Eindhoven University of Technology

MASTER


Aksit, M.

Award date:
1980

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
Group Electronic Circuits
Department of Electrical Engineering
EINDHOVEN UNIVERSITY OF TECHNOLOGY
Eindhoven, The Netherlands

T.H.E.R.M.B.
T.H.E. Reading Machine for Blind
By Means of Parallel Processing
and Pattern Recognition.

Nehmet Akgit

Submitted in partial fulfilment of the requirement for the degree of
Ir. (M.Sc.) at the Eindhoven University of Technology. This work was
carried out from December 1979 until December 1980 in the group of
Electronic Circuits.
TO THE BLIND PEOPLE
PREFACE

I believe that, the contemporary technology can only be something that we, technical persons be proud of, if we apply it to serve human kind. The starting point in this work is to think all human beings whose abilities are limited. I believe that true happiness is that which is shared with others. People who are not able to share our physical capabilities have the right of receiving help from every one, particularly from scientists.

I was fortunate to get the opportunity to develop THERMB (Reading machine for blind) during my final project in the Electronic circuits group at Eindhoven University of Technology. Here, I would like to express my appreciation to Ir. G.G. Persoon for supervising me in the development of this work. Special thanks go to Mr. O.L.J. Stoelinga for stimulating discussions. It is pleasure the acknowledge the encouragement and the interest shown by Prof. dr. ing. J. Jess. I am very grateful to Miss. Monique Hijskens for her help in typing the report. I am indebted to my wife for her patience.

Mehmet Aksit
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Introduction</td>
<td>1.1</td>
</tr>
<tr>
<td>2-Design of reading aids for blind</td>
<td>2.1</td>
</tr>
<tr>
<td>2.1-General</td>
<td>2.1</td>
</tr>
<tr>
<td>2.2-Types of reading machines and practical constraints</td>
<td>2.1</td>
</tr>
<tr>
<td>2.2.A-Direct translators</td>
<td>2.2</td>
</tr>
<tr>
<td>2.2.B-Recognition types</td>
<td>2.2</td>
</tr>
<tr>
<td>2.3-The blind as a reader</td>
<td>2.3</td>
</tr>
<tr>
<td>2.4-THERMB as a reading aid for the blind</td>
<td>2.4</td>
</tr>
<tr>
<td>2.4.A-What is THERMB</td>
<td>2.4</td>
</tr>
<tr>
<td>2.4.B-Some new features</td>
<td>2.4</td>
</tr>
<tr>
<td>2.4.C-Use of sound indications and a reading example</td>
<td>2.5</td>
</tr>
<tr>
<td>2.4.D-A simple reading aid</td>
<td>2.7</td>
</tr>
<tr>
<td>2.4.E-Adapting to different line spacing</td>
<td>2.9</td>
</tr>
<tr>
<td>2.4.F-Output units</td>
<td>2.10</td>
</tr>
<tr>
<td>2.4.G-Conclusions</td>
<td>2.11</td>
</tr>
<tr>
<td>References</td>
<td>2.11</td>
</tr>
<tr>
<td>3-Concurrent processing and THERMB</td>
<td>3.1</td>
</tr>
<tr>
<td>3.1-Principles of concurrent processing</td>
<td>3.1</td>
</tr>
<tr>
<td>3.1.A-Communication between processors</td>
<td>3.1</td>
</tr>
<tr>
<td>3.2-A concurrent language</td>
<td>3.3</td>
</tr>
<tr>
<td>3.3-Architecture of THERMB</td>
<td>3.5</td>
</tr>
<tr>
<td>3.4-Processor scheduling and mutual exclusion</td>
<td>3.7</td>
</tr>
<tr>
<td>3.4.A-Mutual exclusion</td>
<td>3.7</td>
</tr>
<tr>
<td>3.4.B-The scheduler</td>
<td>3.8</td>
</tr>
<tr>
<td>3.5-Processor communications</td>
<td>3.12</td>
</tr>
<tr>
<td>3.6-Deadlock prevention</td>
<td>3.16</td>
</tr>
<tr>
<td>3.7-Automatic task scheduling</td>
<td>3.18</td>
</tr>
<tr>
<td>3.8-Concurrent programming in assembly language</td>
<td>3.23</td>
</tr>
<tr>
<td>3.8.A-What is an assembly language</td>
<td>3.23</td>
</tr>
<tr>
<td>3.8.B-Programming in assembly language</td>
<td>3.24</td>
</tr>
<tr>
<td>References</td>
<td>3.27</td>
</tr>
<tr>
<td>Appendix to chapter 3: Explanation of assembly mnemonics of Z-80 processor</td>
<td>3.28</td>
</tr>
<tr>
<td>4-The reading process</td>
<td>4.1</td>
</tr>
<tr>
<td>4.1-Principles of optical character reading</td>
<td>4.1</td>
</tr>
<tr>
<td>4.2-The information conversion</td>
<td>4.2</td>
</tr>
<tr>
<td>4.2.A-General</td>
<td>4.2</td>
</tr>
<tr>
<td>4.2.B-Reading by vertical diode arrays</td>
<td>4.3</td>
</tr>
<tr>
<td>4.2.C-Matrix organized diode arrays</td>
<td>4.3</td>
</tr>
<tr>
<td>4.3-Picture deformation</td>
<td>4.7</td>
</tr>
<tr>
<td>4.4-The hand carried sensor</td>
<td>4.8</td>
</tr>
<tr>
<td>4.5-The input processor</td>
<td>4.10</td>
</tr>
<tr>
<td>4.5.A-Descriptions</td>
<td>4.10</td>
</tr>
<tr>
<td>4.5.B-The software</td>
<td>4.13</td>
</tr>
<tr>
<td>References</td>
<td>4.25</td>
</tr>
<tr>
<td>Appendix to chapter 4: The structure of the sampled characters in the memory</td>
<td>4.26</td>
</tr>
</tbody>
</table>
1-Introduction

Since about ten years, in semiconductor technology unexpected improvements have been realized. Integration of electronic components on one chip showed great progress. The integrated circuit design turned into integrated system design. This sudden development motivated many scientists to use this available processing power in many fields like "artificial intelligence" and "parallel processing".

The artificial intelligent machines require very fast, strong processors to handle the complex information to recognize, to learn, to think and to control.

Visually handicapped people have been waiting for the necessary developments in the technology to let them be able to cope over their great obstacle. Unfortunately, because this application is not that much profitable, many firms have showed interest only in commercial applications of the reading machines.

The existence of the strong cheap processors, photo-sensitive elements, the sufficient theoretical developments in the pattern recognition techniques, have motivated us to develop a strong, cheap, intelligent machine, which will let the blind read a printed text easily. Therefore THERM has been designed.

THERMB, T.H.E. Reading Machine for Blind is a parallel processing unit programmed to read printed texts by the help of an optical sensor. It is equipped with optional output units which give a sign to the blind to let him understand what the printed text is.

As it is a complete device project, to realize such a system covers many fields like social structure and demands of blind people, parallel processing, optical character reading, pattern recognition, etc. Therefore the project is divided mainly in four parts.
   A)-The reading machine for blind.
   B)-Parallel processing.
   C)-Optical character reading.
   D)-Pattern recognition.

To let the blind read easily, some new features are developed. Also some reading guides are proposed. The analysis of the reading machines for the blind is given in chapter 2.

In chapter 3 concurrent processing is discussed. After studying the basic concepts, how to design a parallel processor is explained. Also an automatic task scheduling process is used to be able to increase the number of processors easily. Without having any extra arrangement, unless the number of task are not exceeded, processors can be increased.

In chapter 4, principles of optical character reading discussed. The realization of combinations of all sequential operations to the naturally sequential inputting process, is explained. Here unlike many developed machines, the preprocessing task is almost completely eliminated from the recognition task.
In chapter 5, syntactic character recognition and realization of this method in THERMB is explained. Because of the required time to complete the device is much longer than the thesis time, with the present version only six capitals are recognized. All the characters of the English alphabet are examined. The recognition strategy makes it possible to increase the number of recognized characters by only adding some software routines based on developed techniques.
2. Design of reading aids for blind

2.1 General;

The purpose of a reading aid is to convert the printed page into an auditory or tactual display which can be understood by blind. For many years considerable efforts have been made for developing reading aids for blind people. It can be said that modern research on this subject began with the invention of photoelectric cells. However because of the difficulties that exist in the nature of the problem, some years had to pass to develop more successful machines. Difficulties were because of the technological limitations and unsatisfactory development of related subjects such as information theory and artificial intelligence.

Development of digital computers, transistors and photodiodes accelerated the work done. But still processors were large, expensive and slow, many related sciences were just introducing. Integration of more components into one chip showed great progress. LSI and VLSI circuits are developed. This decreased the cost and size of circuits dramatically. Microprocessors and large memory units started to appear. This sudden growth motivated many scientists in the field of parallel processing and very large data bases in the use of many fields like pattern recognition and image processing.

Besides that, sensory aids like charge coupled devices are developed to input characters with a good resolution. Also piezoelectric tactile stimulators are available for letting the blind to read easily.

All these available tools, cheap processors, memory units, reasonable growth in the theory of pattern recognition and parallel processing techniques give a lot of promises in the development of a low cost portable successful reading machines.

2.2 Types of Reading Machines and Practical Constraints;

Reading machines can be classified into two categories:

A- Direct translators.
B- Recognition types.
2.2

2.2.A Direct Translators;

In this approach the output of electrical photocell array is converted to different tones or tactile images. Experience has shown that blind readers need to spend a considerable amount of time learning to understand the output of these devices. The reading rate is also limited. A good reader may hope to reach maximum 90 words per minute. But average reading rates will be much lower.

Most known machines designed on this base are optophone, lexiphone and optacon [13,12]. The first two of them produce tones while optacon represents an output image by stimulators. All these machines require over a year of learning time. The excessive learning time is a result of the limited resolution of output devices. Like in the case of optacon copy is realized by 6x24 vibratory pins corresponding to 6x24 integrated photocells. To increase the speed also parallel channels are needed. All these requirements will cause expensive, bulky devices.

One main advantage of direct translators that they are language independent.

2.2.B Recognition Types;

These machines have more complex structure. They are built to read, recognize and put an ideal character code for output devices.

Complexity of these devices is a result of complex electronic circuits, large storage units and complicated algorithms. Technological growth is very much promising for realizing these complex electronic circuits in an economical way.

As the output of direct translators are never ideal or constant, the blind has to select the character between many possible cases. The cost of an unideal output will be paid by the blind by extensive training time. However, leaving recognition task to the machine will obviously let reading be faster than the translator type.

Although they have more complex structure, recognizers may cost much less than translators. For example, in tactile stimulation method, with character recognition, instead of having many tactile elements like 6x24 elements of optacon, 5x7 or even smaller matrices can be used to represent the standard characters. Optacon's main cost is the result of 144 mechanical moving elements.

One may think to sense character with higher resolution and by horizontal and vertical concentration lower it to 5x7 or smaller matrices. However, this concentration procedure will cause too much information loss and unselectable output at the end [33].

Spelled speech and tomorrow's talking machines require character code as an input. This can only be supplied by recognizer reading machines.
2.3. The Blind As a Reader;

It is not possible to design a successful reading aid for the blind, without studying their demands, socio-economic positions and organisations [4].

Demands of the blind can be grouped in the following points:

1- A low cost device.
2- Understandable and correct output.
3- Easy to read any text with any line spacing and font.
4- Easy to work with.
5- Fast enough.
6- Independent of a language.
7- Optional outputs.

The reading machine industry has not proceeded as many other parallel going subjects. Most of the companies which had interest in this field, preferred to develop reading machines for commercial applications such as reading bank checks and documents [4].

Statistics shows us that average income of blind people lies somewhere near minimum. Obviously any one who wishes to develop a machine to be used by large amount of blind people, must take economical constrains as essential starting point.

Whatever type of the machine is, the last recognition will be done by a human being. We do not read a text letter by letter but we read words as a whole. Here the context information plays the major role. Therefore failures at the output can be tolerated by the blind to some degree. However if the output is clear and correct, the blind will proceed much faster in reading.

First of all the machine must be handy. The sensor which is used to pick-up characters must be easily carried in hand. Bounded pages may cause problems in reading. Many readers developed in the past could work only on loose papers [5].

As there is much development in integration technology, miniaturization of this sensor is easily possible in the present day.

Also some control switches and reading guides must be made to be used by the blind in a comfortable way.

In this world there are enough languages and alphabets to disturb the generality of a text reader. A general purpose machine must give opportunity to read as many different alphabets and languages as possible.

Optional output units will give freedom to the blind to select the best suited unit. Some units may be more successful and sophisticated than others. But if their prices are more than the range that the user can afford, it is better to have something less successful but functioning.

Some units may be desirable in some cases more than others. For example if you want to store the output to be used later on, a machine which has recording facility on a magnetic tape will be much more feasible than only a talking one.
2.4. THERMB as a Reading Aid for the Blind;

2.4.A. What is THERMB;

THERMB is T.H.E. Reading Machine for Blind designed to help blind people to read printed texts. It is equipped with a hand carried sensor, a parallel processing unit and optional output devices. The blind person will hold the sensor with one hand to trace lines of the printed text. Depending on the output unit used, while he is tracing the line, he will get tactual or auditory information to let him know what the printed character is.

2.4.B. Some New Features;

While we were developing THERMB, requirements discussed in the previous section has been taken as serious constraints. It is not possible to say that they have been all satisfied. For example at the present model only Latin alphabet is studied. However it is not impossible to make library modules (ROMs) for other languages and alphabets. Some new methods have been developed to guide the blind during his reading. In this section we shall discuss these new features and point out their uses.

Some important features of THERMB are given below:

1- Adapts to reading speed variations.
2- Sound indication if the sensor is not placed on white paper. (Totally black)
3- Sound indication if character is not placed properly at the top side.
4- Sound indication if character is not placed properly at the bottom side.
5- Sound indication if character is properly placed.
6- Adapts to different line spacing. (optional)
7- Recognizes several fonts.
8- Optional output devices.
9- Fully integratable. (Except some output units e.g. loudspeaker)

Buffering capability makes it possible to tolerate different reading speeds. THERMB uses a vertical diode array to scan the character. The sampling rate is 1KHZ. This will tolerate even very fast reading rates e.g. 20 char/sec. In this case any expected delay can only be the result of uncompleted recognition. Some characters may need more time to reach final decision. Also some easy ones will be selected much faster than the average expected time. In practical cases, this will lead to a continuous reading independent of its speed.

The main difficulty for the blind is to trace the line correctly. This can be achieved by using expensive servo-controlled sensor following the line automatically. As THERMB is designed as low cost portable personal reading aid, such an expensive solution will not fit to the main design goal.
Until recently many authors agreed that in the coming future, it is not possible to realize talking machines. However, developments in integration technology and availability of some algorithms shows us that talking machines can be realized in short-term future [7]. When they are developed, these machines can also be connected to our system. It can be said that, when they have their successful form, talking machines will be the most desirable reading aid for the blind.

Piezo-electric stimulators are currently available. Because of their compact structure and because of the ideal output of THERMB, this method can be practically used as an output unit [2].

Three dimensionally printed paper tapes are tried in our group. This method gives also a possibility to store the output for using later on [3].

2.4.6. Conclusions:

In previous sections as a reading aid for blind some features of THERMB were discussed. Although I believe that it is a successful machine, to be sure of its practical success, it has to be tested by some blind people.

All the characters of English alphabet have been examined. The recognition strategy makes it possible to increase the number of recognized characters by only adding some software routines based on developed techniques. Also because of the required time to complete the device is much longer than the Thesis time, with the present version only six capitals are recognized.

REFERENCES:


3. Concurrent Processing and THERMB

3.1. Principles of Concurrent Processing;

Some Definitions:

If executions of processes are overlapping in time, then they are called "Concurrent Processes". If the number of processors are equal to number of processes, then this is "True Concurrency". In other cases if processors are switching to processes with suitably small intervals of time, "Apparent Concurrency" may be viewed. Processes which run concurrently can be disjoint or interactive processes. Concurrent processes which operate on disjoint data sets are "Disjoint Processes". As in many cases, processors which are working on common variables will create "Interactive Processing". This interactive executions of tasks may cause time dependent errors. A system is called "Functional", when the result of processing is a time independent function of the input.

3.1.A. Communication Between Processors;

Following aspects are important for the communication between processors:

A- Mutual Exclusion;

Resources that are used by processors can be shareable or non-shareable. Non-shareable resources can not be used by two or more processors at a time. This non-shareable property can be the result of mainly two reasons;

1- Physical nature. Example, a memory location can not be used by two processors exactly at the same time, otherwise bus contention occurs.

2- If a variable is accessible to more than one process and if one refers to the variable while the other one modifies it then the result is unpredictable. This will cause "Non-Functional" systems. In order to localize this inherent time dependency critical regions are defined. "Critical Regions" are regions where mutual exclusion is necessary to prevent non-functional systems. Following conditions must satisfy about critical regions;
a) Any process wishing to enter into a critical region will be enabled within a finite time.
b) Only one process can enter into a critical region but the order is irrelevant.
c) A process can stay inside the critical region for only a finite time.

B-Synchronization;
-------------

Generally, because of interruption and waiting of processes, the speed of one process relative to the other one is unpredictable. It is very possible that it is necessary to order operations in time. There are certain points at which processors must be synchronized. In this case a process can only proceed until another process has completed some activity.

C-Deadlock;
---------

If one resource is held by one processor which waits for a second resource and if this second resource is also held by another processor which waits for the first resource, none of the processor can proceed. A deadlock is a circular waiting for a condition which will never hold.

```
<table>
<thead>
<tr>
<th>Requires</th>
<th>Processor</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>_________</td>
<td>_________</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 5. A deadlock condition.
3.2. A Concurrent Language;

All the programs for THERMB are written in assembly language. For a person who is not familiar with the assembly language and the system, it may be difficult to trace the operations explained in following sections. Therefore a suitable ALGOL like concurrent language is defined to achieve easiness in system description.

1) - A PROGRAM:

BEGIN
DECLARATIONS;
STATEMENTS;
END.

2) - DECLARATION:

CONSTANT <constantname>; It is known to the system.

SHARED RECORD <common variables> END SHARED RECORD; Declaration of common variables.

VARIABLE TYPES:
--------------

INTEGER <variable name> ;

MEMORYBLOCK <memoryblockname>(SIZE=<size>,ADDRESS=<address>);

Private variables can be attached to a processor as shown:

<variabletype><variablename>(TYPE=<typeprocessor>,NAME=<nameprocessor>);

If parameters are not specified then they are irrelevant.

A program block of a processor is specified as;

PROGRAM BLOCK(TYPE=<typeprocessor>,NAME=<nameprocessor>,PROG=<nameprogram>);
<program>
END;

Any task is a procedure type:

PROCEDURE <procedurename>(procedure parameters);
procedure body
END;
3.4

3) STATEMENTS:

a) Concurrent operations can be started with:

CONCURRENcy BEGIN

PROGRAM BLOCK(TYPE=<processortype>, NAME=<processorname>, PROG=<programname>);
program block
END;

PROGRAM BLOCK(TYPE=<processortype>, NAME=<processorname>, PROG=<programname>);
program block
END;

.......................

CONCURRENcy END;

Precedence graph of concurrent statements are given below:

