A DESCRIPTION OF THE RIKKE 1 SYSTEM

by

Jørgen Staunstrup

DAIMI PB-29

May 1974

Institute of Mathematics University of Aarhus

DEPARTMENT OF COMPUTER SCIENCE

Ny Munkegade - 8000 Aarhus C - Denmark

Phone 06-12 83 55

Contents

Foreword

It is the purpose of this document to give an introductory (yet reasonably detailed) description of the RIKKE 1 System. The bus structure, the registers and functional units attached to it, and the control which can be exercised on these components are discussed. The document is not a reference manual. Rather, it is written entirely from the pedagogical point of view, with the system described in a modular fashion. Examples are introduced after each component is added to the basic bus structure. The examples are written in the RIKKE 1 microassembly language (see [8]). The examples are deliberately kept simple so the reader will not spend time learning a complicated or clever algorithm but will learn the control mechanisms of the particular components involved. Thus, many of the examples are "contrived" and do not perform any particular "useful" data transformations. It is hoped that this approach enhances the reader's understanding and underscores the overall simplicity and homogeneity of the structure and its components.

The present description is a modification of a similar one, describing another slightly different system called MATHILDA (DAIMI PB-13), written by Bruce D. Shriver.

For more detailed information the reader is referred to [9].

A Description of the RIKKE 1 System

by

Jørgen Staunstrup

1.0. Introduction

RIKKE 1 is a dynamically microprogrammable processor which has been designed to be used as a tool in emulation-oriented and processor design research. For the sake of completeness we will discuss briefly a short history of the unit and then some of the criteria which served as a basis for its design.

1.1. Historical Notes

In the spring of 1971 the Department of Computer Science of the University of Aarhus was considering the purchase of a standard minicomputer to act as a controller for a variety of peripherals and to simulate a medium speed batch terminal to the Computer Center's large system. A group of people were, at this time, working on the design of an integrated software and hardware description language called BPL [1]. To support this group and to make the use of such a minicomputer more flexible, it was decided to design and construct a microprogrammable minicomputer within the department itself.

The design was started and completed during the summer of 1971. The resulting machine, RIKKE 0 [2], was constructed and began running in early 1972. In the meantime a number of projects were proposed which were considered not to be compatible with that design. Among these were various projects in numerical analysis [3, 4] in which it was found that the word size and bus width of the RIKKE 0 (16 bit) was too short to obtain an efficient implementation of even standard arithmetic operations on numbers. It was then suggested that a microprogrammed functional unit with a wider data path and special features could be attached to

RIKKE 0 as an I/O device, or "functional unit", together with a wider memory, for use with these projects. A proposal was made to the Danish Research Council to obtain a grant to design and construct such a functional unit. A grant was made i June 1972 in which funds were awarded for hardware and a memory (32K, 64 bit wide, 1.4 µs access time). The manpower for the construction of the unit was, in part, granted by the Research Council; two staff engineers and one staff technician were provided by the Department. The design was started in May 1972 and completed during the summer of 1972. The construction of the resulting machine, MATHILDA, is due to be completed summer 1974.

The motivation for building the MATHILDA instead of purchasing a commercially available machine can be summarized as follows. First, there were (as far as we knew) no commercially available dynamically microprogrammable processors at the time we started our efforts which: (a) were in the price range we could afford, (b) were designed for or supported user written microcode or (c) offered a reasonable experimental and growth oriented structure. We felt that we had the in-house capability to design and construct the machine. The availability of LSI circuits and convenient mounting techniques and our experience with RIKKE 0 supported this view.

It was also decided that the new design for MATHILDE outdated the design of RIKKE 0, and with only minor modifications and additions could be used in the construction of a 16-bit machine, RIKKE 1, which is the subject of this description. Design criteria with respect to construction supported this view, these will be described in the next section.

1.2. General Design Criteria and Constraints

The RIKKE-0 machine is intended to be a research oriented machine. Its main design criteria then, within the money and timing constraints on the project, was to provide a machine on which a large variety of experiments related to processor and emulator design and evaluation could be performed. We attempted to use the "top-down" design approach which quite frequently was tempered by the "forces from below", see Rosin [5]. Therefore, we tried to let various application-oriented and software ideas be reflected in the design.

Two general software concepts had a reasonable impact on design. The one being the ability to multiprogram virtual machines and the other being the concept that virtual machines would be defined through several layers, (e.g., R. Dorin [6]). The effect of these ideas is apparent in the design of the control unit, especially with respect to the capabilities of addressing. Many addressing features known on the virtual level are present here on the micro level.

Another criterion was to have a clean and consistent way of dealing with timing problems. We decided not to force the speed; rather we would have a slower machine than obtainable with the componemtry at hand, and thus one, hopefully, with a reduced set of timing idiosyncrasies. It was also decided to be able to control all elements of the system from an immediate control or a residual control capability, or some combination of both. The residual control was made homogeneous to the user by having a reasonably "standard control register group" where ever such control was provided.

Another design criterion dealt with the actual construction of the unit. It had been decided, prior to the obtaining of the grant from the Danish Research Council, to construct additional RIKKE's by other funding. It became apparent, during the design phase of MATHILDA, that the machine would be reasonably complex and that several features of MATHILDA included or extended similar features on RIKKE 0. Because of the complexity of the design, the limited funds and manpower available, and the fact that we wished to design, construct, and test the machine within 1 year, it was decided that the additional RIKKE's (now called RIKKE 1's) should be modeled after the MATHILDA System. Thus, one design criterion was to ensure a modularity in the hardware design. This would enable an economy in print-lay out and construction to be achieved. As an example, the bus structure is laid out on one print board, 8-bits wide. Two of these boards, interconnected, comprise one RIKKE 1 bus structure with all registers, shifters, etc. Four of these RIKKE 1 boards, interconnected, give the MATHILDA bus structure. (For a description of the MATHILDA see Shriver [7]).

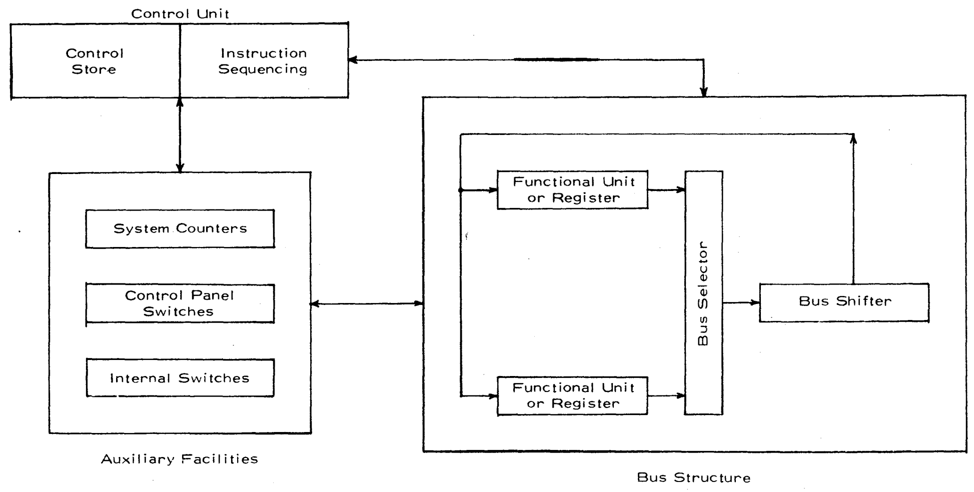
2. 0. The RIKKE 1 System

RIKKE 1, as has been stated earlier, is a microprogrammed controlled bus structure. The major elements of the system are shown in Figure 2. 1. and are the:

1) bus structure.
2) control unit.
3) auxiliary facilities.
4) I/O.
5) Memory.

In the following sections we will describe each of these systems independently and give examples of their utilization.

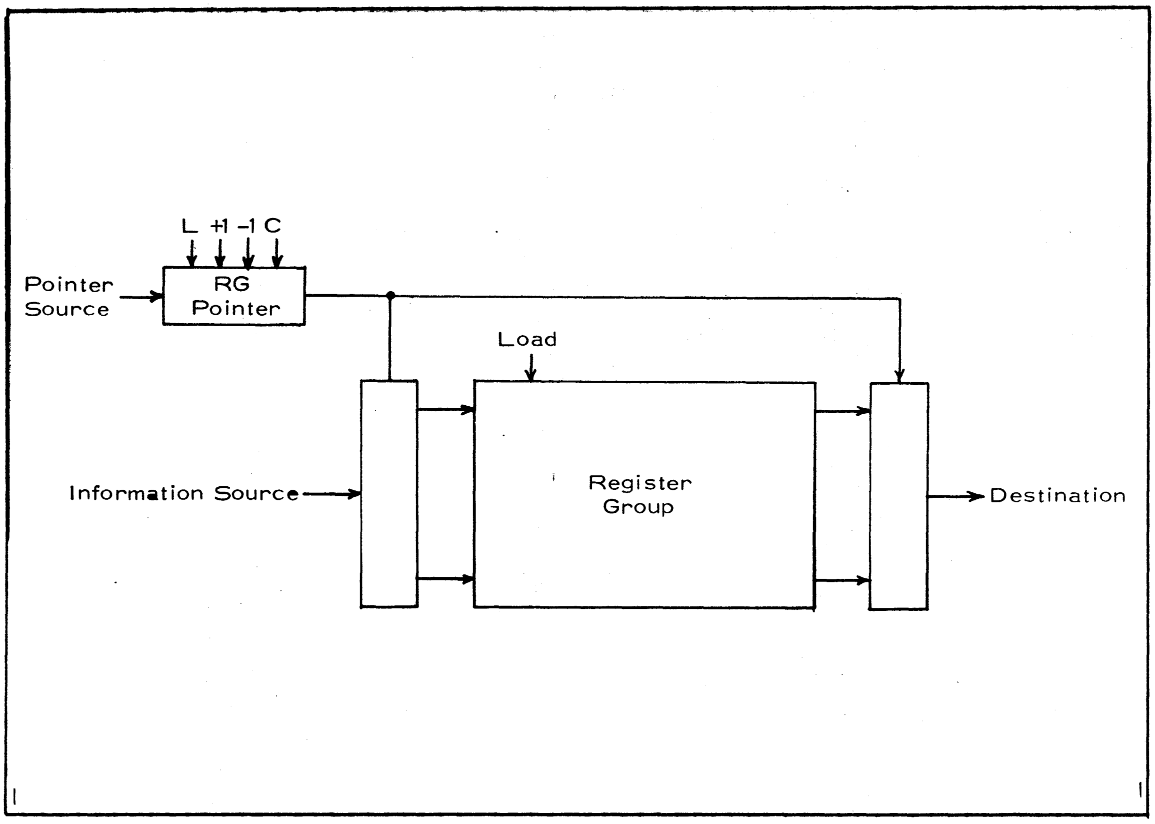
Figure 2. 1.
RIKKE 1 System



2. 1. The Register Group

We begin by introducing a fundamental building block which is used in the various control mechanisms of the system, viz, a Register Group RG*, as shown in Figure 2. 2. A RG is a set of 16 or 256 registers. The width of the registers and the number of registers in a specific RG will be stated when it is introduced. The element of a particular RG, which is to be used as a source or destination for the transfer of information, is pointed to by the RG address register. This register is called the Register Group Pointer, RGP, as shown in Figure 2. 2.

Figure 2. 2.
Typical Register Group



*) After a particular system element is first introduced, an abbreviation for its name is given which, for the sake of brevity, is then used in the text; see the "Tables of First Occurrance of Abbreviations and Symbols", beginning on page 121, for the page of first occurrance.

There are four microoperations associated with an RGP. They are marked L, +1, -1, and C in Figure 2.2. and all subsequent figures.

Table 2.1.
Microoperations for the control of an RG

Symbolic Notation		Microoperation
L	RGP:=Pointer Source	Load the RGP from the Pointer Source
+1	RGP + 1	Increment RGP by 1
-1	RGP - 1	Decrement RGP by 1
C	RGPC	Clear (i.e., set to zero) RGP

The symbolic notation RGP + 1, RGP - 1, etc. is the notation which is used with our microassembler, and all of our examples will be shown using this notation. The abbreviation 'RG' will often be replaced by the abbreviation of the name of the functional unit with which that particular RG is associated. Not all of the RGP's will have the microoperation

RGP:=Pointer Source

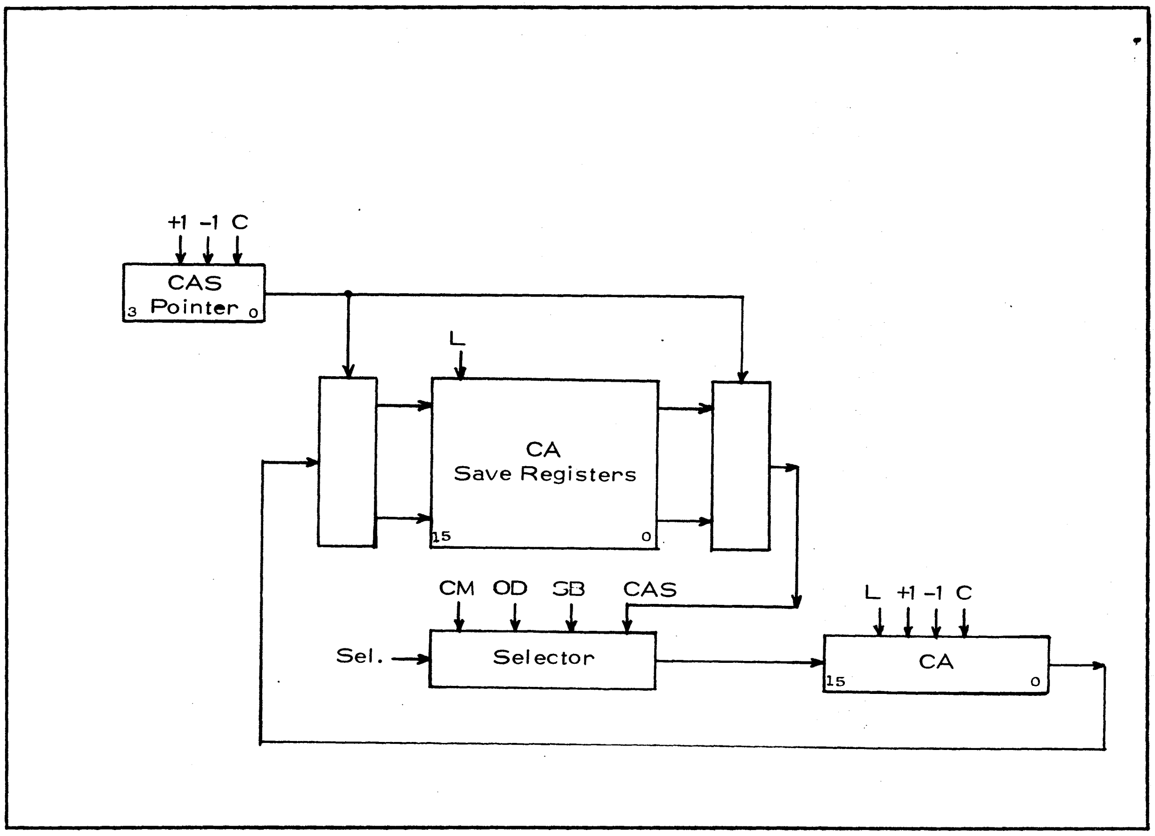
associated with them. For those RGP's which do have this microoperation it will be seen that the Pointer Source data itself can usually be selected to come from any of four different sources.

There is one additional microoperation required for the control of an RG; namely the function labelled "load" in Figure 2.2. If the loading of an RG can be initiated by a microoperation it will be indicated by an "L" on such a diagram.

2. 2. Counter A

We will, from time to time, give small segments of microcode to illustrate the use of a device and its control. In order to make these examples clearer and also to give a more realistic view of how such a code is actually written we introduce the system counter, Counter A, CA. CA is a 16-bit wide counter as shown in Figure 2. 3.

Figure 2. 3.
Counter A, CA



CA has four microoperations associated with it as shown in the box labelled 'CA' in this Figure. These microoperations are given in Table 2. 2.

Table 2. 2.

Microoperations for control of CA

Symbolic Notation		Microoperation
L	CA:=CM|OD|SB|CAS	Load CA from either CM, OD, SB, or CAS. Note the use of "|" to mean "or" in the symbolic notation for this microoperation.
+1	CA + 1	Increment CA by 1
-1	CA - 1	Decrement CA by 1
C	CAC	Clear (i. e., set to zero) CA

Both the box labelled "Selector" in Figure 2. 3. and the explanation of the microoperation "L" in Table 2. 2. state that CA can be loaded from one of four possible sources:

1) Immediate data within the Current Microinstruction, CM,
2) A 16-bit Output Register, OD (discussed in Section 2. 18.),
3) Bits 0 through 15 of the Shifted Bus, SB (discussed in Section 2. 5), and
4) From an element of a 16-bit wide, 16 element RG called the Counter A Save Registers, CAS.

Thus the microoperation

CA:=37

loads CA with the constant 37 from a data field within the CM. While the microoperation

CA:=CAS

loads CA with the contents of the element of CAS which is pointed to by the CAS Pointer, CASP. Notice that the CAS can be loaded with the contents of CA thus allowing one to save the current value of CA. The four microoperations associated with the CAS and CASP are in Table 2. 3.

Table 2. 3.

Microoperations for control of CAS and CASP

Symbolic Notation		Microoperation
L	CAS:=CA	Load the element of CAS pointed to by CASP with CA
+1	CASP + 1	Increment the CASP by 1
-1	CASP - 1	Decrement the CASP by 1
C	CASPC	Clear (i. e., set to zero) CASP

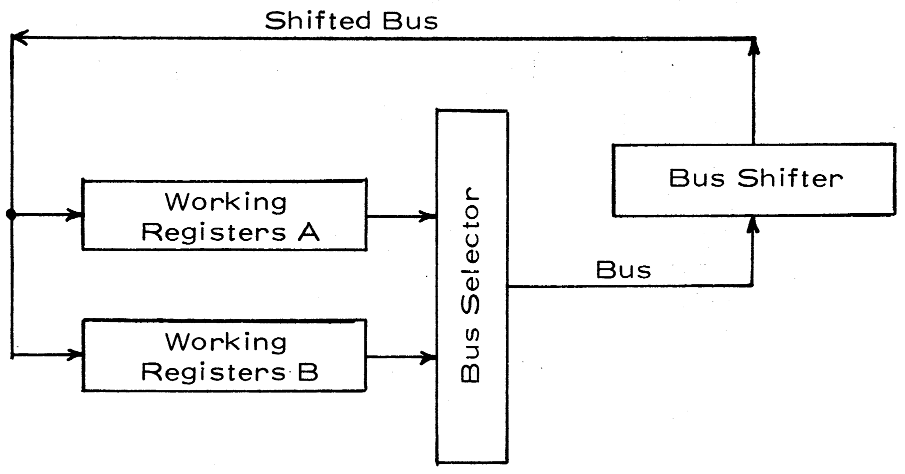
We can test to see if CA contains zero. We will demonstrate the use of this condition and the microoperations in Tables 2. 2. and 2. 3. in subsequent examples.

2. 3. Bus Transport

Having introduced some elementary notions we will now examine in some detail the bus structure, the registers and functional units attached to it, and the control which can be exercised on these components. We will construct the bus structure in a modular fashion - hopefully to enhance the reader's understanding and to underscore the overall simplicity and homogeneity of the structure and its components.

Let us introduce the concept of a bus transport by considering a subsystem of the bus structure consisting of the Working Registers A, WA, Working Registers B, WB, and the Bus Shifter, BS, as shown in Figure 2. 4. The exact nature of WA, WB and BS is not important to us here.

Figure 2.4.
Sub-system of the Bus Structure



The BUS is a 16-bit wide data path. The input to the BUS (its SOURCE) is obtained from a bus selector which has eight inputs, two of which are shown here. i.e., WA and WB. The particular input which is selected as the SOURCE for bus transport may be shifted a specified amount in the BS. The output of the BS, called the Shifted Bus, SB, can then be stored in at least one of seven possible 16-bit destinations (called Bus Destinations, BD, or DESTINATION). Two such BD's are shown in Figure 2.4. i.e., WA and WB. We will in this report specify bus transport information as we do in our microassembler, viz,

DESTINATION:=SOURCE, BS Specification.

If the BS Specification field is empty, i.e., the BS is not to be used (no shift occurs) then the bus transport is given by

DESTINATION:=SOURCE.

As an example, the bus transport WB:=WA has the obvious meaning of a register to register transfer from WA to WB. If a SOURCE is chosen to be transported but not stored in any of the BD's, the bus transport information is written

SOURCE, BS Specification

or

SOURCE

as is appropriate. The SOURCE may be stored in destinations other than BD's during a bus transport. We will learn what functional units or registers can serve as these "other destinations" as this report develops. If the SOURCE is to be stored in more than one destination, the DESTINATION portion of the bus transport specification is written as a list of destinations separated by commas, i.e.,

LIST:=SOURCE, BS Specification

or

LIST:=SOURCE

where

LIST::=d_1 , . . . , d_n .

