_{543} und I_{210} sind in Oktaldarstellung.

Tabelle 4.5: Quelloperanden– und ALU–Funktionsmatrix

die Daten um ein Bit nach links bzw. nach rechts geschoben oder unverändert weitergegeben werden. Das Lesen der Register, die Verknüpfung der Operanden in der ALU und das Zurückschreiben des Ergebnisses erfolgt innerhalb eines Taktzyklus.

Die arithmetisch/logische Einheit kann, gesteuert über die Instruktionsbit I_3 bis I_5, auf den am ALU–R– und am ALU–S–Eingang anliegenden Daten drei arithmetische und fünf logische Operationen ausführen, wobei für die arithmetischen Operationen zu beachten ist, daß diese noch vom Übertrag, d. h. vom Wert des am C_n–Eingang anliegenden Signals, abhängen (siehe Tabelle 4.5). Die Operationen der ALU können der internen Registerdatei (A–Ausgang, B–Ausgang), dem Q–Register und einem unidirektionalen D–Eingang entstammen. Weiterhin kann die Null als Quelloperand auftreten.

Das **Q–Register** kann zur Speicherung von Zwischenergebnissen, als Quotientenregister bei Multiplikations– und Divisionsroutinen oder als Akkumulatorregister dienen. Der Inhalt des Q–Registers kann gleichzeitig mit einer ALU–Operation um ein Bit nach links oder rechts verschoben werden.

Über den **D–Eingang** können Daten aus einer externen Quelle eingelesen werden. In unserem Beispielrechner sind dies Daten, welche aus dem Hauptspeicher über den Datenbus geliefert werden oder Konstanten, die im Konstantenfeld (Bit 58 - 73) des Mikroinstruktionsregisters stehen. Über das Bit 74 des Mikroinstruktionswortes wird eine der beiden Möglichkeiten ausgewählt.In Tabelle 4.2 ist die Definition der Instruktionsbit I_0 bis I_2 zur Steuerung der Auswahl der Quelloperanden für die acht möglichen Kombinationen zu entnehmen. Die Tabelle 4.3 zeigt die Wirkung der ALU auf die Quelloperanden (ALU Funktionssteuerung).

Das Ergebnis einer ALU–Operation kann direkt über den Y–Ausgang ausgegeben und/oder verschiedenen internen Komponenten zugeführt werden. Die am **ALU–F–Ausgang** anliegenden Ergebnisse können, wie oben bereits beschrieben, in die Registerdatei oder in das Q–Register geschrieben werden. Die in die Registerdatei zu schreibenden Daten können zuvor in der RAM–Schiebeeinheit um ein Bit nach rechts oder nach links geschoben werden. Die am **Y–Ausgang** des Bausteins anliegenden Daten stammen entweder vom ALU–F–Ausgang (Ergebnis einer ALU–Operation) oder vom A–Ausgang der Registerdatei (Inhalt einer Registerzelle). Die Zielsteuerung, programmiert über die Instruktionsbit I_6 bis I_8, ist in Tabelle 4.4 definiert. Mit der Codierung RAMF der ALU-Zielsteuerung wird beispielsweise bestimmt, daß das Ergebnis am ALU-F-Ausgang in das Register geschrieben wird, das im Register-B-Adreßfeld spezifiziert wird und gleichzeitig über den Y-Bus ausgegeben wird. Im Unterschied dazu wird bei der Codierung RAMA der Inhalt des im Register–A–Adreßfeld spezifizierten Registers über den Y-Bus ausgegeben.

Der Baustein liefert eine Reihe von Statussignalen, die in einem der Statusregister des Bausteins Am2904 abgelegt werden können (siehe Abschnitt 4.1.2.2).

4.1.2.2 Der Wortrandlogikbaustein Am2904

Der Wortrandlogikbaustein Am2904 wird als Ergänzung zu dem Rechenwerkbaustein Am2901 verwendet, da er die im Zusammenhang mit der ALU notwendigen Funktionen wie die Steuerung des Übertrags, der Schiebeverbindungen, des Ablegens der Statussignale und des Abfragens von Bedingungen auf einem Chip integriert. Abbildung 4.4 zeigt vereinfacht den Aufbau des Wortrandlogikbausteins Am2904.

Der Baustein enthält zwei **Statusregister**, das Maschinenstatusregister (MSR) und das Mikrostatusregister (μSR), die unabhängig voneinander gesteuert und bitweise mit den entsprechenden Statussignalen wie Überlauf (OVR), Übertrag (C), Vorzeichen (N) und Null–Anzeige (Z) aus dem Rechenwerk (Ausgänge OVR, C_{n+4}, F_3, $F = 0$ des höchstwertigen Rechenwerkbausteins) über die vier Statuseingänge I_C, I_N, I_Z und I_{OVR} geladen werden können. Einzelne Bit können gesetzt bzw. zurückgesetzt und die Inhalte der beiden Register ausgetauscht werden, wobei Enablebit das Überschreiben der Register verhindern oder ermöglichen.

Die beiden Statusregister werden über die Instruktionsbit I_0 bis I_5 gesteuert. Die **Operationen auf den Statusregistern** lassen sich in die Gruppen Bit– (nur Mikrostatusregister), Register– und Lade–Operationen einteilen.

