Elementare logische Operationen

Modifizieren, Testen,Vergleichen
In diesem Abschnitt wollen wir zeigen, wie man mit den elementaren logischen Verknüpfungen wichtige Anwendungsaufgaben lösen kann (Abbildung 1.1).

Setzen von Bits
Um bestimmte Bits (in einem Register, Maschinenwort usw.) zu setzen, ist ein OR-Befehl notwendig, wobei der zweite Operand an allen zu setzenden Stellen Einsen enthält und sonst Nullen.

Löschen von Bits
Um bestimmte Bits (in einem Register, Maschinenwort usw.) zu löschen, ist ein AND-Befehl notwendig, wobei der zweite Operand an allen zu löschenden Stellen Nullen enthält und sonst Einsen.

Wechseln von Bits
Um bestimmte Bits (in einem Register, Maschinenwort usw.) in ihrem Wert zu ändern (von 0 nach 1 und umgekehrt), ist ein XOR-Befehl notwendig, wobei der zweite Operand an allen zu ändernden Stellen Einsen enthält und sonst Nullen.

Entnehmen von Bits
Um zusammenhängende Bitfelder oder bestimmte einzelne Bits aus einem Register, Maschinenwort usw. zu entnehmen (andere Redeweisen: ausblenden, maskieren), ist ein AND-Befehl notwendig, wobei der zweite Operand an allen ausgewählten Stellen Einsen enthält und sonst Nullen. (Dieser Operand heißt üblicherweise Maskenoperand.)

Einfügen von Bits
Um zusammenhängende Bitfelder oder bestimmte einzelne Bits in ein Register, Maschinenwort usw. einzufügen, ist auf das Register, Maschinenwort usw. zunächst ein AND-Befehl anzuwenden, dessen zweiter Operand an allen betreffenden Stellen Nullen enthält und sonst Einsen (Maskenoperand). Nachfolgend ist ein OR-Befehl anzuwenden, dessen zweiter Operand an den betreffenden Stellen die einzufügenden Werte enthält und sonst Nullen.

Testen auf gelöschte bzw. gesetzte Bits
Um zu prüfen, ob bestimmte Bits (in einem Register, Maschinenwort usw.) alle gelöscht sind oder ob wenigstens eines dieser Bits gesetzt ist, braucht man einen AND-Befehl, dessen zweiter Operand an allen betreffenden Stellen Einsen enthält und sonst Nullen. Sind alle so geprüften Bits gelöscht, ist das Zero-Flagbit gesetzt. Ist wenigstens eines der geprüften Bits gesetzt, ist das Flagbit gelöscht. Diese Funktion steht beispielsweise in der x86-Architektur als TEST-Befehl zur Verfügung (das ist ein AND-Befehl, der kein Ergebnis zurückschreibt).

Alles löschen
Es gibt mehrere Möglichkeiten (z. B. Laden eines Direktwertes Null oder AND mit Null). Gelegentlich besonders effektiv: ein XOR mit sich selbst, z. B. <R1> := <R1> XOR <R1> (der Befehl hat keinen Direktwert und entspricht oft dem jeweils kürzesten Befehlsformat).

