Wagner
Ausborn
Brandes

Kühlschrank-Inhaltserfassung

Projektarbeit von: Torsten Brandes, Mark Ausborn, Clemens Wagner

brandes@informatik.hu-berlin.de
ausborn@informatik.hu-berlin.de
wagnerc@informatik.hu-berlin.de

INHALTSVERZEICHNIS

EINFÜHRUNG

1. Kühlschrank-Inhaltserfassung

1.1 Problembeschreibung

Jeder kennt das Problem. Man fährt schnell nach der Arbeit noch einkaufen, weil der Kühlschrank inzwischen leer ist, aber man hat dann doch vergessen, einige wichtige Dinge einzukaufen.

Mit diesem Gerät kann das nicht mehr passieren!

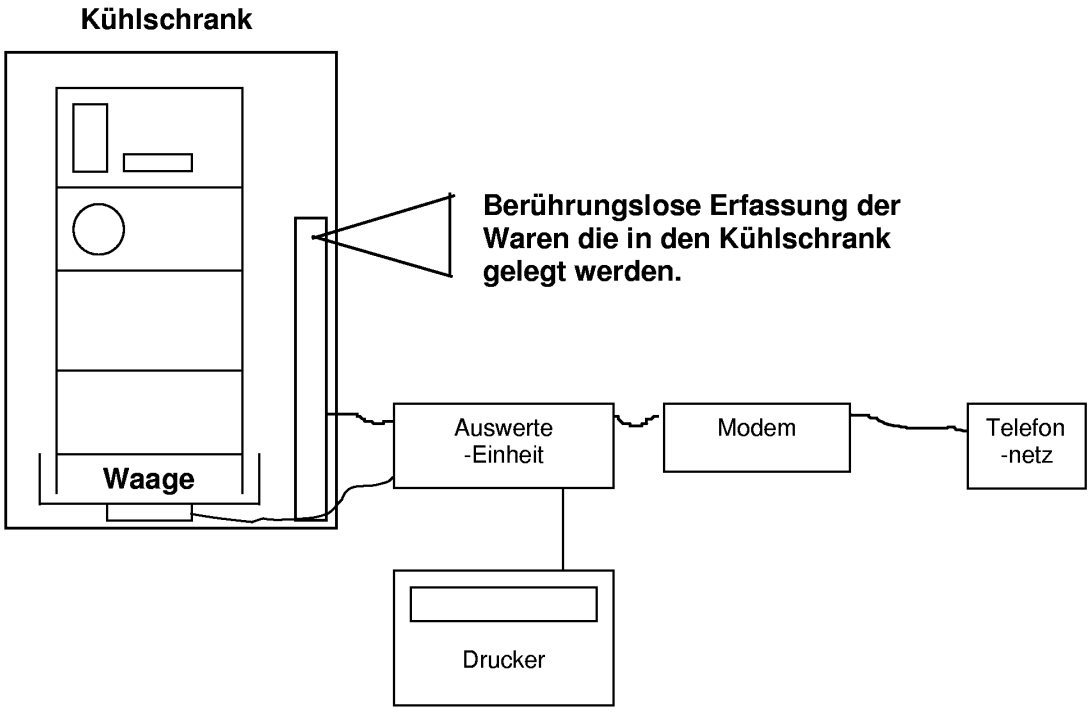
1.2 Die Lösung

Der Bediener hat durch die Programmierung an diesem Gerät bereits festgelegt welche Waren vorhanden sein sollen und in welcher Stückzahl bzw. welches Mindestgewicht davon vorhanden sein muß. Jedesmal wenn Artikel aus dem Kühlschrank entnommen werden, oder zurückgestellt werden, wird dieses über eine Erfassungseinheit registriert.

Unterschreitet ein Artikel ein bestimmtes Gewicht oder ist nicht mehr in ausreichender Stückzahl vorhanden, wird der Artikel automatisch z.B. bei dem ortsansässigen Kaufmann bestellt und die Artikel werden umgehend nach Hause geliefert. Dabei ist auch auswählbar, ob nur eine Einkaufsliste gedruckt werden soll.

1.3 Das Funktionsprinzip

Schematische Darstellung der Kühlschrankeinheit



Auf dem Kühlschrankboden befindet sich eine Waage, auf der die Fächer mit den Regalteilen für die Aufbewahrung der Waren im Kühlschrank gelagert sind. Werden nun Waren in das Regal gelegt oder herausgenommen, werden diese durch den Sensor erfaßt, eine Gewichtsdifferenz durch die Waage registriert und durch Ausgabe von Bitfolgen an die Auswerteeinheit gemeldet.
Damit diese Gewichtsdifferenz einem Produkt zugeordnet werden kann ist eine spezifische Produkterkennung nötig. Hierzu hatten wir zwei Möglichkeiten in Betracht gezogen. Auf jedem verpackten Produkt ist ein Label mit einem Barcode aufgedruckt. Mit einem im Kühlschrank integrierten Barcodereader, kann dieser automatisch gelesen und an die Auswerteeinheit gemeldet werden. Selbiges kann auch durch einen aufgeklebten Transponder erreicht werden, einer Miniatur-Sendeeinheit, die mit Hilfe eines Sendeimpulses eine Codefolge ausgibt, die einem spezifischen Produkt entspricht.
Da der Barcode bereits standardmäßig auf allen Produkten enthalten ist, entschieden wir uns aus Kostengründen für diese Variante.

Die Verwendung eines Transponders läßt zwar vielfältigere Anwendungsmöglichkeiten zu (siehe Ausblick) und ist aber eben (noch) nicht standardmäßig auf sämtlichen Produktverpackungen enthalten.
Die Auswahleinheit kommuniziert mit verschiedenen Peripheriegeräten. Der Dialog zwischen User und Programm wird durch ein Touchscreen geführt. Ein Touchscreen ist eine Bildschirmeinheit, auf der Grafiken und Texte dargestellt werden können. Durch Berührung können Objekte auf dem Bildschirm (z.B. Menüs) ausgewählt werden. Der Benutzer kann mit Hilfe dieses Gerätes die Art der Ausgabe bestimmen. So ist es möglich zu entscheiden ob eine Einkaufsliste ausgedruckt werden soll, oder ob die Einkaufsartikel über E-Mail zu einer bestimmten Zeit automatisch an einen vorgewählten Lieferanten geschickt werden sollen. Dieser könnte die Produkte dann nach Hause liefern. Die Verbindung über das Telefonnetz ließe auch eine externe Abfrage der Fehlbestandsliste über ein Handy zu. Der Besitzer kann also beispielsweise im Supermarkt seinen Kühlschrank über das Handy anrufen und bekommt sofort eine Liste der fehlenden Produkte per SMS zugestellt. Wir haben uns jedoch für die Realisierung des Prozessors im Kühlschrank entschieden, weil uns diese Aufgabe interessanter erschien.
Da nur Adressen für etwa 1000 Produkte vorhanden sind, ist eine Reduktion der Produktzahl durch den vorgegebenen Industriellen Barcode nötig. Das vorhandene Barcodesystem läßt die Codierung von etwa 100000 Produkten zu. Aus Abstraktionsgründen beschränken wir uns auf 1024.