```
  /
 /  ____________
I    /     \
I   /     \
I \
I     \
I   I     I
I   I     I
I \
I     \
I   I     I
I   I     I
I \
I
```

Figure 6. Precedence graph of a concurrent processing.

b) Selection of one of a set of statements:

IF boolean THEN statement1 ELSE statement2;

IF boolean THEN statement;

c) Repetition of a statement while a boolean expression remains true:

WHILE boolean DO statement;

or continuous repetition:

REPEAT statement FOREVER;

d) Waiting until a boolean expression becomes true:

WAIT(boolean);
3.5

e)- Starting tasks waiting for a condition created by this task:

ACTIVATE ;

f)- Holding and releasing a resource :

HOLD(TYPE=<resource>, NAME=<resource>);

RELEASE(TYPE=<resource>, NAME=<resource>);

g) - Executing a task:

RUNPROGRAM(<task number>);

h)- To place data into a memory:

PUT(<variable(index)>, <data>);

Result of this operation is variable(index):= data

i)- To get data from a memory:

GET(<variable(index)>, <data>);

j)- To change the flow of the program :

GO TO <labelname>; Any program piece can be labelled as:

<labelname> Program piece.

3.3. Architecture of THERMB;

After reviewing the basic principles of concurrent processing, in this section we shall analyse the architecture of THERMB.

As THERMB is specially designed for a particular task, its architecture must also fit to the structure of this task. One has to study the technical demands, essentials of the chosen method, availability of software and hardware power, thus available processors with their peripherals, then he has to decide for the suitable structure.

Because of time constraints, parallel processing is a necessity. There are various parallel processing techniques developed to be used for various purposes. Here we shall not discuss all of them. An interested reader may refer to references[1]-[5]. Common used parallel computer architectures are array processors, pipeline processors, associative processors, multi processors.

In our case the pipe-line processor is found to fulfill the conditions best.

Here the term pipe-line is self descriptive and can be realised at various levels. The rate of flow in a petroleum pipeline depends on the rate at which petroleum is pumped into the pipeline and there is no direct relation with the time required to transfer this material. In pipe-line computers a portion of processing is performed by each of several processors and then it is given to the next processor in the line.

In our system pipe-lining is realized at a global view. Characters will be read by input processors and transferred to the central part for recognition and recognized characters will be outputted to output devices. In other words with ideal recognition, characters will be pumped from pages to output devices to be put into a suitable form for the blind. If inputting, outputting and recognition are fast
enough, the rate of character flow through out the machine is limited by reading. Internally Central processors are forming a Multiprocessor structure. This structure provides switching structures that allows each processor to communicate with each other. Processors are sharing the same task. Central processors are switching to these tasks to complete recognition. In figure 7 data flow in THERMB is given. Also in figure 8 block diagram of THERMB is drawn.

![Block diagram of THERMB](image)

Figure 7. Data flow in THERMB.

According to their functions, processors that are used in THERMB can be divided into three groups:

1) Input Processors
2) Central Processors
3) Output Processors
As it is seen in figure B, each processor is having its own program memory. Input processor inputs the data from the sensor and after necessary processing places the data into the common RAM. When the character is completely read in, central processors access to the common RAM to recognize the character. When the recognition is complete, the character code of the recognized character is outputted to the output processor via the output port. Output processor is responsible to control output devices.

3.4. Processor Scheduling and Mutual Exclusion;

3.4.A. Mutual exclusion;

Processors which tend to use common resources, in our case common data RAM, must be ordered to prevent bus contention and non-functional system.

For each processor common memory is defined as addresses between 8000H and FFFFH. To prevent bus contention only one processor must be allowed to use the common RAM at a time. If the common memory is held by one processor all other requesting processors should wait for their turns. Therefore priorities must be assigned to processors to prevent bus contention.

Any processor that wishes to use the common RAM generates a Common-Memory-Request signal. There is a circuit called scheduler which registers the request. If memory is free one processor will be enabled to use the resource. If the memory is busy processors will be scheduled according to priorities and request orders.

A processor may try to enter the common memory because of two reasons.

1) To read or write disjoint data to common RAM or divisible one instruction critical region operations.
2) To read or write inside the critical region where critical region consists of one or more indivisible instructions.

Writing and reading non-critical data requires exclusion only because of physical necessity.

Critical region operations require exclusion because of software necessity. If the critical region consists of only one divisible instruction hardware and software necessities can be combined. That means in case I common memory request signal can be decoded from control and address signals. By that way, common memory will be held by the requesting processor only during the memory request cycle. This cycle is about 200 nanoseconds. The hardware decoding of mutual exclusion request reduces idle times of processors.

If critical region consists of one or more indivisible instructions, software controlled exclusion is necessary. This can be obtained by loading an I/O port which is OR ed with the hardware decoded signal. Exclusion exists till the I/O port is modified by the software.

If the program is written at a higher level, the programmer may not be able to know exact machine instructions. Although it is more time consuming to write programs in machine language, especially for real-time systems, obtained results will be much more efficient than higher level programming.
Two instructions are necessary for mutual exclusion. (In our case the only shared resource is the RAM.)

HOLD;
RELEASE;

To detect critical regions one has to:

A) List all sharable variables.
B) List all references to these variables.
C) Put these program segments found in case B between HOLD RELEASE statements.

If these blocks are having only one write statement (which is not clear with this language), HOLD and RELEASE operations can be omitted.

3.4.8. Scheduler;

The scheduler is built for three processors to use the common memory. It is also possible to extend the capacity of the scheduler. This will be explained in chapter 6. Priorities of processors are:

Processor 3 Highest Priority
Processor 2
Processor 1 Lowest Priority

Process scheduler is a 16 state Moore type sequential circuit where the clock frequency is two times higher than the processor's clock cycles.

A synchronous Moore type sequential machine

\[ M = (I, O, S, \delta, \lambda) \]

Where \( I, O \) and \( S \) are finite, nonempty sets of inputs, outputs and states respectively.

\[ \delta : I \times S \rightarrow S \] is the state transition function
\[ \lambda : S \rightarrow O \] is the output function.

The cartesian product \( I \times S \) is the set containing all pairs of elements \( (I_i, S_j) \). The state transfer function associates with each pair \( (I_i, S_j) \) an \( S_k \) called the next state. Also output function associates for each state an output \( 0_k \).
Block diagram of the scheduler is:

One of the most used way to describe the behaviour of a sequential circuit is to draw its state diagram or transition table. The transition table of the scheduler is given below.

<table>
<thead>
<tr>
<th>INPUT</th>
<th>Present</th>
<th>Noreq.</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P1,P2</th>
<th>P1,P3</th>
<th>P2,P3</th>
<th>P1,P2,P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>J</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>N</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>J</td>
<td>F</td>
<td>E</td>
<td>G</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>J</td>
<td>F</td>
<td>I</td>
<td>K</td>
<td>X</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>J</td>
<td>F</td>
<td>N</td>
<td>E</td>
<td>P</td>
</tr>
<tr>
<td>E</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>J</td>
<td>X</td>
<td>X</td>
<td>E</td>
<td>G</td>
<td>X</td>
</tr>
<tr>
<td>F</td>
<td>X</td>
<td>D</td>
<td>X</td>
<td>X</td>
<td>M</td>
<td>F</td>
<td>X</td>
<td>G</td>
<td>X</td>
</tr>
<tr>
<td>G</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>J</td>
<td>X</td>
<td>X</td>
<td>G</td>
<td>S</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>M</td>
<td>X</td>
<td>X</td>
<td>H</td>
<td>T</td>
<td>E</td>
</tr>
<tr>
<td>I</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>B</td>
<td>X</td>
<td>F</td>
<td>I</td>
<td>K</td>
<td>A</td>
</tr>
<tr>
<td>J</td>
<td>X</td>
<td>D</td>
<td>X</td>
<td>X</td>
<td>J</td>
<td>N</td>
<td>X</td>
<td>L</td>
<td>T</td>
</tr>
<tr>
<td>K</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>F</td>
<td>X</td>
<td>K</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>N</td>
<td>X</td>
<td>L</td>
<td>T</td>
</tr>
<tr>
<td>M</td>
<td>X</td>
<td>X</td>
<td>C</td>
<td>X</td>
<td>M</td>
<td>X</td>
<td>I</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>B</td>
<td>X</td>
<td>N</td>
<td>E</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>I</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>E</td>
<td>P</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

X = Don't care condition

Table 1. State Transition Table of the Scheduler.
State descriptions are given below:

<table>
<thead>
<tr>
<th>STATES</th>
<th>DESCRIPTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>State= A: Request= No</td>
<td>Enable=---- Ordering= ----</td>
</tr>
<tr>
<td>State= B: Request= PR3</td>
<td>Enable=PR3 Ordering= ----</td>
</tr>
<tr>
<td>State= C: Request= PR2</td>
<td>Enable=PR2 Ordering= ----</td>
</tr>
<tr>
<td>State= D: Request= PR1</td>
<td>Enable=PR1 Ordering= ----</td>
</tr>
<tr>
<td>State= E: Request= PR3,PR2.</td>
<td>Enable=PR3 Ordering= PR3,PR2</td>
</tr>
<tr>
<td>State= F: Request= PR3,PR1.</td>
<td>Enable=PR3 Ordering= PR3,PR1</td>
</tr>
<tr>
<td>State= G: Request= PR3,PR2,PR1.</td>
<td>Enable=PR3 Ordering= PR3,PR2,PR1</td>
</tr>
<tr>
<td>State= H: Request= PR3,PR2,PR1</td>
<td>Enable=PR3 Ordering= PR3,PR1,PR2</td>
</tr>
<tr>
<td>State= I: Request= PR2,PR3</td>
<td>Enable=PR2 Ordering= PR2,PR3</td>
</tr>
<tr>
<td>State= J: Request= PR2,PR1</td>
<td>Enable=PR2 Ordering= PR2,PR1</td>
</tr>
<tr>
<td>State= K: Request= PR2,PR3,PR1</td>
<td>Enable=PR2 Ordering= PR2,PR3,PR1</td>
</tr>
<tr>
<td>State= L: Request= PR2,PR1,PR3</td>
<td>Enable=PR2 Ordering= PR2,PR1,PR3</td>
</tr>
<tr>
<td>State= M: Request= PR1,PR2</td>
<td>Enable=PR1 Ordering= PR1,PR2</td>
</tr>
<tr>
<td>State= N: Request= PR1,PR3</td>
<td>Enable=PR1 Ordering= PR1,PR3</td>
</tr>
<tr>
<td>State= O: Request= PR1,PR2,PR3</td>
<td>Enable=PR1 Ordering= PR1,PR2,PR3</td>
</tr>
<tr>
<td>State= P: Request= PR1,PR2,PR3</td>
<td>Enable=PR1 Ordering= PR1,PR3,PR2</td>
</tr>
</tbody>
</table>

Table 2. State descriptions.
The Output function of the Scheduler is also given with table 3.

<table>
<thead>
<tr>
<th>STATE</th>
<th>ENABLE PROCESSOR</th>
<th>IDLE PROCESSOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>A</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>B</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>C</td>
<td>f</td>
<td>T</td>
</tr>
<tr>
<td>D</td>
<td>T</td>
<td>f</td>
</tr>
<tr>
<td>E</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>F</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>G</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>H</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>I</td>
<td>f</td>
<td>T</td>
</tr>
<tr>
<td>J</td>
<td>f</td>
<td>T</td>
</tr>
<tr>
<td>K</td>
<td>f</td>
<td>T</td>
</tr>
<tr>
<td>L</td>
<td>f</td>
<td>T</td>
</tr>
<tr>
<td>M</td>
<td>T</td>
<td>f</td>
</tr>
<tr>
<td>N</td>
<td>T</td>
<td>f</td>
</tr>
<tr>
<td>O</td>
<td>T</td>
<td>f</td>
</tr>
<tr>
<td>P</td>
<td>T</td>
<td>f</td>
</tr>
</tbody>
</table>

T = TRUE
f = FALSE

Table 3. The output function of the scheduler.
3.5 Processor Communications;

Processor communications play a very important role in overall system performance. In THERMB processor communications are used because of these reasons.

1- Data Transfer
2- Synchronization

1- Data transfer.
Data transfer is realized by means of buffers, input output ports and common variables.

A. The Cyclical Buffer.

The buffer consists of finite number of elements called buffer blocks. These elements are arranged in a circle. This circle consists of a sequence of empty blocks to be filled by a producer and series of full blocks to be emptied by a consumer. There are also consumer and producer pointers where producer and consumer processors are referring. These pointers move anti-clockwise around without overtaking each other. There is also integer variable called BUFFERAVAILABLE which contains the number of empty blocks. The structure of the buffer is given in figure 10.

Empty buffer blocks

```
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
```

Producer pointer

```
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
```

Consumer

```
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
```

Pointer

```
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
```

Full buffer blocks

```
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
|               |               |
```

Figure 10. The cyclic buffer.

Implementation of a cyclic buffer;

```
```

Naturally memories are not cyclic in structure.
This cyclic appearance must be given by software. High level language description is;
Cyclic buffer implementation in algorithm 3.1:

BEGIN

CONSTANT MAXLENGTH;
SHARED RECORD
    INTEGER BUFFER AVAILABLE;
    INTEGER PRODUCER POINTER;
    MEMORYBLOCK BUFFERBLOCK;
END SHARED RECORD;

INTEGER CONSUMERPOINTER(TYPE=CENTRAL PROCESSOR);

CONCURRENCY BEGIN

PROGRAM BLOCK(TYPE=INPUT PROCESSOR);
REPEAT
    IF BUFFERAVAILABLE=0 THEN BEGIN
        MAKEPLACE; / WAIT(BUFFERAVAILABLE>0);
    END;
    CHARACTERNOTREADY:=TRUE;
    WHILE CHARACTERNOTREADY DO BEGIN
        INPUTSENSOR;
        PUT(BUFFERBLOCK(PRODUCERPOINTER),PROCESS(SENSOR));
    END;

    PRODUCEPOINTER:=PRODUCEPOINTER+1; BUFFERAVAILABLE:=BUFFERAVAILABLE-1;
    IF PRODUCEPOINTER=MAXLENGTH THEN PRODUCEPOINTER:=0;

FOREVER;
END PROGRAMBLOCK(TYPE=INPUTPROCESSOR);

PROGRAM BLOCK(TYPE=CENTRAL PROCESSOR);
REPEAT
    IF CONSUMERPOINTER=PRODUCERPOINTER THEN
        WAIT(PRODUCERPOINTER NOT EQL CONSUMERPOINTER);

    GET(BUFFERBLOCK(CONSUMERPOINTER),CHARACTER);
    RECOGNIZE(CHARACTER);
    CONSUMERPOINTER:=CONSUMERPOINTER+1; BUFFERAVAILABLE:=BUFFERAVAILABLE+1;
    IF CONSUMERPOINTER=MAXLENGTH THEN CONSUMERPOINTER:=0;

FOREVER;
END PROGRAMBLOCK(TYPE=CENTRALPROCESSOR);

CONCURRENCY END;
END.

The Assembly language listing is also given in section 3.8. An interested reader may refer to its practical realization.
Input output ports are non-shareable resources. That means I/O ports of one processor can only be accessible by that processor. The reason for this is to prevent port scheduling and useless of having common I/O ports for this application. I/O ports of processors are:

1. Input Processor:
   - Number of input ports = 4. Addresses = 0, 1, 2, 3.
   - Used for inputing data from the sensor.
   - Option = Input port. Inputing from Thumbwheel switch.
   - Number of output ports = 1. Address = Any address. Used for:
     - A) - To control the sensor.
     - B) - To command sound circuits.
     - C) - To interrupt other processors.
     - D) - Mutual exclusion request.

2. Central Processors
   - Only processor 2 communicates with the output processor.

   Processor 2
   - Number of I/O ports = 2. Addresses = 0, 1. Used for:
     - A) - To put out the recognized character to the output processor.
     - B) - To interrupt other processors.
     - C) - Mutual memory request.
     - D) - To communicate with other micro-computers (8085) optional.
     - E) - To input data from Thumbwheel switch (optional)

   Other central processors.
   - Number of output ports = 1. Address = any address. Used for:
     - A) - To interrupt other processors.
     - B) - Mutual memory request.

3. Output processor:
   - Number of output ports = Depending on the unit used.
   - Number of input ports = 1. To input the code of the recognized character.
2. Synchronization;

When processors are running dependent tasks, synchronization of processors at certain points are necessary. Unproper synchronization will increase the idle time of processors. In our system, interrupt based inactive synchronization is used. In this method a process can be put in a waiting state by the executing of the HALT statement. In this state the processor does not refer to common flags continuously, but waits inactive until another processor interrupts to indicate the continuation of the process.

In the busy way of waiting, processor refers to the common flags periodically. In figure 11, timing diagram of the busy way of waiting is shown.

\[
\begin{align*}
\text{Figure 11. Timing diagram of busy waiting.}
\end{align*}
\]

In this case, wasted time is \( T1 + T2 + T3 \). \( T1 \) may cause idling for other processors. If \( T2 \) is decreased then average \( T1 \) time will increase.

In the inactive way of waiting, the processor which waits for the other task, will be immediately activated. As common memory is not referred periodically, there will be less memory contention.

For synchronization there are two kinds of important statements.

\[ \text{WAIT(boolean)} ; \]
\[ \text{ACTIVATE} ; \]

The Activate command implies an interrupt to all processors. Those that happen to be in the waiting state will be activated and will then once again examine the condition that they are waiting for.

In the next example, synchronization of two processors are given:
Process synchronization in algorithm 3.2:

BEGIN
SHARED BOOLEAN CONDITION;
CONDITION:=FALSE;
CONCURRENCY BEGIN
PROGRAMBLOCK(TYPE=CENTRAL PROCESSOR);
S1;
S2;
WAIT(CONDITION);
S3;
S4;
END PROGRAMBLOCK(TYPE=CENTRAL PROCESSOR);

PROGRAMBLOCK(TYPE=CENTRAL PROCESSOR);
S5;
CONDITION:=TRUE;
ACTIVATE;
S6;
S7;
END PROGRAMBLOCK(TYPE=CENTRAL PROCESSOR);
CONCURRENCY END;
END.

3.6. Deadlock Prevention;

Existence and effects of deadlock is explained in section 3.1. In order to prevent deadlock, critical region conditions explained in 3.1 must be satisfied. With inspection only one deadlock possibility is found. The possibility of deadlock is more clear if we add HOLD and RELEASE statements in the algorithm 3.1. If the buffer is full and common memory is not released by the input processor, as central processor can not do stack operations, buffer will not be emptied by the central processor. Central processor will wait for the common RAM to write the program counter, input processor (holds common RAM) will wait for the condition that never satisfies.

Deadlock condition in algorithm 3.3:

BEGIN
CONSTANT MAXLENGTH;
SHARED RECORD
  INTEGER BUFFER AVAILABLE;
  INTEGER PRODUCER POINTER;
  MEMORYBLOCK BUFFERBLOCK;
  INTEGER TASKAVAILABLE;
END SHARED RECORD;
INTEGER CONSUMERPOINTER(TYPE=CENTRAL PROCESSOR);
PROCEDURE MAKEPLACE(TYPE=INPUTPROCESSOR);
  TASKAVAILABLE:"NUMBER OF THE TASK TO EMPTY BUFFERBLOCK";
  INTERRUPT OTHER PROCESSOR;
END MAKEPLACE;
PROCEDURE INTERRUPT(TYPE=CENTRAL PROCESSOR);

* Waits here. Needs Common RAM to copy program counter.
RUNPROGRAM(TASKAVAILABLE);
END PROGRAMLOCK;

CONCURRENCY BEGIN

PROGRAM BLOCK(TYPE=INPUT PROCESSOR);
REPEAT
HOLD;
IF BUFFERAVAILABLE=0 THEN BEGIN
MAKEPLACE;
WAIT(BUFFERAVAILABLE > 0);
* Waits here. Condition won't hold
END
RELEASE;
CHARACTERNOTREADY:=TRUE;
WHILE CHARACTERNOTREADY DO
BEGIN
  INPUT(SENSOR);
  PUT(BUFFERBLOCK(PRODUCERPOINTER),PROCESS(SENSOR));
END;
HOLD;
PRODUCEPOINTER:=PRODUCEPOINTER+1;BUFFERAVAILABLE:=BUFFERAVAILABLE-1;
IF PRODUCEPOINTER=MAXLENGTH THEN PRODUCEPOINTER:=0;
RELEASE;
FOREVER;
END PROGRAMBLOCK(TYPE=INPUT PROCESSOR);

PROGRAM BLOCK(TYPE=CENTRAL PROCESSOR);
REPEAT
IF CONSUMERPOINTER=PRODUCERPOINTER THEN
  WAIT(PRODUCERPOINTER NOT EQL CONSUMERPOINTER);
GET(BUFFERBLOCK(CONSUMERPOINTER),CHARACTER);
RECOGNIZE(CHARACTER);
CONSUMERPOINTER:=CONSUMERPOINTER+1;BUFFERAVAILABLE:=BUFFERAVAILABLE+1;
IF CONSUMERPOINTER=MAXLENGTH THEN CONSUMERPOINTER:=0;
FOREVER;
END PROGRAMBLOCK(TYPE=CENTRAL PROCESSOR);

CONCURRENCY END;
END.

In order to prevent deadlock procedure MAKEPLACE must be modified as:

PROCEDURE MAKEPLACE(TYPE=INPUTPROCESSOR);
  TASKAVAILABLE:="NUMBER OF THE TASK TO EMPTY BUFFERBLOCK";
RELEASE;
INTERUPT;
WAIT( interrupt is finished );
HOLD;
END MAKEPLACE;
To prevent deadlock, in procedure MAKEPLACE, Input processor must release the common RAM to let the Central processor empty one buffer block.

### 3.7. Automatic Task Scheduling

One input processor is sufficient for 1 KHZ scanning frequency and 32 bit resolution. 1 KHZ is high enough for a hand carried sensor. However 64 bit resolution might be necessary for lower case characters. One might accommodate this requirement either by a higher processing rate or by reducing the scanning frequency.

Here we shall discuss task scheduling for Central processors. Central processors are responsible for recognition. Because of extensive processing required for some characters, recognition may be the main reason for the delayed output.

An automatic task scheduling principle is used to be able to increase the number of processors easily. Without having any extra arrangement, the number of processors can be increased. With each processor insertion, tasks will be distributed to processors accordingly. When the number of processors are equal to the number of tasks, the maximum time required for recognition is equal to about the time required for the longest task.

The nature of syntactic recognition enables us to run all programs concurrently. Although the recognition process is apparently sequential in nature, it can be divided into tasks that, for the larger part, can be run concurrently. Only the final decisions have to be made sequentially.

Tasks may have these properties:

A) Activates other tasks.
B) Waits for a task.
C) Waits and activates other tasks.

All tasks are having this structure:

```
SEARCH FOR A SHAPE
DECIDE
```

If this decision needs the result of another task:

```
SEARCH FOR A SHAPE
WAIT(OTHER TASK READY)
DECIDE
```

If others need the decision of this task:

```
SEARCH FOR A SHAPE
DECIDE
ACTIVATE
```
Or if both conditions are combined:

- SEARCH FOR A SHAPE
- WAIT(OTHER TASK READY)
- DECIDE
- ACTIVATE

The time required for a decision is much less than the searching time. Soon after a task is activated, it will terminate. Here only capitals "O,C,G,O,S,A" are searched. All others are assumed to be out of the known class. Only processor 2 is allowed to start and stop the recognition process. When a new character is read in, processor 2 will first do pre-processing. If all conditions are ready for recognition, it initializes the next task. The processor which runs the next task will initialize the task following its task. By that way all tasks will be distributed to processors.

When the character is recognized, processor 2 will send the code of the character to the output processor and start the recognition process all again.

In figure 12 the principle of the task initialization process is shown. Here PR2 activates PRA and PRA activates PRB and so on. At the beginning of initialization, tasks are assigned to processors according to their priorities. In this example PRA has the highest PRn has the lowest priority. Any processor which finishes his task will get the waiting available task. This will continue till recognition is completed.

It can be seen from figure 12 that all tasks are even numbered. This is because of some practical realization reasons.

<table>
<thead>
<tr>
<th>TIME (Nonlinear)</th>
<th>PR2</th>
<th>PRA</th>
<th>PRB</th>
<th>PRm</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>WAIT(charac)</td>
<td>IDLE</td>
<td>IDLE</td>
<td>IDLE</td>
</tr>
<tr>
<td>T2</td>
<td>TA:=0</td>
<td>IDLE</td>
<td>IDLE</td>
<td>IDLE</td>
</tr>
<tr>
<td>T3</td>
<td>RUN(TA)</td>
<td>IDLE</td>
<td>IDLE</td>
<td>IDLE</td>
</tr>
<tr>
<td>T4</td>
<td>TA:=TA+2</td>
<td>--------&gt;</td>
<td>IDLE</td>
<td>IDLE</td>
</tr>
<tr>
<td>T5</td>
<td>TA:=TA+2</td>
<td>--------&gt;</td>
<td>IDLE</td>
<td>END</td>
</tr>
<tr>
<td>T6</td>
<td>TA:=TA+2</td>
<td>--------&gt;</td>
<td>IDLE</td>
<td>END</td>
</tr>
<tr>
<td>T7</td>
<td>TA:=TA+2</td>
<td>--------&gt;</td>
<td>IDLE</td>
<td>END</td>
</tr>
<tr>
<td>T8</td>
<td>TA:=TA+2</td>
<td>--------&gt;</td>
<td>IDLE</td>
<td>END</td>
</tr>
<tr>
<td>T9</td>
<td>TA:=TA+2</td>
<td>--------&gt;</td>
<td>IDLE</td>
<td>END</td>
</tr>
<tr>
<td>T10</td>
<td>TA:=TA+2</td>
<td>--------&gt;</td>
<td>IDLE</td>
<td>END</td>
</tr>
<tr>
<td>T11</td>
<td>TA:=TA+2</td>
<td>--------&gt;</td>
<td>IDLE</td>
<td>END</td>
</tr>
</tbody>
</table>

TA=TASKAVAILABLE

Figure 12. The flow diagram of the task initialization.
When a task requires the result of an another task then synchronization is necessary. In this particular application synchronization is only necessary at the decision phase. Tasks are depending on other tasks not for starting but for completing their executions.

To Recognize 6 Capitals "O,O,C,G,A,S", eight tasks are prepared. All these tasks are searching for a specific feature and they are even numbered from 0 till 14. Structure of these task scheduling and synchronization are given in algorithm 3.4. List of tasks are also given below:

<table>
<thead>
<tr>
<th>TASK</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASK 0</td>
<td>ACTIVATES TASK 2</td>
</tr>
<tr>
<td>TASK 2</td>
<td>WAITS TASK 0, ACTIVATES TASK 6</td>
</tr>
<tr>
<td>TASK 4</td>
<td>ACTIVATES TASK 6</td>
</tr>
<tr>
<td>TASK 6</td>
<td>WAITS TASK 2 AND TASK 4, ACTIVATES TASK 8</td>
</tr>
<tr>
<td>TASK 8</td>
<td>WAITS TASK 6, ACTIVATES TASK 12</td>
</tr>
<tr>
<td>TASK 10</td>
<td>ACTIVATES TASK 12</td>
</tr>
<tr>
<td>TASK 12</td>
<td>WAITS TASK 8 AND TASK 10, ACTIVATES TASK 14</td>
</tr>
<tr>
<td>TASK 14</td>
<td>WAITS TASK 12, ACTIVATES &quot;WAIT FOR RECOGNITION&quot;</td>
</tr>
</tbody>
</table>

The possible work flow diagram is given in figure 13. Here task lengths are not necessarily equal to real execution times.

<table>
<thead>
<tr>
<th>TIME (Nonlinear)</th>
<th>PR2</th>
<th>PRA</th>
<th>PRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>WAIT(character)</td>
<td>WAIT(task)</td>
<td>WAIT(task)</td>
</tr>
<tr>
<td>T2</td>
<td>TA:=0</td>
<td>IDLE</td>
<td>IDLE</td>
</tr>
<tr>
<td>T3</td>
<td>RUN(0)</td>
<td>IDLE</td>
<td>IDLE</td>
</tr>
<tr>
<td>T4</td>
<td>-----------&gt; RUN(2)</td>
<td>-----------&gt; RUN(4)</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>TA:=TA+2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>TA:=TA+2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td></td>
<td>WAIT(0)</td>
<td>END(4)</td>
</tr>
<tr>
<td>T9</td>
<td></td>
<td>activate</td>
<td>IDLE</td>
</tr>
<tr>
<td>T10</td>
<td>END(0)----------&gt; RUN(8)</td>
<td>----------&gt; TA:=TA+2</td>
<td></td>
</tr>
<tr>
<td>T11</td>
<td>RUN(10)</td>
<td>----------&gt; TA:=TA+2</td>
<td></td>
</tr>
<tr>
<td>T12</td>
<td>----------&gt; TA:=TA+2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T13</td>
<td></td>
<td>WAIT(6) activate</td>
<td></td>
</tr>
<tr>
<td>T14</td>
<td></td>
<td>RUN(12)&lt;----------- END(6)</td>
<td></td>
</tr>
<tr>
<td>T15</td>
<td></td>
<td>TA:=TA+2</td>
<td>RUN(14)</td>
</tr>
<tr>
<td>T16</td>
<td></td>
<td>STOP</td>
<td>&quot;wait&quot;</td>
</tr>
<tr>
<td>T17</td>
<td></td>
<td>RUN(wait)</td>
<td></td>
</tr>
<tr>
<td>T18</td>
<td></td>
<td>WAIT(task)</td>
<td>activate</td>
</tr>
<tr>
<td>T19</td>
<td>PUTCODEOUT&lt;------</td>
<td>END(12) ----&gt; RECOGNIZED</td>
<td></td>
</tr>
<tr>
<td>T20</td>
<td></td>
<td>RUN(wait)</td>
<td></td>
</tr>
<tr>
<td>T21</td>
<td></td>
<td>TA:=0</td>
<td>WAIT(task)</td>
</tr>
<tr>
<td>T22</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 13. The work flow diagram of scheduling and synchronization.
Recognition process starts at T2 by a character input. PR2 runs the first task while PRA and PRB follow PR2 by switching to tasks TASK 2 and TASK 4 respectively. At T8 processor B finishes his job and starts execution of task 6. Task 2 has a decision state. However as task 0 has not yet ended at TS, processor A waits for PR2. Immediately after task 0 finishes, PRA starts with task 8. The program flow will continue until PRB reaches to recognition. At this point as there are no available tasks PRA and PRB are waiting idle. After outputting the character code, the process will start again from the begin.

Automatic Task Scheduling In Algorithm 3.4.

BEGIN
SHARED RECORD
INTEGER TASKAVAILABLE;

PROCEDURE TASK 0
    TASKAVAILABLE:=2
    RELEASE;
    ACTIVATE;
    SEARCH(CURVE 1);
    DECIDE;
    ACTIVATE;
END;

PROCEDURE TASK 2;
    TASKAVAILABLE:=4
    RELEASE;
    SEARCH(S);
    WAIT(TASK 0);
    DECIDE;
    ACTIVATE;
END;

PROCEDURE TASK 4;
    TASKAVAILABLE:=6
    RELEASE;
    SEARCH(3 BLACKLINES);
    ACTIVATE;
END;

PROCEDURE TASK 6;
    TASKAVAILABLE:=8
    RELEASE;
    SEARCH(CURVE 2);
    WAIT(TASK 2);
    WAIT(TASK 4);
    DECIDE;
    ACTIVATE;
END;

PROCEDURE TASK 8;
    TASKAVAILABLE:=10
    RELEASE;
    SEARCH(A);
    WAIT(TASK 6);
    DECIDE;
    ACTIVATE;
END;
PROCEDURE TASK 10;
  TASKAVAILABLE:=12;
  RELEASE;
  SEARCH(0);
  ACTIVATE;
END;
PROCEDURE TASK 12;
  TASKAVAILABLE:=14;
  RELEASE;
  SEARCH(0)
  WAIT(8);
  WAIT(10);
  DECIDE;
  ACTIVATE;
END;
PROCEDURE TASK 14;
  TASKAVAILABLE:="NO TASK";
  RELEASE;
  SEARCH(C,G);
  WAIT(12);
  DECIDE;
END;
END SHARED RECORD;
CONCURRENCY BEGIN;
  PROGRAM BLOCK(TYPE=CENTRAL PROCESSOR, NAME=PR2);
  REPEAT
    WAIT(CHARACTER READ IN);
    PREPROCESS;
    HOLD;
    RUN PROGRAM(0);
  COMMENT NO TASK IS THE LARGEST POSSIBLE TASK NUMBER ;
  WHILE TASKAVAILABLE < NOTASK DO
    BEGIN;
    HOLD;
    RUN PROGRAM(TASKAVAILABLE);
  END;
  WAIT(RECOGNIZED);
  FOREVER;
END PROGRAM BLOCK;

  PROGRAM BLOCK(TYPE=CENTRAL PROCESSOR);
  REPEAT;
  WHILE TASKAVAILABLE < NOTASK DO
    BEGIN;
    HOLD;
    RUN PROGRAM(TASKAVAILABLE);
  END;
  WAIT(RECOGNIZED);
  FOREVER;
END PROGRAM BLOCK;

  CONCURRENCY END;
END.
3.8. Concurrent Programming in Assembly Language;

3.8.A. What is an assembly language;

Up to now, to explain the operations of THERMB a high level concurrent language is used. However practical realization of software was assembler programming. In order not to let the reader be unaware of actual operations, the assembly language of the processor Z-80 is explained.

A machine level language is a collection of zeros and ones, where they correspond to low and high voltage levels respectively. Assembly language is one to one mnemonic representation of the machine language.

A computer is consisting of:

A)- Central processing unit (C.P.U)
B)- Memory
C)- Input-Output ports (I/O).

A C.P.U. mainly does 3 operations:

A)- Internal operations.
B)- External memory operations.
C)- I/O operations.

I/O and memory operations are combined with internal operations to realize executions.

Main internal operations are:

A)- Instruction decoding.
B)- Data transfer.
C)- Data shifting.
D)- Arithmetical and logical operations.
E)- Condition test.
F)- Change the flow of execution, Call and Jump.
G)- CPU control operations.

External memory and I/O operations are a type:

Read from the resource
Write to the resource

The only difference between I/O and memory operations is that they are addressed separately. A program is a set of meaningful instructions where each consists of:

Operation code
or
Operation code + Data
or
Operation code + Address

The CPU is having internal registers where fast operations occur. They are addressable with their names like

\( A, B, C, D, E, F, H, L, IX, IY, SP, I, R, PC \)

exchange \((A, B, C, D, E, F, H, L)\)
External registers are addressed with their addresses or names assigned to these addresses. Operation codes are represented by their mnemonics. These mnemonic are derived from the English language. The assembly language listing is given in appendix A of this chapter. Direction of data flow is always from right to left e.g.

LD A, B means load A with B.

Indirect addressing is represented by brackets.

LD A, [HL] means load A from the location pointed by HL.

3.8. B. Programming in Assembly language;

HOLD and RELEASE statements:

HOLD LD A, [PORTDATA] Load A with PORTDATA.
RES 0, A First bit of A is 0.
OUT [0], A Output A from port 0.
LD [PORTDATA], A Load PORTDATA with A.

Release statement is the same, except bit 0 of A must be loaded by 1.

RELEASE LD A, [PORTDATA] bit 0 of A is 1.
SET 0, A
OUT [0], A
LD [PORTDATA], A.

Hold and release routines can be called whenever they are needed by labeling them with their names e.g.

CALL HOLD
CALL RELEASE

After completing HOLD and RELEASE routines, inorder to return to the main program, RET instruction must be added at the end.
3.25

TASK INITIALIZING AND EXECUTION:
-----------------------------------------------

Any task can be initialized by loading the location TASKAVAILABLE with a valid task number. It can be also seen from algorithm 3.4. that any initialized task initializes the following task. The memory must be hold during this operation. (critical region)

CALL \ HOLD \ HOLD MEMORY
LD \ A,[TASKAVAILABLE] \ WHAT IS THE TASK
LD \ B,A \ SAVE A IN B
INC \ A \ INCREMENT A TWO TIMES
INC \ [TASKAVAILABLE],A \ LOAD THE NEXT TASK
CALL \ RELEASE \ RELEASE THE MEMORY

Here B has the task number. Now the starting address of this task must be searched from TASKADDRESSES TABLE. This is done as it is shown.

LD \ A,B \ LOAD A WITH B
LD \ HL,TABLE2 \ LOAD HL WITH BEGINNING ADDRESS
OR \ A \ OR A WITH L TO GIVE OFFSET
LD \ E,[HL] \ GET LOW BYTE OF STARTING ADDR.
INC \ HL \ INCREMENT POINTER HL
LD \ D,[HL] \ GET HIGHER BYTE ADDRESS
EXX \ DE,HL \ EXCHANGE DE AND HL
JP \ [HL] \ JUMP TO THE ADDRESS OF HL

SYNCHRONIZATION:
-------------------

As it is mentioned in section 3.5, when a process needs a condition which will be set true by an another process, he will wait inactive till interruption occurs. The implementation of WAIT and ACTIVATE statements is explained by an example.

In this example, one process needs that bit 0 of the location GRUP1 must be set to 1. If this location is not 1, he will wait till interruption occurs. After interruption he will check if the condition satisfied.

WAIT :
LOOKIF \ LD \ HL,GRUP1 \ LOAD HL WITH ADDRESS OF GRUP1
BIT \ 0,[HL] \ TEST BITO OF GRUP1
JR \ NZ,READY \ IF IT IS 1 CONTINUE
HALT \ \ \ WAIT FOR AN INTERRUPT
JR \ LOOKIF \ \ \ AFTER INTERRUPT REPEAT SEARCHING

READY \ \ continue
To activate a processor means to interrupt a processor after satisfying the necessary condition. Interruption is done by loading one pin of the output port first with 0 and later with 1.

The program given below activates the process which is explained above.

```
ACTIVATE
LD HL,GRUP1    HL POINTS TO GRUP1
SET 0,[HL]     BIT 0 OF GRUP1 IS 1.
LD A,[PORTDATA] A HAS THE OUTPUT PORT DATA
RES 2,A        BIT 2 OF A IS SET TO 0
OUT [01],A     OUTPUT A VIA PORT 0
SET 2,A        BIT 2 OF A IS SET TO 1
OUT [01],A     OUTPUT A VIA PORT 0
continue
```

**CYCLIC BUFFER REALIZATION:**

Cyclic buffer operations are explained in section 3.5. The Assembly realization of cyclic buffer operations are given below:

**PRODUCER:**

```
LD HL,BUFFERAVAIL HL POINTS LOCATION BUFFERAVAIL
CALL HOLD CRITICAL REGION HOLD MEMORY
LD A,[HL] NUMBER OF EMPTY BLOCKS IN A
OR A IS IT 0 ?
CALL Z,MAKEPLACE IF 0 EMPTY ONE BUFFER BLOCK
LD HL,PRODUCEPOIJ WHERE IS THE CURRENT PLACE
CALL LOOKFROMTABLE LOOK FOR THE ADDRESS OF THE AVAILABLE BLOCK.
continue
```

**CONSUMER:**

```
LOOKFOR
LD HL,CONSUMERPOIJ HL POINTS CONSUMERPOINTER
LD A,[PRODUCEPOIJ] A HAS PRODUCEPOINTER
CP [HL] COMPARE TWO POINTERS
JR NZ,CONTINUE IF NOT EQUAL CONTINUE
HALT WAIT FOR AN INTERRUPT
JR LOOKFOR REPEAT SEARCHING AGAIN
continue
```
REFERENCES:

1- Gerritsen, F.A.  

2- Ramamoorthy, C.V. and H.F.Li.  

3- Ensor,P.H.  

4- Kuck,D.J.  

5- Thurer,K.J. and L.O.Wald.  

6- Brinch,H.  

7- Brinch,H.  

8- Z-80 ESSENSEY LANGUAGE PROGRAMMING MANUAL.Zilog.(1977)
# Appendix to Chapter 3.

## Explanation of Assembly Mnemonics of Z-80 Processor

<table>
<thead>
<tr>
<th>Alphabetical Assembly Mnemonic</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC HL,ss</td>
<td>Add with Carry Reg. pair ss to HL</td>
</tr>
<tr>
<td>ADC A,ss</td>
<td>Add with Carry operand s to Acc.</td>
</tr>
<tr>
<td>ADD A,n</td>
<td>Add value n to Acc.</td>
</tr>
<tr>
<td>ADD A,r</td>
<td>Add Reg. r to Acc.</td>
</tr>
<tr>
<td>ADD A,(HL)</td>
<td>Add location (HL) to Acc.</td>
</tr>
<tr>
<td>ADD A,(IX+d)</td>
<td>Add location (IX+d) to Acc.</td>
</tr>
<tr>
<td>ADD A,(IY+d)</td>
<td>Add location (IY+d) to Acc.</td>
</tr>
<tr>
<td>ADD HL,ss</td>
<td>Add Reg. pair ss to HL</td>
</tr>
<tr>
<td>ADD IX,pp</td>
<td>Add Reg. pair pp to IX</td>
</tr>
<tr>
<td>ADD IY,rr</td>
<td>Add Reg. pair rr to IY</td>
</tr>
<tr>
<td>AND w</td>
<td>Logical 'AND' of operand s and Acc.</td>
</tr>
<tr>
<td>BIT b,(HL)</td>
<td>Test BIT b of location (HL)</td>
</tr>
<tr>
<td>BIT b,(IX+d)</td>
<td>Test BIT b of location (IX+d)</td>
</tr>
<tr>
<td>BIT b,(IY+d)</td>
<td>Test BIT b of location (IY+d)</td>
</tr>
<tr>
<td>CALL cc,nn</td>
<td>Call subroutine at location nn if condition cc is true</td>
</tr>
<tr>
<td>CALL nn</td>
<td>Unconditional call subroutine at location nn</td>
</tr>
<tr>
<td>CCF</td>
<td>Complement carry flag</td>
</tr>
<tr>
<td>CP a</td>
<td>Compare operand a with Acc.</td>
</tr>
<tr>
<td>CPD</td>
<td>Compare location (HL) and Acc.</td>
</tr>
<tr>
<td>CPDR</td>
<td>Compare location (HL) and Acc.</td>
</tr>
<tr>
<td>CPL</td>
<td>Complement Acc. (1's comp)</td>
</tr>
<tr>
<td>DAA</td>
<td>Decimal adjust Acc.</td>
</tr>
<tr>
<td>DEC m</td>
<td>Decrement operand m</td>
</tr>
<tr>
<td>DEC IX</td>
<td>Decrement IX</td>
</tr>
<tr>
<td>DEC IY</td>
<td>Decrement IY</td>
</tr>
<tr>
<td>DEC ss</td>
<td>Decrement Reg. pair ss</td>
</tr>
<tr>
<td>DI</td>
<td>Disable interrupts</td>
</tr>
<tr>
<td>DJNZ e</td>
<td>Decrement B and Jump relative if B=0</td>
</tr>
<tr>
<td>EI</td>
<td>Enable interrupts</td>
</tr>
<tr>
<td>EX (SP),HL</td>
<td>Exchange the location (SP) and HL</td>
</tr>
</tbody>
</table>
EX (SP), IX
Exchange the location (SP) and IX

EX (SP), IY
Exchange the location (SP) and IY

EX AF, AF'
Exchange the contents of AF and AF'

EX DE, HL
Exchange the contents of DE and HL

EXX
Exchange the contents of BC, DE, HL with contents of BC', DE', HL' respectively

HALT
HALT (wait for interrupt or reset)

IN A, (n)
Load the Acc. with input from device n

IN r, (C)
Load the Reg. r with input from device (C)

INC (HL)
Increment location (HL)

INC IX
Increment IX

INC (IX+d)
Increment location (IX+d)

INC IY
Increment IY

INC (IY+d)
Increment location (IY+d)

INC r
Increment Reg. r

INC ss
Increment Reg. pair ss

IND
Load location (HL) with input from port (C), decrement HL and B

INDR
Load location (HL) with input from port (C), decrement HL and decrement B, repeat until B=0

INI
Load location (HL) with input from port (C), and decrement HL and decrement B

INIR
Load location (HL) with input from port (C), increment HL and decrement B, repeat until B=0

JP (HL)
Unconditional Jump to (HL)

JP (IX)
Unconditional Jump to (IX)

JP (IY)
Unconditional Jump to (IY)

JP cc, nn
Jump to location nn if condition cc is true

JP nn
Unconditional Jump to location nn

JR C, e
Jump relative to PC+e if carry=1

JR e
Unconditional Jump relative to PC+e

JR NC, e
Jump relative to PC+e if carry=0
JR NZ,e  Jump relative to 
         PC+e if non zero (Z=0) 
JR Z,e  Jump relative to 
         PC+e if zero (Z=1) 
LD A,(BC)  Load Acc. with location (BC) 
LD A,(DE)  Load Acc. with location (DE) 
LD A,l  Load Acc. with I 
LD A,(nn)  Load Acc. with location nn 
LD A,R  Load Acc. with Reg. R 
LD (BC),A  Load location (BC) with Acc. 
LD (DE),A  Load location (DE) with Acc. 
LD (HL),n  Load location (HL) with value n 
LD dd,nn  Load Reg. pair dd with value nn 
LD dd,(nn)  Load Reg. pair dd with location (nn) 
LD HL,(nn)  Load HL with location (nn) 
LD (HL),r  Load location (HL) with Reg. r 
LD I,A  Load I with Acc. 
LF IX,nn  Load IX with value nn 
LD IX,(nn)  Load IX with location (nn) 
LD (IX+d),n  Load location (IX+d) with value n 
LD (IX+d),r  Load location (IX+d) with Reg. r 
LD IY,nn  Load IY with value nn 
LD IY,(nn)  Load IY with location (nn) 
LD (IY+d),n  Load location (IY+d) with value n 
LD (IY+d),r  Load location (IY+d) with Reg. r 
LD (nn),A  Load location (nn) with Acc. 
LD (nn),dd  Load location (nn) with Reg. pair dd 
LD (nn),HL  Load location (nn) with HL 
LD (nn),IX  Load location (nn) with IX 
LD (nn),IY  Load location (nn) with IY 
LD R,A  Load R with Acc. 
LD r,(HL)  Load Reg. r with location (HL) 
LD r,(IX+d)  Load Reg. r with location (IX+d) 
LD r,(IY+d)  Load Reg. r with location (IY+d) 
LD r,n  Load Reg. r with value n 
LD r,r'  Load Reg. r with Reg. r' 
LD SP,HL  Load SP with HL 
LD SP,IX  Load SP with IX 
LD SP,IY  Load SP with IY 
LDD  Load location (DE) with location (HL), 
       decrement DE,HL and BC 
LDDR  Load location (DE) with location (HL), 
       decrement DE,HL and BC; 
       repeat until BC=0
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LDI</strong></td>
<td>Load location (DE) with location (HL), increment DE, HL, decrement BC</td>
</tr>
<tr>
<td><strong>LDIR</strong></td>
<td>Load location (DE) with location (HL), increment DE, HL, decrement BC and repeat until BC=0</td>
</tr>
<tr>
<td><strong>NEG</strong></td>
<td>Negate Acc. (2’s complement)</td>
</tr>
<tr>
<td><strong>NOP</strong></td>
<td>No operation</td>
</tr>
<tr>
<td><strong>OR s</strong></td>
<td>Logical ‘OR’ of operand s and Acc.</td>
</tr>
<tr>
<td><strong>OTDR</strong></td>
<td>Load output port (C) with location (HL), decrement HL and B, repeat until B=0</td>
</tr>
<tr>
<td><strong>OTIR</strong></td>
<td>Load output port (C) with location (HL), increment HL, decrement B, repeat until B=0</td>
</tr>
<tr>
<td><strong>OUT (C),r</strong></td>
<td>Load output port (C) with Reg. r</td>
</tr>
<tr>
<td><strong>OUT (n),A</strong></td>
<td>Load output port (n) with Acc.</td>
</tr>
<tr>
<td><strong>OUTD</strong></td>
<td>Load output port (C) with location (HL), decrement HL and B</td>
</tr>
<tr>
<td><strong>OUTI</strong></td>
<td>Load output port (C) with location (HL), increment HL and decrement B</td>
</tr>
<tr>
<td><strong>POP IX</strong></td>
<td>Load IX with top of stack</td>
</tr>
<tr>
<td><strong>POP IY</strong></td>
<td>Load IY with top of stack</td>
</tr>
<tr>
<td><strong>POP qq</strong></td>
<td>Load Reg. pair qq with top of stack</td>
</tr>
<tr>
<td><strong>PUSH IX</strong></td>
<td>Load IX onto stack</td>
</tr>
<tr>
<td><strong>PUSH IY</strong></td>
<td>Load IY onto stack</td>
</tr>
<tr>
<td><strong>PUSH qq</strong></td>
<td>Load Reg. pair qq onto stack</td>
</tr>
<tr>
<td><strong>RES b,m</strong></td>
<td>Reset Bit b of operand m</td>
</tr>
<tr>
<td><strong>RET</strong></td>
<td>Return from subroutine</td>
</tr>
<tr>
<td><strong>RET cc</strong></td>
<td>Return from subroutine if condition cc is true</td>
</tr>
<tr>
<td><strong>RETI</strong></td>
<td>Return from interrupt</td>
</tr>
<tr>
<td><strong>RETN</strong></td>
<td>Return from non maskable interrupt</td>
</tr>
<tr>
<td><strong>RLA</strong></td>
<td>Rotate left Acc. through carry</td>
</tr>
<tr>
<td><strong>RLC (HL)</strong></td>
<td>Rotate location (HL) left circular</td>
</tr>
<tr>
<td><strong>RLC (IX+d)</strong></td>
<td>Rotate location (IX+d) left circular</td>
</tr>
<tr>
<td><strong>RLC (IY+d)</strong></td>
<td>Rotate location (IY+d) left circular</td>
</tr>
<tr>
<td><strong>RLC r</strong></td>
<td>Rotate Reg. r left circular</td>
</tr>
<tr>
<td><strong>RLCA</strong></td>
<td>Rotate left circular Acc.</td>
</tr>
<tr>
<td><strong>RLD</strong></td>
<td>Rotate digit left and right between Acc. and location (HL)</td>
</tr>
<tr>
<td><strong>RR m</strong></td>
<td>Rotate right through carry operand m</td>
</tr>
<tr>
<td><strong>RRA</strong></td>
<td>Rotate right Acc. through carry</td>
</tr>
<tr>
<td><strong>RRC m</strong></td>
<td>Rotate operand m right circular</td>
</tr>
<tr>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RRCA</td>
<td>Rotate right circular Acc.</td>
</tr>
<tr>
<td>RRD</td>
<td>Rotate digit right and left between Acc. and location (HL)</td>
</tr>
<tr>
<td>RST p</td>
<td>Restart to location p</td>
</tr>
<tr>
<td>SBC A,s</td>
<td>Subtract operand s from Acc. with carry</td>
</tr>
<tr>
<td>SBC HL,ss</td>
<td>Subtract Reg. pair s from HL with carry</td>
</tr>
<tr>
<td>SCF</td>
<td>Set carry flag (C=1)</td>
</tr>
<tr>
<td>SET b,(HL)</td>
<td>Set Bit b of location (HL)</td>
</tr>
<tr>
<td>SET b,(IX+d)</td>
<td>Set Bit b of location (IX+d)</td>
</tr>
<tr>
<td>SET b,(IY+d)</td>
<td>Set Bit b of location (IY+d)</td>
</tr>
<tr>
<td>SET b,r</td>
<td>Set Bit b of Reg. r</td>
</tr>
<tr>
<td>SLA m</td>
<td>Shift operand m left arithmetic</td>
</tr>
<tr>
<td>SRA m</td>
<td>Shift operand m right arithmetic</td>
</tr>
<tr>
<td>SRL m</td>
<td>Shift operand m right logical</td>
</tr>
<tr>
<td>SUB s</td>
<td>Subtract operand s from Acc.</td>
</tr>
<tr>
<td>XOR s</td>
<td>Exclusive 'OR' operand s and Acc.</td>
</tr>
</tbody>
</table>
4. The Reading Process

4.1. Principles of Optical Character Reading;

The extensive research on optical character reading (O.C.R.) has started since some decades of years. Although there are many fields where the O.C.R. can be applied, because of some basic technological limitations, the obtained results were not satisfactory. Since 1954 various O.C.R. units have been introduced. Prices of these O.C.R. units were much too high to bring them to the public use [1].

With the advent of large scale integration revolution, for their designers, O.C.R. has received a very different appearance. Where slow and expensive processing was an obstacle, large scale integration brought cheap easy solutions. The O.C.R. devices suffered mainly because of two reasons.

A) Expensive, bulky, slow processors.
B) Expensive, bulky sensors with a limited resolution.

Before going deep into the analysis of a character reader, it is found useful to summarize the developed character scanners. There are mainly 4 types of scanners available.

A) Mechanical scanners
B) Flyingspot scanner
C) Photocells
D) Vidicon

At the present time, none of the scanners are promising as much as integrated photo-diodes do. For the short-term future, introducing of charge coupled devices has brought such possibilities that even the present television cameras can be replaced by compact, integrated photo-sensitive elements [2]. There is a considerable improvement in available components during the start of THERMB project (end of 1979) and now. In the present model of THERMB, a photo diode array with 64 bit resolution is used. Now, with even lower prices, solid state scanners with 256, 1024, 1728 elements resolution are available [3]. Although higher number of bits will require more memory and faster information handling, this will not be a big problem for the future VLSI parallel computing systems.
4.2. The Information Conversion;

4.2. A. General;

In order to handle an information in a digital computer, first all input variables must be converted into a digital form. A typical converter for an O.C.R. may have this structure.

**Illumination**

