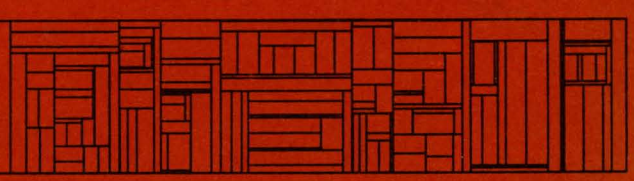
AN OVERVIEW OF THE MATHILDA SYSTEM

Peter Kornerup
Bruce D. Shriver

DAIMI PB-34
August 1974

Institute of Mathematics University of Aarhus
DEPARTMENT OF COMPUTER SCIENCE
Ny Munkegade - 8000 Aarhus C - Denmark
Phone 06-12 83 55



An overview of the MATHILDA system *

by

Peter Kornerup, University of Aarhus, Aarhus, Denmark
Bruce D. Shriver, University of Southwestern Louisiana,
Lafayette, Louisiana

Abstract

A dynamically microprogrammable processor called MATHILDA is described. MATHILDA has been designed to be used as a tool in emulator and processor design research. It has a very general micro-instruction sequencing scheme, sophisticated masking and shifting capability, high speed local storage, a 64-bit wide main data path, a horizontally encoded microinstruction, and other facilities which make it reasonably well suited for this purpose. This paper presents an overview of the MATHILDA system.

* This work is being partially supported by NATO Grant No. 755 and the Danish Research Council Grant 511-1546.

1. INTRODUCTION

1.0 Background and general design criteria

In the spring of 1971 the Department of Computer Science at the University of Aarhus was considering buying a standard minicomputer to control some peripherals and to simulate a standard medium-speed batch-terminal. At the same time a group was working with an integrated software and hardware description language called BPL [1]. To support this group, and, additionally, make the use of such a minicomputer more flexible, it was instead decided to design and construct a microprogrammable mini within the department itself. The design was started and went on during the summer 71. The resulting machine, the RIKKE-0, was partially constructed, and started running in early 1972. In the meantime, a number of additional departmental projects were proposed, some of them were started while others were considered not to fit in with the present design. Among the latter projects were some from numerical analysis, the problem being the data width. RIKKE-0 has a short word size (16-bit) and in order to obtain an efficient implementation of even standard arithmetic operations a wider word was needed.

It was therefore suggested that a microprogrammed functional unit with a wider data path could be attached to RIKKE-0 as an I/O device, together with a wider memory. This organization would allow the problems of numerical analysis and those of the system-software to be more or less separated on the independent units. It was the functional unit which eventually became the MATHILDA machine (a detailed description of this machine can be found in [2]).

The design of MATHILDA went on during mid-1972. It became apparent during the design phase that those features which were felt needed and those which came as side effects, covered and extended those of the prototype RIKKE-0. As it was previously decided to construct more RIKKEs by other funding, the MATHILDA design was adapted for the further constructed RIKKEs with the (almost only) exception of the data widths being respectively 64-bit and 16-bit. Thus, together with a modularity and homogeneity in the hardware design, an economy in print-layout and construction was desired. As an example, all resources of the main data path are laid out on one print board, 8-bits wide. Two of the boards comprise one whole RIKKE bus structure with all registers, shifters,etc. Again, four of these RIKKE boards give the MATHILDA bus. Also the microprogrammed control unit for each machine is identical and is, in fact, also made up of several modules.

Modularity and homogeneity were general criteria for the design. It was, from the very beginning, apparent that the design was going to be rather complex, and we had to keep the number of different features low. This was partwise achieved by using a standard "control-register group" for controlling the various resources of the system.

Two general software ideas used in applications had a good deal of impact on the design: (1) the ability to run, on the higher level, a number of virtual machines which are to be multiprogrammed (or multiplexed) on the microlevel, and (2) the virtual machines are to be defined through several

layers, e.g. in a block-structured environment. These ideas greatly influenced the design of the control unit, especially with respect to the capabilities of addressing. Many addressing facilities known at the virtual level, and in some cases more than those, are here visible on the micro level.

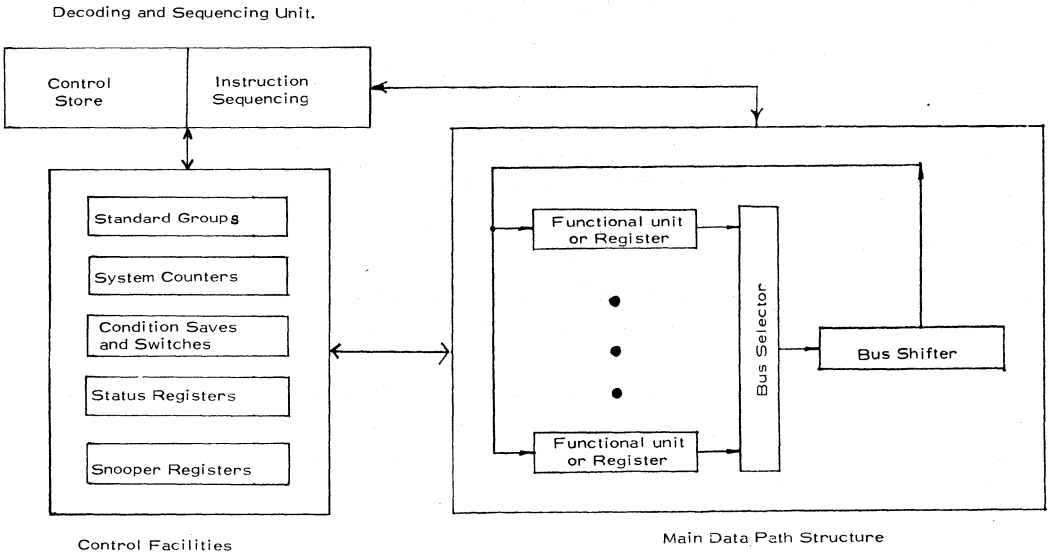
Another criteria was to have a clean and consistent way of dealing with the timing problems. We decided not to force the speed, rather we would have a slower but less tricky machine.

2. A BRIEF DESCRIPTION OF MATHILDA

2.0 An overview

One may consider MATHILDA, shown in Figure 1, as composed by three layers, each of which is controlling the lower ones:

a) The decoding and sequencing unit.
b) Control-facilities.
c) The main data path.



The MATHILDA Processor.
Figure 1.

MATHILDA has a single 64-bit wide data path (called the Main Data Path, MDP) with 8 inputs consisting of various 64-bit wide registers, shifters, masks, and so on. It also has a number of control-information data paths and storage elements dedicated to those places where control of system resources can be exercised. The MDP can itself act as a data transformational element. All information transported over the MDP is subject to the following three transformations, in this order:

1) Masking by a selected, stored mask which has been specified by the user.

2) Shifting by a specified number of bit-positions via a right cyclic barrel shifter.

3) Masking by either an end-off mask of specified amount from the right or left, or by a selected stored mask as in (1).

Furthermore, during data transport, two functional units are operating on the transported information (as present between step 1 and 2 above), and without effecting the information, providing the system with control information (bus parity, test if the BUS ≡ 0, and bit-encoding, see 2.3.3).

The amount of parallelism in data transfers which MATHILDA allows for is restricted, in some sense, on the MDP. The parallelism does not concern the moving of actual data, but controlling the source, data path, destination of the transported data, and the control of functional units operating on the actual data.

The control-facilities consist of storage-elements, data paths and functional units which allows for a high degree of parallelism and permits a variety of external, stored or computed control upon the units on the main data bus. The possible sources of control-information are the following:

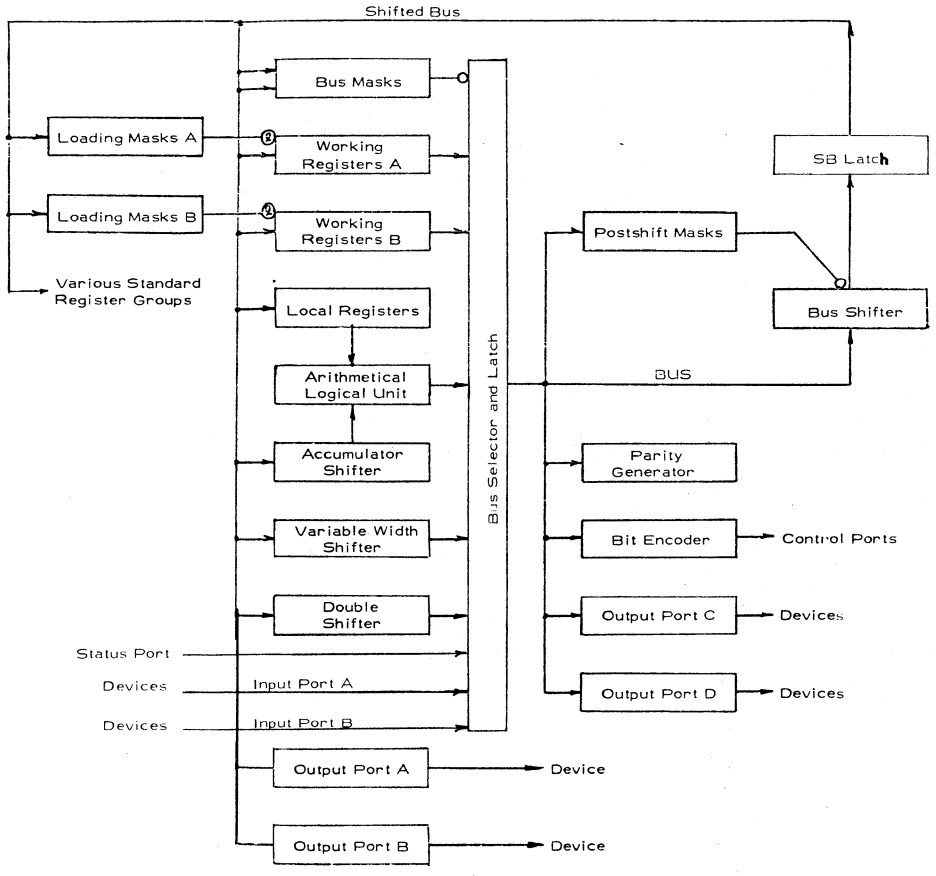
1) CM: the current microinstruction
2) EX: an external register (from controlling processor)
3) BE: the bit-encoder unit (see 2.3.3)
4) SB: rightmost bits of the shifted bus
5) SG: registers for residual or saved control

It should be noted that item (1) above allows the user "immediate control" over the system's resources whereas items (2-5) offer varying degrees of "residual control" in the context of Flynn and Rosin [3]. Furthermore, among the control facilities are system counters, local storage groups, and condition save register. There are also "status" and "snooper" facilities, which can be used to gather data useful in computing statistics concerning the system and thus enhance the experimental nature of this machine.

A microinstruction is 64-bits wide. Particular fields of a microinstruction are highly encoded while others are minimally encoded. The minimally encoded fields (from 1 to 3 bits) control particular highly used system resources and the instruction sequencing. The highly encoded fields (typ. 8-bits) are mostly used for exercising the control facilities associated with the arithmetic logical unit, the residual control standard groups, addressing of local storage elements, and so on. Up to 4 highly encoded microoperations can be specified in one microinstruction while at least 7 minimally encoded operations can occur.

2.1 The Main Data Path, MDP

The MDP forms the base of the system, and we will begin a more detailed discussion of the system with it, the lowest level, where the actual data handling takes place, controlled by the upper levels. The MDP is shown in Figure 2.



MATHILDA Main Data Path

Figure 2

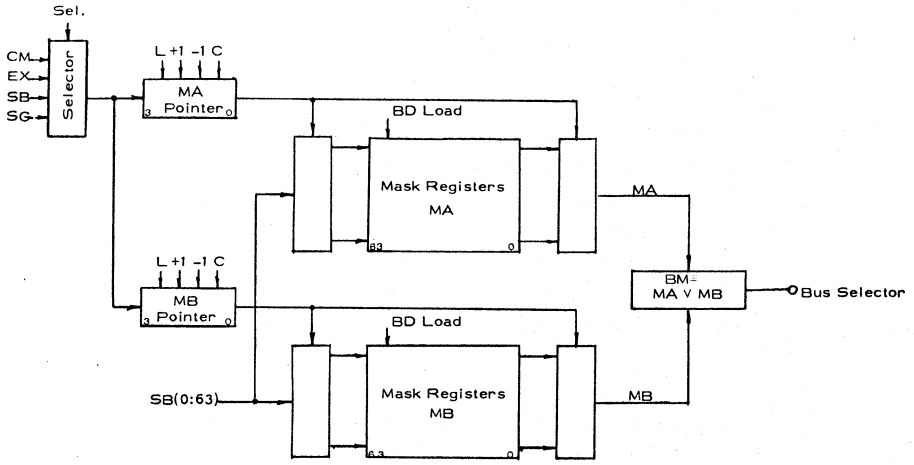
In every microinstruction it is possible to take the contents of a specified 64-bit wide source and mask it by use of a mask, BM, which has been specified by the user. The masked data is buffered in a latch, the output of which is termed the BUS.

The data on the BUS is continuously being encoded, yielding various types of control-information (parity and bit-encoding, see 2.3.3). Furthermore, the BUS information may directly be used for loading into various special destinations, e.g. output buffers. The data on the BUS is passed through the bus shifter, BS, which when enabled shifts the data n bit position right cyclic; where $1 \leq n \leq 63$. The physical shifter is constructed in a

manner quite similar to the barrel shifter of the CDC 6400 series machines. After shifting, the information is masked again, this time by a combined mask PM ∨ PG, PM being similar to BM, and PG as an end-off mask generator, which allows the Bus-shifter combined with PG to result in a logical shift. The information after shifting and postshift-masking is buffered in a latch, called the SB Latch. The output of the SB Latch is called the Shifted Bus, SB, and is finally loaded into selected destinations.

The Bus Mask, BM

The Bus Mask, BM, is composed from two independently stored masks: BM = MA ∨ MB. Both MA and MB are read from a store each containing 16 such masks. The stores have their own address register, as shown in Figure 3.

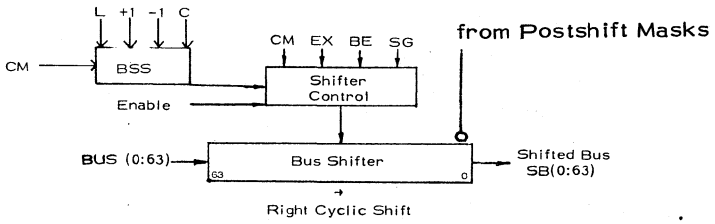


Bus Masks, BM
Figure 3

The reason for the inclusion of double masks is that one group of masks (say MB) containing a no-mask and an all-mask can be used to enable/disable the other group of masks (say MA). Data is loaded into MA and MB from the SB.

The Bus Shifter, BS

The implementation of the BS is 64 parallel selectors: one selector for every position in the output which selects which of the 64 input lines of the BUS is to be used (in a right cyclic connection) as the corresponding output bit. When no shift is required, the selectors all reside in a standard no shift position by disabling the selector via a dedicated bit in each microinstruction. When the BS is enabled, a BS Source Selector, BSS, see Figure 4, determines the source which is to be used as the shift specification.

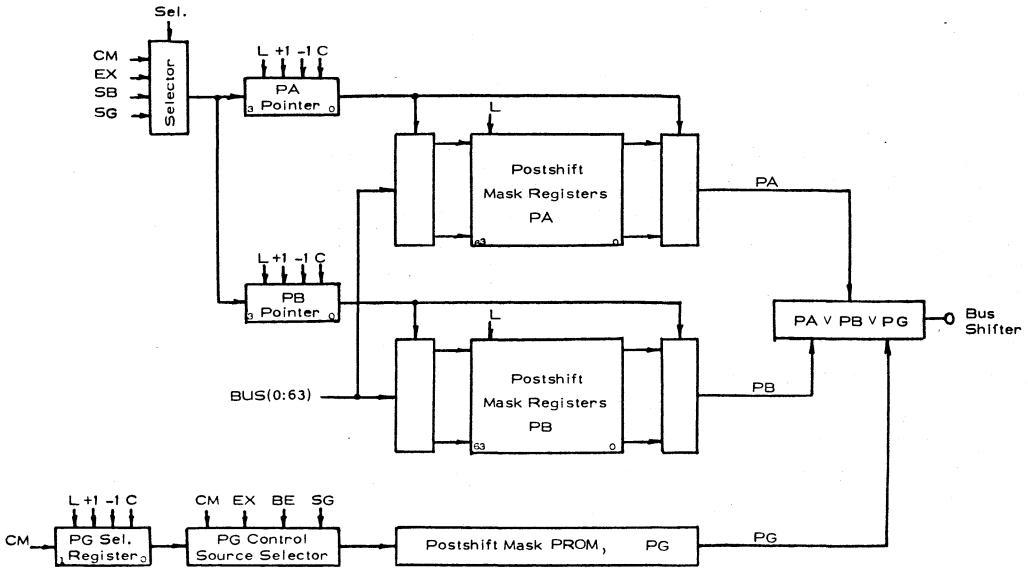


Bus Shifter, BS

Figure 4.

The Postshift Masks, PM and PG.