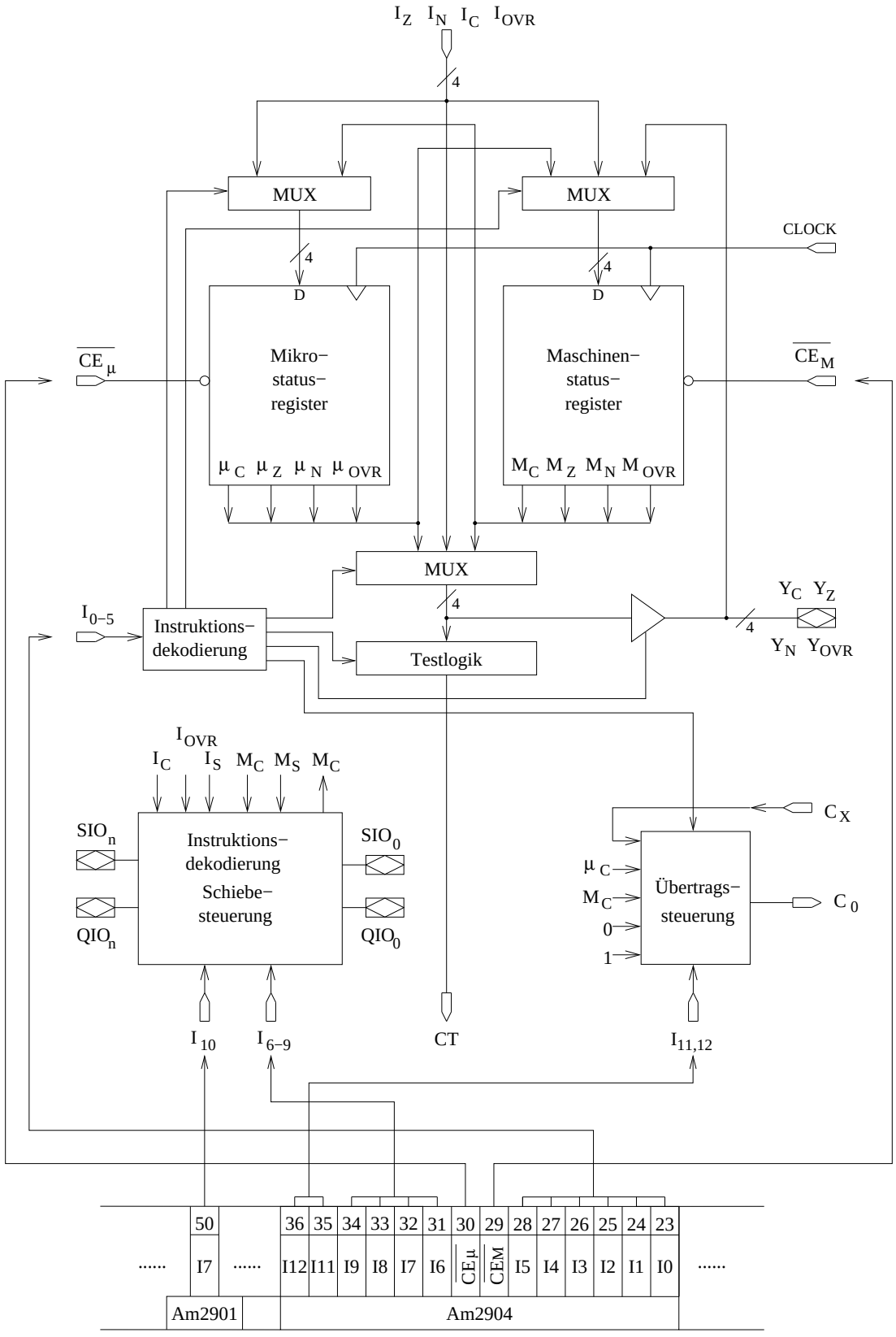


Abbildung 4.4: Funktionsschaltbild des Wortrandlogikbausteins Am2904

Tabelle 4.6 zeigt die vollständige Decodierung dieser Instruktionsbit; ausgelassen ist lediglich ihr Einfluß auf die Übertragssteuerung (s. Tabelle 4.9). Die mit μSR, MSR und CT beschrifteten Zeilen der Tabelle geben jeweils an, wie sich die Operation auf die Statusregister und die Testlogik (CT-Ausgang) auswirken. Man erkennt aus der Tabelle, daß die Codierung der Operationen relativ komplex und insbesondere nicht orthogonal ist (Aus diesem Grund gibt es auch keine mnemotechnischen Bezeichner für die Operationen). Zur Vereinfachung sind jedoch die wichtigsten Operationen auf den Statusregistern in Tabelle 4.7 zusammengestellt. Zu beachten ist, daß die dort angegebenen möglichen Operationscodes jeweils auch Nebeneffekte verursachen, die aus Tabelle 4.6 entnommen werden können. So löscht Z.B. der Code $I_{5..0} = 001000$ nicht nur das Z-Bit im Mikrostatusregister, sondern lädt auch das Maschinenstatusregister mit invertiertem C-Bit vom I-Eingang und setzt den CT-Ausgang entsprechend dem Ergebnis der Bedingung $\mu_C \vee \mu_Z = 0$.

Ein unerwünschtes Verändern der Statusregister kann mit Hilfe der Enablebit $\overline{CE_\mu}$ und $\overline{CE_M}$ (Bit 30 bzw. 29 des Mikroinstruktionsregisters) verhindert werden. Ein Statusregister wird nur dann verändert, wenn das entsprechende Enablebit auf LOW gesetzt ist.

Die Testlogik auf dem Wortrandlogikbaustein Am2904 gestattet die **Abfrage der Statusregister** μSR und MSR. Der ausgewählte Bedingungskode wird am Ausgang CT ausgegeben. Der CT–Ausgang ist mit dem Testeingang $\overline{CC}$ des Sequencerbausteins Am2910 verbunden.

Über die Instruktionsbit I_4 und I_5 kann bestimmt werden, in welchem Statusregister die Bedingungen abgefragt werden sollen. Mit $I_{5,4} = 01$ (Mnemo: MI) werden die Statusbit des Mikrostatusregisters abgefragt und mit $I_{5,4} = 10$ (Mnemo: MA) die des Maschinenstatusregisters. Mit den Instruktionsbit I_0 bis I_3 wird eine von 16 Bedingungen ausgewählt (siehe die CT-Zeilen in Tabelle 4.6). Acht dieser 16 Bitkombinationen liefern den Wert eines der Statusbit oder dessen Komplement. Die acht restlichen Bitkombinationen liefern als Ergebnis logische Verknüpfungen von Statusbit an den Ausgang CT.

Diese Bitkombinationen gestatten beispielsweise den Vergleich zweier nicht vorzeichenbehafteter oder Zweierkomplement–Zahlen nach den Kriterien „gleich“, „größer gleich“, „kleiner“ oder „kleiner gleich“. In Tabelle 4.8 sind als Beispiel verschiedene Kriterien aufgeführt, die nach der Subtraktion zweier Zahlen A und B geprüft werden können, wobei zwischen nicht vorzeichenbehafteten und Zweierkomplement–Zahlen unterschieden wird. Für jede Relation (Spalte 1) wird der Wert des bzw. der betreffenden Statusbit (Spalte 2 und 6) angegeben. In den Spalten 4,5 und 8,9 stehen die Bedingungscodes (jeweils oktal und hexadezimal), die zu programmieren sind, wenn die entsprechenden Bedingungen abzufragen sind. Wenn die Bedingung erfüllt ist, dann erscheint am CT-Ausgang und damit auch am Testeingang $\overline{CC}$ des Sequencerbausteins Am2910 eine Null. In den Spalten 3 und 7 sind die wichtigsten Mnemos für die Bedingungscodes aufgeführt.

Bezüglich des Zustands des **C-Statusbits nach der Subtraktion** vorzeichenloser Zahlen ist eine Besonderheit der ALU im Am2901 zu berücksichtigen. Die ALU realisiert intern eine Subtraktion $A - B$ duch Addition von A mit dem Komplement von B. Dies bedingt, daß bei der Subtraktion vorzeichenloser Zahlen das C-Statusbit entgegen der üblichen Erwartung dann gesetzt wird, wenn A größer oder gleich B ist (siehe Tabelle 4.8).

Falls vom Rechenwerk eine **Schiebeoperation** durchgeführt werden muß, so ist neben der entsprechenden Operation der ALU-Zielsteuerung (siehe Tabelle 4.4) auch der Wortrandlogikbaustein Am2904 zu programmieren.

Der Baustein erlaubt bei entsprechender Verbindung mit den Rechenwerkbausteinen 32 verschiedene Schiebe– und Ringschiebeaktionen zu mikroprogrammieren. Die Schiebeein– bzw. ausgänge SIO_0, SIO_n, QIO_0 und QIO_n sind mit den entsprechenden Eingängen der Rechenwerkbausteinen verbunden (SIO_0 mit RAM_0 des niedrigstwertigen Am2901–Bausteins, SIO_n mit RAM_3 des höchstwertigen Am2901–Bausteins, QIO_0 mit QIO_0 des niedrigstwertigen Am2901–Bausteins und QIO_n mit QIO_n des höchstwertigen Am2901–Bausteins, siehe Abbildung 4.7).

