Technische Informatik II 2007

Projekt: Multikriterienfeuermelder

Michael Fiedler (508342), Stefan Przybilla (504432), Yves Radunz (509242)

Inhaltsverzeichnis

1 Darstellung des Problems **4**
- 1.1 Recherche . . . 4
- 1.2 Problemanalyse . . . 4
- 1.3 Festlegung der Basisfunktionalität . . . 5
- 1.4 Einschränkungen . . . 5
- 1.5 Systemanforderungen . . . 5
- 1.6 Anwendungsszenrario zum Schaubild . . . 6

2 Architekturentwurf **8**
- 2.1 Datentypen und -formate . . . 8
- 2.2 Sensoren und Aktuatoren . . . 8
- 2.3 Sensorenauswertung . . . 9
- 2.4 Kommunikation mit den benachbarten Meldern . . . 10
- 2.5 Weitere Komponenten . . . 11
- 2.6 Speicher . . . 12

3 Mikroprozessor **13**
- 3.1 Prozessoraufbau . . . 13
- 3.2 Flags . . . 14
- 3.3 Steuersignale . . . 14
- 3.4 Mikrocode-Kodierung . . . 17
- 3.5 Befehlssatz + Mikrocode . . . 19
  - 3.5.1 Aufbau der Befehlsworte und Operanden . . . 19
  - 3.5.2 Verarbeitung des Mikrocodes . . . 19
  - 3.5.3 Fetch . . . 20
  - 3.5.4 FetchOperand 1 . . . 20
  - 3.5.5 FetchOperand 2 . . . 21
  - 3.5.6 Arithmetische Befehle . . . 23
  - 3.5.7 Sprungbefehle . . . 24
  - 3.5.8 Flagoperationen . . . 26
  - 3.5.9 Gerätesteuerung . . . 27
  - 3.5.10 Spezialbefehle . . . 28
  - 3.5.11 Sonstige Befehle . . . 38
  - 3.5.12 WriteBack . . . 38
- 3.6 Assemblerprogramm . . . 39

4 Schlussbemerkungen **43**
- 4.1 Bewertung der geplanten Einsatzmöglichkeiten . . . 43
- 4.2 Bewertung der Einschränkungen . . . 43
- 4.3 Fehlerbetrachtung . . . 43
- 4.4 Timingbedingungen bei zeitkritischen Abläufen . . . 43
- 4.5 Testmöglichkeiten und mögliche Erweiterungen . . . 43
- 4.6 Aufgabenverteilung . . . 44
- 4.7 Literatur- und Quellenverzeichnis . . . 44

1 Darstellung des Problems

1.1 Recherche

Die allgemeine Zielstellung unseres Projektes ist die Entwerfung einer Brandschutzanlage, bestehend aus einem System von Multikriterienmeldern, das eine Brandentwicklung als solche möglichst zuverlässig und schnell erkennen und melden kann. Der grundsätzliche Unterschied eines solchen Melders im Gegensatz zu herkömmlichen Modellen ist die Tatsache, dass mehrere verschiedene Sensoren benutzt werden, die alle auf unterschiedliche Merkmale der Brandentwicklung ausgerichtet sind. Solche „Brandkenngrößen“ sind beispielsweise Messwerte für Rauchdichte und Temperatur, sowie die Konzentration von Brandgasen wie Kohlenmonoxid und Kohlendioxid. Im Prozessor unseres Melders soll es demnach zu einem Brandkenngrößen-Mustervergleich kommen, in dem die Muster der Signale, die die Sensoren empfangen haben mit den Mustern, die typisch für Brände sind, verglichen werden. Beispiele für solche Muster, die auf einen Brand hindeuten wären:

- ein bestimmter Temperaturanstieg bei gleichzeitiger überschreitung einer Mindesttemperatur
- ein gleichzeitiger Anstieg der CO-Konzentration und der Rauchdichte
- Temperaturanstieg bei gleichzeitigem Auftreten von Rauch mit einer vorgegebenen Rauchdichte
- Ein starkes Lichtsignal, dass sich mit der für Flammen typischen Flackerfrequenz ändert

Beispiele für Muster, die keinen Brand erwarten lassen wären:

- Ein sehr langsamer Anstieg eines Rauchdichtesignals, der über Monate hinweg andauert
- Plötzliche hohe Signale von kurzer Dauer

Durch diese Form der Signalverarbeitung soll die Gefahr eines falschen Alarms stark vermindert werden, so dass beispielsweise Zigarettenqualm oder Wasserdampf keinen derartigen Alarm mehr auslösen. Das macht vor allem in größeren Gebäuden wie Firmenkomplexen oder Universitäten Sinn, wo solch ein Alarm in der Regel auch gleich die Feuerwehr kontaktiert. Wurde der Brand erkannt und dem System gemeldet, werden Gegenmaßnahmen eingeleitet und die Feuerwehr gerufen.

1.2 Problemanalyse

Zwei Problemschwerpunkte sind bei dieser Thematik zu betrachten. Der erste ist die zuverlässige Branderkennung. Natürlich kann Feueralarm jederzeit manuell ausgelöst werden, allerdings liegt der Schwerpunkt natürlich bei der automatischen Erkennung. Diesbezüglich sind Fehlalarme oder gar das Nichtauslösen des Alarms im Falle eines Brandes Szenarien, die es weitestgehend zu vermeiden gilt. Das System nutzt hierzu Brandkenngrößen und manuelle Eingaben, die es mit den aufgenommenen Daten der Sensoren ständig vergleicht. So existieren sogenannte Schwellenwerte. Werden sie bei der Gesamtauswertung überschritten, kommt es zum Alarm. Ein probates Hilfswerkzeug zur flexiblen Gesamtauswertung, ist dabei die Fuzzy Logic. Der zweite Problemschwerpunkt ist die Brandbekämpfung. Das Einschalten des Alarms und der Sprinkler ist hierbei keine große Sache. Erst die Deaktivierung der Gegenmaßnahmen, vor allem der Sprinkler, zum richtigen Zeitpunkt ist ein Problem, da der Brand vollständig gelöscht sein muss, die Sprinkler selber aber keine unnötigen Wasserschäden verursachen sollen. Darüber hinaus haben wir Fluchtschilder in das System implementiert, die den Menschen im Gebäude anzeigen, ob die jeweilige Route sicher oder riskant ist, um ihnen im Idealfall ein sicheres Herauskommen ermöglichen.

1.3 Festlegung der Basisfunktionalität

Unser Brandschutzsystem erkennt und bekämpft Brände zuverlässig. Zudem hilft es Betroffenen das Gebäude sicher zu verlassen. Zur Branderkennung werden zum einen Sensoren zur Gasanalyse eingesetzt. Diese messen den CO, CO_2 und O_2 Anteil in der Luft und erkennen darüber hinaus andere Schwebstoffe in der Umgebung. Zum Anderen gibt es einen Wärmesensor, der die Deckentemperatur misst und eine Wärmebildkamera, die die Temperatur auf Oberflächen wie beispielsweise Böden, Möbel oder Personen erfasst und bis zu 25,6m mal 25,6m Raum abdeckt. Die so ermittelten Kenngrößen werden nun einzeln einer Bewertung zugeordnet und dann zu einer Gesamtwertung verrechnet. Diese wird wiederum mit einem Schwellenwert verglichen. Ist der Vergleich positiv, wird Alarm ausgelöst, es kommt zur Brandbekämpfung und die Fluchtschilder werden aktiviert. Alarm und Fluchtschilder schalten sich erst ab, nachdem die Anlage auf „Reset“ gesetzt wurde.

1.4 Einschränkungen

Die wohl größte Einschränkung des Systems ist das Unvermögen Leute, die sich während des Brandes innerhalb des Gebäudes aufhalten mit einer optimal errechneten Route sicher heraus zu lotsen. Es existieren zwar die Fluchtschilder, doch diese zeigen den Betroffenen im wahrsten Sinne des Wortes nur wo es brennt. Ein Fluchtplan in jedem größeren Raum, der via Leuchtdioden den Leuten zeigt auf welchem Wege sie am Sichersten rauskommen wäre eine sinnvolle Erweiterung des Systems. Das hätte aber die Kapazitäten unseres Prozessors überbeansprucht. Ansonsten kann bisher jeder Melder mit bis zu 14 Nachbarn verbunden werden und die Wärmebildkamera erfasst einen Raum bis zu 25,6m mal 25,6m. Als letzte Einschränkung ist der manuelle Feueralarm nicht in unserem System implementiert, kann aber als simple Erweiterung verstanden werden.

1.5 Systemanforderungen

Die Systemanforderungen des Systems wären mehrere Nachbar CPUs mit einer Taktfrequenz von 10 MHz, da gerade die Wegfindung für die Fluchtschilder im schlimmsten Fall 45 Millionen Takten benötigt. Darüber hinaus ist diese Taktfrequenz vonnöten, um die Daten, die die Wärmebildkamera liefert schnell genug auszuwerten.

1.6 Anwendungsszenrario zum Schaubild

Ruhezustand

Im Ruhezustand sammelt das System über die Sensorabfrage Daten, prüft, ob das Resetflag gesetzt wurde, wertet die Daten aus, testet, ob das Ergebnis den Schwellenwert überschreitet, prüft, ob beim letzten Durchlauf ein Brand festgestellt wurde und nach der Feststellung, dass kein globaler Alarm aktiviert wurde, beginnt ein neuer Programmdurchlauf.

Brandfall

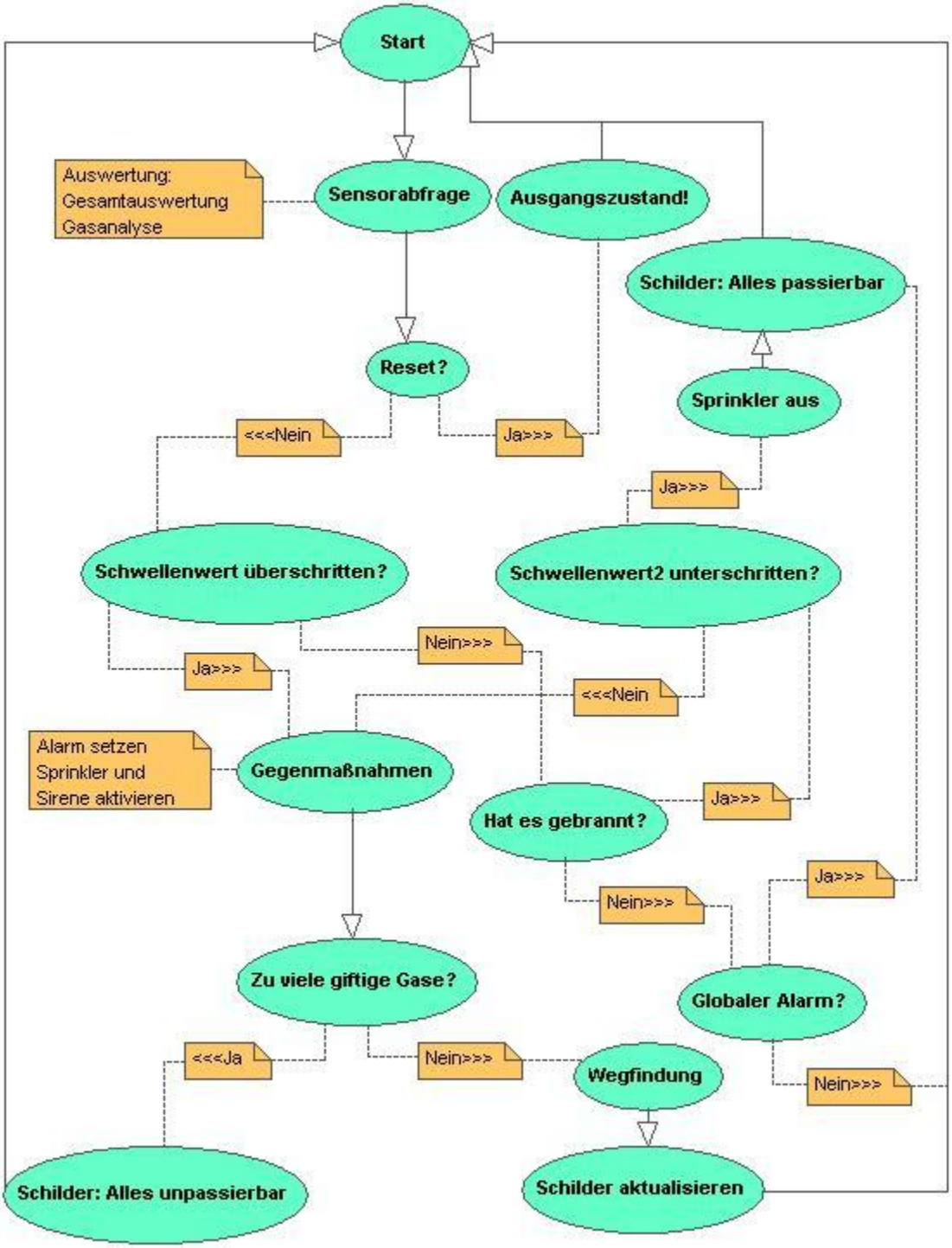
Im Brandfall ist die Schwellenwertüberschreitung positiv und es kommt zur Einleitung der Gegenmaßnahmen. Zuerst wird lokaler und globaler Alarm ausgelöst. Der globale Alarm lässt die Sirenen aller Feuermelder aufheulen und schaltet sich nur dann ab, wenn die Anlage auf Reset gesetzt wurde. Der lokale Alarm wird nur bei den Meldern ausgelöst, die einen Brand identifiziert haben. Er ist für die Steuerung der Sprinkler und der Fluchtschilder wichtig. Schaltet sich demnach dieser Alarm ein, gehen auch alle Sprinkler in diesem Raum los. Nun kommt es zu einer Analyse der zuletzt gemessenen Daten auf die Konzentration von Giftgasen. Ist diese Konzentration in einem intoleranten Bereich, werden alle Fluchtschilder auf Unpassierbar gesetzt und ein neuer Programmdurchlauf wird gestartet. Ist das aber nicht der Fall, kann die Anlage mit der Wegfindung fortfahren. Sie benutzt dazu den lokalen Alarm, um zu bestimmen in welchen Räumen die Brandherde sind. Mithilfe dieser Daten können die Fluchtschilder aktualisiert werden und beispielsweise auf riskant oder passierbar gesetzt werden. Anschließend kommt es zu einem neuen Programmdurchlauf.