The output of the BS is again masked by the masks PM ∨ PG, see Figure 5. Here PM ≡ PA ∨ PB is identical to BM with the single exception that PA and PB are loaded from the BUS whereas MA and MB are loaded from the SB. The PG is a PROM whose contents can be combined to yield the 128 masks which are required to make the BS appear as a logical left/right shifter as well as a cyclic left/right shifter. The enabling of the PG is determined by the PM, i.e. PM contains both a no-mask and an all-mask, thus allowing it to be thought of as a switch for the operation of PG. The control of the PG, i.e., the specification of which mask to apply is similar to the specification of the shift amount for the BS, i.e., a selector PGS, as shown in Figure 5, determines the source.



Postshift Masks.

Figure 5

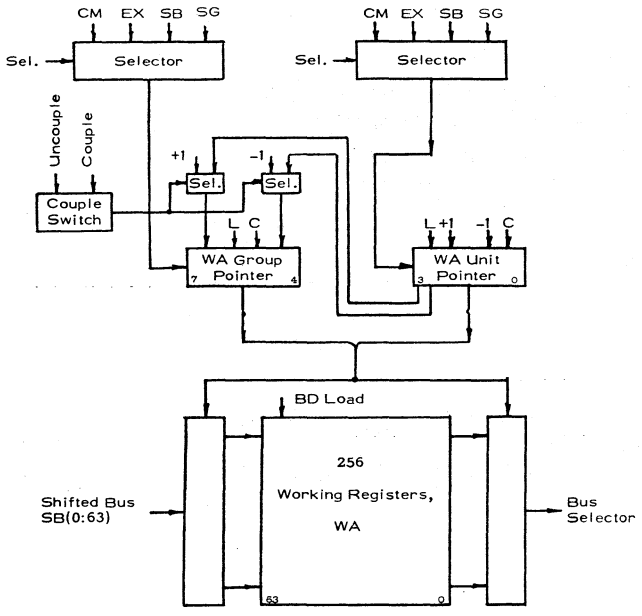
If the same source of control is used for BS and PG, the six low order bits will specify an amount n of right cyclic shift in BS, and the seventh bit will specify whether a logical rightshift of n places, or a logical leftshift of 64-n bit positions will be the result when PG is enabled (i.e., PM ≡ PA ∨ PB ≡ no mask).

2.2 Additional MDP resources

The other MDP resources, shown in Figure 2, will now be briefly discussed. These resources are either possible sources of a bus-transport, or destinations for a transport, or both. Except for the arithmetic-logic unit, ALU, they are all some sort of registers, some are pure storage elements (WA, WB, LR), some are shifters (AS, VS, DS), and the rest for I/O communication (IA, IB, OA, OB, OC, OD).

Working registers (WA and WB) and their Loading Masks (LA and LB)

The system contains two local storages, WA and WB both containing 256 words. They are composed of 80 nsec, 256 x 1 bit chips. As they are identical we will only consider one of them, say WA, which is shown in Figure 6. Addressing of WA is made by a 8-bit pointer, WAP, which determines which location to read or write. WAP is, in fact, composed of two 4-bit pointers which may be coupled together (or be decoupled), which allows for considering WA as either 256 elements, or as 16 groups of 16 registers.



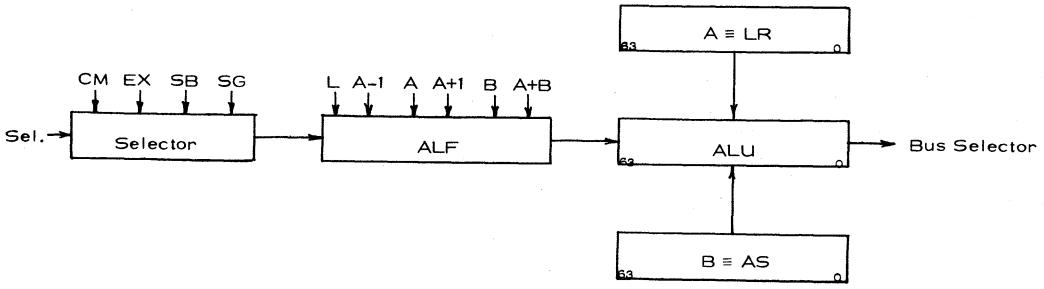
Working Registers A, WA

Figure 6.

The writing of data into WA is also masked by means of a loading mask, LA. LA makes it possible to load only selected fields of the addressed WA-word, without affecting the remaining word. This permits the construction of say the result of a floating point operation, by loading the fields of the packed representation separately as they are being computed. LA (and LB on WB) is again a group of 16 masks with its own address-mechanism. (In fact LA is identical to a register group as shown in Figure 10.)

The Arithmetic and Logical unit, ALU

This is a standard 64-bit wide ALU, see Figure 7, operating within the cycle time of the machine. It is implemented in 3 level carry look ahead logic. It can give 16 arithmetic and 16 logical operations on two operands. The operation to be performed and the carry-input is specified by the contents of a function register, ALF, which can be preset to certain standard functions or loaded from a standard group.



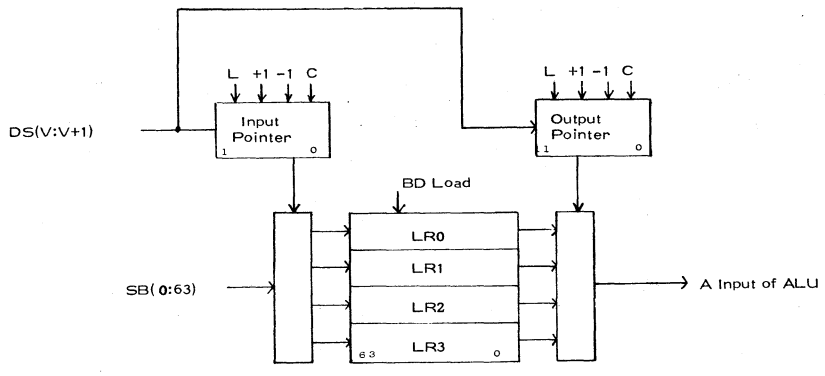
Arithmetical Logical Unit, ALU

Figure 7

The inputs to the ALU, i.e. LR and AS, are discussed below.

The Local Registers, LR

One of the inputs to the ALU is the output of a 4 element register group, the Local Registers, LR, see Figure 8. LR has independent address mechanisms for reading and writing. The 2-bit read-address LROP (LR-output pointer) can be specified (loaded) in such a way that it allows for a fast small table-look-up. The input address pointer, LRIP, is loaded similarly and both addresses can be incremented and decremented (wrap-around). The DS (V:V+1) input to LRIP is discussed below. LR has no direct access to the bus, but its contents must be gated through the ALU.

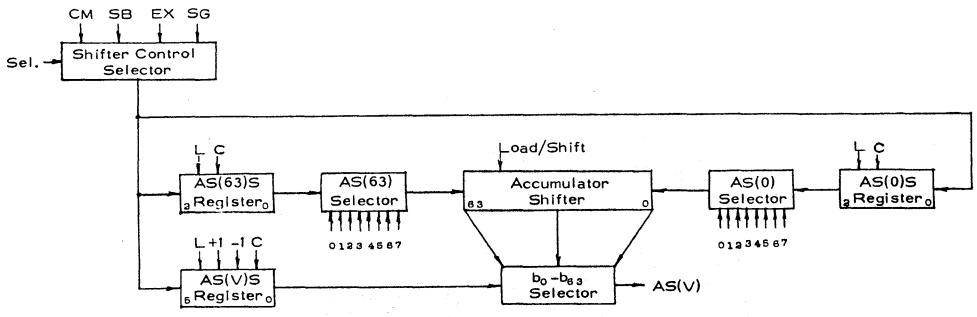


Local Registers, LR

Figure 8.

The Accumulator Shifter, AS

The other input to ALU is the Accumulator Shifter, AS, see Figure 9. The AS is a register which can shift one position left or right in one operation. Any 1 of 8 possible specified sources can be loaded into the vacated bit position. One of these inputs may be an arbitrary selected bit of AS itself, thus allowing AS to act as a cyclic shifter of an arbitrary length n, $1 \leq n \leq 64$. AS has no direct path to the bus but must gate its contents through the ALU.



Source no.	AS(63) Input	AS(0) Input
0	0	0
1	1	1
2	AS(0)	AS(63)
3	AS(63)	BUS(63)
4	CR	SB(63)
5	DS(V+1)	DS(V+1)
6	AS(V)	AS(V)
7	VS(V)	VS(V)

Accumulator Shifter, AS

Figure 9.

The Variable Width Shifter, VS

The variable width shifter, VS, is logically identical to the AS, but has direct output to the bus. Together with the AS, it can form a cyclic shifter of arbitrary length n, $1 \leq n \leq 128$.