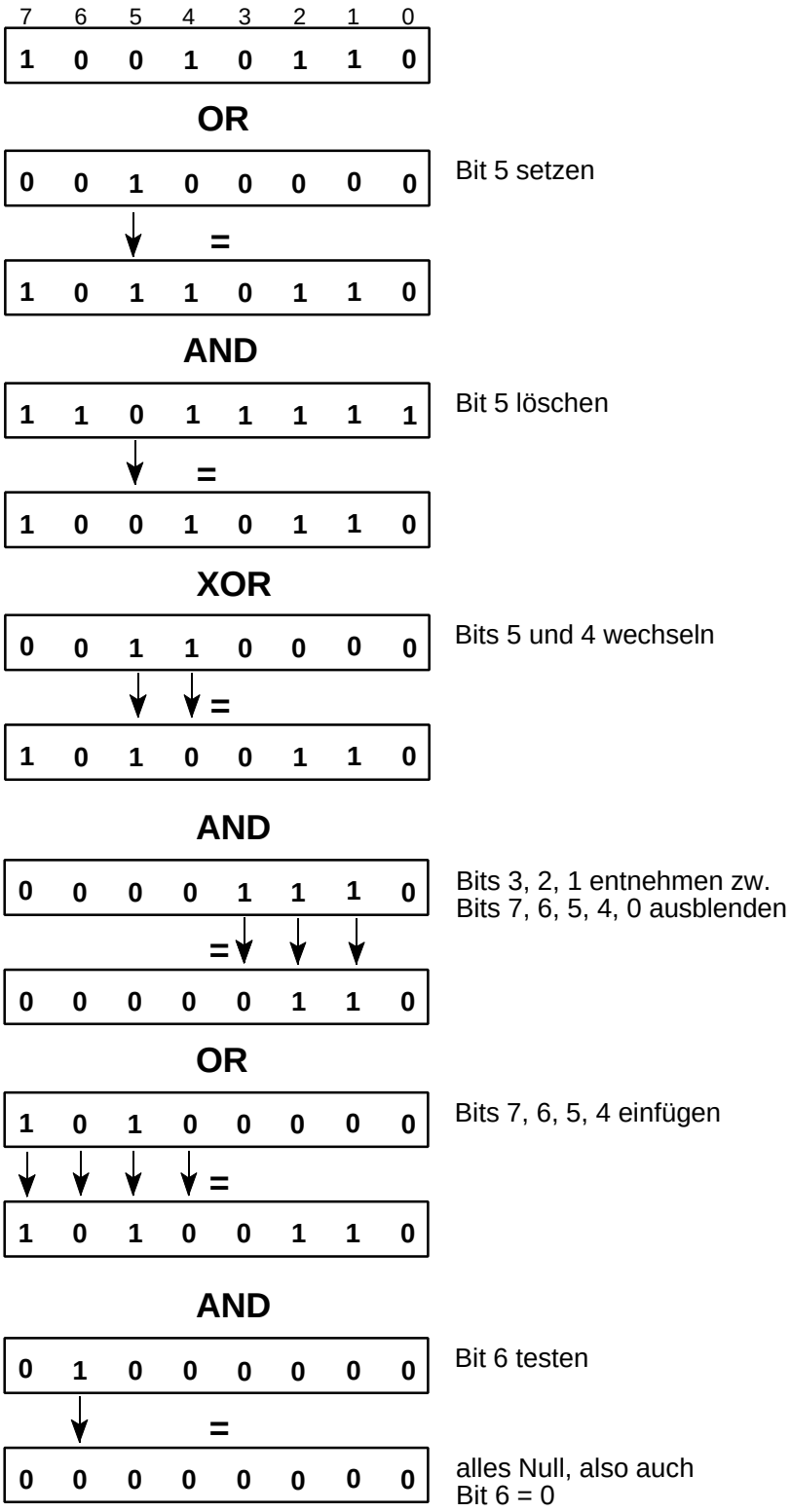

Abbildung 1.1 Elementare Bitoperationen anhand von Beispielen

Vergleichen
Um zu prüfen, ob zwei Bitmuster einander gleich sind oder nicht, kann man einen XOR-Befehl verwenden. Bei Gleichheit ist das Zero-Flag gesetzt, bei Ungleichheit gelöscht. XOR-Befehle ohne Zurückschreiben des Ergebnisses gibt es in manchen Architekturen als "logische" Vergleichsbefehle (Compare Logical). Will man nur ausgewählte Bits miteinander vergleichen, so müssen die anderen Bits ausgeblendet werden, und zwar entweder durch OR mit Einsen oder durch AND mit Nullen (man muß nur beide Operanden auf gleiche Weise maskieren).