Gelöschter Brand

Der gelöschte Brand wird natürlich identifiziert, da neue Daten über die Sensoren zur Verfügung stehen und die Gesamtwertung unter dem Schwellenwert liegt. Die Anlage wurde auch noch nicht auf Reset gesetzt und stellt sich demnach die Frage, ob es gebrannt hat. Da beim vorangegangenen Programmdurchlauf der Schwellenwert überschritten wurde , wird diese Frage vom Programm mit einem Ja beantwortet. Nun wird getestet, ob ein zweiter, weitaus sensibler Schwellenwert beim Vergleich mit der aktuellen Gesamtwertung unterschritten wird, da so sicher gestellt wird, dass der Brand wirklich gelöscht wurde. Wurde der Schwellenwert nicht unterschritten, geht es zurück zu den Gegenmaßnahmen, die aktiviert bleiben. Ist er aber unterschritten, werden lokaler Alarm und Sprinkler deaktiviert und die Fluchtschilder auf „Alles passierbar“ aktualisiert. Danach folgt ein neuer Programmdurchlauf.

Rücksetzung

Bei der Rücksetzung bleibt erstmal alles beim Alten mit dem Unterschied, dass jetzt der Test, ob es gebrannt hat negativ ausfällt. Was folgt ist die Prüfung nach dem globalen Alarm, der natürlich weiterhin aktiv ist und mit ihm auch die Sirenen des Feueralarms. Dieser Zustand bleibt genau so lange erhalten bis die Anlage per Reset zurückgesetzt wurde. Globaler Alarm und Sirenen schalten sich ab und das System befindet sich wieder im Ruhezustand.

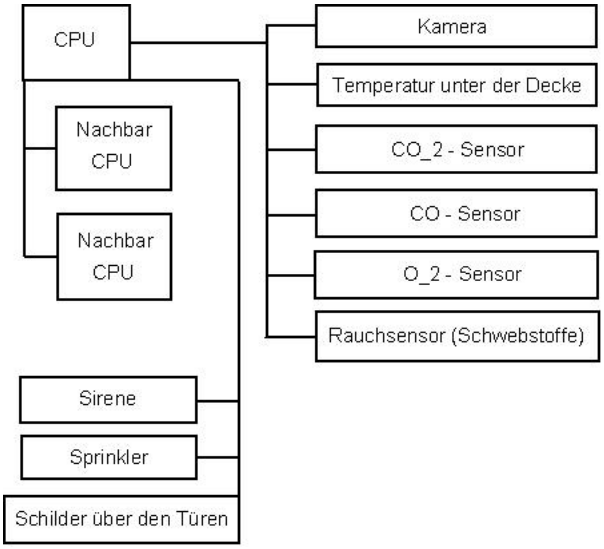

Start
Auswertung:
Gesamtauswertung
Gasanalyse
Sensorabfrage
Ausgangszustand!
Schilder: Alles passierbar
Reset?
Sprinkler aus
<<<Nein
Ja>>>
Ja>>>
Schwellenwert überschritten?
Schwellenwert2 unterschritten?
Nein>>>
Ja>>>
<<<Nein
Alarm setzen
Sprinkler und
Sirene aktivieren
Gegenmaßnahmen
Hat es gebrannt?
Ja>>>
Ja>>>
Nein>>>
Zu viele giftige Gase?
Globaler Alarm?
<<<Ja
Nein>>>
Wegfindung
Nein>>>
Schilder: Alles unpassierbar
Schilder aktualisieren

2 Architekturentwurf

2.1 Datentypen und -formate

Die Sensoren liefern 8 Bit Messwerte und geben diese in Form von 16 Bit Ganzzahlen an den Prozessor weiter. Ansonsten werden noch einige Werte als 16 Bit Festkommawerte interpretiert bei denen alle Bits Nachkommastellen sind und das Komma vor den höchstwertigen Bit steht.

2.2 Sensoren und Aktuatoren



Wärmebildkamera

Die Wärmebildkamera liefert ein Bild von 256×256 Pixeln. Dabei nimmt jeder Pixel die Werte 0 bis 255 Grad Celsius an und entspricht einer Fläche von 10×10cm. Die Kamera sendet ihr Bild permanent pixelweise, indem sie periodisch auf einer Leitung den Pixelwert angibt und auf einer zweiten Leitung die Pixeladresse sendet. Dabei ist die Pixeladresse ein 16 Bit Wert, dessen höherwertiges Byte die Y-Koordinate und das niederwertige Byte die X-Koordinate.

Wärmesensor an der Decke

Der Wärmesensor an der Decke liefert den Prozessor die Temperatur in Grad Celsius von 0 bis 255 in 16 Bit codiert.

Rauchsensor

Der Rauchsensor misst optisch die Schwebstoffkonzentration in der Luft und liefert die Daten an den Prozessor im Wertebereich von 0 bis 255.

Sensoren zur Gasanalyse

Die CO-, CO_2- und O_2-Sensoren messen den jeweiligen Anteil in der Luft und liefern ebenfalls die Daten im Wertebereich von 0 bis 255 an den Prozessor.

Sirene und Sprinkler

Ein Alarmton, der erzeugt wird, ist die Sirene. Sie hat 2 Steuerleitungen. Wenn auf der ersten Leitung ein Spannungsimpuls wirkt, wird die Sirene aktiviert, unabhängig davon in welchem Zustand sie sich vorher befand. Wirkt auf die zweite Leitung ein Impuls, wird die Sirene wieder unabhängig vom Zustand vorher deaktiviert. Der Sprinkler ist eine Löschmaßnahme im Brandfall. Auch dieser verfügt über zwei Steuerleitungen, die sich analog zur Sirenensteuerung verhalten.

Fluchtschilder

Bei den Fluchtschilder unterscheidet man zwischen 4 Statusbits. Diese zeigen den Betroffenen was einem hinter der nächsten Tür erwartet.

Bezeichnung	Beschreibung
WFA	Es gibt einen Weg zu einem gelöschten Feuer.
WF	Es gibt einen Weg zu einem brennenden Feuer.
WRF	Es gibt einen Weg nach draussen, der durch einen brennenden Raum hindurch führt. Dabei wird gewährleistet, dass ein ausreichend breiter Weg (mindst. 90×90cm zu jedem Zeitpunkt) durch die brennenden Räume zur Verfügung steht.
WR	Es gibt einen Weg nach draussen, der durch keine brennenden Räume führt.

2.3 Sensorenauswertung

Es werden aus den Sensoren acht Kenngrößen ermittelt. Dabei werden aus der Wärmebildkamera drei Kenngrößen ermittelt und aus den anderen Sensoren jeweils eine. Dies sind folgende Kenngrößen:

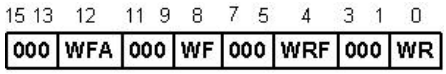
1. CO_2-Gehalt in der Luft
2. CO-Gehalt in der Luft
3. O_2-Gehalt in der Luft
4. Rauch- bzw. Schwebstoffgehalt der Luft
5. Temperatur an der Decke
6. Anzahl der zu heißen bzw. brennenden Felder
7. Temperatur des heißesten Pixels, welches die Schwellentemperatur überschritten hat
8. Höchste Temperaturdifferenz, um welche die Schwellentemperatur eines Pixels überschritten wurde.

Jede dieser Kenngrößen wird mit einer Bewertungsfunktion (deren Werte in einer Look-Up-Table gespeichert sind) eine Wertung zugewiesen, welche die Wahrscheinlichkeit eines Feuers darstellt. Diese Bewertungen werden durch Fuzzy-Logic verknüpft, um eine Gesamtwertung zu bilden. Anhand dieser Wertung wird entschieden, ob es in dem Raum brennt.
Die ersten 4 Kenngrößen werden verwendet, um festzustellen, ob die Luft im Raum gesundheitsschädlich ist.

2.4 Kommunikation mit den benachbarten Meldern

Die Kommunikation verläuft über festgelegte Speicherstellen die eine entsprechende Verdrahtung aufweisen, außerdem werden noch die beiden FlagBits für den globalen Alarm und *AlarmReset* in die Kommunikation mit eingebunden.

Datenformat für die Kommunikation



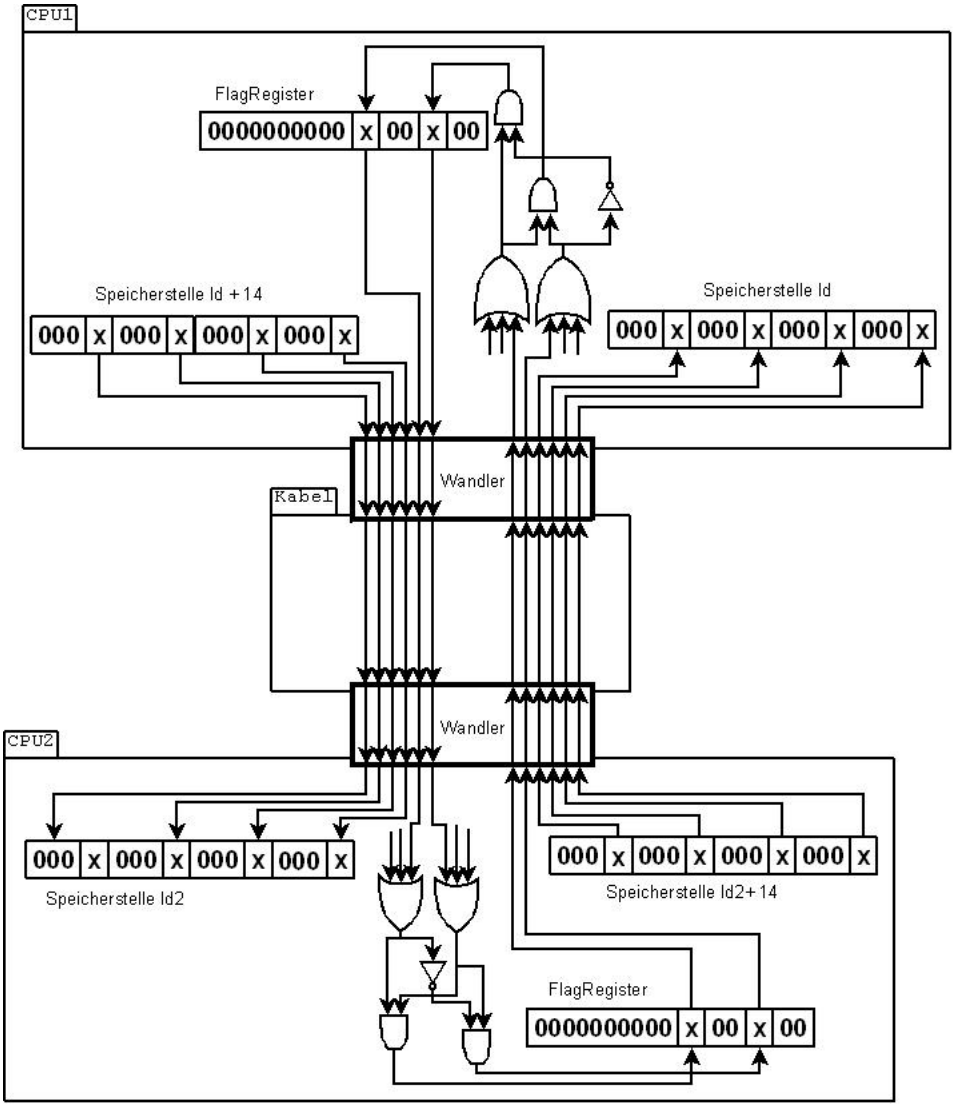
WFA = Weg zu gelöschtes Feuer
WF = Weg zu aktivem Feuer
WRF = Weg nach Draußen mit Feuer
WR = Weg nach Draußen ohne Feuer

Die Speicherstellen können nur Werte annehmen bei denen eine beliebige Kombination der speziellen Bits in der obigen Grafik gesetzt sind.