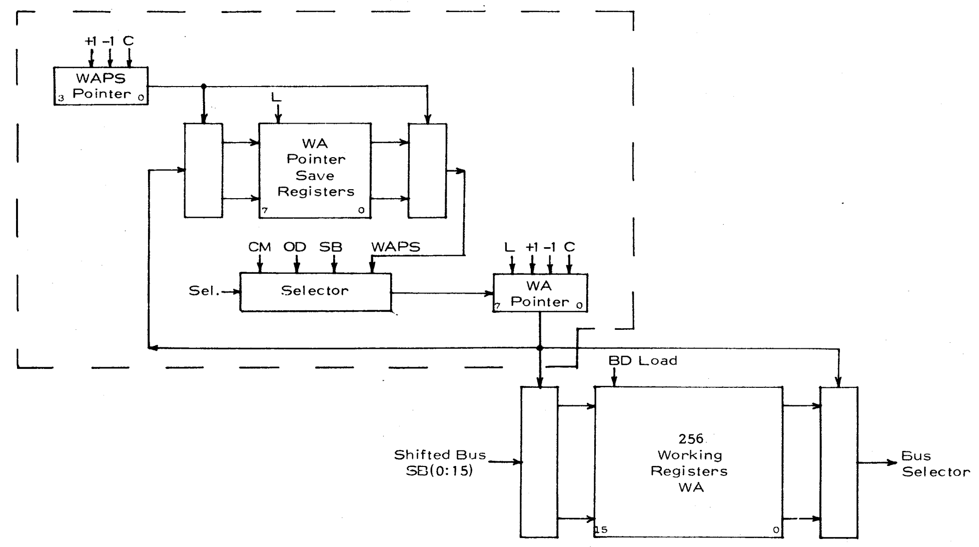
The value of n and the units which can serve as destinations, d_1 , will be discussed later.

2.4. Working Registers

WA and WB, introduced in the previous section, are not single registers but each is a 16-bit wide, 256 element RG. Figure 2.5. shows WA; WB, not shown, is identical.

The first thing we wish to point out in this figure is that the WA Pointer, WAP, is a mechanism identical to CA except that it is 8-bit wide and not 16-bit wide. (Note the dashed-line box in Figure 2.5.). Therefore, WAP not only points to which element of WA can be used as a SOURCE for bus transport (or used as a BD), but also can be stored in an RG

Figure 2. 5.
Working Registers, A, WA



called the WAP Save registers, WAPS. This is identical to CA being saved. Also, as indicated in the box labelled "Selector" in Figure 2. 5. the WAP can be loaded from any of four sources:

1) immediate data from the CM
2) the least significant 8-bits from OD *)
3) the least significant 8-bits of the SB, *) and
4) an element of WAPS.

This is identical to the loading of CA. Thus the microoperations WAP:= 37 and WAP:=WAPS have well defined analogous meanings.

The WA (and WB) registers are not loaded by a microoperation but rather as a result of being chosen as a BD in a bus transport specification; thus the loading of these registers is shown by the function "BD Load" on Figure 2. 5. This notation will be used in all subsequent drawings.

*) WB is different with respect to 2) and 3) in the sense that loading of WBP takes place from the most significant 8-bits of OD and SB.

There are 8 microoperations shown in Figure 2. 5. associated with the use of WA. These are listed along with the corresponding microoperations for WB in symbolic form in Table 2. 4. The actual microoperation descriptions can be extracted from the previous tables and are not repeated here.

Table 2. 4.
Microoperations for control of WA and WB

WAP:=CM | OD_{0-7} | SB_{0-7} | WAPS	WBP:=CM | OD_{8-15} | SB_{8-15} | WBPS
WAP + 1	WBP + 1
WAP - 1	WBP - 1
WAPC	WBPC
WAPS:=WAP	WBPS:=WBP
WAPSP + 1	WBPSP + 1
WAPSP - 1	WBPSP - 1
WAPSPC	WBPSPC

2. 4. 1. Microinstruction Format and a Few Examples

In order to present a few examples we will introduce the microinstruction format which we use in our microassembler. The format of a microinstruction is:

"A: bus transport; microoperations and data; microinstruction sequencing. ",

where

a) "A" is a symbolic name for the address of the microinstruction,

b) "Bus transport" is a field giving the bus transport information as explained previously in Section 2. 3. ,

c) "microoperations and data" is a field of up to 7 microoperations and immediate data to be executed or used during this microinstruction (the exact combination of microinstructions and data which can be included in this field and precise details of the timing of microoperations are given in Section 3. 0.).

d) "microinstruction sequencing" information will be written in the form

if c then A_t else A_f

which is to mean: if a particular selected condition c is true then choose address A_t as the address of the next microinstruction else choose A_f .

It is not necessary or appropriate at this point to list all of the conditions which are testable by the system nor how A_t and A_f are functions of the address of the current microinstruction, A. These matters will be dealt with in Section 2. 20. 1. However, conditions and address functions will be introduced as needed for examples. If no condition is to be considered, i. e. , if $A_t = A_f$, the sequencing information will merely be written A_t (and not "if c then A else A " where c is an arbitrary condition).

Thus, the microinstruction labelled A ,

A: WA:=WB; WBP + 1; A + 1 .

means: load the element of WA pointed to by WAP from the element of WB which is pointed to by WBP without shifting it during the bus transport; Then increment WBP by 1; then obtain the next microinstruction from A + 1 . The action associated with <u>every</u> microoperation specified in a microinstruction is completed <u>before</u> the next microinstruction is executed. For example, in the above microinstruction if WBP had been set to 9 before the beginning of the execution of this instruction, then WB9 would be the SOURCE for the bus transport. At the end of execution of the instruction, the WBP would be set to 10. If, in the next microinstruction WB were again selected as the SOURCE, then the contents of WB10 would be gated onto the BUS.

In order to give an example of a microinstruction using conditional branching, we establish the following convention for the testing of conditions which will be used in all of our examples (unless stated explicitly otherwise): all conditions which arise as a result of bus transport and microoperation execution specified by a particular microinstruction, M, are testable in the next microinstruction to be executed after M is executed. This means that all the conditions available or changed during the execution of microinstruction M are "saved". These "saved" conditions are those tested in the next instruction to be executed. Therefore, our microinstruction can be thought of being executed in the following sequential way:

(a) save the conditions of the previous microinstruction
(b) execute bus transport
(c) execute microoperations
(d) execute microinstruction sequencing based on saved conditions.

Let us introduce the notation that bit 15 of the WA input to the bus selector is testable, that is, bit 15 of the element of WA which is pointed to by WAP. If we wish, for example, to test bit 15 of WA7, and if it is set to 1, jump to the microinstruction labelled BITON, else continue with the next microinstruction, we could write,

```
A-1:  ; WAP:=7
A  :  ; if WA(15)   then BITON else A+1
A+1:
```

We could not write

```
A  :  ; WAP:=7; if WA(15)   then BITON else A+1 ,
```

according to our current convention. It is possible to conditionally repeat the same instruction. Let us give an example of this. Assume there is at least one register in WA which contains bit 15 set to 1, the following four microinstructions will: search WA starting with register 0 and transfer the first register of WA encountered with bit 15 set to 1 to register 0 of WB; then, store the address of the WA register which was transferred in register 0 of WAPS; and then continue with the next microinstruction.

```
        ; WAPC, WAPSPC, WBPC .
LOOP:   ; WAP + 1; if WA(15)   then SAVE else LOOP .
SAVE:   ; WAP - 1.
WB:=WA ; WAPS:=WAP.                                ■
```

We have introduced some standard defaults in this example:

a) If the bus transport field is empty it means that an unspecified source is selected for bus transport but is not stored anywhere.

b) If the microoperations field is empty it means that no microoperations are to be exectuted during this particular microinstruction.

c) An empty microinstruction sequencing field implies the next microinstruction to be executed is that in A + 1 if the address of the current microinstruction is A . If you wish to use comments these must start with ". " (period).

d) Any instruction sequence shown is assumed to be located sequentially in control store and the symbolic address name is used only when needed in the microinstruction sequencing field of some other instruction.

e) The symbol ■ will be used to indicate the end of the group of microinstructions in the example.

The symbolic names HERE-1, HERE, and HERE+1 are used often in the microinstruction sequencing field to mean A-1, A, and A+1 assuming the address of the current microinstruction is A . As an example, the instruction labelled LOOP above could have been written

```
        ; WAP+1; if WA(15)   then HERE+1 else HERE. ■
```

Through the use of CA the assumption that at least one register of WA contains bit 15 set to 1 is not required. CA can be used to control the number of elements of WA we will search. If we establish a routine labelled NONE which handles the situation when no element of WA contains bit 15 set to 1, then the code to perform the same task as related above is,

```
           ; WAPC, WAPSC, WBPC.
           ; CA:=255; TEST.
           ; WAP+1, CA-1; if CA   then NONE else HERE + 1.
    TEST:; if WA(15)   then HERE+1 else HERE-1.
  WB:=WA ; WAPS:=WAP. ■
```

The final example in this section uses the capability of loading CA from the SB. In the previous example CA was loaded with N-1 where N($2 \le N \le 256$) is the number of registers of WA to be searched. Let us suppose that this number is in register 0 of WB and furthermore that you wish to save it in register 0 of CAS because it may be written over if a transfer is made to WB. A possible code segment is,

```
           ; WAPC, WAPSPC, WBPC.
  WB       ; CASPC, CA:=SB.
           ; CAS:=CA; TEST.
           ; WAP+1; if CA   then NONE else HERE+1.
    TEST:; CA-1; if WA(15)   then HERE+1 else HERE-1.
  WB:=WA ; WAPS:=WAP. ■
```

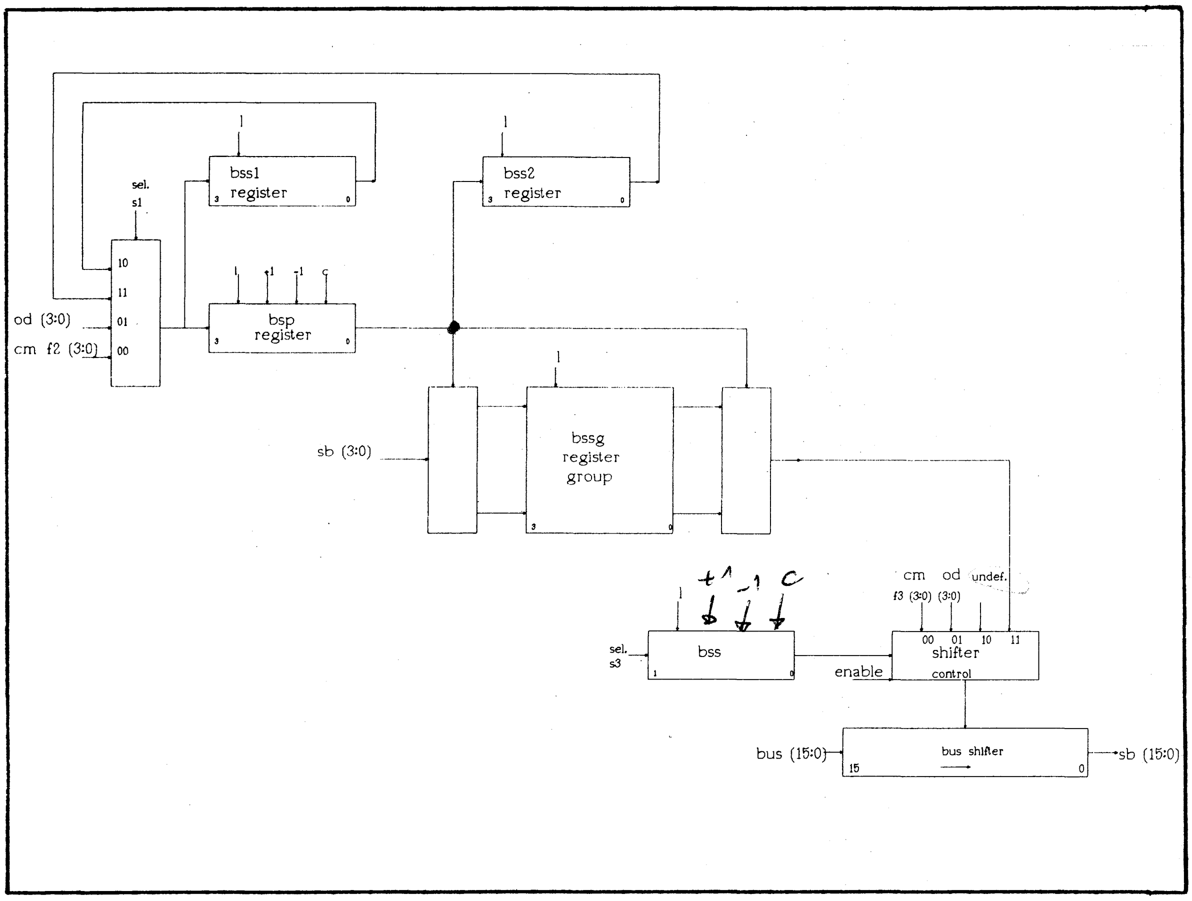
If the A_f address is HERE+1 we will only write, from now on, if c then A_t . Thus, the fourth instruction of the above example would be written

```
           ; WAP+1; if CA   then NONE. ■
```

2. 5. The Bus Shifter

The Bus Shifter, BS, introduced in Figure 2. 4. and shown in more detail in Figure 2. 6. is a 16-bit wide right cyclic shifter which can be set to shift n bits, 0≤n≤15. There exists a dedicated bit in each microinstruction to control the BS which indicates whether or not the BS should be used (enabled) during the current bus transport. If the BS is not enabled, no shift will occur.

Figure 2. 6.
Bus Shifter, BS



If we wish to use the BS, the amount of shift can be selected from one of three possible sources as shown in the box labelled "Shift Control" in Figure 2. 6. , i. e. , from

1) a data field in the CM,
2) the least significant 4 bits of the OD register,
3) an element of a 4-bit wide 16 element RG called the BSSG.

Which of these four sources is used is determined by BSS. This is loaded from CM S3(0:1). By default BSS:=CM, and you are advised to reset the BSS if you change it. The bus transport specification

WA:=WB

means: take the element of WB pointed to by the WBP and store it in the element of WA pointed to by the WAP without shifting is. While the bus transport specification

WA:=WB, $\rightarrow$ 3

means: take the element of WB pointed to by the WBP, shift it 3 bits right cyclic and then store it in the element of WA pointed to by WAP, assuming that the BSS is set to select CM as the datasource. This will be assumed to be the standard setting of BSS in the following.

A 16-bit left cyclic shifter and a 16-bit right cyclic shifter are related by the expression

lcs = 16 - rcs

where

lcs is the amount of left cyclic shift and
rcs is the amount of right cyclic shift.

We can therefore write as a notational convenience

WB:=WA, $\leftarrow$ 5

to mean the same thing as

WB:=WA, $\rightarrow$ 11

thus using ←(left shift) or →(right shift) whichever makes the understanding of the processing clearer. The microassembler will do the proper computation and insert the correct amount for right shifting in the datafield.

The BS specification in the bus transport field of the microinstruction is given by

$\left\{\begin{smallmatrix}\rightarrow\\ \leftarrow\end{smallmatrix}\right\}$ CM|OD(X |BSSG

The BSS-selector chooses from which source the shifter-control data is to be taken, whether or not you indicate the source in the actual microinstruction.You can load BSS by the microoperation BSS:=CM|OD |BSSG.

Having seen how the BS is controlled and how we specify this control, let us turn our attention to the BS register group Pointer, BSP. We see in Figure 2. 6. that the data which can be loaded into the BSP can also be loaded into an additional register called the BS Save1 register, BSS1. If, for example, we know in advance the address of a particular register in the BSSG, which we will want to use as shift data (e. g., some highly used shift constant), we can store this pointer in BSS1 by loading BSS1 from the CM,

BSS1:=CM.

Whenever we wish to use this stored pointer we can load it into the BSP by executing

BSP:=BSS1.

Now notice in Figure 2. 6. that the BSP not only points to the element of the BSSG which can be chosen as data for the shift control unit, but also can be stored in a register called the BS Save2 register, BSS2. Suppose we are pointing at a particular element of the BSSG for the current shift control data and in the next microinstruction we wish to have register 9 of the BSSG to be used as shift data, but we do not wish to loose the pointer to our current control data. The following microinstruction achieves this,

; BSS2:=BSP, BSP:=9. ■

Thus at some later time if we execute

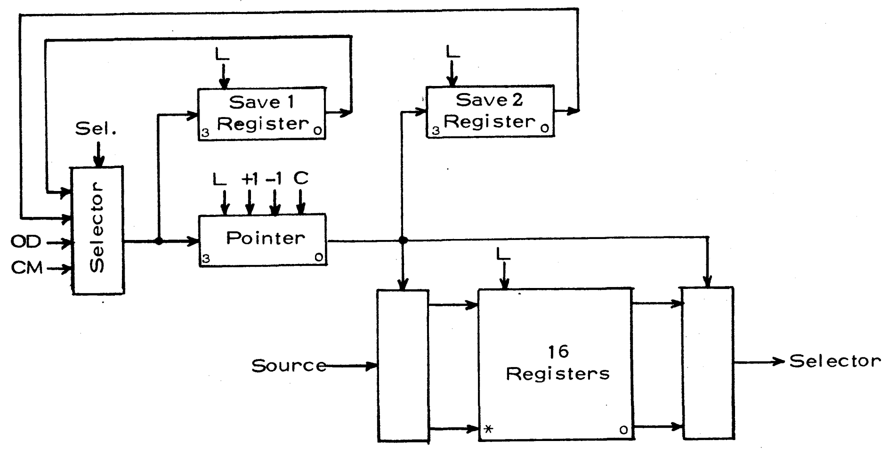
BSP:=BSS2

the pointer information which had been saved in BSS2 would be restored.

A 16 element RG with the two Save registers and Pointer as shown in Figure 2.7. is a fundamental control element in the system and will be used with many devices in the subsequent sections. It will be referred to as a Standard Group (SG) and will be noted on drawings as such, i.e., it will not be explicitly be drawn out each time as it was in Figure 2.6. Each SG will, however, be given a name closely associated with the particular functional unit to which it is connected as, for example, in the current discussion the SG associated with the BS is called the BSSG.

Figure 2.7.

Typical Standard Group



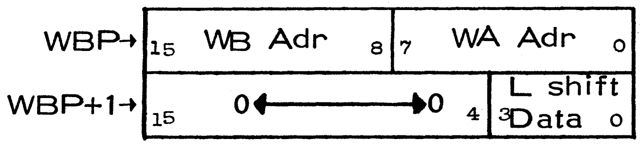
* The width of the registers depends on the particular selector involved.

Table 2.5, below, lists the seven microoperations associated with the BS in their symbolic form; their meanings should be obvious from previous tables and the text. Note that the BSSG is loaded with the least significant 4 bits of the SB i.e., SB(0:3).

Table 2.5.
Microoperations for control of the BS

BSP:=CM|OD|BSS1|BSS2
BSP+1
BSP-1
BSPC
BSS1:=CM|OD|BSS1|BSS2
BSS2:=BSP
BSSG:=SB

Let us assume the following information to be in WBP and WBP+1:



We wish to take a given WB register (WB Adr), shift it a given amount (L Shift Data), and store it in a given WA register (WA Adr).

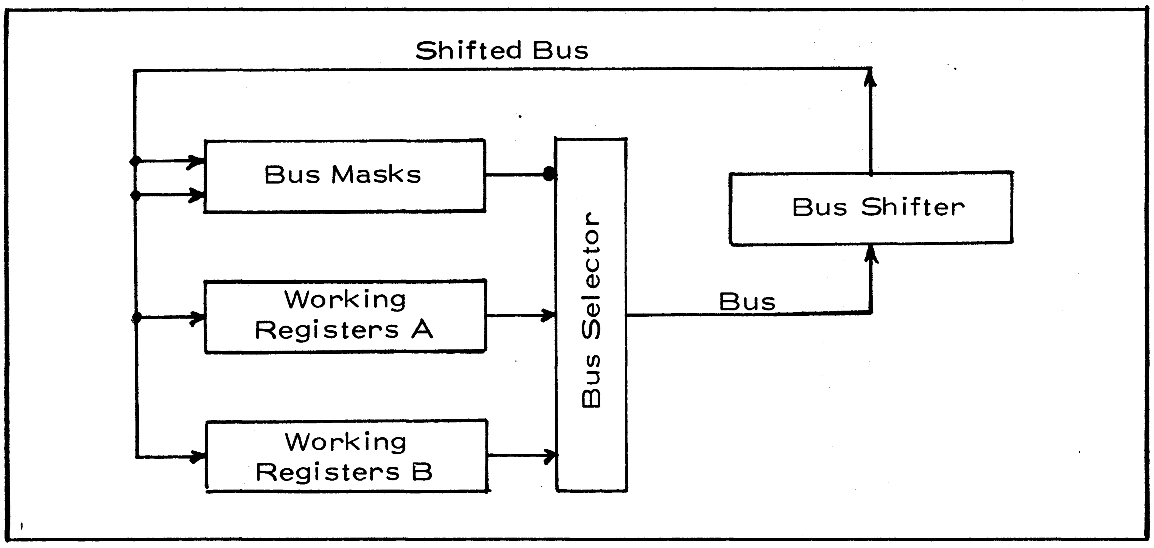
The following code will: Load the BSSG with the L shift Data, Save the current WBP, load WBP with the WB Adr, Load WAP with the WA Adr, transfer the WB register pointed to by WB Adr to the register pointed to by WA-adr shifting it left cyclic by the amount L shift Data during transport, restore the old WBP, and then continue.

```
WB          ; WAP:=SB, WBP + 1 .
WB          ; BSSG:=SB, WBP - 1 .
WB          ; WBP:=SB, WBPS:=WBP , BSS:=BSSG.
WA:=WB,←    ; WBP:=WBPS .
            ; BSS:=CM.                        ■
```

2.6. Bus Masks

Let us now expand the initial bus structure given in Figure 2.4. by adding the Bus Masks, BM, as shown in Figure 2.8.