```
\\\\
\\\\ B -------> light Lenses -----> O : Analog signal: Digital
```

Figure 14. Conversion to the digital information.

The structure of the sensor is mainly determined by the photo-diode scanner. There are two possible ways for the organization of diodes.

A) Vertical array

B) Matrix form.

Vertical diode arrays are currently available in large quantities and with high resolutions. Therefore in present applications they may be preferable to matrix ones.

```
0 00000000000
0 00000000000
0 00000000000
0 00000000000
0 00000000000
0 00000000000
0 00000000000
0 Matrix organized diodes
0
```

Figure 15. Diode array organization.

Because of different reading principles, both diode arrangements will be handled separately.
4.3

4.2.8. Reading by Vertical Diode Arrays;

Character inputting can be realized simply by shifting the diode array over the character.

```
0
0
0*
***0***
*** 0 **
*******0******
** 0 **
** 0 **
0
```

Figure 16. Reading by a vertical diode array.

This simple looking method brings in practice some sampling difficulties. Photo-diodes must be sampled to detect white and black information. The sampling can be realized in two ways.

A) Continuous sampling.
B) Sampling only when the sensor is moved.

Continuous sampling:

It is very easy to realize but will result into useless information accumulation. In order to have speed independent reading, scanning frequency must be relatively high. As we do not have any knowledge of movement, simply the reader can leave the sensor for some time on a paper, we must find a method to separate useful and useless information from each other. In order to prevent excessive data accumulations a differential mapping technique is proposed.

```
* * * * * *
Mapping function
* * * * * *
Y =-------------------------------> *
Δ > Threshold
* * * * * *
Δ = Difference between two adjacent columns
```

Figure 17. Mapping from the paper to the semiconductor memory.

Differential mapping technique is detecting the rate of change of the input. If the change is more than a threshold, then the new input will be accepted. Algorithm 4.1 detects the total number of differences between two adjacent columns. If the difference is higher than the threshold, the column is accepted.
Algorithm 4.1 has some disadvantages. In order to accept the new column, all column differences are counted. However, for some characters local changes may give the main information to differentiate them from others. Therefore two kinds of decision can be made, local and global decision. To apply local tests each column must be divided into equal segment pieces. The number of differences must be tested in each segment piece. If none of the test succeeds then the global test must be made. The complete character reading based on local and global tests are given in algorithm 4.2.

The complete character inputting in algorithm 4.2.

```
ARRAY CHAR[I:DIODENUM, I:COLUMNNUM];% CHAR CONTAINS THE INPUTTED CHARACTER
INTEGER DIFF,K; % K IS THE COLUMN COUNTER

DIFF:=0;
FOR I:=1 STEP 1 UNTIL DIODENUM DO
  IF CHAR[I,K-1] NOT EQL CHAR[I,K] THEN
    DIFF:= DIFF+1;
  IF DIFF > THRESHOLD THEN K:=K+1;
  GOTO REPEAT;

Algorithm 4.1 has some disadvantages. In order to accept the new column, all column differences are counted. However, for some characters local changes may give the main information to differentiate them from others. Therefore two kinds of decision can be made, local and global decision. To apply local tests each column must be divided into equal segment pieces. The number of differences must be tested in each segment piece. If none of the test succeeds then the global test must be made. The complete character reading based on local and global tests are given in algorithm 4.2.

The complete character inputting in algorithm 4.2.

ARRAY CHAR[I:DIODENUM, I:COLUMNNUM];
INTEGER I,J,K,L,TOTALDIFF,LOCALDIFF;
BOOLEAN ACCEPT;
CONSTANT SEGMENTLENGTH, LOCALTHRESHOLD, GLOBALTHRESHOLD, DIODENUM, COLUMNNUM;

K:=1;
FOR I:=1 STEP 1 UNTIL DIODENUM DO
  INPUT(CHAR[I,K]) ; % INPUT FIRST COLUMN
  K:=K+1;

REPEAT: FOR I:=1 STEP 1 UNTIL DIODENUM DO
  INPUT(CHAR[I,K]); % INPUT NEXT COLUMN

  IF INPUTWHITE THEN GO TO END PROGRAM;
  L:=L:=TOTALDIFF:=LOCALDIFF:=0;
  ACCEPT:=FALSE;
  WHILE NOT ACCEPT AND NOT(I=DIODENUM) DO;
    BEGIN
      LOCALDIFF:=0
      FOR J:=L STEP 1 UNTIL L+SEGMENTLENGTH DO;
        IF LOCALDIFF > LOCALTHRESHOLD THEN ACCEPT:=TRUE;
        L:=L+SEGMENTLENGTH; I:=I+1;
        TOTALDIFF:=TOTALDIFF+LOCALDIFF;
    END WHILE;
```
IF ACCEPT OR TOTALDIFF > GLOBALTHRESHOLD THEN K:=K+1;
GOTO REPEAT;
END PROGRAM:

In THERMB continuous scanning technique is used.
The advantages of this method are:
1- Very economic, easy to apply. Completely electronic.
2 - Filters noise on curves, inclined and declined lines.
3 - Number of columns gives information about the character (W will have much more columns than 1).

Disadvantages:
1 - Distorts vertical and straight lines.
2 - No space information is supplied.

As columns are only accepted with a reasonable change, characters like E, F, I, L, H, J are largely deformed. In figure 19, characters read by this method is shown. It can be seen from the figure that while vertical and horizontal lines are deformed, curves and inclined, declined lines are read successfully. The lack of space information gives problems for talking machines. Wherever the combination of letters are necessary, another method must be used.

Equi-Distance Sampling:

In this method character is sampled with a certain move of the sensor.

Figure 18. Equi - Distance Sampling.

This method looks much more attractive than the differential mapping. The character is transformed to the memory without deforming its shape (fine samples). The main difficulty here is to detect the movement of the sensor. Two methods are proposed to detect the movement of the sensor:
Figure 19. The effect of differential mapping on some characters.
1)-By rotating wheels placed on the paper.
2)-By magnetic strips and a magnetic head.

In the first method, wheels are placed under the sensor. When the sensor is moved, these wheels will rotate. There is an optical system which detects the movement of wheels. At every certain rotation, an interrupt pulse can be sent to the computer to accept the data from the sensor.

In the second method, a magnetic head can be placed in the sensor and a marked magnetic strip on the page. When the sensor is shifted on the paper, the induced e.m.f. across the magnetic head can be used as an interrupt pulse for the computer to accept data.

Both of these methods are more costly than the continuous scanning. If a very fine sampling is required, (e.g. 0.1 mm) for the first method, an expensive mechanical system must be used. The reliability of mechanical units are limited.

In the second method, in order to prevent drop-outs, the gap between the strip and head must be very small. It may be also undesirable to have magnetic strips on the paper.

4.2.C. Matrix Organized Diode Arrays;

Matrix organized arrays are overcoming the difficulties arises from the column inputting. No detection of movement is necessary because diodes can be continuously scanned. Faster inputting can be realized than by means of vertical arrays. The whole character can be inputted at a time. At present, matrix organized arrays have two disadvantages:

A)- Limited resolution
B)- Input contamination when the sensor covers two characters at a time.

Limited resolution is only the problem of today. One can be sure of having more resolution per chip in the near future. Also isolation of characters can be realized by software. With the development of semiconductor technology, matrix scanners may take the place of array scanners.

4.3. Picture Deformation;

The information flow in a typical O.C.R. system is shown in figure 20. The image of the picture in the memory must be very close to the original one. As it is seen in figure 20, information flows through four different blocks. The effect of these blocks are explained below.

A

B

C

D

[Illuminated] [===>] Lenses [====>] Light [Diodes] [An. Signal] [Digital]

[Light] [===>] [Photo] [A/D] [===>] A/D

[Sampling] [control]

Figure 20. The information flow.
A) Illumination:

Cares must be taken for these points.

1) A sufficient illumination. This seems to be very important for the picture quality. When there is a sufficient light a) Small lens surfaces can be chosen to prevent lens errors. b) Diodes can be saturated with white input to prevent effects of different diode outputs.

2) Direction of light. When the light is not carefully directed the sharpness of the picture decreases.

3) Homogeneous illumination. When there is no sufficient light for saturation of diodes, a very homogeneous distribution of light over the picture is necessary.

Illumination power is limited because of the heat arises from the light sources.

B) Lenses:

Important points are:

1) Good quality lenses are necessary to form a good image. To lower lens faults, a diaphragm can be placed to decrease the effective surface. This will also give more depthness to the picture.

2) For an easy focusing a fine mechanical adjustment unit is necessary. Because of the short focus length, adjusting the sharpness of the picture is very critical.

C) Photo-Diodes:

Photodiodes are converting photo-information to electric signals. Important points with photodiodes are:

1) Sensitivity. High sensitivities will let us use lower light intensities.

2) High resolution. It is important for the picture detail.

3) Balanced outputs. Diodes must have the same sensitivity. Saturation of diodes will prevent the effects of unbalanced outputs.

4) Sampling method. Sampling method defines the mapping function. The effects of different sampling techniques are discussed in section 4.2.8.

D) Analog/Digital Conversion:

When no gray information is necessary, one bit A/D convertor is sufficient. When the white level of different papers are not constant the threshold level for a black-white decision can be critical. If the image is sharp and diodes are saturated with any white level, quantization of the signal will not be a serious problem.

4.4. The Hand Carried Sensor;

The sensor which is used in THERMB is based on some previous practical works carried in group EEB(5). The sensor is equipped with:
1) Lenses.
2) Scanner I.C.
3) A light source.
4) Amplifier.
5) 1 bit A/D converter and latches.

The lenses are placed in an aluminium block and screwed to the P.C. board where the scanner I.C. lays. On the I.C. there is a window and photo-diodes are placed under this window. The image of the character must be formed on the window. The dimensions of lenses and scanner I.C. are given in figure 21. With this arrangement a character height of max 6 mm can be read. More detailed information can be found in the reference [5].

Figure 21. Position of lenses and the scanner.
Although the principle is the same, some improvements are brought to the new version of the sensor to have better performance. These modifications are given below.

1) A diaphragm is added to the lens system.
2) The light source is brought much more closer to the character.
3) A more powerful light source is used.
4) Better printed circuit board is used to decrease the clock noise.

Points 1 to 3 are for decreasing the lens errors, saturating the amplifier for any white input, giving more deepness for easy focusing.

When THERMB is put into its practical form, a more handy focusing mechanism must be used. Also the shape of the sensor must be optimized to be able to carry it more easily.

4.5. The Input processor;

4.5 A. Descriptions;

In order to obtain fast reading a special processor is assigned for inputting the character. Successful inputting will not only improve the recognition process, but also decreases the duties of other processors. The input processor is responsible of the following operations:

A) To Initialize the system.
B) To control the sensor and input data from it.
C) To check correctness of reading (indicates fault or correct reading).
D) To do all necessary pre-processing for preserving the parallelism of the system.
E) To initialize central processors for recognition, when a character is completely read in.

The most important function of the Input processor is to read the character from the page and place it in the buffer.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W =========&gt;</td>
<td>Input</td>
</tr>
<tr>
<td></td>
<td>Processor</td>
</tr>
</tbody>
</table>

The character to be read in.

Buffer

Figure 22. Character inputting.

To increase the parallelism of the system, the input processor does not only read the character in, but also supplies all the necessary information in the buffer. Buffer is divided into buffer blocks. The structure of the buffer block is shown in Figure 23.
Figure 23. Buffer block organization.

Descriptions of these variables are given below:

CHARACTER: Contains the inputted character. Size = 140 bytes. For 32 bits of column resolution maximum 35 columns are available.

<table>
<thead>
<tr>
<th>Row number</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>***</td>
</tr>
<tr>
<td>2</td>
<td>*****</td>
</tr>
<tr>
<td>3</td>
<td>**</td>
</tr>
<tr>
<td>4</td>
<td>*</td>
</tr>
<tr>
<td>5</td>
<td>***</td>
</tr>
<tr>
<td>6</td>
<td>***</td>
</tr>
<tr>
<td>7</td>
<td>***</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>123456789</td>
</tr>
</tbody>
</table>

Column number.

Figure 24. Typical appearance of the inputted character in the buffer block.

TOP WHITE: White segments which are between the top of the buffer block and the black segments.

BOTTOM WHITE: White segments which are between the bottom of the buffer block and the black segments.
To explain the meanings of variables, TOPCOL, BOTCOL, TOPCADRES, BOTCADRES and HEIGHT, figure 25 is drawn.

**Figure 25. Definitions of some variables.**

**TOPCOL:** The column where topwhite min occurs. Size=1 byte.

**BOTCOL:** The column where bottomwhite min occurs. Size=1 byte.

**TOPCADRES:** The address which points to topwhite min. Size=2 bytes.

**BOTCADRES:** The address which points to bottomwhite min. Size=2 bytes.

**HEIGHT:** Contains the height of the character.

The meanings of WHITE SEGMENT, BLACK SEGMENT, COLUMN DESCRIPTION, LAST WHITE, LAST BLACK will be explained by the help of figure 26.

**Figure 26. The character Z in the memory.**
The appearance of the character Z in the memory is shown in figure 26. It is much more convenient to represent the picture as a one dimensional string. The picture description language is a string representation of the two dimensional picture.

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in the (===========&gt;)(...... String .................)</td>
</tr>
<tr>
<td>buffer</td>
</tr>
<tr>
<td>--------</td>
</tr>
</tbody>
</table>
```

Figure 27. The picture description language.

The description of the picture is done column by column. To be able to describe the picture completely, each column is divided into black and white segments. For example, the first column in figure 26 can be described as, a column with one black segment surrounded by two white segments. The lengths of white segments from bottom to top are, 9 and 1 respectively. The length of the black segment is 4. To construct this language three variables are defined.

WHITE SEGMENTS: The list of lengths of white segments from bottom to top.
BLACK SEGMENTS: The list of lengths of black segments from bottom to top.
COLUMN DESCRIPTION: The number of black segments in the column.

The description of the picture given in figure 26 is:

WHITE SEGMENTS = 9, 1, 1, 6, 1, 1, 3, 1, 1, 3, 1, 1, 4, 1, 2, 5, 2, 1, 4, 1, 1, 1, 1, 2,
BLACK SEGMENTS = 4, 3, 3, 7, 2, 2, 2, 6, 1, 4, 3, 1, 1, 1, 1,
COLUMN DESCRIPTION = 01, 02, 02, 03, 02, 02, 04, 01,

The description of other variables are also given below:
LAST WHITE: Address of the last white segment data.
LAST BLACK: Address of the last black segment data.
COLUMNNUMBER: Number of columns.

4.5.13. The software;

The state diagram of processing is given in figure 28. The input processing consists of 6 states. A successful reading will carry the processor from state 1 to state 5. If the reading proceeds correctly, the processing will loop through state 3-4-5. The structural division of a task simplifies the tracing of workflow.
Here state descriptions are:

State 1: Initialization.
State 2: Wait for white.
State 3: Wait for black.
State 4: Put first column in.
State 5: If the new column is different than previous one input it.
State ERROR: Give a sound according to the input data.

Transitions with interrupts:
White = Boolean, True if all input is white. (Mask.Intr.)
Black = Boolean, NOT White. (Mask.Intr.)
Error = Character is not placed properly. (Mask.Intr.)
Clear error = Entered by user. (Nonmask.Intr.)

For the input processing, the whole task can be divided into two main parts, "Interrupt program" and "Main program". The Interrupt program is responsible of inputting data. When the Interrupt program terminates, the inputted column will be placed into the memory with its description language. The Interrupt program consists of 10 states.

The Input processing

---

Figure 28. The state table of the Input processing.

Figure 29. The program division.
THE MAIN PROGRAM:

-------------------

The detailed explanation of states of the Main program is given below:

State 1:
-------

This is an initialization state. When the RESET button is pressed or when the power is first applied, processor will enter into state 1. During this RESET state input processor will:

A) Copy the buffer block addresses from local ROM to the common RAM.
B) Clear some memory locations and define system constants used by all processors.
C) Command the sensor to start with character scanning.

The H.L.L.D (High level language description) of state 1 is:

Algorithm 4.3.

HOLD
INITIALIZE;
RELEASE;
STARTSCANNING;

State 2:
-------

Here Processor waits for a WHITE input. That means, first the sensor must be placed on white paper. If the input is BLACK, and if an error-state exists, the sound circuit will be commanded to generate a 250HZ signal. H.L.L.D of State 2 is given below:

Algorithm 4.4.

WAIT(INPUT);
WHILE INPUTBLACK DO
BEGIN
IF ERRORSTATE THEN GENERATE(250HZ);
WAIT(INPUT);
END;

State 3:
-------

Here processor is waiting for a BLACK input to start character reading process. It also clears some memory locations which are used during data inputting from the sensor. H.L.L.D. is
Algorithm 4.5

WHILE NOTBLACK DO
BEGIN
CLEAR(MEMORYPLACES);
WAIT(INPUT);
END;

State 4:
--------
In this state following operations will be carried out:

a) Compute the length of white and black segments.

b) Search for the current buffer block. If there is no empty buffer block interrupt Central processors to empty one.

c) Compute the necessary addresses used in this buffer block.

d) Put the first column in.

e) Store the description of the column.

During interrupt routine, after the column inputting, description of this column is also made. However before knowing buffer block addresses, it is impossible to store this data to its own place. Therefore until the buffer block is known, the column description strings are stored in a temporary storage. Computing the length of white and black segments will let us to restore them again.

When the conditions are ready for column inputting, first an empty buffer block must be found. After finding an empty buffer block, necessary addresses used in this block must be computed.

When all the addresses are known, inputted column and column description string can be stored. After all, processor has to wait for the second interrupt to continue with reading. H.L.L.D. is:

Algorithm 4.6.