Kommunikationsschema

Für jeden der maximal 14 Nachbaren existieren 2 solche Speicherstellen. Eine für die Daten zum Nachbarn und eine für die Daten vom Nachbarn. Die Daten zum Nachbarn drücken aus, was für Wege vom Nachbarn durch diesen Raum existieren, sprich gibt es einen Weg von diesem Raum nach draußen der von diesem Nachbarn erreichbar ist und so weiter. Die Daten vom Nachbarn X sind dann entsprechenden Kopien der entsprechenden Speicherstelle des Nachbarn X für diesen Raum. Die Daten vom Nachbarn setzen dann auch automatisch den Zustand der Schilder über den Türen/Wegen zu diesem Nachbarn über eine feste Verdrahtung.
Dabei gleicht ein Feuermelder die Flags für Alarm und Alarm-Reset mit denen seiner direkten Nachbarn ab.
Er setzt sein Flag für den globalen Alarm genau dann, wenn einer der Nachbarn den globalen Alarm gesetzt und keiner seiner Nachbarn sein Reset-Flag gesetzt hat. Er setzt sein Reset-Flag, wenn einer der Nachbarn das Flag für den globalen Alarm und einer der Nachbarn das Reset-Flag gesetzt hat. Das Reset-Flag bleibt dann so lange gesetzt, bis es von einem Programm-Befehl gelöscht wird. Damit wird sichergestellt, dass das Reset-Signal sich wirklich durch das gesamte Feuermelder-Netz ausbreiten kann und damit jeden Feuermelder dazu veranlassen kann, den Alarm zu beenden.
Das Setzen der Flags erfolgt durch eine feste Verdrahtung und ist damit von dem Zustand des Feuermelders unabhängig.

Kommunikationsschaltung



2.5 Weitere Komponenten

Zu den weiteren Komponenten zählt der manuelle Auslöser des Feueralarms, der von dem Feuermelder wie ein benachbarter Feuermelder behandelt wird. Der Auslöser verhält sich also wie ein zusätzlicher Feuermelder, welcher per Tastendruck einen globalen Alarm auslösen kann.
Weiterhin verfügt das Feuermelder-Netz über eine Automatik, welche die Feuerwehr alarmiert, sobald die Feuermelder einen globalen Alarm melden. Diese Komponente wird von den im Netz enthaltenen Feuermeldern ebenfalls wie ein Nachbar behandelt.

Zuletzt gibt es noch den Reset-Knopf, der an Stelle eines Nachbarn an den Feuermelder angeschlossen wird. Wie der manuelle Auslöser und die Alarmierung der Feuerwehr wird auch der Reset-Schalter vom Feuermelder wie ein Nachbar behandelt. Dieser virtuelle Nachbar ignoriert alle eingehenden Signale und dient nur zum Setzen des Reset-Flags.

2.6 Speicher

Der Feuermelder verfügt über mehrere unabhängige Speicher:

1. Ein ROM für den Mikro-Code. Dieser Speicher verfügt über 1024×77 Bit Speicher und enthält den Mikrocode der Befehle, der Abschnitte für das Laden der Operanden und des Writebacks.

2. Ein ROM für die Mikro-Code-Sprungziele in die verschiedenen Etappen der Befehlsverarbeitung. Dieser Speicher hat eine Größe von 128×10 Bits und enthält die Zieladressen der Beschreibungen für das Laden der Operanden, der eigentlichen Ausführung des Befehls und des Zurückschreibens des Ergebnisses.

3. Ein ROM für die Schwellenwerte der Temperatur der von der Kamera beobachteten Felder mit einer Größe von 655536×16 Bits. Dieser Speicher enthält die Temperaturen, ab denen ein bestimmtes Pixel als zu heiß (bzw. brennend) erkannt wird.

4. Ein ROM für die Vergleichswerte der Gaskonzentrationen mit einer Größe von 4×8 Bits, welche als 16-Bit-Zahlen auf den Bus gelegt werden (führende Nullen werden ergänzt). Wenn eine der Gaskonzentrationen den Vergleichswert überschreitet (bzw. wenn die Sauerstoff-Konzentration den Vergleichswert unterschreitet) wird der Raum als unpassierbar angesehen.

5. Ein ROM für die Positionen der Türen mit 129×16 Bits. Hier werden die Positionen der Türen der Nachbaren gespeichert. In der Speicherzelle 80_{16} wird die Anzahl der Nachbarn gespeichert. Wenn ein Nachbar weniger als 8 (dies ist die maximal mögliche Anzahl der Türen zu einem Nachbarn) hat, dann werden die überzähligen Türen an die Position der ersten Tür gesetzt. In der Speicherzelle n ($n \in \{0, ..., 127\}$) ist die Position der Tür mit der Nummer n gespeichert. Diese Tür gehört dann zum Nachbarn mit der Nummer n mod 16.

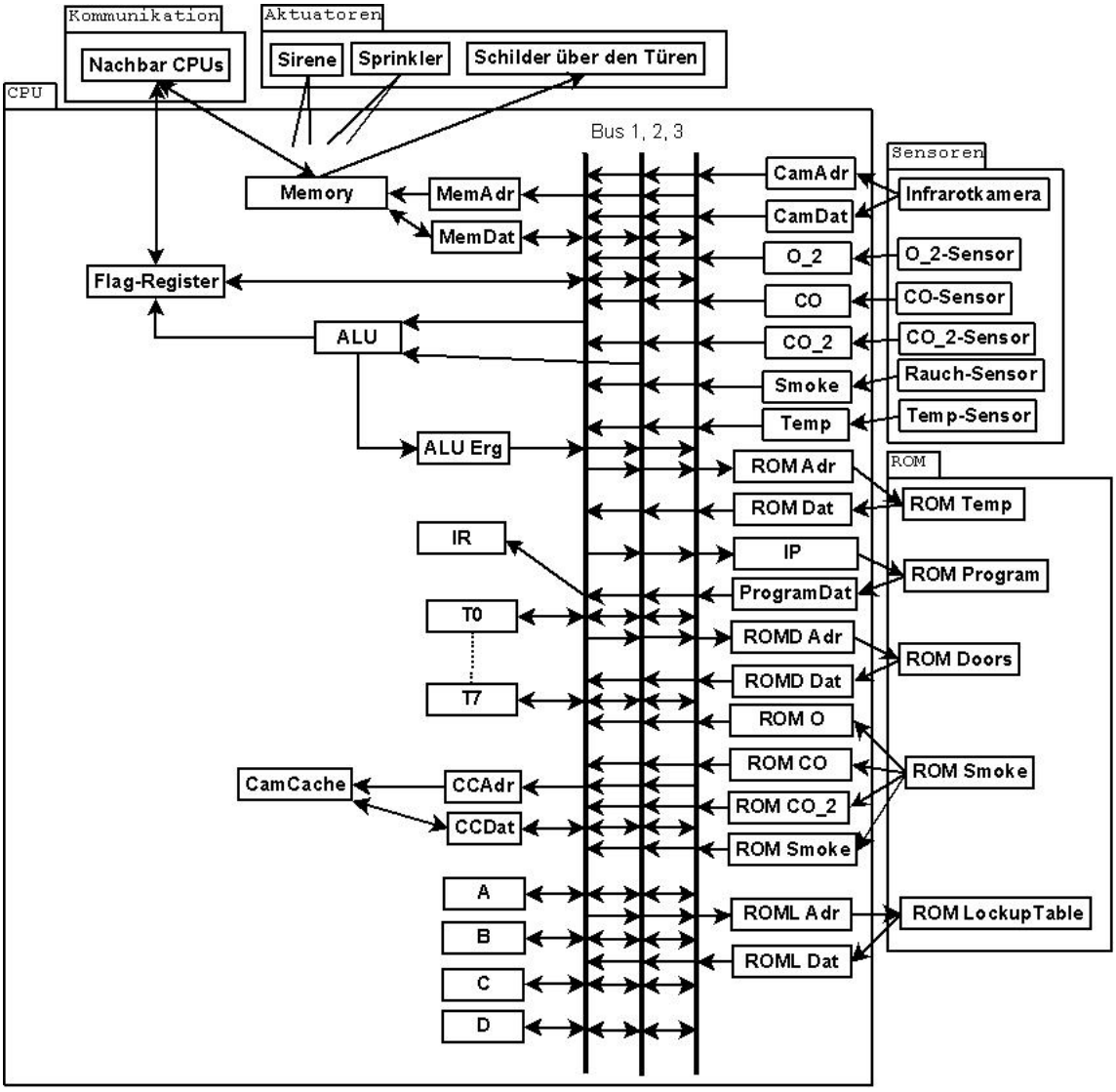
6. Ein ROM für die Look-Up-Table mit 2048×16 Bits, in welchem die Gefahrenbewertung für die gemessenen Kenngrößen gespeichert ist, damit diese nicht ständig neu berechnet werden müssen.

7. Ein schneller Speicher als Cache für die Kamera-Werte (intern im Prozessor) mit 65536×16 Bits, welcher für die Wegfindung (im Befehl `Search`) genutzt wird.

8. Ein schneller Arbeitsspeicher für die Kommunikation und zum Speichern einiger Zwischenwerte mit 256×16 Bits Größe, dessen Speicherzellen 0 bis 13 für die eingehende und 14 bis 27 für die ausgehenden Kommunikation mit den Nachbarn genutzt werden. Der Rest des Speichers dient zur Rückgabe der im Befehl `Search` berechneten Zusammenhangskomponenten.

3 Mikroprozessor

3.1 Prozessoraufbau



3.2 Flags

Flag	Bedeutung
0	Nicht negativ
1	Null
2	Globaler Alarm (im gesamten Gebäude)
3	Lokaler Alarm (nur im Raum)
4	Hat gebrannt
5	Reset-Signal
6	Rauch

3.3 Steuersignale

Signal	Aktion
Register beschreiben	
`AIn`N	Wert des N. Datenbusses im Register A speichern
`BIn`N	Wert des N. Datenbusses im Register B speichern
`CIn`N	Wert des N. Datenbusses im Register C speichern
`DIn`N	Wert des N. Datenbusses im Register D speichern
`FIn`N	Wert des N. Datenbusses im Flagregister speichern
`IPIn`N	Wert des N. Datenbusses als Befehlsadresse speichern
`T`n`In`N	Wert des N. Datenbusses im temporären Register n speichern
`IRIn`N	Wert des N. Datenbusses im Befehlsregister speichern
Register lesen	
`AOut`N	Wert des Registers A auf den N. Datenbus legen
`BOut`N	Wert des Registers B auf den N. Datenbus legen
`COut`N	Wert des Registers C auf den N. Datenbus legen
`DOut`N	Wert des Registers D auf den N. Datenbus legen
`FOut`N	Wert des Flagregisters auf den N. Datenbus legen
`IPOut`N	Befehlsadresse auf den N. Datenbus legen
`T`n`Out`N	Wert des temporären Registers n auf den N. Datenbus legen
Registeroperationen in den Speicher- / ROM- / Cache- / Sensor-Schnittstellen	
`CamAdrOut`N	Nummer des aktuellen Pixels der Kamera auf den N. Datenbus legen
`CamDatOut`N	Temperatur des aktuellen Pixels der Kamera auf den N. Datenbus legen
`CamCacheAdrIn`N	Adresse der aktuellen Speicherzelle des Kamera-Caches vom N. Datenbus laden
`CamCacheDatIn`N	Wert des N. Datenbusses in das Schnittstellenregister des Kamera-Caches speichern
`CamCacheDatOut`N	Wert des Schnittstellenregisters des Kamera-Caches auf den N. Datenbus legen
`DoorsAdrIn`N	Wert des N. Datenbusses in das Adressregister des Türenverzeichnis laden
`DoorsDatOut`N	Inhalt der angegebenenen Speicherzelle des Türenverzeichnisses auf den N. Datenbus legen

`RomAdrIn`N	Adresse der aktuellen ROM-Speicherzelle vom N. Datenbus laden
`RomDatOut`N	Wert der adressierten ROM-Speicherzelle auf den N. Datenbus legen
`LookUpAdrIn`N	Adresse der aktuellen Look-Up-Table-Speicherzelle vom N. Datenbus laden
`LookUpDatOut`N	Wert der adressierten Look-Up-Table-Speicherzelle auf den N. Datenbus legen
`ProgOut`N	Durch IP adressiertes Wort aus dem Programm-ROM auf den N. Datenbus legen
`MemAdrIn`N	Adresse der aktuellen Speicherzelle vom N. Datenbus laden
`MemDatIn`N	Wert des N. Datenbusses im Schnittstellenregister des Speichers abgelegen
`MemDatOut`N	Wert des Schnittstellenregisters des Speichers auf den N. Datenbus legen
`SmokeOut`N	Gemessenen Rauch- und Schwebstoffwert auf den N. Datenbus legen
`SmokeRefOut`N	Rauch- und SchwebstoffSchwellenwert auf den N. Datenbus legen
`OOut`N	Gemessenen Sauerstoffwert auf den N. Datenbus legen
`ORefOut`N	SauerstoffSchwellenwert auf den N. Datenbus legen
`COOut`N	Gemessenen Kohlenstoffmonooxidwert auf den N. Datenbus legen
`CORefOut`N	KohlenstoffmonooxidSchwellenwert auf den N. Datenbus legen
`COIIOut`N	Gemessenen Kohlenstoffdioxidwert auf den N. Datenbus legen
`COIIRefOut`N	KohlenstoffdioxidSchwellenwert auf den N. Datenbus legen
`TempOut`N	Gemessene Deckentemperatur auf den N. Datenbus legen