Figure 2.8.
Expanded Bus Structure



The BM allow one to specify which bits of the SOURCE (i.e., the particular input to the bus selector which has been selected for bus transport) are actually to be transported. A mask is a string of 16-bits. If bit i $(0 \le i \le 15)$ of a mask is a 1, then bit i of the SOURCE is to be transmitted; if bit i of the mask is a 0, then the value 0 is to be transmitted. Since the BM are not an input to the bus selector but affect the transmission of the SOURCE, they are shown connected to the bus selector with the symbol $\longrightarrow\!\bullet$(which we will interpret to mean "mask") and not by the symbol $\longrightarrow$ (which means "input"). WARNING!!! <u>When the Busmask is loaded it is the inverted SB which is loaded into BM.</u>

The SOURCE is masked during <u>every</u> bus transport by the mask which is specified to be

MA v MB

where

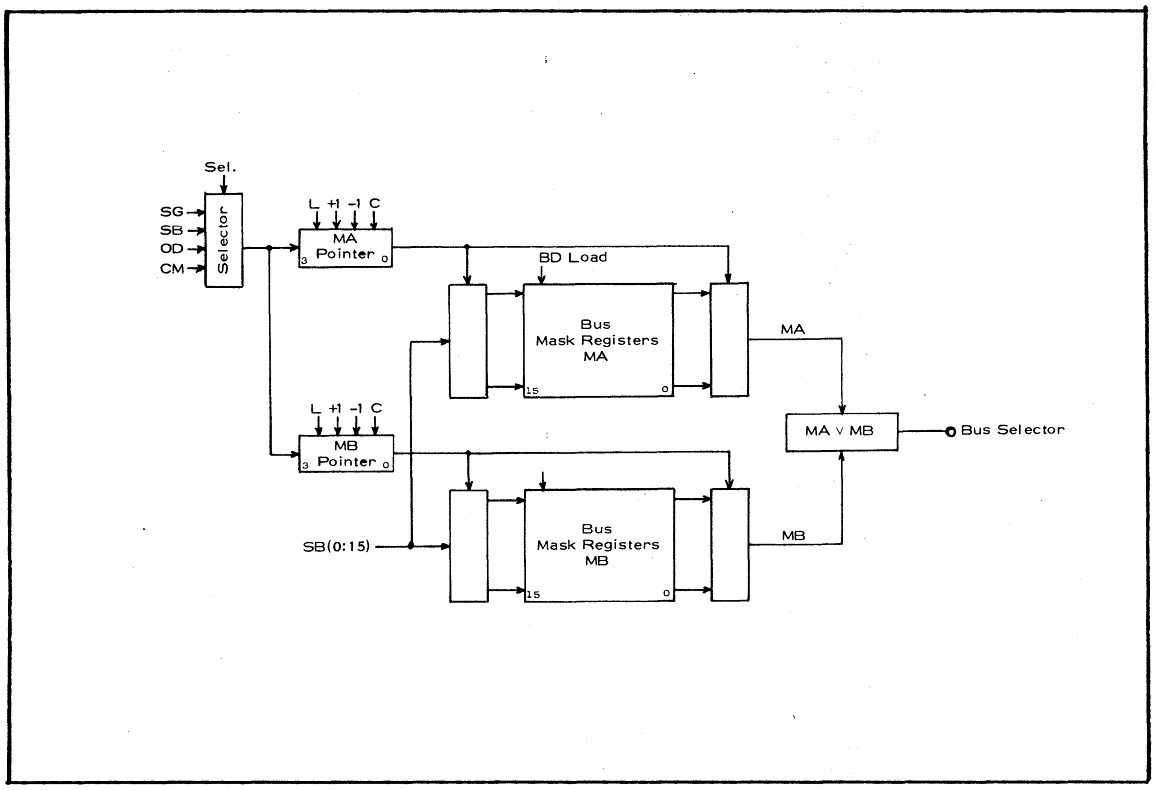
MA = an element of a 16-bit wide, 16 element RG called the Mask A registers,

MB = an element of a 16-bit wide, 16 element RG called the Mask B registers,

v = logical "inclusive or".

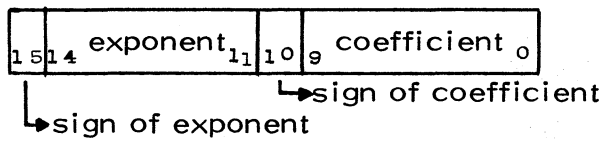
MA and MB are shown in Figure 2.9. Upon dead start, the system is

Figure 2.9.
Bus Masks, MA and MB

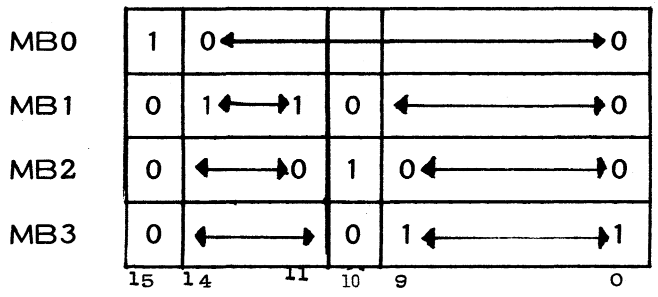


such that the "no mask", i.e., 15 I's, is in register 0 of MA and the "bus clear mask", i.e., 16 0's, is in register 1 of MA. We will assume this to be the case throughout normal operation of the system. One can then look upon the pointer MAP as a switch for the use of the bus masks: If MAP = 0 then the BUS is not masked, if MAP = 1 then the BUS is masked by the mask specified by MB. This is, of cource, not the only interpretation of the use of the BM but it is a convenient one and one which we will normally employ unless otherwise stated.

As an example, with no sensible applications, assume we are representing very small floating point numbers in the following sign magnitude format,



Suppose the following 4 masks are available in the first 4 registers of MB.



The following code will decompose a floating point number found in the register of WA pointed to by WAP and store the information as follows,

1) sign of the exponent in bit 15 of WB0
2) magnitude of the exponent in WB1(15:12)
3) sign of coefficient in bit 15 of WB2
4) magnitude of the coefficient in WB3(15:6)

```
               ; MAPC.
               ; MAP+1, MBPC, WBPC.
WB:=WA         ; MBP+1, WBP+1.
WB:=WA, ← 1 ; MBP+1, WBP+1.
WB:=WA, ← 5 ; MBP+1, WBP+1.
WB:=WA, ← 6 ;                            ■
```

It is suggested by this example that when one is decomposing formatted information (e. g., a virtual machine instruction) one may wish to co-ordinate the use of the BS with the use of the BM. Let us therefore suppose the shift constants 0, 15, 11, and 10 to be stored in the first 4 registers of the BSSG. The above decomposition and storage could be written as the following 3 microoperations:

; CA:=3, MAPC, BSS:=BSSG.
; BSPC, WBPC, MBPC, MAP+1.
WB:=WA ; BSP+1, WBP+1, MBP+1, CA-1; if CA then $\overset{+1}{\text{HERE}}$ else HERE~~+1~~.
; BSS:=CM. ■

The MA Pointer (MAP) and the MB Pointer (MBP) both of which were used in the above examples are loadable either separately or together; thus we can execute the microoperations

```
     MAP:=CM|OD|SB|SG,
     MBP:=CM|OD|SB|SG, or
MAP, MBP:=CM|OD|SB|SG .
```

The name of the SG associated with the BM is the Bus Mask Pointer (BMP) Standard Group. The following table lists the microoperations associated with MA, MB, and BMP.

Table 2.6.
Microoperations for control of the BM

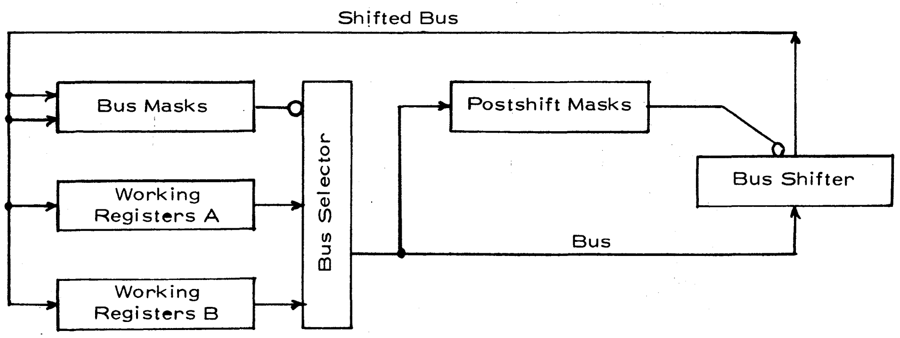
MAP+1 MAP-1 MAPC MAP:=CM|OD|SB|SG	MBP+1 MBP-1 MBPC MBP:=CM|OD|SB|SG
MAP, MBP:=CM|OD|SB|SG	

BMP:=SB
BMPP:=CM|OD|BMPS1|BMPS2
BMPP+1
BMPP-1
BMPPC
BMPS1:=CM|OD|BMPS1|BMPS2
BMPS2:=BMPP	

2.7. Postshift Masks

The Bus Masks, as described in the previous section, are applied to the SOURCE as it is gated onto the BUS and thus before the SOURCE is shifted in the BS. There is also a possibility of masking the SOURCE after it has been shifted by using the Postshift Masks, PM, as shown in Figure 2.10.

Figure 2. 10.
Expanded Bus Structure



One of the purposes of the PM is to apply a mask to the output of the BS which will mask off the unwanted "cyclic" bits and replace them with 0's thereby simulating a logical shift. As an example, if the bus transport

WB:=WA, ← 2

is executed with the postshift mask

$$\boxed{{}_{15}1 \longleftrightarrow 1\,0\,0_{2\,1\,0}}$$

applied to the output of the BS, then we have taken a WA register, shifted it 2 bits left logical, and stored it in a WB register. Similarly, the bus transport

WB:=WA, → 6

with the mask

$$\boxed{{}_{15}0000001_{10\;9} \longleftrightarrow 1_{0}}$$

applied to the output of the BS means a WA register is shifted 6 bits right logical and then stored in a WB register. The output of the BS is masked during <u>every</u> bus transport by the mask which is specified to be

PA $\vee$ PG

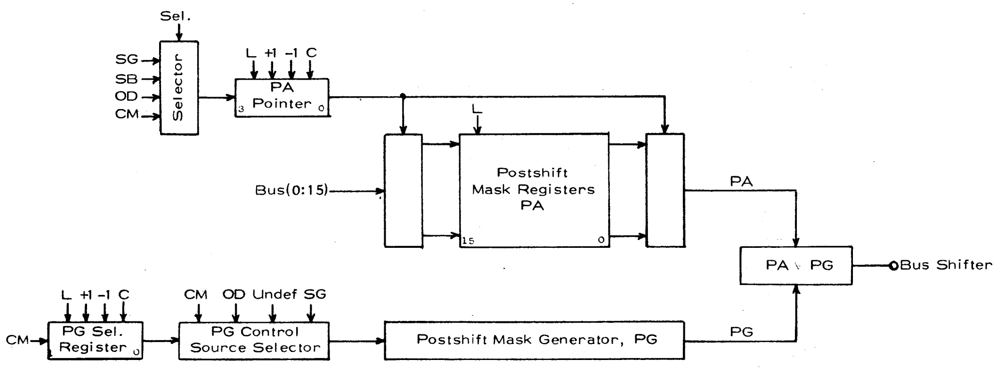
where,

PA	=	an element of a 16 bit wide, 16 element RG called the Postshift Mask A registers,
PG	=	a functional unit called the Postshift mask Generator,
$\vee$	=	logical "inclusive or".

PA and PG are shown in Figure 2.11. This is quite similar to the BM where PG now takes the place of MB.

WARNING!!! As with the BM, <u>when PA is loaded from the bus, it is the inverted bus which is loaded</u>.

Figure 2.11.
Postshift Masks, PA and PG



The PG is a 32 word ROM which can be addressed through PGS. The contents of the ROM is

Table 2.7.
Table representing the READ-ONLY-MEMORY
containing the 32 Masks for the PG

PG-DATA		MASK SELECTED IN POSTSHIFT MASK GENERATOR																$1 \leq n \leq 15$
DEC	BINARY																	
0	00000	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	PG→0, PG←0
1	00001	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
2	00010	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
3	00011	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	
4	00100	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	
5	00101	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	
6	00110	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	
7	00111	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	PG→n
8	01000	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
9	01001	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	
10	01010	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	
11	01011	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	
12	01100	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	
13	01101	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	
14	01110	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	
15	01111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
16	10000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PG→16, PG←16
17	10001	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
18	10010	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
19	10011	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
20	10100	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
21	10101	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	
22	10110	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	
23	10111	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	
24	11000	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	PG←n
25	11001	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	
26	11010	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	
27	11011	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	
28	11100	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	
29	11101	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	
30	11110	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	
31	11111	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	

The PG can generate the 32 masks required to view the BS as both a logical and cyclic shifter. As is seen from Figure 2.11. the postshift mask generation data can come from one of three sources, CM|OD|SG.

Which particular source is to be used as data for the mask generation is determined by the contents of a 2-bit Postshift mask Generator Selection register (PGS) as shown in this figure and in Table 2.8. below.

Table 2.8.
Source of Data for Postshift Mask Generation

Contents of PGS	Source of DATA
00	CM
01	OD
10	(undef)*
11	SG

If, what we will assume as standard, the PGS has been set to point to the CM as the data source, then the PG data are specified in the "microoperations and data" field of the microinstruction in the following symbolic way,

PG "arrow" n

where,

n = the number of 0's to be generated and the "arrow" (←|→) indicates from which direction they should be generated; $0 \leq n \leq 16$.

Thus, the previous two examples could have been written (assuming PGS and BSS points to the CM as the data sources).

WB:=WA, ← 2; PG←2
WB:=WA, → 6; PG→6

Upon dead start, the system is such that the mask of all 1's is in register 0 of PA, and the mask of all 0's is in register 1 of PA. This is identical to the situation in MA. We will assume this to be the case

*) At the moment undefined

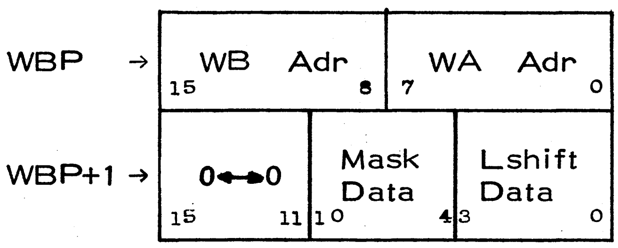
throughout normal operation of the system. One can then look upon the pointer PAP as a switch for the use of the Postshift mask Generator: if PAP = 0 then the mask generator is not used, if PAP = 1 then the postshift mask which is to be applied will be that generated by the mask generator. This is, of course, not the only interpretation of the use of the postshift masks, but it is a convenient one and one which we shall normally employ unless otherwise stated.

Table 2.9. is a list of the microoperations associated with the PM. The first half of this table deals with PA. The second half of this table deals with the PG. The name of the SG associated with the PG control is the Postshift Mask Generator SG (PGSG). Note, the name of the SG associated with the PA pointer is the Postshift AB Pointer (PABP). It is not discussed here but in Section 2.28.

Table 2.9.
Microoperations for the control of the PM

Operations associated with PA
PA :=BUS
PAP:=CM|OD|SB|SG
PAP +1
PAP -1
PAPC
Operations associated with PG and PGSG
PGS:=CM
PGS +1
PGS -1
PGSG:=SB
PGP :=CM|OD|PGS1|PGS2
PGP +1
PGP -1
PGPC
PGS1:=CM|OD|PGS1|PGS2
PGS2:=PGP

Let us extend the example of Section 2.5. in which we emulated a virtual machine instruction which performed a register to register transfer combined with left/rigth cyclic shifting. As shown below, if we use the PG we can execute an instruction which will take a given WB register (WB Adr), shift it left/right logical or cyclic (Shift & Mask Data), and then store it in a WA register (WA Adr). If the data for the instruction is in the current WB register pointed at by WBP in the form



a possible code sequence would be:

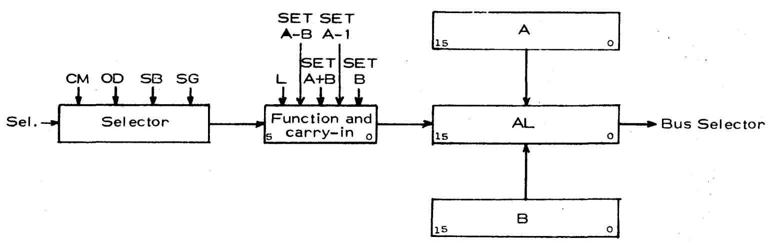
```
WB,            ; WAP:=SB, WBP+1.
WB, →4         ; PG SG:=SB.
WB             ; BSSG:=SB, WBP-1.
WB             ; WBP:=SB, WBPS:=WBP.
               ; PAP+1, PGS:= SG . BSS:=SG
WA:=WB,←       ; ~~PG+~~, WBP:=WBPS,
               ; PAPC, PGS:= CM . BSS:=CM        ■
```

Note well, there are two important assumptions in this example. The first is that MAP = 0 upon entry to this code, i. e., a bus mask is not applied to the source, and the second is that PAP = 0 upon entry to this code, i. e., no postshift masking occurs. Indeed, we will make these assumptions in all examples which follow (unless stated explicitly otherwise). They can be summarized as follows: bus transport normally occurs in an unmasked fashion; if a particular code segment requires the use of a masking facility it is responsible for leaving the system in this normal state after such masking occurs.

2. 8. The Arithmetical and Logical Unit

We will now add additional computational capability to the bus structure in addition to the shifting and masking already encountered by introducing the Arithmetical and Logical Unit (AL). The AL, shown in Figure 2. 12., is a functional unit with 2 inputs which, for the moment we will call A and B.

Figure 2. 12.
Arithmetical Logical Unit, AL



6 bits are required to control the AL: 5 bits to select one of the 32 operations listed in Table 2.10. which this unit can execute on A and B and 1 bit which specifies the carry-in bit into the AL for any arithmetic operations.

Table 2.10.
AL Functions

ARITHMETIC	LOGICAL
A	$\overline{A}$
$A \vee B$	$\overline{A} \wedge \overline{B}$
$A \vee \overline{B}$	$\overline{A} \wedge B$
minus 1*	all 0's
$A + (A \wedge \overline{B})$	$\overline{A} \vee \overline{B}$
$(A \vee B) + (A \wedge \overline{B})$	$\overline{B}$
$A-B-1$	$A \not\equiv B$
$(A \wedge \overline{B})-1$	$A \wedge \overline{B}$
$A + (A \wedge B)$	$\overline{A} \vee B$
$A + B$	$A \equiv B$
$A \vee \overline{B} + (A \wedge B)$	B
$(A \wedge B)-1$	$A \wedge B$
$A + A$	all 1's
$(A \vee B)+A$	$A \vee \overline{B}$
$(A \vee \overline{B})+A$	$A \vee B$
$A-1$	A

* in 2's complement; the arithmetic operations are shown with the carry-in set to 0. If the carry-in is 1, then the AL Function is F+1 where F is the specified arithmetic function. The logical functions are not affected by the carry-in.

The 6 control bits which specify the current operation for the AL are the contents of the AL Function and Carry-in register, ALF, which can be loaded, ALF:=CM|OD|SB|SG, or set to the arithmetic addition operation A + B and other standard settings. The SG associated with the ALF is called the AL Standard Group (ALSG). The microoperations associated with the AL are given in table 2. 11.

Table 2. 11.
Microoperations for control of the AL

ALF:=CM|OD|SB|SG
SET ALF +
SET ALF -
SET ALF B
SET ALF A - 1
ALSG:=SB
ALP:=CM|OD|ALS1|ALS2
ALP +1
ALP -1
ALPC
ALS1:=CM|OD|ALS1|ALS2
ALS2:=ALP

If the ALF is to be loaded with an operation specification from the CM, we will note this symbolically merely by writing the required function in the symbolic form which appears in Table 2.10. in the ALF assignment statement, i. e.,

ALF:=A + B
ALF:=A $\wedge$ B
etc.

The AL is always running. If the ALF is changed in a microinstruction, then the result of the newly computed function is available for bus transport in the very next microoperation. Thus the microinstructions

```
          ; ALF:=all 1 s, PAP +1,
   WA:=AL; PG → 9, PAP -1 .                    ■
```

will put a string of 7 1's in the WA register pointed to by WAP. The 1's will be least significant bits, b_0, justified.

