HUMBOLDT-UNIVERSITÄT ZU BERLIN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR RECHNERORGANISATION UND –KOMMUNIKATION, PROF. MALEK

Projekt:

Autonome Gewächshaussteuerung

von

Nick Walther – 139995 – nwalther@informatik.hu-berlin.de
Martin Sommerfeld – 155833 – martin@elborn.de
Felix Schlademann – 180722 – felix.schlademann@web.de

INHALTSVERZEICHNIS

1. DARSTELLUNG DER PROBLEMSTELLUNG

1.1 PROJEKTIDEE / EINSCHRÄNKUNG DER BEARBEITUNG

Zum vorgegebenen Thema „Eingebettete Systeme“ wählten wir den Bereich der Automatisierung in Haus und Industrie und entschieden uns letztlich dafür, ein möglichst flexibles und vor allem autonomes Gewächshaus mitsamt Steuerung zu entwerfen.

Zu Beginn sammelten wir dabei Ideen über Ideen was denn in einem Gewächshaus genau autonom oder beinahe autonom gesteuert werden könnte und stellten dabei fest, dass mit dem entsprechenden Aufwand so ein System tatsächlich *vollautomatisch* betrieben werden könnte.

Da die Überlegungen jedoch größtenteils über ein studienbegleitendes Projekt mit einer Dokumentation von circa 20 Seiten Umfang hinaus gingen, beschlossen wir uns auf die wichtigsten beiden Punkte in einem Gewächshaus zu konzentrieren: Temperaturregelung und Bewässerung.

Weitere Details, wie etwa Düngung oder den Pflanzen zumutbare Lichtintensität / Sonnenscheindauer, könnten als mögliche Erweiterungen aufgefasst werden.

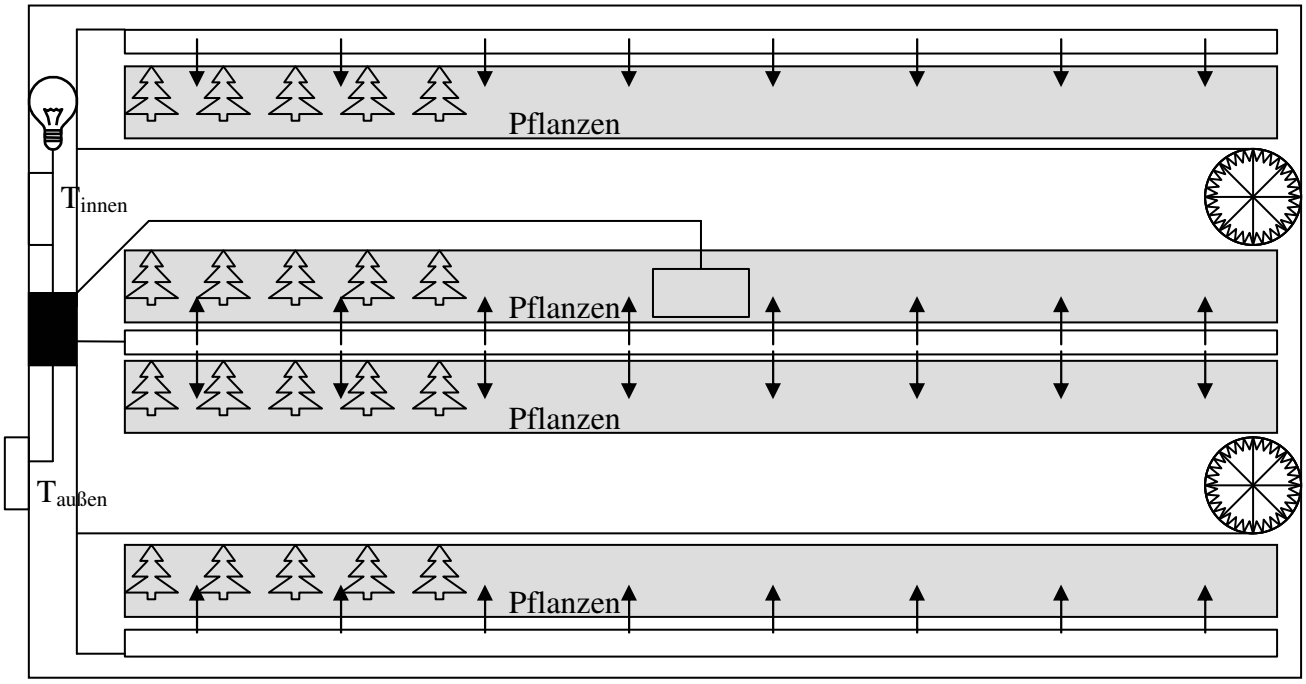
1.2 PROBLEMANALYSE & BASISFUNKTIONALITÄT

Will man Temperatur und Wasserzufuhr kontrollieren, so muss man sich Gedanken machen in welcher Art und Weise diese Kontrolle denn erfolgen soll.

Der Fall der ***Temperatur*** ist dabei leicht einsichtig: Sie soll möglichst in festgelegten Grenzen verbleiben, damit die Pflanzen das für sie optimale Klima vorfinden und so der Ernteertrag maximiert wird.
Sinkt also die Temperatur unter einen festzulegenden (und pflanzenabhängigen) *Minimalwert*, so muss etwas geschehen. Dazu erarbeiteten wir zwei Alternativen: Liegt der seltene Fall vor, dass die Temperatur außerhalb des Gewächshauses höher als innerhalb ist, so wird unsere Steuerung die Lüftung starten und so die Temperatur innen langsam der außen anpassen, dies geschieht zur Energieersparnis, ein nicht unerheblicher Faktor bei einem wenig isolierten Gebäude wie einem Glashaus. Im Normalfall jedoch wird die Temperatur außerhalb nicht höher sein als innen, weswegen ein Heizungssystem anläuft. Um im Falle dass die aktuelle Temperatur schwankt und nur knapp unter dem festgelegten Minimalwert liegt ein ständiges Ein- und Ausschalten der Heizung zu verhindern, wird erst eine festgelegte Zeit gewartet und nach dieser geprüft ob die Temperatur immer noch unter dem Sollwert ist. Erst dann starten wir die Heizung.
Ist die Temperatur wieder in den gewünschten Bereich gestiegen, so schaltet sich das Heizungssystem von unserer Anlage gesteuert automatisch ab.
Steigt jedoch die Temperatur über einen festzulegenden (und ebenso pflanzenabhängigen) *Maximalwert*, so gibt es nur eine Lösung: Es muss gelüftet werden. Wir diskutierten in diesem Zusammenhang den Einbau einer Klimaanlage, dies erschien uns jedoch für ein normales Ertrags-Gewächshaus, in dem nicht die prämierten königlichen Rosen gezüchtet werden sollen, als sehr übertrieben und ganz einfach als zu teuer und von daher als unrealistisch. Ist die Außentemperatur jedoch höher als die innerhalb unseres Gewächshauses und können wir also durch Lüften die Innentemperatur nicht senken, so wird unsere Autonome Gewächshaussteuerung mangels der Klimaanlage wohl oder übel versagen – sie kann nichts unternehmen und deshalb wird ein Alarmsignal gesetzt, das dem zuständigen Gärtner einen Hinweis auf das Geschehen gibt.

Die ***Bewässerung*** betreffend mussten wir eine Entscheidung treffen: Soll diese nach einem festlegbaren strengen Zeitplan erfolgen? Zwar um die Wahrsagekompetenz der Gärtnerschaft wissend, jedoch skeptisch ob der Vorhersagbarkeit von Hitzewellen und Dürre entschieden wir uns dagegen. Denn der erfolgreiche Gärtner hat seinen „grünen Daumen“ auch deswegen, weil er flexibel auf die Großwetterlage reagiert und sich nicht stumpf an feste Pläne hält. Ebenso sollte unsere Anlage die Bedingungen prüfen und entsprechend Wässern.
Wir lassen also die Bodenfeuchtigkeit an unser System übermitteln und diese dort mit einem festlegbaren *Sollwert* vergleichen. Ist er unterschritten, so wird die Bewässerung angestellt und eine leichte Beregnung beginnt - die Pflanzen werden also nicht ertränkt, sondern es wird, wie in Gewächshäusern üblich, ein leichter Sprühregen von über den Pflanzen angebrachten Leitungen auf diese verteilt. Ist der Feuchtigkeitswert wieder im erlaubten Bereich, so wird die Bewässerung ausgestellt.
Damit ist bei jedem Wetter und jeder Temperatur die korrekte und autonome Wasserversorgung der Pflanzen gewährleistet, denn verbrauchen sie mehr an Wasser, so sinkt automatisch die Bodenfeuchtigkeit und die Anlage beginnt zu Wässern. Sorgen eine hohe Luftfeuchtigkeit und niedrige Temperatur bereits selber für beinahe ausreichende Verhältnisse, so wird auch die Bodenfeuchtigkeit annähernd und langfristig stabil bleiben und erst nach einer entsprechenden Periode werden die Pflanzen dort das Wasser aufgenommen haben, was zur Beregnung führt.

1.3 SCHAUBILD



Legende:

Pflanzenreihen

Das Gewächshauskontrollsystem mit Steuerleitung(en) nach aussen

Wasserleitungen zur Erzeugung von „Sprühregen“

Lüfter (in Seitenwänden eingelassen)

Thermometer (eins innen, eins außen)

Feuchtigkeitsmesser

Alarmanzeige

2. ARCHITEKTURENTWURF

2.1 DATENGEWINNUNG UND -EINGABE

Als Eingabewerte sollen zum einen die Innentemperatur des Gewächshauses, die Außentemperatur, die Zeit und die Bodenfeuchte im Gewächshaus dienen.

- Temperatur:
  Innerhalb eines Gewächshauses tritt normalerweise kein sehr starkes Temperaturgefälle auf. Durch die gleichmäßige Beheizung, Belüftung und den gleichmäßigen Lichteinfall durch das Dach, herrscht im gesamten Gewächshaus ein gleichmäßiges Klima. Daher ist es ausreichend, die Temperatur an einer Stelle des Gewächshauses zu messen.
  Ähnliches gilt auch für die Außentemperatur, unter der Vorraussetzung, dass die Messwerte an einer repräsentativen Stelle entnommen werden, da die Lufttemperatur innerhalb eines relativ kleinen Gebietes, wie das einer Gärtnerei, zu einem gegebenen Zeitpunkt ebenfalls einen fast konstanten Wert aufweist. Daher wird auch hier ein Messwert als ausreichend erachtet.