Steuerung der Speicher- / ROM- / Cache- / Sensor-Schnittstellen

`WaitForCamAdrChange`	Warten, bis sich die Adresse des gerade von der Kamera übertragenen Pixels ändert
`CamCacheWrite`	Cache-Speicherzelle mit dem Inhalt des Cache-Schnittstellenregisters beschreiben
`CamCacheRead`	Cache-Schnittstellenregister mit dem Inhalt der Cache-Speicherzelle laden
`MemWrite`	Speicherzelle mit dem Inhalt des Schnittstellenregisters beschreiben
`MemRead`	Schnittstellenregister mit dem Inhalt der Speicherzelle laden

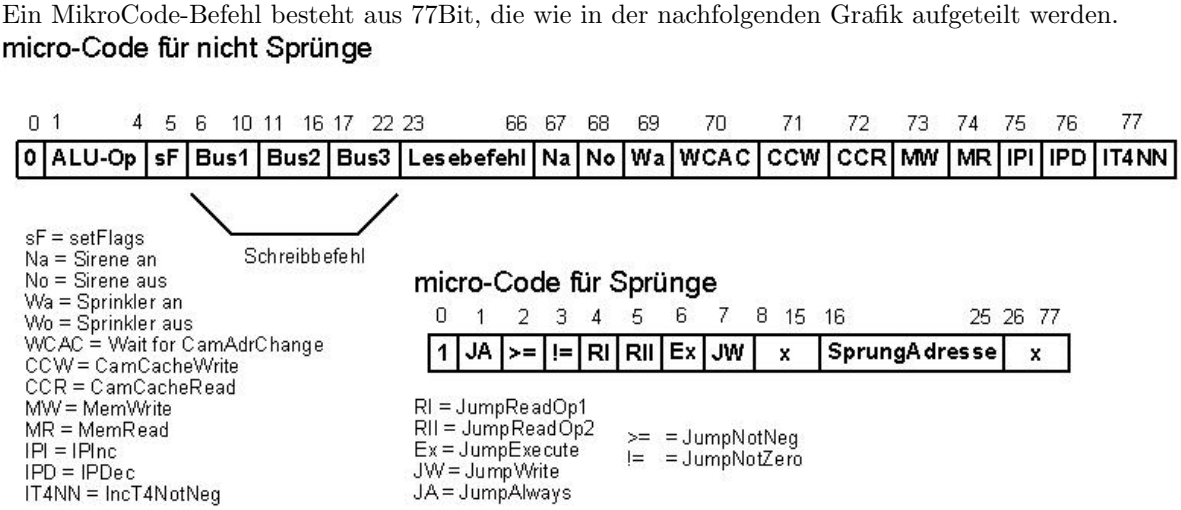
ALU-Operationen

`Xor`	Logisches Exklusives Oder
`Eqv`	Logische Äquivalenz
`And`	Logisches Und
`Or`	Logisches Oder
`Add`	Addition
`Sub`	Subtraktion
`Dec`	Dekrement
`Inc`	Inkrement
`Max`	Maximum
`MulH`	Multipliziert die Operanden, interpretiert das Ergebnis als 32-Bit-Zahl und gibt die höchstwertigsten 16 Bits zurück. Es gilt $Ergebnis = \lfloor \frac{Wert1 \cdot Wert2}{65536} \rfloor$.
`Check`	Setzt das NotZero-Flag genau dann, wenn bei einem der beiden Operanden eines der 8 höchstwertigen Bits gesetzt ist.
`Adress`	Berechnet den Wert $256 \cdot X + Y$, wobei X der Wert auf dem ersten Datenbus und Y der Wert auf dem 2. Datenbus ist.
`SpecialOr`	Übernimmt das höherwertige Byte unverändert in das Ergebnis, verodert

	die niederwertigsten Halbbytes, speichert das Ergebnis dieser Veroderung in dem Halbbyte mit der zweitniedrigsten Wertigkeit und setzt das unterste Halbbyte auf 0.
Konstanten	
`0000Out`N	Lege 0_{16} auf den N. Datenbus
`0004Out`N	Lege 4_{16} auf den N. Datenbus
`0008Out`N	Lege 8_{16} auf den N. Datenbus
`000FOut`N	Lege F_{16} auf den N. Datenbus
`0010Out`N	Lege 10_{16} auf den N. Datenbus
`0020Out`N	Lege 20_{16} auf den N. Datenbus
`0040Out`N	Lege 40_{16} auf den N. Datenbus
`0080Out`N	Lege 80_{16} auf den N. Datenbus
`00FFOut`N	Lege FF_{16} auf den N. Datenbus
`0100Out`N	Lege 100_{16} auf den N. Datenbus
`0200Out`N	Lege 200_{16} auf den N. Datenbus
`0300Out`N	Lege 300_{16} auf den N. Datenbus
`0400Out`N	Lege 400_{16} auf den N. Datenbus
`0500Out`N	Lege 500_{16} auf den N. Datenbus
`0600Out`N	Lege 600_{16} auf den N. Datenbus
`0700Out`N	Lege 700_{16} auf den N. Datenbus
`8000Out`N	Lege 8000_{16} auf den N. Datenbus
`FF00Out`N	Lege $FF00_{16}$ auf den N. Datenbus
`FFF0Out`N	Lege $FFF0_{16}$ auf den N. Datenbus
`FFFFOut`N	Lege $FFFF_{16}$ auf den N. Datenbus
Externe Geräte	
`NoiseOn`	Sirene einschalten
`NoiseOff`	Sirene ausschalten
`WaterOn`	Sprinkler einschalten
`WaterOff`	Sprinkler ausschalten
Sonstige	
`SetFlags`	Flagregister nach der Ausführung der ALU-Operation aktualisieren
`Jump`	Springe zu der angegebenen Stelle im Mikrocode
`JumpNotEqual`	Springe zu der angegebenen Stelle im Mikrocode, wenn das NotZero-Flag gesetzt ist
`JumpNotNeg`	Springe zu der angegebenen Stelle im Mikrocode, wenn das NotNeg-Flag nicht gesetzt ist
`NextStep Read`N	Lade den N. Operanden
`NextStep Execute`	Den Befehl ausführen
`NextStep Write`	Das Ergebnis zurückschreiben
`IPInc`	Vergrößere den Befehlspointer um 1
`IPDec`	Verkleinere den Befehlspointer um 1
`IncT4NotNeg`	Vergrößere das Register `T4` um 1, wenn das Flag NotNeg gesetzt ist und der Wert des Registers nicht 255 ist.
`F[0]Out`N	Flag 0 (NotNeg) auf jedes Bit des N. Datenbusses legen

3.4 Mikrocode-Kodierung

Ein MikroCode-Befehl besteht aus 77Bit, die wie in der nachfolgenden Grafik aufgeteilt werden.



Dabei beschreibt der Teil Schreibbefehl, welche Signale auf den Bus gelegt werden und zwar unterteilt in `Bus1`, `Bus2` und `Bus3`. In der nachfolgenden Tabelle wird die Kodierung der 6 Bit eines Busses dargestellt, wobei die letzten zwei Bits xx nach der Reihenfolge der darauf folgenden Beschreibungen auf 00, 01, 10 und 11 gesetzt werden und frei meint das den folgenden Bitkombinationen keine Bedeutung zugewiesen wird.

Bits	Bedeutung
$0000xx$	A, B, C, D
$0001xx$	FlagRegister, ALUErg, MemDat, CCDat
$0010xx$	T0, T1, T2, T3
$0011xx$	T4, T5, T6, T7
$0100xx$	O_2, CO, CO_2, Smoke
$0101xx$	ROM O_2, ROM CO, ROM CO_2, ROM Smoke
$0110xx$	ROM Dat, ProgramDat, ROM Dat, ROML Dat
$0111xx$	CamAdr, CamDat, Temp, Flag0
$1000xx$	0, 4_{16}, 8_{16}, F_{16}
$1001xx$	10_{16}, 20_{16}, 40_{16}, 80_{16}
$1010xx$	FF_{16}, frei
$1011xx$	100_{16}, 200_{16}, 300_{16}, 400_{16}
$1100xx$	500_{16}, 600_{16}, 700_{16}, frei
$1101xx$	frei
$1110xx$	8000_{16}, frei
$0101xx$	frei
$1111xx$	$FF00_{16}$, $FFF0_{16}$, $FFFF_{16}$, frei

Der Lesebefehl beschreibt welche Register von welchem Bus Werte lesen. Dies geschieht in dem jedes Register was vom Bus lesen kann 2 Bits zu geordnet werden mit den Bedeutungen: nicht lesen 00, von Bus 1 lesen 01, von Bus 2 lesen 10 und von Bus 3 lesen 11. Die Reihenfolge der 22 Register die vom Bus lesen können im Lesebefehl ist wie folgt:
`IP`, `ROM Adr`, `ROMD Adr`, `ROML Adr`, `A`, `B`, `C`, `D`, `CCAdr`, `CCDat`, `T0`, `T1`, `T2`, `T3`, `T4`, `T5`, `T6`, `T7`, `IR`, `MemAdr`, `MemDat`, `FlagRegister`

ALUOp beschreibt ob die ALU sich Operanden vom Bus holen und verarbeiten soll oder nichts tun soll. Die ALU holt sich den ersten Operanden immer von Bus 1 und den zweiten immer von Bus 2. Die Kodierung dafür befindet sich in der folgenden Tabelle.

Bits	ALU-Operation
0000	keine
0001	Max
0010	MulH
0011	keine
0100	OR
0101	AND
0110	EQV
0111	XOR
1000	Add
1001	Sub
1010	Dec
1011	Inc
1100	Adress
1101	Check
1110	SpecialOr
1111	keine

3.5 Befehlssatz + Mikrocode

3.5.1 Aufbau der Befehlsworte und Operanden

Jeder Befehl hat den folgenden Aufbau (die Zahlen in den Klammern geben die Größe in Bits an):

Wort 1 (16)				Wort 2 (16)	Wort 3 (16)
X	OpCode (5)	Operand 1 (5)	Operand 2 (5)	Wert 1 (Immediate/Adr.)	Wert 2 (Immediate/Adr.)

Dabei sind Wert1 und Wert2 optionale Immediates bzw. Speicheradressen. Die obige Konstellation entspricht dem Assemblerbefehl `OpCode Operand2, Operand1`. Das mit X markierte Feld wird bei der Verarbeitung ignoriert.

3.5.2 Verarbeitung des Mikrocodes

Die Steuereinheit verfügt über einen internen Mikro-Programm-Zähler, welcher zu Beginn auf 0 steht und mit jedem Takt um 1 vergrößert wird. Somit wird zunächst der Inhalt der Mikrocodespeicherzelle 0 (Fetch) ausgelesen und verarbeitet, dann die Speicherzelle 1 usw.
Alternativ kann durch ein Steuersignal veranlasst werden, dass statt der Inkrementierung ein neuer Wert direkt in das Register gelesen wird. Damit werden auch die Mikro-Sprungbefehle realisiert.
Diese neue Adresse kann auf zwei verschiedene Arten ermittelt werden:

1. Sie wird aus dem aktuellen Mikrocode-Wort gelesen.

   Dieser Fall tritt genau dann auf, wenn $\texttt{Jump} \vee (\texttt{JumpNotNeg} \wedge \texttt{NotNeg}) \vee (\texttt{JumpNotEqual} \wedge \texttt{NotZero})$ wahr ist. Dann werden die Bits 16 bis 25 des Mikrocodes auf auf die Eingänge dies Mikrocode-Zählregisters statt auf die eigentlichen Steuersignale umgeleitet.

2. Sie wird aus dem Adress-ROM des Mikrocodes gelesen.

   Dieser enthält 128 Speicherzellen (0 bis 127), in welchen die Einsprungadressen für die verschiedenen Etappen der Befehlsausführung gespeichert sind. Im Fall von $\texttt{RI} \vee \texttt{RII} \vee \texttt{Ex} \vee \texttt{JW}$ wird der Datenausgang dieses ROMs auf die Setz-Eingänge des Zählregisters gelegt, welches dann den Wert übernimmt.

   Die Adresse, von welcher das Sprungziel aus dem ROM geladen wird, ist:

   (a) 00<*Operand*1>, wobei <*Operand*1> die 5 Bits aus dem Befehlswort sind, welche den ersten Operanden beschreiben.
   Dieser Fall tritt bei $\texttt{RI} = 1$ (Steuersignal `NextStep Read1`) auf.

   (b) 01<*Operand*2>, wobei <*Operand*2> die 5 Bits aus dem Befehlswort sind, welche den zweiten Operanden beschreiben.
   Dieser Fall tritt bei $\texttt{RII} = 1$ (Steuersignal `NextStep Read2`) auf.

   (c) 10 <*Opcode*>, wobei <*Opcode*> die 5 Bits aus dem Befehlswort sind, welche den Opcode enthalten.
   Dieser Fall tritt bei $\texttt{Ex} = 1$ (Steuersignal `NextStep Execute`) auf.

   (d) 11<*Operand*1>, wobei <*Operand*1> die 5 Bits aus dem Befehlswort sind, welche den ersten Operanden beschreiben.
   Dieser Fall tritt bei $\texttt{JW} = 1$ (Steuersignal `NextStep Write`) auf.