There are many testable conditions concerning the operation of the AL. A few of these are

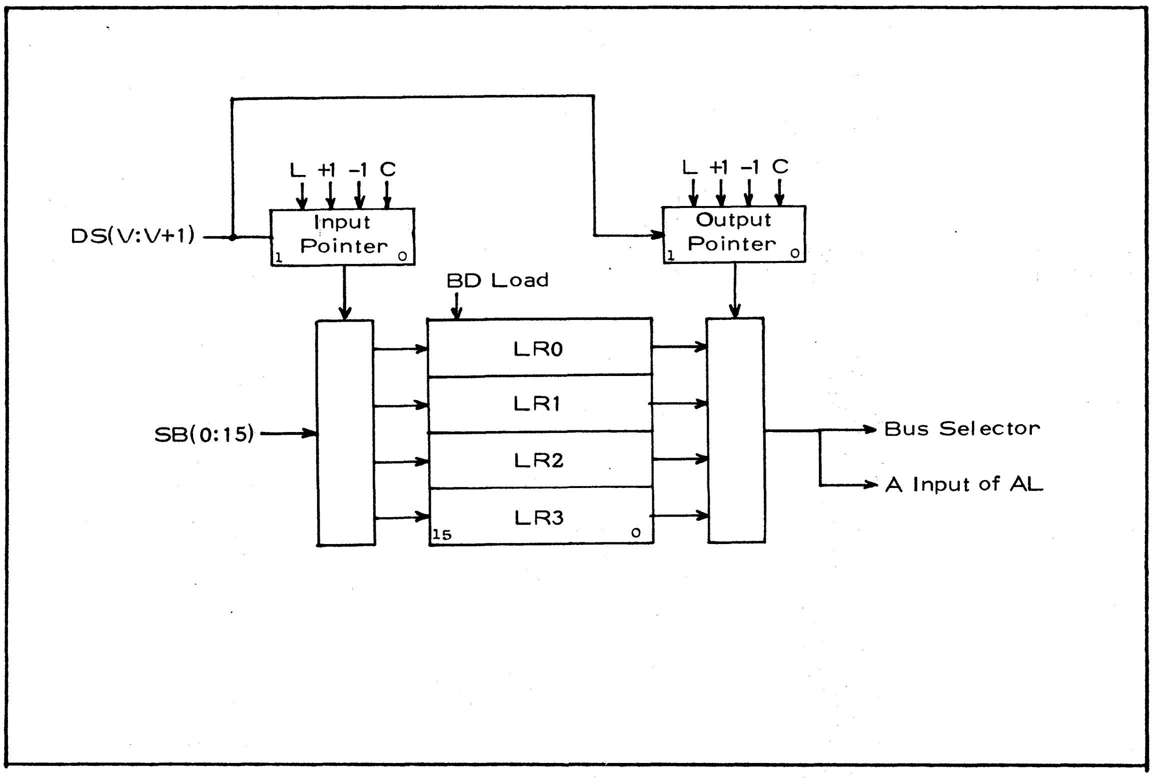
Symbolic Notation	Condition
AL	result of AL operation all 1's
AL(0)	bit 0 of the result of the AL operation
AL(15)	bit 15 of the result of the AL operation
ALOV	Al overflow (equivalent to a carry-out during addition and a borrow-in during subtraction)

Before giving examples of the control of the AL let us first discuss the nature of its inputs, A and B.

2.9. The Local Registers

The Local Registers, LR, serve as the A input to the AL in the context of the AL Functions shown in Table 2.10. The LR, shown in Figure 2.13, are 4 16-bit wide registers which have independent input and output pointers. The input pointer, LRIP, points to a LR which can be used as a BD for the current bus transport. The output pointer, LROP, points to a LR which can be used as either the A input to the AL or as the SOURCE for the current bus transport.

Figure 2. 13.
Local Registers, LR



Both the LR input pointer, LRIP, and the LR output pointer, LROP, are incrementable, decrementable, clearable, and loadable with two bits from the Double Shifter, DS(V:V+1), see Section 2. 12. The utility of this last feature will be demonstrated with examples when the Double Shifter is introduced. Table 2. 12. gives the microoperations associated with the control of the LR.

Table 2.12.
Microoperations for control of the LR

LRIPC
LRIP + 1
LRIP - 1
LRIP:=DS(V:V+1)
LROPC
LROP + 1
LROP - 1
LROP:=DS(V:V+1)
LRPC
LRP + 1
LRP - 1
LRP:=DS(V:V+1)

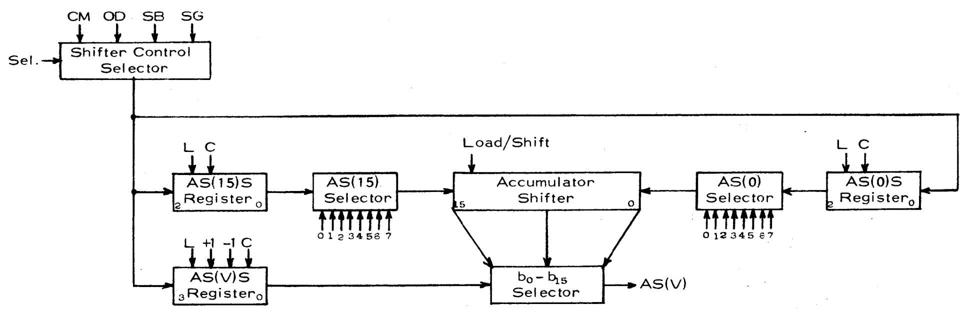
The last four microoperations allow for the clearing, incrementing, decrementing, and loading of both the IP and the OP simultaneously.

2.10. The Accumulator Shifter

The Accumulator Shifter, AS, serves as the B input to the AL in the context of the AL functions shown in Table 2.10. The AS can serve as a bus DESTINATION; but to be read, its contents must be gated through the AL with the ALF set to B. The AS, shown in Figure 2.14., is a 1-bit shifter which can shift left, shift right, be loaded, or remain idle during the execution of any given microinstruction.

Figure 2. 14.

Accumulator Shifter, AS



Source nc.	AS(15) Input	AS(0) Input
0	0	0
1	1	1
2	AS(0)	AS(15)
3	AS(15)	BUS(15)
4	Undef	SB(15)
5	DS(V+1)	DS(V+1)
6	AS(V)	AS(V)
7	VS(V)	VS(V)

There are 2 interesting features of this shifter:

a) its variable width characteristic and

b) its connection to other elements of the system.

The features are discussed in the following:

a) Although the shifter is 16-bits wide it may, in connection with either the BM or PM, be viewed as being m-bits wide ($1 \leq m \leq 16$). This is accomplished by having each of the 16 bits of the AS input to a selector (labelled the b_0 - b_{15} selector in Figure 2. 14). The output of this selector (called the variable bit, V) can then be a possible input into either the left or right end of the shifter, depending upon what particular type of shift one requires. When the AS is selected as a source for bus transport by gating it through the AL, after the desired shift has occurred, the bits not considered to be a part of the shifter must be masked off. This can be done either by using the BM or the PM. The width of the shifter is then determined by the contents of the AS(V) Selection register, AS(V)S, as shown in the above figure and the use of of an appropriate mask.

The AS(V)S can be loaded by the following microoperation

AS(V)S:=CM|OD|SB|SG.

Thus, for example, if we wish to consider the AS as a 12 bit left cyclic shifter, we would execute the microoperation

AS(V)S:=11

while making sure that AS(V) be used as the input to bit AS(0) during the shift operation. Subsequent use of the AS as a source could be accompanied by use of the PG masking off bits b_{15} - b_{12}, e.g.

```
        ; SET ALF B .
WA:=AL; PG→4
```

■

b) In Figure 2.14. it is seen that bits AS(0) and AS(15) can be filled by one of a variety of sources during a shift operation. Which source is to be used to fill the vacated bit position is determined by the contents of the AS(0) and AS(15) source selection registers, AS(0)S and AS(15)S respectively. An examination of the table in Figure 2.14. shows that the AS can be considered a logical shifter, a 1's fill shifter, a cyclic shifter, and a right arithmetic shifter. It can also be connected to another 1 bit shifter, called the variable width shifter, VS, to yield a long variable width shifter. It can be connected to a 2-bit shifter called the Double Shifter, DS, so it can be used in the merging of 2 bit streams into 1 or the diverging of 1 bit stream into 2. It can also be connected to the BUS and SB. These latter input is of an experimental nature and uses will be demonstrated in later examples.

Thus to use the AS, one must load the AS(V)S to set the width of the shifter and must load either the AS(0)S or AS(15)S to point to the source to be used as the input into the vacated bit position, i. e., one must set what the type of shift is, e. g., logical, 1's fill, long, etc. That both of these operations need not be done each time the shifter is used, but only when one is "changing" the width or type of shifter is obvious. Table 2. 13. lists the microoperations associated with the control of the AS. Note the AS can be set to a logical left, ASLL, or logical right, ASLR, shift.

Table 2. 13.
Microoperations for control of the AS

AS(0)S :=CM|OD|SB|SG	
AS(15)S:=CM|OD|SB|SG	
AS(V)S :=CM|OD|SB|SG	
ASLL	(≡ AS(0)SC)
ASLR	(≡ AS(15)SC)
AS(V)SC	
AS(V)S+1	
AS(V)S-1	

There are 2 bits in each microinstruction which control the operation of the AS: shift left, AS←, shift right, AS→, load, i. e., AS:=SB(0:15), or be idle. When the AS is to be shifted, the operation is put in the "microoperation and data" field of the microinstruction; when the AS is to be loaded, the operation is specified in the "bus transport" field of the microinstruction. As an example, the microinstruction

WA:=AL; AS← .

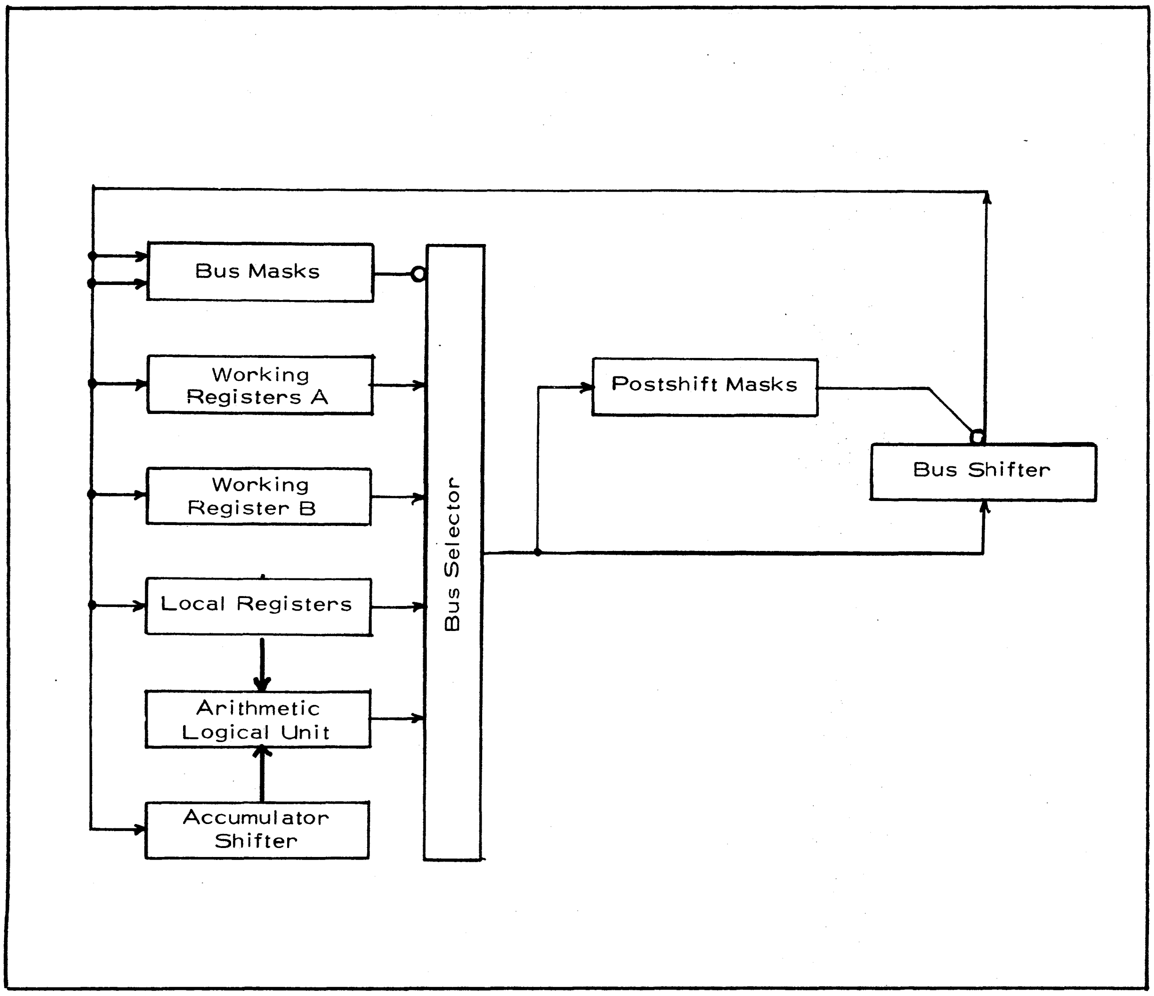
stores the output of the AL in a WA register and then shifts the AS left, while the microinstruction

LR, AS:=WB; WBP + 1 .

stores a WB in both the AS and a LR and then increments the WB pointer. If the AS is not employed during a given microinstruction, it does not appear in the specification of that microinstruction.

Having introduced the AL and its inputs, LR and AS, we now have knowledge of the expanded bus structure as shown in Figure 2.15.

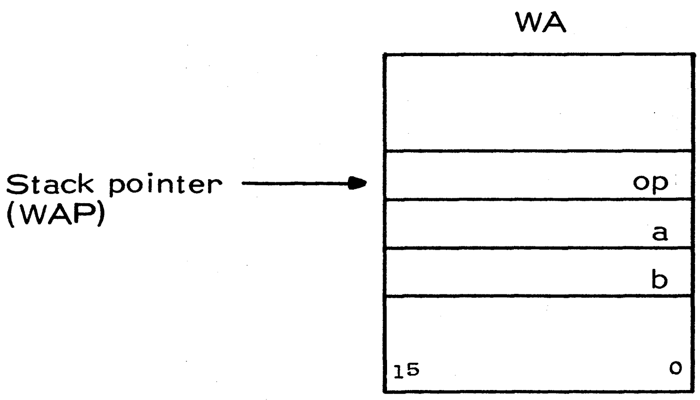
Figure 2.15.
Expanded Bus Structure



Let us now give a few examples using these resources to demonstrate the use of their associated microoperations.

Example 1

Let us consider WA as a stack as shown below

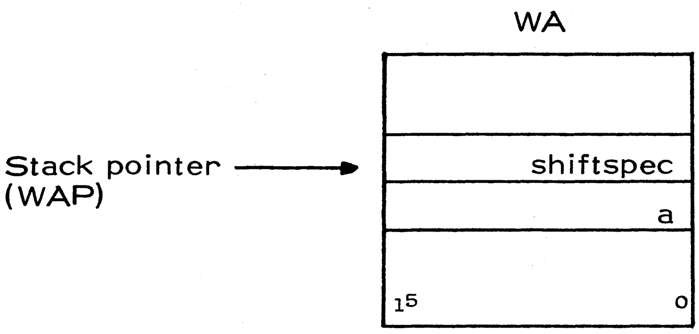


We wish to take two operands, a and b, and an arithmetical or logical operator, op, from the stack and place a op b on the new top of stack. The following microinstruction sequence does this.

```
WA     ; ALF:=SB, WAP- 1, LRPC.
LR:=WA; WAP - 1 .
AS:=WA.
WA:=AL .
```

Example 2

Let us again consider WA as a stack.



We wish to treat the AS as a left shifter whose characteristics are given by shiftspec. We wish to shift a n-times and return the result to the new top of stack after removing shiftspec and a. Let us assume shiftspec to have the following format.

$_{15}$ n $_{12}$	$_{11}$pgnsk$_7$	$_6$width$_3$	$_2$type$_0$

where

type	=	encoding found in the table of Figure 2.14 for logical, cyclic, etc. shift,
width	=	width of shifter – 1, 1 ≤ width of shifter ≤ 16
pgmsk	=	PG mask specification,
n	=	number of shifts – 1, 1 ≤ number of shifts ≤ 16

The following microinstructions execute the desired operation.

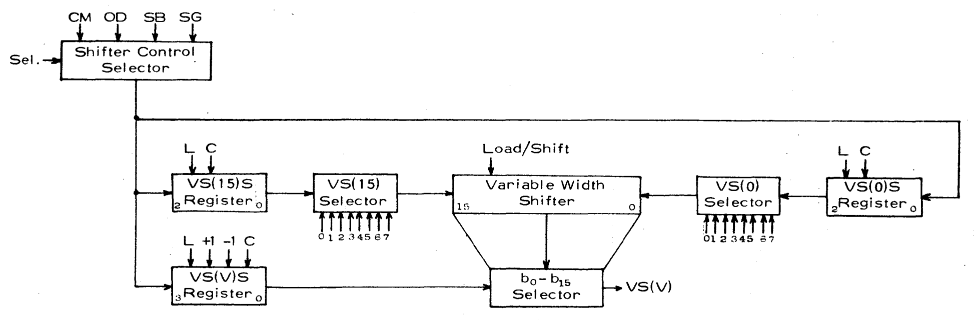
```
WA      ; AS(0)S:=SB.
WA, → 3; AS(V)S:=SB.
WA, → 7; PGSG:=SB.
WA, →12; CA:=SB, WAP + 1.
AS:=WA; PGS:=SG, PAP + 1, SET ALF B.        [illegible]
        ; AS← ; if CA then HERE+1 else HERE.   , CA-1
WA:=AL ; PAP-1, PGS:=CM.                      ■
```

2.11. The Variable Width Shifter

The Variable Width Shifter, VS, is a shifter functionally identical to the AS. The reason one is called the Accumulator Shifter is that not only does it serve as an input to the AL, but also it will serve as the accumulator required in the realization of the basic arithmetic operations (e.g. multiplication). The VS can be a SOURCE or DESTINATION for a bus transport. It is shown in Figure 2.16.

Figure 2. 16.

Variable Width Shifter, VS



Source no.	VS(15) Input	VS(0) Input
0	0	0
1	1	1
2	VS(0)	VS(15)
3	VS(15)	BUS(14)
4	Undef	SB(14)
5	DS(V)	DS(V)
6	VS(V)	VS(V)
7	AS(V)	AS(V)

The microoperations associated with the VS are identical to those associated with the AS and are listed below in Table 2. 14.

Table 2. 14.

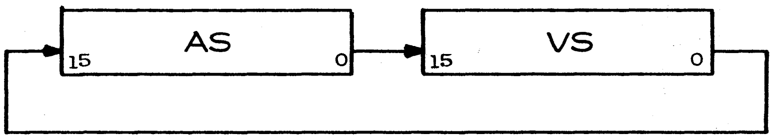
Microoperations for control of the VS

```
VS(0)S  :=CM|OD|SB|SG
VS(15)S:=CM|OD|SB|SG
VS(V)S  :=CM|OD|SB|SG
VSLL       ( ≡ VS(0)SC)
VSLR       ( ≡ VS(15)SC)
VS(V)SC
VS(V)S + 1
VS(V)S - 1
```

One of the important features of the AS and VS, as seen from the tables in Figures 2.14. and 2.16., is that they can be connected together. This allows, for example, the AS and VS to be viewed as a "long" shifter when coupled together. The microinstructions,

```
; AS(15):=7, VS(15):=7.
; AS(V)SC, VS(V)SC.                    ■
```

connect the AS and VS together so that they can be viewed as a right cyclic 32-bit shifter as shown below.



Just as with the AS, there are 2 bits in each microinstruction which control the operation of the VS: shift left, VS←, shift right, VS→, load, i.e., VS:=SB(0:15), or remain idle.

Assuming the previous AS/VS connection has been made, subsequent execution of the microoperations

AS→, VS→

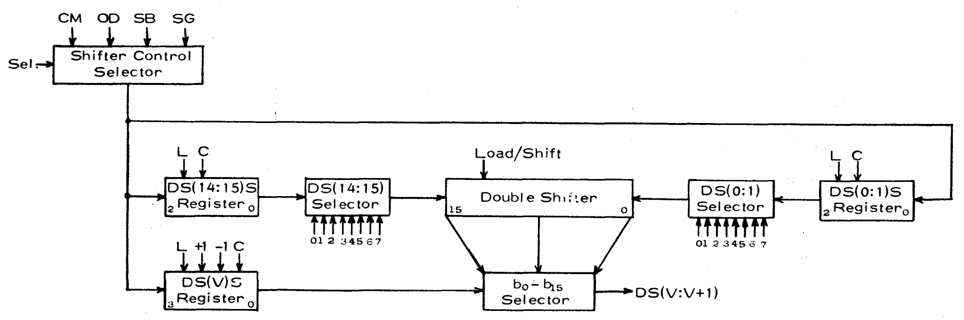
shifts this 32-bit shifter 1 bit right cyclic. Other "long shifters", e.g. left logical, right logical, right arithmetic, etc., result from appropriate set up sequences.

2.12. Double Shifter

The Double Shifter, DS, is a shifter with functional characteristics similar to those of the AS and VS, except that it shifts 2 bits at a time and not 1. Bits DS(0) and DS(1) require input during a left shift and DS(14) and DS(15) require input during a right shift. The DS is shown in Figure 2.17. The DS can be a SOURCE for or a DESTINATION of a bus transport.

Figure 2. 17.

Double Shifter, DS



Source	Inputs DS(15)	Inputs DS(14)	Inputs DS(1)	Inputs DS(0)
0	0	0	0	0
1	1	1	1	1
2	DS(1)	DS(0)	DS(15)	DS(14)
3	DS(15)	DS(15)	BUS(15)	BUS(14)
4	Undef	Undef	SB(15)	SB(14)
5	DS(V+1)	DS(V)	DS(V+1)	DS(V)
6	AS(V)	VS(V)	AS(V)	VS(V)
7	BUS(1)	BUS(0)	Undef	Undef

The microoperations which are associated with the DS are directly comparable to those for the AS or VS and are shown in Table 2. 15.

Table 2. 15.