COMPUTE(LENGTHS);
SEARCH FOR A BUFFER BLOCK;
IF NO EMPTY BUFFER BLOCK THEN BEGIN
MAKEPLACE;
WAIT(EMPTY BUFFER BLOCK);
END;
COMPUTE(BUFFER BLOCK ADDRESSES);
PUT(PRODUCER POINTER, BUFFER BLOCK, COLUMN);
PUT(DESCRIPTOR POINTERS, BUFFER BLOCK, DESCRIPTIONS);
WAIT(INPUT);

State 5:
-------
In state 5, columns of the character will be inputted after each sensor interrupt. Operations in State 5 can be summarized below:
a) Input the column from the sensor.
b) Compare the column with the previous one. Accept the new column if there is sufficient difference between two adjacent columns.
c) If the inputted column is white this means end of the character, the last processing will be made. Buffer pointers will be updated.

H.L.L.D of state S is given below.

Algorithm 4.7.

```
WAIT (INPUT);
WHERE INPUTBLACK DO
BEGIN
COMPARE (TWO COLUMNS);
If DIFFERENCE > THRESHOLD THEN ACCEPT (COLUMN)
ELSE REFUSE (COLUMN);
END;
If COLUMNNUMBER < MINIMUM COLUMN NUMBER
THEN REFUSE (CHARACTER)
ELSE BEGIN
LASTPROCESS;
UPDATEBUFFER POINTERS;
END ELSE;
```

In section 4.2.B. the effect of local and global tests are explained. In state S, local and global differences are tested separately. The algorithm 4.2. is also valid here.

During the last process, TOPCOL, BOTCOL, TOPCADRES, BOTCADRES and HEIGHT are computed. The computation of these variables are given in algorithms 4.8 and 4.9.

Computation of TOPCOL and TOPCADRES in algorithm 4.8.

```
Z == COMMENT
A =< B ==> A LESS OR EQUAL TO B
BEGIN
ARRAY COLUMNDESCRIPTION[1:COLUMNNUMBER];
ARRAY WHITESEGMENT[1:WHITESEGMENTLENGTH] INTEGER TOPCADRES, TOPCOL, I, J;
I:= TOPCOL := J := 1;
J:= J + COLUMNDESCRIPTION[I]; % DESCRIPTION OF FIRST COLUMN
TOPCADRES := WHITESEGMENT[I];
J:= J + 1; % NEXT COLUMN
FOR I:=2 STEP 1 UNTIL COLUMNNUMBER DO XSEARCH FOR OTHER COLUMNS
BEGIN
J:= J + COLUMNDESCRIPTION[I]; % I'th COLUMN.
IF TOPCADRES =< WHITESEGMENT[I] THEN
BEGIN
TOPCOL:= I; TOPCADRES:=WHITESEGMENT[I];
END;
J:= J + 1; % NEXT COLUMN
END;
END.
```
Computation of BOTCOL and BOTCADRES in algorithm 4.9.

BEGIN
ARRAY COLUMNDESCRIPTION[I:COLUNNUMBER];
ARRAY WHITESEGMENT[I:WHITESEGMENTLENGTH]
INTEGER BOTCADRES,BOTCOL,I,J ;

BOTCOL := 1 := J := 1 ;  % FIRST COLUMN
BOTCADRES:= WHITESEGMENT[J] ;
J:= J + COLUMNDESCRIPTION[I] ;
J:=J + 1 ;  % SECOND COLUMN
FOR I:= 2 STEP 1 COLUMNNUMBER DO
BEGIN
IF WHITESEGMENT[J] =< BOTCADRES THEN
BEGIN
BOTCADRES:=WHITESEGMENT[J] ;
BOTCOL := I ;
END
J:= J + COLUMNDESCRIPTION[I];
J := J + 1 ;  % NEXT COLUMN
END ;
END ;

Computation of height is simply ,

HEIGHT := DIODENUMBER - (TOPCADRES + BOTCADRES) ;

INTERRUPT ROUTINE :
-----------------------

The Interrupt routine plays a very important role
in input processing. The Interrupt routine is responsible for the following
operations.

a)-To input data from the sensor.
b)-To start character scanning.
c)-To detect if the sensor is placed properly. If not command the
sound circuits for indication.
d)-Generate the description of the inputted column. If it is the
first column then store the language into a temporary storage.

When the sensor is ready with the character scan-
ning, it sends an interrupt signal to the input processor. When the input
processor is ready for character inputting, it responds to this interrupt
request and executes the interrupt routine. Execution of the interrupt ro-
utine must be ready before the next interrupt comes. At the same time it is
also desirable that all the serial operations are combined with inputting
which is naturally sequential. Therefore the serial operation, "language
generation" is combined with the inputting routine. Because of time con-
straints, a special program is developed to realize this operation. The des-
crption of the interrupt routine is given below.
Algorithm 4.10. The Interrupt routine.

PROCEDURE INTERRUPT;
INPUT(SENSOR);
START SCANNING;
COMMENT LOOK IF AN ERROR CONDITION EXISTS;
IF (TOPDIODE IS BLACK) AND (BOTTOMDIODE IS BLACK) THEN
BEGIN
GENERATE(2KHZ);
GOTO WAITFORNEWINPUT;
END
ELSE IF TOPDIODE IS BLACK THEN
BEGIN
GENERATE(1 KHZ);
GOTO WAITFORNEWINPUT;
END
ELSE IF BOTTOMDIODE IS BLACK THEN
BEGIN
GENERATE(500 Hz);
GOTO WAITFORNEWINPUT;
END;
IF STATE IS AN ERROR STATE AND NOT WHITE THEN
BEGIN
GENERATE(250 Hz);
GOTO BEGIN OF STATE2 I LOOK AT ALG. 4.4
END;
COMMENT IF THERE IS NO ERROR STATE START FROM HERE;
IF IT IS THE FIRST COLUMN THEN
BEGIN
REPLACE WHITESegment WITH TEMPORARYWHITESegment;
REPLACE BLACKSegment WITH TEMPORARYBLACKSegment;
REPLACE COLUMNDescriptor WITH TEMPORARYCOLUMNDescriptor;
END;
GENERATE PICTURE DESCRIPTION LANGUAGE;
RETURN FROM THE INTERRUPT ROUTINE;
END INTERRUPT;

In algorithm 4.10 the interrupt routine is given. Here the most time consuming part is the language generation. Without having a special arrangement, it is not possible to complete the interrupt routine before the next interrupt comes. Therefore a special program is developed for the language generation. This part of the program is divided into 10 states. The processing starts with checking each bit from bottom to top. With each change of bit processor enters into a different state which describes the column up to that movement. When the top of the character comes, at which state the processing terminates, that state is the description of that column. In figure 30 the state diagram of the language generation is shown. It can be seen from the figure that, although more memory space is necessary, this way of language generating proceeds very fast. Whatever the structure of the column is, execution time is about constant. Because test, branch, increment operations are very fast, the total execution time is relatively short.
Figure 30 The flow diagram of language generation

Explanation of symbols:

O : Bit test. If it is a WHITE state and if the bit is 1 then branch to the next state. Or if the state is BLACK and if the bit is 0 then branch to the next state.

---: Transition with 0 bit. Increments the white segment counter

---: Transition with 1 bit. Increments the black segment counter

---: Transition with 0 or 1 bit. Increments black and white counter.

Whitesegment : Whsg
Blacksegment : Blsg
Column description : Cdes.
State descriptions:

W1 = White 1. All column is white.
B1 = Black 1, Inputted column: Wsng-Blsng. Cdes: 01.
Error: More than 4 black segments. Cdes: 00.

The algorithm 4.11 describes the language generation process.

Language generation in algorithm 4.11.

PROCEDURE GENERATE PICTURE DESCRIPTION LANGUAGE;

INTEGER WHCOUNTER, BLCOUNTER, I, L;

WHCOUNTER := J := L := 1;
BRANCH := FALSE;
I := 2; X:1 DIODE COUNTER. FIRST DIODE IS WHITE, NO ERROR.

STATE WHITE 1

WHILE NOT BRANCH AND NOT(I=DIOGENUMBER-1) DO
BEGIN
IF CHAR[I, K] = 1 THEN BEGIN
BEGIN
WHITESEGMENT[J] := WHCOUNTER;
J := J + 1;
BLSEGMENT[L] := BLCOUNTER;
L := L + 1;
END;
END WHILE;
IF NOT BRANCH THEN RETURN FROM INTERRUPT; % ALL WHITE RETURN
BRANCH := FALSE;
WHCOUNTER := 0;

STATE BLACK 1

WHILE NOT BRANCH AND NOT(I=DIOGENUMBER-1) DO
BEGIN
IF CHAR[I, K] = 0 THEN BEGIN
BEGIN
BRANCH := TRUE;
WHCOUNTER := 1;
BLSEGMENT[L] := BLCOUNTER;
L := L + 1;
END;
END WHILE;
IF NOT BRANCH THEN BLCOUNTER := BLACKCOUNTER + 1;
I := I + 1;
END WHILE;
IF NOT BRANCH THEN BEGIN
    COLUMN DESCRIPTION[KJ] := 01;
    WHITESEGMENT[J] := 1; % NO ERROR - LAST DIODE-
    J := J + 1; % MUST BE WHITE.
    RETURN FROM INTERRUPT;
END;

BRANCH := FALSE;
BLACKCOUNTER := 0;

STATE WHITE 2 XXXXXXXXXXXXXXXXXXXXXXXXX
WHILE NOT BRANCH AND NOT(I=DIODENUMBER-1) DO
BEGIN
    IF CHAR[I,K] = 1 THEN
        BEGIN
            BRANCH := TRUE;
            WHITESEGMENT[J] := WHCOUNTER;
            J := J + 1;
            BLSEGMENT[L] := BLCOUNTER;
            L := L + 1;
        END;
        IF NOT BRANCH THEN WHCOUNTER := WHCOUNTER + 1;
        I := I + 1;
    END WHILE;
    IF NOT BRANCH THEN BEGIN
        WHITESEGMENT[J] := WHCOUNTER + 1 % NO ERROR -
        J := J + 1; % TOP IS WHITE.
        COLUMN DESCRIPTION[KJ] := 01;
        RETURN FROM INTERRUPT;
    END;

BRANCH := FALSE;
BLACKCOUNTER := 0;

STATE BLACK 2 XXXXXXXXXXXXXXXXXXXXXXXXX
WHILE NOT BRANCH AND NOT(I=DIODENUMBER-1) DO
BEGIN
    IF CHAR[I,K] = 0 THEN
        BEGIN
            BRANCH := TRUE;
            WHCOUNTER := 1;
            BLSEGMENT[L] := BLCOUNTER;
            L := L + 1;
        END;
        IF NOT BRANCH THEN BLACKCOUNTER := BLACKCOUNTER + 1;
        I := I + 1;
    END WHILE;
    IF NOT BRANCH THEN BEGIN
        COLUMN DESCRIPTION[KJ] := 02;
        WHITESEGMENT[J] := 1; % NO ERROR - LAST DIODE-
        J := J + 1; % MUST BE WHITE.
        RETURN FROM INTERRUPT;
    END;

BRANCH := FALSE;
BLACKCOUNTER := 0;
STATE WHITE 3

WHILE NOT BRANCH AND NOT(I=DIODENUMBER-1) DO
BEGIN
IF CHAR[I,K] = 1 THEN
BEGIN
BRANCH := TRUE;
WHITESEGMENT[J] := WHCOUNTER;
J := J + 1;
BLCOUNTER := 1;
END;
IF NOT BRANCH THEN WHCOUNTER := WHCOUNTER + 1
I := I + 1;
END WHILE;
BEGIN IF NOT BRANCH THEN
WHITESEGMENT[J] := WHCOUNTER + 1 % NO ERROR -
J := J + 1;
COLUMNDESCRIPTION[K] := 02;
RETURN FROM INTERRUPT;
END;
BRANCH := FALSE;
WHCOUNTER := 0;

STATE BLACK 3

WHILE NOT BRANCH AND NOT(I=DIODENUMBER-1) DO
BEGIN
IF CHAR[I,K] = 0 THEN
BEGIN
BRANCH := TRUE;
WHCOUNTER := 1;
BLSEGMENT[L] := BLCOUNTER;
L := L + 1;
END;
IF NOT BRANCH THEN BLCOUNTER := BLACKCOUNTER + 1
I := I + 1;
END WHILE;
BEGIN IF NOT BRANCH THEN
COLUMNDESCRIPTION[K] := 03;
WHITESEGMENT[J] := 1; % NO ERROR.LAST DIODE-
J := J + 1; % MUST BE WHITE.
RETURN FROM INTERRUPT;
END;
BRANCH := FALSE;
BLCOUNTER := 0;

STATE WHITE 4

WHILE NOT BRANCH AND NOT(I=DIODENUMBER-1) DO
BEGIN
IF CHAR[I,K] = 1 THEN
BEGIN
BRANCH := TRUE;
WHITESEGMENT[J] := WHCOUNTER;
J := J + 1;
BLCOUNTER := 1;
END;
IF NOT BRANCH THEN WHCOUNTER := WHCOUNTER + 1
I := I + 1;
END WHILE;
IF NOT BRANCH THEN BEGIN
    WHITESEGMENT[J] := WHCOUNTER + 1 ;% NO ERROR -
    J := J + 1 ;% TOP IS WHITE.
    COLUHMDESCRIPTION[K] := 04 ;
    RETURN FROM INTERRUPT ;
END ;

BRANCH := FALSE ;
WHCOUNTER := 0 ;

STATE BLACK 4
WHILE NOT BRANCH AND NOT(I=DIODENUMBER-1) DO
BEGIN
    IF CHAR[I,K] = 0 THEN
    BEGIN
        BRANCH := TRUE ;
        WHCOUNTER := 1 ;
        BLSEGMENT[I] := BLCOUNTER ;
        L := L + 1 ;
    END ;
    IF NOT BRANCH THEN BLCOUNTER := BLACKCOUNTER + 1 ;
    I := I + 1 ;
END WHILE ;

IF NOT BRANCH THEN BEGIN
    COLUMN DESCRIPTION[K] := 04 ;
    WHITESEGMENT[J] := 1 ;% NO ERROR.LAST DIODE -
    J := J + 1 ;% MUST BE WHITE .
    RETURN FROM INTERRUPT ;
END ;

BRANCH := FALSE ;
BLCOUNTER := 0 ;

STATE WHITE 5
WHILE NOT BRANCH AND NOT(I=DIODENUMBER-1) DO
BEGIN
    IF CHAR[I,K] = 1 THEN
    BEGIN
        BRANCH := TRUE ;
        WHITESEGMENT[J] := WHCOUNTER ;
        J := J + 1 ;
        BLCOUNTER := 1 ;
    END ;
    IF NOT BRANCH THEN WHCOUNTER := WHCOUNTER + 1 ;
    I := I + 1 ;
END WHILE ;

IF NOT BRANCH THEN BEGIN
    WHITESEGMENT[J] := WHCOUNTER + 1 ;% NO ERROR -
    J := J + 1 ;% TOP IS WHITE.
    COLUHMDESCRIPTION[K] := 04 ;
    RETURN FROM INTERRUPT ;
END ;

WHCOUNTER := 0 ;
STATE ERROR

WHILE NOT(I=DIODENUMBER-1) DO
  BEGIN
    IF CHAR[I,KJ] = 0 THEN
      WHCOUNTER := WHCOUNTER + 1
    ELSE
      BLCOUNTER := BLCOUNTER + 1;
    I := I + 1;
  END WHILE;

  COLUMN DESCRIPTION[KJ] := 00;
  WHITESEGMENT[J] := 1 + WHCOUNTER;
  J := J + 1;
  BLSEGMENT := BLCOUNTER;
  L := L + 1;
  RETURN FROM INTERRUPT;

END;

REFERENCES

3. RETICON CHARGE COUPLED PHOTODIODE ARRAYS DATA SHEETS. No CCPD-256, CCPD-1024, CCPD-1728. Sunnyvale.
APPENDIX TO CHAPTER 4

THE STRUCTURE OF THE SAMPLED CHARACTERS IN THE MEMORY

In this appendix, the memory view of the three character set is given.

TIMES EUROPA

Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm Nn Oo Pp Qq Rr Ss
Tt Uu Vv Ww Xx Yy Zz

SPECTRUM

Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm Nn Oo Pp Qq Rr
Ss Tt Uu Vv Ww Xx Yy Zz

ORATOR

An Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm Nn Oo Pp Qq Rr Ss Tt Uu Vv Ww Xx Yy Zz

CAPITALS OF

TIMES EUROPA

SPECTRUM

ORATOR (fast reading)
5.1. Principles of Pattern Recognition;

Pattern recognition is a pattern classification process, where for each input called the pattern, a pattern class is assigned. The block representation of a pattern recognizer is given in figure 31.

<table>
<thead>
<tr>
<th>Pattern recognizer</th>
<th>( \rightarrow ) B outputs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern</td>
<td>( \rightarrow ) C</td>
</tr>
<tr>
<td>( \rightarrow ) A</td>
<td>( \rightarrow ) ( \rightarrow ) N mutually exclusive</td>
</tr>
</tbody>
</table>

Figure 31. The block diagram of a pattern recognizer.

Pattern recognition techniques can be broadly divided into two groups.

A) Decision-theoretic approach or statistical recognition.
B) Syntactic or structural approach.

In the early years of development in pattern recognition techniques, the subject is considered as a particular case of the statistical decision theory. Continuous research and unrealizable attempts arising from the unsatisfactory developments in the theory directed the efforts to a somewhat different subject than the statistical decision theory (1). Most of the publications until beginning of 70's deal with the decision-theoretic approach. However, after the 70's syntactic methods in pattern recognition have gained considerable attention.

5.1. A. Decision-theoretic approach;

In this approach first some features are extracted from the patterns. So the features (measurements) or the pattern vector is:

\[
X = \begin{pmatrix}
X_1 \\
X_2 \\
\vdots \\
X_n
\end{pmatrix}
\]

If measurements are binary, according to the characteristics of the pattern, \( X_i \) has a value of 1 or 0. It is useful to think of a pattern as a point in an \( n \)-dimensional Euclidean space. The patterns belonging to the same class are forming a region of the measurement space. Recognition of each pattern, that means assignment of the pattern to a particular class is made by partitioning the feature space. In the two dimensional case, two disjoint pattern classes can be seen in figure 32.
S.2

Figure 32. Two disjoint pattern classes.

Here pattern vectors are in the form of:

\[ T \]

\[ X = (X_1, X_2) \]

Where \( T \) indicates transposition.

5.1.8. Syntactic Recognition;

When the structural information of the pattern is important, the syntactic recognition may be preferable. The syntactic recognition gives the possibility not only to assign the pattern to a particular class but also capability to specify characteristics which make the pattern impossible to belong to an another class.

Syntactic recognition tries to draw an analogy between the structure of patterns and the syntax of a language. In this method patterns are divided into simpler subpatterns. Each simpler sub-pattern is again divided into even simpler subpatterns. Patterns are specified by combination of their subpatterns as sentences are built up by composition of words and words are built up by juxtaposition of characters.

In figure 33 hierarchical description of the picture S is shown. Also in figure 34 the hierarchical structure of an English sentence is given. The analogy between the two structures is obvious.

![Diagram of patterns]

Figure 33. The hierarchical description of the picture S.

* Juxtapositioning of two objects means, placing the objects together without losing the identity of each object. Concatenation of two objects is similar to juxtapositioning, however objects may lose their identity.
**Figure 34. Hierarchical structure of a sentence.**

5.2. The Formal Language Theory;

**DEFINITIONS:**

An "Alphabet" is any finite set of symbols. A "Sentence" is a string of finite length composed of symbols from the alphabet.

Example:

Alphabet = \{X, Y\}
Valid Sentences = \{X, XX, XY, YX, \ldots\}

An "Empty Sentence" is a sentence with no symbols. \(V^*\) is the set of all sentences composed of symbols including the empty sentence. Here empty sentence will be denoted by \(S_0\). \(V^*\) is for denoting the set of sentences \(V^* = \{S_0, X, Y, XY, XX, YX, \ldots\}\).

**Example:**

Alphabet = \{X, Y\}
\(V^* = \{S_0, X, Y, XY, XX, YX, \ldots\}\)
\(V^+ = \{X, Y, XY, XX, YX, \ldots\}\)

A "Language" is a set of sentences constructed from an alphabet.

A "Grammar" is a fourtuple

\[ G = (V_n, V_t, P, S) \]

Where

\(V_n\) is a set of nonterminals (variables).
\(V_t\) is a set of terminals (constants).
\(P\) is a set of production (rewriting) rules.
\(S\) is the start symbol.

\(S\) belongs to the set \(V_n\). \(V_n\) and \(V_t\) are disjoint.
In order to make these definitions more clear, comparison of these definitions with an English sentence is given. The example sentence is "The boy walks carefully". The generation of this sentence starts with an abstract concept called <sentence>. This represents all correct sentences in the English language. The <sentence> can be replaced by <noun phrase><verb phrase>. Noun phrase is again replaced by <article><noun> and verb phrase by <verb><adverb>. Further productions will map <article> into "the", <noun> into "boy", <verb> into "walks", <adverb> into "carefully".

Here
S = {sentence}
Vn = {{sentence},<noun phrase>,<verb phrase>,<article>,<verb>,<adverb} }
Vt = {the,boy,walks,carefully}

Productions are:
S -----) <noun phrase><verb phrase> 
<noun phrase> -----) <article><noun>
<article> -----) the
<noun> -----) boy
<verb phrase> -----) <verb><adverb>
<verb> -----) walks
<adverb> -----) carefully

The Language "L", generated by the grammar G is the set of strings that are composed only of terminals. These strings can be derived from S by suitable productions.

5.3.Problem Formulation;

We assume that the input is available as a set of nonterminals or pattern primitives. The string which is formed from these pattern primitives is forming a sentence of a language. The language that provides the structural description of patterns by means of its primitives and their composition of operations is called "Picture Description Language". There exists a grammar G, where the sentences which it generates are belonging to only one class of patterns. The syntactic recognition process is performing a syntax analysis or "Parsing" of a sentence to determine if the strings are grammatically correct with respect to a given language.

In the case of N classes there exists N grammars and N languages. An unknown pattern is classified into class Ui, if it is a sentence of L(Gi).
5.4. Application of Syntactic Pattern Recognition to the Character Recognition;

5.4.A. Problem Description;

Characters in the Latin alphabet are:

- Capitals = \{A, B, C, ..., Z\}
- Lower case letters = \{a, b, ..., z\}
- Numerics = \{0, 1, ..., 9\}

Alpha-numeric is a set of characters called classes.