Ausgewählte Gotchas:

1. Vergleichen. Manche als “logischer Vergleich” (Logical Compare) bezeichnete Befehle führen keine XOR- oder XNOR-Verknüpfungen aus, sondern nur UND-Verknüpfungen. Es sind also keine echten Vergleichsbefehle (mit Vergleichsaussage *gleich/ungleich* als Ergebnis). Sie testen vielmehr nur gesetzte Bits (Aussage: *im 1. Operanden ist keines der im 2. Operanden gesetzten Bits gesetzt/es ist wenigstens eines der besagten Bits gesetzt*). Beispiel TEST (x86/IA-32).

2. Negation (Einerkomplement) und Zweierkomplement. Beides ist nicht das gleiche. Aufpassen - die Assembler-Mnemonics sind nicht immer so selbsterklärend, wie man es naiverweise erwartet. 1. Beispiel: x86/IA-32: Zweierkomplement = NEG, Negation = NOT. 2. Beispiel: Atmel AVR; Zweierkomplement = NEG, Negation (Einerkomplement) = COM.

Höherentwickelte Bitoperationen
Diese Operationen betreffen im allgemeinen Fall variabel lange Bitketten (Bitstrings) im Speicher. Manche Architekturen unterstützen nur Maschinenworte in Registern.

Einzelbitbefehle:

- Bit testen,
- Bit löschen,
- Bit setzen,
- Bit wechseln (0 nach 1, 1 nach 0).

Wichtig: wie wird die Bitadresse angegeben?

a) die einfachste Lösung: Bitadresse ist lediglich Festwert im Befehl. Typisch für Mikrocontroller. Nur für einfache E-A-Vorgänge geeignet. Weitergehenden Nutzung ggf. in Form selbstmodifizierender Programme (Programm fügt berechnete Bitadresse in den Befehl ein). Funkktioniert aber nur bei Programmausführung aus RAM.

b) vielseitiger: Bitadresse ist zweiter Operand (in Register oder Speicher).

Auffinden der niedrigstwertigen Eins (First Occurrence)

1. Variante: Es wird eine Bitkette zurückgeliefert, in der nur die niedrigstwertige Eins gesetzt ist.

2. Variante: Es wird eine Binärzahl zurückgeliefert, die die Position (Bitadresse) der niedrigstwertigen Eins angibt.

Auffinden der höchstwertigen Eins (Last Occurrence)

1. Variante: Es wird eine Bitkette zurückgeliefert, in der nur die höchstwertige Eins gesetzt ist.

2. Variante: Es wird eine Binärzahl zurückgeliefert, die die Position (Bitadresse) der höchstwertigen Eins angibt.

Ermittlung der Anzahl der Einsen (Quersumme, Number of Occurrences, Population Count)
Es wird eine Binärzahl zurückgeliefert, die die Anzahl der Einsen in der Bitkette angibt.

Operationen mit variablen Bitfeldern
Der höchste Grad an Flexibilität ist erreicht, wenn man nicht nur Bytes, Worte usw. verarbeiten kann, sondern wenn es möglich ist, beliebige Bitfelder zu transportieren und miteinander zu verknüpfen. Abbildung 1.2 veranschaulicht solche Bitfelder und die beiden Elementaroperationen, die wir hier als Entnehmen (EXTRACT) und Einfügen (DEPOSIT) bezeichnen wollen.

