Redcode-Maschine

Die Redcode-Maschine wurde zuerst von A. K. Dewdney in der Ausgabe vom Mai 1984 in *Scientific American* vorgestellt. Die folgende Spezifikation orientiert sich daran, klärt aber auch einige Punkte, die im originalen Artikel nicht genau definiert sind. Es gab später zahlreiche Erweiterungen der Architektur, auf die hier jedoch nicht weiter eingegangen wird.

Eine Redcode-Maschine besteht aus 8000 Speicherzellen, die von 0 bis 7999 adressiert werden. Es wird nur ganzzahlig gerechnet mit Werten von 0 bis 7999, jeweils modulo 8000. Entsprechend ergibt beispielsweise 7999 + 1 die 0. Wenn von der 0 die Zahl 10 abgezogen wird, dann ist das Resultat 7990.

Jede Speicherzelle besteht aus einer Instruktion (auch *opcode* genannt) und zwei Operanden. Jeder Operand besteht wiederum aus einem Adressierungsmodus (kurz *am*) und einem ganzzahligen Wert aus dem Bereich 0 bis 7999 (*value*):

Ein Operand gibt entweder einen Wert direkt an (Adressierungsmodus *direct*) oder adressiert eine Speicherzelle, von der der gewünschte Wert zu holen ist. Dabei erfolgt die Adressierung von Speicherzellen immer relativ zum Programmzähler (*program counter*) bzw. bei der indirekten Adressierung auch relativ zu der Speicherzelle. Im folgenden bezeichne *mem*[*i*] die *i*-te Speicherzelle und *pc* den Programmzähler. Die vier unterstützten Adressierungsmodi werden dann wie folgt interpretiert:

Modus	Symbol	Wert	Interpretation
immediate	#	0	*value*
direct		1	*mem*[*pc* + *value*]
indirect	@	2	*mem*[*pc* + *value* + *mem*[*pc* + *value*]]
decrement	<	3	*mem*[*pc* + *value* + --*mem*[*pc* + *value*]]

Zu beachten ist hier, dass die Adressierungsmodi *direct* bis *decrement* jeweils eine gesamte Speicherzelle adressieren. Bei *immediate* wird implizit aus dem gegebenen Wert temporär eine entsprechende Speicherzelle mit der *DAT*-Instruktion (siehe weiter unten) konstruiert. Abgesehen von dem Fall *immediate* liegt immer eine adressierbare Speicherzelle vor, die auch von einer Instruktion verändert werden kann. Der Effekt eines Versuchs, einen *immediate*-Wert zu verändern, ist undefiniert und dies sollte normalerweise auch nicht vorkommen.

Das jeweils genannte Symbol wird bei der Assemblernotation verwendet und dem Operanden vorangestellt. Wenn kein solches Symbol verwendet wird, wird der Adressierungsmodus *direct* benutzt.

Im Objektformat wird ein Operand durch zwei ganze Zahlen repräsentiert, wovon die erste den Adressierungsmodus darstellt und die zweite den Wert. Der Wert ist entsprechend der Modulo-Arithmetik in einen Wert zwischen 0 und 7999 abzubilden. Auch negative Zahlen sind erlaubt, aus -2 wird dann beispielsweise 7998.

Eine Speicherzelle besteht dann aus einer Instruktion (*opcode*, repräsentiert durch eine ganze Zahl von 0 bis 8) und den zwei Operanden *a* und *b*:

opcode	a_{am}	a_{value}	b_{am}	b_{value}

Der Programmzähler hat einen Wert zwischen 0 und 7999 und verweist immer auf die nächste auszuführende Instruktion. Der Programmzähler wird bei jeder Instruktion implizit (per Modulo-Arithmetik) um 1 erhöht, sofern nichts anderes angegeben ist. Im einzelnen gibt es folgende Instruktionen:

DAT			#	*value*

Assemblernotation: DAT #*value*
Objektformat: 0 *value*

Eine *DAT*-Instruktion dient dazu, Daten zu speichern und als solche zu kennzeichnen. Der Versuch, eine *DAT*-Instruktion auszuführen, führt zum Absturz des entsprechenden Prozesses (womit das entsprechende Programm in einem *Core Wars*-Wettbewerb verliert). Wenn der numerische Wert einer Speicherzelle extrahiert wird, dann wird immer der zweite Operand genommen und der Adressierungsmodus ignoriert (bzw. *immediate* vorausgesetzt). Sollte eine andere Instruktion wie eine *DAT*-Instruktion behandelt werden, weil sie durch einen Operanden adressiert wird, dann ist das zulässig und in diesem Fall wird ebenfalls der Wert aus dem zweiten Operanden verwendet.

MOV	src_{am}	src_{value}	$dest_{am}$	$dest_{value}$

Assemblernotation: MOV *src*,*dest*
Objektformat: 1 src_{am} src_{value} $dest_{am}$ $dest_{value}$

Die durch *src* adressierte Speicherzelle wird kopiert zu der durch *dest* adressierten Speicherzelle. Eine solche Kopie schließt die Instruktion und beide Operanden mit ein und dient somit dazu, sowohl Daten als auch Programme zu kopieren. Wenn *src* den Adressierungmodus *immediate* hat, entspricht dies einer *DAT*-Speicherzelle mit dem gegebenen Wert (im zweiten Operanden). Wenn *dest* einen *immediate*-Adressierungsmodus hat, ist der Effekt undefiniert.