Die Codierungen der fünf Instruktionsbit I_6 bis I_{10} bestimmen die Schiebeaktion, wobei I_{10} entscheidet, in welche Richtung geschoben wird. Da die Schieberichtung für beide Bausteine (Am2901 und Am2904) gleich sein muß, ist der Pin I_{10} des Am2904 fest mit dem Pin I_7 des Bausteins Am2901 verbunden (Bit 50 im Mikroinstruktionswort). Die 16 Rechts–Schiebeaktionen sind in Tabelle 4.5 und die 16 Links–Schiebeaktionen sind in Tabelle 4.6 aufgeführt. Die dritte Spalte der beiden Tabellen zeigt jeweils die Wirkung der Schiebeaktion auf die Schiebeeinheiten der ALU und auf das Statusbit M_C des Maschinenstatusregisters. Dieses Statusbit wird dabei unabhängig vom Zustand des Freigabesignals $\overline{CE_M}$ (Bit 29 im Mikroinstruktionsformat) gesetzt. In den weiteren vier Spalten dieser Tabellen ist jeweils aufgeführt, welches Signal an den jeweiligen Schiebeein– und –ausgängen anliegt.

		I_{210}							
I_{543}		000	001	010	011	100	101	110	111
000	μSR	$M_X \rightarrow \mu_X$	$1 \rightarrow \mu_X$	$M_X \leftrightarrow \mu_X$	$0 \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_{Z,C,N} \rightarrow \mu_{Z,C,N}$ $I_V \vee \mu_V \rightarrow \mu_V$	$I_{Z,C,N} \rightarrow \mu_{Z,C,N}$ $I_V \vee \mu_V \rightarrow \mu_V$
	MSR	$Y_X \rightarrow M_X$	$1 \rightarrow M_X$	$M_X \leftrightarrow \mu_X$	$0 \rightarrow M_X$	$I_{Z,N} \rightarrow M_{Z,N}$ $M_V \leftrightarrow M_C$	$\overline{M}_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$
	CT	$(\mu_N \oplus \mu_V) \vee \mu_Z = 0$	$(\mu_N \oplus \mu_V) \vee \mu_Z = 1$	$\mu_N \oplus \mu_V = 0$	$\mu_N \oplus \mu_V = 1$	$\mu_Z = 0$	$\mu_Z = 1$	$\mu_V = 0$	$\mu_V = 1$
001	μSR	$0 \rightarrow \mu_Z$	$1 \rightarrow \mu_Z$	$0 \rightarrow \mu_C$	$1 \rightarrow \mu_C$	$0 \rightarrow \mu_N$	$1 \rightarrow \mu_N$	$0 \rightarrow \mu_V$	$1 \rightarrow \mu_V$
	MSR	$I_{Z,N,V} \rightarrow M_{Z,N,V}$ $\overline{I}_C \rightarrow M_C$	$I_{Z,N,V} \rightarrow M_{Z,N,V}$ $\overline{I}_C \rightarrow M_C$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$
	CT	$\mu_C \vee \mu_Z = 0$	$\mu_C \vee \mu_Z = 1$	$\mu_C = 0$	$\mu_C = 1$	$\overline{\mu_C} \vee \mu_Z = 0$	$\overline{\mu_C} \vee \mu_Z = 1$	$I_N \oplus M_N = 0$	$I_N \oplus M_N = 1$
010	μSR	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$
	MSR	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$
	CT	$(\mu_N \oplus \mu_V) \vee \mu_Z = 0$	$(\mu_N \oplus \mu_V) \vee \mu_Z = 1$	$\mu_N \oplus \mu_V = 0$	$\mu_N \oplus \mu_V = 1$	$\mu_Z = 0$	$\mu_Z = 1$	$\mu_V = 0$	$\mu_V = 1$
011	μSR	$I_{Z,N,V} \rightarrow \mu_{Z,N,V}$ $\overline{I}_C \rightarrow \mu_C$	$I_{Z,N,V} \rightarrow \mu_{Z,N,V}$ $\overline{I}_C \rightarrow \mu_C$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$
	MSR	$I_{Z,N,V} \rightarrow M_{Z,N,V}$ $\overline{I}_C \rightarrow M_C$	$I_{Z,N,V} \rightarrow M_{Z,N,V}$ $\overline{I}_C \rightarrow M_C$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$
	CT	$\mu_C \vee \mu_Z = 0$	$\mu_C \vee \mu_Z = 1$	$\mu_C = 0$	$\mu_C = 1$	$\overline{\mu_C} \vee \mu_Z = 0$	$\overline{\mu_C} \vee \mu_Z = 1$	$\mu_N = 0$	$\mu_N = 1$
100	μSR	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$
	MSR	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$
	CT	$(M_N \oplus M_V) \vee M_Z = 0$	$(M_N \oplus M_V) \vee M_Z = 1$	$M_N \oplus M_V = 0$	$M_N \oplus M_V = 1$	$M_Z = 0$	$M_Z = 1$	$M_V = 0$	$M_V = 1$
101	μSR	$I_{Z,N,V} \rightarrow \mu_{Z,N,V}$ $\overline{I}_C \rightarrow \mu_C$	$I_{Z,N,V} \rightarrow \mu_{Z,N,V}$ $\overline{I}_C \rightarrow \mu_C$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$
	MSR	$I_{Z,N,V} \rightarrow M_{Z,N,V}$ $\overline{I}_C \rightarrow M_C$	$I_{Z,N,V} \rightarrow M_{Z,N,V}$ $\overline{I}_C \rightarrow M_C$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$
	CT	$M_C \vee M_Z = 0$	$M_C \vee M_Z = 1$	$M_C = 0$	$M_C = 1$	$\overline{M_C} \vee M_Z = 0$	$\overline{M_C} \vee M_Z = 1$	$M_N = 0$	$M_N = 1$
110	μSR	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$
	MSR	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$
	CT	$(I_N \oplus I_V) \vee I_Z = 0$	$(I_N \oplus I_V) \vee I_Z = 1$	$I_N \oplus I_V = 0$	$I_N \oplus I_V = 1$	$I_Z = 0$	$I_Z = 1$	$I_V = 0$	$I_V = 1$
111	μSR	$I_{Z,N,V} \rightarrow \mu_{Z,N,V}$ $\overline{I}_C \rightarrow \mu_C$	$I_{Z,N,V} \rightarrow \mu_{Z,N,V}$ $\overline{I}_C \rightarrow \mu_C$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$	$I_X \rightarrow \mu_X$
	MSR	$I_{Z,N,V} \rightarrow M_{Z,N,V}$ $\overline{I}_C \rightarrow M_C$	$I_{Z,N,V} \rightarrow M_{Z,N,V}$ $\overline{I}_C \rightarrow M_C$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$	$I_X \rightarrow M_X$
	CT	$\overline{I}_C \vee I_Z = 0$	$\overline{I}_C \vee I_Z = 1$	$I_C = 0$	$I_C = 1$	$\overline{I_C} \vee I_Z = 0$	$\overline{I_C} \vee I_Z = 1$	$I_N = 0$	$I_N = 1$