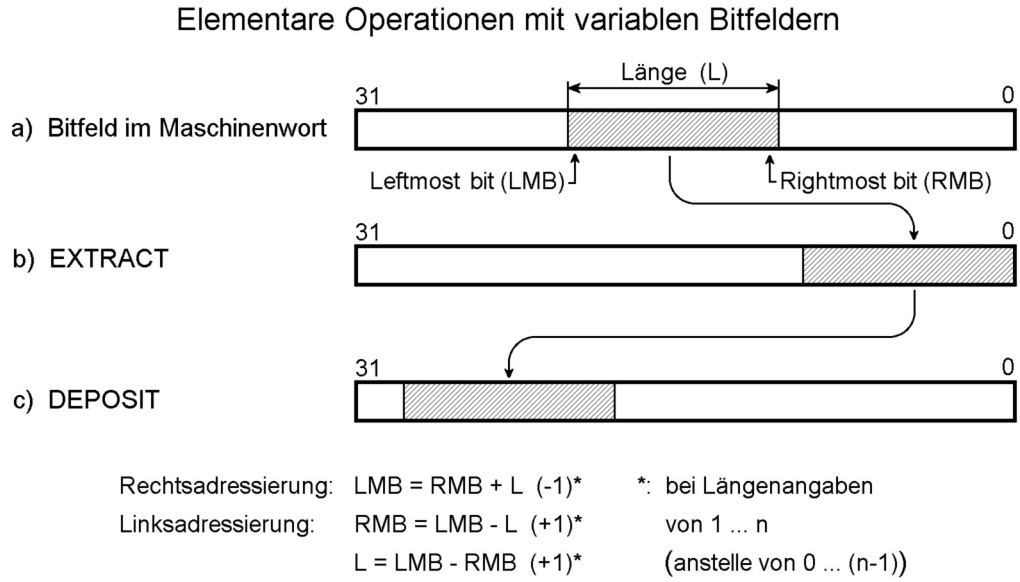

Abbildung 1.2 Elementare Operationen mit variablen Bitfeldern

Welche Angaben bestimmen ein Bitfeld?
Ein Bitfeld ist eindeutig bestimmt (1) durch die Bitadresse (den Index) des ersten Bits und (2) durch die Länge (Anzahl der Bits) oder - alternativ - durch den Index des letzten Bits.

Entnehmen (EXTRACT)
Das Bitfeld wird aus dem Operanden entnommen und rechtsbündig im Rahmen der Verarbeitungsbreite bereitgestellt. Die verbleibenden Bits werden mit Nullen belegt.

Einfügen (DEPOSIT)
Ein rechtsbündig bereitstehendes Bitfeld wird in den Operanden eingefügt. Die verbleibenden Bits des Operanden werden nicht verändert.

Maskierte Logikoperationen
Viele nützliche Operationen erfordern eine "Maskierung". Damit sollen nur bestimmte Bits bei der Verarbeitung berücksichtigt, die anderen hingegen davon ausgeschlossen werden. Diese Maskierung erfordert oft zusätzliche Befehle, z. B. ein AND mit dem Maskenoperanden, um die auszuschließenden Bitpositionen auf Null zu setzen. Es gibt Anwendungsfälle, in denen solche Operationen leistungsbestimmend sind. Deshalb hat man maskierte Verknüpfungen gelegentlich (in Mikrocontrollern) als Maschinenbefehle vorgesehen. Diese Befehle haben einen zusätzlichen (dritten) Masken-Operanden bzw. sie beziehen sich implizit auf ein Maskenregister.

Verschieben und Rotieren
Beim *Rechtsverschieben* um ein Bit gelangt Bit 1 in Bitposition 0, Bit 2 in Bitposition 1 usw. Entsprechend gelangt beim *Linksverschieben* um ein Bit Bit 6 in Bitposition 7, Bit 5 in Bitposition Bit 6 usw. Sinngemäß werden die Daten bewegt, wenn eine Verschiebung um n Bits auszuführen ist (so kommt beim Linksverschieben um zwei Bits Bit 5 in Bitposition 7, Bit 4 in Bitposition 6 usw.).

Betrachten wir das Innere einer Datenstruktur, so bereitet es uns keine Schwierigkeiten, die Bewegung der Bits beim Verschieben zu verstehen. Was geschieht aber links und rechts, also gewissermaßen an den Rändern? - Dies ist der Punkt, in dem sich Verschiebe- und Rotationsabläufe unterscheiden (Abbildung 1.3).