The Double Shifter, DS

The Double Shifter, DS, is also logically identical to AS, except for the fact that it shifts two positions at a time. The two variably addressed bit positions (besides from being used as input to vacated bits in shifts) may be used to load LRIP and LROP as shown in Figure 8. Thus the 64-bit content of DS may be used in 32 steps to control algorithms two bits in one step (e.g. in a multiplication algorithm) by table look-up in LR. The DS, AS, and VS can be interconnected so that two 32-bit words can be merged (interlaced) together and so that one 64-bit word can be decomposed in the reverse fashion.

The Status Port, SP

This port provides a path between various control and microinstruction sequencing facilities and the MDP. Its functioning is explained in section 2.3.5.

The Input Ports IA and IB

Each of the ports can potentially access 16 different input devices, addressed by associated device address-registers IAD and IAB.

The Output Ports OA and OB
Similarly to the input ports, OA and OB can access 16 devices, although ports may be dedicated to a single device in the case that a device is expected to be very heavily used. As an example a standard memory will be considered as such a device, where an input port and an output port may be dedicated.

The Output Ports, OC and OD
These are identical to OA and OB, except for the fact that they are loaded from the BUS and not as normal destinations of the SB, but loading is specified by microoperations.

2.3 Control facilities

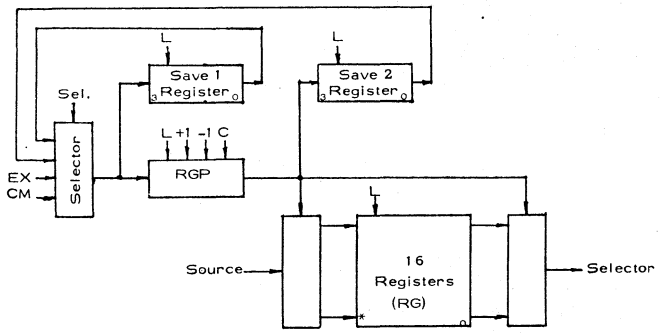
As it can be seen from the previous description of the MDP and its associated storage and functional elements, most of these resources require some sort of control-information. It may be an address (e.g. for WA and WB), data for a functional unit (e.g. a shift specification for BS or mask specification for PG), a functional specification (e.g. control of ALU function) or a selector-specification (e.g. for vacated bit input to a shifter).

2.3.1 The standard groups
As one of the main objectives of the design was to offer a host upon which a wide variety of experiments could be conducted, it was decided to delay the binding of as many design decisions as possible and to give the user capability to bind some decisions. With respect to the specification of control information for all functional units, the user is able to choose between immediate control and some form of residual control.

Each resource of the system can be controlled from a field within any particular microinstruction, thus allowing the immediate control source. Another such control source might be some storage element that could have been set up by a particular microinstruction to be used by a later microinstruction, i.e., residual control. Possible residual control sources might be from the external world (e.g. another computer), the data carried on the bus, or a result of some computation thereon. In MATHILDA, control information may be selected among four possible sources, among which CM (the current microinstruction), EX (external register), and the output of a RG (a register group) form three inputs to a control selector. The fourth input is either SB (least significant bits of the shifted bus) or the output of BE (the bit-encoder, see section 2.3.3), the choice of which is bound by the designers depending on the needs of the unit in question.

The selection of the control source is normally specified in the very same microinstruction which specifies the use of a particular resource. The control information normally is buffered (e.g. as in the ALU-function buffer, ALF, a working register address pointer, or in a resource itself as in the case with system counters). Only where the control information is directly affecting the bus transport, as with BS and PG, was it necessary to bind the selection of source prior to the use of the information. This is obviously because the selected information is not being buffered, but directly affects the bus transport when the unit is enabled.

The amount of residual control facilities is large and could have been rather complex. However, a great deal of simplicity and modularity has been achieved both logically and physically by use of a uniform residual control concept, the standard group and its selector, see Figure 10.



* The width of the registers depends on the particular selector involved.

Typical Standard Group

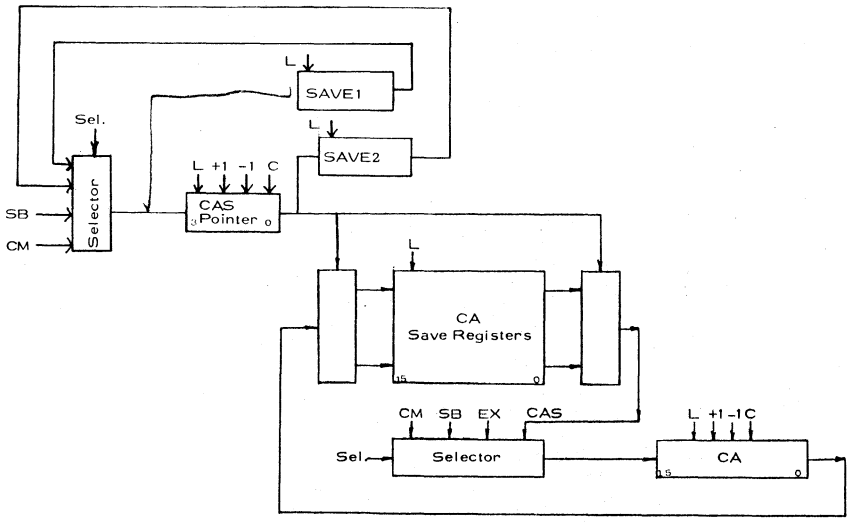
Figure 10.

A Standard Group is a storage element of 16 words with an address mechanism. The local storage associated with a standard group is called a Register Group, RG, and is of an appropriate width depending on which unit the standard group is associated with. The storage is used for residual control, and in these cases where the information residing there is part of a fixed environment, the loading of the storage takes place from the SB. However, in connection with some units, it was more natural to load the storage from the unit itself, so that the information there could be saved and later restored. Loading a standard group from the SB then becomes a two-step proces: first load the unit from SB, and then "save it".

The reading of and writing into the storage of a register group is controlled by an address or pointer contained in a register, the RGP which can be loaded, cleared, incremented or decremented. Loading of a RGP can also take place from four sources, two of which are local to the register group, the Save-1 and Save-2 registers. The timing is so that it is possible in one microoperation to save the contents of a RGP in its Save-2 (S2)-register, and load new data into RGP (which might be the contents of its S1-register).

2.3.2 System counters CA and CB

Among the control-information units are also two 16-bit system counters CA and CB, which can be used in controlling algorithms, e.g. for counting in loops, etc., see Figure 11. It should be noted in Figure 11 that the content of a counter can be saved and later restored, so that one counter can control up to 16 levels of embedded loops.



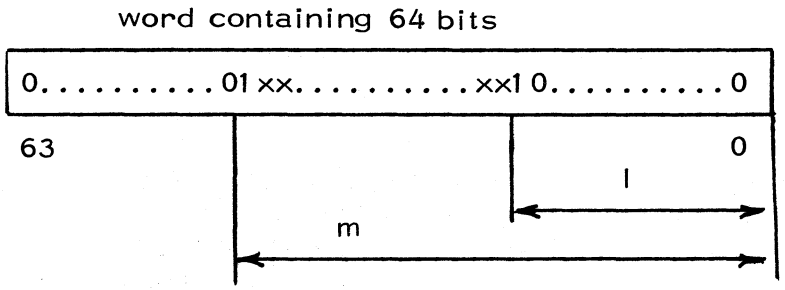
Counter A, CA
Figure 11.

The counter itself can be incremented, decremented and cleared, and the actual content of the counter can be tested towards zero, and some of the bits can be tested individually.

The only difference between CA and CB is that where the sources for loading CA are: CM, EX, SB, and CA-save-group, the similar sources for CB are CM, EX, BE, and CB-save-group (BE is the output of the bit-encoder, described below).

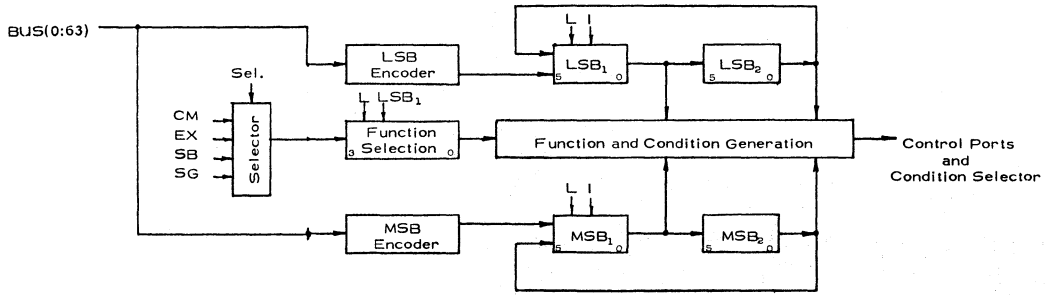
2.3.3 The Bit-encoder BE

One resource of the MATHILDA is to our knowledge a new invention. It is what we call the Bit-encoder, BE. In many algorithms it would be useful to have easy access to information about a given bit-pattern as to where is the first bit on, and where is the last:



We call the quantities l and m respectively LSB (least significant bit) and MSB (most significant bit). The bit-encoder can provide the user with such numbers, and certain computations using LSB and MSB.