3.5.3 Fetch

Lädt den nächsten Befehl aus dem Speicher. IP wird direkt zur Adressierung des nächsten Befehlswortes verwendet.
Takte: 1

`0000 ProgOut1, IRIn1, IPInc, NextStep Read1`	Befehlswort in `IR` laden

3.5.4 FetchOperand 1

None

Lädt keinen ersten Parameter.
Takte: 1

`0001 NextStep Read2`	Gleich weiter mit dem zweiten Parameter

ReadImmediate

Lädt das Immediate als ersten Parameter.
Takte: 1

`0002 ProgOut1, T1In1, IPInc, NextStep Read2`	Wert in `T1` laden

ReadIndirekt

Lädt den Wert der Speicherzelle, welche durch das Immediate adressiert wird, als ersten Parameter.
Takte: 2

`0003 ProgOut1, MemAdrIn1, IPInc, MemRead`	Immediate als Adresser verwenden
`0004 MemDatOut1, T1In1, NextStep Read2`	Wert in `T1` laden

ReadRegister

Lädt das Register <Reg> als ersten Parameter. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 1

`0005 <Reg>Out1, T1In1, NextStep Read2`	Wert des Registers speichern

ReadRegInd

Lädt den Wert der Speicherzelle [<Reg>] als ersten Parameter. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 2

`0009 <Reg>Out1, MemAdrIn1, MemRead`	Registerwert als Adresse verwenden
`000A MemDatOut1, T1In1, NextStep Read2`	Wert der entsprechenden Speicherzelle laden

ReadRegIndPreDec

Verringert den Wert des Registers <Reg> um 1 und verwendet anschließend den Wert des Registers als Adresse des ersten Operanden. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 3

`0011 <Reg>Out1, Dec`	Wert verringern
`0012 AluOut1, <Reg>In1, MemDatIn1`	Adresse speichern
`0013 MemDatOut1, T1In1, NextStep Read2`	Operand speichern

ReadRegIndPreInc

Verringert den Wert des Registers <Reg> um 1 und verwendet anschließend den Wert des Registers als Adresse des ersten Operanden. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 3

`001D <Reg>Out1, Inc`	Wert verringern
`001E AluOut1, <Reg>In1, MemDatIn1`	Adresse speichern
`001F MemDatOut1, T1In1, NextStep Read2`	Operand speichern

ReadRegIndPostInc

Verwendet das Register <Reg> als Adresse für den ersten Operanden und vergrößert den Wert des Registers anschließend um 1. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 2

`0029 <Reg>Out1, MemDatIn1, Inc`	Wert des Registers als Adresse verwenden und vergrößern
`002A MemDatOut1, T1In1, AluOut2, <Reg>In2, NextStep Read2`	Operand und inkrementierten Registerwert speichern

3.5.5 FetchOperand 2

None

Lädt keinen zweiten Parameter.
Takte: 1

`0031 NextStep Execute`	Gleich weiter mit der Ausführung

ReadImmediate

Lädt das Immediate als ersten Parameter.
Takte: 1

`0032 ProgOut1, T1In1, IPInc, NextStep Execute`	Wert in `T1` laden

ReadIndirekt

Lädt den Wert der Speicherzelle, welche durch das Immediate adressiert wird, als ersten Parameter.
Takte: 2

`0033 ProgOut1, MemAdrIn1, IPInc, MemRead`	Immediate als Adresser verwenden
`0034 MemDatOut1, T1In1, NextStep Execute`	Wert in `T1` laden

ReadRegister

Lädt das Register <Reg> als zweiten Parameter. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 1

`0035 <Reg>Out1, T1In1, NextStep Execute`	Wert des Registers speichern

ReadRegInd

Lädt den Wert der Speicherzelle [<Reg>] als zweiten Parameter. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 2

`0039 <Reg>Out1, MemAdrIn1, MemRead`	Registerwert als Adresse verwenden
`003A MemDatOut1, T1In1, NextStep Execute`	Wert der entsprechenden Speicherzelle laden

ReadRegIndPreDec

Verringert den Wert des Registers <Reg> um 1 und verwendet anschließend den Wert des Registers als Adresse des zweiten Operanden. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 3

`0041 <Reg>Out1, Dec`	Wert verringern
`0042 AluOut1, <Reg>In1, MemDatIn1`	Adresse speichern
`0043 MemDatOut1, T1In1, NextStep Execute`	Operand speichern

ReadRegIndPreInc

Verringert den Wert des Registers <Reg> um 1 und verwendet anschließend den Wert des Registers als Adresse des zweiten Operanden. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 3

`004D <Reg>Out1, Inc`	Wert verringern
`004E AluOut1, <Reg>In1, MemDatIn1`	Adresse speichern
`004F MemDatOut1, T1In1, NextStep Execute`	Operand speichern

ReadRegIndPostInc

Verwendet das Register <Reg> als Adresse für den zweiten Operanden und vergrößert den Wert des Registers anschließend um 1. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 2

`0059` <Reg>`Out1`, `MemDatIn1`, `Inc`	Wert des Registers als Adresse verwenden und vergrößern
`005A` `MemDatOut1`, `T1In1`, `AluOut2`, <Reg>`In2`, `NextStep Execute`	Operand und inkrementierten Registerwert speichern

3.5.6 Arithmetische Befehle

Add

Addiert die Werte der Operanden.
Takte: 2

`0061` `T0Out1`, `T1Out2`, `Add`, `SetFlags`	Addieren und Flags setzen
`0062` `AluOut1`, `T0In1`, `NextStep Write`	Wert speichern und zurückschreiben

AddS

Die beiden niederwertigsten Halbbytes des zweiten Operanden werden durch ein bitweises Oder verknüpft und das Ergebnis dieser Operation als zweitniedrigstes Halbbyte des Zwischenergebnisses verwendet. Das niedrigste Halbbyte des Zwischenergebnisses wird auf 0 gesetzt und das höherwertige Byte wird gleich dem höherwertigen Byte des zweiten Operanden gesetzt. Dann wird das Zwischenergebnis zum Wert des ersten Operanden addiert und das Ergebnis der Addition im ersten Operanden gespeichert.
Takte: 3

`0063` `T1Out1`, `SpecialOr`	2. Operanden verarbeiten
`0064` `AluOut1`, `T0Out2`, `Add`, `SetFlags`	abschließende Addition
`0065` `AluOut1`, `T0In1`, `NextStep Write`	Wert speichern und zurückschreiben

Sub

Subtrahiert die Werte der Operanden.
Takte: 2

`0066` `T0Out1`, `T1Out2`, `Sub`, `SetFlags`	Addieren und Flags setzen
`0067` `AluOut1`, `T0In1`, `NextStep Write`	Wert speichern und zurückschreiben

Inc

Vergrößert den Wert des Operanden
Takte: 2

`0068` `T0Out1`, `Inc`, `SetFlags`	Vergrößern und Flags setzen
`0069` `AluOut1`, `T0In1`, `NextStep Write`	Wert speichern und zurückschreiben

3.5.7 Sprungbefehle

Jump

Führt einen unbedingten Sprung zu der als Immediate übergebenen Adresse aus.
Takte: 1
`006A T1Out1, IPIn1, Jump 0000`

JumpGreaterEqual

Führt einen Sprung zu der als Immediate übergebenen Adresse aus, wenn die letzte Operation ein nicht-negatives Ergebnis hatte.
Takte: 2

`006B JumpNotNeg 006D`	Wenn das Ergebnis nicht negativ war (d.h. größer oder gleich Null), dann wird gesprungen.
`006C Jump 0000`	
`006D T1Out1, IPIn1, Jump 0000`	Sprung

JumpLess

Führt einen Sprung zu der als Immediate übergebenen Adresse aus, wenn die letzte Operation ein negatives Ergebnis hatte.
Takte: 2

`006E JumpNotNeg 0070`	Wenn das Ergebnis nicht negativ war (d.h. größer oder gleich Null), dann wird gesprungen.
`006F T1Out1, IPIn1, Jump 0000`	Sprung
`0070 Jump 0000`	

JumpGreater

Führt einen Sprung zu der als Immediate übergebenen Adresse aus, wenn die letzte Operation ein positives Ergebnis hatte.
Takte: $2 - 3$

`0071 JumpNotNeg 0073`	
`0072 Jump 0000`	negativ $\Rightarrow$ kein Sprung
`0073 JumpNotEqual 0075`	
`0074 Jump 0000`	null $\Rightarrow$ kein Sprung
`0075 T1Out1, IPIn1, Jump 0000`	nicht negativ und nicht null $\Rightarrow$ Sprung

JumpLessEqual

Führt einen Sprung zu der als Immediate übergebenen Adresse aus, wenn die letzte Operation kein positives Ergebnis hatte.

Takte: 2 − 3

```
0076 JumpNotNeg 0078
0077 T1Out1, IPIn1, Jump 0000
0078 JumpNotEqual 007A
0079 T1Out1, IPIn1, Jump 0000
007A Jump 0000
```

negativ ⇒ Sprung

null ⇒ Sprung

nicht negativ und nicht null ⇒ kein Sprung

JumpLocAlert

Führt einen Sprung zu der als Immediate übergebenen Adresse aus, wenn das LocAlert-Flag gesetzt ist.
Takte: 3

```
007B FOut1, 0008Out2, And, SetFlags
007C JumpNotEqual 007E
007D Jump 0000
007E T1Out1, IPIn1, Jump 0000
```

Flag extrahieren

Flag gesetzt ⇒ Sprung

Sprung

JumpNoGlobAlert

Führt einen Sprung zu der als Immediate übergebenen Adresse aus, wenn das GlobAlert-Flag nicht gesetzt ist.
Takte: 3

```
007F FOut1, 0004Out2, And, SetFlags
0080 JumpNotEqual 0082
0081 T1Out1, IPIn1, Jump 0000
0082 Jump 0000
```

Flag extrahieren

Flag gesetzt ⇒ kein Sprung

Sprung

JumpDidBurn

Führt einen Sprung zu der als Immediate übergebenen Adresse aus, wenn das DidBurn-Flag gesetzt ist.
Takte: 3

```
0083 FOut1, 0010Out2, And, SetFlags
0084 JumpNotEqual 0086
0085 Jump 0000
0086 T1Out1, IPIn1, Jump 0000
```

Flag extrahieren

Flag gesetzt ⇒ Sprung

Sprung

JumpReset

Führt einen Sprung zu der als Immediate übergebenen Adresse aus, wenn das Reset-Flag gesetzt ist.
Takte: 3

```
0087 FOut1, 0020Out2, And, SetFlags
0088 JumpNotEqual 008A
0089 Jump 0000
```

Flag extrahieren

Flag gesetzt ⇒ Sprung

`008A T1Out1, IPIn1, Jump 0000`	Sprung

JumpSmoke

Führt einen Sprung zu der als Immediate übergebenen Adresse aus, wenn das Smoke-Flag gesetzt ist.
Takte: 3

`008B FOut1, 0040Out2, And, SetFlags`	Flag extrahieren
`008C JumpNotEqual 008E`	Flag gesetzt ⇒ Sprung
`008D Jump 0000`	
`008E T1Out1, IPIn1, Jump 0000`	Sprung

3.5.8 Flagoperationen

ClearLocAlert

Löscht das Flag für den lokalen Alarm.
Takte: 3

`008F FOut1, 0008Out2, Or`	Flag setzen
`0090 AluOut1, 0008Out2, Xor`	Flag umschalten
`0091 AluOut1, FIn1, Jump 0000`	Werte in Flagregister speichern

SetLocAlert

Setzt das Flag für den lokalen Alarm.
Takte: 2

`0092 FOut1, 0008Out2, Or`	Flag setzen
`0093 AluOut1, FIn1, Jump 0000`	Flags speichern

ClearGlobAlert

Löscht das Flag für den globalen Alarm.
Takte: 3

`0094 FOut1, 0004Out2, Or`	Flag setzen
`0095 AluOut1, 0004Out2, Xor`	Flag umschalten
`0096 AluOut1, FIn1, Jump 0000`	Flags speichern

SetGlobAlert

Setzt das Flag für den globalen Alarm.
Takte: 2

`0097 FOut1, 0004Out2, Or`	Flag setzen
`0098 AluOut1, FIn1, Jump 0000`	Flags speichern

ClearDidBurn

Löscht das Flag für den Hat-gebrannt-Status.
Takte: 3

`0099 FOut1, 0010Out2, Or`	Flag setzen
`009A AluOut1, 0010Out2, Xor`	Flag umschalten
`009B AluOut1, FIn1, Jump 0000`	Flags speichern

SetDidBurn

Setzt das Flag für den Hat-gebrannt-Status.
Takte: 2

`009C FOut1, 0010Out2, Or`	Flag setzen
`009D AluOut1, FIn1, Jump 0000`	Flags speichern

ClearReset

Löscht das Reset-Flag.
Takte: 3

`009E FOut1, 0020Out2, Or`	Flag setzen
`009F AluOut1, 0020Out2, Xor`	Flag umschalten
`00A0 AluOut1, FIn1, Jump 0000`	Flags speichern

3.5.9 Gerätesteuerung

NoiseOn

Schaltet die Sirene ein.
Takte: 1

`00A1 NoiseOn, Jump 0000`	Sirene einschalten

NoiseOff

Schaltet die Sirene aus.
Takte: 1

`00A2 NoiseOff, Jump 0000`	Sirene ausschalten

WaterOn

Schaltet die Sprinkleranlage ein.
Takte: 1

`00A3 WaterOn, Jump 0000`	Sprinkler einschalten

WaterOff

Schaltet die Sprinkleranlage aus.
Takte: 1

`00A4 NoiseOn, Jump 0000`	Sprinkler ausschalten

3.5.10 Spezialbefehle

ClearMem

Löscht den Speicherbereich, dessen Anfang und Ende durch den ersten bzw. zweiten Operanden angegeben werden.
Takte: $5 - 196610$