Beim *Rotieren* werden die hinausgeschobenen Bits am jeweils anderen Ende wieder zurückgeführt (Wrap Around). Rotieren ist also ein zyklisches Verschieben in den Grenzen der jeweiligen Operandenlänge. Wird ein Byte um ein Bit nach links rotiert, gelangt Bit 7 nach Bit 0, bei Rechtsrotation entsprechend Bit 0 nach Bit 7.

Hingegen gehen beim eigentlichen *Verschieben* die hinausgeschobenen Bits verloren. Die Bitpositionen, die am anderen Ende frei werden, werden meist mit Nullen aufgefüllt. In manchen Architekturen (beispielsweise IA-32) gibt es zusätzlich Verschiebebefehle mit Erweiterung. Diese haben einen zweiten Operanden, aus dem die besagten freigewordenen Bitpositionen aufgefüllt werden.

In vielen Architekturen werden bestimmte Flagbits in Verschiebe- bzw. Rotationsabläufe einbezogen (Abbildungen 1.4, 1.5). Zumeist ist das jeweils zuletzt hinausgeschobene oder umlaufende Bit in einem Flagbit direkt abfragbar. Ist das Flagbit am Rotieren beteiligt, so laufen die

herausgeschobenen Bits nicht unmittelbar am anderen Ende des Operanden wieder ein, sondern gelangen zunächst in das Flagbit und von dort aus in das eigentliche Ergebnis zurück.

Arithmetisches Rechtsschieben
Das ist ein Rechtsschieben mit Vorzeichenerweiterung. Das höchstwertige Bit (= Vorzeichen) bleibt erhalten und wird in die beim Schieben freiwerdenden Bitstellen übernommen.

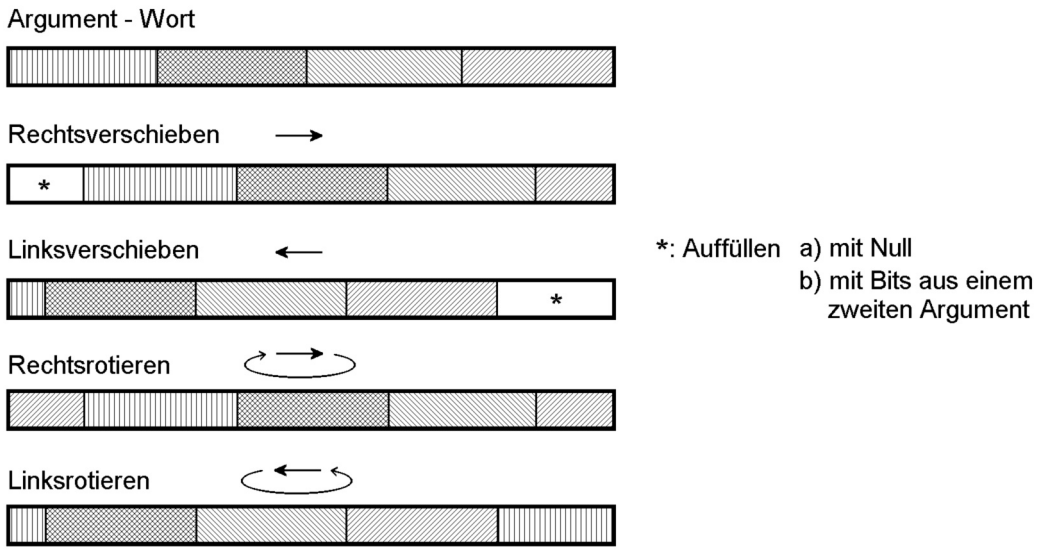

Abbildung 1.3 Verschieben und Rotieren

Schieben und Arithmetik
Das Linksschieben um n Bits entspricht einem Multiplizieren mit 2^n, das Rechtsschieben einem Dividieren durch 2^n.