Microoperations for control of the DS

DS(0:1)S	:=CM|OD|SB|SG
DS(14:15)S	:=CM|OD|SB|SG
DS(V)S	:=CM|OD|SB|SG
DSLL	(≡ DS(0:1)SC)
DSLR	(≡ DS(14:15)SC)
DS(V)SC	
DS(V)S + 1	
DS(V)S - 1	

The b_0 - b_{15} selector specifies two bits DS(V:V+1) as output, these may be used in coupling the shifters, or as input to the LRIP and LROP-pointers.

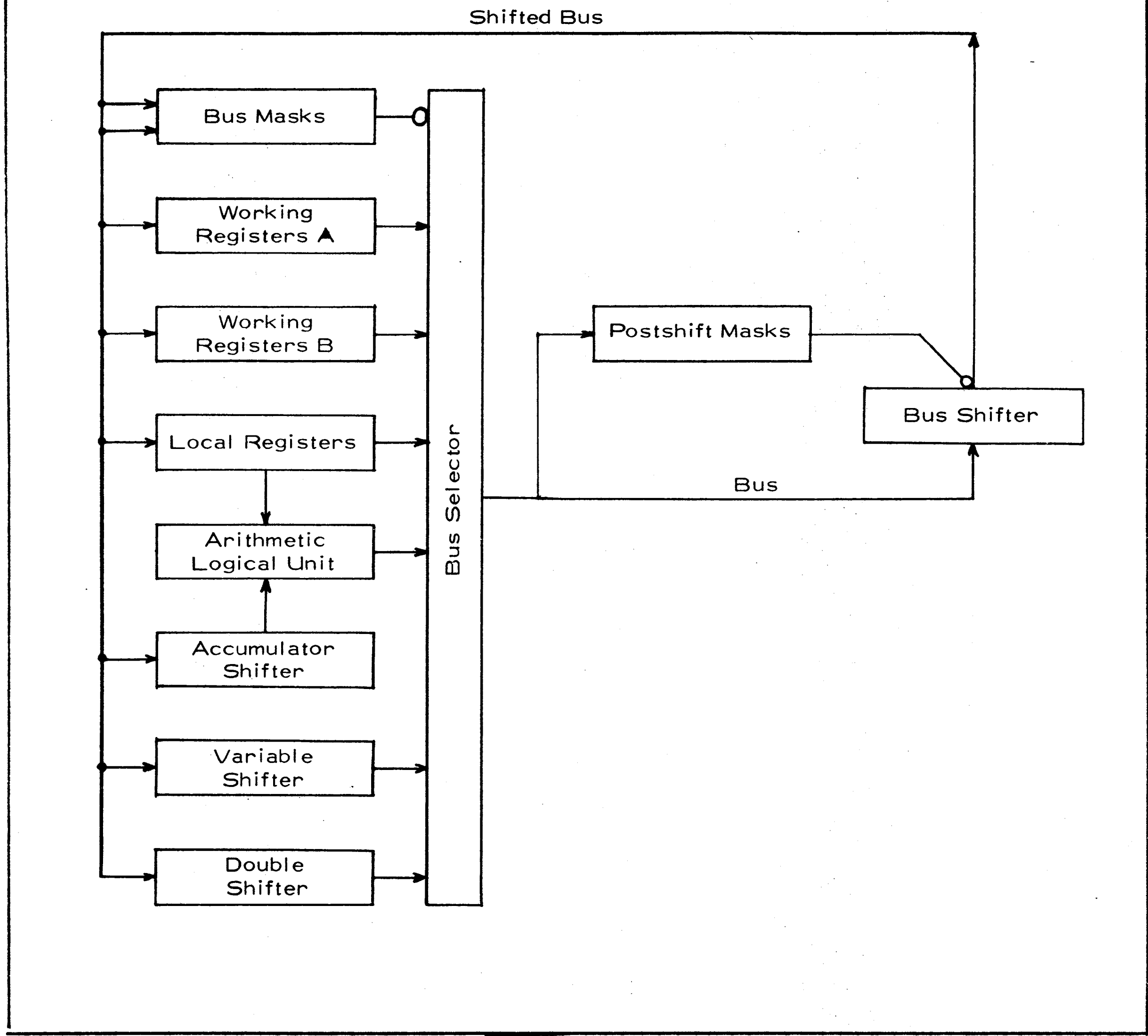
There are 2 bits in each microinstruction which control the operation of the DS: shift left, DS←, shift right, DS→, load i. e., DS:=SB(0:15), or remain idle.

2. 12. 1. Two examples using the shifters

The AS, VS, and DS are collectively referred to as the "Shifters" whereas the Bus Shifters are not included in this term. The expanded bus structure is shown in Figure 2. 18.

Figure 2. 18.

Expanded Bus Structure



Example 1

Suppose we wish to count the number of bits which are set to 1 in the WA register pointed to by WAP and leave this number in the same cell. The following algorithm will do this

a) Load the LR with the following constants

LR0:=0
LR1:=1
LR2:=1
LR2:=2

b) Clear the AS (considered here as an accumulator)

c) Set the AL to addition

d) Transfer the data to the DS

e) Do the following 8 times and then do f)

```
  i)  if DS(0:1) ≡ 00 then accumulate LR0 + AS  in AS
      if DS(0:1) ≡ 01 then accumulate LR1 + AS  in AS
      if DS(0:1) ≡ 10 then accumulate LR2 + AS  in AS
      if DS(0:1) ≡ 11 then accumulate LR3 + AS  in AS
 ii)  shift DS→
```

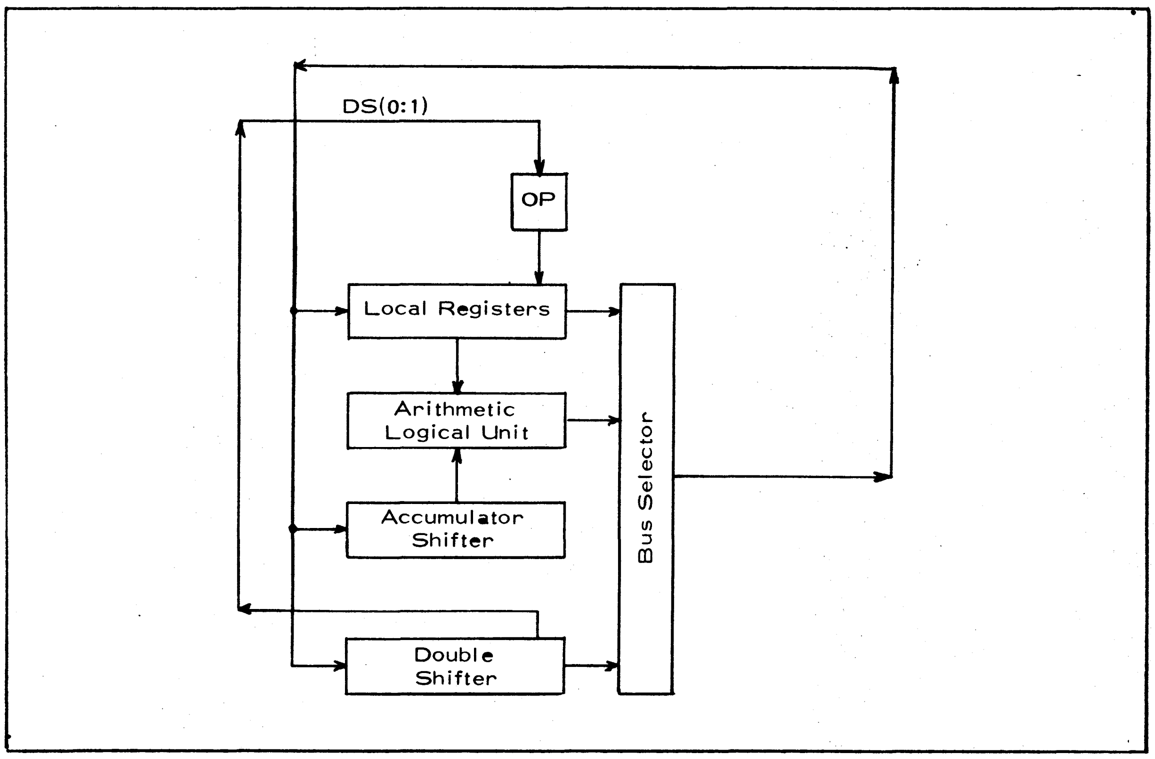
f) Store the accumulated result which is in AS

The following microinstruction sequence accomplishes this. It is assumed the PG data source is the CM.

```
DS:=WA      ; ALF:=all 0 s, LRPC.
AS, LR:=AL; ALF:=A+1; LRIP+1.              LR0:=0
LR:=AL      ; LRP+1, CA=7.                 LR1:=1
LR:=LR      ; LRP+1, DS(V)SC.              LR2:=1
LR:=A~~R,~~L  ; ALF:=A+B, LROP:=DS.          LR3:=2
AS:=AL      ; CA - 1, DS→2. LROP:=DS;
              if CA then HERE+1 else HERE
WA:=AL      .                                      ■
```

The subset of the bus which is used during the counting loop instruction (AS:=AL) is shown in Figure 2.19. This may help in understanding the algorithm and code.

Figure 2.19.
Counting Loop for Counting Number of Bits set to 1 in a Word

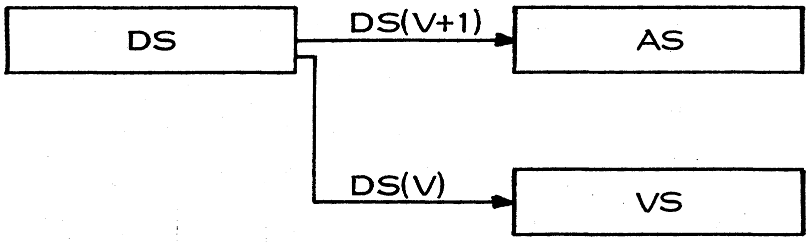


Example 2

Consider the contents of the current WA register as a string of 16 bits. It is desired to pack all of the even numbered bits (b_0, b_2, etc.) in the left 8 bits of the current WB register and the odd numbered bits (b_1, b_3, etc.) in the right 8 bits of this register so that the result appears as

b_{15} . . . . . b_5 b_3 b_1	b_{14} . . . . . b_4 b_2 b_0

Because the DS, AS, and VS can be connected as shown below,



one can accomplish the stated requirement in the following way:

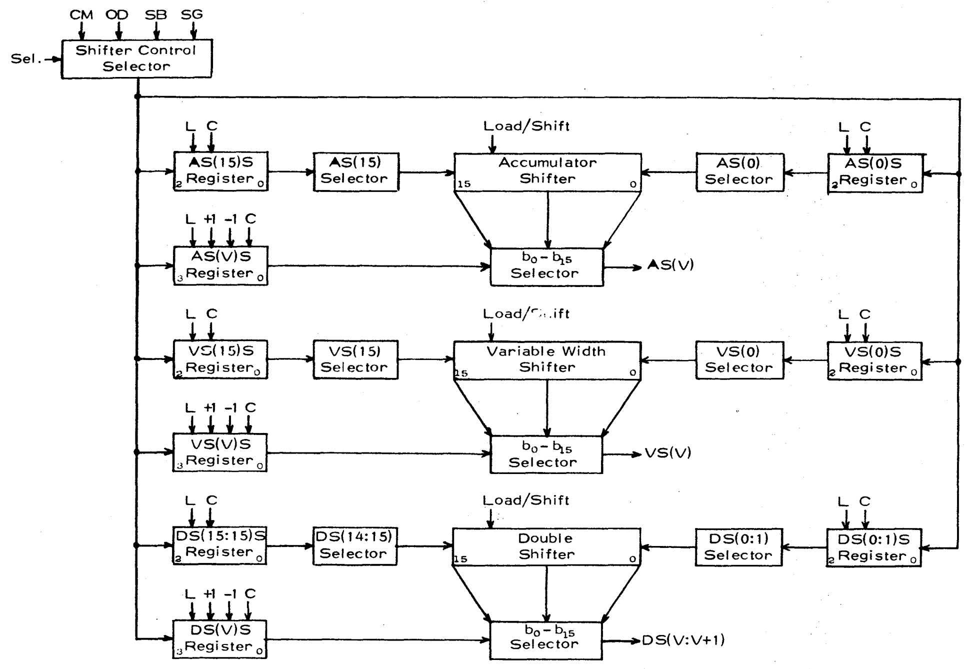
```
             ; ALF:=all 0 s, LRPC.
AS, VS:=AL;  AS(15):=5, VS(15):=5, DS(V)SC.
DS:=WA     ; CA:=7.
           ; CA - 1, AS→, VS→, DS→; if CA then HERE+1
             else HERE.
LR:=VS, →8 ; ALF:=A ∨ B.
WB:=AL     .
```

■

2.13. The Common Shifter Standard Group

The Shifter Control Selector shown in Figures 2.14., 2.16. and 2.17. is the same selector. This is, perhaps, made a bit clearer in Figure 2.20.

Figure 2.20.
AS, VS, and DS Control



The SG which is associated with this selector is called the Common Shifter SG. Various shifter control data can be stored in this SG for various shifter interconnections and then used in environment prologues. The microoperations associated with the CS SG are shown in Table 2.16.

Table 2. 16.
Microoperations for control of the CS SG

CSP :=CM|OD|S1|S2
CSP + 1
CSP - 1
CSPC
CSS1 :=CM|OD|S1|S2
CSS2 :=CSP
CSSG:=SB

in addition there are several microoperations which allow control of the AS, VS, and DS to be executed in parallel. These are shown in Table 2. 17.

Table 2. 17.
Parallel CS Microoperations

Notation	Microoperation
CSLL	Set AS, VS, DS to logical left shift
CSLR	Set AS, VS, DS to logical right shift
CS(0)S :=CM|OD|SB|SG	Load AS(0), VS(0), and DS(0:1) Source register from CM|OD|SB|SG
CS(15)S:=CM|OD|SB|SG	Load AS(15), VS(15), and DS(14:15) Source register from CM|OD|SB|SG
CS(V)S :=CM|OD|SB|SG	Load AS(V), VS(V), and DS(V) Selection register from CM|OD|SB|SG
CS(V)SC	Clear AS(V), VS(V), and DS(V) Selector register

2.14. Loading Masks

Associated with WA there is a SG of loading masks called Loading Masks, A, LA. Associated with WB there is a SG of loading masks called Loading Masks B, LB. In what follows we will describe only LA; LB is identical in function. The purpose of the loading masks, LA and LB, is to be able to specify which bit positions in a working register WA can be loaded as the result of WA being chosen as the DESTINATION of a bus transport while leaving the nonspecified bits unchanged. As an example, if the loading mask

```
| 0 .......... 00 111111   |
|15              6 5      0|
```

were pointed at by the LA pointer, LAP, then, when the bus transport

W~~B~~A:=AL

is executed, bits SB(0:5) would be gated into the WA register pointed to by W~~B~~AP in bit positions b_0 through b_5 respectively while bits b_6 through b_{15} would not change their value. When WA is selected as a SOURCE for bus transport the mask LA acts in the following fashion: if bit i ($0 \leq i \leq 15$) of the mask is a 1, then bit i of WA is transmitted. If bit i of the mask is a 0, then bit i which is transmitted is indeterminate.

As an example if the loading mask

```
|15 0 .......... 00 111111 0|
|                 6 5       |
```

were pointed at by the LA pointer, LAP, then, when the bus transport

WB:=WA

is executed, bits SB(0:5) would be gated into the WB register pointed to by WBP in bit positions b_0 through b_5 respectively while b_6 through b_{15} would be indeterminate.

The relationship between the loading masks and the working registers is represented by the symbol ——$\textcircled{\ell}$ where the script ℓ in the mask notation ——$\textcircled{\ell}$ indicates the special nature of these masks. Figure 2.21. shows the expanded bus structure with the loading masks added.

Figure 2.21.
Expanded Bus Structure

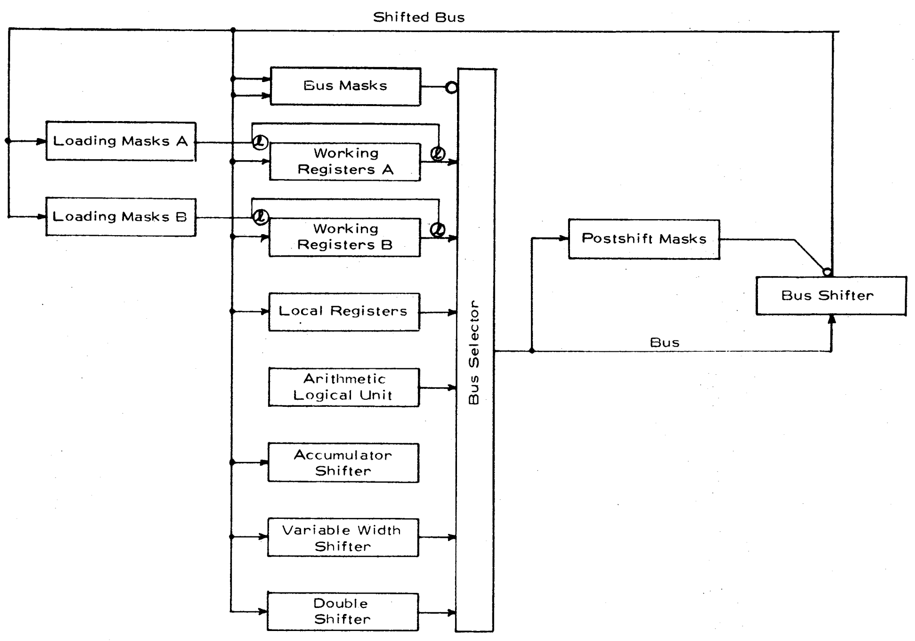
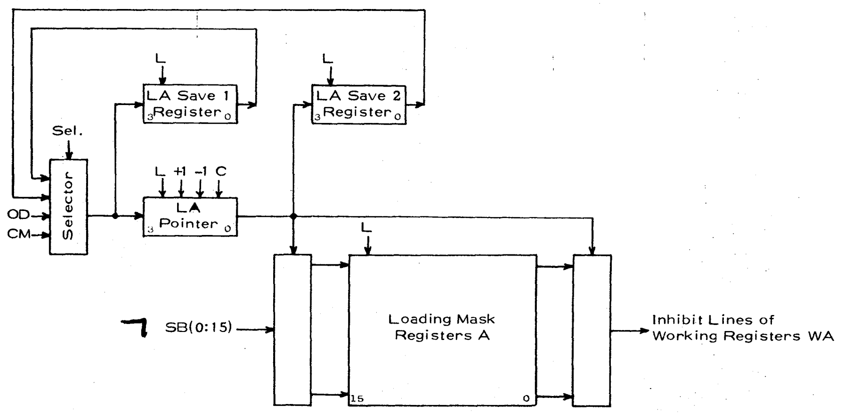


Figure 2.22. shows a more detailed sketch of LA; LB, not shown, is identical.

Figure 2. 22.

Loading Mask Registers A, LA



There are 7 microoperations shown in Figure 2. 22. associated with the use of LA. These are listed along with the corresponding microoperations for LB in symbolic form in Table 2. 18.

Table 2. 18.

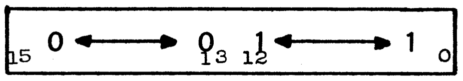
Microoperations for control of LA and LB

LA :=SB	LB :=SB
LAP :=CM|OD1|S1|S2	LBP :=CM|OD3|S1|S2
LAP + 1	LBP + 1
LAP - 1	LBP - 1
LAPC	LBPC
LAS1:=CM|OD1|S1|S2	LBS1:=CM|OD3|S1|S2
LAS2:=LAP	LBS2:=LBP

NB!	OD1	means	OD(0:3)
	OD3	means	OD(8:11)

Upon the dead start, the system is such that the "full load" and "full read out" mask, i. e., 16 1's is in register 0 of LA and register 0 of LB. We will assume this to be the case throughout normal operation of the system. One can then look upon the pointers LAP and LBP as selection switch for the use of the loading masks. If LAP=0 then no loading mask is applied to WA, if LAP $\neq$ 0 then WA is masked by the mask specified by LAP; similar statement can be made for LBP. This is, of course, not the only interpretation of the use of the loading masks, but it is a convenient one and one which we will normally employ unless otherwise stated. When you load LA (or LB) from SB you actually get ¬ SB (i. e. the inverted SB) into LA (or LB).

As an example, suppose we wish to place the high order 13 bits of the output of the DS into the least 13 bits of WB0 leaving the high order 3 bits the same. If the mask



is in LB9, the following microinstruction sequence accomplishes this:

```
                ; LBP:=9, WBPC
      WB:=DS, →3; LBPC.                               ■
```

This mask could have been generated by use of the PG and AL. The code, (remember that we have to generate the inverted mask)

```
                ; ALF:=all 1 s, LBP:=9
                ; PGS:=CM, PAP + 1.
      AL        ; PG←13, LB:=SB, PAP-1.               ■
```

generates the mask and stores it in LB9. It should be reasonably obvious now how the loading masks can be used to store the result of various data transformations as they are determined, e. g., in the implementation of signed-magnitude arithmetic, the magnitude of the exponent, its sign, the magnitude of the coefficient and its sign can be stored in a given word as they are obtained.

We will henceforth assume in all examples (unless explicitly stated otherwise) that LAP = 0 and LBP = 0, i. e., that no loading masks are applied to either set of working registers. If a particular code segment uses the loading mask facility it is responsible for leaving the system operating in this fashion. The treatment of the loading masks then becomes quite identical with that of the bus masks and postshift masks as stated in Section 2. 7.