The Bit-encoder is continuously encoding the information on the BUS, yielding the two quantities l and m corresponding to the transported bit-pattern. By a microoperation these can be loaded into two registers LSB_1 and MSB_1. The computational network of the Bit-encoder further contains two additional registers LSB_2 and MSB_2, which can be loaded from LSB_1 and MSB_1 respectively or be interchanged with these. Assuming LSB_2 and MSB_2 contain the encodings from a previous BUS-transport, the circuitry of the Bit-encoder continuously computes the following quantities:



	BE Functions F and G
F	LSB_1
	$LSB_1 - 1$
	MSB_1
	$MSB_1 + 1$
	L_1
	$AL = L_1 - L_2$
	$LSB_2 - LSB_1$
	$MSB_2 - MSB_1$
G	$\left[\frac{F}{2}\right] + 1$
	[ ] ::= integer part of

$L_i = MSB_i - LSB_i$, i = 1, 2

The Bit Encoder and its associated functions

Figure 12.

The output of the BE may be used in various control elements of the system. Furthermore the Bit-encoder always provides the following testable conditions:

Conditions

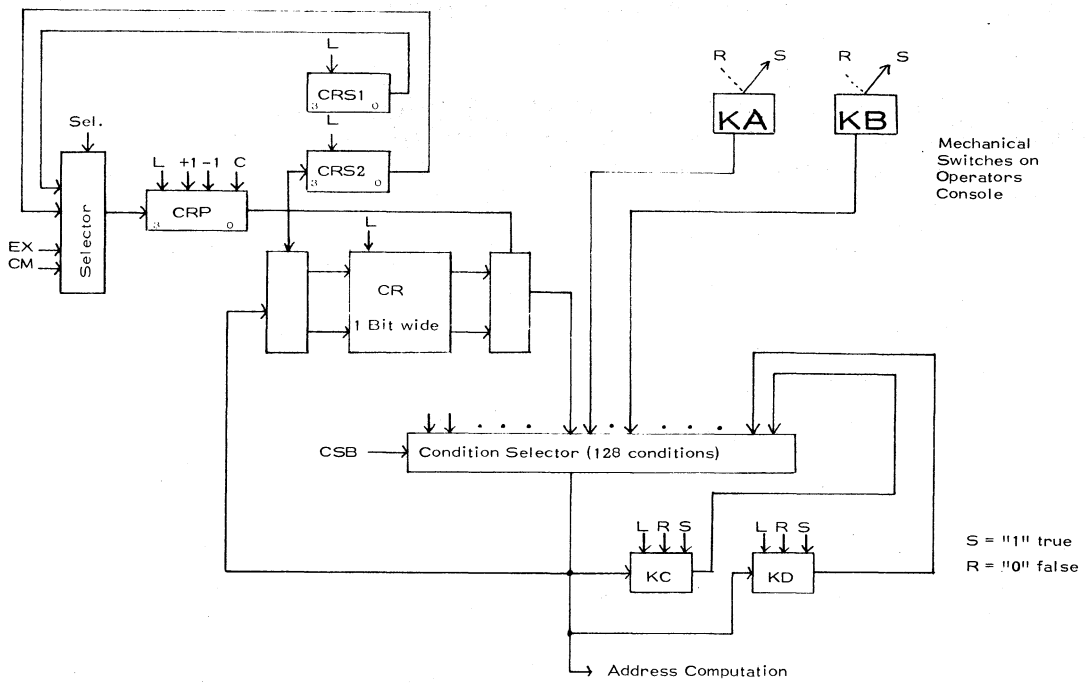
$LSB_1 \equiv 0$
$MSB_1 \equiv 63$
$L_1 \equiv 0$ (i.e., $LSB_1 = MSB_1$)
$L_2 \equiv 0$ (i.e., $LSB_2 = MSB_2$)
$L_1 = L_2$
sign $(L_2 - L_1)$
$LSB_1 = LSB_2$
sign $(LSB_2 - LSB_1)$
$MSB_1 = MSB_2$
sign $(MSB_2 - MSB_1)$
$BUS \equiv 0$

Based on these conditions, it is possible to compare some of the characteristics of the two bit-patterns whose encodings have been loaded to: a) direct decisions about the algorithm, b) choose BE-function, or c) interchange (LSB_1, MSB_1) and (LSB_2, MSB_2) before selecting the BE-function and use the selected information.

Since bit-patterns and bit-matrices play an intensive role in many non-numerical algorithms, fundamental operations providing the encodings l and m, may prove to be useful on the virtual machine-level, and also high-level languages.

2.3.4 Condition save registers, CR, and switches, KA, KB, KC and KD

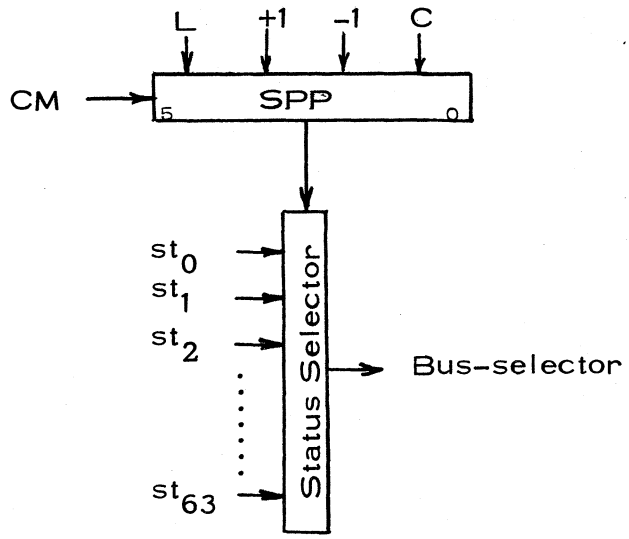
A number of system conditions (almost 128) are continuously sampled and in every instruction any one of these may be selected for testing. The main use of such a selected condition is of course in the sequencing which will be described later (S. 2.4.4), but it may be useful to be able to save a condition as it arises to be used in a later sequencing decision. The Condition save registers (CR) are a 1-bit wide (save register) group where the value of a condition may be saved and later retrieved, see Figure 13. Switches exist in two variants: KA and KB are console-switches, KC and KD are programmable switches that furthermore also can be loaded with the selected condition.



Condition Save Registers and Switches.
Figure 13.

2.3.5 The Status Facility

The Status Facility establishes a data path between various control registers, address pointers, functional units (e.g. counters, and the bit-encoder) and the MDP. The Status Facility provides a basis for gathering both data concerning the operation of the system and data for use in algorithmic processes.



Status Facility

Figure 14

64 different sources of data are fed into a status selector. Which particular source is to be gated onto the MDP as the bus-input is determined by the content of the Status Port Pointer (SPP) register. It should be noted here that st_0 is a 16-bit field from a microinstruction so that constants can be put on the MDP with relative ease.

2.3.6 The Snooper Facility

The Snooper Facility consists of a) a Snooper Control Store and b) Snooper Resources (e.g. 2 groups of 16-registers, counters, and comparators). The Snooper unit works in the following way: when the address of the next microinstruction to be executed is sent to the MATHILDA Control Store address buffer, it is also gated into the Snooper Control Store address buffer; at the same time the microinstruction is fetched so that it can be executed, the contents of its associated Snooper Control Store location is fetched; in parallel with the microinstruction being executed, the contents of its associated Snooper Control Store just fetched is used to control the operation of the Snooper Resources. Snooper Control Store (80 nanosecond storage) is 16-bit wide and has the same number of words as the MATHILDA Control Store. A snooper word can specify, for example, any two registers which can be counted up (or down). The Snooper Facilities can be written or read through the normal input/output ports of the system in much the same way as the Status Fa-

cilities. Snooper Control Store is writable so that different data gathering routines can be associated with the same segment of microcode without changing the microcode. The user is allowed to establish the correspondence between any particular snooper resource and the routine upon which it is snooping.

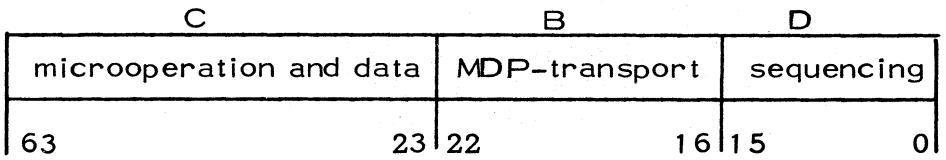
2.4 Microinstruction sequencing and execution

One microinstruction execution of the machine may be considered as consisting of four major sequentially executed steps:

A: Microinstruction fetch
B: Data transport on MDP
C: Execution of microoperations
D: Address calculation (for the next microinstruction)

Steps B, C, and D are controlled by fields within the 64 bit wide microinstruction. Logically, to the user, these steps and substeps within these may be considered sequentially within the microinstruction execution, although of course many of the activities take place in parallel in the physical implementation. The execution of one microinstruction is to be considered as totally completed before the next microinstruction is executed, i.e., actions initiated by the execution of a particular microoperation do not span several microinstruction executions.