  Die Messung der Innen- und Außentemperatur erfolgt mit Sensoren, welche an den Kommunikationsbus angebunden sind.

- Bodenfeuchtigkeit
  Zur Messung der Bodenfeuchte ist ebenfalls ein Messpunkt innerhalb des Gewächshauses repräsentativ, da im gesamten Gebäude das gleiche Klima herrscht und bei der ständig fortschreitenden Intensivierung der Produktion in der Regel auch eine Kultur angebaut wird.

  Die Messung erfolgt ebenfalls durch einen Sensor, der an den Kommunikationsbus angebunden ist.

- Zeit
  Eine Zeitmessung ist zur Realisierung der Verzögerungszeit vor dem Einschalten der Heizung erforderlich.

2.2 DATENTYPEN UND -FORMATE

- Innen- und Außentemperatur (6 Bit)
  Der optimale Temperaturbereich für die in Mitteleuropa am häufigsten angebauten Gewächshauskulturen (u.a. verschiedene Gemüsesorten) liegt zwischen 5°C und 40°C.
  Bei der Außentemperatur ist zwar eine größere Schwankung zu beobachten, aber Werte, die weit über bzw. unter dem Optimalbereich liegen sind insofern irrelevant für die Gewächshaussteuerung, als sie immer dieselbe Reaktion hervorrufen, wie ein Wert, der nur knapp über bzw. unter diesem liegen.
  Eine Messwertbreite von 6 Bit gestattet erlaubt Temperaturen zwischen 0°C und 63°C darzustellen, womit der gesamte optimale Wachstumsbereich überdeckt wird. Zudem ist sogar der Einsatz in Gewächshäusern, in denen bestimmten Spezialkulturen angebaut werden sollen möglich, vor allen in solchen, wo ein

höherer Temperaturbereich als im Gemüsebau gefordert ist, z.B. Botanische Gärten.
Die Temperatursensoren liefern in diesem Bereich genaue Messwerte, während eventuell auftretende höhere bzw. niedrigere Temperaturen auf die jeweiligen maximal bzw. Minimalwerte gemapped werden.

- Bodenfeuchtigkeit (6 Bit)
  Die Bodenfeuchtigkeit wird als prozentualer Wert der maximalen Sättigung des Bodens gemessen. Da eine Genauigkeit von 1% nicht notwendig ist, und die Übertragung der Daten durch den Kommunikationsbus eine Zerlegung der Daten in 3-Bit Einheiten erfordert (siehe unten) werden die Prozentwerte 0% bis 100% auf die Zahlen 0 bis 63 gemapped.

- Zeit (2Bit)
  Die Verzögerungszeit der Heizung wurde als ein fester Standardwert angenommen, so dass nur Beginn und Ende dieser Zeit bekannt sein müssen. Da sie während der gesamten Prozesssteuerung nur dazu benötigt wird, die Heizungsanlage erst nach Ablauf einer festen Verzögerungszeit einzuschalten, wäre eine externe Zeitmessung und Übertragung der Werte über den Kommunikationsbus hier mit einem großen Aufwand verbunden. Vielmehr bietet sich eine CPU interne Lösung an, in der die Zeitmessung an den Prozessortakt gekoppelt wird. Der Timer-Chip stellt dabei 2 Bit Information zur Verfügung. Erstens, wurde eine Zeitmessung gestartet und zweitens ist die Verzögerungszeit abgelaufen.

2.3 VERARBEITUNG UND SPEICHERUNG

- Übertragung der Daten (Temperatur und Bodenfeuchte) durch den Kommunikationsbus
  Das 6-Bit Format der Bodenfeuchte- und Temperaturmesswerte erlaubt eine bequeme Zerlegung in je 3-Bit, die jeweils auf demselben Kanal übertragen werden können, indem die niederen 3 Bits (3,2 und 1) mit einer 0 als führendem Markierungsbit als erstes, die höheren drei Bits (5,4 und 3) mit einer 1 als führendem Markierungsbit als zweites Paket geschickt werden. Die Zerlegung der Eingabewerte erfolgt als letzter Schritt der Datengewinnung im Sensor.

- CPU – Wortbreite (4 Bit) und internes Datenformat (8 Bit)
  Die einfachste und preisgünstigste Möglichkeit, um die Temperatur und Feuchtigkeitsdaten im Mikroprozessor mit gegebenen Minimal- , Maximal- bzw. einem Sollwert vergleichen zu können ist durch eine Subtraktion mit anschließendem Vorzeichentest oder ggf. Test auf Null realisierbar. Da beide Datentypen die Eingabebreite von 6 Bit zur Darstellung des Wertebereiches benötigen, muss sie um mindestens 1 Bit vergrößert werden, um negative Werte darstellen zu können. Die Entscheidung acht Bit zu benutzen beruht darauf, dass die Addition und Subtraktion von 8 Bit Werten sequenziell in 4-Bit Blöcken erfolgen kann und somit ein Mikroprozessor mit einer Wortbreite von 4 Bit ausreicht um diese Operationen durchzuführen. Dadurch werden die Hardwarekosten gesenkt.

2.4 DATENAUSGABE

- Heizung, Lüftung, Bewässerung und Alarm

  Alle drei Anlagen sind durch den Kommunikationsbus an den Mikroprozessor angebunden. Der Steuerungsmechanismus ist dafür verantwortlich sie Ein- bzw. Auszuschalten. Dafür muss nur zwischen zwei diskreten Werten, einer für Einschalten, einer für Ausschalten unterschieden werden, was sich durch den 4 Bit breiten Kommunikationsbus problemlos realisieren lässt.

3. MIKROPROZESSORUMGEBUNG

3.1.SCHNITTSTELLEN

- 4-bit-Kommunikationsbus für die Anbindung an Sensoren und Aktuatoren über Sende- und Empfangseinheit
- 12-bit-Adressbus und 4-bit-Datenbus zur Anbindung an den RAM-Speicher
- serieller Kommunikationsbus zur Verbindung von Sensoren und Aktuatoren an Sende- und Empfangseinheit.
    - Daten werden in 4-bit-Blöcken durch den Bus gesendet. Dabei immer abwechselnd Kanalnummer 1, Datensatz 1,
      Kanalnummer 2,
      ...
      Kanalnummer 7, Datensatz 7,
      Kanalnummer 1, Datensatz 1,
      ...

3.2.SENSOREN UND AKTUATOREN

AUSGABEN:

- Alarm: Leuchtsignal, gesteuert durch Zustands-Flip-Flop.
- Wasser, Heizung, Lüftung: Ein-/Ausschalter für die jeweiligen Geräte.

EINGABEN:

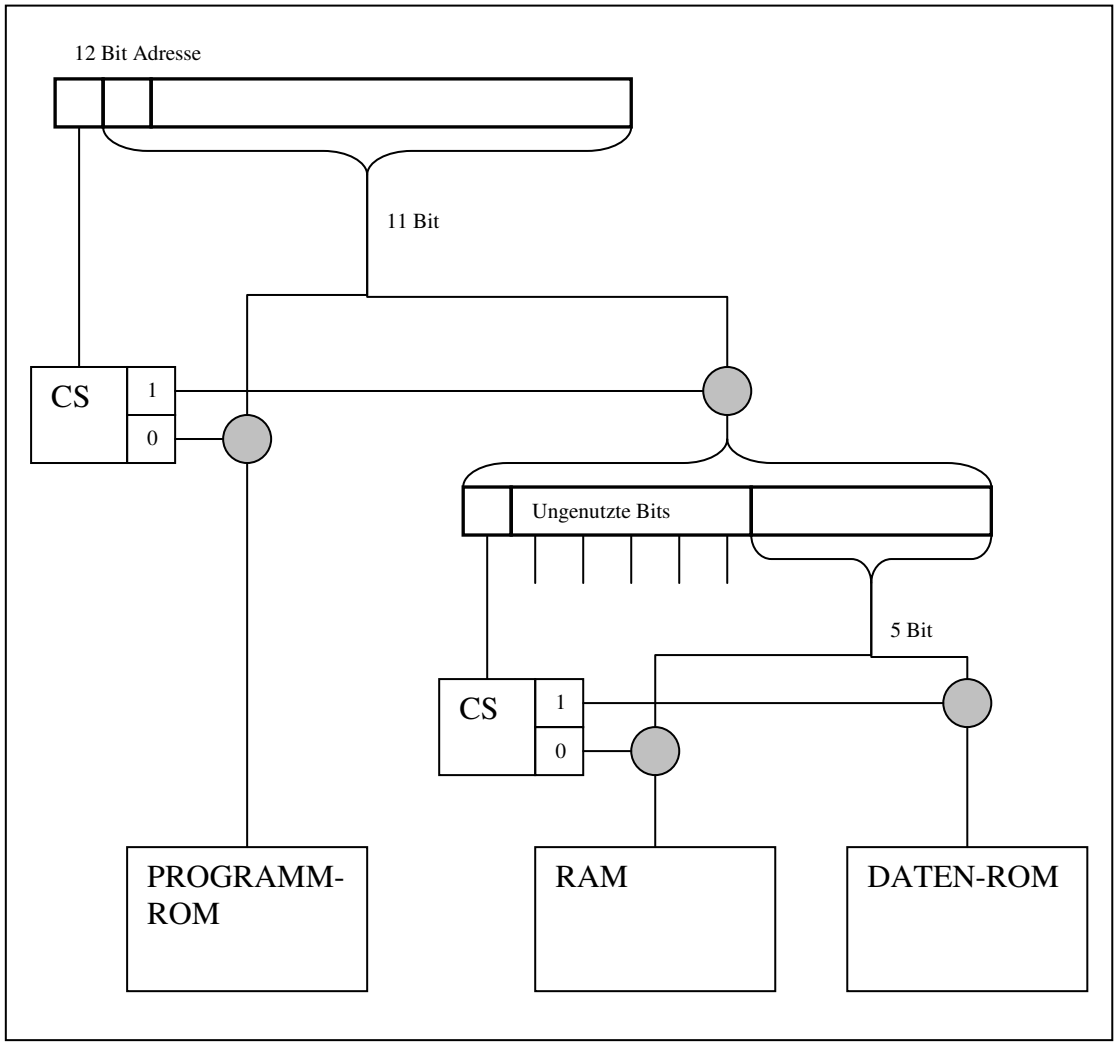
- T_{innen} (Innentemperatur): Temperaturfühler
- $T_{außen}$ (Außentemperatur): Temperaturfühler
- H (Bodenfeuchtigkeit): Feuchtigkeitssensor