V steht abkürzend für OVR, X steht stellvertretend für $\{Z, C, N, OVR\}$

Tabelle 4.6: (Fast) vollständige Decodierung der Am2904 Instruktionsbits $I_{5..0}$

Operation	Beschreibung	I_{543210} oktal	I_{543210} hex
$I_X \rightarrow \mu_X$	Lade die am Statuseingang I anliegenden Signale ins μSR	04, 05 20 – 27 32 – 47 52 – 67 72 – 77	04, 05 10 – 17 1A – 27 2A – 37 3A – 3F
$I_Z \rightarrow \mu_Z$ $I_N \rightarrow \mu_N$ $I_{OVR} \rightarrow \mu_{OVR}$ $\overline{I_C} \rightarrow \mu_C$	Lade die am Statuseingang I anliegenden Signale ins μSR wobei das C-Bit invertiert wird	30, 31 50, 51 70, 71	18, 19 28, 29 38, 39
$I_X \rightarrow M_X$	Lade die am Statuseingang I anliegenden Signale ins MSR	06, 07 12 – 27 32 – 47 52 – 67 72 – 77	06, 07 0A – 17 1A – 27 2A – 37 3A – 3F
$I_Z \rightarrow M_Z$ $I_N \rightarrow M_N$ $I_{OVR} \rightarrow M_{OVR}$ $\overline{I_C} \rightarrow M_C$	Lade die am Statuseingang I anliegenden Signale ins MSR wobei das C-Bit invertiert wird	10, 11 30, 31 50, 51 70, 71	08, 09 18, 19 28, 29 38, 39
$1 \rightarrow \mu_X$	Setze alle Bit des μSR	01	01
$0 \rightarrow \mu_X$	Lösche alle Bit des μSR	03	03
$1 \rightarrow M_X$	Setze alle Bit des MSR	01	01
$0 \rightarrow M_X$	Lösche alle Bit des MSR	03	03
$\mu_X \leftrightarrow M_X$	Tausche die Inhalte von μSR und MSR	02	02
$1 \rightarrow \mu_Z$	Setze das Z-Bit im μSR	11	09
$0 \rightarrow \mu_Z$	Lösche das Z-Bit im μSR	10	08
$1 \rightarrow \mu_C$	Setze das C-Bit im μSR	13	0B
$0 \rightarrow \mu_C$	Lösche das C-Bit im μSR	12	0A
$1 \rightarrow \mu_N$	Setze das N-Bit im μSR	15	0D
$0 \rightarrow \mu_N$	Lösche das N-Bit im μSR	14	0C
$1 \rightarrow \mu_{OVR}$	Setze das OVR-Bit im μSR	17	0F
$0 \rightarrow \mu_{OVR}$	Lösche das OVR-Bit im μSR	16	0E

X steht stellvertretend für $\{Z, C, N, OVR\}$

Tabelle 4.7: Codierungen für wichtige Operationen der Statusregister

	Nicht vorzeichenbehaftete Zahlen		I_{3210}		Zweierkomplementzahlen		I_{3210}	
Relation	Status	Mnemo	oktal	hex	Status	Mnemo	oktal	hex
$A = B$	$Z = 1$	Zero	05	5	$Z = 1$	Zero	05	5
$A \neq B$	$Z = 0$	NotZero	04	4	$Z = 0$	NotZero	04	4
$A \geq B$	$C = 1$	UGTEQ	13	B	$\overline{N \oplus OVR} = 1$	SGTEQ	02	2
$A < B$	$C = 0$	ULT	12	A	$N \oplus OVR = 1$	SLT	03	3
$A > B$	$C \wedge \overline{Z} = 1$	UGT	14	C	$\overline{N \oplus OVR} \wedge \overline{Z} = 1$	SGT	00	0
$A \leq B$	$\overline{C} \vee Z = 1$	ULTEQ	15	D	$(N \oplus OVR) \vee Z = 1$	SLTEQ	01	1

$I_{5,4}$ = 01 (Mnemo: MI) $\longrightarrow$ Abfrage des Mikrostatusregisters
$I_{5,4}$ = 10 (Mnemo: MA) $\longrightarrow$ Abfrage des Maschinenstatusregisters

Tabelle 4.8: Bedingungskodes für den Vergleich zweier Zahlen A und B nach der Operation $A - B$.

Die zur Ausführung der arithmetischen Operationen benötigten **Übertragssignale** werden dem Rechenwerkbaustein Am2901 vom Wortrandlogikbaustein Am2904 über den Übertragsausgang C_0 bereitgestellt. Dieser Ausgang ist mit dem Übertragseingang C_n des niedrigstwertigen Rechenwerkbausteins Am2901 verbunden (siehe Abbildung 4.7). Die Übertragssteuerungslogik des Bausteins Am2904 erlaubt das Belegen des Übertragsausgangs C_0 aus maximal sieben Quellen. Dabei bestimmen die Instruktionsbit I_{11} und I_{12}, welche Quelle den Wert für den C_0–Ausgang liefert. Wenn I_{12} mit 0 belegt ist, dann erscheint am C_0–Ausgang der Wert von I_{11}.
Falls I_{12} mit 1 und I_{11} mit 0 belegt ist, dann liegt am C_0–Ausgang der Wert des C_X–Eingangs der Übertragssteuerung an, der in unserem Beispielrechner allerdings nicht beschaltet ist. Ist I_{12} und I_{11} jeweils mit 1 belegt, dann liegt der Wert des CARRY–Bits des Maschinen– oder Mikrostatusregisters (μ_C, M_C) oder dessen Komplement ($\overline{\mu_C}, \overline{M_C}$) an, was durch die Belegung der Instruktionsbit I_5, I_3, I_2 und I_1 bestimmt wird.
In Tabelle 4.9 ist der Wert des Übertragsausgang C_0 in Abhängigkeit der Instruktionsbit I_{12}, I_{11} und I_5, I_3, I_2, I_1 dargestellt.

Zur Steuerung des Bausteins Am2904 werden insgesamt bis zu 13 Instruktionsbit (I_0 bis I_{12}) sowie neun Enablebit benötigt. Von den 9 Enablebit wurden nur 2, nämlich $\overline{CE_\mu}$ und $\overline{CE_M}$, beschrieben, da alle anderen Bits bei unserem Beispielrechner auf einen festen Pegel gelegt und somit dem Mikroprogrammierer nicht zugänglich sind.