2. 15. The Parity Generator

The parity generator is a circuit which determines the parity of the 16 bits which compose the bus transport. It posts the result of this evaluation as a testable condition, the bus parity, BP, condition. If BP = 1, the BUS is odd parity; if BP = 0, the BUS is of even parity. This condition can be used, obviously, in any processing wherein parity information is variable, e. g., in communicating with devices which transmit words of a particular parity. The parity generator functions during each bus transport and has no microoperations associated with it. Since its input is the BUS, we show it attached to the bus structure.

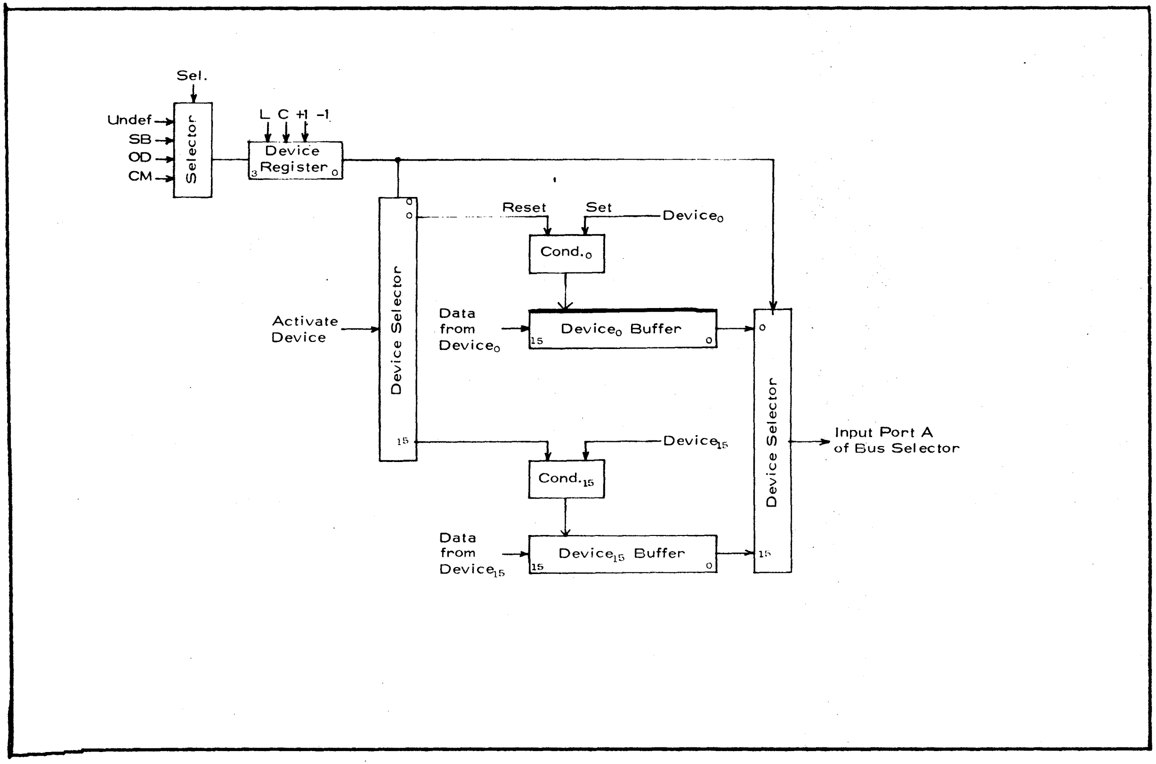
2. 16. The Bit Encoder

The RIKKE 1 is prepared for a Bit-encoder, BE, but this is not implemented. In those places where BE has been used the value will be undefined. For a detailed description se Shriver [7].

2.17. Input Ports

There are two input ports through which external devices may be connected to the bus selector. They are called Input Port A, IA, and Input Port B, IB. Up to 16 devices can be connected to each of these input ports. IA is shown in Figure 2.23.; IB, not shown, is identical.

Figure 2.23.
Input Port A, IA



The particular device which is selected by the IAD register to be read is pointed to by a Device Register. There are two conditions associated with a selected device:

a) Data available, IADA, and
b) Mark-bit set , IADM

All devices must be able to set the first condition. The second condition can be set by devices which can transmit two different sorts of information, for example control data and information. When a device is activated the IADA condition is reset. The microoperations associated with the control of IA and IB are given in Table 2.19.

Table 2. 19.

Microoperations for control of IA and IB

IAA	Activate Port, i. e., read IA
IAD:=CM|OD|SB*)	Load IA Device Register from CM|OD|SB
IADC	Clear IA Device Register
IAD + 1	Increment IA Device Register
IAD - 1	Decrement IA Device Register
IBA	Activate Port, i. e., read IB
IBD:=CM|OD|SB*)	Load IB Device Register from CM|OD|SB
IBDC	Clear IB Device Register
IBD + 1	Increment IB Device Register
IBD - 1	Decrement IB Device Register

As an example, if we wish to read a piece of data from device 9 on IB and store it in AS, we can write the following classical wait loop:

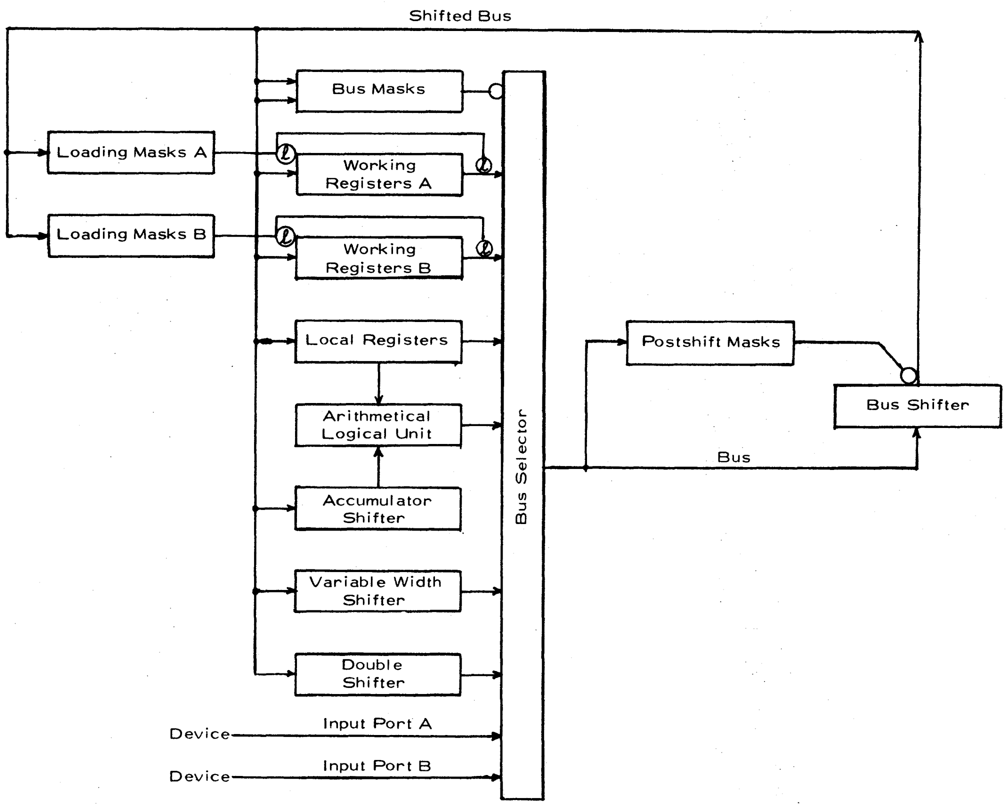
```
      ; IBD:=9, IBA.
AS:=IB; if IBDA then HERE+1 else HERE.
```
■

The expanded bus structure can now be shown as Figure 2.24.

*) The value of the fourth input is undefined

Figure 2. 24.

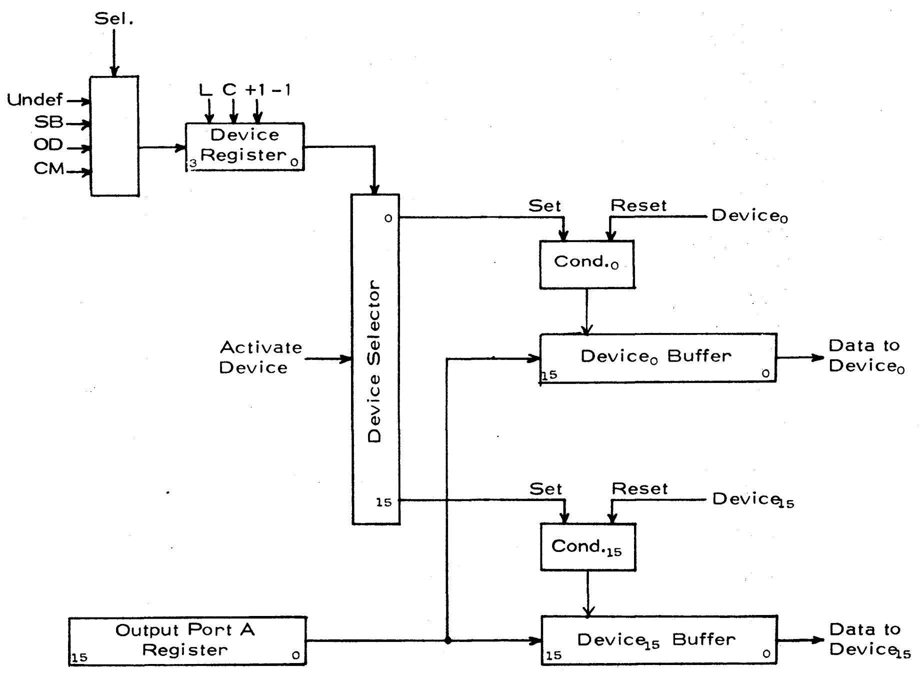
Expanded Bus Structure



2.18. Output Ports

There are four output ports through which output to external devices may occur. They are called Output Ports A, B, C, and D; OA, OB, OC, and OD respectively. They are identical in operation with the exception that OA and OB are loaded from the SB and can be selected as bus DESTINATIONS whereas OC and OD are loaded from the BUS and cannot be selected as bus DESTINATIONS, but must be loaded by a microoperation. OA is shown in Figure 2.25; OB, OC, and OD, not shown, are identical.

Figure 2.25.
Output Port A, OA



The particular device which is selected for output is pointed to by a device register. There is a condition associated with a selected device: space available, OASA. The microoperations associated with the control of OA and OC are shown in Table 2.20. The microoperations for OB are identical to those for OA and the microoperations for OD are identical to those for OC.

Table 2.20.
Microoperations for control of OA and OC

OAA	Activate Port, i.e. write OA
OAR	Reset condition on OA, selected device
OAD:=CM|OD|SB	Load OA register from CM|OD|SB
OADC	Clear OA Device Register
OAD + 1	Increment OA Device Register
OAD - 1	Decrement OA Device Register
OCA	Activate Port, i.e. write OC
OCR	Reset condition on OC, selected Device
OCD:=CM|OD|SB	Load OC register from CM|OD|SB
OCDC	Clear OC Device Register
OCD + 1	Increment OC Device Register
OCD - 1	Decrement OC Device Register

Table 2.21.
Microoperations for loading of OC and OD

OC:=BUS	Load Output port C from the Bus
OD:=BUS	Load Output port C from the Bus

As an example, suppose we wish to write out the output of the AL onto device 13 of output port C. We could then write,

```
AL; OC:=BUS, OCD:=13.
  ; if OCSA then HERE+1 else HERE .
  ; OCA.                                    ■
```

There is one additional feature associated with the "activate" microopetion. Recall that on the input ports it is possible to test a mark bit which is set by a device. Analogous with this, it is possible on output to write out an extra mark bit in addition to the data. The device can, for example, treat this extra bit as a selector between two different modes of operations. The microoperations for output port activate are now given by

OAA1	activate with mark bit set to 1
OAA0	activate with mark bit set to 0
OAA	activate with mark bit undefined.

Special purpose output Port D.

The Output Port D is dedicated for control, so far we have used the mnemonic OD in a lot of the selectors (f. ex. in the BS standard group). This means that all these units can be controlled from Output Port D.

Notice that since the port has been dedicated, all operations on ODD, as well the operations ODA and ODR, has no effect. The only operation left with an effect is

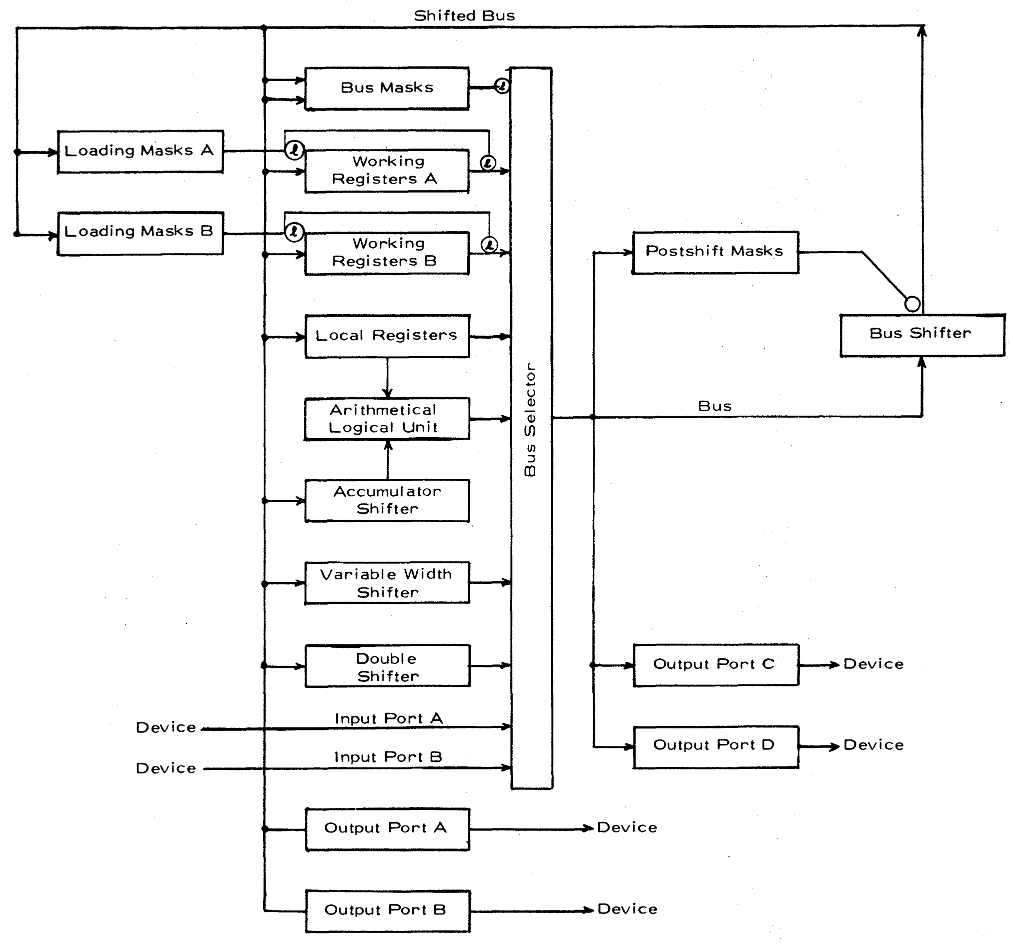
```
OD:=BUS
```

which will save the information on the Bus for subsequent use through one of the selectors.

2.19. The Bus Structure

With the introduction of the output ports in the previous section we have completed a description of (with only very minor modifications) the RIKKE-1 Bus Structure, the registers and functional units attached to it, and the control which can be exercised on these components. The Bus Structure is now shown in Figure 2.26.

Figure 2.26.
RIKKE-1 Bus Structure



Let us summarize some of the information with respect to bus SOURCE s and DESTINATION s. We have the following SOURCE s and DESTINATIONs for a bus transport:

a) <u>SOURCEs for Bus Transport</u>

WA
WB
LR
AL
VS
DS
IA
IB

b) <u>DESTINATIONS for 16-bit Load of SB with BD Load</u>

MA
MB
WA
WB
LR
OA
OB

c) <u>Shifters which can load 16-bit SB via dedicated bits in every microinstruction</u>

AS
VS
DS

Thus in the bus transport specification

LIST:=SOURCE,

the LIST can consist of at most 1 destination from (b) above and any list of the shifters, i. e.,

BD_b [,AS][,VS][,DS]:=SOURCE,

where the [ ] indicates the option of inclusion in the LIST.

Recall that the inverted SB can be loaded into LA and LB by execution of appropriate microoperations and, the inverted BUS can be loaded into PA, PB and the BUS into OC and OD by execution of appropriate micro-operations. Also, a subfield of the SB (normally a contiguous string starting with bit b_0) can be loaded into various SG's and control ports throughout the system by execution the appropriate microoperation. Thus, many parallel loads of both the BUS and the SB may occur in any given microinstruction.

There are three important restrictions on the above bus transport specifications:

a) the specifications WA:=WA or WB:=WB are not allowed,

b) the specification LR:=LR is only defined when LRIP $\neq$ LROP,

c) one cannot use a mask (MA, MB, PA, LA, LB) and load the register containing that mask in the same microinstruction.

d) it is not possible to shift in one of the shifters (AS, VS and DS) while loading the same shifter (these operations are mutually exclusive).

On the other hand the timing allows you shifting in one of the shifters AS, VS and DS while using it as the source of bustransport. This will <u>not</u> affect the transport, the shift will only change the old content of the shifter. (The shift takes place <u>after</u> the transport).

2.20. The Control Unit

The control unit of the RIKKE-1 system, shown in Figure 2.1. on page 5, consists of (1) a control store and (2) a microinstruction sequencing capability. The random access control store consists of up to 4.096 words of 64-bit wide, 80 nanosecond monolithic storage. The microinstruction sequencing is described below.

2. 20. 1. Microinstruction Sequencing

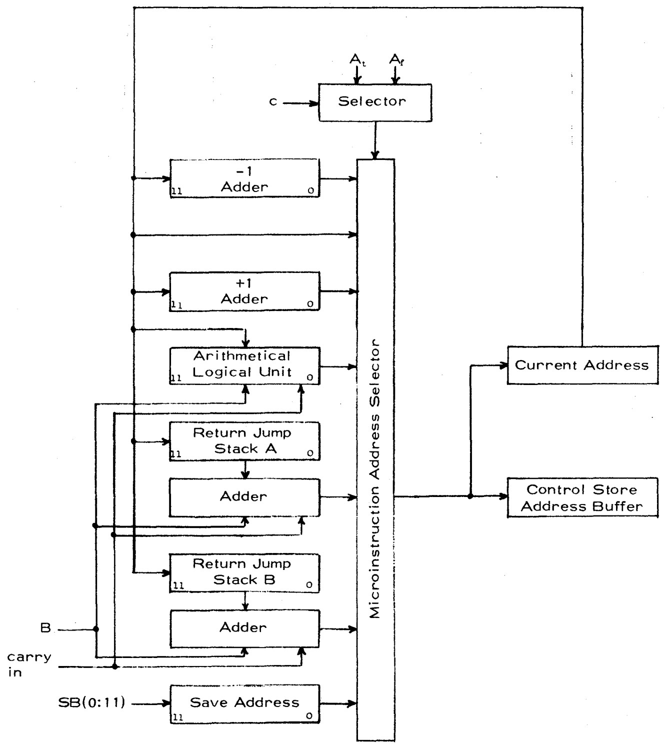
The microinstruction sequencing hardware is a physical embodiment of the "if c then A_t else A_f" clause we have been using in our microprogramming examples. This is accomplished in the following way. The addresses A_t and A_f are selected from 8 possible address sources. Let A be the address of the current microinstruction and let B be data which is specified in the current microinstruction. The 8 possible address sources, which are explained in more detail shortly, are listed in Table 2. 22.

Table 2. 22.
Microinstruction Address Sources

A - 1	Current address - 1
A	Current address
A + 1	Current address + 1
AL(A, B)	A function of A and B as computed by an arithmetical logical unit
RA + B	The contents of the top of a return jump stack, RA, added to B
RB + B	The contents of the top of a return jump stack, RB, added to B.
SA	The contents of the save address register, SA
	spare input (value is 4095)

These address sources are realized by providing a microinstruction address bus which is shown in a limited form in Figure 2. 27.

Figure 2. 27.
Microinstruction Address Bus (Preliminary)



One can see from this figure how the "if, then, else"-clause is realized. There are 3-bits in each microinstruction which specify one of the 8 address sources of Table 2.22. to be used as the true branch address, denoted A_t. There are 3-bits in each microinstruction which specify one of the 8 address sources of Table 2. 22. to be used as the false branch address, denoted A_f. There are 7 bits in each microinstruction used to specify 1 of 128 conditions which are testable in the system; the selected condition is denoted c. The state of the selected condition c determines which source, A_t or A_f, will be used to select the next microinstruction address source. If c = 1 then A_t will be used to select the address of the next microinstruction; if c = 0, then A_f will be used for this purpose. When a microinstruction address is selected, it is loaded into the Control Store Address Buffer so it can be used to fetch the microinstruction, and it is also loaded into the Current Address register so that it can be used in the next address computation, if required. The contents of the Current Address register has been used in previous examples under the symbolic name HERE.