3.3. WEITERE KOMPONENTEN

- Sende- und Empfangseinheit (laut Vorgabe)
- Speicher:
  Der Speicher ist in 3 Teile unterteilt, den Programm-ROM, den Daten-ROM, sowie den RAM, welche durch ein gemeinsames Adressformat von 12 Bit Wort- adressierbar sind, wobei ein Wort 4-Bit hat.
  Im Programm-ROM ist das gesamte Maschinenprogramm des Steuerungsalgorithmus hinterlegt. Er beginnt bei Speicheradresse $0000\ 000\ 0000_2$, was auch die Startadresse des Programms ist und umfasst 2^{11} Adressen a 4 Bit. Insgesamt steht also 1kByte Wort-adressierbarer Programmspeicher zur Verfügung. Dieser ist auch notwendig, da das Maschinenprogramm des Steueralgorithmus 705 Byte groß ist. Der Programm-ROM ist als EPROM ausgeführt, um die Möglichkeit eines Steuerprogramm-Updates offenzuhalten.
  Der Daten-ROM ist dafür gedacht, die Minimal- und Maximal- Temperatur Parameter, sowie den Sollwert für die Bodenfeuchtigkeit aufzunehmen. Da diese Parameter von Pflanzenart zu Pflanzenart variieren, ist er als auswechselbarer EPROM Chip ausgeführt, so dass die Werte angepasst werden können. Die 3 Werte belegen lediglich 6 Adressen (je 2 Worte). Somit wären 4 Byte (8 Adressen) eigentlich völlig ausreichend. Im Hinblick auf zukünftige Erweiterbarkeit wurde jedoch entschieden 16 Byte zu benutzen.

Der RAM dient als flüchtiger Schreib- Lese Speicher der temporären Datenspeicherung. Auch hier kann darauf verzichtet werden, die zur Verfügung stehende Speicherbandbreite auszuschöpfen – 16 Byte (32 Adressen) sind auch hier ausreichend.

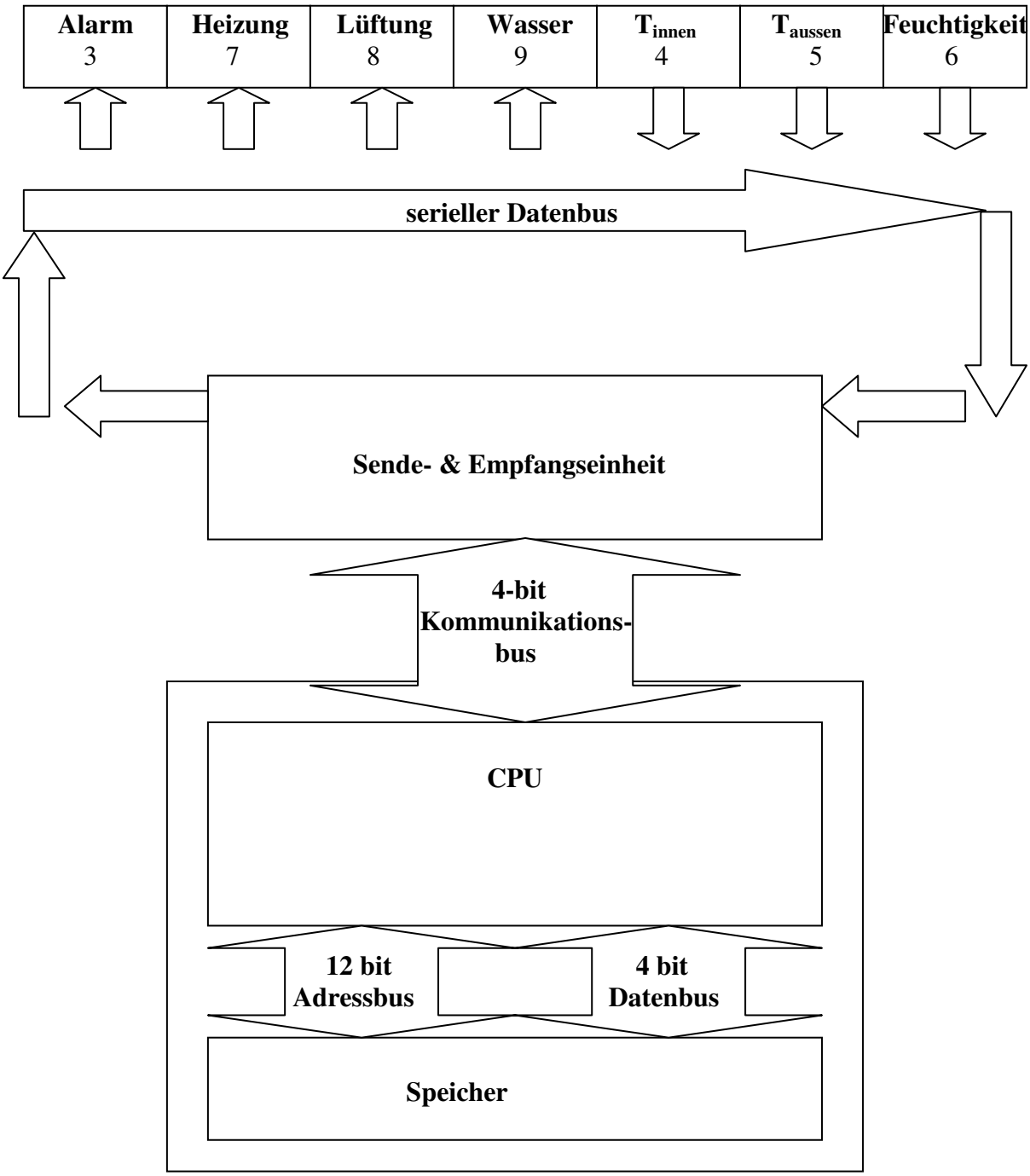
Funktionsweise des Speichers:
Durch höchste Bit der 12-Bit Adresse, Bit 11, wird erkannt, ob auf den Programm-ROM, oder einen der beiden anderen Speicherbereiche zugegriffen werden soll. Ist es nicht gesetzt (0) werden die folgenden 11 Bit als Adresse im Programm-ROM ausgewertet, indem durch das Cable-Select (CS) die Adressleitungen zum Programm ROM freigegeben werden.
In gleicher Weise erfolgt die Unterscheidung, ob auf den RAM oder den Daten-ROM zugegriffen werden soll. Ist bei gesetztem 11. Bit das 10. Bit ebenfalls gesetzt, wird auf den Daten-ROM, ist es 0 auf den RAM zugegriffen. Da RAM und Daten-ROM jeweils nur 16 Byte groß, also jeweils mit 5 Bit vollständig adressiert werden können, werden jeweils nur die Bits 4 bis 0 zur Adressierung genutzt.



Adressierung des Speichers

Die CPU ist mittels des 12-bit-Adressbus der am Memory-Adress-Register (MAR) anliegt, das aus den drei 4-Bit Teilregistern MAR2, MAR1, MAR0 besteht, und dem 4-bit-Datenbus an Memory Data Register (MDR) an den Speicher angebunden. Außerdem wird über das 1 Bit Signal MFC (Memory Fetch Complete) angezeigt, wenn nach einer Leseanforderung an den Speicher die Daten in das MDR geschrieben wurden. Die MFC Leitung wird in diesem Fall auf 1 gesetzt, sonst auf 0.

3.4.SCHAUBILD



4. MIKROPROZESSOR

4.1. SPEZIFIKATIONEN (Prozessorkomponenten und Funktionen)

4.1.1. ALU-Funktionen

- Es wird eine 4-bit-ALU verwendet mit folgenden ALU-Operationen:

  F = A + B + C — A_{in}, B_{in}, Carry C werden addiert, Ergebnis nach Z_{out}

  F = A xor B — A_{in}, B_{in} werden XOR-verknüpft, Ergebnis nach Z_{out}

  F = shiftleft A — A_{in} wird links-geshiftet, dabei werden leere Stellen mit Nullen gefüllt, Ergebnis nach Z_{out}

  F = shiftright A — A_{in} wird rechts-geshiftet, dabei werden leere Stellen mit Nullen gefüllt, Ergebnis nach Z_{out}

  ALU-Flags

  C — 1-Bit: Carry-Bit, wird gesetzt, wenn bei ALU-Operation ein Übertrag entsteht.

  V — 1-Bit: V-Flag (Vorzeichen) wird gesetzt, wenn MSB des Akku 1 ist.

  Z — 1-Bit: Z-Flag (Zero Flag) gesetzt, falls der Wert im Akku der Alu 0 ist

4.1.2. IR, MAR, PC

- Die Adressen des Speichers sollen 12 bit lang sein. Der CPU-interne Bus und die Alu können nur 4-bit-Blöcke transportieren bzw. verarbeiten. Daher ist es notwendig, überall dort, wo in der CPU mit Adressen gearbeitet wird, diese 12-bit-Blöcke in je 3 4-bit-Teile zu zerlegen und in der richtigen Reihenfolge zu verarbeiten bzw. nach außen weiterzugeben.
- Besondere Beachtung gilt hierbei dem Carry beim Incrementieren.
- Aufteilung der 12 Adressbits:

msb lsb

Register 2	Register 1	Register 0

11, ... 8, 7, ... 4, 3, ... ,0

4.1.3. General Purpose Register R0–R7, Memory Data Register MDR

- R0 bis R7 sind 4-Bit General Purpose Register mit Lese- und Schreibzugriff, die für die Instruktionen als Operanden zur Verfügung stehen.
- Das Memory-Data-Register (MDR) dient als Schnittstelle zwischen CPU und Arbeitsspeicher. Zum Lesen eines Wertes aus dem Speicher wird dessen 12-Bit Speicheradresse in die 3 Register des MAR eingetragen, woraufhin der Wert von der Speicherstelle in das MDR übertragen wird. Anschließend wird das Signal Memory-Fetch-Complete auf 1 gesetzt. Zum Schreiben eines 4-Bit Wertes im Speicher wird er

in das MDR geschrieben und ebenfalls die Adresse in die MAR geschrieben. Das MDR wird steht nur CPU intern zur Verfügung.

4.1.4. Memory Fetch Complete (MFC)

- Das Memory-Fetch-Complete (MFC) ist eine 1-Bit Signalleitung. Sie wird auf 1 gesetzt, um anzuzeigen, dass ein 4-Bit Datenwort aus dem Speicher in das MDR fertig geschrieben wurde.