The microinstruction consists of three major fields, corresponding to the control of the previously mentioned steps B, C, and D:



The dealing of dynamic conditions has been made consistent. The machine can be run in two modes, under program control:

Long cycle: Any condition arising as an effect of the execution of steps A, B, C can be tested and used for sequencing in step D.

Short cycle: All conditions used in the instruction is of the state of the machine immediately prior to step A.

2.4.1 Microinstruction fetch (step A)

The control store may consist of up to 4096 words of 64-bits (80 nS). Initially, 512 words of control store has been implemented. It is writeable under program-control from an output port of the system itself, or from a 16-word deadstart PROM. The control store is addressed by the contents of an address buffer, which is normally determined by the sequencing part of the previously executed microinstruction. [However, after either deadstart or an interrupt, the control store address buffer is forced to contain a zero.]

2.4.2 MDP-transport (step B)

The specification of the MDP transport is given in this field. The source of the transport, which is the input to the Bus Selector and latch (see Figure 2), is specified by 3-bits as is the destination of the SB. 1-bit is used to enable the BS and, if necessary, data is supplied from the C-field.

The sub-steps of step B are:

B_1: Selection of source
B_2: Masking by MB
B_3: Buffering in the BUS-latch
B_4: Shifting by BS if enabled
B_5: Masking by PM $\vee$ PG
B_6: Buffering in the SB-latch
B_7: Loading into a selected MDP destination

2.4.3 Execution of microoperations (step C)

The C-field is divided as follows:

Data and mops		Shifter control	
63	29	28	23

The "shifter control" field contains three 2-bit fields associated with AS, VS, and DS. Each 2-bit fields specifies: shift left, shift right, load or "do nothing", for each shifter respectively. The "data and mops" field is again subdivided into several fields as follows:

F1	S1	M2	F2	M3	F3	S3	M4	F4

Fields F1, F2, F3, and F4 are each 7 bits wide, fields S1 and S2 are 2 bits wide, and fields M2, M3, and M4 are 1-bit fields. The mode-bits M2, M3, and M4 determines whether F2, F3, and F4 are to be decoded as microoperations ("mops"), or to be used as data. F1 is always decoded as a "mop", so up to four "mops" in addition to the AS, VS, and DS control may be specified in one instruction. Many places in the system may be supplied with data from the fields F2, F3, and F4. Most often the data is supplied through a selector, which also needs control. Furthermore, the loading of the data into its destination is specified by a "mop". The standard "set-up" for such a load is to specify the load-mop in say F1, the selector-setting in S1, and if the setting requires data from CM, this is taken from F2, i.e., M2 is to be set to disable the decoding of F2 as a mop. Similarly, F3, S3, M4, and F4 may be used together to control some resources.

*) The MDP destinations AS, VS, and DS have their own separate load/shift control bits in the C field.

Data requirements are normally 4-bit or 6-bit wide and are usually taken from F2 and F4. PG requires 7-bits (F2), and CA and CB need 16 bits which are supplied from the fields F2, F3, and S3 concatenated, acting as an "expanded F2" field. F3 is used for BS shift-specification, when the BSS specifies CM as the data source, and BS is being enabled. Furthermore data from F3 and F4 may be used in sequencing (step D) for addressing computation.

When F1, F2, F3, or F4 are being decoded as mops, they can specify a variety of actions around the whole system. The decoding allows for 512 different interpretations, some are only duplications of the same action, so that some space-conflicts among mops and data can be avoided.

Each mop has a clock-specification, dividing the set of mops into two classes, clock 1-mops and clock 2-mops. The clocks are such that the execution of a clock 1-mop is completely finished before clock 2-mops are being initiated.

The execution of operations in step C can be thought of as being performed in the following substeps:

C1: Gate the information from S-fields and from F2, F3, F4 fields to their destinations irrespective of their expected or non-expected use.

C2: Decode the enabled F-fields depending on the enabling by the respective M fields.

C3: Execute (clock) the specified clock 1-mops

C4: Execute the specified load/shift actions in AS, VS, and DS.

C5: Execute (clock) the specified clock 2-mops

The class division of mops with respect to clocking, allows as an example for the loading of data into a particular RG, and then changing that particular RG's address pointer within the same microinstruction.

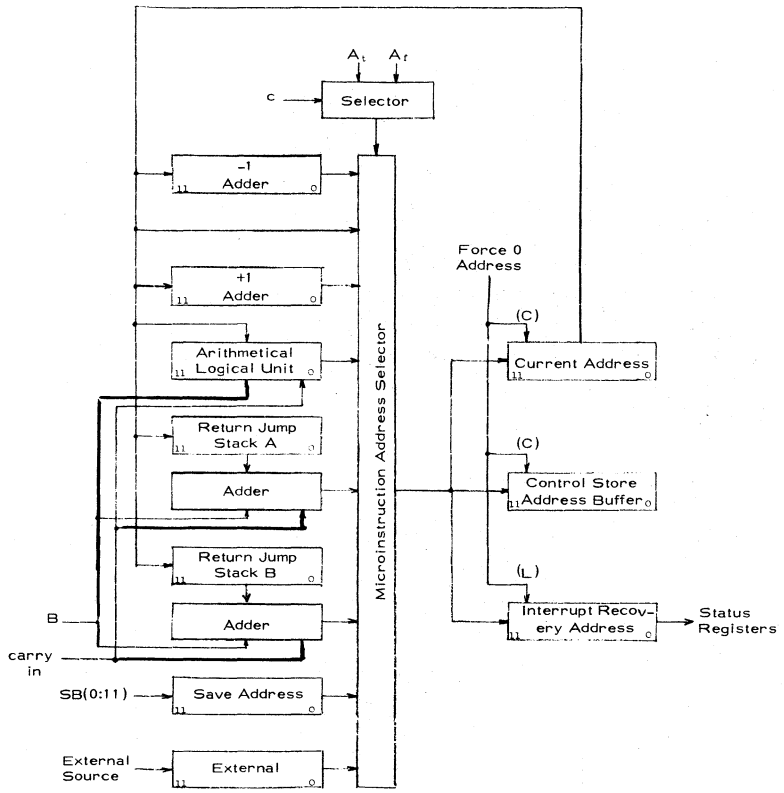
2.4.4 Address-calculation (step D)

The determination of the address of the next microinstruction to be executed is an implementation of the well known if-then-else construction. The selection is among two modes of address-calculations (rather than addresses themselves). Possible modes of address-calculations (as well as the way the if-then-else clause is realized by the conditional use of the A_{true} or the A_{false}) are illustrated in Figure 15.

The D-field of the microinstruction consists of the following subfields:

BISB	CISB	Condition Selection	A_{false} selection	A_{true} selection

where A_{true} and A_{false} are 3-bits, Condition selection is 7-bits, BISB is 2-bits, and CISB is a 1-bit field. The particular condition which is selected by the 7 CISB bits is denoted by "c" in Figure 15.



Microinstruction Address Bus.

Figure 15.

Addressing thus allows straightforward transfer of control to neighbour instructions (A-1, and A+1, where A is the address of the current instruction). The arithemtical-logical unit, CUAL, provides an address which is computed from A and B (address-constants as described below); such computed addresses provide, for example, relative and absolute addressing. Furthermore the current address may be pushed onto 1 of 2 return jump stacks (RA and RB) each of which is 16 levels deep. The top of either stack may be used in later steps through adders. The RA and RB stacks are automatically popped when they are being selected. Finally the Save address-buffer (SA) provides a path between the Shifted Bus and the address-bus, and the external buffer (EX) provides a path from an external source. A force-zero possibility exists, it is used on deadstart and for "hard" interrupts.

The B-input, which can be used in various address computations, is specified to be one of the following four sources, determined by the BISB-field:

0:	A constant zero
t^{l}:	Sign-extended version of t (t:6 bits from F4 of C-field)
T∘t:	T concatenated with t (T:6 bits from F3 of C-field)
0∘SA(0:5):	The zero concatenated with the least significant part of the save-address buffer.

The CISB-bit is used in specification of the carry-input to the address (+1) returns without need for non-zero B-input, and jumps to one of two consecutive locations depending on a condition.

The execution of step D consists of the following logical substeps:

D_1: Choose the selected condition, c. [In short cycle c is the value of the selected condition prior to step A, in long cycle the new value.]