1.4 Rückblick

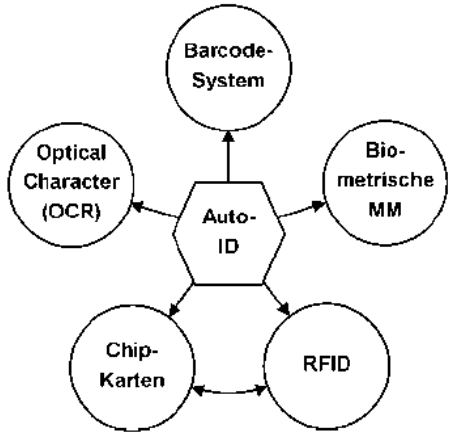
Die ersten Kühlschränke wurden um 1800 in Großbritannien gebaut. Sie waren jedoch mit giftigen Stoffen wie Äther ausgerüstet. Da Äther ein flüchtiges Gas ist, explodierten diese Kühlschränke hin und wieder mit verheerenden Folgen.
Um 1920 veränderten zwei junge Ingenieure des Royal Institute of Technology von Stockholm die Welt der Kühlschränke.
Bultzar von Platen und Carl Munters erfanden den Verdampfungsprozeß auf der Basis von Ammoniak. Ammoniak verdampft bereits bei 37°C und entzieht dabei seiner Umgebung Wärmeenergie. Das nun gasförmige Ammoniak wird durch Kompression wieder verflüssigt und der Kreislaufprozeß kann wieder von Vorne beginnen. Der Prozeß ist örtlich derart gestaltet, daß der Verdampfungsprozeß Wärmeenergie aus dem Inneren des Kühlschranks entzieht und nach anschließender Kompression, die Hitze an die Außenluft abgibt. Später wurden anstelle von Ammoniak Fluorchlorkohlenwasserstoffe (FCKW`s) als Kühlflüssigkeit eingesetzt, da sie sehr reaktionsträge sind und demnach als unbedenklich galten. In der Stratosphäre werden sie radikalisch gespalten und zerstören die Ozonschicht. Umweltfreundliche Kühlschränke werden inzwischen mit Chlorfreien Fluorkohlenwasserstoffen FKW`s ausgestattet.

Aufgrund der großen Nachfrage ist der Kühlschrank im Laufe der Zeit zu einem Gerät herangewachsen, das in den heutigen Haushalten nicht mehr wegzudenken ist.
Dieses war der Beweggrund der zur Idee dieses Projektes führte. Die Schilderung der Schwierigkeiten beim der Einkaufsorganisation in einer Wohn-

gemeinschaft inspirierte uns diese Kühlschrankinhaltserfassung zu realisieren.
Wir sehen die Idee gleichwohl als gelungen und zukunftsträchtig an, da jeder Berufstätige über seine Freizeit frei verfügen und diese nicht mit lästigen Pflichten verbringen möchte.
Außerdem läßt sich diese Idee kostengünstig und als sinnvolle Ergänzung in jeden Haushalt einführen.

1.5 Transponder versus Barcode

Zur Identifikation unserer in den Kühlschrank eingeführten Produkte stellte sich uns die Frage wie wir unsere Artikel erfassen sollten. Eine Recherche gab uns eine Vielzahl an Lösungsmöglichkeiten, aus der wir nach eigenen, unten aufgeführten Kriterien schließlich den Barcodereader als Lösung auswählten.
Hier eine Übersicht über automatische Identifizierungssysteme:



Aus der unteren Tabelle ist zu sehen, daß die zu übertragende Datenmenge bei Barcodesystemen relativ gering und bei Radio Frequency Identification-Systemen (RFID, Transponder) dagegen relativ hoch ist.

Tabelle 1.1: Der Vergleich verschiedener RFID-Systeme zeigt deren Vor- und Nachteile.

System: / Parameter:	Barcode	OCR	Sprecherer-kennung	Biometrie	Chipkarte	RFID-Systeme
Typische Datenmenge/ Byte:	1 100	1 100	–	–	16 64k	16 64k
Datendichte	gering	gering	hoch	hoch	sehr hoch	sehr hoch
Maschinenlesbarkeit	gut	gut	aufwendig	aufwendig	gut	gut
Lesbarkeit durch Personen	bedingt	einfach	einfach	schwer	unmöglich	unmöglich

System: / Parameter:	Barcode	OCR	Sprecherer-kennung	Biometrie	Chipkarte	RFID-Systeme
Einfluß von Schmutz/ Nässe	sehr stark	sehr stark	–	–	möglich (Kontakte)	kein Einfluß
Einfluß von (opt.) Abdeckung	totaler Ausfall	totaler Ausfall	–	möglich	–	kein Einfluß
Einfluß von Richtung und Lage	gering	gering	–	–	eine Steck-richtung	kein Einfluß
Abnutzung/Verschleiß	bedingt	bedingt	–	–	Kontakte	kein Einfluß
Anschaffungskosten/ Leseelektronik	sehr gering	mittel	sehr hoch	sehr hoch	gering	mittel
Betriebskosten (z.B. Drucker)	gering	gering	keine	keine	mittel (Kontakte)	keine
unbefugtes Kopieren/ Ändern	leicht	leicht	möglich*) (Tonband)	unmöglich	unmöglich	unmöglich
Lesegeschwindigkeit (incl. Handhabung des Datenträgers)	gering 4 s	gering 3 s	sehr gering 5 s	sehr gering 5 ... 10 s	gering 4 s	sehr schnell 0,5 s
Maximale Entfernung zwischen Datenträger und Lesegerät	0 ... 50 cm	1 cm (Scanner)	0 ... 50 cm	direkter Kontakt **)	direkter Kontakt	0 ... 5 m, Mikrowelle

Die geringen Kosten für die Leseelektronik sind ein großer Vorteil des Barcodes. Die hohe Reichweite zwischen Sender und Empfänger von bis zu 5 m lassen interessante Erweiterungen beim RFID-System zu.

Eine beispielhafte Darstellung für die Realisierung des RFID-Systems.
Die Applikation: Computer steht für die Erfassungseinheit im Kühlschrank.

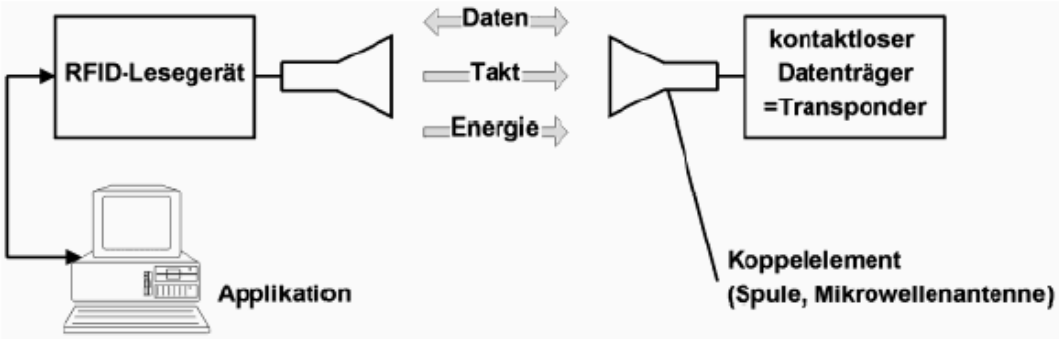


Bild 1.6: Lesegerät und Transponder sind die Grundbestandteile jedes RFID-Systems.