- Set of Alpha-numeric classes = \{A, B, ..., Z, a, b, ..., z, 0, ..., 9\}
- There exists 62 classes and 62 grammars which generate 62 different languages.
- Set of grammars = \{GA, GB, ..., GZ, Ga, Gb, ..., Gz, Go, ..., L9\}
- Set of languages = \{LA, LB, ..., LZ, La, Lb, ..., Lz, Lo, ..., L9\}

The input in an unknown character stored in the memory. The character has a two dimensional form,

\[
\text{CHAR}[i, j] = \begin{cases} 
1 & \text{black point}, \\
0 & \text{white point}.
\end{cases}
\]

The first step is to generate a language which is capable of describing the character completely. The description language is a string representation of the two dimensional picture. Generation of this language is explained in section 4.5.A. By the help of this language the complete picture is represented by black and white segments.

![Figure 35. The black and white segments](image)

The Language has the form,

\[
\{(WH)(BL) \ldots (WH)\} \{(WH)(BL) \ldots (WH)\} \ldots \{(WH)(BL) \ldots (WH)\}
\]

Here primitives of the language is white and black segments of the column. The structure of the language is given in figure 36.
In order to classify this language to a pattern class, we apply a syntax check with respect to grammars

\[ G = \{ G_A, G_B, \ldots, G_Z, G_{a}, \ldots, G_{z}, G_0, \ldots, G_9 \} \]

Only one syntax check will succeed and the input will be classified to that class. Syntactic character recognition can be summarized as:

A) Input the two-dimensional picture into the memory.
B) Generate a language which describes the picture.
C) Apply a syntax check with respect to available grammars.
D) Select the class to which the successful grammar belongs.

5.4.B. Preprocessing;

Any character which is placed into the memory, needs preprocessing before the recognition starts. The preparing the data for the recognition process is strictly serial. Therefore preprocessing time must be kept as short as possible. However, the reading of characters into the memory is also serial. If the preprocessing is done while inputting, no additional time is required for this sequential processing. How to combine the necessary preprocessing and inputting is explained in the previous chapter. This sequential task is completely the duty of the input processor. The only sequential operation for the central processors is to compute the buffer addresses. The data preparation for the recognition is summarized below.

1) Input the character into the memory.

```
*
* *
* *
----------
```

Character into the memory.

It is a two-dimensional array, has a language notation:

```
ARRAY CHARACTER[1:rownumber,1:columnnumber];
```
2) Generate the language to describe the pattern.
   It consists of 3 variables. Language notation (explained in 4.5.A)

   ARRAY WHITE SEGMENT[1:WHITE SEGMENf LENGTH];
   ARRAY BLACK SEGMENT[1:BLACK SEGMENT LENGTH];
   ARRAY COLUMN DESCRIPTION[1:COLUMN NUMBER];

   All these arrays describe the pattern completely.

3) Place the pattern into a window.

   Language notations, (explained in 4.5.A).

   INTEGER TOPCOL, TOPCADRES, BOTCOL, BOTCADRES, HEIGHT ;

   After these steps, inputted pattern is ready for recognition. The only sequential operation for central processors is to compute the addresses in the buffer block.

5.4.C. Grammar Generation;

   Recognition of a character means applying rules of available grammars to the language of this character. Therefore first grammars of the each character must be defined. Grammar definition is covering the major part in the syntactic pattern recognition. For the grammar generation following sequences must be taken:

   A) Read and print all the required characters for recognition.
      (Repeat many times)
   B) Collect all prints and study the properties of each character.
      Try to select some unique properties.
   C) Classify the characters which have common properties into the same class.
   D) Study the properties of each character that are in the same class. Try to select unique properties for each character. Form lower classes if necessary.
   E) Repeat C and D until only one character is left in one class.
   F) Form grammars according to the unique properties that are found during classification.

Example:

   Character set={C1, C2, C3, C4, C5, C6}

   After the first study they are divided into two classes.
   {C1, C5} and {C2, C3, C4, C6}

   Again
   {C1}, {C5} and {C2, C4, C6}, {C3}
   {C1}, {C5} and {C2}, {C4}, {C6}, {C3}
5.4.D. Picture Descriptors;

In the previous section generation of grammars are explained. Actually to define a grammar means to extract unique shapes for that character. In Figure 36 (Section 5.4.A.) the hierarchical structure of the description language is given. There the character is divided into columns and columns into white and black segments. However not each column defines a particular shape. Most of the time, combinations of some columns form a specific shape. Therefore before branching into columns a higher level subclasses are necessary.

Example:

A is a character consists of the shape "/", placed on the left side of the shape "\".

"/= inclined line "\"= declined line

*  *  *
*  *  *
*  *  *

Columns 123456789

Figure 37. A under analysis.

The hierarchical structural description of "A" is given in the next figure.

Character A

\[
\begin{array}{ccccccc}
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\end{array}
\]

First Column  Topcol  Topcol+1  Last column
\[
\begin{array}{ccccccc}
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\end{array}
\]

\[
\begin{array}{ccccccc}
/ & / & / & / & / & / & / \\
/ & / & / & / & / & / & / \\
/ \ldots & / & / & / & / & / & / \\
Wh Bl Wh Bl... & Wh Bl Wh... & Wh Bl Wh... & Wh Bl Wh...
\end{array}
\]

Figure 38. Hierarchical structural description of A.

As it is seen in Figure 38, Columns 1, 2, 3, 4, 5 define the shape "/" and columns 6, 7, 8, 9 define the shape "\". Each shape consists of columns and each column consists of black and white segments.
According to this example grammar which generates the character A is:

\[ V_n = \{ /, \}, Y \] 
\[ V_T = \{ \text{WHITESEG} \} \]
\[ P : S \rightarrow "/" \text{ AND } "\"

"/" \rightarrow From the first column until TOPCOL
\[ \text{GREAT(TOP(WHITESEG,COLUMN),TOP(WHITESEG,COLUMN+1))} \]

"\" \rightarrow From TOPCOL until the last column
\[ \text{GREAT(TOP(WHITESEG,COLUMN+1),TOP(WHITESEG,COLUMN))} \]

DESCRIPTORS:

\[ \text{GREAT}(a,b) \] \text{ a is greater than b.}
\[ \text{TOP}(a,b) \] \text{ a is the top segment of the column b.}

To form the grammars, such picture descriptors can be used. These descriptors are used to define relations between pattern subclasses.

5.4.E. Parsing;

When all the grammars are known, recognition is realized by determining whether or not a given pattern represents a valid sentence. This procedure is called "Parsing". There are two kinds of parsing techniques.

A)-Top-down parsing.
B)-Bottom-up parsing.

As it is seen in figure 34, the generation of a language forms a tree like structure. Here the top is the start symbol of the tree. The top down technique starts with the root symbol S and by repeated applications of the productions of the grammar, tries to reach to the given sentence. The bottom-up approach starts with the given sentence and tries to reach to the top of the tree. For the both case, if the parse fails, the given pattern is an incorrect sentence and therefore is not recognized.

The parsing proceeds by means of rules of syntax of the grammar. Syntax is juxtaposition and concatenation of objects. The rule of syntax determines permissible or not permissible relations of objects. To make it more clear parsing procedure will be explained by an example.
Example. The grammar of \( A \) is defined as.

\[
P: S \rightarrow "/" \text{ AND } "\"
\]

\[
/ \rightarrow \text{ From column 1 until Topcol}
\]

\[
\text{Great(Top(Whiteseg,column),Top(Whiteseg,column+1))}
\]

\[
\rightarrow \text{ From column Topcol Until End}
\]

\[
\text{Great(Top(Whiteseg,column+1),Top(Whiteseg.column))}
\]

Here the bottom-up parsing starts as follows.

A)-Find top whitesegments of columns 1 and 2.

B)-Is the top whitesegment of column 1 longer than the second? If so then repeat this comparision for columns 1,2...until topcol is reached. If any failure is found then parsing is unsuccessful. Otherwise the production of "/" succeeds.

C)-Find the topwhite segments of columns topcol and topcol +1.

D)-Is the topwhite segment of column topcol shorter than the next one. If so, then repeat the comparison for columns topcol +1,..., until the last column is reached. If any failure is found then parsing is unsuccessful. Otherwise the production of "\" is successful.

E)-If the productions of "/" and "\" are successful then the character is recognized as \( A \).

5.5.THERMB as a Recognizer:

5.5.A.General;

In this section we shall discuss the realization of the recognition process in THERMB. As it is explained in section 5.4.C, first the grammars of the alpha-numeric must be constructed. To do this, many characters have been sampled from different character sets. The properties of all characters are tried to be extracted. After this extensive study, it has been realized that:

1)-Because of the differential sampling method, noise on curves, inclined and declined lines are filtered. Therefore these figures give reliable informations about the character.

2)-Vertical and horizontal lines are very much deformed. If a character is having any curves, inclined and declined lines search for these, but not for horizontal and vertical lines.

3)-Characters L and T are not distinguishable from each other.

4)-For characters, E, F, T, L, J a special attention must be paid. If necessary, statistical approaches must be combined with the syntactic recognition.
According to their properties capitals can be classified as:

a) \{A, C, G, O, Q, S, \}

b) \{B, D, F, R, \}

c) \{K, M, N, W, V, Y, Z, \}

d) \{E, F, I, J, L, T, U, \}

The class a has a property of monotonously decreasing top white segments until the column TOPCOL. The possible functions of top whites segments is drawn in figure 39.

![Figure 39. Top white segment functions.](image)

The class b consists of characters with well defined right white segments. These segments are horizontal white segments between the last right black segments and the right end of the buffer block.

![Figure 40. Right white segment functions for the character B and D.](image)

The class c is identified with its inclined and declined lines.
5.5.B. Constructing the grammars of the characters "A,C,O,G,S".

All these characters are distinguishable from others because of the properties of their top white segments. This property is defined by "CURVE 1"

CURVE 1:

Curve 1 is a monotonously increasing function "f1" or "f1" and a limited constant function "f2". The length of "f1" must be long enough with respect to the number of columns. Occurrence of one error will be tolerated. Descriptions of "f1", "f2" and an error condition is given in figure 41.

f1:

* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *

Error on f1:

f2:

Error on f2: ****** ****** CURVE1: ******

Figure 41. Definition of CURVE 1.
If "CURVE 1" is found then the character is one from the class A. The statement of the recognition of this class is,

IF CURVE 1 THEN CHARACTER:=A OR C OR G OR O OR OR S
ELSE NOT RECOGNIZED;

The program which searches for CURVE 1 is called "TASK0". The program listing of TASK0 is given in Algorithm 5.1.

**TASK0** in Algorithm 5.1.

```
BEGIN
   ARRAY COLUMNDESCRIPTION[I:COLUMNNUMBER];
   ARRAY WHSEGMENT[I:WHSEGMENTLENGTH];
   BOOLEAN ERROR,CURVE1,TASKREADY;
   INTEGER I,TOPCOL,COLUMNNUMBER,INCREASE,EQUAL,TASKAVAILABLE;
   CONSTANT K1,K2,K3,LIMIT;

   TASKREADY:=FALSE; % initialization of the recognition.
   CURVE1:=FALSE;
   HOLD;
   TASKAVAILABLE:=2;
   RELEASE;
   ACTIVATE;
IF TOPCOL=1 THEN GOTO END;
EQUAL:=INCREASE:=0;ERROR:=FALSE;
   FOR 1:=2 STEP 1 UNTIL TOPCOL DO
      BEGIN
         IF WHSEGMENT[I-1] > WHSEGMENT[I]
            THEN BEGIN
               IF NOT ERROR THEN
                  INCREASE:=INCREASE+1;
               EQUAL:=0;
            END
         ELSE
            IF WHSEGMENT[I-1] = WHSEGMENT[I]
               THEN BEGIN
                  EQUAL:=EQUAL+1;
                  IF EQUAL>LIMIT THEN
                     GO TO END;
               END
            ELSE
               IF ERROR THEN GO TO END
            ELSE ERROR:=TRUE;
      END UNTIL TOPCOL DO;
IF COLUMNNUMBER > K1 THEN
   BEGIN
      IF INCREASE>K2 THEN
         CURVE1:=TRUE
      ELSE GOTO END
   END
   ELSE
      IF INCREASE>K3 THEN CURVE1:=TRUE;
END:
   TASK0 READY:=TRUE;
END.
```
After executing the "TASK0" there is a sufficient knowledge for selecting S. S is defined as follows: for S from column 1 until column number/2 the total number of one black segments are less than a constant K.

TASK 2 is responsible for searching "S".

TASK 2 in Algorithm 5.2.

BEGIN

ARRAY COLUMNDESCRIPTION[1:COLUMNNUMBER];
INTEGER I,COLUMNNUMBER,TASKAVAILABLE,ONEBLSEGMENT,CHARACTER;
BOOLEAN CURVE1,TASK1READY,TASK2READY;
CONSTANT K;

TASK2 READY:=FALSE;
HOLD;
TASKAVAILABLE:=4;
RELEASE;
ONEBLSEGMENT:=0;
IF COLUMNDESCRIPTION[I] NOT EQUAL 1 THEN GOTO MAYBES;

FOR I:=1 STEP 1 UNTIL COLUMNNUMBER/2 DO;
    IF COLUMNDESCRIPTION[I] = 1 THEN
        ONEBLSEGMENT:=ONEBLSEGMENT+1;
    IF ONEBLSEGMENT < K THEN
        MAYBES: BEGIN
            WAIT(TASK 0 READY);
            IF CURVE 1 THEN
                BEGIN
                    CHARACTER:=ASCII(S);
                    HOLD;
                    TASKAVAILABLE:="PUT CODE OUT";
                    RELEASE;
                    ACTIVATE;
                    GOTO REPEAT;
                END
            ELSE TASK2READY:=TRUE;
            ACTIVATE;
        END
    END
END.

After searching for the character "S", if the recognition fails, in order to distinguish other characters, a feature called "CURVE 2" must be searched. CURVE 2 is defined as:

CURVE 2:

CURVE 2 is a monotonously decreasing function "f3" OR "f3" AND a limited constant function "f2". The length of "f3" must be long enough with respect to the number of columns. Occurrence of one error will be tolerated. Descriptions of "f3" and "f2" and error condition is given in figure 42.
Figure 42. Definition of CURVE 2.

There are two characters which have the shape "CURVE 1" but not "CURVE 2". They are special C and G fonts. The C and G fonts which will not pass CURVE 2 test is given in figure 43.

When CURVE 2 is not found then the character is one of the C or G. In this case, G is separated by detecting 3 black segments from the middle of the character until the end. The task which searches for CURVE 2 is called TASK 6. The structure of TASK 6 is given in the algorithm 5.4.

Figure 43. Characters C and G, which doesn't satisfy CURVE 2 test.

Before starting to search for "CURVE 2" it is necessary to detect whether or not the character has a three black segments from the middle till the end. The task which searches for 3 black segments is called TASK 4.
TASK 4 in algorithm 5.3.

BEGIN

ARRAY COLUMNSDESCRIPTION[1:COLNUMBER];
INTEGER I,TOPCOL, COLNUMBER, TASKAVAILABLE;
BOOLEAN TASKREADY, THREEBLSEG;

TASKREADY := FALSE;

HOLD;
TASKAVAILABLE := 6;
RELEASE;

I := COLNUMBER/2;
THREEBLSEG := FALSE;
WHILE NOT THREEBLSEG AND NOT(=COLNUMBER)DO
BEGIN
IF COLUMNSDESCRIPTION[I] = 3 THEN
THREEBLSEG := TRUE;
I := I + 1;
END

TASK 4 READY := TRUE;
END.

TASK 6 in algorithm 5.4.

BEGIN

ARRAY COLUMNSDESCRIPTION[1:COLNUMBER];
ARRAY WSEGMENT[1:WSEGMENTLENGTH];
BOOLEAN ERROR, CURVE1, TASK2READY, TASK4READY, CURVE2, THREEBLSEG;
INTEGER I, TOPCOL, COLNUMBER, DECREASE, EQUAL, TASKAVAILABLE,
CHARACTER;
CONSTANT K1, K2, K3, LIMIT;

TASK 6 READY := CURVE2 := FALSE;
HOLD;
TASKAVAILABLE := 8;
RELEASE;

IF COLNUMBER - TOPCOL < 2 THEN GOTO END;

EQUAL := DECREASE := 0;
ERROR := FALSE
FOR I := TOPCOL + 1 STEP 1 UNTIL COLNUMBER DO
BEGIN
IF WSEGMENT[I - 1] < WSEGMENT[I]
THEN BEGIN
IF NOT ERROR THEN
DECREASE := DECREASE + 1;
EQUAL := 0;
END
ELSE
IF UHSEGMENT[I-1]=UHSEGMENT[I]
THEN BEGIN
    EQUAL:=EQUAL+1;
    IF EQUAL > LIMIT THEN
        GOTO END;
END
ELSE
IF ERROR THEN GOTO END
ELSE ERROR:=TRUE;
END UNTIL COUMNUMBER DO;
IF COUMNUMBER> K1 THEN
    IF DECREASE > K2 THEN
        CURVE2:=TRUE
    ELSE GO TO END
ELSE
    IF DECREASE > K3 THEN CURVE2:=TRUE;
END:
WAIT(TASK 2 READY);
WAIT(TASK 4 READY);
IF CURVE 1 AND NOT CURVE 2 THEN
    BEGIN
        IF THREEBLSEG THEN
            BEGIN
                CHARACTER:=ASCII(G);
                HOLD;
                TASKAVAILABLE:="PUT CODE OUT";
                RELEASE;
                ACTIVATE;
                GOTO REPEAT;
            END
        ELSE
            BEGIN
                CHARACTER:=ASCII(C);
                HOLD;
                TASKAVAILABLE:="PUT CODE OUT";
                RELEASE;
                ACTIVATE;
                GOTO REPEAT
            END
        END CURVE 1 AND NOT CURVE 2;
    TASK6READY:=TRUE;
    ACTIVATE;
END.

GRAMMAR OF A
-------------

When the executions of tasks 0,2, and 6 are ready, a decision can be made whether or not an inputted character is A. The grammar of A is defined as
a)-Curve 1 and Curve 2 are both true.  
b)-First column is having one black segment.  
c)-Height/L1 must be smaller than a constant value. 

The recognition of A is realized by Task 8.  
The listing of Task 8 is given by Algorithm 5.5. 

---

**Figure 44. The definition of the first column of the character A.**

Task 8 in Algorithm 5.5.

BEGIN  
ARRAY COUNDESCRIPTION [1:COLUMNNUMBER];  
ARRAY WHSEGMENT [1:WHSEGMENTLENGTH];  
ARRAY BLSEGMENT [1:BLSEGMENTLENGTH];  
BOOLEAN TASK 4 READY,TASK 6 READY,CURVE 1,CURVE 2;  
INTEGER TASKAVAILABLE,CHARACTER,HEIGHT,BOTCADRES;  
CONSTANT K;  

TASK 8 READY:=FALSE;  
HOLD;  
TASKAVAILABLE:=10;  
RELEASE;  

IF COUNDESCRIPTION [1]=1 AND  
HEIGHT/(HEIGHT+BOTCADRES)-(BLSEGMENT[1]+WHSEGMENT[1])  
< K THEN  
BEGIN  
WAIT(TASK 6 READY);  

IF CURVE 1 AND CURVE 2 THEN  
BEGIN  
CHARACTER:= ASCII(A);  
HOLD;  
TASKAVAILABLE:="PUT CODE OUT";  
RELEASE;  
ACTIVATE;  
GOTO REPEAT  
END;  

END;  

TASK 8 READY:=TRUE;  
END.
GRAMMAR OF 0:

In order to decide that the input is "0" the following conditions must be hold:

A) Curve 1 and Curve 2 are both true.
B) There are no THREEBLACKSEGMENTS.
C) Starting from the last column, before the first "3" or "2" black segments, there must be enough one "blacksegments" such that column number/number of one blacksegments < K.

The character "0" is searched by algorithm 5.6.

Task 10 in Algorithm 5.6.

BEGIN
    ARRAY COLUMNDESCRIPTION [1:COLUMNNUMBER] ;
    BOOLEAN TASK 10 READY, TASK 0 READY, CURVE 1, CURVE 2, THREEBLSEG, FOUND;
    INTEGER CHARACTER, TASKAVAILABLE, I, COLUMNNUMBER, COUNTER ;
    CONSTANT K;
    TASK 10 READY:=FALSE;
    HOLD;
    TASKAVAILABLE:=12;
    RELEASE;
    I:=COLUMNNUMBER; FOUND:=FALSE;
    WHILE NOT(COLUMNNUMBER/2) AND NOT FOUND DO;
        IF COLUMNDESCRIPTION [I] NOT EQUAL 1 THEN
            FOUND:=TRUE
        ELSE COUNTER:=COUNTER+1;
        I:=I - 1;
    END WHILE;
    WAIT(TASK8) ;
    IF CURVE 1 AND CURVE 2 AND NOT THREEBLSEG THEN
        IF FOUND:=FALSE ;
            IF COUNTER NOT EQUAL 0 THEN
                IF COLUMNNUMBER/COUNTER < K THEN
                    FOUND:=TRUE;
                    IF FOUND THEN BEGIN
                        CHARACTER:=ASCII(0);
                        HOLD;
                        TASKAVAILABLE:="PUT CODE OUT";
                        RELEASE;
                        ACTIVATE,
                        GOTO REPEAT;
                    END;
                END;
            END;
        END;
    TASK 10 READY:=TRUE;
    ACTIVATE;
END.
GRAMMAR OF Q

The only possible characters which may not succeed up to the TASK 12 are C, G, Q. In order to separate Q from others following conditions must hold.

a) CURVE1 and CURVE2 are present.
b) If the three black segment test succeeds then the ratio of height/L must be larger than a constant K.
c) If there is no three black segment then starting from the right end of the character, after meeting the first two or three black segments there exists a column with one black segment until the column/2 is reached. The explanation of points b and c are done by figures 45 and 46.

Figure 45. Q under analysis.

```
  ****
  *** **
  *** **
  *** **
  *** **
  ****
```

Column/2 : The one black segment before column/2.

Figure 46. Q which does not have three black segments.

GRAMMAR OF G:

The grammar of G is defined as:

a) CURVE1 and CURVE2 must be present.(some G fonts do not have CURVE2. Discussed previously.)
b) Three black segments must exist.
c) Height / L ratio must be smaller than K
Figure 47. G under analysis.

GRAMMAR OF C:

The grammar of C is defined as:

a) CURVE1 and CURVE2 must be present. (Some C fonts do not have CURVE2. Discussed previously.)
b) No three black segments.
c) Starting from the right end of the character, after meeting the first two or three black segments, there is no one black segment until the column/2 is reached.

The realization of the syntax check of the grammars C, O and G, is similar to the previous tasks explained before.