Mnemo	I_{12}	I_{11}	I_5	I_3	I_2	I_1	C_0
CI0	0	0	X	X	X	X	0
CI1	0	1	X	X	X	X	1
CIX	1	0	X	X	X	X	C_X
CIC	1	1	0	0	X	X	μ_C
CIC	1	1	0	1	1	X	μ_C
CIC	1	1	0	1	X	1	μ_C
CIC	1	1	0	1	0	0	$\overline{\mu_C}$
CIC	1	1	1	0	X	X	M_C
CIC	1	1	1	1	1	X	M_C
CIC	1	1	1	1	X	1	M_C
CIC	1	1	1	1	0	0	$\overline{M_C}$

X: Don’t Care

Tabelle 4.9: Übertragssteuerung des Bausteins Am2904

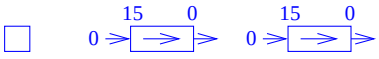
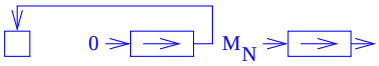
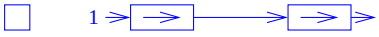
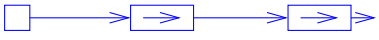
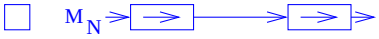
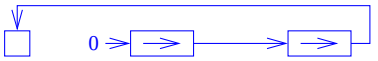
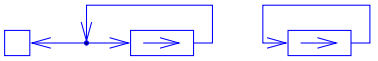
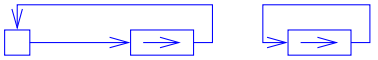
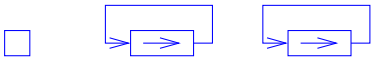
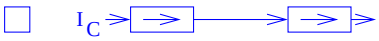
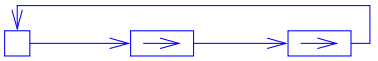
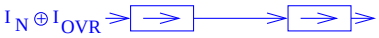
I_{9876}	Mnemo	M_C RAM Q	SIO_0	SIO_n	QIO_0	QIO_n	M_C
0	RSL		Z	0	Z	0	
1	RSH		Z	1	Z	1	
2	RSCONI		Z	0	Z	M_N	SIO_0
3	RSDH		Z	1	Z	SIO_0	
4	RSDC		Z	M_C	Z	SIO_0	
5	RSDN		Z	M_N	Z	SIO_0	
6	RSDL		Z	0	Z	SIO_0	
7	RSDCO		Z	0	Z	SIO_0	QIO_0
8	RSRCO		Z	SIO_0	Z	QIO_0	SIO_0
9	RSRCIO		Z	M_C	Z	QIO_0	SIO_0
A	RSR		Z	SIO_0	Z	QIO_0	
B	RSDIC		Z	I_C	Z	SIO_0	
C	RSDRCI		Z	M_C	Z	SIO_0	QIO_0
D	RSDRCO		Z	QIO_0	Z	SIO_0	QIO_0
E	RSDXOR		Z	$I_N \oplus I_{OVR}$	Z	SIO_0	
F	RSDR		Z	QIO_0	Z	SIO_0	

Abbildung 4.5: Rechts–Schiebeaktionen des Bausteins Am2904 ($I_{10} = 0$)

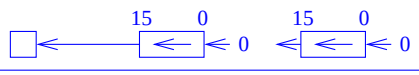
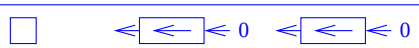
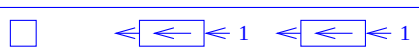
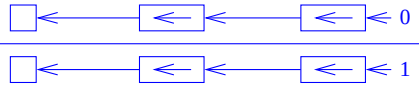
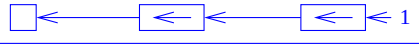
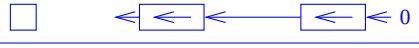
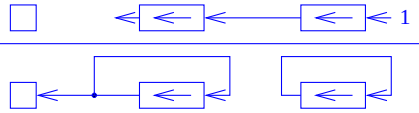
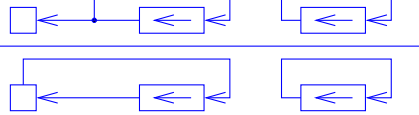
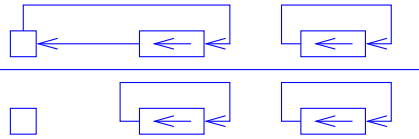
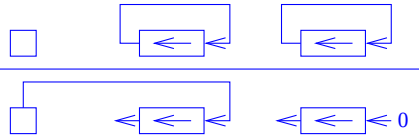
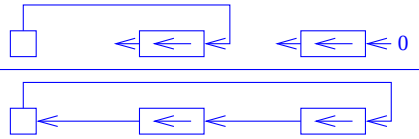
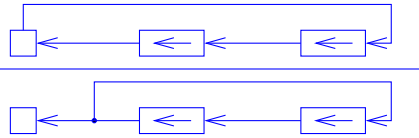
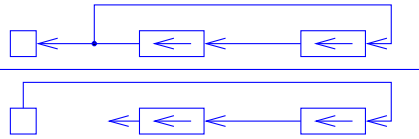
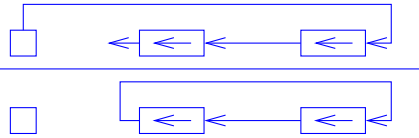
I_{9876}	Mnemo	M_C RAM Q	SIO_0	SIO_n	QIO_0	QIO_n	M_C
0	LSLCO		0	Z	0	Z	SIO_n
1	LSHCO		1	Z	1	Z	SIO_n
2	LSL		0	Z	0	Z	
3	LSH		1	Z	1	Z	
4	LSDLCO		QIO_n	Z	0	Z	SIO_n
5	LSDHCO		QIO_n	Z	1	Z	SIO_n
6	LSDL		QIO_n	Z	0	Z	
7	LSDH		QIO_n	Z	1	Z	
8	LSCRO		SIO_n	Z	QIO_n	Z	SIO_n
9	LSCRIO		M_C	Z	QIO_n	Z	SIO_n
A	LSR		SIO_n	Z	QIO_n	Z	
B	LSLICI		M_C	Z	0	Z	
C	LSDCIO		QIO_n	Z	M_C	Z	SIO_n
D	LSDRCO		QIO_n	Z	SIO_n	Z	SIO_n
E	LSDCI		QIO_n	Z	M_C	Z	
F	LDSR		QIO_n	Z	SIO_n	Z	

Abbildung 4.6: Links–Schiebeaktionen des Bausteins Am2904 ($I_{10} = 1$)

4.1.2.3 Aufbau des Rechenwerks des Beispielrechners

In Abbildung 4.7 ist das Rechenwerk des Beispielrechners mit den Bausteinen Am2901, Am2902 und Am2904 dargestellt, wobei der Übersichtlichkeit wegen eine Reihe von Verbindungen nur angedeutet oder nicht eingezeichnet sind. Insbesondere fehlen alle Enablesignale.

Die vier Rechenwerksbausteine Am2901 bilden ein 16 Bit breites Rechenwerk. Die Y–Ausgänge der Rechenwerkbausteine sind über einen Multiplexer mit dem Adreß– und dem Datenbus verbunden, so daß die über die Y–Ausgänge des Am2901 ausgegebenen Daten auf den Adreß– oder Datenbus gelegt werden können. An den D–Eingängen des Am2901 liegen die über den Datenbus kommenden Daten an.

Die Übertragssteuerung zwischen den vier kaskadenartig aufgebauten Rechenwerkbausteinen übernimmt der Baustein Am2902 (Carry–Look–Ahead–Generator), der mithilfe eines Übertragsermittlungsschaltnetzes die für die Generierung des vorab ermittelten Übertrags (Carry–Look–Ahead) notwendigen Signale „carry propagate“ und „carry generate“ liefert.