Die Bedeutung der Codenummern (des codierten Strichkodes) beim Barcode:

Länder-kenn-zeichen		Bundeseinheitliche Betriebsnummer bbn					individuelle Artikel-nummer des Herstellers					PZ
4	0	1	2	3	4	5	0	8	1	5	0	9
BRD		Fa. Musterwerk Identstrasse 1 80001 München					Schokoladenhase 100g					

Bild 1.2: Beispiel für den Aufbau eines Barcodes in EAN-Codierung (EAN Europäische Artikelnumerierung).

Für unsere Anwendung im Kühlschrank haben wir nur ein Maximum von etwa 1000 Produkten angenommen. Die Reduzierung der Artikel könnte durch ein zusätzliches Programm, welches den einzelnen Barcodes der Kühlschrank-Produktpalette Begriffe und die Zahlen von 0 bis 1000 zuordnet, die in unserer Erfassungseinheit gegeben sind, realisiert werden. Die individuellen Strichcodes können durch Etiketten vom Drucker gedruckt und durch den Benutzer auf die Ware geklebt werden.

Beispiel eines gegebenen Strichkodes nach Code 39 der verarbeitenden Industrie:

1.6 Ausblick / Erweiterungen

Gerade die Verwendung von Transpondern würde interessante Anwendungen eröffnen.
Da sie über eine größere Distanz übertragen werden können wäre beispielsweise eine Art Peilgerät denkbar, das den Käufer im Supermarkt den Weg zum richtigen Regal weisen könnte. Dies wäre auch besonders für sehbehinderte Menschen eine zusätzliche Erleichterung.

2. Projektablauf

2.1 Organisation und Aufgabenverteilung

Unser Projekt entstand ausschließlich in Teamarbeit. Wir trafen uns wöchentlich und besprachen Ideen und Probleme in Hinsicht auf das Projekt. Anschließend wurden die einzelnen Arbeitsbereiche verteilt und nach der Fertigstellung durch die anderen Projektteilnehmer korrigiert und ergänzt.

2.2 Simulation des Problems

Um einen ersten Eindruck von den Aufgaben zu bekommen, die unsere CPU zu erfüllen hat, wurde folgendes Programm zur Simulation der Vorgänge erstellt:

```
/************************ einzubindende Dateien ********************/

#include <string.h>
#include <stdio.h>

/************************ Funktionsprototypen **********************/

void main(void);
void init();
void update_weight(void);
void make_list(void);
void wak(void);
void anzeigen(void);
void add_to_list();
void loop(void);
void iosenden(void);

/********************** globale Variablen ***************************/

int SR1=0,SR2=0,SR3=0;  /* Statusregister */
int R1=0,R2=0,R3=0,R4=0,R6=0; /* allgemeine Register */
int speicher[16];  /* Speichergroesse zu Demo-Zwecken auf 8 Produkte
beschränkt */
int eingabe; /* Hilfsvariable für das Eingabemenue */
```

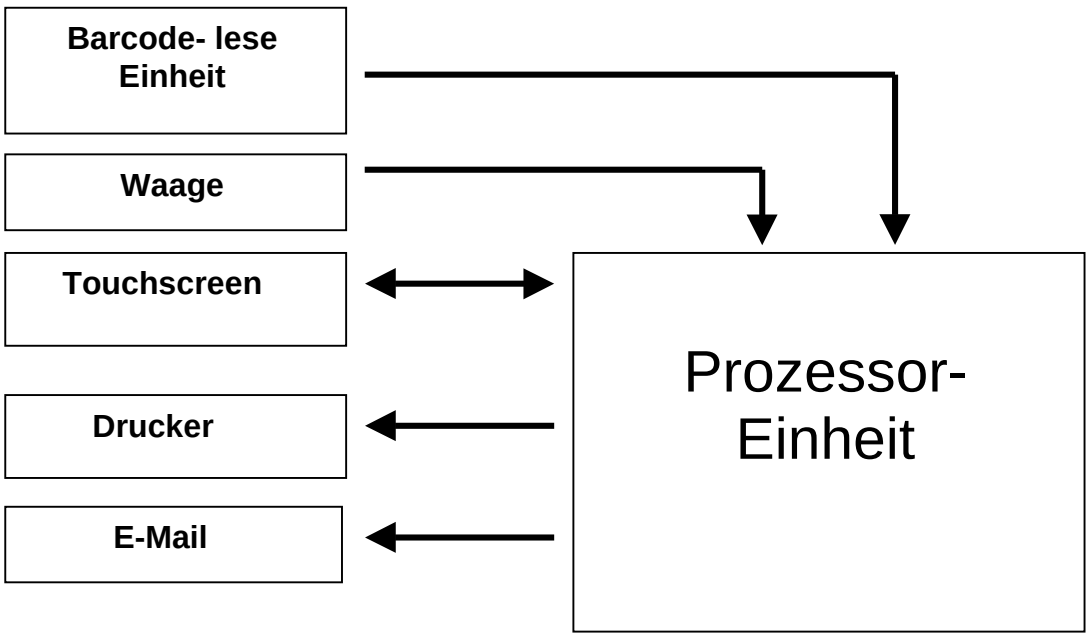
```
/************************ Hauptprogramm ****************************/

void main(void)
{
    init();
    for(;;)   /* Endlosschleife */
    {
          printf("\n\nMit Taste n+[ENTER] passiert folgendes:\n    1
Sensormeldung auslösen\n    2 I/O Meldung auslösen\n    3 Speicherin-
halt angezeigen\n\n    0 bricht das Programm ab.\n\n Eingabe:");
          scanf("%d",&eingabe);
          if(eingabe==0) break;
          if(eingabe==1) SR1=1;
          if(eingabe==2) SR3=1;
          if(eingabe==3) anzeigen();
          if(SR1==1) update_weight();
          if(SR3==1) make_list();
    }
}

void init(void)  /* legt die SOLL-Werte fest */
{
    int i;

    for(i=8;i<=16;i++)
    {
          speicher[i]=(i+2)*10;
    }
}

void update_weight(void)
{
    printf("Bitte eine Produktnummer eingeben (0-7):\n");
    scanf("%d",&R1);
    R3=R2;
    wak();
    R3=R2-R3;
    speicher[R1]=speicher[R1]+R3;
    SR1=0;
}

void make_list(void)
{
    R6=0;
    loop();
    SR3=0;
}

void loop(void)
{
    R3=speicher[R6];
    R4=speicher[R6+8];

    R4=R4-R3;
    if(R4>0) add_to_list();

    R6=R6+1;

    R4=R6-8;
    if(R4!=0) loop();
}
```

```
void add_to_list(void)
{
    SR3=2;
    while(SR3==2)
    {
          iosenden();
    }
}

void iosenden(void)
{
    /* Diese Information bekommt eigentlich die I/O-Einheit: */

    printf("Produkt Nr.: %d muß bestellt werden!\n Bitte mit RETURN
bestätigen.\n",R6);
    getchar();
    SR3=1;
}

void wak(void)
{
    printf("Altes Gesamtgewicht: %d dg\n",R2);
    printf("Bitte neues Gesamtgewicht eingeben! (in dg) \n");
    scanf("%d",&R2);
}

void anzeigen(void)
{
    int i;
    for(i=0;i<=7;i++) printf("Adr[%2d]=%d
Adr[%2d]=%d\n",i,speicher[i],i+8,speicher[i+8]);
}
```