4.1.5. transparente Hilfsregister RS1, RS2

- Als Puffer zur Zwischenspeicherung von Speicheradressen stehen die von außen transparenten 4-Bit Register RS1, RS2 zur Verfügung. Sie werden nur intern von den Befehlen: MOVEMR, MOVERM, BGZ, BEZ, BRA verwendet. Da jede Adresse 12 Bit breit ist, werden 3 Speicherabfragen benötigt, um die Adresse aus dem Speicher zu laden. Deshalb darf während dieser Zeit der PC (Sprungbefehle) bzw. das MAR (MOVE-Befehle) nicht überschrieben werden und die ersten beiden 4-Bit Blöcke müssen temporär gespeichert werden. Somit wird vermieden, dass General-Purpose-Register zur Zwischenspeicherung benutzt werden, welche möglicherweise mit noch benötigten Daten belegt sind.

4.1.6. „Set-to-1111“

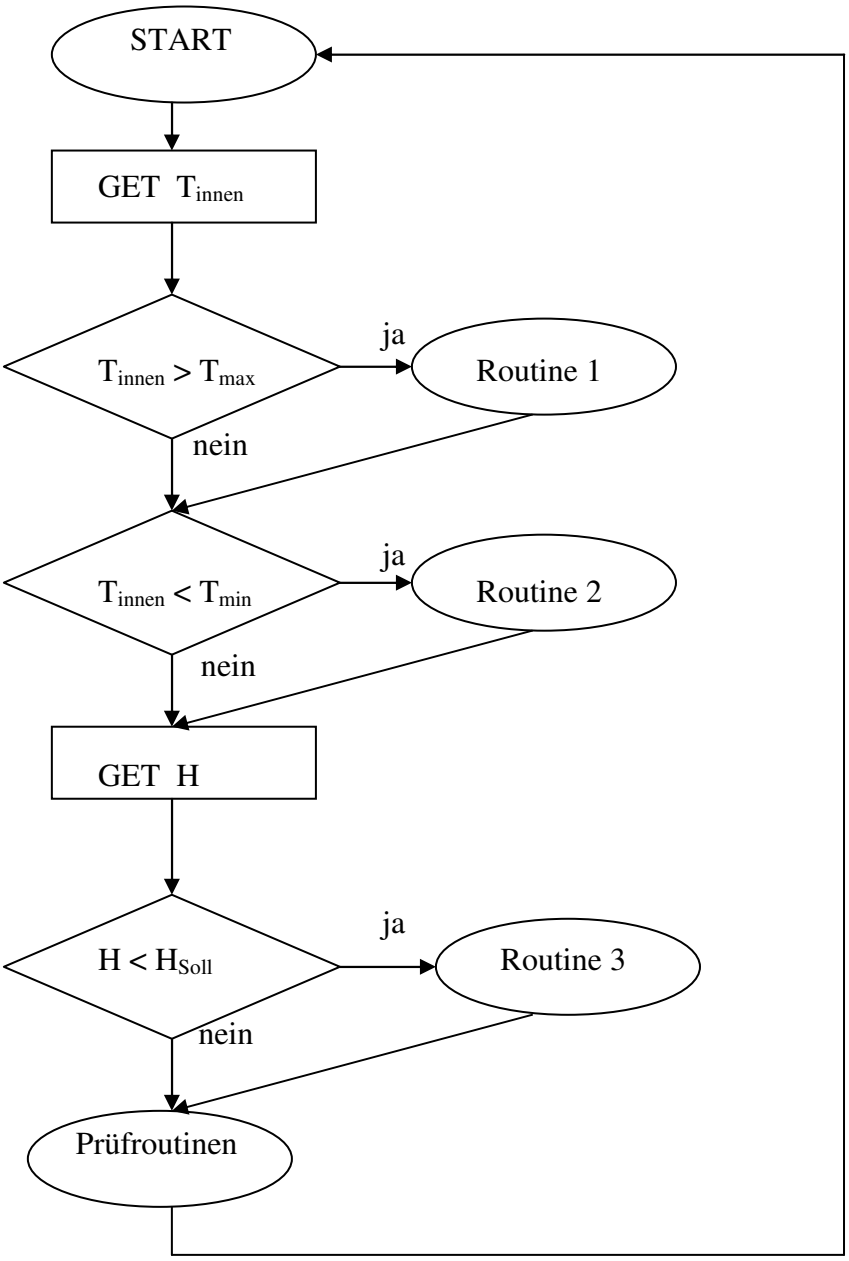
- Das Set-to-1111 Register ist ein von außen transparentes 4-Bit Register, das den 4-Bit Wert $15_{10} = 1111_2$ auf den internen Bus schreibt. Dieser wird für den Befehl NEG benötigt.

4.1.7. Timer und TimerRegister (TR)

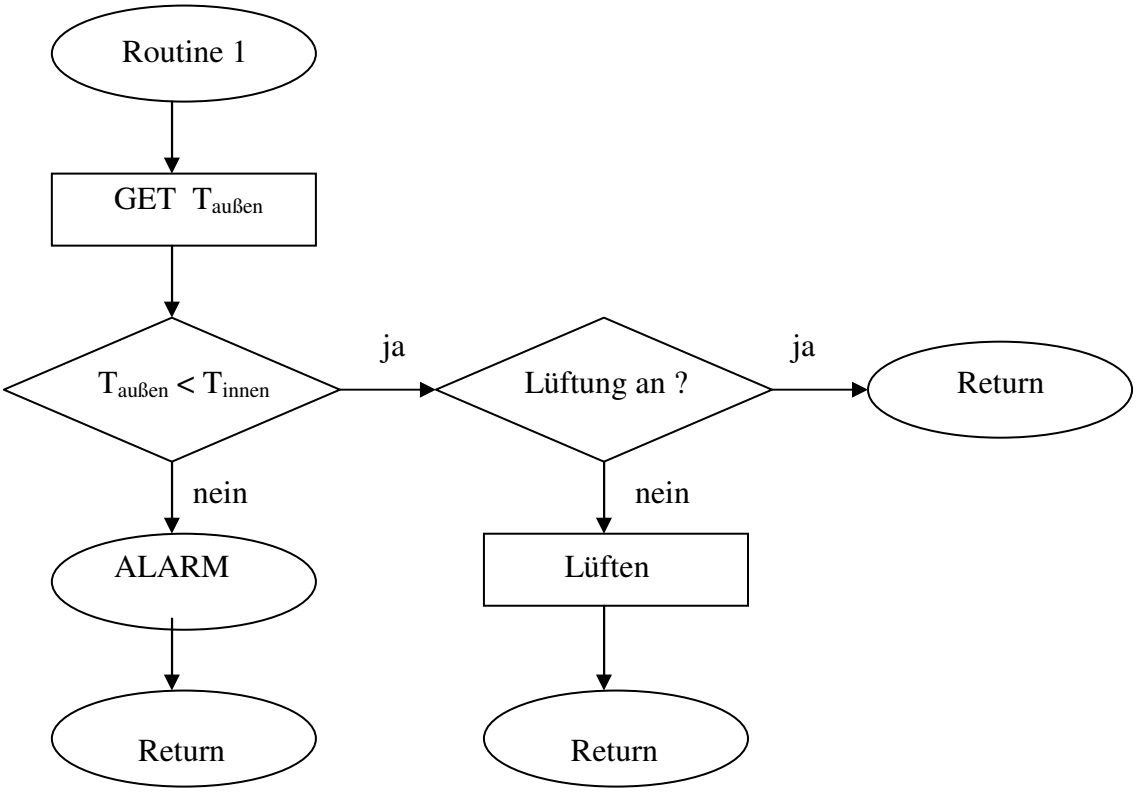
- Der Timer ist dafür verantwortlich, die Verzögerungszeit vor dem Einschalten der Heizung zu realisieren. Er kann durch das Steuerungsprogramm explizit gestartet und gestoppt werden. Wird er durch den Assembler-Befehl STARTTM gestartet, so wird das LSB (Bit 0) des TR auf eins gesetzt. Falls er nicht durch STOPTM gestoppt wird, wobei das Timerregister auf 0_{10} zurückgesetzt wird, wird nach Ablauf der vorgegebenen Zeitspanne das Bit 1. auf 1 gesetzt, um anzuzeigen. Das Timerregister ist ein 4 Bit Register, dessen höhere Bits immer auf 0 gesetzt sind, während die unteren mit dem Timer verbunden sind und in der oben beschriebenen Weise benutzt werden. Das TR kann mit dem Befehl READTM gelesen werden.
- Die interne Zeitmessung des Timers könnte durch einen an den Takt gekoppelten Zähler realisiert werden.

4.2. ABLAUFDIAGRAMME

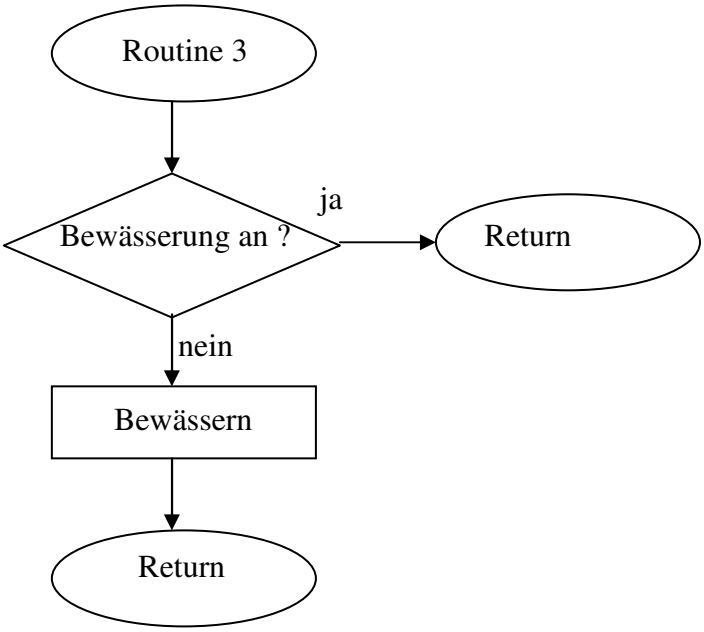
4.2.1. Hauptprogramm:



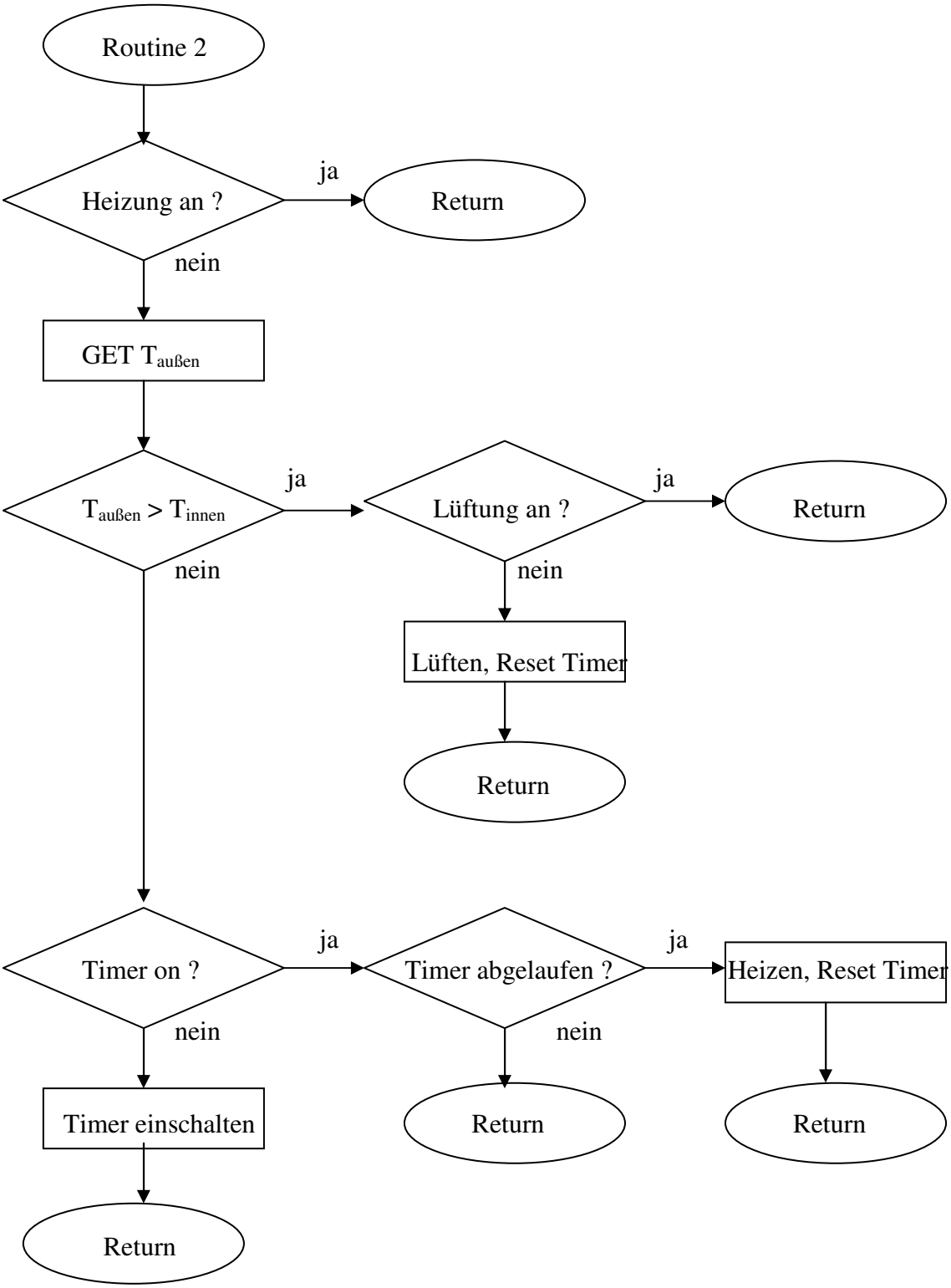
4.2.2. Routine 1 / Lüften (Temperatur ist zu hoch)



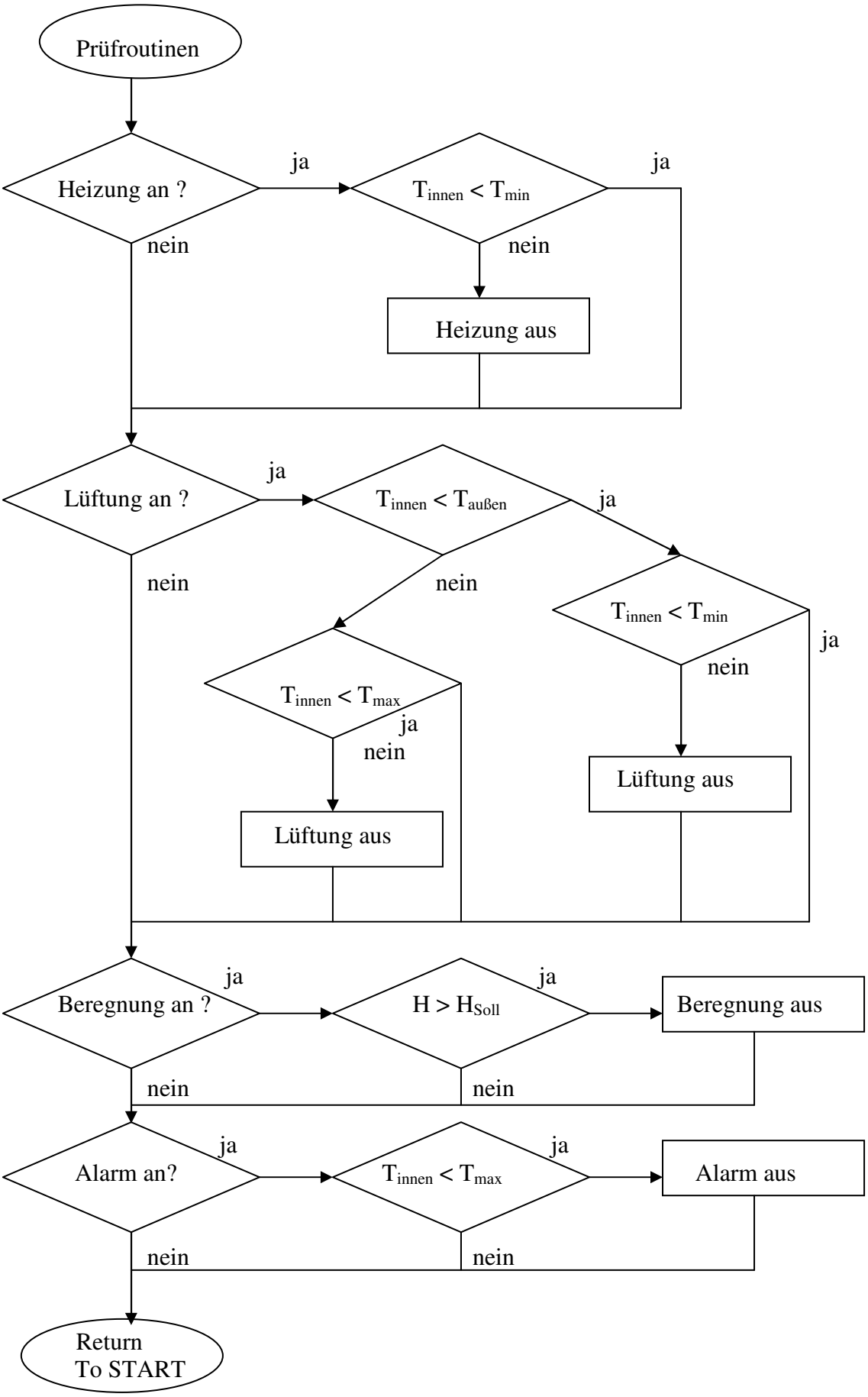
4.2.3. Routine 3 / Bewässern (Boden ist zu trocken)



4.2.4. Routine 2 / Heizen (Temperatur ist zu niedrig)



4.2.5. Prüfroutinen / Abschalten



4.3. Datentypen und –formate

- Speicher-Adressen: 12 bit als Erweiterung an das Befehlswort angehängt
  werden verwendet bei:
  - Speicher- Schreib/Lese -befehlen (MOVERM, MOVEMR)
  - Sprungbefehlen (BEZ, BGZ, BRA)

- Immediate-Werte: 4 bit als Erweiterung an das Befehlswort angehängt
  - Nur bei SET möglich.
- Register-Adressen: 3 bit max. 2 im Befehlswort enthalten
  - werden im Maschinencode durch die binäre binäre Repräsentation ihrer Nummer kodiert
- Keine explizite Angabe des Adressmodus nötig, da wenn verschiedene Adressmodi, dann extra Befehl.

4.4. Befehlsformate

Der Mikroprozessor wurde als 2 Adress-Maschine konzipiert.

4.4.1. NULL-Operanden-befehle

6 bit Opcode		

- STOPTM :stoppt den Timer
- STARTTM :startet den Timer

4.4.2. EIN-Operanden-befehle

6 bit Opcode	3 bit Register	

- SHL RDEST :shift left in RDEST, auffüllen mit Nullen
- SHR RDEST :shift right in RDEST, auffüllen mit Nullen
- READTM RDEST :liest den Timer aus und schreibt Ergebnis in RDEST

6 bit Opcode		
12 bit Speicheradresse		

- BGZ ADRESSE :springt an ADRESSE, wenn Ergebnis von vorhergehender Operation positiv (ALU-V-Flag nicht gesetzt)
- BRA ADRESSE :unbedingter Sprung an ADRESSE
- BEZ ADRESSE :springt an ADRESSE, wenn Ergebnis von vorhergehender Operation Null (ALU-Z-Flag gesetzt)

4.4.3. ZWEI-Operanden-befehle

6 bit Opcode	3 bit Register	3 bit Register

- GET : empfängt Kanaldaten
- SEND : sendet Kanaldaten
- ADD : addiert und setzt ALU-Carry
- NEG : bildet aus einer 8-bit-Zahl das Zweierkomplement
- MOVERR : kopiert von Register in Register

6 bit Opcode	3 bit Register	
12 bit Speicheradresse		

- MOVERM : kopiert von Register in Memory
- MOVEMR : kopiert von Memory in Register

6 bit Opcode	3 Bit Register	
4 Bit Direktoperand		

- SET RDEST, Wert : Lädt den Direktoperanden Wert in das Register RDEST

4.5. Befehlssatz

4.5.1. Assemblerbefehle und Bedeutung

ADD RSOURCE, RDEST

- RSOURCE, RDEST : Register
- Funktionsweise : RDEST = RDEST + RSOURCE
- Setzt : Carry Bit falls Ergebnis Source + Dest > 15
  Z-Flag: falls RDEST = 0
  V-Flag: falls MSB des RDEST = 1

NEG RHIGH, RLOW

- RHIGH, RLOW: Register
- (RHIGH,RLOW) = Zweierkomplement(RHIGH,RLOW)
- zur Negierung einer positiven 8 Bit Zahl, deren Teile in den beiden 4 Bit Registern gespeichert ist, durch Zweierkomplementbildung
- es erfolgt keine Prüfung, ob RHIGH und RLOW verschieden sind
- Setzt: Z-Flag: falls RHIGH = 0
  V-Flag: falls MSB des RHIGH = 1

SHL REG

- REG: Register
- REG = SHL(REG)
- Bitweises Shift nach links, dabei wird von links mit 0 aufgefüllt

SHR REG

- REG: Register
- REG = SHR(REG)
- Bitweises Shift nach rechts, dabei wird von rechts mit 0 aufgefüllt

SET RDEST, #Wert

- Schreibt den Direktoperanden Wert in das Register RDEST

MOVEMR RDEST, SOURCE

- Die Daten aus der Speicherstelle mit Adresse SOURCE werden nach RDEST geschrieben.