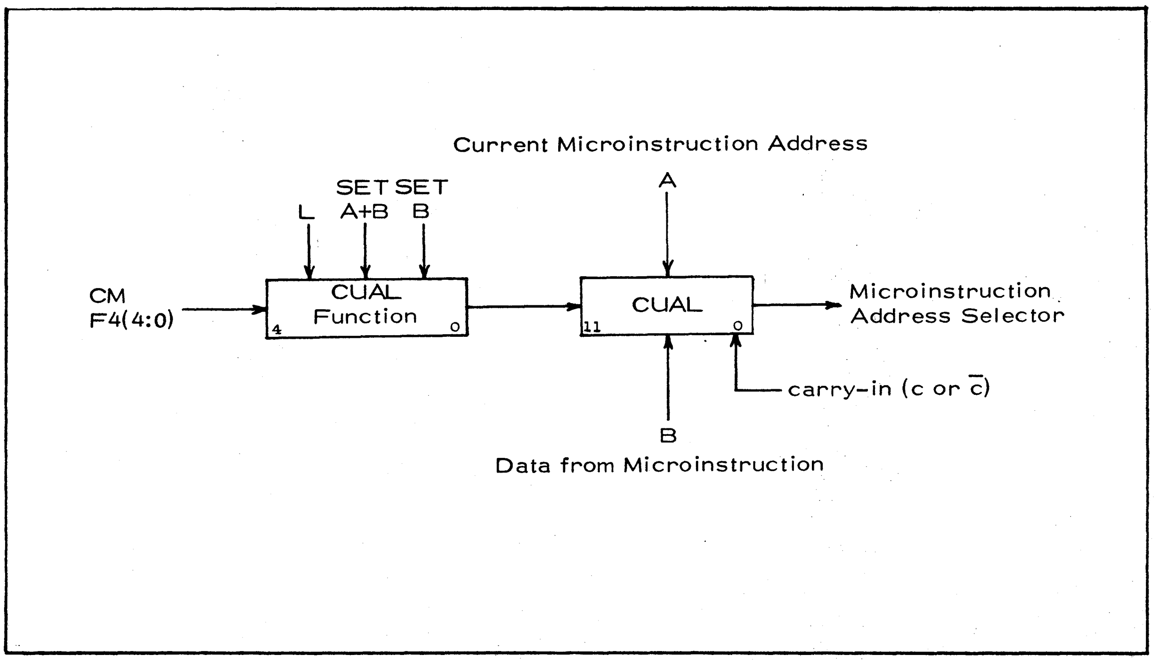
The address sources A - 1, A, and A + 1 are straight forward and need not be dealt with. It should be mentioned, however, that Control Store addresses are interpreted modulo the size of the Control Store. At the current version of RIKKE-1 the Control Store is 512 words, this implies that only the first 9 bits of the address are significant.

2.20.2. The Control Unit Arithmetical Logical Unit

The Control Unit Arithmetical Logical Unit, CUAL, is functionally identical to the arithmetical logical unit which is connected to the RIKKE-1 bus structure except that it is 12-bits wide and not 16-bits wide. The CUAL functions are identical to those of the AL and are given in Table 2. 10.The "A input" to these computations is the address of the current microinstruction and the "B input" is data specified in the current microinstruction. The CUAL is shown as in Figure 2.28.

Figure 2.28.

Control Unit Arithmetical Logical Unit


Current Microinstruction Address
A
SET SET
L A+B B
CM F4(4:0)
CUAL Function
4
0
CUAL
11
0
Microinstruction Address Selector
carry-in (c or $\bar{c}$)
B
Data from Microinstruction

First, note that the CUAL Function register can only be loaded from the CM, i. e., CUALF:=CM. One can set the CUALF to add A and B, i. e., SCUALF + and also to the logical function B, i. e., SCUALF B. These are the only three microoperations associated with the CUAL. Cnly 5 bits are used to specify the function; the carry-in, when required, is specified in another way. Let c denote the selected condition used to control the address selection and let $\bar{c}$ be its negation. There is a bit in each microinstruction, called the Carry-Input Selection Bit, CISB, which is used to determine the carry-in as shown in Table 2. 23.

Table 2. 23.

Carry-in Selection

CISB	Carry-in
0	$\bar{c}$
1	c

Example 1

Suppose the CUALF is set to A + B; this is a relative jump. If CISB=0 the specification

if c then CUAL else HERE

can be interpreted to mean:

if c then HERE + B else HERE.

Whereas, if CISB = 1, the specification can be interpreted to mean:

if c then HERE + B + 1 else HERE.

Example 2:

Suppose the CUALF is set to B; this is an absolute jump. This is a logical function and not affected by the carry-in.

if c then CUAL else CUAL

can be interpreted to mean:

if c then B else B.

In our microassembler, the specification of the CISB will be given implicitly. If one chooses the CUAL output as microinstruction address source, we write

CUAL + Carry-in.

Choice of this specification as either an A_t or A_f will dictate the setting of the CISB.

For the first interpretation of Example 1 to be valid the specification would have to be written

if c then CUAL else HERE

whereas if we meant the second interpretation we would have to write

if c then CUAL + 1 else HERE.

It should be obvious that the specification

if c then CUAL + 1 else CUAL + 1

is an example of a microinstruction sequencing specification which is imcompatible with the specification capability described above. Indeed if one wished to choose the address specification CUAL + 1 irrespective of condition, one merely need write

CUAL + 1

in the microinstruction sequencing field of the microinstruction. This would have the same effect as writing, for example,

if TRUE then CUAL + 1 else CUAL.

where TRUE is a manifest system constant set to 1. There is also a manifest system constant, FALSE which always has the value 0.

In order to complete the discussion of the CUAL we must discuss the specification of the data B. There are two 6-bit fields in the microinstruction which we shall call T and t . T and t are input into a function box which makes the computations shown in Table 2. 24. There are 2 bits in every microinstruction, called the B-Input Selection Bits, BISB, which determine which of these computations will be used as the B data, if required, in the current address computation.

Table 2.24.
B data Selection

BISB	B data
00	0
01	$t_{sign}t$
10	T0
11	Tt

The notation $t_{sign}t$ means the 12 address bits are given by

$$t_5\ t_5\ t_5\ t_5\ t_5\ t_5\ t_5\ t_4\ t_3\ t_2\ t_1\ t_0\ ,$$

i.e., in "sign extended" form. With the CUALF set to A + B and BISB = 01 we then have a relative addressing capability of ± 32. The notations Tt and T0 denote concatenation.

In our microassembler, the specification of the BISB will be given implicitly. One specifies the B value explicitly as a decimal number in the address specification and this will dictate the setting of the BISB.

We will hence forth write the CUAL specifications as

CUAL (A, B) + Carry-in.

Both CU and A are redundant information since this is written in the microinstruction sequencing field of the microinstruction and we will use the shorter form

AL(B) + Carry-in

where B is a signed integer, $-2048 \leq B \leq 2048$, when combined in an arithmetic function with A, but may obviously lie in the interval $0 \leq B \leq 4095$ when used for absolute jumps.

Example 1

If the CUALF is set to A + B and BISB = 01, then the specification

if c then AL(-18).

can be interpreted to mean

if c then HERE-18 else HERE + 1.

Example 2

If the CUALF is set to A + B and BISB = 01, then the specification

if c then AL(12) else AL(12) + 1

can be interpreted to mean

if c then HERE + 12 else HERE + 13

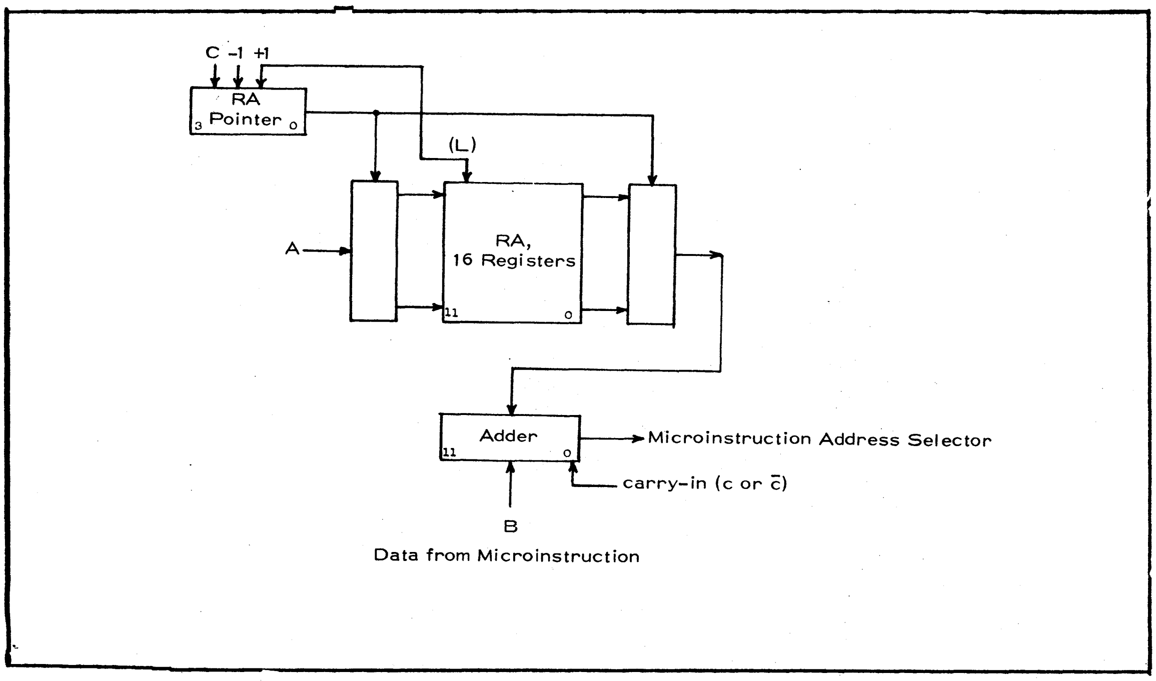
thus giving a conditional branch to one of two sequentially located microinstructions.

2.20.3. Return Jump Stacks A and B

There are two return jump stacks associated with the microinstruction addressing facility. They are called RA and RB. Each is a 12-bit wide, 16 element RG. RA is shown in Figure 2.29. RB, not shown, is identical.

Figure 2.29.

Return Jump Stack A, RA



The microoperations associated with RA are shown in Table 2.25. The instructions for RB are identical

Table 2.25.

Microoperations for control of RA

Notation on fig. 2.29.	Notation map	Microoperation
+ 1 ∧ (L)	RA ↓	Increment RAP and then load RA with the address of the current microinstruction.
- 1	RA ↑	Decrement RAP
c	RAPC	Clear the RAP

Whenever the top of the RA stack is used in the computation of the address of the next microoperation, the microoperation RA↑ is executed, i.e., the stack pointer is automatically maintained any time something is added to the stack or whenever the stack is used in an address computation. The use of RA is specified by writing

RA + B + Carry-in.

This is seen immediately from Figure 2.29. The B data and the Carry-in selection are exactly the same as those specified for the CUAL. The specification RA + 1 or RB + 1 will be interpreted to mean B = 0 and the carry-in = 1.

<u>Example 1</u>

Suppose we are in a routine at step n and wish to jump to a routine at step n + m. At step j of the second routine wish to return to n + 1. Assuming the CUALF:=A + B we could write

```
n:    ; RA ↓      ; AL(m)

m:
 .
 .
 .
j:    ;           ; RA + 1.
```

<u>Example 2</u>

It should be noted that the availability of 2 return jump stacks may facilitate the implementation of coroutines. For example, the microinstruction

```
n:    ; RA ↓      ; RB + 1.
```

stores the current address in one stack while simultaneously using the other stack as a source in the computation of the address of the next microinstruction.

<u>Example 3</u>

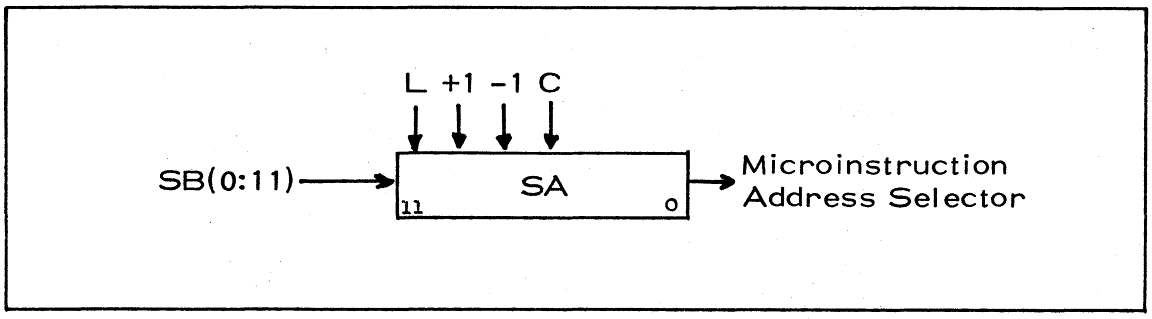
A conditional return entry point can be obtained by using the specification

if c then RA + B + 1 else RA + B.

2.20.4. The Save Address Register

The Save Address register, SA, is shown in Figure 2.30.

Figure 2.30.
The Save Address Register, SA



The microoperations associated with this register are shown in Table 2.26.

Table 2.26.
Microoperations for control of SA

SA:=SB
SA + 1
SA - 1
SAC

SA provides a data path between the bus structure of RIKKE-1 and the control unit which controls the transactions on this structure. It can be used, for example, during the loading of control store. (See Section 2.20.6.).

2.20.5. The Microinstruction Address Bus

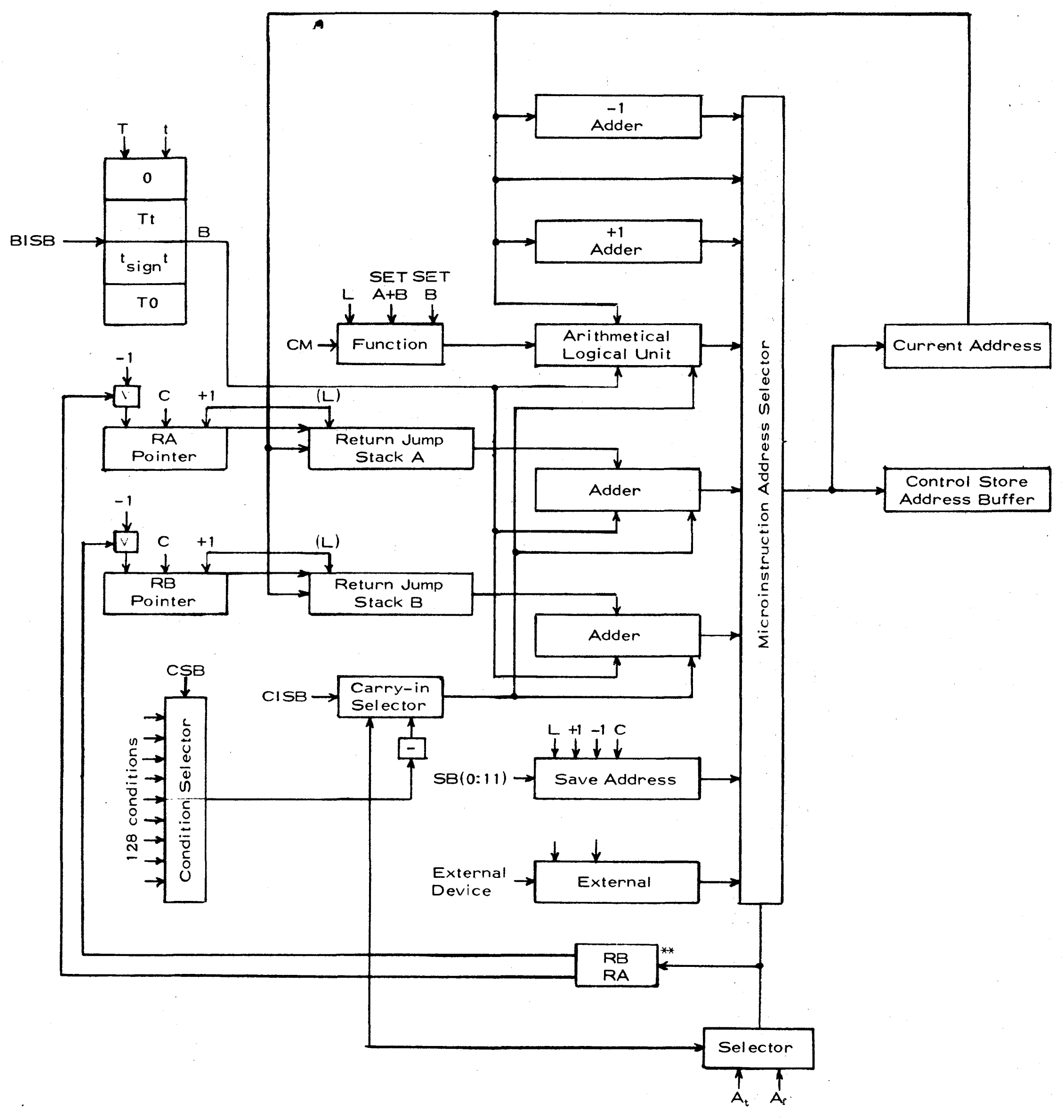
Having gained insight into the nature of the various address sources which can be used during microinstruction sequencing, we can now present a more detailed picture of the microinstruction address bus and it is shown as Figure 2.32. Because the number of control elements is small, they are also shown on this figure.

The microoperations associated with the control unit are brought together, for convenience, in Table. 2. 27. All but the last microoperations have been explained in previous sections. The CS Load operation is discussed next.

Table 2.27.
Microoperations associated with the Control Unit

SA:=SB
SA + 1
SA - 1
SAC
CUALF:=CM
S CUALF B
S CUALF +
RA ↑
RA ↓
RAPC
RB ↑
RB ↓
RBPC
CS Load
LCC

Figure 2.31.
Microinstruction Address Bus (Detailed)

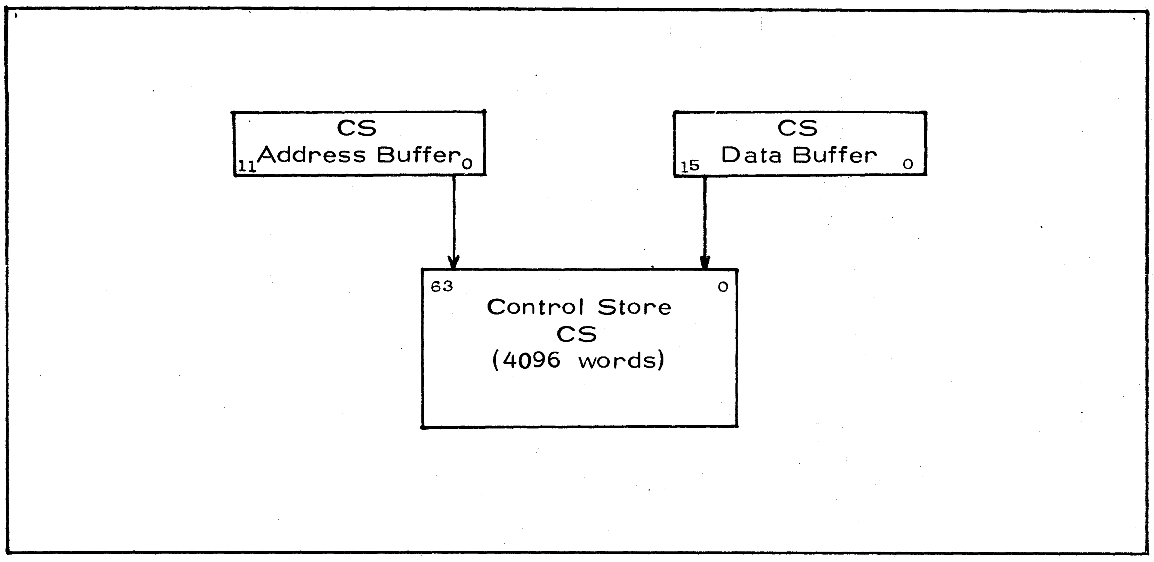


** the address selector bits are decoded to determine if RA or RB are selected.

2.20.6. Control Store Loading

Control Store has both an address and a data buffer, as shown below in figure 2.32.

Figure 2.32.
Control Store



The CS Address Buffer is loaded from the Microinstruction Address Selector as shown in Fig. 2.32. The CS Data Buffer is actually Device no. 0 associated with Output Port B.

Since the Output-Port is only 16 bits wide and the Control Store is 64 bits wide, the loading of 1 Control Store word takes at least 4 microoperations. Associated with the Control Store is a Loading counter, LC. The LC indicates whether the next word loaded should be directed to bit 0-15 16-31, 32-47 or 48-63 of the Control Store word pointed to by CS Address Buffer. The load counter is automatically increased when the CS Load microoperation i s executed. Furthermore one can clear the LC by the microoperation LCC.

Let A be the address of the current microinstruction.

The microoperation CS Load, if executed in the current microinstruction, can be interpreted as follows

CS Load ≈
Load the content of the CS Data Buffer into the bits indicated by LC of the Control Store Location pointed to by the CS address Buffer. In crement LC and then choose A + 1 as the address of the next microinstruction.