REFERENCES:

6. The Hardware Design

6.1. General Description;

In this chapter, the circuit design of THERMB will be explained. The circuit design of THERMB can be grouped into two parts.

a) The design of the sensor.
b) The design of the processing unit.

The processing unit design can be grouped into four points,

a) The design of the Input processor.
b) The design of the Central processors.
c) The design of the scheduler and the Common memory.
d) The design of the Output processor.

The hardware design of the present version of THERMB is not having a final structure. Although the present hardware is working properly, it is not an optimum one for a mass production. The reasons are,

a) The system consists of some circuits for the software and hardware debugging.
b) When the project was started, there was no software developing system in the group. Therefore some circuits have been built to load and execute programs.
c) This was the first similar system developed. After this project, it may be possible to modify the hardware to achieve some practical improvements.
d) While developing the recognition routines, it is necessary to have a system where the program flow and structure of the data can be viewed. Therefore THERMB is connected to an another microprocessor that has a program loading, executing and displaying capability. When all the recognition programs are ready, this second processor is not necessary. But in this present model, some software and hardware is used in order to communicate with the second microcomputer.

6.2. The sensor;

The sensor is built around the 64 bit photo-diode scanning array. Although 64 bit resolution is possible, in this first model 32 bits are used. In the sensor there are two prints, where their circuit diagrams are drawn in figures 50 and 51. Mainly, only 4 pins of the sensor are used. All the other terminals must be connected to plus 5 and minus 10 volts. The important connections of the scanner I.C. are,

a) Start scanning. Low active input signal, where the scanner is activated to start scanning the photo-diodes.
b) End of scan. High active. When the scanning is complete, this pulse is generated.
c) Output. The response of diodes are serially outputted from here. The output is divided into two parts. 1 - Odd video. 2 - Even video.

When all the 64 bits of elements are required, then these two outputs must
be connected together. In our case the output is obtained from the pin 8, which this is an even video signal.

d)-Scan clock. With each clock pulse, the response of one diode will be outputted.

The start scanning pulse is given by the input processor. When the scanning starts, after each two clock pulses, the output of one even diode will appear at the pin 8. These pulses are narrow and they need a very high amplification before being used by the input processor. Following cares must be taken for the start scanning pulse.

a) - The start scanning pulse must be synchronized with the sampling signal.

b) - This input is sensitive even when the previous scanning is not completed. Therefore any spike on this line will disturb the triggering pattern completely.

The reason for the synchronization of the start scanning pulse and the sampling signal is explained in figure 48. The sampling frequency is 32 KHz and it is obtained by dividing the scan clock which is 64 KHz. The output of diodes are available, soon after the clock pulses.

\[\text{Scan Clock} \rightarrow \text{Start scanning} \rightarrow \text{Output signal} \]

![Figure 48. Not synchronized character scanning.](image)

As it is seen in figure 48, the start scanning pulse must be synchronized with the sampling signal. The sampling signal is obtained from the scan clock. For the output signal there are two possibilities. Therefore the output signal must appear at the same time with the sampling edge. This is only possible by latching the output pulse until the sampling occurs and then clear the latch for the next output pulse. The start scanning pulse is generated by the software. Thus, the software and hardware must be synchronized. This is done in the processor unit. The start scanning pulse which is applied to the sensor has exactly same duration as the sampling pulse. In figure 49, the timing diagram of the synchronized signals are given. Here the output of diodes are converted to pulses and then latched until sampling occurs. After the sampling, the latch is cleared by the low active "clear latch pulse". This pulse is obtained by dividing the scan clock.
In figure 50, the diagram of the scanner and the amplifier is given. The output of the "end of scan pulse" is applied to the base of the transistor to improve the driving capacity. Here when the sensor is on white paper, the amplifier must saturate. The active operation of the amplifier is not necessary. The start scanning pulse is applied by the input processor. Also the scan clock is supplied from the system clock. The output of the amplifier is applied to another circuit which is built in the sensor. The diagram of that circuit is given in figure 51.

In figure 51, IC4a and IC4b are responsible for the one bit analog digital conversion. Here, IC4a forms a Schmitt trigger with an adjustable triggering level. The output of this Schmitt trigger is changing with the adjustment of the pot-meter. Therefore a comparator is used to let the output be independent of this adjustment. Also this comparator supplies very sharp rising pulses to the latch IC3b. IC3a is used as an T type flip-flop to divide the scan clock for clearing the latch. The output of the latch is inverted with respect to the output of the amplifier because the black level is chosen as a high voltage.

End of scan pulse is latched by two NAND gates. The output of this latch is applied to the input processor to interrupt for the data input. This latch and IC4a are cleared by the start scanning pulse. The reason for the latched interrupt pulse is that, the processor may be still busy with the previous inputted data and it can only respond to the interrupt request, when that routine is ready.
Figure 50. The circuit diagram of the Scanner I.C. and the amplifier.
Figure 51. The circuit diagram of the A/D converter and latches.
6.3. The Processor Design;

The problem formulation can be grouped into these points:

a) A low cost device.
b) A very fast processor.
c) A pattern recognizer.
d) Can do inputting from the particular sensor and do outputting to the particular output device.
e) Equiped with some facilities to make reading easy for the blind.

Reasons for low costs are discussed in chapter 2. By using standard components, omitting mechanical units and mainly depending on electronic components will reduce the cost of the device. Available microprocessors are not fast enough to carry the task alone. To use more processors for a common task is discussed in chapter 3. The costs of microprocessors are motivating the use of more processors for our application.

To design a pattern recognizer is a matter of a software. The hardware can cause limitations in a memory sense, or in speed-system power considerations. The structure of the hardware must fit to the architecture of the system. The system architecture is discussed in chapter 3.

To do inputting from the particular sensor and do outputting to output devices is realized by attaching separate processors for the input and output.

The selection of the microprocessor is quite important. The instruction set of the microprocessor must fit to the task. Z80 is chosen as a microprocessor because of the following reasons:

a) Z80 has more internal registers with respect to many other microprocessors. (10x8 bit active registers and 4x16 bit index registers). As referring to the external memory will be less frequent, the processing will be faster.

b) Bit addressing and testing capability. The recognition routines are requiring bit operations quite often. The Z80 processor can set, reset or test any bit of the external or internal memories.

c) Short instructions. Z80 has relative 2 byte jump instructions. Although these are only page jumps, most of the time the range of jumps are not more than the size of a page. The short instructions are decreasing the required memory size. (Do not decrease the execution time!)

d) Looping instructions such as: block inputting and outputting, block transfer, block search, repeat until the condition is true. These instructions are very often required. The software design of these routines are much more easy and their executions times are shorter than other processors which do not have these instructions.

6.4. The Input processor;

The design of the input processor must be realized by taking the following constraints into account:

a) 32 bit serial input.
b) An output port. This output port will be used for,
1-mutual memory request. 2-interruption of other processors. 3-start character scanning. 4-command sound output.
c) Local program memory.
d) Tri-state bus drivers.
e) Logic for "mutual memory request" decoding.
f) Other logic circuits for decoding, time delay etc.

In figure 52, the clock circuitry, the input microprocessor, two EPROMs are drawn. Here IC6a and IC6b are responsible for the clock generation. This clock will be used by the whole system. The clock frequency will be lowered until 32 KHz, to be used in different parts of the system. For example, 4 MHz is used by the scheduler, 2 MHz is used by the processors, 64 KHz is used for scanning the photo-diodes and 32 KHz is used to latch the sensor output. There is also a switch to select the processor's clock frequency between 2 or 4 MHz.

The clock is applied to the processor via IC6c and via the PNP transistor. Here the transistor is supplying a high current to charge the input capacitors.

IC7a, IC7c, IC6e, IC6f, IC8c are responsible for address decoding.

IC2 and IC3 two 2K EPROMS where IC2 has an address range of 0-2K bytes and IC3 2-4 K bytes. They are programmed for the input processing. The logical formulation of the local memory decoding is:

\[ \neg = \text{NOT} \]
\[ \ast = \text{AND} \]
\[ + = \text{OR} \]

\[
\text{Select IC2} = \text{MREQ} + A_{15} + A_{11}
\]
\[
\text{Select IC3} = \text{MREQ} + A_{15} + A_{11}'
\]

MREQ = Memory request

In figure 53, input-output ports of the input processor are given. The 32 bit serial input is converted to parallel data by the help of the four serial input-parallel output shift registers (IC14-IC17). This parallel data is latched by 4 8-bit D type flip-flops (IC10-IC13). These flip-flops are the input ports of the input processor. Their addresses are 0, 1, 2, 3. They are selected by 8205, which is a 3 to 8 decoder. Here the logical expressions are:

\[
\text{Low active, IC10 enable} = \text{RD} + \text{IOREQ} + A_0 + A_1 + A_2
\]
\[
\text{IC11 enable} = \text{RD} + \text{IOREQ} + A_0' + A_1 + A_2
\]
\[
\text{IC12 enable} = \text{RD} + \text{IOREQ} + A_0 + A_1' + A_2
\]
\[
\text{IC13 enable} = \text{RD} + \text{IOREQ} + A_0 + A_1 + A_2'
\]

RD = Read

IOREQ = Input output request
Figure 52. The circuit diagram of the Input processor.
Figure 53. The circuit diagram of the I/O ports of the input processor
IC18 is an output port which is selected when IOREQ and WR signals are both low. The pin connection of the output port is given below:

- bit 0 - Common memory request
- bit 1 - Start scanning
- bit 2 - Interrupt central processors
- bit 3 - Non-maskable interrupt (Central processors)
- bit 4 - Sound control
- bit 5 - Sound control
- bit 6 - Sound control
- bit 7 - Sound control.

In section 6.2, the necessity of the synchronization of the start scanning with the sampling pulse is explained. This synchronization is realized by IC8 and IC9. The timing diagram of this circuit is given in Figure 54. When the start scanning pulse is generated by the output port, Q1 of the IC9a will rise to high. This voltage will open the gate IC8a and the 32 kHz sampling pulse will pass through this gate to trigger the scanner. With the rising edge of this pulse IC9b will be clocked. This will bring Q2 and Q1 low. Then the gate IC8a is closed. This will continue until the next start scanning pulse, which is delivered by the output port.

---

Figure 54. The synchronization of software with hardware.

Also IC8d and IC6d are used to stop clocking the shift registers when end of scan pulse is generated.

The logic circuitry which is given in Figure 55 is used for the time delaying and decoding. Here IC19a, IC19b, IC20a, IC20b, are used to generate mutual memory request signal. This signal is equal to:

Low active

Mutual Memory Request = (MREQ + A15').bit0 outputport.

The generation of this signal is given in Figure 56.
Figure 55. The logic circuits used for the input processing.
When the mutual memory is free to use the scheduler will generate an enable signal. However, the processor must be only enabled when it works with the mutual memory. This means data bus driver must be only enabled when:

\[ \text{Low active} \]
\[ \text{Bus driver enable} = \text{Enable} + \text{A15} + \text{MREQ} \]

This is obtained by IC21 and IC20e.

When the mutual memory is busy, a wait signal will be send by the scheduler. This wait signal is OR'ed with the single step and delay signal.

\[ \text{Low active} \]
\[ \text{Wait Processor} = \text{Single Step, Delay, Wait} \]

Single stepping is realized by IC24 and IC23c. When the switch 2 is in wait position, with each press of switch 3 only one operation will be carried. By that way, hardware and software debugging of the system can be realized. When the single step operation is not desired, this circuit can be omitted.

When the mutual memory is slow and when the processor is waiting for the mutual memory, after being enabled, the access time for the memory may not be long enough. Therefore, a time delay circuit is built. IC22 and IC18d are responsible for a one clock delay. Processor samples the wait input at the end of the clock cycle 2. When this input is low, it inserts a delay until the wait signal goes high again. In figure 57 the timing diagram is drawn. At the end of T1 processor generates a mutual memory request signal. As the memory is still busy the scheduler sends a wait signal. Processor waits until the wait signal goes high. However, the access time is now about one clock cycle less than the normal operation. Therefore, if the memory is not fast enough, one clock extra delay must be inserted.
Figure 57: The memory access with wait cycles.

In figure 58 the circuit diagram of the bus drivers are drawn. Here IC25 and IC26 are 8-bit tri-state D type flip-flops. When the enable goes low input will be latched to output. Because of the bi-directional property, data bus is driven by a bi-directional bus driver. The direction of data flow is controlled by the T input.

6.5. The Scheduler and the Common Memory;

The operation principle of the scheduler is explained in section 3.4.8. The processor scheduler is a 16 state Moore type sequential circuit. The circuit diagram of the scheduler is given in figure 59. The scheduler is formed of two bipolar PROMS and two double D-type flip-flops.

IC1 and IC4 are 256x4 bit and 32x8 bit PROMS respectively. IC1 is responsible for the state transfer function. The combination of the input and the present state is the input address of IC1. This address will access to the code of the next state which is programmed previously. The output of IC1 will be latched by IC2 and IC3. The outputs of these latches are decoded by IC4. These decoded signals are used to control the access of processors to the mutual resource.

The contents of these PROMS are given in Appendix A of this chapter.

The scheduler is designed for three processors. The capacity of the scheduler can be increased by using more schedulers connected together. e.g. In the case of 9 processors, 4 of such circuits are required.

In figure 60 the circuit diagram of the mutual memory is given. Mutual memory is built around 5 1-K byte static RAMs. These RAMs are decoded by IC5. IC5 is enabled when any processor requests the common memory.
Figure 58. The bus drivers of the input processor.
Figure 59. The scheduler.
Figure 60. The mutual memory.
6.6. Central Processors;

The circuit diagram of the central processor is given in figure 61. In this figure IC3, IC4, IC5, are tri-state bus drivers. IC6 is a parallel I/O port used to communicate with the output processor. Here IC2 is a 2K EPROM, where the recognition programs exist. These drivers and the local memory are enabled by some hardware logic. In figure 62, the circuit diagram of the necessary logic circuit is drawn. This circuit is very much similar to the one which is drawn in figure 55. Again here, IC7 is used for the time delay. IC9a, IC9c, IC8e, IC10a, IC8f are used to decode the mutual memory request. IC10b is used to enable the tri-state bus driver IC5, only when it is necessary. IC10c is used to select the local memory.

REFERENCES:

Figure 61. The central Processor.
Figure 62. The logic used by the central processor.
## APPENDIX A TO CHAPTER 6

### CONTENTS OF PROMS 27520 AND 27519

------------------------

### 27520

Addresses 0-127 All 00.

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>DATA</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>ADDRESS</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>128 STATE A</td>
<td>0110</td>
<td>173</td>
<td>0000x</td>
<td>217</td>
<td>0000x</td>
</tr>
<tr>
<td>129</td>
<td>1001</td>
<td>174</td>
<td>0000x</td>
<td>218</td>
<td>1101</td>
</tr>
<tr>
<td>130</td>
<td>0101</td>
<td>175</td>
<td>0000x</td>
<td>219</td>
<td>0000x</td>
</tr>
<tr>
<td>131</td>
<td>0011</td>
<td>176 STATE G</td>
<td>0110</td>
<td>220</td>
<td>0000x</td>
</tr>
<tr>
<td>132</td>
<td>0100</td>
<td>177</td>
<td>1001</td>
<td>221</td>
<td>0000x</td>
</tr>
<tr>
<td>133</td>
<td>0010</td>
<td>178</td>
<td>0000x</td>
<td>222</td>
<td>0000x</td>
</tr>
<tr>
<td>134</td>
<td>0001</td>
<td>179</td>
<td>0000x</td>
<td>223</td>
<td>0000x</td>
</tr>
<tr>
<td>135</td>
<td>0000</td>
<td>180</td>
<td>0000x</td>
<td>224 STATE N</td>
<td>1110</td>
</tr>
<tr>
<td>136</td>
<td>0110</td>
<td>181</td>
<td>0000x</td>
<td>225</td>
<td>1100</td>
</tr>
<tr>
<td>137</td>
<td>1001</td>
<td>182</td>
<td>0000x</td>
<td>226</td>
<td>0000x</td>
</tr>
<tr>
<td>138</td>
<td>0101</td>
<td>183</td>
<td>0000x</td>
<td>227</td>
<td>0000x</td>
</tr>
<tr>
<td>139</td>
<td>0011</td>
<td>184 STATE H</td>
<td>0111</td>
<td>228</td>
<td>1000</td>
</tr>
<tr>
<td>140</td>
<td>0100</td>
<td>185</td>
<td>1100</td>
<td>229</td>
<td>0010</td>
</tr>
<tr>
<td>141</td>
<td>0010</td>
<td>186</td>
<td>0000x</td>
<td>230</td>
<td>0000x</td>
</tr>
<tr>
<td>142</td>
<td>0001</td>
<td>187</td>
<td>0000x</td>
<td>231</td>
<td>0000x</td>
</tr>
<tr>
<td>143</td>
<td>0000</td>
<td>188</td>
<td>0000x</td>
<td>232 STATE N</td>
<td>1111</td>
</tr>
<tr>
<td>144</td>
<td>1010</td>
<td>189</td>
<td>0000x</td>
<td>233</td>
<td>0000x</td>
</tr>
<tr>
<td>145</td>
<td>1001</td>
<td>190</td>
<td>0000X</td>
<td>234</td>
<td>1101</td>
</tr>
<tr>
<td>146</td>
<td>0101</td>
<td>191</td>
<td>0000X</td>
<td>235</td>
<td>0000x</td>
</tr>
<tr>
<td>147</td>
<td>0011</td>
<td>192 STATE I</td>
<td>1010</td>
<td>236</td>
<td>0100</td>
</tr>
<tr>
<td>148</td>
<td>1000</td>
<td>193</td>
<td>0000x</td>
<td>237</td>
<td>0000x</td>
</tr>
<tr>
<td>149</td>
<td>0010</td>
<td>194</td>
<td>0101</td>
<td>238</td>
<td>0001</td>
</tr>
<tr>
<td>150</td>
<td>0001</td>
<td>195</td>
<td>0000x</td>
<td>239</td>
<td>0000x</td>
</tr>
<tr>
<td>151</td>
<td>0000</td>
<td>196</td>
<td>1000</td>
<td>240 STATE O</td>
<td>1110</td>
</tr>
<tr>
<td>152 STATE D</td>
<td>1111</td>
<td>197</td>
<td>0000x</td>
<td>241</td>
<td>0000x</td>
</tr>
<tr>
<td>153</td>
<td>1100</td>
<td>198</td>
<td>0001</td>
<td>242</td>
<td>0000x</td>
</tr>
<tr>
<td>154</td>
<td>1101</td>
<td>199</td>
<td>0000x</td>
<td>243</td>
<td>0000x</td>
</tr>
<tr>
<td>155</td>
<td>0011</td>
<td>200 STATE J</td>
<td>1011</td>
<td>244</td>
<td>1000</td>
</tr>
<tr>
<td>156</td>
<td>0100</td>
<td>201</td>
<td>1001</td>
<td>245</td>
<td>0000x</td>
</tr>
<tr>
<td>157</td>
<td>0010</td>
<td>202</td>
<td>1101</td>
<td>246</td>
<td>0000x</td>
</tr>
<tr>
<td>158</td>
<td>0001</td>
<td>203</td>
<td>0011</td>
<td>247</td>
<td>0000x</td>
</tr>
<tr>
<td>159</td>
<td>0000</td>
<td>204</td>
<td>0000x</td>
<td>248 STATE P</td>
<td>1111</td>
</tr>
<tr>
<td>160 STATE E</td>
<td>0110</td>
<td>205</td>
<td>0000x</td>
<td>249</td>
<td>0000x</td>
</tr>
<tr>
<td>161</td>
<td>1001</td>
<td>206</td>
<td>0000x</td>
<td>250</td>
<td>0000x</td>
</tr>
<tr>
<td>162</td>
<td>0000x</td>
<td>207</td>
<td>0000x</td>
<td>251</td>
<td>0000x</td>
</tr>
<tr>
<td>163</td>
<td>0000x</td>
<td>208 STATE K</td>
<td>1010</td>
<td>252</td>
<td>0100</td>
</tr>
<tr>
<td>164</td>
<td>0100</td>
<td>209</td>
<td>0000x</td>
<td>253</td>
<td>0000x</td>
</tr>
<tr>
<td>165</td>
<td>0010</td>
<td>210</td>
<td>0101</td>
<td>254</td>
<td>0000x</td>
</tr>
<tr>
<td>166</td>
<td>0000x</td>
<td>211</td>
<td>0000x</td>
<td>255</td>
<td>0000x</td>
</tr>
<tr>
<td>167</td>
<td>0000x</td>
<td>212</td>
<td>0000x</td>
<td>256</td>
<td>0000x</td>
</tr>
<tr>
<td>168 STATE F</td>
<td>0111</td>
<td>213</td>
<td>0000x</td>
<td>257</td>
<td>0000x</td>
</tr>
<tr>
<td>169</td>
<td>1100</td>
<td>214</td>
<td>0000x</td>
<td>258</td>
<td>0000x</td>
</tr>
<tr>
<td>170</td>
<td>0101</td>
<td>215</td>
<td>0000x</td>
<td>259</td>
<td>0000x</td>
</tr>
<tr>
<td>172</td>
<td>0011</td>
<td>216 STATE L</td>
<td>1011</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x = Don't care
# Addresses 0-15 all 00

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>DATA</th>
<th>ADDRESS</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>00</td>
<td>16</td>
<td>(0000)A X1111111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26</td>
<td>(1010)K X0010101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17</td>
<td>(0001)B X0111110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>27</td>
<td>(1011)L X0010101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18</td>
<td>(0010)C X0111101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28</td>
<td>(1100)M X0101011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19</td>
<td>(0011)D X0111011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29</td>
<td>(1101)N X0110011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>(0100)E X0101110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>(1110)D X0100011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21</td>
<td>(0101)F X0011110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31</td>
<td>(1111)P X0100011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22</td>
<td>(0110)G X0001110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>23</td>
<td>(0111)H X0001110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24</td>
<td>(1000)I X0110101</td>
</tr>
</tbody>
</table>
The RL-64P solid-state line scanner is intended for OCR, OPH and industrial control applications requiring up to 64 elements of resolution. Its basic design is similar to that of other Reticon line scanners, but in addition it features a number of important innovations which add to its versatility and ease of application.

The block diagram and pin configuration of the device are shown in Figure 1. It consists of a row of 64 photodiodes spaced on 2 mil centers having a 1.6 mil x 2 mil aperture. The diodes operate in the charge storage mode, i.e., they are continuously active and are discharged by photogenerated current at a rate proportional to the local light intensity. The diodes are accessed in sequence by a shift counter which can be operated at any desired rate between 1 KHz and 1 MHz. When each diode is accessed, it is recharged through a common video line. Thus, a signal which appears on the video line is a train of 64 charge pulses each having a magnitude proportional to the light intensity on the corresponding photodiode and to the time interval between successive scans. It is apparent that one can tailor the speed for sensitivity. By running the shift counter at a slower rate, the diodes can integrate light for a longer time and lower light levels can be detected.