Vom Baustein Am2904 führt eine Leitung vom Übertragsausgang C_0 zum Übertragseingang C_n des niedrigstwertigen Rechenwerkbausteins Am2901 und zum C_n–Eingang des Bausteins Am2902. Der Statusausgang OVR des höchstwertigen Am2901–Bausteins ist mit dem Statuseingang I_{OVR} verbunden. Von den Statusausgängen $F3$ und C_{n+4} des höchstwertigen Am2901–Bausteins führen Leitungen direkt zu den entsprechenden Statuseingängen I_N und I_C. Die Statusausgänge $F = 0$ aller vier Rechenwerkbausteine sind durch ein verdrahtetes ODER miteinander verknüpft. Die Nullanzeige liegt am Statuseingang I_Z des Am2904 an.

Der Ausgang CT des Am2904 ist direkt mit dem Testeingang $\overline{CC}$ des Sequencerbausteins Am2910 verbunden. Da dieser Testeingang invertiert ist („active LOW“), sind die zu überprüfenden Bedingungen so zu programmieren, daß am CT–Ausgang eine Null erscheint, wenn die Bedingung erfüllt ist. Dies ist in den Tabellen 4.6 und 4.8 bereits berücksichtigt.

Die Schiebeein– bzw. –ausgänge der Rechenwerkbausteine Am2901 sind untereinander und mit denen des Wortrandlogikbausteins Am2904 verbunden, so daß insgesamt 32 Schiebeaktionen durchgeführt werden können. Die Schieberichtung wird vom Instruktionsbit I_7 des Am2901 (Bit 50 des Mikroinstruktionsworts) bestimmt, da das Instruktionsbit I_{10} des Am2904 fest mit diesem verknüpft ist.

4.1.3 Der Hauptspeicher

Der Hauptspeicher des mikroprogrammierbaren Beispielrechners enthält 64K Worte zu je 16 Bit und ist nur wortweise adressierbar. Ein Zugriff auf den Speicher (lesend oder schreibend) benötigt immer zwei Takte (d.h. zwei Mikroinstruktionen). Im ersten Takt wird die Adresse auf den Adressbus ausgegeben. Die Adresse kann dabei aus dem Befehlszähler oder über den Y-MUX aus der ALU stammen. In diesem Takt ist über das $\overline{MWE}$-Bit im Mikroinstruktionswort (Bit 0) zu programmieren, ob ein Lese- oder ein Schreibzyklus angestoßen werden soll. Bei einem **Lesezyklus** liegen im zweiten Takt die Daten auf dem Datenbus an und können in das Instruktionsregister, den Befehlszähler oder über den K-MUX in die ALU geladen werden. Bei einem **Schreibzyklus** müssen im zweiten Takt die zu schreibenden Daten auf den Datenbus gelegt werden, wobei in diesem Takt auch $\overline{MWE}$ wieder auf 1 zurückzusetzen ist. Ein Pipelining, d.h. die teilweise zeitliche Überlappung von Speicherzugriffen ist in unserem Beispielrechner nicht möglich.

4.1.4 Beschreibung des Mikroinstruktionsformats

Abbildung 4.8 zeigt das Mikroinstruktionsformat des mikroprogrammierbaren Rechners. Nachfolgend sind die einzelnen Felder des Mikroinstruktionsformates beschrieben. Ebenso sind die mnemotechnischen Namen für die jeweils wichtigsten Codierungen (wenn sie nicht bereits in den Tabellen im Text genannt worden sind) angegeben.