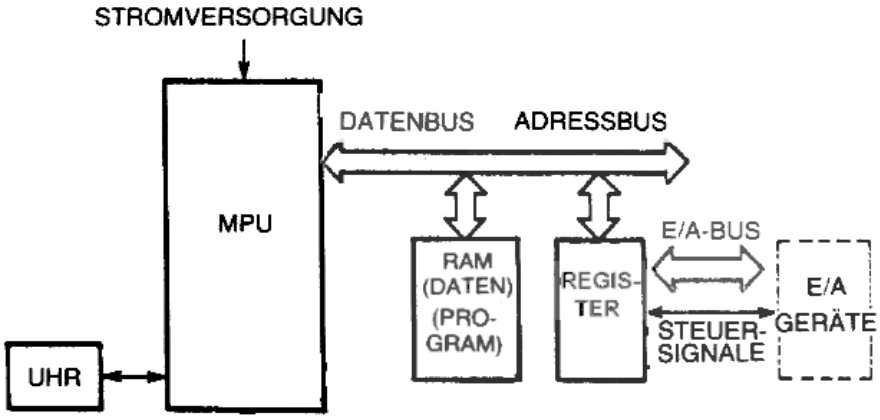
DIE HARDWARE

1. Schaltung/Prozessor-Environment

1.1 Übersicht: Komponenten



Die Prozessoreinheit kommuniziert mit verschiedenen Geräten. Aus dem Barcodelesesignal, den Daten über das aktuelle Gewicht der D/A Wandeleinheit der Waage und den vorgegebenen Bestandsinformationen, welche über die Bildschirmeingabeeinheit ausgegeben werden, führt die Steuereinheit eine Bestandsaufnahme durch. Bildschirm, Drucker und SMS-Mail Ausgaben erfolgen wiederum direkt über die Geräte. Es wird in dieser Arbeit von Übergabeprotokollen abgesehen und davon ausgegangen, daß die I/O Geräte ihre Daten direkt in Register schreiben.
Diese Vereinfachung nehmen wir vor, da in dieser Arbeit das Augenmerk auf der Programmierung der Prozessoreinheit liegt und Beispielhaft ein Prozessor entworfen werden soll.



Die Daten und das Programm werden in den RAM Speicher der Programmiereinheit gespeichert. Zur Pufferung der Daten falls der Strom ausfallen sollte dient ein Akku. Die mit der Prozessoreinheit verbundene Uhr dient zur automatischen Nachbestellung der fehlenden Produkte.

2. Der Prozessor

2.1 Erste Überlegungen/Annahmen

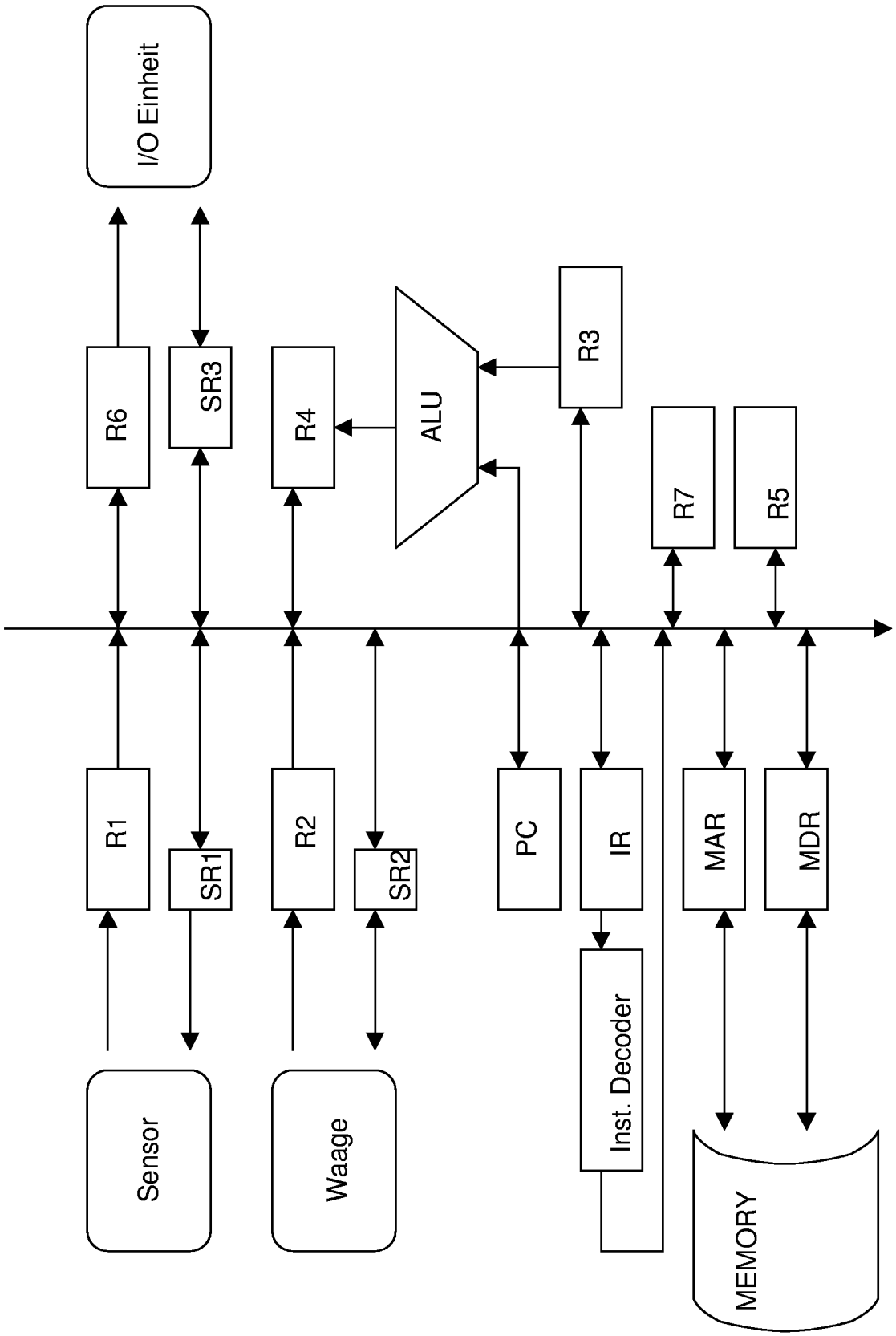
- Es existiert ein Scanner, der einen Produktcode optisch oder per Transponder berührungslos erfassen kann. Jedes einzelne Produkt ist gekennzeichnet, Mehrfachverpackungen enthalten mehrere selbstklebende Kennungen, mit denen man Einzelprodukte markieren kann. Es wird immer nur eine Produktart am Scanner vorbeigeführt (die Produkte werden nacheinander aus dem Kühlschrank genommen)
- Es gibt ca. 1000 existierende Produkte, die der Kühlschrank unterscheiden muß
- Es gibt für jedes Produkt einen vom Benutzer vorgegebenen Gewichts-Sollwert. Der Ist-Wert wird durch den Kühlschrank bestimmt
- Diese Daten können zu allen 1000 verschiedenen Produkten gespeichert werden (Speichergröße)
- Es existiert eine Benutzerschnittstelle über die die Sollwerte im Speicher geändert werden können.
- Der Sollwert beträgt maximal 50% des Ist-Wertes.
- Das Gewicht eines Produkts ist ≤ 10Kg
- Das Gesamtgewicht aller Produkte ist ≤ 100Kg
- Die Genauigkeit von 10g (= 1dg) bei der Istwertbestimmung ist ausreichend
- Die I/O-Benutzerschnittstelle ist intelligent. Sie bietet auch einen Sollwerteditor.