MOVERM RSOURCE, DEST

- Die Daten aus RSOURCE werden an die Speicherstelle mit Adresse DEST geschrieben.

MOVERR RSOURCE, RDEST

- RSOURCE, RDEST: Register
- Die Daten aus RSOURCE werden nach RDEST geschrieben.

BRA Zieladresse

- (Unbedingter Sprung)
- Springt an die Zieladresse

BGZ Zieladresse

- Springt an die Zieladresse, falls ALU-Flag V = 0 ist.

BEZ Zieladresse

- Springt an die Zieladresse, falls ALU-Flag Z = 1 ist.

STOPTM

- Der Timer wird angehalten
- Das Timerregister (TR) wird auf 0000 gesetzt

READTM RDEST

- Das Timerregister (TR) wird nach RDEST geschrieben

STARTTM

- Der Timer wird gestartet
- Das Timerregister (TR) wird auf 0001 gesetzt

SEND RKanal, RDaten

- RKanal, RDaten: Register
- sendet Daten aus RDaten an das Empfangsgerät dessen Kanalnummer in RKanal angegeben ist.

GET RKanal, RDaten

- RKanal, RDaten: Register
- schreibt das nächste in der Komunikaitionseinheit verfügbare Kanal-Daten Paar in die Zielregister RKanal, RDaten

4.5.2. OPCODE - Decodierung

MNEMONIC	**x_5**	**x_4**	**x_3**	**x_2**	**x_1**	**x_0**
ALU-OPs	**1**				**0/1**	**0/1**
ADD	1	x	x	x	0	0
NEG	1	x	x	x	0	1
SHL	1	x	x	x	1	0
SHR	1	x	x	x	1	1
Bewg. Daten	**0**	**1**			**0/1**	**0/1**
SET	0	1	x	x	0	0
MOVEMR	0	1	x	x	0	1
MOVERM	0	1	x	x	1	0
MOVERR	0	1	x	x	1	1
Branching	**0**	**0**	**1**		**0/1**	**0/1**
BRA	0	0	1	x	0	0
BGZ	0	0	1	x	0	1
BEZ	0	0	1	x	1	0
Timer-OPs	**0**	**0**	**0**	**1**	**0/1**	**0/1**
STOPTM	0	0	0	1	0	0
READTM	0	0	0	1	1	0
STARTTM	0	0	0	1	1	1
Rest	**0**	**0**	**0**	**0**	**0/1**	
SEND	0	0	0	0	0	x
GET	0	0	0	0	1	x

- Theoretisch könnten 64 – 16, also 48 weitere Assemblerbefehle kodiert werden.
- Durch die gewählte Codierung ist jedoch eine sehr einfache Decodierung der Befehle möglich, was die Kostensenkung der Hardware beiträgt, wie man sieht ist etwa zum Erkennen eines ALU-Befehls nur die Kontrolle von drei Bits nötig, um Daten zu bewegen nur von vier.

4.6. Steuereinheit

4.6.1. Mikrocodierungen der Assemblerbefehle

Außer bei den Befehlen *get* und *send*, muss in jeder Mikrocodezeile das Low-Aktive Signal *Auswahl* auf 1 gesetzt sein, um die Sende-Empfangs Einheit abzuschalten. Es wurde der Übersichtlichkeit halber weggelassen.

Fetching-Makro (wird zum Laden neuer Befehle ausgeführt)

$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $IR2_{in}$
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $IR1_{in}$
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $IR0_{in}$, Reset Carry

GET Kanalziel, Datenziel
[FETCH]
Richtung, $Kanalziel_{in}$
Richtung, Adresse, $Datenziel_{in}$

SEND Kanalquelle, Datenquelle
[FETCH]
$Kanalquelle_{in}$
Adresse, $Datenquelle_{in}$
Wait for „Senden beendet“

ADD Source, Destination (Dest = Dest + Source)
[FETCH]
RS_{out}, X_{in}
RD_{out}, A_{in}, B_{in}, ALU-ADD, Z_{in}
Z_{out}, RD_{in}

SHL Register
[FETCH]
R_{out}, ALU-SHL, A_{in}, Z_{in}
Z_{out}, R_{in}

SHR Register
[FETCH]
R_{out}, ALU-SHL, A_{in}, Z_{in}
Z_{out}, R_{in}

NEG RegisterHigh, RegisterLow
[FETCH]
RL_{out}, X_{in}
Set to 1111_{out}, A_{in}, B_{in}, ALU-XOR, Z_{in}
Z_{out}, RL_{in}
RH_{out}, X_{in}
Set to 1111_{out}, A_{in}, B_{in}, ALU-XOR, Z_{in}
Z_{out}, RH_{in}
Reset X, Set Carry, RL_{out}, A_{in}, ALU-ADD, Z_{in}
Z_{out}, RL_{in}
Reset X, RH_{out}, A_{in}, ALU-ADD, Z_{in}
Z_{out}, RH_{in}, Reset Carry

STOPTM
[FETCH]
Stoptm, ResetTR

STARTTM
[FETCH]
Starttm

READTM RDEST
[FETCH]
TR_{out}, $RDEST_{in}$

SET RDEST, Wert
[FETCH]
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $RDEST_{in}$

MOVERR RSOURCE, RDEST
[FETCH]
$RSOURCE_{out}$, $RDEST_{in}$

MOVERM RSOURCE, DEST

[FETCH]
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $RS1_{in}$
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $RS0_{in}$
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $MAR0_{in}$, Reset Carry
$RS0_{out}$,$MAR1_{in}$
$RS1_{out}$,$MAR2_{in}$
Wait for MFC
$RSOURCE_{out}$,MDR_{in}

MOVERM RDEST, SOURCE

[FETCH]
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $RS1_{in}$
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $RS0_{in}$
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $MAR0_{in}$, Reset Carry

$RS0_{out}$,$MAR1_{in}$
$RS1_{out}$,$MAR2_{in}$
wait for MFC
MDR_{out}, $RDEST_{in}$

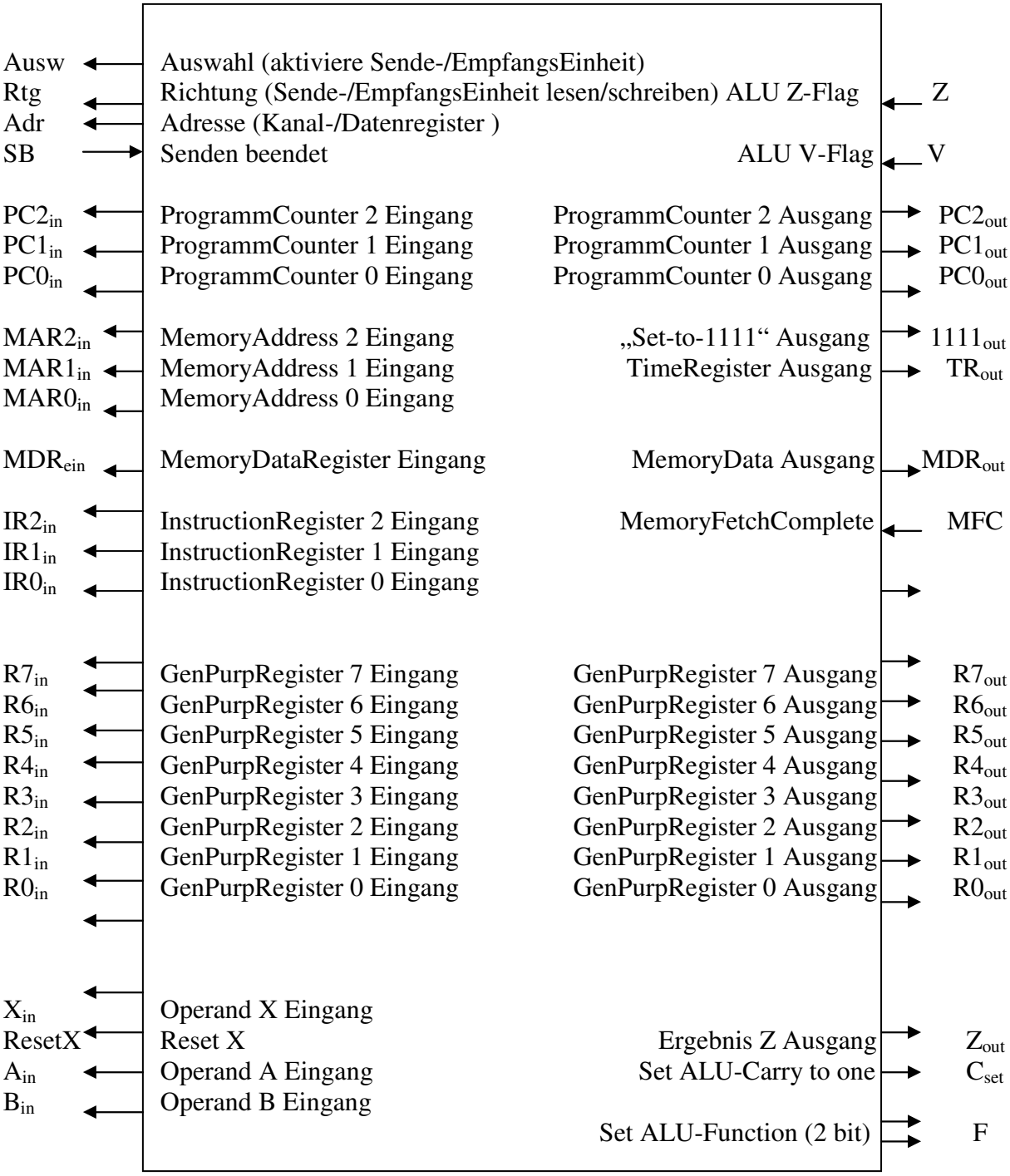
BGZ Zieladresse
[FETCH]
Falls V=0 : weiter; sonst µJmp nach END
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $RS1_{in}$
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $RS0_{in}$
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $PC0_{in}$, Reset Carry
$RS0_{out}$, $PC1_{in}$
$RS1_{out}$, $PC2_{in}$
END