Achtung:
Bei Zweierkomplementarithmetik funktioniert das nur, wenn mit vorzeichenlosen*) oder mit positiven Zahlen gerechnet wird (z. B. bei der Adreßrechnung).

*): zum Dividieren vorzeichenloser Zahlen ist die "logische" Rechtsverschiebung zu verwenden (die die freiwerdenden Stellen mit Nullen auffüllt).

Beim Rechnen mit negativen Zahlen

- kann beim Linksschieben das Vorzeichen verlorengehen (manche Maschinen*) haben Befehle, die das Erkennen dieses Sonderfalls unterstützen),
- kann es sein, daß beim Dividieren gerundet wird (SHIFT ergibt anderes Ergebnis als ganzzahlige Division).

*): so auch IA-32; vgl. die folgenden Abbildungen 1.4 und 1.5

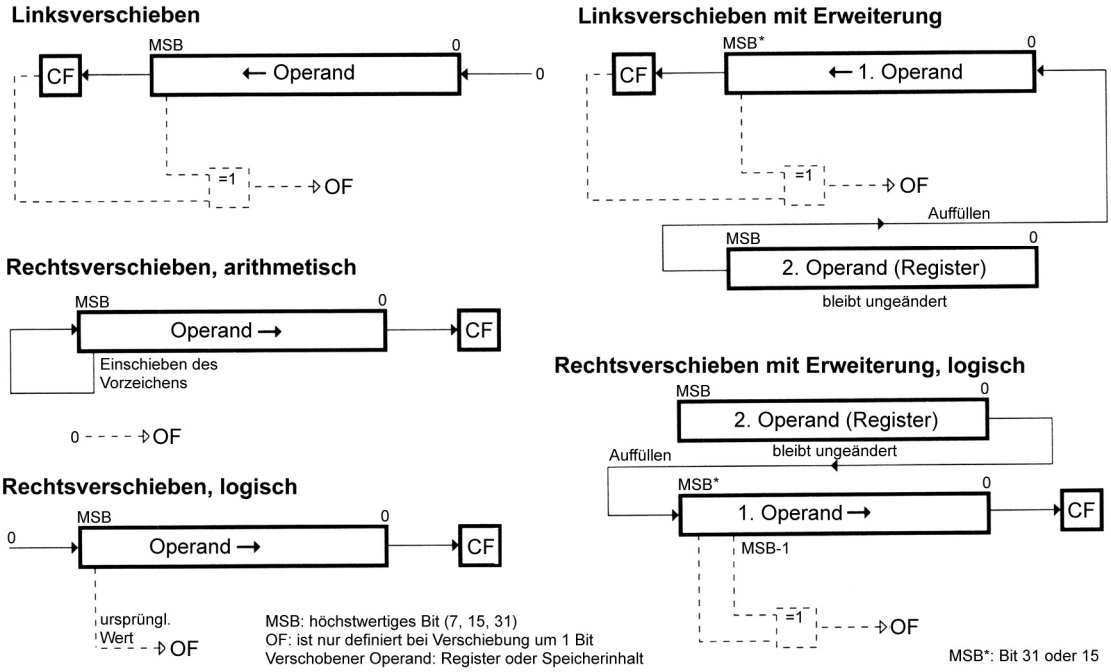

Abbildung 1.4 Verschiebeabläufe

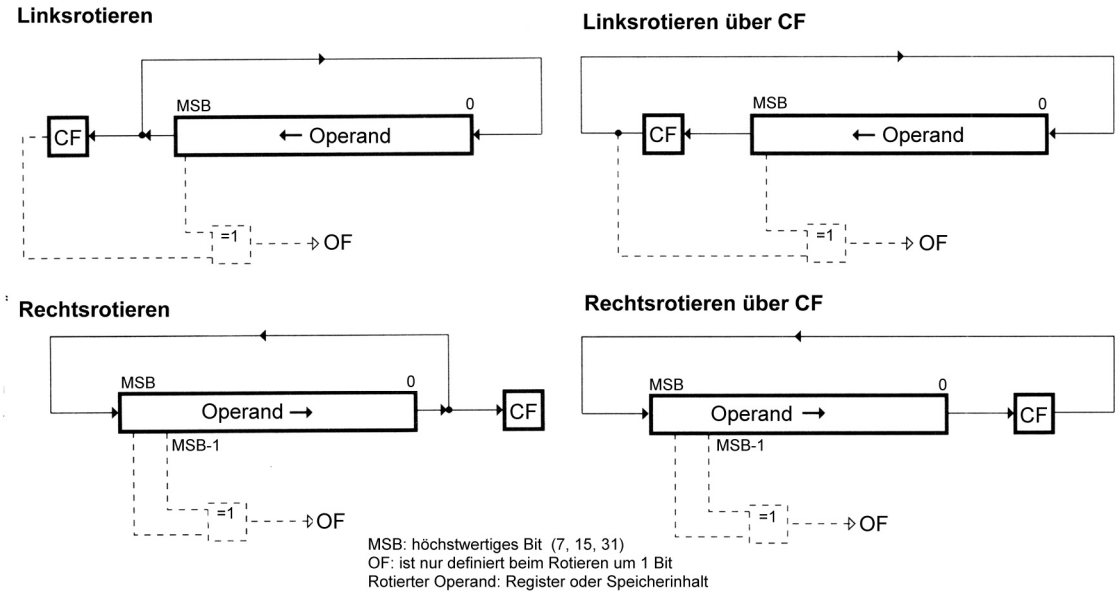

Abbildung 1.5 Rotationsabläufe

Dividieren durch Linksschieben:

a) Operand vorzeichenlos oder positiv: das Herausschieben des bzw. der niedrigstwertigen Bits wirkt als Rundung in Richtung Null - wie beim richtigen Dividieren. Es ergibt sich ggf. ein niedrigerer Wert. Z. B. 7 : 2 = 3 Rest 1.