The shift counter is driven by a single external clock consisting of a train of minus 5 volt pulses, which can be easily derived from TTL logic. A start pulse which is required to initiate each scan is generated internally and starts a new scan five clock periods after completion of the previous scan. If a longer or shorter delay between scans is desired, the user can supply the start pulse externally after any arbitrary delay.

The shift counter normally scans all 64 photodiodes before a new start pulse can be loaded into the counter. However, if the user requires less than 64 elements, a terminate-scan pulse can be put in after any arbitrary number of elements have been scanned. Thus the RL-64P can be readily "programmed" to operate as an array with effectively any number of elements between 1 and 64 spaced on 2 mil centers.

Additional flexibility is achieved because odd and even numbered elements are internally connected to operate when less than 64 elements are used. This greatly extends the number of applications for which this device can be used. For example, when 32 elements on 4 mil centers are connected to operate a 4 mil x 2 mil aperture, the number of elements between one and 32 spaced on 4 mil centers can be readily

**POWER REQUIREMENTS**

In the specifications all voltages are stated with respect to common. However, for compatibility with TTL clocks and ease of signal extraction it is recommended that common (pin 5) be run at plus 5 volts and VDD at minus 10 to minus 13 volts. The maximum scan rate is somewhat dependent on the total voltage swing between VDD and common. With 15 volts the device will operate up to about 700 KHz. With 18 volts it will operate at 1 MHz.

Two VDD pins (1 and 16) are provided. These are normally connected together to the minus supply. However, in some cases there may be a slight imbalance in the complementary clocks which are generated on chip. Such unbalances lead to odd/even fixed-pattern noise which is present can be filtered out by placing a 500 ohm potentiometer between pins 1 and 16 and connecting the VDD supply to the center tap.
CLOCK AND START REQUIREMENTS

The RL-64P is driven by a single TTL clock connected to pin 2. The clock frequency should be set equal to the desired scan rate (1 KHz to 1 MHz). The proper clock waveform and a circuit capable of generating it are shown in Figure 2. A start pulse is required to initiate each scan but this is provided internally simply by connecting pin 15 to common and pin 14 to VCC. Using this internally generated start pulse, there is a delay of 5 clock periods between the last element of one scan and the first element of the next scan. If no delay or a longer or shorter delay is required, pin 14 is connected to common and an externally generated 5 volt start pulse is provided on pin 15. This start pulse should be 30 to 50 nsec longer than the clock pulses. A circuit capable of generating the required start pulse is shown in Figure 3.

SIGNAL EXTRACTION

The video output of the RL-64P is a train of N-64 pulse on each scan. The odd numbered pulses appear on pin 9 while the even-numbered ones appear on pin 8. Pin 8 and 9 are normally connected together externally so that all N elements of video appear on the common connection. However, if desired, either pin 8 or pin 9 or both may be used individually to provide N/2(2n) elements of video corresponding to either the 32 even numbered photodiodes or the 32 odd numbered photodiodes spaced on 4 mil centers. The unused video line should be connected to ground to bias the unused diodes.

To obtain usable signal levels the video output must be amplified using either a current or charge amplifier. A suitable current amplifier which utilizes the Reticon CA-10A op amp is shown in Figure 4.

Fig. 2 Drive Circuit Using Internal Start

Fig. 3 Drive Circuit Using External Start

TERMINATION OF SCAN

Unless there is an input to pin 3, all 64 elements of the array are scanned before a new scan can be started. By applying a 5 volt pulse to pin 3 after N-64 elements, the scan is terminated and the array behaves in all respects like an N element array. This allows the user who requires a specific number of elements to use a standard product in his application. The same type of preset counter circuit as that shown in Figure 3 to generate a start pulse can be used to generate a terminate scan pulse. Either internal or external start pulses can be used in conjunction with the terminate scan feature. Using the internal start, a new scan is begun 5 clock periods after the previous scan is terminated using external start; a new scan can be initiated with any arbitrary delay after termination of the previous scan if the terminate scan feature is not used pin 3 should be connected to array common.

The choice of op amp in such a circuit depends on the scan rate to be used. It should have a minimum open-loop gain of 40 dB at the scan frequency and should have a slew rate (volts/sec) of at least 30 times the scan frequency (MHz). To minimize clock noise in the video output, the resistor R7 should be chosen so that the video signal just recovers to the base line between pulses at the maximum clock frequency to be used. The feedback resistor R5 should then be chosen to give the desired output voltage at saturation. At 250 KHz suitable values for R5 and R7 are 20K and 60K ohms respectively. These values, which give about 5 volts output at saturation should be proportionately increased at lower scan rates and decreased at higher scan rates.

At the expense of somewhat more complex circuitry, greater dynamic range can be achieved using a resettable-integrator/sample-and-hold circuit. Such circuits may utilize the internal reset-switch MOS transistors which are connected to the video lines (See Figure 1). These are enhancement mode p-channel devices with a threshold voltage of -2 volts. If they are not used, pins 6, 7, 10 and 11 should be connected to array common.

A peripheral circuit with improved dynamic range appears in block-diagram form in Figure 5. The circuit available from Reticon as the RC-64P peripheral circuit provides a charge amplifier and a sampler and hold output proportional to the light-level input. This circuit provides greater than 2001 dynamic range (peak signal to peak noise). Figure 6 shows the circuit card which provides the required functions, and an oscilloscope
Since the RL-64P operates in the charge-storage mode, the output of each diode (below saturation) is proportional to the light intensity times the time interval between successive scans. Thus, the sensitivity can be specified in terms of charge output per unit exposure. A plot of output versus exposure is shown in Figure 9.

Fig. 9 Charge Output vs. Exposure for a 2500 lx Tungsten Source

Spectral response of the RL-64P is typical of high quality diffused silicon photodiodes, covering the range from 0.4 to 1.1 μ and peaking at 0.85 μ. Typical spectral response and quantum efficiency curves are shown in Figure 10.
### Simple Circuit for Low Frequency Applications

A simple and inexpensive circuit suitable for operation of the RL-64P at scan rates between 5 KHz and 200 KHz is shown in Figure 11. The device is operated in the self-starting mode and the clock drive is provided by a simple unijunction transistor oscillator. The scan rate is set by the resistor R1 with 50 K ohms corresponding to about 100 KHz. The amplifier is a µA711 op amp which gives about 6 volts output at saturation exposure.

Inset in Figure 11 is an oscilloscope photograph of the output of the circuit operated at 100 KHz scan rate. A portion of the array is in the shadow cast by parallel light incident on a 20 mil diameter wire. This photograph illustrates how the RL-64P may be used for size determination simply by counting the missing pulses. Larger or smaller objects may be measured by imaging them onto the array with an appropriate lens.

---

### ELECTRICAL CHARACTERISTICS (25°C)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video Line Bias</td>
<td>-5</td>
<td>-5</td>
<td>-8</td>
<td>volt</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>-15</td>
<td>-15</td>
<td>-18</td>
<td>volts</td>
</tr>
<tr>
<td>Clock Pulse Amplitude</td>
<td>-5</td>
<td>-5</td>
<td>-10</td>
<td>µsec</td>
</tr>
<tr>
<td>Clock Pulse Width</td>
<td>0.25</td>
<td>1.0</td>
<td></td>
<td>µsec</td>
</tr>
<tr>
<td>Start Pulse Amplitude</td>
<td>-3.5</td>
<td>-3.5</td>
<td>-10</td>
<td>volts</td>
</tr>
<tr>
<td>Start Pulse Width</td>
<td>0.03</td>
<td>0.3</td>
<td>1.0</td>
<td>µsec</td>
</tr>
<tr>
<td>Terminant Scan Pulse Amplitude</td>
<td>-5</td>
<td>-5</td>
<td>-10</td>
<td>µsec</td>
</tr>
<tr>
<td>Terminant Scan Pulse Width</td>
<td>0.25</td>
<td>1.0</td>
<td></td>
<td>µsec</td>
</tr>
<tr>
<td>Clock Repetition Rate</td>
<td>10^1</td>
<td></td>
<td>10^6</td>
<td>Hz</td>
</tr>
<tr>
<td>Video Line Capacitance (pin 8 plus pin 9, at -5 volts bias)</td>
<td>19</td>
<td></td>
<td></td>
<td>pF</td>
</tr>
<tr>
<td>End of Scan Output Resistance</td>
<td>5</td>
<td></td>
<td></td>
<td>Kohms</td>
</tr>
<tr>
<td>Power Dissipation</td>
<td>150</td>
<td></td>
<td></td>
<td>watt</td>
</tr>
</tbody>
</table>

### ELECTRO-OPTICAL CHARACTERISTICS (25°C)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Photocell Sensitivity¹</td>
<td>2.3</td>
<td>2.3</td>
<td>7</td>
<td>pA cm²/µwatt</td>
</tr>
<tr>
<td>Uniformity of Sensitivity</td>
<td>&gt;3</td>
<td>&gt;3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Saturation Exposure¹</td>
<td>1.7</td>
<td>1.7</td>
<td></td>
<td>µwatt sec/cm²</td>
</tr>
<tr>
<td>Saturation Charge (+ 5 volts)</td>
<td>3.9</td>
<td></td>
<td></td>
<td>pCoul</td>
</tr>
</tbody>
</table>

### MECHANICAL CHARACTERISTICS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Diodes</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Center to Center Spacing</td>
<td>2 mils</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Effective Diode Aperture</td>
<td>1.6 mils x 2 mils</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Package Size (nominal)</td>
<td>1600 mils x 500 mils</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Package Configuration</td>
<td></td>
<td></td>
<td>Std. 16 lead DIP with quartz lid</td>
</tr>
</tbody>
</table>

### ABSOLUTE MAXIMUM RATINGS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage with Respect to Common</td>
<td>0</td>
<td>-20</td>
<td>volts</td>
</tr>
<tr>
<td>Storage Temperature</td>
<td>-55</td>
<td>125</td>
<td>°C</td>
</tr>
<tr>
<td>Temperature Under Bias</td>
<td>-55</td>
<td>85</td>
<td>°C</td>
</tr>
</tbody>
</table>

¹ All voltages stated with respect to Common

---

![PACKAGE OUTLINE](image_url)
T.H.E.R.H.B. (Technische Hogeschool Eindhoven Reading Machine for Blind) is a pipe-line processing unit designed to help the blind to read printed texts.

In this report the complete hardware and software design of THERHB is given. Some new features are developed to guide the blind while reading. Also a special parallel processor is developed to achieve the optimum performance for this particular task. Considerable attention is paid to processor communications and task scheduling.

Characters are read by the help of a hand carried optical sensor. The inputted characters are recognized by central processors. When the recognition is complete, an auditory or tactual display is generated to let the blind understand what the printed text is.
In order to make line tracing possible, THERMB is equipped with different sound indications, where each sound indicates the condition of reading.

Even without having any extra guides, reading by hand-carried sensor is quite possible. However, by using simple tools like a ruler, a very fast reading can be obtained.

Also a low cost mechanical unit is proposed to achieve fast page reading.

2.4. C. Use of Sound Indications and a Reading Example;

In order to make it clear, how to read by THERMB will be explained with an example.

A) |
    | No Sufficient light (Sensor is raised up)
    |
    Diode Array

Sound: A High tone of 2 KHz.

B) |
    | A, B, C, D...
    | Top end is black
    | A, B, C, D...
    Diode Array

Sound: Tone of 1 KHz.

C) |
    | A, B, C, D...
    | Bottom end is black
    | A, B, C, D...
    Diode Array

Sound: Tone of 500 Hz.
A, B, C, D, ...

D) |
    |
    |
    | A, Character is placed properly.
    | A, B, C, D, ..

Diode Array

Sound: Tone of 250 HZ. (Only when the system is in the error-state as a result of a previous error.)

E) |
    |
    | All white
    |

Diode Array

No Sound is generated

Switches:

1- Reset = Resets the system, makes buffer empty, waits for the sensor to be placed on white paper.
2- Clear Error = Clears the Error-State.

When there is no error-state and when no error conditions occur, characters will be read without any sound output. But as soon as an error condition arises, whatever the output is, corresponding tone will be generated. Only total white does not generate any tone.

When clear Error switch is pressed, characters will be accepted until a new error condition occurs.

When an Error condition occurs, only the last uncompleted input will be rejected. All other buffer blocks will be preserved.

Example:

1- Press Reset button. Clears error state and all buffer blocks.

2- Hold the sensor in hand. High tone of 2 KHz is generated. Error state enters. All diodes are black.

3- Put the sensor on the page. No sound. (white paper) Error state exists.

4- Slide down the sensor till you find the first line.

Diode array

T H E R E B I S A M A C H I N E .............
5- If you get the lowest tone, (character at the middle) move the sensor to the most left side. No sound.

Diode array

```
THERM B IS A MACHINE ............
```

6- Press Clear Error and move sensor horizontally to the right. No sound is generated. Characters are read one by one.

Diode array

```
THERM B IS A MACHINE ............
```

7- Error condition occurred. Bottom end is exceeded. A tone of 500 Hz is generated. A is rejected.

Diode array

```
THERM B IS A MACHINE ............
```

```
...... WHERE THIS METHOD IS USED .......
```

8- Move the sensor upwards. Obtain a lower tone (250 Hz).

Diode array

```
THERM B IS A MACHINE ............
```

```
...... WHERE THIS METHOD IS USED .......
```

9- Move the sensor to the left till no sound is generated.

Diode array

```
THERM B IS A MACHINE ............
```

```
...... WHERE THIS METHOD IS USED .......
```

10- Press Clear Error and move the sensor to the right. No tone is generated. Characters are read one by one.

As it is shown with this example, by the help of these sound indications, it is very easy to read a text.

2.4.D. A Simple Reading Aid;

It is also possible to build a low cost mechanical unit to improve the reading speed with a considerable degree. This unit can be constructed from a ruler which its right end can be fixed and a vertical guide to let ruler slide up and down.
When the most right character is properly placed, the first screw can be fixed. By shifting the sensor to the left and by the help of the generated tones, the correct position of the ruler can be determined. At this stage the second screw must be fixed. After all this line can be read at once.

Like type machines, by pressing Shift Down button ruler can be moved downwards with constant increments. The incrementing value can be selected by a switch. When the suitable increment value is determined, all the page can be read line by line without searching for correct position of the ruler at each step.

The schematic drawing of this unit is shown below.

Figure 1. The view of the reading aid.

1- Sensor.
2- Ruler.
3- First screw.
4- Second screw.
5- Release button. Is used to move ruler up and down manually.
6- Line feed. Moves ruler one step downwards.
7- Line space adjustment.
2.4.E. Adapting to Different line Spacing;

Use of generated tones is explained in section 2.4.C. When the top and/or bottom end diodes get less light, Error State is introduced. During Error State any kind of input generates a different sound. Also characters are not accepted for recognition.

Top

<table>
<thead>
<tr>
<th>No light =====&gt; Error State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Bottom

<table>
<thead>
<tr>
<th>No light =====&gt; Error State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Photo diode array

Figure 2. Conditions for Error State.

Character on the same line which are going to be read must not be longer than the acceptable height. In our case maximum allowable height is 6 mm. Many printed texts are having character heights of about 3 mm. Although by using lenses the effective height can be decreased, in most practical cases this will be unnecessary.

Experiences show us that in the case of less line spacing problems may arise. If the distance between one line and two adjacent lines plus the character height is not bigger than the effective length of photo diodes then the reading is impossible. In order to read successfully following conditions must hold;

Character height < Effective Length

Character height + 2(Line spacing) > Effective length

<table>
<thead>
<tr>
<th>0</th>
<th>TO HAVE A SUCCESS........</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Effective: 0</td>
</tr>
<tr>
<td></td>
<td>Length: THERM IS A ..........</td>
</tr>
<tr>
<td></td>
<td>TO ADAPT TO DIFF........</td>
</tr>
</tbody>
</table>

Photo diode array

Figure 3. Problems arising from insufficient line spacing.
One simple solution to that problem is to decrease the effective length. To do this we propose such a simple mechanism.

Figure 4. A Thumbwheel switch.

As it is shown in figure 4, on market there are push button Binary Coded Decimal switches. With each press, the number which is placed on the side of the button will be incremented by one. The output of thumbwheel switches are B.C.D. of this number. Most of the time these numbers are printed in three dimensional way. Therefore blind can feel what the present number is.

Here, 0 will correspond to full effective range. When the button is pressed, with each increase of number, the range will decrease. So the effective length is:

\[ \text{Effective Length} = \text{Full Range} - \text{Number} \]

When the reading is not possible because of the insufficient line spacing, the blind can easily decrease the effective length by pressing this button.

The output of this switch will be inputted by the computer to detect what the effective length should be. Then the effective length will be adapted by means of a software.

2.4.F. Output Units;

Output units which can be used by THERMB are:

a) Different sounds.
b) Spelled speech synthesizers.
c) Talking machines.
d) Piezo-electric tactile stimulators.
e) Three dimensionally printed paper tapes.

All these output units can be adapted to the system by designing the output processor according to the characteristics of this attached unit.

To use different sounds for each character is proved to be unsuccessful. Instead of using sounds, spelled speech synthesizers can be used [6]. These machines are practically realizable and can be added to THERMB.
photograph of the actual performance waveform is shown in Figure 7. Shown is the response from a 30-rad front illuminated band imaged onto the array using 1 optical. The “box car” type sampled-and-held output can be easily thresholded or A/D converted into multiple gray levels.

![Fig. 5 Block Diagram of RC-64P Charge-Amplifier Circuit](image)

**SENSITIVITY AND SPECTRAL RESPONSE**

Since the RL-64P operates in the charge-storage mode, the output of each diode (below saturation) is proportional to the light intensity times the time interval between successive scans. Thus, the sensitivity can be specified in terms of charge output per unit exposure. A plot of output versus exposure is shown in Figure 9.

![Fig. 8 Circuit for Obtaining End-of-Scan Output Pulse](image)

**SYNCHRONIZATION**

Either an external start pulse or a terminate-scan pulse can be used to synchronize an oscilloscope display or other circuitry to the periodic scanning of the array. On the other hand, if the array is used in the self-starting mode with no termination of scan, the end-of-scan output on pin 12 can be used for synchronization purposes. In order to use this output without introducing unwanted “glitches” into the video, the voltage excursion on pin 12 should be minimized by using a circuit such as that shown in Figure 8. If it is not used, pin 12 should be connected to array condition.

![Fig. 6 Actual Response of RL-64P in RC-64P Circuit](image)

![Fig. 9 Charge Output vs. Exposure for a 2870K Tungsten Source](image)

Spectral response of the RL-64P is typical of high-quality diffused silicon photodiodes, covering the range from 0.4 to 1.1 μ and peaking at 0.85 μ. Typical spectral response and quantum efficiency curves are shown in Figure 10.

![Fig. 10 Typical Spectral Response and Quantum Efficiency as a Function of Wavelength](image)
# Simplified Circuit for Low Frequency Applications

A simple and inexpensive circuit suitable for operation of the RL-64P at scan rates between 5 KHz and 200 KHz is shown in Figure 11. The device is operated in the self starting mode and the clock drive is provided by a simple unijunction transistor oscillator. The scan rate is set by the resistor R with 50 K ohms corresponding to about 100 KHz. The amplifier is a pA775 op amp which gives about 6 volts output at saturation exposure.

Inset in Figure 11 is an oscilloscope photograph of the output of the circuit operated at 100 KHz scan rate. A portion of the array is in the shadow cast by parallel light incident on a 30 mil diameter wire. This photograph illustrates how the RL-64P may be used for size determination simply by counting the missing pulses. Larger or smaller objects may be measured by imaging them onto the array with an appropriate lens.

---

## Package Outline

---

### Electrical Characteristics (25°C)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video Line Bias</td>
<td>–5</td>
<td>0</td>
<td>–8</td>
<td>volt</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>–15</td>
<td>–5</td>
<td>–18</td>
<td>volts</td>
</tr>
<tr>
<td>Clock Pulse Amplitude</td>
<td>–3.5</td>
<td>–5</td>
<td>–10</td>
<td>volts</td>
</tr>
<tr>
<td>Clock Pulse Width</td>
<td>0.1</td>
<td>0.25</td>
<td>1.0</td>
<td>μsec</td>
</tr>
<tr>
<td>Start Pulse Amplitude</td>
<td>–3.5</td>
<td>–5</td>
<td>–10</td>
<td>volts</td>
</tr>
<tr>
<td>Start Pulse Width</td>
<td>0.15</td>
<td>0.3</td>
<td>1.0</td>
<td>μsec</td>
</tr>
<tr>
<td>Terminate Scan Pulse Amplitude</td>
<td>0.1</td>
<td>0.25</td>
<td>1.0</td>
<td>μsec</td>
</tr>
<tr>
<td>Terminate Scan Pulse Width</td>
<td>10^7</td>
<td></td>
<td></td>
<td>Hz</td>
</tr>
<tr>
<td>Video Line Capacitance (pin 8 plus pin 9, at</td>
<td>19</td>
<td></td>
<td></td>
<td>pF</td>
</tr>
<tr>
<td>5 volts bias)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>End of Scan Output Resistance</td>
<td>5</td>
<td></td>
<td></td>
<td>Kohms</td>
</tr>
<tr>
<td>Power Dissipation</td>
<td>150</td>
<td></td>
<td></td>
<td>mwatt</td>
</tr>
</tbody>
</table>

### Electro-Optical Characteristics (25°C)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Photodiode Sensitivity</td>
<td>2.3</td>
<td></td>
<td>pA cm²μwatt</td>
</tr>
<tr>
<td>Uniformity of Sensitivity</td>
<td>±1.3</td>
<td>±1.7</td>
<td>%</td>
</tr>
<tr>
<td>Saturation Exposure</td>
<td>1.7</td>
<td></td>
<td>μwatt sec/cm²</td>
</tr>
<tr>
<td>Saturation Charge (-5 volts)</td>
<td>3.9</td>
<td></td>
<td>pC</td>
</tr>
</tbody>
</table>

### Mechanical Characteristics

Number of Diodes: 64  
Center to Center Spacing: 2 mils  
Effective Diode Aperture: 1.6 mils x 2 mils  
Package Size (minimum): 800 mils x 300 mils  
Std. 16 lead DIP with quartz lid

### Absolute Maximum Ratings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage with respect to Common</td>
<td>0</td>
<td>20</td>
<td>volts</td>
</tr>
<tr>
<td>Storage Temperature</td>
<td>55</td>
<td>±125</td>
<td>°C</td>
</tr>
<tr>
<td>Temp. Range Under Bias</td>
<td>55</td>
<td>±85</td>
<td>°C</td>
</tr>
</tbody>
</table>