BRA Zieladresse
[FETCH]
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $RS1_{in}$
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $RS0_{in}$
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$

$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $PC0_{in}$, Reset Carry
$RS0_{out}$, $PC1_{in}$
$RS1_{out}$, $PC2_{in}$

BEZ Zieladresse
[FETCH]
Falls Z=1 : weiter; sonst µJmp nach END
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $RS1_{in}$
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $RS0_{in}$
$PC0_{out}$, $MAR0_{in}$, A_{in}, Reset X, Set Carry, ALU-ADD, Z_{in}
Z_{out}, $PC0_{in}$
$PC1_{out}$, $MAR1_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC1_{in}$
$PC2_{out}$, $MAR2_{in}$, A_{in}, Reset X, ALU-ADD, Z_{in}
Z_{out}, $PC2_{in}$, Warten auf MFC
MDR_{out}, $PC0_{in}$, Reset Carry
$RS0_{out}$, $PC1_{in}$
$RS1_{out}$, $PC2_{in}$
END

4.6.2. Steuereinheit – Schaubild



- Kodierung der ALU Befehle:

  Um die Anzahl der Steuerleitungen zu reduzieren werden die 4 ALU-Befehle in 2 Bit kodiert $F=F_1F_0$. Sie werden dann in der ALU wieder dekodiert.

Microcode Bezeichnung	ALU-Befehl	F_1F_0
ALU-ADD	F = A + B + C	00
ALU-XOR	F = A xor B	01
ALU-SHL	F = A shl 1	10
ALU-SHR	F = A shr 1	11

4.7. Assemblercode

Zur Darstellung des Assemblercodes werden Makros verwendet, die selbst wieder in Assemblerbefehle aufgeschlüsselt sind. (Führenden Zahlen im Hauptprogramm sind Label von Speicherstellen)
Sollwerte befinden sich an festen, reservierten Speicherstellen und werden zusammen mit dem Programm dorthin geschrieben.
Messwerte werden an ebenfalls feste, reservierte Speicherstellen während des Programmablaufs geschrieben, um Register freizuhalten.

4.7.1. Reservierte Speicherzellen für Soll- und Messwerte

L bzw. H markieren jeweils, ob es sich um die höherern(H) oder niederen(L) 4 Bits des Wertes handelt.

- Daten-Rom (Adressbereich: 3103 bis 3072)
  Minimale Innentemperatur: T_{min} : 3098 LT_{min} 3099 HT_{min}
  Maximale Innentemperatur: T_{max} : 3100 LT_{max} 3101 HT_{max}
  Optimalwert der Bodenfeuchte: H_{Soll}: 3102 LH_{Soll} 3103 HH_{Soll}

- RAM (Adressbereich: 2048 - 2079)
  Aktuelle Aussenteperarur: $T_{außen}$: 2070 LT_{aussen} 2071 $HT_{außen}$
  Aktuelle Innentemperatur: T_{innen} : 2072 LT_{innen} 2073 HT_{innen}
  Aktuelle Bodenfeuchte: H_{ist}: 2074 LH_{ist} 2075 HH_{ist}

  Zustände der jeweiligen Ausgabegeräte:
  M_ALARM 2076 an = 1111 / aus = 0000
  M_Heizung 2077 an = 1111 / aus = 0000
  M_Lüftung 2078 an = 1111 / aus = 0000
  M_Wasser 2079 an = 1111 / aus = 0000

4.7.2. Verwendete Makros

MAKRO #GET_VALUE (Kanal: Kanalnummer; MEM_H, MEM_L: 4Bit Speicheradressen)

```
Anf_L:    GET R1, R3
          SET R5, #Kanal
          SET R0, #0
          SET R4, #0
          ADD R1, R5
          ADD R0, R4
          BEZ Kan_OK
          BRA Anf_L
Kan_OK:   ADD R0, R3          //R3 <= R3 + 0
          BGZ fnd_Low
          BRA Anf_L
fnd_Low:  MOVERR R3, R7       //R7 <= Daten (0 und untere 3 Bit)  = [0x2x1x0]
Anf_H:    GET R1, R3
          SET R5, #Kanal
          SET R0, #0
          SET R4, #0
          ADD R1, R5
          ADD R0, R4
          BEZ Kan_OK
          BRA Anf_H
Kan_OK:   ADD R0, R3          //R3 <= R3 + 0
          BGZ Anf_H
          BEZ Anf_H
          MOVERR R3, R6       R6 <= Daten (1 und Bits 5,4,3)  = [1x5x4x3]
          MOVERR R6, R0
          SHL R0              R0 = [x5x4x30]
          SHL R0              R0 = [x5x4x30]
          SHL R0              R0 = [x3000]
          ADD R0, R7          R7 = [x3000] + [0x2x1x0] = [x3x2x1x0]
          MOVERR R6, R0
          SHL R0              R0 = [x5x4x30]
          SHR R0              R0 = [0x5x4x3]
          SHR R0              R0 = [00x5x4]
          MOVERM R7, MEM_L
          MOVERM R6, MEM_H
```

Das Makro liest die nacheinander auf dem übergebenen Kanal eintreffenden, kodierten 4-Bit Daten, dekodiert sie und schreibt sie an als 8 Bit – Wert an die Speicherstellen MEM_H, MEM_L, wobei die höheren 4 Bit in MEM_H, die niedrigeren in MEM_L gespeichert werden.

MAKRO #SWITCH_ON (Kanal: Kanalnummer, MEM_K: Speicherstelle)

```
SET R0, #15
SEND #Kanal, R0
MOVERM R0, MEM_K
```

Das Makro sendet “15” auf dem Kanal mit der Nummer „Kanal“, um den zugehörigen Aktuator (Heizung, Lüftung, Bewässerung) einzuschalten. Der gesendete Wert „15“ wird außerdem an der zugehörigen Speicherstelle MEM_K (M_Heizung, M_Lüftung, M_Wasser) abgelegt.

MAKRO # SWITCH_OFF (Kanal: Kanalnummer, MEM_K: Speicherstelle)

```
SET R0, #15
SEND #Kanal, R0
MOVERM R0, MEM_K
```

Das Makro sendet „0” auf dem Kanal mit der Nummer „Kanal“, um den zugehörigen Aktuator (Heizung, Lüftung, Bewässerung) auszuschalten. Der gesendete Wert „0“ wird 8außerdem an der zugehörigen Speicherstelle MEM_K (M_Heizung, M_Lüftung, M_Wasser) abgelegt.

MAKRO #CHECK_STATE(MEM_K, M_ON, M_OFF: Speicherstellen)

```
MOVEMR R0, #MEM_K
SET R1, #1
ADD R0, R1
BEZ on
BRA off
```

Das Makro überprüft, ob ein Aktuator angeschaltet wurde, indem geprüft wird, ob der Wert der zugehörigen Speicherstelle MEM_K (= M_Heizung, M_Lüftung, M_Wasser) „15“ ist. Falls ja wird die Programmausführung an der übergebenen Adresse M_ON, sonst bei M_OFF, fortgesetzt.

MAKRO #CHECK_LT(X_1, X_0, Y_1, Y_0, ja, nein: Speicheradressen)

```
MOVEMR R0, X1
MOVEMR R1, X0
MOVEMR R2, Y1
MOVEMR R3, Y0
NEG R0, R1
ADD R1, R3
ADD R0, R2
BGZ ja
BRA nein
```

Das Makro überprüft, ob die in X_1, X_0 übergebene 8-Bit Zahl X kleiner als die in Y_1, Y_0 übergebene 8-Bit Zahl Y ist. Falls ja, Sprung nach Adresse „ja“, sonst nach „nein“.

4.7.3. Assemblerprogramm mit Makros:

```
1   #GET_VALUE(#4, LT_innen , HT_innen)
2   #CHECK_LT(LT_max, HT_max, LT_innen , HT_innen, 7,  3)
3   #CHECK_LT(LT_innen , HT_innen, LT_min, HT_min, 12, 4)
4   #GET_VALUE(#6, HH_ist, LH_ist)
5   #CHECK_LT(HH_ist, LH_ist, HH_soll, LH_soll, 37, 6)
6   BRA 40
7   #GET_VALUE(#5, LT_außen , HT_aussen)
8   #CHECK_LT(LT_außen , HT_außen, LT_innen , HT_innen, 9,  56)
9   #CHECK_STATE(M_Lüftung, 10, 3)
10  #SWITCH_ON(#8, M_Lüftung)
11  BRA 3
12  #CHECK_STATE(M_Heizung, 13, 4)
13  #GET_VALUE(#5, HT_außen, LT_aussen)
14  #CHECK_LT(LT_innen , HT_innen, LT_außen , HT_außen, 15,  19)
15  #CHECK_STATE(M_Lüftung, 16, 4)
16  #SWITCH_ON(#8, M_Lüftung)
17  STOPTM
18  BRA 4
19  READTM R0
20  MOVERR R0, R1
21  SHL R1
22  SHL R1
23  SHL R1
24  SET R2, #0
25  ADD R1, R2
26  BGZ 28
27  BRA 30
28  STARTTM
29  BRA 4
30  SHL R0
```