b) Operand negativ: das Herausschieben des bzw. der niedrigstwertigen Bits wirkt als Rundung in Richtung -∞. Somit kann sich - anders als beim richtigen Dividieren - auch hier ein niedrigerer (= mehr negativer) Wert ergeben. Z. B. - 7 : 2 = - 3 Rest 1 beim Dividieren, aber - 4 beim (arithmetischen) Rechtsschieben. Wann das nicht passieren kann: beim Rechnen im Einerkomplement...

Verschieben über mehr als ein Maschinenwort
Die herausgeschobenen Bits müssen in das jeweils nächste Maschinenwort einlaufen. Typische Programmiertechniken:

1. bitweise schieben (Ablauf ggf. als Schleife programmieren):
   - Bit in das Carry-Flag (CF) herausschieben,
   - Bit durch Rotieren über CF einschieben.

2. um mehrere Positionen schieben:
   - Verschieben im jeweils letzten Maschinenwort,
   - Einfügen der Bits aus dem vorhergehenden Maschinenwort usw.

Unterstützt die Maschine ein Verschieben mit Erweiterung (Abbildung 1.4), so lassen sich beide Abläufe vereinigen (beim Herausschieben werden die freiwerdenden Bitpositioen aus dem jeweils vorhergehenden Wort nachgefüllt, dann ist dieses Wort zu verschieben, wobei wiederum dessen freiwerdende Bitpositionen aus dem vorhergehenden nachgefüllt werden usw.).