Daraus folgt:

- pro Produkt müssen folgende Informationen gespeichert werden:
- 10 Bit (2^{10}=1024) für den Gewichts-Istwert (0-10230g in 10g Schritten)
- 9 Bit (2^{9}=512) für den Gewichts-Sollwert (0-5110g in 10g Schritten)
- minimale Speichergröße (Speicherwortbreite 10 Bit): 1024 Produkte x 20 Bit = 2,5 KByte
- Der Speicher ist in Blöcken organisiert. Der Produktcode ist gleichzeitig Speicheradresse des Ist-Gewichts

Die Adressbreite wäre folglich für den Datenspeicher 11 Bit. Wie nehmen an, daß der Programmspeicher nochmal ca. 2,5 KByte groß ist, die Adressbreite also 12 Bit beträgt.
Da der größte zu verarbeitende Wert (Gesamtgewicht des Kühlschranks) jedoch 14 Bit zur Kodierung benötigt, legen wir die Wortbreite (Bus, Speicherwort und alle Register) des Systems auf 14 Bit fest. Eine Ausnahme bilden die Statusregister.
Die resultierende Speichergröße wäre also einmal 14Bit x 2048 = 3.5 KByte.
Dazu kämen ca. 2.5 KByte Speicher für das Programm, also insgesamt ungefähr 6 KByte.

2.2 WAB CPU-Design



2.3 Beschreibung der verwendeten Register

Bez.	Größe	Beschreibung
allgemeine Register:		
R1	14	Schnittstelle zum Sensor; nach dem Vorbeiführen eines Produktes am Scanner liegt hier der Produktcode des Produktes an Aktualisierung beim Erfassen eines Produkts (SR1=1)
R2	14	Schnittstelle zur Waage; Gewicht des gesamten Kühlschrankinhalts; Aktualisierung, wenn SR2 gesetzt ist
R3	14	**ALU**-Hilfsregister (Operand)
R4	14	**ALU**-Hilfsregister (Ergebnis)
R5	14	allgemeines Hilfsregister (keine spezielle Verwendung)
R6	14	Kommunikation mit der **I/O - Einheit** (Produktcode) Wird von selbiger ausgelesen, wenn SR3 gesetzt ist
R7	14	Zeiger auf aktuelle Speicherstelle
MAR	14	Memory Adress Register (Speicherzugriff)
MDR	14	Memory Data Register (Speicherzugriff)
Statusregister:		
SR1	1	=1: **Sensor** hat ein Produkt erfaßt, Produktcode liegt in R1 vor , Sensor wartet auf SR1=0
SR2	1	=1: **Waage** soll das aktuelle Gewicht in R2 ablegen, dann setzt sie SR2=0
SR3	2	=1: **I/O Einheit** fordert Liste an, nach dem Auslesen setzt sie SR3=1, =2: R6 bereit zum Auslesen durch **I/O Einheit**
Programmablaufregister:		
PC	14	Programmschritt-Zähler
IR	14	Instruktionsregister

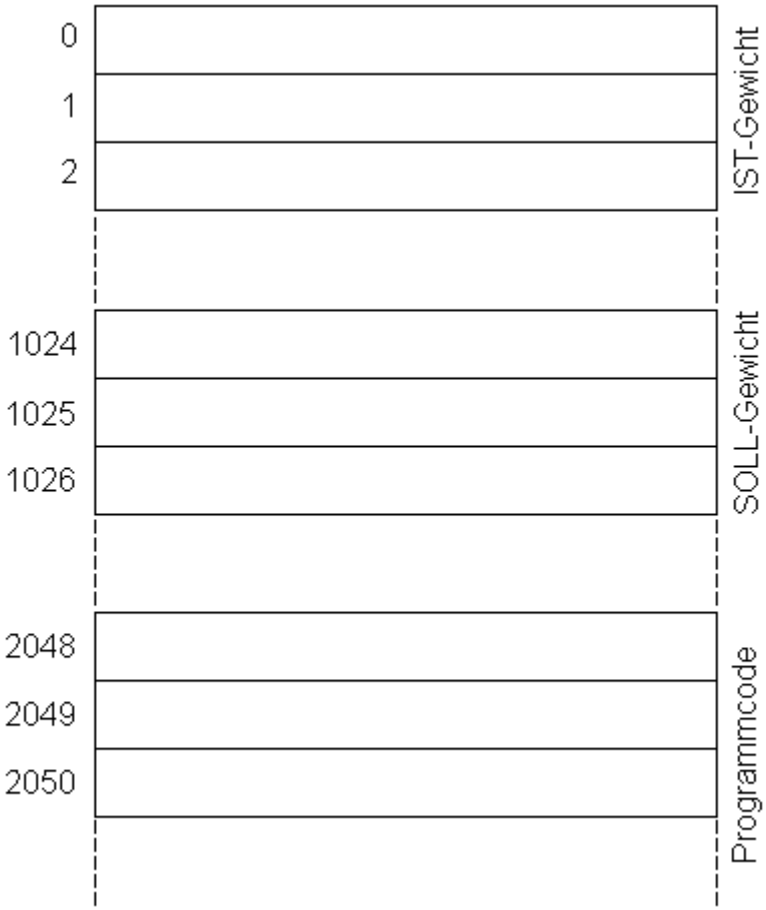
2.4 Die ALU

Die ALU wird von uns ausschließlich zum Addieren und Subtrahieren benutzt. Eine extra Schaltung für die logischen Funktionen entfällt also.
Aus Effizenzgründen sind die arithmetischen Befehle folgendermaßen definiert:

- add arg1, arg2 := arg1 + R3 $\Rightarrow$ R4
- sub arg1, arg2 := arg1 - R3 $\Rightarrow$ R4

d.h., vor der Ausführung muß darauf geachtet werden, den zweiten Operanden nach R3 zu laden.