Hauptprogramm

Routine 1

Routine 2

```
31  SHL R0
32  ADD R0, R2
33  BGZ 4
34  #SWITCH_ON(#7, M_Heizung)
35  STOPTM
36  BRA 4
37  #CHECK_STATE(M_Wasser, 38, 6)
38  #SWITCH_ON(#9, M_Wasser)
39  BRA 6
40  #CHECK_STATE(M_Heizung, 41, 43)
41  #CHECK_LT(LT_innen , HT_innen, LT_min, HT_min, 43, 42)
42  #SWITCH_OFF(#7, M_Heizung)
43  #CHECK_STATE(M_Lüftung, 44, 49)
44  #CHECK_LT(LT_innen , HT_innen, LT_außen , HT_außen, 49,  48)
45  #CHECK_LT(LT_innen , HT_innen, LT_min, HT_min, 49 , 46)
46  #SWITCH_OFF(#8, M_Lüftung)
47  BRA 49
48  #CHECK_LT(LT_innen , HT_innen, LT_max, HT_max, 7,  3)
49  #CHECK_STATE(M_Wasser, 50, 1)
50  #CHECK_LT(HH_soll, LH_soll, HH_ist, LH_ist , 51, 1)
51  #SWITCH_OFF(#9, M_Wasser)
52  #CHECK_STATE(M_Alarm, 53, 1)
53  #CHECK_LT(LT_innen , HT_innen, LT_max, HT_max, 53 , 1)
54  #SWITCH_OFF(#3, M_Alarm)
55  BRA 1
56  #SWITCH_ON(#3, M_Alarm)
57  BRA 3
```

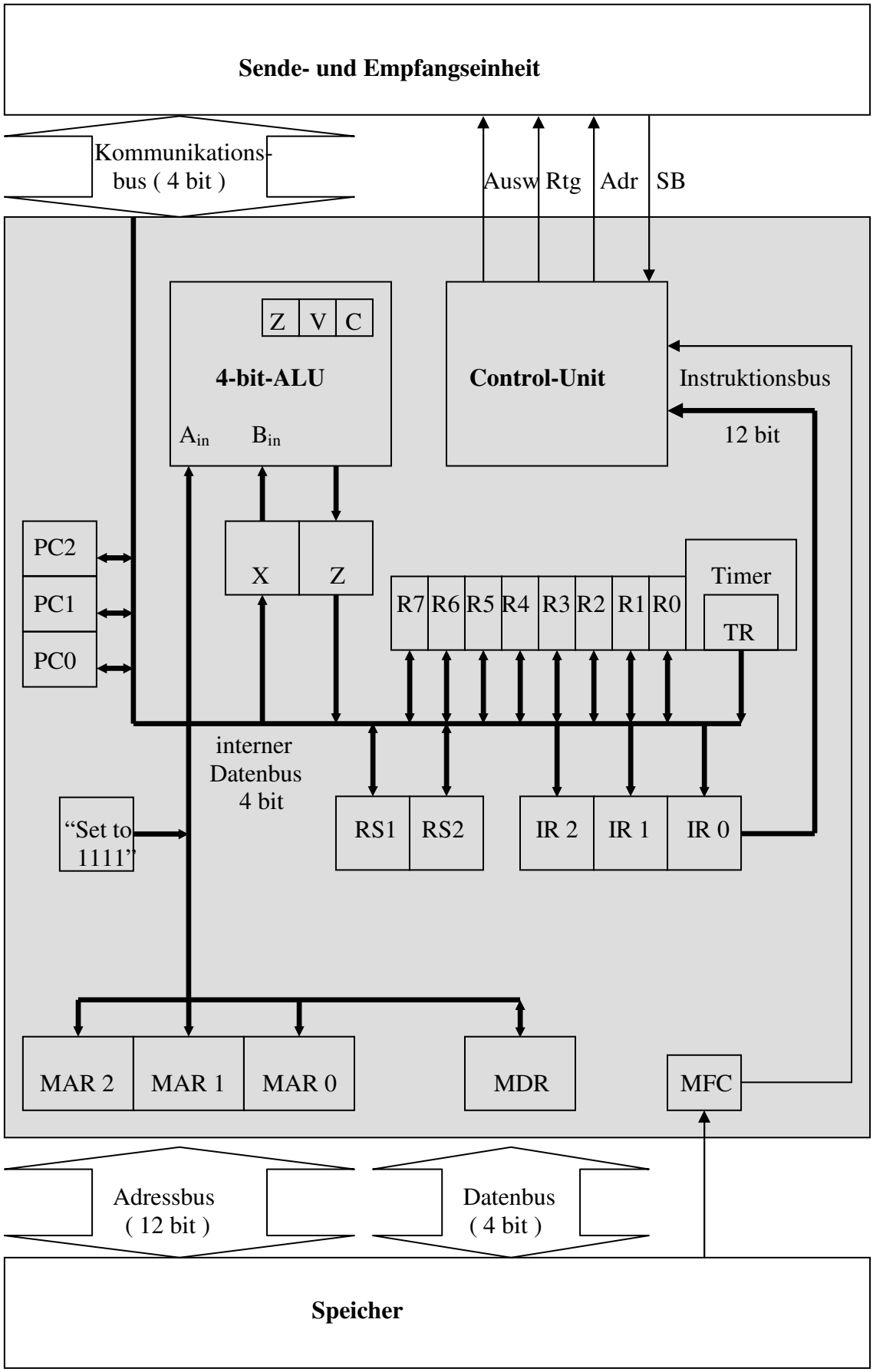
Fortsetzung Routine 2

Routine 3

Prüfroutinen

Alarmroutine

4.8.AUFBAU CPU (Blockschaltbild)



5. SCHLUSSBEMERKUNGEN

5.1. Bewertung der geplanten Einsatzmöglichkeiten

Im Vordergrund des Projektes stand für uns die Entwicklung eines funktionsfähigen Mikrocontrollers. Wir haben daher weniger Augemerk auf die Kosten für verwendete Sensoren etc. gelegt.
Für Temperatur- und Feuchtigkeitssteuerungen gibt es bereits preisgünstige Lösungen, z.B. Thermostatventile bzw. Klimaanlagen. Großer Vorteil der mikroprogrammierten Lösung ist allerdings, wie der Name bereits sagt, die Programmierbarkeit. So ist es besonders in großen Anlagen vorteilhaft, nicht jedes Thermostatventil einzeln einstellen zu müssen, sondern statt dessen einfach die Sollwerte neu zu programmieren.
Sollte es möglich sein, Komponenten und Mikroprozessor preisgünstig zusammenzustellen, so gäbe es durchaus praktische Einsatzmöglichkeiten in Gewächshäusern.

5.2. Bewertung der Einschränkungen

Einschränkungen sind in unserem Falle Temperaturen, die außerhalb der von uns festgelegten Grenzwerte liegen. Um dieses zu umgehen müsste man entweder das Datenformat vergrößern, bzw. die Messwerte grober mappen (statt 1°C-Schritten mit 2°C) und könnte so den Messbereich erweitern. Weitere Einschränkungen liegen bisher in der von uns vorausgesetzten Linearität der Aktuatoren. Will man zum Beispiel 10°C mehr erzielen, so reicht es im Normalfall nicht, ein Heizungsventil einfach zu öffnen. Um möglichst genau zu arbeiten müsste man das Ventil um einen gewissen Prozentgrad öffnen und diesen abhängig vom normalerweise runden Leitungsquerschnitt errechnen. Dieses wären allerdings Ergänzungen, die problemlos in unsere Implementation eingefügt werden könnten. Wir haben uns jedoch entschieden, darauf zu verzichten, da so nicht mehr an Funktionalität dazugekommen wäre, der Umfang des Projektes jedoch erheblich zugenommen hätte.

5.3. Problemabhängige Sicherheits- und Fehlerbetrachtungen

Da ein Unterschreiten der von uns gesetzten Sollwerte nicht möglich ist, da dann unsere Aktuatoren aktiv werden und die Istwerte anheben, ist die verbleibende Fehlerquelle ein Überschreiten der Sollwerte, bzw. Fehler im Programmablauf.
Überschreiten der Sollwerte ist bei der Temperatur nur möglich, wenn die Innentemperatur und gleichzeitig die Außentemperatur über dem Sollwert liegen. In diesem Fall gibt unser System solange Alarm, bis die Temperatur wieder unter den Sollwert sinkt. Der Alarm selbst ist natürlich nicht in der Lage, die Temperatur zu senken.
Bei der Bodenfeuchtigkeit ist es theoretisch möglich, dass zu viel gewässert wird, wenn z.B. nur ein Liter Wasser notwendig gewesen wäre, aber durch die Öffnung des Wasserventils 1o Liter zufließen. Dieses fällt jedoch zum einen unter die Einschränkungen und ist zum anderen nur sehr kurzfristig möglich, da permanent geprüft wird und geregelt wird.

5.4. Zeitkritische Abläufe

Da unsere Regelung permanent erfolgt, sind keine Zeitprobleme zu erwarten.
Der einzig wirklich zeitkritische Ablauf ist das Regeln der Heizung, da der Messwert über einen bestimmten Zeitraum unter den Sollwert gefallen sein muss. Dieser Zeitraum könnte theoretisch durch den Programmablauf überschritten werden bevor die zweite Prüfung stattfindet. Da der Zeitraum aber vom Timer abhängt, genügt es, die Zeit, nach der der Timer das Signal ändert, so groß zu wählen, dass das Programm auf jeden Fall kürzer läuft.

5.5. Testmöglichkeiten

Um die Funktion zu Testen müssten folgende Szenarien simuliert werden:

$T_{innen} < T_{außen}$, $T_{innen} < T_{Min}$ $\Rightarrow$ Fenster auf, Heizung an ?
$T_{innen} < T_{außen}$, $T_{innen} > T_{Max}$ $\Rightarrow$ Alarm ?
$T_{innen} > T_{außen}$, $T_{innen} < T_{Min}$ $\Rightarrow$ Heizung an ?
$T_{innen} > T_{außen}$, $T_{innen} > T_{Max}$ $\Rightarrow$ Fenster auf ?

$H > H_{Soll}$ $\Rightarrow$ Wasser aus ?
$H < H_{Soll}$ $\Rightarrow$ Wasser an ?

Dabei müsste man die verschiedenen Aufeinanderfolgen der einzelnen Möglichkeiten simulieren, um zu prüfen wie in diesen Fällen die Reaktion der Anlage ist.

5.6. Erweiterungen

Erweitert werden könnte die Anlage durch Routinen und Messgeräte zur Steuerung von Lichteinstrahlung, Düngemittelgehalt des Bodens und zur Kühlung. Des Weiteren könnte man die Regelung der Temperatur und der Feuchtigkeit dynamisch an die Wachstumsphasen der Pflanzen anpassen.