`00A5 T0Out1, 0000Out2, MemDatIn2, Or`	Die Null zum Schreiben an den Speicher übergeben und die aktuelle Adresse in den ALU-Ausgang schieben.
`00A6 AluOut1, T0In1, MemAdrIn1, T1Out2, MemWrite, Xor, SetFlags`	0 in die aktuelle Speicherzelle schreiben und überprüfen, ob bereits alle zu löschenden Zellen erreicht wurden
`00A7 T0Out1, Inc`	Pointer auf den zu löschenden Speicherbereich inkrementieren
`00A8 JumpNotEqual 00A7`	Schleife ggf. fortsetzen
`00A9 Jump 0000`	

Magic

Berechnet halbbyteweise $\max\{\texttt{C} - \mathit{Operand}2, 0\}$. Wenn das Ergebnis in einem Halbbyte größer als 0 ist, so wird das Halbbyte auf 1 gesetzt, sonst auf 0. Anschließend wird das Ergebnis im ersten Operanden gespeichert.
Takte: 9

`00AA COut1, FFF0Out2, And`	In den höherwertigen drei Halbbytes muss kein Unterlauf abgefangen werden.
`00AB T1Out1, FFF0Out2, AluOut3, T0In3, And`	
`00AC T0Out1, AluOut2, Sub`	Subtraktion ohne Unterlaufschutz durchführen
`00AD COut1, 000FOut2, And, AluOut3, T0In3`	Das niederwertigste Halbbyte extrahieren
`00AE T1Out1, 000FOut2, AluOut3, T2In3, And`	
`00AF T2Out1, AluOut2, Sub, SetFlags`	Subtraktion durchführen
`00B0 FOut01, AluOut2, And`	Im Fall eines negativen Ergebnisses wird das Ergebnis auf 0 gesetzt.
`00B1 T0Out1, AluOut2, Or`	Mit dem Ergebnis der höherwertigeren drei Halbbytes verknüpfen
`00B2 AluOut1, T0In1, NextStep Write`	Ergebnis speichern

GetSensorState

Lädt den aktuellen Status der Kamera und bestimmt die Anzahl der zu heißen Pixel, die maximale Temperaturdifferenz und die Temperatur des heißesten Pixels, welches die Schwelltemperatur überschreitet. Die Differenzen, um welche die Schwelltemperaturen überschritten werden, werden im Cache gespeichert.
Takte: ≥ 655386
Teil 1:
T0 - Adresse des ersten verarbeiteten Pixel
T1 - Zwischenspeicher für $65535 \cdot F[0]$
T2 - Temperatur des heißesten Pixels, das den Schwellenwert überschritten hat
T3 - Maximale Temperaturdifferenz, um welche der Schwellenwert überschritten wurde
T4 - Anzahl der Pixel, wo der Schwellenwert überschritten wurde
Kenngrößen: CO_2, CO, O_2, Rauch, Deckentemperatur, #Feuer, max. Temperatur, max. ΔTemperatur

`00B3 0000Out1, T2In1, T3In1, T4In1, WaitForCamAdrChange`	Kamera-Daten auswerten Register auf 0 setzen und warten, bis die Kamera den Wert des nächsten Pixels sendet, damit das übertragene Pixel nicht während der Verarbeitung des aktuellen Pixels wechselt.
`00B4 CamAdrOut1, T0In1, RomAdrIn1, CamCacheAdrIn1`	Die Adresse des ersten Pixels speichern, um eine Abbruchbedingung zu haben.
`00B5 CamDatOut1, RomDatOut2, Sub, SetFlags`	Temperatur des gemessenen Pixels bzw. Feldes mit dem Schwellenwert vergleichen
`00B6 F[0]Out1, T1In1, IncT4NotNeg, AluOut2, And`	Differenz auf 0 setzen, wenn die Schwelltemperatur nicht überschritten wurde
`00B7 AluOut1, T3Out2, Max`	Maximale Differenz berechnen
`00B8 T1Out1, CamDatOut2, And, AluOut3, T3In3`	Maximale Differenz speichern und Temperatur des zu heißen Pixels bestimmen
`00B9 AluOut1, T2Out2, Max`	Maximale Temperatur eines zu heißen Feldes berechnen
`00BA CamDatOut1, 0040Out2, Sub, SetFlags, AluOut3, T2In3`	Überprüfen, ob die Temperatur 64°C überschreitet, maximale Temperatur der zu heißen Pixel speichern
`00BB F[0]Out1, 0001Out2, And`	Berechnet 0, wenn das NotNeg-Flag nicht gesetzt und 1, wenn es gesetzt ist.
`00BC AluOut1, CamCacheDatIn1, CamCacheWrite, WaitForCamAdrChange`	Wenn das Pixel nicht kälter als 64°C ist, dann den entsprechenden Speicherplatz im Kamera-Cache mit 1 (sonst mit 0) füllen. Anschließend auf die Übertragung des nächsten Pixelwertes warten
`00BD CamAdrOut1, RomAdrIn1, CamCacheAdrIn1, T0Out2, Xor, SetFlags`	Abbruchbedingung prüfen
`00BE JumpNotEqual 00B5`	Nächstes Pixel verarbeiten

00BF	COIIOut1, LookUpAdrIn1, COIIRefOut2, Sub, SetFlags	Rauchzustand auswerten CO_2-Wert mit dem Grenzwert vergleichen und als Adresse für die Look-Up-Table verwenden
00C0	COOut1, CORefOut2, Sub, SetFlags, F[0]Out3, T0In3	CO-Wert mit dem Grenzwert vergleichen und CO_2-Vergleichsergebnis speichern
00C1	F[1]Out, T0Out2, Or, LookUpDatOut3, T1In3	CO- und CO_2-Vergleichsergebnisse kombinieren, Wert aus der Look-Up-Table laden
00C2	ORefOut1, OOut2, Sub, SetFlags, AluOut3, T0In3	O-Wert untersuchen und bisheriges Ergebnis speichern
00C3	F[1]Out, T0Out2, Or	O-, CO- und CO_2-Vergleichsergebnisse verknüpfen
00C4	SmokeOut1, SmokeRefOut2, Sub, SetFlags, AluOut3, T0In3	Rauch-Wert untersuchen und bisheriges Ergebnis speichern
00C5	F[1]Out, T0Out2, Or, SetFlags	Gesamt-Rauchwert bestimmen
00C6	FOut1, 0040Out2, Or	Flags verarbeiten
00C7	JumpNotEqual 00C9	Wenn das Ergebnis nicht 0 ist, wird das Rauch-Flag nicht wieder gelöscht.
00C8	AluOut1, 0040Out2, Xor	
00C9	COOut1, 0100Out2, AluOut3, FIn3, Or	Flags speichern und Adresse der Look-Up-Table für die CO-Auswertung bestimmen
00CA	OOut1, 0200Out2, AluOut3, LookUpAdrIn3, Or	Adresse der CO - Konzentrationsbewertung verwenden und Adresse zur Bewertung der O_2-Konzentration berechnen
00CB	T1Out1, LookUpDatOut2, AluOut3, LookUpAdrIn3, MulH	Adresse der O_2-Konzentration an Look-Up-Table senden, die Bewertungen der CO- und CO_2-Konzentration verknüpfen
00CC	SmokeOut1, 0300Out2, AluOut3, T1In3, Or	Verknüpfte Bewertungen speichern, Berechnung der Adresse der Rauchkonzentrationsbewertung
00CD	T1Out1, LookUpDatOut2, AluOut3, LookUpAdrIn3, MulH	Adresse der Rauchkonzentrationsbewertung zur Look-Up-Table schicken, Verknüpfung der ersten 3 Bewertungen berechnen
00CE	TempOut1, 0400Out2, AluOut3, T1In3, Or	Bewertung speichern, Berechnung der Adresse der Deckentemperaturenbewertung
00CF	T1Out1, LookUpDatOut2, AluOut3, LookUpAdrIn3, MulH	Adresse der Deckentemperaturenbewertung zur Look-Up-Table schicken, Verknüpfung der ersten 4 Bewertungen berechnen
00D0	T4Out1, 0500Out2, AluOut3, T1In3, Or	Bewertung speichern, Berechnung der Adresse der Bewertung der Anzahl der zu heißen Felder

00D1	T1Out1, LookUpDatOut2, AluOut3, LookUpAdrIn3, MulH	Adresse der Bewertung der Anzahl der zu heißen Felder zur Look-Up-Table schicken, Verknüpfung der ersten 5 Bewertungen berechnen
00D2	T2Out1, 0600Out2, AluOut3, T1In3, Or	Bewertung speichern, Berechnung der Adresse der Bewertung der höchsten Temperatur
00D3	T1Out1, LookUpDatOut2, AluOut3, LookUpAdrIn3, MulH	Adresse der Bewertung der höchsten Temperatur zur Look-Up-Table schicken, Verknüpfung der ersten 6 Bewertungen berechnen
00D4	T3Out1, 0700Out2, AluOut3, T1In3, Or	Bewertung speichern, Berechnung der Adresse der Bewertung der höchsten Temperaturdifferenz
00D5	T1Out1, LookUpDatOut2, AluOut3, LookUpAdrIn3, MulH	Adresse der Bewertung der höchsten Temperaturdifferenz zur Look-Up-Table schicken, Verknüpfung der ersten 7 Bewertungen berechnen
00D6	AluOut1, LookUpDatOut2, MulH	Verknüpfung aller Bewertungen berechnen
00D7	FFFFOut1, AluOut2, Sub	$1 - x$ bestimmen, damit größere Werte auch größere Feuergefahr bedeuten.
00D8	AluOut1, T0In1, NextStep Write	Gesamtwertung im ersten Operanden ablegen

Search

Sucht nach allen durch passierbare Wege verbundenen Türen.
Die Ausgabe erfolgt im Hauptspeicher ab der durch den ersten Parameter angegebenen Adresse. Das Ausgabeformat ist eine Auflistung aller Zusammenhangskomponenten und der in ihnen enthaltenen Türen, welche durch ihre Nummer (0 bis 127) dargestellt werden. Als Trennzeichen zwischen den Zusammenhangskomponenten wird die Zahl 65535 = $FFFF_{16}$ verwendet. Anfang und Ende der Aufzählung der Zusammenhangskomponenten werden durch zusätzliche Auftreten dieser Zahl markiert.
Takte: ≤ 45000000

Verwendung der temporären Register bei der Umformatierung des Kamera-Caches
T0 - Adresse der Rückgabe
T1 - X-Koordinate (zur Zeit zu beschreibende Zelle)
T2 - Y-Koordinate (zur Zeit zu beschreibende Zelle)
T3 - X-Koordinate (zur Zeit zu lesende Zelle, relativ zur zu beschreibenden Zelle)
T4 - Y-Koordinate (zur Zeit zu lesende Zelle, relativ zur zu beschreibenden Zelle)

Verwendung der temporären Register bei der Zusammenhangskomponentenbestimmung
T0 - Adresse in der Rückgabe
T1 - Nummer der Tür
T2 - aktuelle Position
T3 - aktuelle Position (Zwischenspeicher)
T4 - Anfang des Rückgabebereichs
T5 - Nummer des Raums beim Vergleich zur Vermeidung von Mehrfacheinträgen

T6 - Pointer zum suchen eventueller Mehrfacheinträge in der Zusammenhangskomponente
T7 - Anzahl der Türen

00D9 FFFFOut1, Inc	Formatierung des Kamera-Cache-Inhalts In den folgenden Schleifen wird der Inhalt des Kamera-Caches so modifiziert, dass in jedem Eintrag steht, ob eines der benachbarten Felder brennt.
00DA FFFFOut1, AluOut2, T1In2, Inc	1. Schleife: $\texttt{T1} = 0, ..., 255$
00DB T1Out1, 0004Out2, AluOut3, T2In3, Sub	2. Schleife: $\texttt{T2} = 0, ..., 255$
00DC T2Out1, 0004Out2, AluOut3, T3In3, Sub	3. Schleife: $\texttt{T3} = \texttt{T1} - 4, ..., \texttt{T1} + 4$
00DD T3Out1, AluOut2, T4In2, Check, SetFlags	4. Schleife: $\texttt{T4} = \texttt{T2} - 4, ..., \texttt{T2} + 4$
00DE JumpNotEqual 00E3	Keine gültige Position
00DF T3Out1, T4Out2, Adress	Adresse berechnen
00E0 AluOut1, CamCacheAdrIn1, CamCacheRead	Wert des Nachbarfelds laden
00E1 CamCacheDatOut1, 0001Out2, And, SetFlags	
00E2 JumpNotEqual 00F9	Nachbarfeld brennt, also kann das aktuelle Feld nicht betreten werden.
00E3 T4Out1, T2Out2, Sub	Abbruchbedingung der 4. Schleife
00E4 AluOut1, 0004Out2, Xor, SetFlags	
00E5 JumpNotEqual 00E7	Abbruchbedingung nicht erfüllt
00E6 Jump 00E9	Abbruchbedingung erfüllt
00E7 T4Out1, Inc	
00E8 Jump 00DD	4. Schleife fortsetzen
00E9 T3Out1, T1Out2, Sub	Abbruchbedingung der 3. Schleife
00EA AluOut1, 0004Out2, Xor, SetFlags	
00EB JumpNotEqual 00ED	Abbruchbedingung nicht erfüllt
00EC Jump 00EF	Abbruchbedingung erfüllt
00ED T3Out1, Inc	
00EE Jump 00DC	3. Schleife fortsetzen
00EF T2Out1, 00FFOut2, Xor, SetFlags	Es wurde kein brennendes Nachbarfeld gefunden, also muss die aktuelle Zelle nicht als unpassierbar markiert werden. Bzw.: Abbruchbedingung der 2. Schleife
00F0 JumpNotEqual 00F2	Abbruchbedingung nicht erfüllt
00F1 Jump 00F4	Abbruchbedingung erfüllt
00F2 T2Out1, Inc	
00F3 Jump 00DB	2. Schleife fortsetzen
00F4 T1Out1, 00FFOut2, Xor, SetFlags	Abbruchbedingung der 1. Schleife