D_2: Select the carry-in and B-input into the return jump stack adders and the CUAL.

D_3: Compute the results of the return jump stack addition and the CUAL function.

D_4: Select the new address, using A_t if c = 1 or A_f if c = 0 as the bus-selection, and load the address-buffers.

D_5: If RA and RB has been selected then pop the stack that was used.

D_6: If a force-zero situation has occurred then load the interrupt recovery address, and clear the address-buffers.

3. CONCLUSIONS AND EXPERIENCE

3.1 Design and hardware considerations

The design of MATHILDA as an experimental tool for research in emulation was partially initiated based on design experience gained earlier by another group in the department on the construction of the RIKKE-0 machine. It was felt at the time when the MATHILDA design process was started, that no commercially marketed processor was available within the funding we could afford (or apply for), when justified as an initially pure research processor.

As a department in a non-engineering university, our hardware-staff was very restricted (2 engineers + 1 technical assistant). This staff was mainly intended for minor construction and interfacings, together with service on all standard equipment in the department. The design was originally intended to be for a functional unit, very intimately connected to the controlling RIKKE-0. However, during the design process a number of complex facilities were added. Thus the functional unit grew into a selfcontained processor and, of course, our original time schedule for the construction was not satisfied. Besides the growth in design, unexpected other duties of the technicians, delivery problems, lack of project management experience, and the fact that the project was, in retrospect, a bit overambitious for the staff, delayed the project. It was, however, decided to test the design on the 16-bit RIKKE version before the construction of the 64-bit MATHILDA.

The basic design, including the bus structure and the concepts of standard groups bit encoder etc.) was made by the authors in the period from February to August, 1972. The sequencing part and instruction format was designed in the fall, while the printboards for the bus-structure were laid out and actual mounting of RIKKE started. By August, 1973, the control unit and control facilities (register groups) were ready for initial hardware testing. In late October, the 16-bit bus structure was added for testing, and in January 1974 initial testing of the RIKKE processor was considered completed. However no main memory was added before March due again to delivery problems. At the time of this writing (June 1974), the RIKKE has interfaced to it a high speed paper tape reader and a small mini-printer. The RIKKE machine is complete as the MATHILDA machine described in this document except that the following facilities are not yet implemented: Snoopers, Status, Bit-Encoder, and the Force-0 address and IRA facility.

The construction of MATHILDA was started in August, 1973, but since testing of the MDP (and the print boards) was not done before January 1974, the "go ahead" for the construction of the 64-bit bus structure could not be given before then. Here again delivery problems for the print boards caused a further two-month delay. Currently, the mounting of the bus structure is halfway through.

The cost of the construction is not easily calculated. An estimated cost of $ 12,000 for components and $ 10,000 for mounting assistance was granted by the Danish Research Council. The support from the staff-technicians cannot be computed that simply because of their other duties and projects. An estimate of 2 man-years of design and documentation from the technicians might be adequate. The experience gained in the department during this process cannot be underestimated, we learned a lot about what to do, and especially what not to do in such a project.

3.2 Experience with programming

Since the summer 73 as assembler, MARIA [4], and a simulator of the system have been in use. Four simulators for RIKKE [5] and MATHILDA [6,7] have been written in addition to emulators for an O-code machine for BCPL [8] and a P-code machine for Pascal [9]. Furthermore, basic software (bootstrap loader, normalizer, etc.) and a large number of routines for the implementation of arithmetics has been implemented. Experience with the programming of the processors shows that the design seems to be suitable for experimental purposes, although coding is not so straightforward because of the horizontal nature of the microinstructions as on processors with more highly encoded (vertical) but more primitive instruction formats (e.g. the B1700). Certain facilities of the system has proven to be extremely useful, the easy and natural sequencing possibilities, the BS, BM, PG, and especially the BE.

Although the design of MATHILDA may seem somewhat complicated, experience from a course on computer architecture given at the University of Southwestern Louisiana indicates that the students learned the MATHILDA design with reasonable ease. We are at present only in the very beginning of the real use of the RIKKE and MATHILDA processors, and only in the future can real evaluations of the suitability be made.

3.3 Acknowledgements

The authors want to express their thanks to the Danish Research Council, who granted the construction costs and the stay of one of the authors during the design phase. Also thanks are directed towards NATO, Division of Scientific Affairs, which is supporting further collaboration between the authors and their present institutions. Finally thanks are expressed to our colleagues for their comments and advice, to the students who helped us with supportive work and software development, and to the technicians who patiently attended numerous discussions, accepted changes and additions from the authors during the design process.

Appendix A.

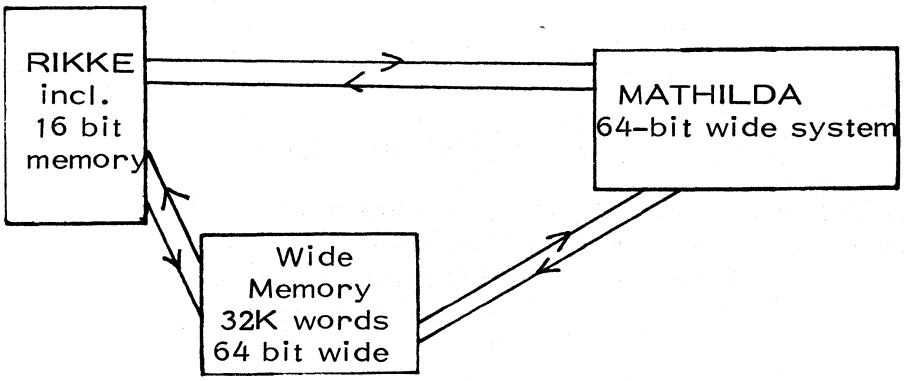
GLOBAL SYSTEM DESIGN.

A. 1 The MATHILDA-RIKKE-Wide Memory system.

Although the organization of the MATHILDA processor is the key issue of this report, and the use of it is in no essential way constrained to the specific environment it is intended to operate within, it seems reasonable to explain the actual proposed system.

The fact that MATHILDA is to be treated by RIKKE, as an I/O device, as shown in Figure 16, offers a great flexibility and proposed some interesting projects. A particular problem associated with such an interconnection was whether synchronisation was necessary or not. It was decided that, except for a few control signals, every information interchange should be queued up so that asynchronous operation was possible. The same principle was also used with respect to a 64-bit Wide Memory, this being a common resource to the system, and not dedicated to MATHILDA.

In this way the system was designed to be three independent units operating asynchronously and communicating through queues:



The MATHILDA-RIKKE-Wide Memory system.

Figure 16.

The Wide Memory is exclusively controlled by RIKKE, i. e. it is the only unit that can deliver requests for memory access. The data transfer can take place on a number of memory ports. RIKKE itself will be attached to a set of I/O-ports of the memory system. RIKKE also has its own private memory (up to 64K 16-bit words). Standard I/O-devices are therefore intended to be coupled either directly to RIKKE or to other minicomputers communicating with it.

The idea of operation is to let RIKKE decode virtual machine instructions, and to do all address calculations involved, while MATHILDA was to perform the actual data transformation required. The philosophy of the design of MATHILDA was to give it capabilities for doing transformations upon a wide data word. Complicated "macro" routines could be implemented in MATHILDA microcode and thereby define a "Nano-machine" which can be called upon from the microcode in RIKKE to define the complete virtual machine. In fact it is not a nano-machine in the QM-1 sense [10], but it may be operated in such a way that a similar effect

is achieved. Among the differences in these approaches is that RIKKE, running in parallel with MATHILDA, normally will be ahead of it, preparing instructions and operands for it.

A. 2 Projects related to the MATHILDA-RIKKE-Wide Memory system.

The cooperation of RIKKE-MATHILDA-Wide Memory in a synchronous or asynchronous fashion seems to offer fundamental questions similar to those posed in other recent types of computer architecture, like the Illiac-IV or the CDC-Star computer systems. We are undertaking a wide range of projects both at the University of Aarhus in Denmark and the University of Southwestern Louisiana in Lafayette, Louisiana which are related to this system.

A. 2. 1 Projects within the computer systems and language area.
A major topic in the system area is the construction of distributed systems architecture, that is, for example to evaluate operating systems and compilers for computer complexes with more than one processor. Notice here that the present system differs in a fundamental way from standard multi-unit, MATHILDA is a resource to a RIKKE, or a system of RIKKE's.

It is, of course, not at all obvious what the architecture of the best machine should be in order to assist in the efficient construction of various complex systems and architectures. We intend to investigate the possibility of formalizing the capability of various host architectures, to identify what primitives are needed and can be implemented in microcode to support compilers, high level languages and operating systems. Thus MATHILDA is a tool in such research and not the answer to the questions which are raised.