Fehlen derartige Abläufe, so muß man sie programmseitig nachbilden (z. B. über ein Hilfsregister: 1. Wort schieben, 2. Wort nach Hilfsregister. Dort die einzuschiebenden Bitpositionen ausblenden (AND-Befehl) und an das jeweils andere Ende verschieben, dann Hilfsregisterinhalt in 1. Wort einfügen (OR-Befehl) usw.).

Übungsaufgaben
Wir beziehen uns auf den fiktiven Prozessor P/F, Stand 1.4. Es dürfen der gesamte Registersatz sowie der Stack ausgenutzt werden.

Hinweis:
Stellen Sie dem eigentlichen Ablauf einige LDI-Befehle voran, die die jeweiligen Register mit passend gewählten Beispielwerten laden.

1. Erzeugen Sie einen Binärvektor im Register R1, der links Nullen und rechts Eisen enthält (Muster 000...0011..11B). Die Anzahl der Einsen finden Sie im Register R2 als Binärzahl vor.

2. Erzeugen Sie einen Binärvektor im Register R1, der links Einsen und rechts Nullen enthält (Muster 111...1100..00B). Die Anzahl der Einsen finden Sie im Register R2 als Binärzahl vor.

3. Erzeugen Sie im Register R1 ein Bitfeld, das aus Einsen besteht, die von Nullen umgeben sind (Muster 00...0011...1100...00). Parameterübergabe: R2: Länge des Bitfeldes; R3: Position der niedrigstwertigen Eins.

4. Schreiben Sie einen Ablauf, der die EXTRACT-Operation nachbildet. Parameterübergabe: R1: das ursprüngliche Maschinenwort; R2: Länge des Bitfeldes; R3: Position der niedrigstwertigen Eins des zu extrahierenden Bitfeldes. Rückgabe des extrahierten Bitfeldes in R4. Der gesamte Befehlsvorrat ist nutzbar. Programmieren Sie so, daß es schnell geht (kürzeste Ausführungszeit).

5. Das in Aufgabe 4 extrahierte Bitfeld enthalte eine ganze Binärzahl. Erweitern Sie diese Binärzahl vorzeichengerecht auf ein 32-Bit-Maschinenwort.

6. Schreiben Sie einen Ablauf, der die DEPOSIT-Operation nachbildet. Parameterübergabe: R1: das Maschinenwort, in das das Bitfeld einzusetzen ist; R2: Länge des Bitfeldes; R3: Position der niedrigstwertigen Eins des zu extrahierenden Bitfeldes; R4: das einzusetzende Bitfeld. Der gesamte Befehlsvorrat ist nutzbar. Programmieren Sie so, daß es schnell geht (kürzeste Ausführungszeit).

7. Wiederholen Sie Aufgabe 4, aber ohne Einzelbitbefehle und bei Beschränkung auf Verschieben/Rotieren um jeweils eine Bitposition (Programmierweise z. B. für Mikrocontroller, die keine Mehrbitverschiebung vorsehen).

8. Wiederholen Sie Aufgabe 6, aber ohne Einzelbitbefehle und bei Beschränkung auf Verschieben/Rotieren um jeweils eine Bitposition (Programmierweise z. B. für Mikrocontroller, die keine Mehrbitverschiebung vorsehen).

9. Bestimmen Sie die Position der niedrigstwertigen Eins in einem Maschinenwort, das in Register R1 bereitsteht. Rückgabe des Zahlenwertes in Register R2.

10. Bestimmen Sie die Anzahl der Einsen in einem Maschinenwort, das in Register R1 bereitsteht. Rückgabe des Zahlenwertes in Register R2.

11. Die Register R1 und R2 enthalten ein 64-Bit-Wort. Verschieben Sie dieses Wort nach links. Die Anzahl der Bits, um die zu verschieben ist, finden Sie in Register R4 vor.

12. Die Register R1, R2 und R3 enthalten ein 96-Bit-Wort. Verschieben Sie dieses Wort nach links. Die Anzahl der Bits, um die zu verschieben ist, finden Sie in Register R4 vor.