ADD	src_{am}	src_{value}	$dest_{am}$	$dest_{value}$

Assemblernotation: ADD *src*,*dest*
Objektformat: 2 src_{am} src_{value} $dest_{am}$ $dest_{value}$

Der Wert des Operanden *src* wird zu *dest* addiert. Dies entspricht also *dest* += *src*. Es gilt die beschriebene Modulo-Arithmetik.

SUB	src_{am}	src_{value}	$dest_{am}$	$dest_{value}$

Assemblernotation: SUB *src*,*dest*
Objektformat: 3 src_{am} src_{value} $dest_{am}$ $dest_{value}$

Der Wert des Operanden *src* wird von *dest* subtrahiert. Dies entspricht also *dest* -= *src*. Es gilt die beschriebene Modulo-Arithmetik.

JMP	$dest_{am}$	$dest_{value}$		

Assemblernotation: JMP *dest*
Objektformat: 4 $dest_{am}$ $dest_{value}$

JMP ist ein unbedingter Sprung, der den Programmzähler auf die durch *dest* adressierte Speicherzelle setzt. Wenn *dest* den Adressierungsmodus *immediate* hat, ist der Effekt undefiniert.

JMZ	$dest_{am}$	$dest_{value}$	src_{am}	src_{value}

Assemblernotation: JMZ *dest*,*src*
Objektformat: 5 $dest_{am}$ $dest_{value}$ src_{am} src_{value}

JMZ ist eine bedingte Sprunganweisung, die den Programmzähler genau dann auf die durch *dest* adressierte Speicherzelle setzt, wenn der Wert von *src* 0 ist. Wenn *dest* den Adressierungsmodus *immediate* hat, ist der Effekt undefiniert.

JMG	$dest_{am}$	$dest_{value}$	src_{am}	src_{value}

Assemblernotation: JMG *dest*,*src*
Objektformat: 6 $dest_{am}$ $dest_{value}$ src_{am} src_{value}

JMG ist eine bedingte Sprunganweisung, die den Programmzähler genau dann auf die durch *dest* adressierte Speicherzelle setzt, wenn der Wert von *src* größer als 0 ist. (Zu beachten ist hier, dass wegen der beschriebenen Modulo-Arithmetik keine negativen Werte vorkommen.) Wenn *dest* den Adressierungsmodus *immediate* hat, ist der Effekt undefiniert.

DJN	$dest_{am}$	$dest_{value}$	$counter_{am}$	$counter_{value}$

Assemblernotation: DJN *dest,counter*
Objektformat: 7 $dest_{am}$ $dest_{value}$ $counter_{am}$ $counter_{value}$

DJN dekrementiert die durch *counter* adressierte Speicherzelle und setzt genau dann den Programmzähler auf die durch *dest* adressierte Speicherzähler, wenn der dekrementierte Wert ungleich 0 ist. Wenn *dest* den Adressierungsmodus *immediate* hat, ist der Effekt undefiniert.

CMP	a_{am}	a_{value}	b_{am}	b_{value}

Assemblernotation: CMP *a,b*
Objektformat: 8 a_{am} a_{value} b_{am} b_{value}

CMP vergleicht *a* und *b*. Haben beide den gleichen Wert, dann wird der Programmzähler um zwei erhöht, sonst nur um eins.

Beispiel

Folgendes ist das *Dwarf*-Beispiel, das von Dewdney in seinem Artikel vorgestellt worden ist, in Assembler-Notation:

```
ptr     DAT   #0
start   ADD   #5,ptr
        MOV   #0,@ptr
        JMP   start
```

Der *Dwarf* beginnt seine Ausführung mit der ersten Nicht-Dateninstruktion, also mit dem *ADD*-Befehl. Hierbei erhöht er die zu Beginn auf 0 initialisierte *ptr*-Variable um 5. Mit dem darauffolgenden *MOV*-Befehl wird eine Daten-0 (also entsprechende *DAT*-Speicherzelle mit dem Wert 0) dorthin kopiert, wo *ptr* hinzeigt. Wenn *ptr* den Wert 5 hat, wird dies relativ zur Position von *ptr* interpretiert, d.h. die erste 0 wird auf die übernächste Speicherzelle hinter der *JMP*-Instruktion geschrieben. Im weiteren Verlauf bombardiert der *Dwarf* den gesamten Speicher mit diesen *DAT*-Zellen in der Hoffnung, dass der Gegner eine davon ausführen möge und die Speichergröße durch 5 teilbar ist (was bei 8000 natürlich der Fall ist).

Wenn dieses Programm vom Assembler in das Objektformat übersetzt wird, entsteht dabei folgende Zahlensequenz:

```
0 0
2 0 5 1 -1
1 0 0 2 -2
4 1 -2
```

Zu beachten ist hier, dass die symbolischen Namen wie *ptr* oder *start* jeweils durch Adressierungen ersetzt worden sind, die relativ zur Position der jeweiligen Instruktion zu interpretieren sind. Im Objektformat kommen hier auch negative Zahlen vor, da auf diese Weise noch nicht festgelegt werden muss, wie groß der Speicher im ausführenden System sein muss.

Wenn diese Zahlensequenz geladen wird, sollten die negativen Zahlen dann entsprechend der Modulo-Arithmetik konvertiert werden, so dass dann folgende Speicherzellensequenz entsteht:

DAT			#	0
ADD	#	5		7999
MOV	#	0	@	7998
JMP		7998		