`00F5 JumpNotEqual 00F7`	Abbruchbedingung nicht erfüllt
`00F6 Jump 00FE`	Abbruchbedingung erfüllt
`00F7 T2Out1, Inc`	
`00F8 Jump 00DA`	1. Schleife fortsetzen
`00F9 T1Out1, T2Out2, Adress`	aktuelles Feld als unbenutzbar markieren
`00FA AluOut1, CamCacheAdrIn1, CamCacheRead`	
`00FB CamCacheDatOut1, 8000Out2, Or`	
`00FC AluOut1, CamCacheDatIn1, CamCacheWrite`	
`00FD Jump 00EF`	Die 2. Schleife fortsetzen, um weitere Felder prüfen zu können.
`00FE 0000Out1, T1In1, 0080Out2, DoorAdrIn2, FFFFOut3, MemDatIn3`	Die Markierung aller unpassierbaren Felder durch einen Wert ungleich 0 ist abgeschlossen. Jetzt werden alle unpassierbaren Felder auf den Wert 8000_{16} gesetzt.
`00FF T0Out1, CamCacheAdrIn1, CamCacheRead`	
`0100 CamCacheDatOut1, 8000Out2, And`	
`0101 T0Out1, AluOut2, CamCacheDatIn2, CamCacheWrite, Inc, SetFlags`	
`0102 JumpNotEqual 00FF`	nächstes Feld verarbeiten
`0103 T0Out1, MemAdrIn1, DoorDatOut2, T7In2, 0000Out3, T1In3, Inc, MemWrite`	Vorbereitung der Bestimmung der Zusammenhangskomponenten: Das erste Trennzeichen der Ausgabe in den Speicher schreiben, Anzahl der Nachbarräume in `T7` speichern
`0104 AluOut1, T0In1, 000FOut2, And`	Türnummer in Raumnummer umwandeln
`0105 T7Out1, AluOut2, T0In2, Sub, SetFlags`	Ist die Raumnummer größer oder gleich der Anzahl der Räume?
`0106 JumpNotNeg 010E`	Die Tür gehört zu einem Raum, der nicht existiert.
`0107 0000Out1, T1In1, DoorAdrIn1`	Schleife: `T1` $= 0, ..., 127$ Diese Schleife trägt die Türen im Kamera-Cache ein, deren Raumnummer nicht größer oder gleich der Anzahl der Räume ist. Zunächst werden die Türpositionen geladen.
`0108 DoorDatOut1, CamCacheAdrIn1, CamCacheRead`	Position der Tür laden, Inhalt der zugehörigen Kamera-Cache-Zelle lesen
`0109 CamCacheDatOut1, T1Out2, Or`	Türnummer mit dem Inhalt der Cache-Zelle verknüpfen
`010A T1Out1, AluOut2, CamCacheDatIn2, CamCacheWrite, Inc`	Wert im Kamera-Cache Speichern

010B	AluOut1, T1In1, DoorAdrIn1, 0080Out2, Xor, SetFlags	Abbruchbedingung der Schleife prüfen
010C	JumpNotEqual 0104	Es wurden noch nicht alle Türen verarbeitet.
010D	Jump 0110	
010E	T1Out1, Inc	Die Tür nicht eintragen - wir versuchen es mit der nächsten Tür nochmal...
010F	Jump 010B	
0110	0000Out1, 0000Out2, Or	Nach den Zusammenhangskomponenten suchen Schleife: T1 $= 0, ..., 127$
0111	AluOut1, T1In1, DoorAdrIn1, 0080Out2, Xor, SetFlags	Adresse der ersten Tür einer Zusammenhangskomponente laden und Abbruchbedingung prüfen
0112	JumpNotEqual 0114	Abbruchbedingung nicht erfüllt
0113	Jump 017B	Alle Zusammenhangskomponenten getestet
0114	T1Out1, 000FOut2, And	Adresse der ersten Tür einer Zusammenhangskomponente laden
0115	T7Out1, AluOut2, Sub, SetFlags	Überprüfen, ob die Raumnummer einem existierenden Raum entspricht
0116	JumpNotNeg 0142	Raum existiert nicht
0117	DoorDatOut1, CamCacheAdrIn1, T3In1, CamCacheRead	Wert der zugehörigen Kamera-Cache-Zelle laden
0118	CamCacheDatOut1, FF00Out2, And, SetFlags	Feld unpassierbar?
0119	JumpNotEqual 0142	Feld unpassierbar
011A	CamCacheDatOut1, 0200Out2, Or	Feld kann betreten werden
011B	AluOut1, CamCacheDatIn1, 0007Out2, And, CamCacheWrite	Feld als Beginn der Suche markieren
011C	T0Out1, MemAdrIn1, AluOut2, MemDatIn2, Inc, MemWrite	Tür (bzw. den zugehörigen Raum) in die Ausgabe einfügen
011D	T3Out1, 00FFOut2, AluOut3, T0In3, And, SetFlags	(neuen) Ausgabepointer speichern, befinden wir uns am linken Rand?
011E	JumpNotEqual 0144	Ein Feld nach links
011F	T3Out1, FF00Out2, And, SetFlags	Können wir nach oben?
0120	JumpNotEqual 014F	Ein Feld nach oben
0121	T3Out1, 00FFOut2, And	Können wir nach rechts?
0122	AluOut1, 00FFOut2, Xor, SetFlags	
0123	JumpNotEqual 015A	Ein Feld nach rechts
0124	T3Out1, FF00Out2, And	Können wir nach unten?
0125	AluOut1, FF00Out2, Xor, SetFlags	

```
0126 JumpNotEqual 0165                                  Ein Feld nach unten

0127 T3Out1, CamCacheAdrIn1, CamCacheRead               Rückkehr aus der Rekursion
0128 CamCacheDatOut1, 0600Out2, And                     Ist dies das initiale Startfeld?
0129 AluOut1, 0200Out2, Xor
012A JumpNotEqual 012F                                  nicht das initiale Startfeld
012B T0Out1, MemAdrIn1, FFFFOut2, MemDatIn2,            Zusammenhangskomponente abschließen
     MemWrite, Inc
012C AluOut1, T0In1
012D T1Out1, Inc
012E Jump 0111

012F CamCacheDatOut1, 0700Out2, And                     aktuelle Zusammenhangskomponente weiter
                                                        verfolgen
0130 AluOut1, T4In1, 0400Out2, Xor, SetFlags
0131 JumpNotEqual 0135                                  kein Aufruf von rechts
0132 T3Out1, Inc                                        Rückkehr von einem Aufruf von rechts
0133 AluOut1, T3In1
0134 Jump 011F
0135 T4Out1, 0500Out2, Xor, SetFlags                    kein Aufruf von unten?
0136 JumpNotEqual 013A                                  kein Aufruf von unten
0137 T3Out1, 0100Out2, Add                              Rückkehr von einem Aufruf von unten
0138 AluOut1, T3In1
0139 Jump 0121
013A T4Out1, 0600Out2, Xor, SetFlags                    kein Aufruf von links?
013B JumpNotEqual 013F                                  kein Aufruf von links
013C T3Out1, Dec                                        Rückkehr von einem Aufruf von links
013D AluOut1, T3In1
013E Jump 0124
013F T3Out1, 0100Out2, Sub                              Rückkehr von einem Aufruf von oben
0140 AluOut1, T3In1
0141 Jump 0127

0142 T1Out1, Inc                                        Das Feld ist unpassierbar oder der Raum mit
                                                        der getesteten Tür existiert nicht.
                                                        Wir fügen die erste Tür nicht der Zusammen-
                                                        hangskomponente hinzu und probieren es mit
                                                        der nächsten Tür.
```

```
0143 Jump 0111                                          Weiter mit der nächsten Tür.

0144 T3Out1, Dec                                        Rekursionsaufruf: Ein Feld nach links
0145 AluOut1, CamCacheAdrIn1, T4In1, CamCacheRead
0146 CamCacheDatOut1, FF00Out2, And, SetFlags
0147 JumpNotEqual 011F                                  Abbruch, wenn das Feld brennt
0148 T4Out1, T3In1, T0Out2, MemAdrIn2
0149 CamCacheDatOut1, 0080Out2, And
014A JumpNotEqual 014D                                  Tür vorhanden
014B T0Out1, 0000Out2, Or                               ALU mit der aktuellen Ausgabeadresse füllen
014C Jump 011F                                          Tür bzw. Raum nicht eintragen
014D CamCacheDatOut1, 0400Out2, Or
014E Jump 016F                                          nächstes Feld

014F nachoben: T3Out1, 0100Out2, Sub                    Rekursionsaufruf: Ein Feld nach oben
0150 AluOut1, CamCacheAdrIn1, T4In1, CamCacheRead
0151 CamCacheDatOut1, FF00Out2, And, SetFlags
0152 JumpNotEqual 0121                                  Abbruch, wenn das Feld brennt
0153 T4Out1, T3In1, T0Out2, MemAdrIn2
0154 CamCacheDatOut1, 0080Out2, And
0155 JumpNotEqual 0158                                  Tür vorhanden
0156 T0Out1, 0000Out2, Or                               ALU mit der aktuellen Ausgabeadresse füllen
0157 Jump 0121                                          Tür bzw. Raum nicht eintragen
0158 CamCacheDatOut1, 0500Out2, Or
0159 Jump 016F                                          nächstes Feld

015A nachrechts: T3Out1, Inc                            Rekursionsaufruf: Ein Feld nach rechts
015B AluOut1, CamCacheAdrIn1, T4In1, CamCacheRead
015C CamCacheDatOut1, FF00Out2, And, SetFlags
015D JumpNotEqual 0124                                  Abbruch, wenn das Feld brennt
015E T4Out1, T3In1, T0Out2, MemAdrIn2
015F CamCacheDatOut1, 0080Out2, And
0160 JumpNotEqual 0163                                  Tür vorhanden
0161 T0Out1, 0000Out2, Or                               ALU mit der aktuellen Ausgabeadresse füllen
0162 Jump 0124                                          Tür bzw. Raum nicht eintragen
0163 CamCacheDatOut1, 0600Out2, Or
```

0164	`Jump 016F`	nächstes Feld
0165	nachunten: `T3Out1`, `0100Out2`, `Add`	Rekursionsaufruf: Ein Feld nach unten
0166	`AluOut1`, `CamCacheAdrIn1`, `T4In1`, `CamCacheRead`	
0167	`CamCacheDatOut1`, `FF00Out2`, `And`, `SetFlags`	
0168	`JumpNotEqual 0127`	Abbruch, wenn das Feld brennt
0169	`T4Out1`, `T3In1`, `T0Out2`, `MemAdrIn2`	
016A	`CamCacheDatOut1`, `0080Out2`, `And`	
016B	`JumpNotEqual 016E`	Tür vorhanden
016C	`T0Out1`, `0000Out2`, `Or`	ALU mit der aktuellen Ausgabeadresse füllen
016D	`Jump 0127`	Tür bzw. Raum nicht eintragen
016E	`CamCacheDatOut1`, `0700Out2`, `Or`	
016F	`AluOut1`, `CamCacheDatIn1`, `0007Out2`, `CamCacheWrite`, `And`	In die aktuelle Zelle eintragen, aus welcher Richtung wir sie betreten haben. Damit wird sie als besucht markiert und kann als Stack verwendet werden.
0170	`T0Out1`, `AluOut2`, `T5In2`, `Dec`	Überprüfung, ob der Raum bereits in der Zusammenhangskomponente eingetragen wurde. Zunächst setzen wir `T6` = `T1` − 1 und `T5` auf die Nummer des hinzuzufügenden Raumes.
0171	`AluOut1`, `T6In1`, `MemAdrIn1`, `MemRead`	Speicherzellenwert laden
0172	`MemDatOut1`, `T5Out2`, `Xor`, `SetFlags`	Speicherzellenwert mit der neuen Raumnummer vergleichen
0173	`JumpNotEqual 0175`	Werte sind nicht gleich
0174	`Jump 011D`	Werte sind gleich. Raum nicht eintragen
0175	`MemDatOut1`, `Inc`, `SetFlags`	Ist die aktuelle Zusammenhangskomponente durchsucht?
0176	`JumpNotEqual 0179`	Suche noch nicht abgeschlossen.
0177	`T0Out1`, `MemAdrIn1`, `T5Out2`, `MemDatIn1`, `MemWrite`, `Inc`	Suche abgeschlossen, keine Übereinstimmung gefunden ⇒ Raum in die Zusammenhangskomponente eintragen
0178	`Jump 011D`	Nächstes Feld
0179	`T6Out1`, `Dec`	Weiter suchen
017A	`Jump 0171`	

`017B T1Out1, MemAdrIn1, FFFFOut2, MemDatIn2, MemWrite`	Zusammenhangskomponenten wurden eingetragen. Ausgabe wird durch ein doppeltes Trennzeichen abgeschlossen

3.5.11 Sonstige Befehle

Cmp

Vergleicht die Werte der Operanden.
Takte: 1

`017C T0Out1, T1Out2, Sub, SetFlags, Jump 0000`	Addieren und Flags setzen

Mov

Speichert den Wert des 2. Operanden im 1. Operanden ab.
Takte: 1

`017D T1Out1, T0In1, Jump 0000`	Wert speichern und zurückschreiben

Clear

Setzt den Zieloperanden auf 0.
Takte: 1

`017E 0000Out1, T0In1, NextStep Write`	Ergebnis auf 0 setzen und zurückschreiben

3.5.12 WriteBack

None

Der erste Operand existiert nicht, also gibt es kein Ziel für das Zurückschreiben des Ergebnisses.
Takte: 1

`017F Jump 0000`	Gleich weiter mit dem nächsten Befehl

WriteRegister

Lädt das Ergebnis in das durch <Reg> angegebene Register. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 1

`0180 <Reg>In1, T0Out1, Jump 0000`

WriteRegInd

Lädt das Ergebnis in die Speicherzelle [<Reg>]. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 1

`0184 <Reg>Out1, MemAdrIn1, T0Out2, MemDatIn2, MemWrite, Jump 0000`

WritePreDec, WritePreInc

Lädt das Ergebnis in die Speicherzelle [<Reg>]. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 1

0188 <Reg>Out1, MemAdrIn1, T0Out2, MemDatIn2, MemWrite, Jump 0000	Der Wert des Registers wurde bereits beim Laden des Operanden verkleinert (PreDec) oder vergrößert (PreInc).