2.5 Speicherorganisation der WAB-CPU



Die Speicherwortbreite beträgt 14 Bit.
Wir haben den Speicher in drei Blöcken organisiert.
Der erste Block kann direkt durch den Produktcode adressiert werden, da es nach unserer Festlegung 1024 "Kühlschrankrelevante" Produkte zu kaufen gibt.
An der Speicherstelle n steht also das Ist-Gewicht des Produkts mit dem Produktcode n. Diese Lösung ist äußerst effizient, da auf zusätzliche Speicherung des Produktcodes verzichtet werden kann. Außerdem entfallen komplizierte Suchalgorithmen.
Der zweite Block wird dann beim Durchsuchen der Liste nach fehlenden oder in zu geringer Menge vorhandenen Produkten jeweils durch einen speziellen Befehl angesprungen.
An der Speicherstelle n+1024 steht das Soll-Gewicht des Produkts mit dem Produktcode n.

Der Programmcode steht dann ab Adresse 2048 im Speicher.

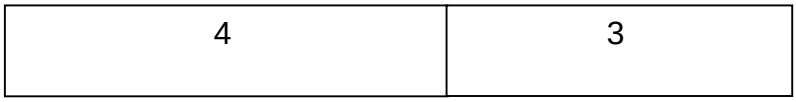
Für diese Lösung, die ziemlich viele Sprünge erfordert haben wir uns entschieden, weil so die Produktdaten direkt hintereinander im Speicher stehen und direkt durch den Produktcode adressiert werden können. Damit entfallen aufwendigen Suchen oder eine Umkodierung der Produktkennung.
Mit der selben Argumentation ist auch zu begründen, daß wir immer 2048 Speicherstellen für sämtliche existierende Produkte "bereithalten", obwohl der Fall, daß ein Benutzer alle 1024 Produkte im Kühlschrank hat sehr unwahrscheinlich ist.
Wir gehen außerdem davon aus, daß die Kostenreduzierung die bei einer Beschränkung der Speichergröße auf ca. 100 verschiedene, im Kühlschrank gleichzeitig enthaltene Produkte resultieren würde, gegenüber den Zeit- und Materialmehrkosten bei der Entwicklung einer Umkodierung vernachlässigbar ist.
Weiterhin ist eine wiederaufladbare Batterie notwendig, um den Speicherinhalt gegen eventuelle Stromausfälle sichert.
Noch sinnvoller wäre sicherlich ein Sekundärspeicher, wie eine Diskette zur Sicherung der Daten.

2.6 Das Befehlsformat

- Es existieren 11 Befehle, die mit 4 Bit kodiert werden können.
- Der erste Operand kann folgendermaßen adressiert werden: Register-direkt, immediate, Speicher indirekt durch Register. Der zweite Operand kann folgendermaßen adressiert werden: Speicher indirekt durch Register, Register direkt
- Da wir die Befehle so definiert haben, daß das Operandenformat jeweils feststeht, werden keine weiteren Bits zur Kodierung der Adressierungsart benötigt.
- Für einen Immediate-Operanden werden 10 Bit benötigt (maximal)
- Die beiden anderen Adressierungsarten benötigen 3 Bit um die 7 Register anzusprechen

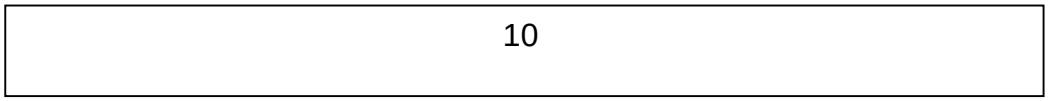
Da die Wortbreite nur 14 Bit beträgt, wird der Operand 1 im nächstfolgenden Speicherwort abgelegt, wenn ein immediate-Wert benötigt wird. Dies ist bei zwei Befehlen der Fall: sub und ad*.

Speicherwort n:



Befehl Operand 2

Speicherwort n+1:



Operand 1

2.7 WAB Befehlssatz

Microcode	Beschreibung als Assemblerbefehl
$Ri_{out,}$ Rj_{in}	**load Ri, Rj** (lade von Register nach Register)
set #1, SR2 $R2_{in,}$, WFWC	**wak** (veranlasse Waage zum Senden des aktuellen Kühlschrankgewichts)
Ri_{out}, sub to R4 $R4_{out}$, Rj_{in}	**sub Ri, Rj** (rechne Ri – R3 und speichere nach RJ)
$PC_{out,}$ set carry-in to ALU, add to R4 $R4_{out}$, MAR_{in}, PC_{in} $M[MAR]_{out}$, MDR_{in}, WFMFC MDR_{out}, IR_{in} $Op(IR)_{out}$ sub to R4 $R4_{out}$, Ri_{in}	**sub #x , Ri** (rechne x – R3 und speichere nach Ri) Hier muß das nächste Befehlswort ausgelesen werden, um den Wert x zu erhalten. PC inkrementieren neuer Fetch
Ri_{out}, MAR_{in} $M[MAR]_{out}$, $MDR_{in,}$ WFMFC MDR_{out}, ADD to R4 $R4_{out}$, MDR_{in} Rj_{out}, MAR_{in} MDR_{out}, $M[MAR]_{in}$, WFMFC	**add (Ri), (Rj)** (rechne (Ri) + R3 und speichere nach (Rj))
Ri_{out}, MAR_{in} $M[MAR]_{out}$, MDR_{in}, WFMFC MDR_{out}, Rj_{in}	**load (Ri), Rj** (Lade Speicherzelle (Ri) nach Register Rj)

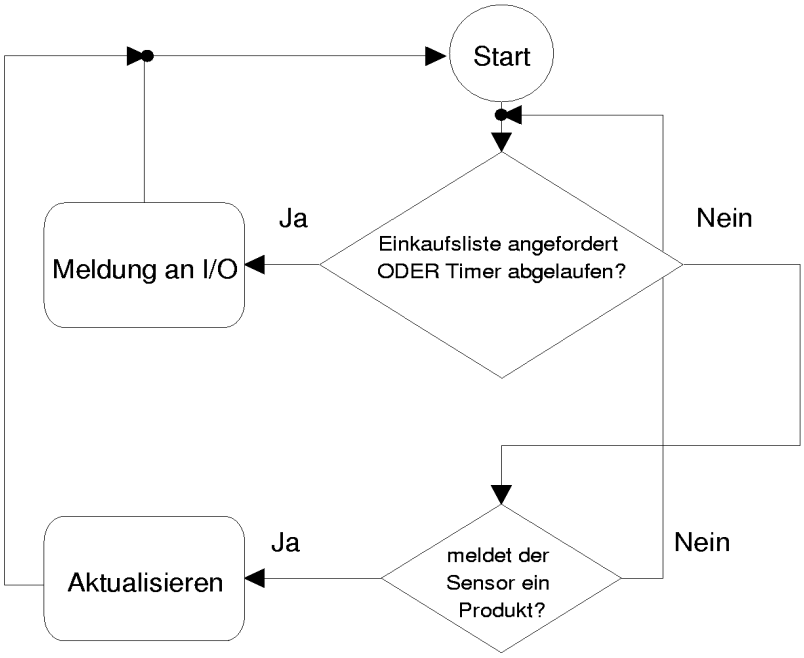
WAB-Befehlssatz: (Fortsetzung)