An immediate project is the direct emulation of different virtual machines, either standard computer systems or given experimental machines e. g. the BCPL language running on emulated virtual 0-code machine [8]. Such an emulator has been implemented on the standalone RIKKE, this allowing a simple transfer of already accessible software in BCPL to be available on the system, (e. g. [11]).

The RIKKE-MATHILDA-Wide Memory system offers a complex tool for handling of composite data structures. Most computer systems today operate within a multilevel store environment. A basic question within the field of data structures concerns how the distribution of different primitives in a program to the appropriate kind of store and processors is to be made. As an example, the handling of list structures may be performed in RIKKE, while MATHILDA is doing the computations at the nodes. Also in the support of special peripheral devices on RIKKE, e. g. a graphical display or an electrostatic plotter, there is a demand for high speed access to large and complex data structures.

A core project in the systems area is to define and implement a micromonitor, resident in RIKKE control store, to allocate the resources of the system, I/O as well as processors, and which will allow multiprogramming among emulators, [12].

Another project concerns the implementation of the language Pascal [13] to support the immediate applications. The basic tool in this and other language implementations is intended to be a compiler-generator using LR(k) techniques (the BOBS - system[14]), which has already been implemented at Aarhus, and being transferred to the RIKKE-MATHILDA system.

The target system for the Pascal-compiler is a stack-machine (called P-code machine [9]), running in such a way that evaluation stacks exists in RIKKE as well as in MATHILDA.

The implementation of Pascal is considered as the first step in an experiment with developing a language for numerical algorithms in non-standard arithmetic. The goal of this project is to define a language where several real-datatypes can co-exist and be controlled. This will allow the experimentation with various "virtual arithmetic units", [15], for non-standard arithmetic to be performed on MATHILDA, controlled by RIKKE. There are also projects being undertaken which deal with an integrated approach to fault tolerant virtual systems and their performance measurement.

A. 2. 2 Projects in numerical analysis.

We are currently undertaking projects where MATHILDA could be applied in making non-standard arithmetic available. An outline of such projects may be found in [16]. Many subroutines in both high-or low-level languages have been written to allow the use of extended range, extended precision, significant digit, unnormalized interval, rational and complex arithmetic. Special purpose hardware is much too expensive, but the general structure and microprogrammability of MATHILDA certainly will offer more efficient implementation of the arithmetic primitives. The overall structure of the system will allow extensive experiments on various arithmetics, by changing underlying structure.

A key question in the study and implementation of machine arithmetics as on higher levels will be to extract the fundamental "core" of the operations, i. e. to determine a fundamental set of operations on different bases in such a way that a structured (layered) development of any particular arithmetic will follow.

This approach will be combined with the "top-down" analysis from the application and language point of view. The idea being to move step-wise the border between the implemented virtual machine towards the higher level machine. The projects within the systems area will fit those in the numerical analysis part, and provide fundamental software needed for these projects.

As one of the major problems in the application of digital computers in numerical analysis, is representation of numbers, and the operations upon them, the most immediate project will be to conduct theoretical and experimental studies in machine arithmetic.

Appendix B:

Physical summary of MATHILDA.

MATHILDA is mounted on 7 frames 8 × 45 × 60 cm, each of which is turnable around a vertical axis like pages in a book. Each frame is closed from both sides with removable boards of plexiglas, thus forming a closed box. Each "box" is equipped with three small ventilators blowing a stream of air up through the frame. Signal-interconnections between the frames are through standard cables containing 20 signal-ground wound pairs of wires. Plugs are mounted along the vertical sides of the frames.
The printboards are two-sided and are either special prints or standard prints only containing power and ground where signal-interconnections are in the wiring. Special purpose prints exists only in three variants:
a) 8-bit wide of the whole MDP, 25 × 40 cm,
b) 4 × (4 bit of a standard register group), 15 × 40 cm,
c) 256 words of control-store, 64-bit wide, 15 × 40 cm.
All of the special purpose printboards furthermore contain some room for additional circuitry.

No attempt has been made to carry signals to the edges of the boards for board-to-board and board-to-plug interconnections, all such connections are made with wires from the proper places on the prints.

All circuits are from the TI 74 series or equivalent. A large amount of signals (data buffers) have been made visible on the boards by light-emitting diodes (for diagnosis and step-mode).

A survey of the content on the frames are given below:

Frame 0: (Sequencing and Control Store)
1 pc 30 × 40 standard print containing sequencing, clock-generators and deadstart. Masterclock (40 nS steps) and its input into a shift register which pulses various clocks. At present there are 9 steps in one (short) cycle, but 7 steps should be the ultimate (short) cycle.
2 pc type c (above) prints control store, 128 pc TI 74200 + amplifiers.

Frame 1: (CA, CB, WAP, WBP, LA, LB)
4pc type b prints (standard-groups).

Frame 2: (Standard groups for shifters, PG, BS, BM, PM, AL, etc.)
3pc type b prints + 1 pc 15 × 40 standard print.

Frame 3: (End-connection for bus modules, BE + various).
1 pc type b print + standard print.

Frame 4-7: (Each contains 16 bits of MDP with all registers and ALU)
2 pc type a prints + 1 pc 10 × 40 standard print on each frame.

Power supply: 8 pc, 5V, max 15 amp.

Console: (preliminary)
Buttons: Deadstart, Run, Stop after deadstart, Step-Stop (two pushes pr. cycle, first instruction fetch, second execution).
Switches: KA, KB conditions, 2 stop-switches (stop on execution of specific mops, i. e. not as testable condition).

Deadstart: 16 words of battery-driven CMOS-PROM is copied into control store repeatedly. Execution is forced to location zero.

References.

[1] Madsen, Ole B., "BPL-A hardware and software description language", RECAU Report, University of Aarhus, Aarhus, Denmark, 1972.

[2] Shriver, B.D., "A description of the MATHILDA System", Department of Computer Science Report PB-13, University of Aarhus, Aarhus, Denmark, April 1973.

[3] Flynn, M., and Rosin, R.F., "Microprogramming an introduction and viewpoint", IEEE TC, C-20, No.7, 727-731, July, 1971.

[4] Sørensen, I.H., "The extended simulator for RIKKE", Department of Computer Science Internal Document, University of Aarhus, Aarhus, Denmark, June, 1974.

[5] Lynning, E., Kressel, E., Anderson, H.O.S., and Sørensen, I.H., "A users manual for the simulated RIKKE-MATHILDA system on the CDC-6400", Department of Computer Science Report, University of Aarhus, Aarhus, Denmark, 1974.

[6] Caillouet, P. and Landry, S., "A MATHILDA simulator written in SNOBOL", Department of Computer Science, University of Southwestern Louisiana, Lafayette, Louisiana, Jan. 1974.

[7] Bullard, S., Caillouet, P., Landry, S., and Pye, J, "A users manual for the simulated MATHILDA machine on the Univac 70/46 G", Department of Computer Science, University of Southwestern Louisiana, Lafayette, Louisiana, May, 1974.

[8] Sørensen, O., "The emulated 0-code machine for the support of BCPL", Department of Computer Science Document, University of Aarhus, Aarhus, Denmark, to appear.

[9] Kristensen, B.B., Madsen, O.L., and Jensen, B.B., "A PASCAL environment machine (P-code)", Department of Computer Science, Report PB-28, University of Aarhus, Aarhus, Denmark, April, 1974.

[10] Rosin, R.F., Frieder, G., and Eckhouse, R., "An environment for research in microprogramming and emulation", CACM, 15, No. 8, 197-212, August, 1972.

[11] Strachey, C., and Stoy, J, "The text of OSPub", Oxford University Computing Laboratory, Programming Research Group, Oxford, England, 1972.

[12] Rosin, R.F., "Proposal for a nucleus I/O system", Department of Computer Science Report PB-23, University of Aarhus, Aarhus, Denmark, January, 1974.

[13] Wirth, N., "The programming language PASCAL", Acta Informatica, 1, No. 1, 35-63, 1971.

[14] Jensen, B.B., Madsen, O.L., Christensen, B.B., and Eriksen, S.H., "A short description of a translator writing system (BOBS-system)", Department of Computer Science Report PB-11, University of Aarhus, Aarhus, Denmark, February, 1973.

[15] Podlaska-Lando, S. "A proposed implementation scheme for the partial realization of integer floating-point arithmetics on MATHILDA", Department of Computer Science Internal Document, University of Aarhus, Aarhus, Denmark, October, 1973.

[16] Shriver, B.D., " A small group of research projects in machine design for scientific computation", Department of Computer Science Report PB-14, University of Aarhus, Aarhus, Denmark, April, 1973.

Kornerup, Peter.
An overview of the MATHILDA system / by Peter Kornerup and Bruce D. Shriver.-- Aarhus, Denmark: Department of Computer Science, Institute of Mathematics, University of Aarhus, 1974.
(DAIMI; PB-34)

I. Joint auth or. II. Title.