WritePostInc

Lädt das Ergebnis in die Speicherzelle [<Reg>]. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 2

018C <Reg>Out1, T1In1, Dec	Registerwert verkleinern, da er bereits beim Laden des Operanden vergrößert wurde, und den alten Wert zwischenspeichern
018D AluOut1, MemAdrIn1, T0Out2, MemDatIn2, T1Out3, <Reg>In3, MemWrite, Jump 0000	alten Wert wieder in das Register schreiben und den um 1 verkleinerten Wert als Rückschreibeadresse verwenden

WriteIndirect

Lädt das Ergebnis in die Speicherzelle mit der als Immediate übergebenen Adresse. Diese Mikrocode-Sequenz wird für jedes Register einmal in aufeinanderfolgenden Bereichen im Mikrocode abgelegt.
Takte: 4

```
0194 IPDec
0195
0196 ProgOut1, MemAdrIn1, MemRead, IPInc
0197 MemDatOut1, MemAdrIn1, T0Out2, MemDatIn2,
     MemWrite, Jump 0000
```

3.6 Assemblerprogramm

```
Begin:    GetSensorState
          JumpReset AlarmEnd
          Cmp D, #Schwellenwert_Feuer        //in D befindet sich die aktuelle Feuerwertung
          JumpGreaterEqual Fire             //falls Feuerwertung Feuer anzeigt
          JumpLocAlert NoFire               //falls es im vorherigen Zyklus im Raum gebrannt
                                            //hat
          JumpNoGlobAlert Begin             //falls alles ok und nirgends Feuer
          NoiseOn                           //es gibt globalen Alarm, aber Raum brennt nicht
          Jump StdWays                      //damit Fall alle Tueren miteinander verbunden

NoFire:   Cmp D, #Schwellenwert_KeinFeuer    //Testen wie weit das Feuer abgeschwaecht ist
          JumpGreaterEqual GetWays          //Feuer noch zu stark gilt immer noch als Brand
          ClearLocAlert                     //Feuer gilt als aus, also lokalen Alarm aus und
```

```
          WaterOff                        //setzen das es gebrannt hat
          SetHadFire
          Jump StdWays                    //Fall alle Tueren miteinander verbunden, da
                                          //globaler bestehen bleibt

Fire:     SetLocAlert             //Brannt im Raum erkannt Gegenmasnahmen einleiten
          SetGlobAlert
          WaterOn
          NoiseOn

GetWays:  JumpSmoke NoWays        //Noch passierbare Wege durch den Raum berechnen und falls
                                  //Rauch zu dicht/giftig Raum als nicht passierbar setzen
          Search 28               //sonst suche Zusammenhangskomponenten
          Jump StatusN            //Aktualisierung der Schilder nach Zusammenhangskomponenten

NoWays:   Mov [14], 256           //Raum voller Rauch und Feuer
          Mov [15], 256
          Mov [16], 256
          Mov [17], 256
          Mov [18], 256
          Mov [19], 256
          Mov [20], 256
          Mov [21], 256
          Mov [22], 256
          Mov [23], 256
          Mov [24], 256
          Mov [25], 256
          Mov [26], 256
          Mov [27], 256
          Jump begin

StdWays:  Mov A, #Aussentuer      //Raum brennt selbst nicht
          Cmp A, 0
          JumpGreater StdOutD     //hat Aussentuer
          JumpHadFire StdHadF     //es hatte im Raum gebrannt
          Clera C                 //Fall keine Aussentuer und es hat hier nie gebrannt
StdWEnd:  Add C, [0]
          Add C, [1]
          Add C, [2]
          Add C, [3]
          Add C, [4]
          Add C, [5]
          Add C, [6]
          Add C, [7]
          Add C, [8]
          Add C, [9]
          Add C, [10]
          Add C, [11]
          Add C, [12]
          Add C, [13]
```

```
          Magic [14],[0]
          Magic [15],[1]
          Magic [16],[2]
          Magic [17],[3]
          Magic [18],[4]
          Magic [19],[5]
          Magic [20],[6]
          Magic [21],[7]
          Magic [22],[8]
          Magic [23],[9]
          Magic [24],[10]
          Magic [25],[11]
          Magic [26],[12]
          Magic [27],[13]
          Jump begin

StdHadF:  Mov C, 4096             //keine Aussentuer und es hatte hier gebrannt
          Jump StdWEnd
StdOutD:  JumpDidBurn stdODHF
          Mov C, 1                //hat Aussentuer und es hatte hier nie gebrannt
          Jump StdWEnd
StdODHF:  Mov C, 4097             //hat Aussentuer und es hatte hier gebrannt
          Jump StdWEnd

StatusN:  Mov [14], 256           //Setzen des Wertes fuer eventuell brennende
          Mov [15], 256           //Tueren,da diese in keiner
          Mov [16], 256           //Zusammenhangskomponente sind
          Mov [17], 256
          Mov [18], 256
          Mov [19], 256
          Mov [20], 256
          Mov [21], 256
          Mov [22], 256
          Mov [23], 256
          Mov [24], 256
          Mov [25], 256
          Mov [26], 256
          Mov [27], 256
          Mov A, 29
loop1:    Mov C, 256

loop2:    Mov B, [A]              //Zaehlen der Statusbits in der Zusammenhangskomponente
                                  //in Vorwaertsrichtung
          Cmp B, 14
          JumpLess Uploop2
          JumpGreater loop3
          Add C, 16               //Aussentuer in der Zussammenhangskomponente
          Inc A
          Jump loop2
Uploop2:  AddS C, B               //normale Tuer in Zusammenhangskomponente
```

```
          Inc A
          Jump loop2

loop3:    Mov B, [--A]            //Berechnung der erreichbaren Stausbits in der
                                  //Zussamenhangskomponente in Rueckwaertsrichtung
          Cmp B, 14
          JumpLess UpLoop3
          JumpGreater loop4
          Jump loop3
Uploop3:  Mov D, 14
          Add D, B
          Magic [D], [B]
          Jump loop3

loop4:    Mov B, [++A]
          Cmp B, 14
          JumpLessEqual loop4
          Mov B, [++A]
          Cmp B, 14
          JumpLessEqual loop1
          Jump begin

AlarmEnd: WaterOff                //Alarm abschalteln und Ausgangsstatus herstellen
          NoiseOff
          ClearGlobAlert
          ClearLocAlert
          ClearHadFire
          ClearReset
          ClearMem 14, 27         //von ... bis ...
          Jump begin
```

4 Schlussbemerkungen

4.1 Bewertung der geplanten Einsatzmöglichkeiten

Diese Brandschutzanlage sollte natürlich eher in großen Komplex wie Schulen, Universitäten oder Bürogebäuden eingesetzt werden, als im privaten Haushalt, da gerade das Vorhandensein von Sprinkler und Fluchtschilder in Wohnungen und kleinen Häusern doch recht unorthodox ist. Natürlich kann man sich bei solchen Haushalten nur auf den Multikriterienmelder beschränken, der aufgrund seiner vielen Sensoren ein zuverlässiger aber nicht gerade billiger Schutzengel sein dürfte.

4.2 Bewertung der Einschränkungen

Eine der interessantesten Ideen zu diesem Projekt, die optimal errechnete Fluchtroute, konnte leider nur durch einen Kompromiss eingebaut werden, der die Leistung auf „Passierbar“, „Riskant“ oder „Unpassierbar“ reduziert, dafür aber Ressourcen schont. Wir sind der Meinung, dass es sich mit diesem Kompromiss gut leben lässt, da das primäre Ziel einer Brandschutzanlage, die Branderkennung, gerade bezüglich Arbeitsaufwand und Ressourcennutzung Vorrang haben sollte.

4.3 Fehlerbetrachtung

Der fehleranfälligste Zeitpunkt ist die Branderkennung. Leider sind einige der Brandkenngrößen nicht universell einsetzbar und müssen an die jeweilige Umgebung angepasst werden. Dies geschieht über die manuelle Eingabe einiger Schwellenwerte wie beispielsweise eine vom System maximal geduldete Deckentemperatur. Auch die Wärmebildkamera kann auf die jeweilige Umgebung justiert werden. Eine „Fischaugenoptik“ ermöglicht der Kamera auch bei relativ niedrigen Decken möglichst viel Fläche abzudecken. Damit aber keine Kerze oder Herdplatte die Werte der Kamera zu sehr in die Höhe treibt, wird analysiert wie viel Hitze die Kamera wahrnimmt und über welche Fläche diese verteilt ist. Diese manuellen Anpassungen sollten Fehler weitestgehend vermeiden.

4.4 Timingbedingungen bei zeitkritischen Abläufen

Timingbedingungen sind nur bei der Wärmebildkamera vonnöten. Und zwar darf sie nicht mehr als 500.000 Pixel die Sekunde bei einer Prozessor-Taktung von 10MHz ausgeben, da ansonsten die Verarbeitung eines Pixels nicht abgeschlossen wurde bevor das nächste Pixel übertragen wird.

4.5 Testmöglichkeiten und mögliche Erweiterungen

Es würde uns schon reizen zumindest solch einen Melder nachzubauen, um diesen unter realen Bedingungen zu testen. Alternativ wäre es möglich die Arbeit der Sensoren zu übernehmen und den Prozessor mit verschiedenen Daten zu füttern. Daten, die sofort auf einen Brand schließen lassen, aber auch Daten bei denen das nicht so leicht ist. Spielt man so verschieden Szenarien durch, hat man zumindest schon mal einen groben Überblick, ob das System funktioniert. Sinnvolle Erweiterungen des Systems wären der zuvor erwähnte Fluchtplan, der Einsatz verschiedener Löschmittel wie Schaum, Löschpulver oder verschiedenste Gase und die automatische Steuerung der Brandschutztüren. So kann das System beispielsweise über den Einsatz von Stickstoff als Löschmittel bestimmen, wenn der Raum verschlossen ist und sich keine Menschen in ihm aufhalten.

4.6 Aufgabenverteilung

1. Michael Fiedler
   (a) 1.2 Problemanalyse
   (b) 1.3 Festlegung der Basisfunktionalität
   (c) 2.4 Kommunikation mit den benachbarten Meldern
   (d) 2.5 Weitere Komponenten
   (e) 3.4 Mikro-Code Kodierung
   (f) 3.6 Assembler-Code
   (g) 4.2 Bewertung der Einschränkungen
   (h) 4.3 Fehlerbetrachtung
2. Stefan Przybilla
   (a) 1.1 Recherche
   (b) 1.6 Anwendungsszenario zum Schaubild
   (c) 2.1 Datentypen und -formate
   (d) 2.2 Sensoren und Aktuatoren
   (e) 3.1 Prozessoraufbau
   (f) 3.2 Flags
   (g) 3.3 Steuersignale
   (h) 4.1 Bewertung der geplanten Einsatzmöglichkeiten
   (i) 4.5 Testmöglichkeiten und mögliche Erweiterungen
3. Yves Radunz
   (a) 1.4 Einschränkungen
   (b) 1.5 Systemanforderungen
   (c) 2.3 Sensorenauswertung
   (d) 2.6 Speicher
   (e) 3.5 Befehlssatz + Mikrocode
   (f) 4.4 Timingbedingungen bei zeitkritischen Abläufen

4.7 Literatur- und Quellenverzeichnis

1. http://www.holzforschung.ch/Dokumente/Kotthoff_Ingolf_Physik_des_Feuers_Text.pdf (Stand: 19.05.2007)
2. http://de.wikipedia.org/wiki/Feuermelder (Stand: 10.05.2007)