MAR_{out}, $R3_{in}$ $PC_{out,}$ set carry-in to ALU, add to R4 $R4_{out}$, MAR_{in}, PC_{in} $M[MAR]_{out}$, MDR_{in}, WFMFC MDR_{out}, IR_{in} $Op(IR)_{out}$ Add to R4 $R4_{out}$, MAR_{in}	**ad*$_{inc}$ #x** (erhöhe MAR um x) Hier muß das nächste Befehls-wort ausgelesen werden, um den Wert x zu erhalten. PC inkrementieren, neuer Fetch
$Op(IR)_{out}$, Ri_{in}	**set #x, Ri** (setze #x in Register i)
	jz label (springe zu label, wenn das zero – flag 1 ist)
	j label (springe zu label)
	jp label (springe zu label, wenn positiv–flag 1 ist)

DIE SOFTWARE

1. Ablaufbeschreibungen (Flußdiagramme)

1.1 Das Hauptprogramm

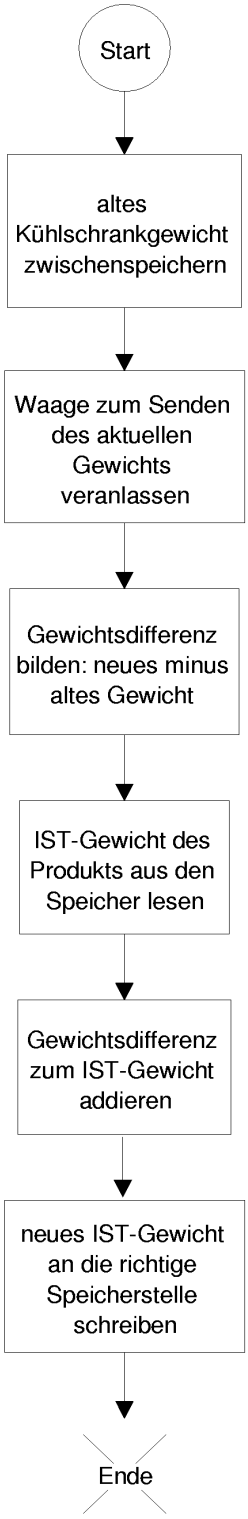


Beschreibung:
Das Programm befindet sich im Ausgangszustand in einer Endlosschleife. Dabei werden ständig die Statusregister abgefragt, die den Zustand der angeschlossenen Peripherie beschreiben.
Dabei kann es zu Verzweigungen zu zwei verschiedenen Unterprogrammen kommen:

1. Wird vom Benutzer am I/O Gerät die Aktualisierung der Einkaufsliste verlangt, oder nimmt das I/O Gerät nach einer definierten Periode selbständig eine Aktualisierung seiner gespeicherten Liste vor, so wird in ein Programmteil verzweigt, der den Speicher nach Produkten durchsucht die in zu geringer Menge vorhanden sind.
2. Führt der Benutzer ein Produkt am Sensor vorbei (Produktentnahme/Produkteinlagerung), so wird das Gewicht des gesamten Kühlschrankinhalts (Waage) das Einzelgewicht des Produkts bestimmt und gespeichert. Wir gehen hierbei davon aus, daß erst dann verzweigt wird, wenn die Waage eine Gewichtsveränderung registriert!

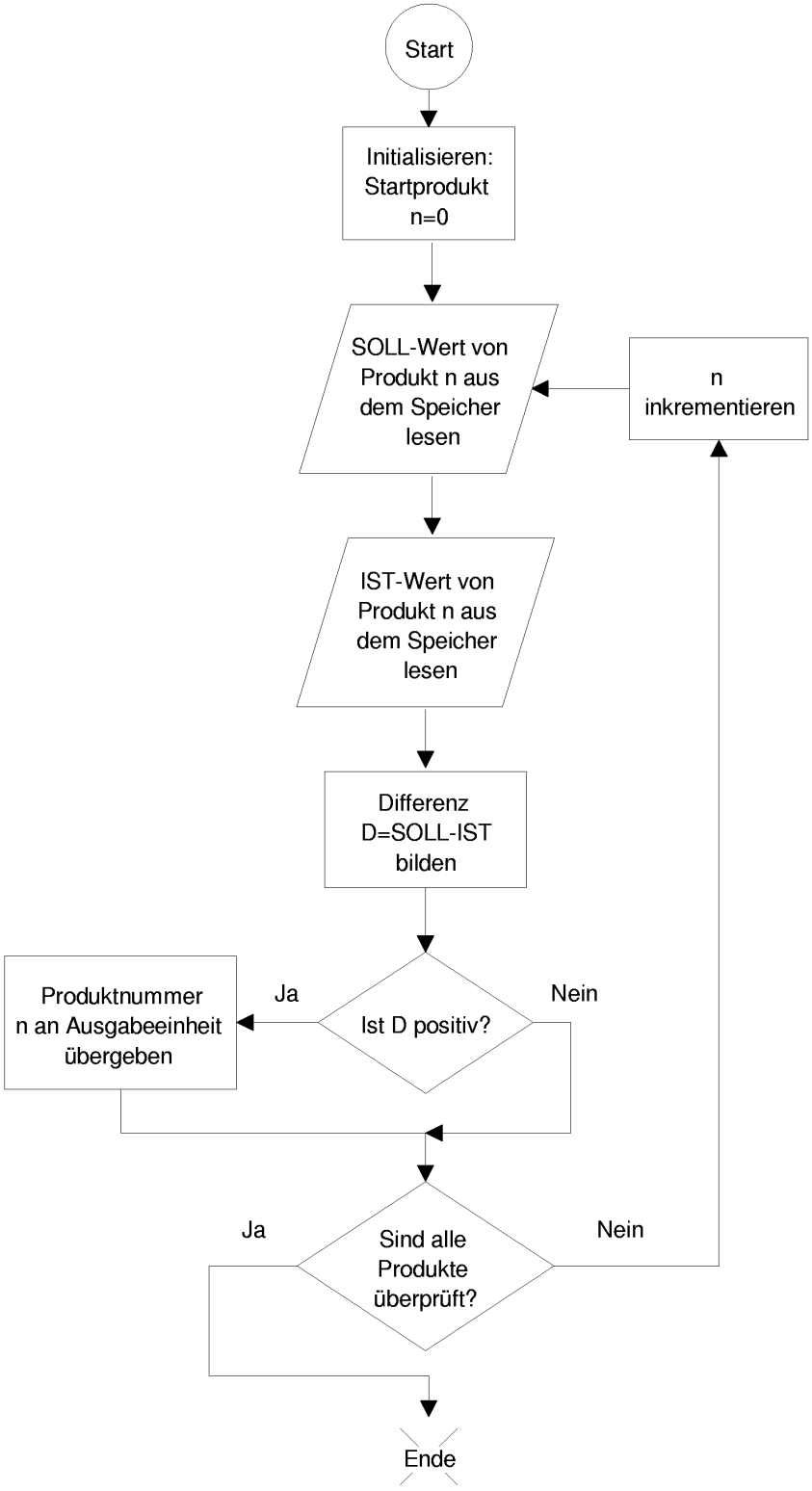
Nach Ablauf der Unterprogramme befindet sich das Programm wieder in der Warteschleife. Durch die Aufgabenverteilung (I/O Gerät fordert Aktualisierung der Einkaufsliste an) sparen wir uns die Realisierung einer Zeitmessungseinheit, die für eine turnusmäßige automatische Aktualisierung erforderlich wäre.