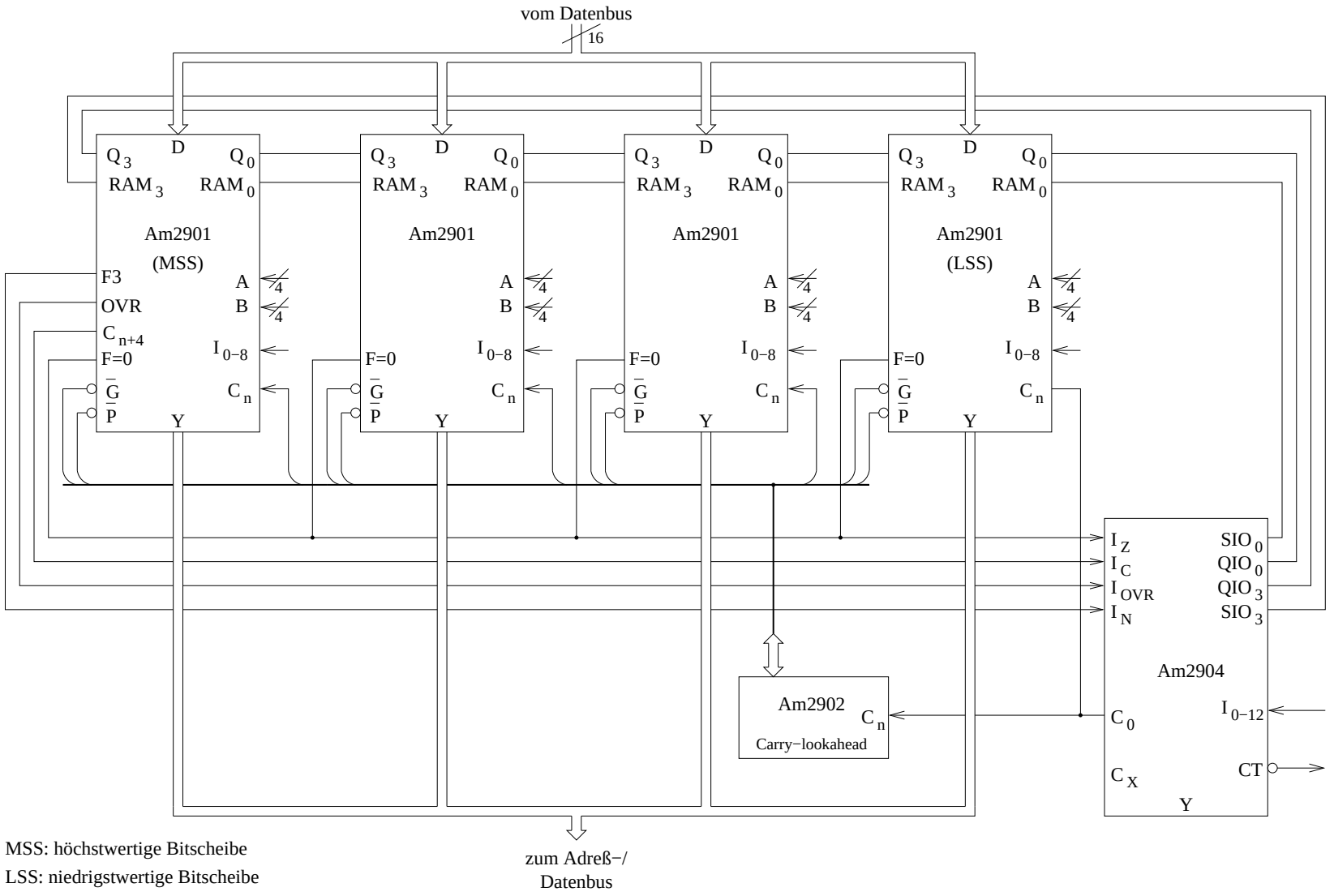


Abbildung 4.7: Aufbau des Rechenwerks des Beispielrechners

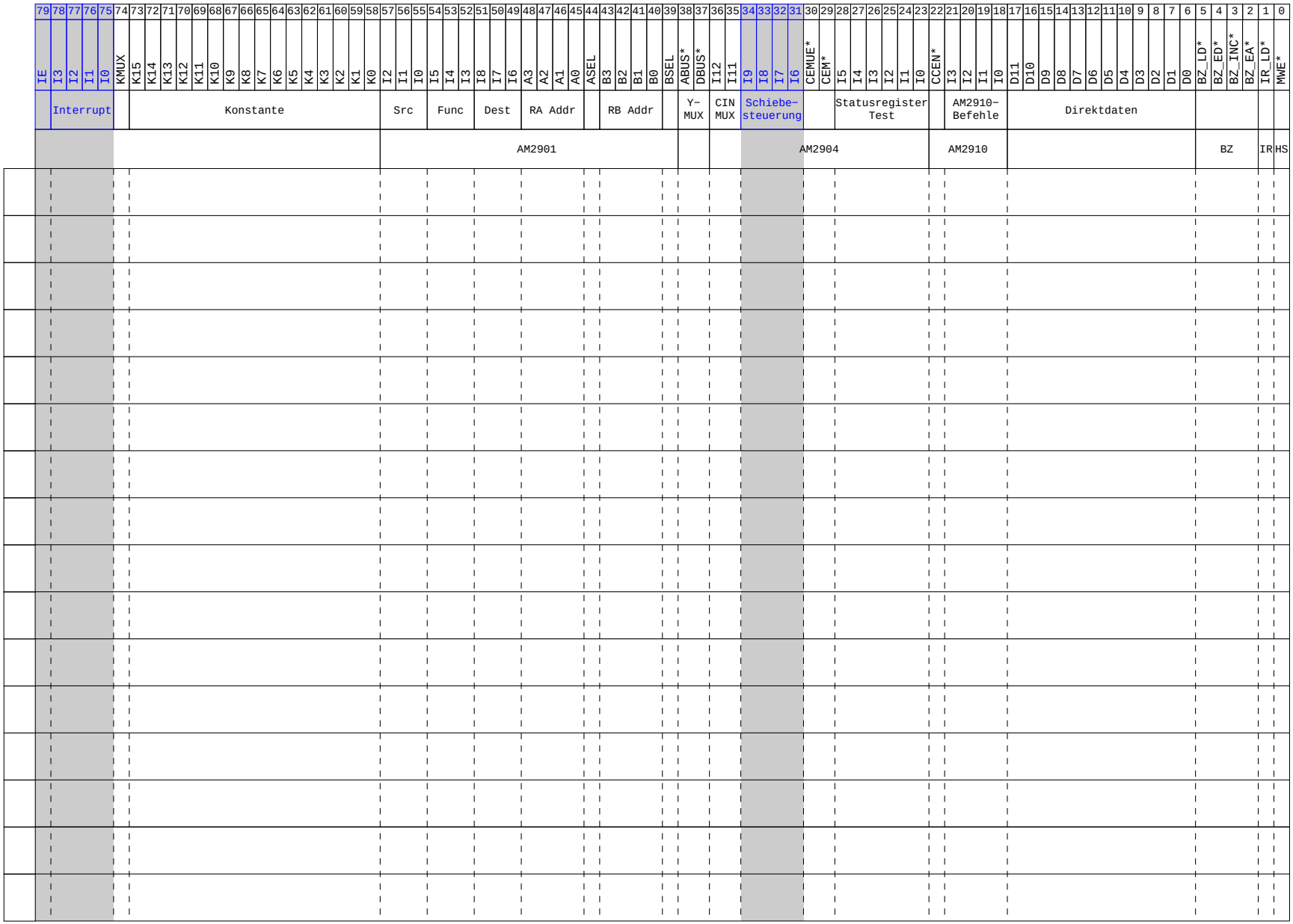


Abbildung 4.8: Mikroinstruktionsformat

MI–Feld	Erläuterung	Vereinbarung mnemotechnischer Namen
MI_0	$\overline{MWE}$ *(Memory–Write–Enable)* In diesem Feld gibt der Mikroprogrammierer an, ob Daten aus dem Hauptspeicher gelesen ($\overline{MWE} = 1$) oder Daten in den Hauptspeicher geschrieben ($\overline{MWE} = 0$) werden sollen. Bei einem Hauptspeicherzugriff ist der gewünschte Wert im ersten Taktzyklus zu programmieren. Im zweiten Taktzyklus ist immer der Wert 1 (R) zu programmieren. Findet kein Hauptspeicherzugriff statt, dann muß eine 1 (R) programmiert werden.	0: W 1: R
MI_1	$\overline{IR\_LD}$ *(Instruktionsregister Laden)* Mit $\overline{IR\_LD} = 0$ wird das Instruktionsregister mit dem gerade auf dem Datenbus liegenden Datum (Befehl) geladen. $\overline{IR\_LD} = 1$ verhindert ein Ändern des Inhalts des Instruktionsregisters.	0: L 1: H
MI_2	$\overline{BZ\_EA}$ *(Befehlszähler Enable Adreßbus)* Mit $\overline{BZ\_EA} = 0$ wird der Inhalt des Befehlszählers auf den Adreßbus gegeben. $\overline{BZ\_EA} = 1$ verhindert, daß der Inhalt auf den Adreßbus ausgegeben wird.	0: E 1: H
MI_3	$\overline{BZ\_INC}$ *(Befehlszähler Inkrement)* Mit $\overline{BZ\_INC} = 0$ wird der Inhalt des Befehlszählers um 1 erhöht. $\overline{BZ\_INC} = 1$ verhindert ein Inkrementieren des Befehlszählers.	0: I 1: H
MI_4	$\overline{BZ\_ED}$ *(Befehlszähler Enable Datenbus)* Mit $\overline{BZ\_ED} = 0$ wird der Inhalt des Befehlszählers auf den Datenbus gegeben. $\overline{BZ\_ED} = 1$ verhindert, daß der Inhalt auf den Datenbus ausgegeben wird.	0: E 1: H
MI_5	$\overline{BZ\_LD}$ *Befehlszähler Laden)* Der Befehlszähler wird mit dem aktuellen Datum auf dem Datenbus geladen, wenn $\overline{BZ\_LD} = 0$ ist. $\overline{BZ\_LD} = 1$ verhindert, daß der Befehlszähler mit einem auf dem Datenbus liegenden Datum geladen wird.	0: L 1: H
MI_6..17	BAR *(Direktdatenfeld)* Über das Direktdatenfeld (BAR) kann der Mikroprogrammierer eine Absolutadresse für den Mikroprogrammspeicher in der aktuellen Mikroinstruktion angeben, um so einen Sprung, eventuell in Abhängigkeit einer Bedingung, zu programmieren. Ebenso kann im Direktdatenfeld ein Wert für das Zählerregister des Am2910 stehen. Ein im Direktdatenfeld stehendes Datum liegt am D–Eingang des Am2910 an.	
MI_18..21	$I_{0..3}$ *(Fortschaltbefehle des Am2910)* In diesem Feld werden die Fortschaltbefehle für den Sequencerbaustein Am2910 kodiert.	siehe Tabelle 4.1
MI_22	$\overline{CCEN}$ *(Condition Code Enable Am2910)* Wenn $\overline{CCEN} = 1$ ist, dann wird der am $\overline{CC}$–Eingang anliegende Wert des Am2910 ignoriert und der Am2910 arbeitet als ob am $\overline{CC}$–Eingang ein Signal mit Wert 0 anliegt (PASS–Bedingung). Falls eine Aktion in Abhängigkeit einer am $\overline{CC}$–Eingang anliegenden Testbedingung durchgeführt werden soll, dann muß $\overline{CCEN} = 0$ sein.	0: C 1: PS

Tabelle 4.10: Beschreibung des Mikroinstruktionsformats

MI–Feld	Erläuterung	Vereinbarung mnemotechnischer Namen
MI_23..28	$I_{0..5}$ *(Instruktionen des Am2904 – Statusregister / Test)* Aus diesem Feld werden die Statusregisteroperationen für das Mikrostatusregister und das Maschinenstatusregister kodiert, die in den Tabellen 4.6 und 4.7 aufgelistet sind. Die Statusregister werden jedoch nur verändert, wenn das entsprechende Enablesignal $\overline{CE_\mu}$ (MI_30) für das Mikrostatusregister bzw. $\overline{CE_M}$ (MI_29) für das Maschinenstatusregister den Wert 0 hat. Über dieses Feld wird auch der Bedingungsmultiplexer des Bausteins gesteuert. Im Feld MI_23..26 (Instruktionsbit $I_{0..3}$) wird der gewünschte Test programmiert und im Feld MI_27..28 (Instruktionsbit $I_{4..5}$) wird angegeben, in welchem Teilwerk des Bausteins die Bedingung geprüft werden soll. Die entsprechenden Instruktionskodes mit den menmotechnischen Namen sind der Tabelle 4.8 zu entnehmen, wobei den Namen für die Bedingungskodes MI oder MA (für Mikro- bzw. Maschinenstatusregister) voranzustellen ist.	Bedingungscodes: siehe Tabelle 4.8
MI_29	$\overline{CE_M}$ *(Enablebit für das MSR)* Der Inhalt des Maschinenstatusregisters des Am2904 kann nur dann verändert werden, wenn das Enablebit $\overline{CE_M} = 0$ ist.	0: L 1: H
MI_30	$\overline{CE_\mu}$ *(Enablebit für das μSR)* Der Inhalt des Mikrostatusregisters des Am2904 kann nur dann verändert werden, wenn das Enablebit $\overline{CE_\mu} = 0$ ist.	0: L 1: H
MI_31..34	$I_{6..9}$ *(Instruktionsbit des Am2904 – Schiebesteuerung)* Die Schiebesteuerung des Am2904 erlaubt die in Tabelle 4.5 und Tabelle 4.6 aufgeführten Schiebeaktionen. In Verbindung mit der ALU–Zielsteuerung des Am2901 im Feld MI_49..51 wird eine der 32 möglichen Schiebeoperationen programmiert, damit die gewünschte Schiebeaktion im Rechenwerk durchgeführt werden kann.	siehe Tabelle 4.5 und Tabelle 4.6
MI_35..36	$I_{11..12}$ *(Instruktionsbit des Am2904 – Übertragssteuerung)* Die Übertragssteuerung erlaubt das Belegen des Übertragsausgangs C_0 aus maximal sieben Quellen (siehe Abschnitt 4.1.2.2).	siehe Tabelle 4.9
MI_37	$\overline{DBUS}$ *(Datenbus Select)* Die Daten, die über den Y–Ausgang des Am2901 ausgegeben werden, kommen auf den Datenbus.	0: DB 1: H
MI_38	$\overline{ABUS}$ *(Adreßbus Select)* Die Daten, die über den Y–Ausgang des Am2901 ausgegeben werden, kommen auf den Adreßbus.	0: AB 1: H

Tabelle 4.11: Beschreibung des Mikroinstruktionsformats

MI–Feld	Erläuterung	Vereinbarung mnemotechnischer Namen
MI_39	$BSEL$ *(RB ADDR Select)* Bei $BSEL = 1$ wird die Adresse für das Register im Register–B–Adreßfeld des Mikroinstruktionsregisters angegeben. Bei $BSEL = 0$ wird die Adresse für das Register im Register–B–Adreßfeld des Instruktionsregisters spezifiziert.	0: IR 1:MR
MI_40..43	*RB ADDR (Register B Adresse)* Der Inhalt des in diesem Feld adressierten Registers der internen Registerdatei des Am2901 wird über den B–Ausgang ausgegeben. Das Zielregister für zu schreibende Daten wird ebenfalls in diesem Feld adressiert.	0_{hex}: r0 1_{hex}: r1 ⋮ F_{hex}: r15
MI_44	$ASEL$ *(RA ADDR Select)* Bei $ASEL = 1$ wird die Adresse für das Register im Register–A–Adreßfeld des Mikroinstruktionsregisters angegegben. Bei $ASEL = 0$ wird die Adresse für das Register im Register–A–Adreßfeld des Instruktionsregisters spezifiziert.	0: IR 1:MR
MI_45..48	*RA ADDR (Register A Adresse)* Der Inhalt des in diesem Feld adressierten Registers der internen Registerdatei des Am2901 wird über den A–Ausgang ausgegeben.	0_{hex}: r0 1_{hex}: r1 ⋮ F_{hex}: r15
MI_49..51	$I_{6..8}$ *(Instruktionsbit des Am2901 – ALU Zielsteuerung)* Die Steuerung, wohin die Ergebnisse der ALU des Am2901 geschrieben und welche Schiebeaktionen ausgeführt werden, erfolgt über dieses Feld. Die acht möglichen Kombinationen sind in Tabelle 4.4 dargestellt. Bei Schiebeaktionen ist zusätzlich im Feld MI_31..34 die gewünschte Schiebeaktion anzugeben.	siehe Tabelle 4.4
MI_52..54	$I_{3..5}$ *(Instruktionsbit des Am2901 – ALU Funktionen)* Die ALU kann die in Tabelle 4.3 angegebenen drei arithmetischen und fünf logischen Operationen ausführen, wobei eine der acht möglichen ALU–Funktionen in diesem Feld zu kodieren ist.	siehe Tabelle 4.3
MI_55..57	$I_{0..2}$ *(Instruktionsbit des Am2901 – ALU Quelloperanden)* Die Operanden für die ALU des Am2901 können seiner internen Registerdatei, seinem Q–Register und dem unidirektionalen D–Eingang entstammen. In diesem Feld werden die Operanden für den ALU–R– und den ALU–S–Eingang bestimmt. Die acht möglichen Kombinationen sind in Tabelle 4.2 angegeben.	siehe Tabelle 4.2
MI_58..73	$K_{0..15}$ Konstantenfeld	
MI_74	*KMUX* Auswahl der Quelle für den D-Eingang des Rechenwerks. Die Daten können entweder vom Konstantenfeld des Mikroinstruktionsregisters kommen (*KMUX* = K) oder vom Datenbus (*KMUX* = D).	0: K 1: D
MI_75..78	$I_{0..3}$ *(Interruptsteuerung)* Die Interruptsteuerung ist der Einfachheit wegen nicht beschrieben.	
MI_79	IE *(Interrupt Enable)* Die Interruptsteuerung ist der Einfachheit wegen nicht beschrieben.	

Tabelle 4.12: Beschreibung des Mikroinstruktionsformats