1.2 Aktualisierung des gespeicherten Einzelgewichts eines am Sensor erfaßten Produkts



Beschreibung:
Wird ein Produkt entnommen oder eingelagert, so wird in diesem Programmteil über die Differenz von Gesamtgewicht vor diesem Vorgang und dem aktuellen Gewicht das Einzelgewicht des Produktes berechnet. Mit diesem Ergebnis wird dann die Liste im Speicher entsprechend aktualisiert.
Hierbei geschieht im Einzelnen folgendes: Am Anfang liegen sowohl der Produktcode dieses Produkts, als auch das Gewicht des Kühlschranks vor diesem Vorgang vor. Dieses Gewicht muß nun zwischengespeichert werden und die Waage wird veranlaßt, das aktuelle Gesamtgewicht zu senden. Nun wird von dem aktuellen Wert das zwischengespeicherte alte Gewicht abgezogen. Die Differenz (positiv bei Einlagerung, negativ bei Entnahme) entspricht nun genau dem Einzelgewicht des Produkts, dessen Produktcode ja bekannt ist (Sensor).
Anhand dieses Codes kann jetzt der Speicher adressiert werden. Der gespeicherte Ist-Wert wird ausgelesen, zu dem ermittelten Einzelgewicht addiert, und wieder in den Speicher zurückgeschrieben.

1.3 Reaktion auf die Anforderung einer aktuellen Liste in zu geringer Menge vorhandener Produkte



Beschreibung:
Fordert die I/O Einheit eine aktuelle Liste der Produkte an, die in zu geringer Menge vorhanden sind, so wird von diesem Unterprogramm die Liste im Speicher nach Produkten durchsucht, die dieses Kriterium erfüllen. Die Produktnummern werden dabei nacheinander an die I/O Einheit übergeben.
Dieser Vorgang kann wie folgt beschrieben werden:
Es wird bei Produkt 0 begonnen. Ist- und Soll-Gewicht werden aus dem Speicher gelesen, voneinander Subtrahiert und bei positiver Differenz (Differenz=Soll-Ist) wird die Speicheradresse, die ja zugleich Produktnummer ist, an die I/O Einheit übergeben. Beim Erreichen des letzten Listeneintrags wird dieses Unterprogramm wieder verlassen.
Eine Übergabe der fehlenden Menge eines Produkts halten wir zunächst nicht für sinnvoll, da der Soll-Wert (und damit eine mindestens vorhandene Menge) ja vom Benutzer vorgegeben werden kann und somit ungefähr klar ist, wieviel von einem Produkt noch vorhanden ist.
Sicherlich wäre es aber manchmal auch interessant, wenn man jederzeit genau wüßte wieviel Dezigramm eines Produkts sich noch im Kühlschrank befinden. (z.B.: Partys, Reisevorbereitungen) Der Aufwand für eine Implementierung dieses Features würde sich in Grenzen halten.

Um die Editierung der Soll-Werte kümmert sich unser Programm nicht. Wir gehen davon aus, daß dies entweder noch implementiert wird, oder die I/O Einheit direkt auf den Speicher zugreifen kann.

1.4 Abschätzung der maximalen Ablaufdauer

Um abzuschätzen wieviel Zeit die WAB-CPU bei ihrer aufwendigsten Aufgabe benötigt (der Erstellung einer Fehlbestandsliste) sind wir folgendermaßen vorgegangen:
Dazu haben wir bei den im entsprechenden Unterprogramm vorkommenden Befehlen eine Laufzeitabschätzung (in Zyklen) anhand des Mikrocodes vorgenommen (inklusive Fetch).
Dabei ergab sich für das einfach Auswerten des Speicherinhalts eine Anzahl von 38982 Zyklen.
Zudem sind wir davon ausgegangen, daß jedes Produkt bestellt und damit an die I/O Einheit übermittelt werden muß, deshalb kommen nocheinmal 53248 Zyklen dazu.
In diesem (denkbar ungünstigsten) Fall würde also die WAB-CPU (bei einer angenommenen Taktrate von 8MHz) 11,52875 Millisekunden für diesen Vorgang brauchen.
Im günstigsten Fall (alle Produkte in ausreichender Menge vorhanden) wären es nur 4,87275 Millisekunden.

2. WAB Assemblerprogramm

```
@wait_for_sensor:

    load SR1, R3          Überprüfung, ob ein Produkt entnom-
    sub #1, R3            men oder hinzugefügt wurde.
    jz update_weight

@wait_for_I/O:

    load SR3, R3          Überprüfung, ob das Statusregister SR3
    sub #1, R3            der I/O Unit geschaltet ist.
    jz make_list
    j wait_for_sensor

@update_weight:           Alter Gewichtswert nach R3.
                          Neuen Gewichtswert nach R2 laden las-
    load R2, R3           sen. Subtrahieren ⇒ Gewichts-differenz
    wak                   liegt in R3 vor. R1 zeigt auf das aktuelle
    sub R2, R3            Produkt. Summe nach (R1) addieren
    add (R1), (R1)        und Statusregister löschen.
    set #0, SR1           Waage setzt SR2 wieder auf 0.

@make_list:

    set #0, R7            Enthält das erste Produkt.

    @loop:

        load R7, R3
        sub #1024, R3     auf EOF testen
        jz EOF

        load (R7), R5     Speicherinhalt laden.
        ad*inc #1024      Zu Speichersegment 2 springen.
        load (R4), R3     Sollwert laden.
        sub R5, R4        Wenn die Gewichtsdifferenz positiv ist,
        jp add_to_list    bestellen.
```

ad*$_{inc}$ #(-1023)	Zurück zu Speichersegment 1+1.
load R4, R7	Pointer R7 aktualisieren
j loop	Wiederholen bis EOF.

@add_to_list:

ad*$_{inc}$ #(-1024)	Zurück zu Speichersegment 1.
load R4, R6	Zu bestellendes Produkt in R4 (siehe Def. ad*$_{inc}$)
set #2, SR3	SR3 auf read setzen. I/O Unit kann Wert aufnehmen und SR wieder auf 1 setzen.
ad*$_{inc}$ #1	
load R4, R7	
j loop:	

@EOF:

```
set #0, SR3
j wait_for_sensor
```