Example

Load the contents of WA1 - WA4 into the CS storage Location specified by the rightmost 12 bits of WA0

```
              ; WAPC, OBD:=0, CA:=3.
              ; SA:=SB, WAP+1
LOAD: OB:=WA  ; if OBSA then HERE+1 else HERE.
              ; OBA;
              ; CS Load; SA.
              ; WAP+1, CA-1, if CA then HERE+1
                else LOAD.                        ■
```

2. 21. Control Panel Switches KA and KB

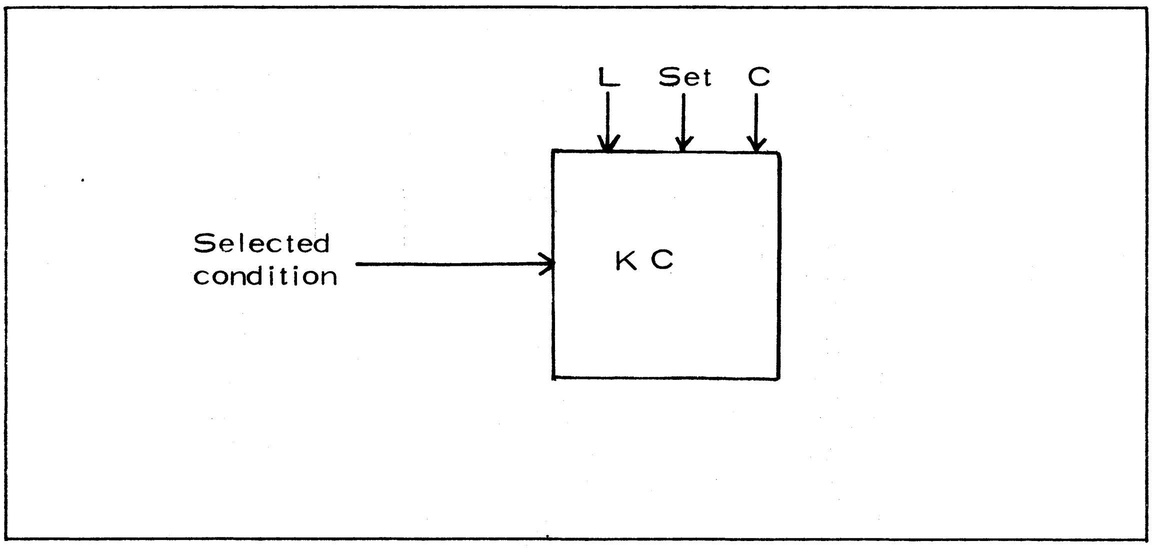
KA and KB are two switches on the control panel which can be set/reset by the operator and tested as any other condition in the microinstruction condition part.

2. 22. Internal Flags KC and KD

KC and KD are two flip-flop's which can be loaded, reset and tested in the microoperation. Fig. 2..33. shows KC, KD not shown is identical.

Figure 2. 33.

Internal Flag KC



KC and KD can be tested as any other condition in the microinstruction condition part.

The microoperations associated with KC and KD are

Table 2. 28.

Microoperations for KC and KD

KCC	clear KC
SET KC	set KC
KC:=SC	load KC with selected condition
KDC	clear KD
SET KD	set KD
KD:=SC	load KC with selected condition

2.23. The Conditions and Condition Selector

There is the possibility of testing 128 conditions in the system. At this writing there have been 100 specified, leaving a reasonable amount of expandability in the system. The conditions and their symbolic notation are given in Table 2.29.

The conditions in this table are grouped according to the functional unit with which they are associated. For convenience, the units are listed in alphabetical order.

Table 2. 29.

Condition List

Unit	Symbolic Notation	Condition
AL	AL	all bits AL(0:15)≡1
	ALOV	Al carry-out bit
	AL(0)	bit 0 of AL input to bus selector
	AL(15)	bit 15 of AL input to bus selector
	TWOOV	2's complement overflow
AS	AS(0)	bit 0 of the AS
	AS(V)	the variable bit of the AS
	AS(15)	bit 15 of the AS
BP	BP	BUS parity, BP=1 ⇒ odd parity
BUS	BUS	BUS(0:15) ≡ 0
CA	CA	is CA zero
	CA(3)	bit 3 of CA
	CA(4)	bit 4 of CA
	CA(5)	bit 5 of CA
	CA(6)	bit 6 of CA
	CASPOV	CASP ≡ 1111 (CASP overflow)
CB	CB	is CB zero
	CB(3)	bit 3 of CB
	CB(4)	bit 4 of CB
	CB(5)	bit 5 of CB]
	CB(6)	bit 6 of CB
	CBSPOV	CBSP ≡ 1111 (CBSP overflow)
CU	RAPOV	RAP≠1111 (RAP overflow)
	RAPUN	RAP ≡ 0000 (RAP underflow)
	RBPOV	RBP≠1111 (RBP overflow)
	RBPUN	RBP ≡ 0000 (RBP underflow)
	CUALOV	CUAL overflow

Fejl!

UNIT	Symbolic Notation	Condition
DS	DS(j), j=0, . . , 15	the indicated bit of the DS
	DS(j), j=V, V+1	the variable bits of the DS
I/O	IADA	data available on IA
	IADM	mark bit IA
	IBDA	data available on IB
	IBDM	mark bit IB
	OASA	space available on OA
	OBSA	space available on OB
	OCSA	space available on OC
	ODSA	space available on OD
KA	KA	KA button set
KB	KB	KB button set
KC	KC	KC flag set
KD	KD	KD flag set
LR	LR(0)	bit 0 of LR input to bus selector
	LR(15)	bit 15 of LR input to bus selector
SB	SB(0)	bit 0 of the shifted bus
	SB(1)	bit 1 of the shifted bus
	SB(14)	bit 14 of the shifted bus
	SB(15)	bit 15 of the shifted bus
System	TRUE	a binary one
	FALSE	a binary zero
VS	VS(0)	bit 0 of VS input to bus selector
	VS(V)	the variable bit of the VS
	VS(15)	bit 15 of the VS
WA	WA(0)	bit 0 of WA input to bus selector
	WA(15)	bit 15 of WA input to bus selector
	WAPOV	WAP $\equiv$ 11111111 (WAP overflow)
	WAPSPOV	WASP $\equiv$ 11111111 (WAPSP overflow

Unit	Symbolic Notation	Condition
WB	WB(0)	bit 0 of WB inputto bus selector
	WB(15)	bit 15 of WB input to bus selector
WB	WBPOV	WBP ≡ 11111111 (WBP overflow)
	WBPSPOV	WBPSP ≡ 11111111 (WBPSP overflow

All 128 conditions are input into a condition selector. There are 7 bits in each microinstruction, called the Condition Selection Bits, CSB, which select a particular condition. The selected condition is input into

a) The A_t-A_f address selector (Section 2. 20. 1.),

b) The carry-in selector (Section 2. 20. 2.)

2.24. Short and Long Cycle

It is obviously important to know when one can test a condition. The system can execute microinstructions in two different cycle times: a "short" cycle time and a "long" cycle time. The difference in these two cycles as it relates to the testing of conditions can easily be stated:

Long cycle
When the machine is operating in long cycle mode all conditions which arise as a result of bus transport and microoperation execution are testable in the same microinstruction in which they arise.

short cycle
When the machine is operating in short cycle mode all conditions which arise as a result of bus transport and microoperation execution are testable in the next microinstruction to be executed.

Thus if we are in long cycle and we write

```
        WA:=WB; WAP + 1; if BUS then RA+1.              ■
```

we are testing whether or not if the current bus transport (WA:=WB) is such that BUS $\equiv 0$. Whereas, in short cycle, this microinstruction would mean we are testing the previous bus transport's condition. In order to test WA:=WB we would have to write 2 microinstructions,

```
        WA:=WB ; WAP + 1.
               ; if BUS then RA+1.                      ■
```

Thus, a microinstruction can be throught of being executed in the following sequential way:

Short cycle:

a) Microinstruction fetch and saving of conditions
b) Bus-Transport
c) Execution of microoperations
d) Calculation of the address of next microinstruction based on saved conditions.

Long cycle:

a) Microinstruction fetch
b) Bus-transport
c) Execution of microoperations
d) Calculation of the address of next microinstruction based on the actual state of machine (new values of conditions).

The difference between short and long cycle is that step d) is delayed in long cycle to wait for conditions affected by b) and c).

The above mentioned steps may be considered as being executed sequentially, (this implies that one step is completely finished before the next is entered) similarily each of the steps may be broken up in a number of sequential steps (each of which is completed before the next is initiated), these will be described in section 3. 2.

NB! At the current version of RIKKE-1 it is not yet possible to switch between short and long cycle. The RIKKE-1 is meanwhile operating in short cycle.

2.25. The Real Time Clock

The RIKKE-1 will be supplied with a Real Time Clock, but this is not yet designed.

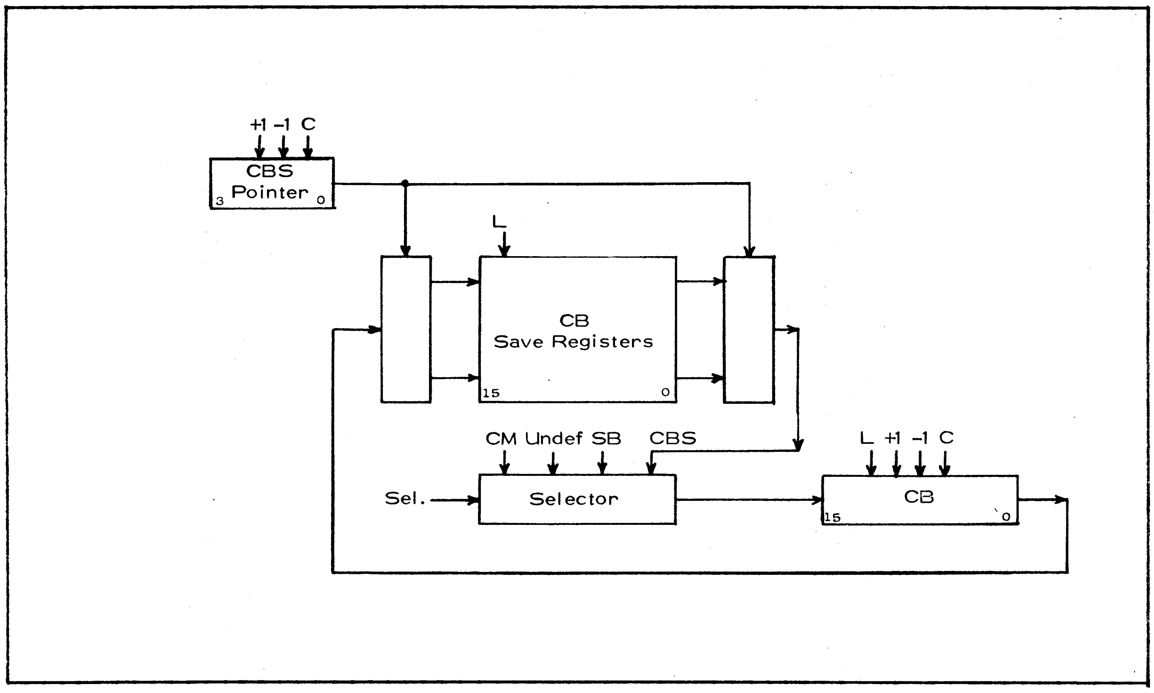
2. 26. Auxiliary Facilities

The auxiliary facilities associated with the RIKKE-1 system as shown in Figure 2. 1. i. e., the system counters and main storage will now be discussed.

2. 26. 1. Counter B

The system has 2 counters associated with it: Counter A, CA, has been introduced in Section 2. 2., Counter B, CB, introduced here is shown in Figure 2. 34.

Figure 2. 34.
Counter B, CB



A comparison of this figure with Figure 2. 3. which shows CA reveals that CB is identical with CA except that CA can be loaded from the OD register which is not the case with CB, i. e., we have

$$CA:=CM|SB|OD|CAS$$

and $$CB:=CM|SB|\ *\ |CBS\ .$$

*) Undef.

The microoperations associated with CB, CBS, and CBSP are given in Table 2.30. These are, of course, apart from the above difference, identical to those associated with CA and merely shown here for convenience.

Tabel 2.30.
Microoperations for control of CB, CBS, and CBSP

CB:=CM|SB| * |CBS
CB + 1
CB - 1
CBS
CBS:=CB
CBSP + 1
CBSP - 1
CBSPC

I should be quite obvious that CA and CB are not connected in any way whatsoever and may be used independent of one another. One may count up in CA while counting down in CM, for example,

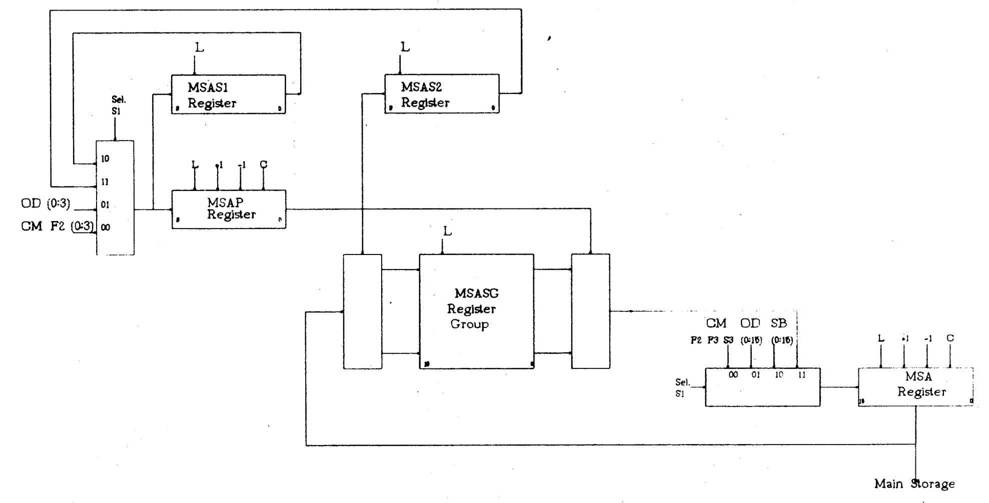
; CA + 1, CB - 1 . ■

2.26.2. Main Memory

The RIKKE 1 has a memory of up to 64K 16 bits words called MS. The addressing is provided through a main storage pointer, MSA.

*) Undefined.

Figure 2. 35.
Main Storage Address



The reading of Main Memory is going to take place from Input Port A, (device indifferent), and writing through Output Port A, (device indifferent), although the assembler will recognize MSW as OAA and MSR as IAA.

The microoperations associated with MSA, MSASG and MSAP are given in table 2. 31.

Table 2.31.

MSA:=CM|OD|SB|MSPSG	
MSA + 1	
MSA - 1	
MSAC	
MSASG:=MSA	
MSAP:=CM|OD|S1|S2	
MSAP + 1	
MSAP - 1	
MSAPC	
MSAS2:=MSAP	
MSAS1:=CM|OD|S1|S2	
MSR	The assembler recognizes these as synonyms for IAA and OAA
MSW	

It is possible to check the content of MSA against the actual physical size of main storage. The condition MSAOR is a 1 if the content of MSA is greater than the actual size of main storage, else 0. Furthermore it is possible to test if MSA is busy (i.e. main store is using MSA), this condition is named MSAB.

<u>Example</u>

Assume we want to store the contents of the WA-register pointed to by WAP in the main storage location pointed to by the AS. We can write this as

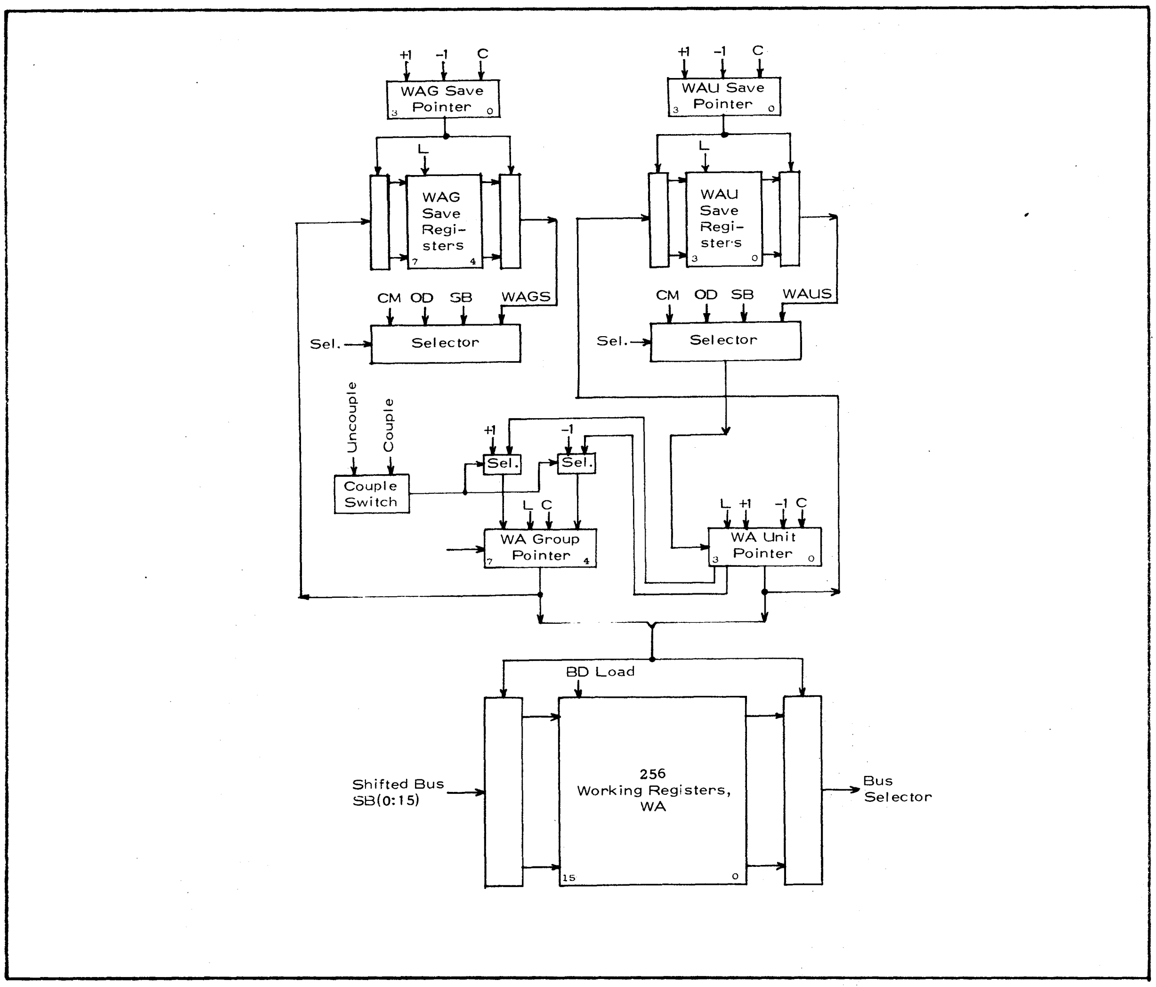
```
ALF:=B ; if MSAB then HERE else HERE+1
AL     ; MSA:=SB.
OA:=WA; MSW.
```

■

2.27. An Alternate View of the Working Registers

The description of WA which was given in Section 2.4. introduced WA as a 256 element RG. In Figure 2.5. the address pointer, WAP, was shown to be 8-bits wide so that the WA registers could be addressed as 256 contiguous registers. In fact, the address pointer actually consists of two 4-bit pointers which had been "coupled" together to give the 8-bit wide pointer described in Section. 2.4. Figure 2.36. shows WA with its two 4 -bit pointers called the Group and Unit pointer; WB, not shown, is identical.

Figure 2.36.
Working Registers A, WA (Detailed)



When the microoperation CPL A is executed, the Group and Unit pointers are connected together to give the 8-bit wide pointer, WAP. After the microoperation UNCPL A is executed, the Group and Unit pointers function as independent pointers. The low order 4-bits of the 8-bit address required to specify a particular register are given by the WA Unit pointer, WAU; high order 4-bits of the address are given by the WA Group pointer, WAG. Thus, WA can be considered to be 16 RG's, each RG having 16 registers.

The microoperations associated with the WAU and WAG pointers are given in Table 2.32. (The similar microoperations for WB are also shown.)

Table 2.32.
Microoperations for control of the WAU/WBU and WAG/WBG pointer

WAU:=CM $\mid$ OD $\mid$ SB_{03} $\mid$ WAUS	WBU:=CM $\mid$ OD_{8-11} $\mid$ SB_{8-11} $\mid$ WBUS
WAU + 1	WBU + 1
WAU - 1	WBU - 1
WAUC	WBUC
WAG:=CM $\mid$ OD_{4-7} $\mid$ SB_{4-7} $\mid$ WAGS	WBG:=CM $\mid$ OD_{12-15} $\mid$ SB_{12-15} $\mid$ WBGP
WAG + 1	WBG + 1
WAG - 1	WBG - 1
WAGC	WBGC

If we wanted to point to the 9th unit of group 3 and then transfer its contents to the DS, we could write, assuming the pointers are uncoupled,

```
          ; WAG:=3, WAU:=9.
DS:=WA.
```
■

The microoperations associated with WAP in Table 2.4. can now be given their appropriate meaning in terms of the microoperations in Table 2.32.

```
WAP + 1::=WAU + 1
WAP - 1::=WAU - 1
WAPC   ::=WAUC and WAGC
WAP     :=CM|OD|SB|WAPS::=WAU:=CM|OD|SB|WAUS
                              and  CM|OD|SB|WAGS.
```

Let us now turn our attention to the pointer save capability shown in Figure 2.36. When WA is considered as 16 groups of 16 registers, the WAU and WAG pointers may be saved independent of one another. The microoperations associated with this facility are given in Table 2.33.

Table 2.33.
Microoperations for control of WAUS and WAGS

WAUS:=WAU
WAUSP + 1
WAUSP - 1
WAUSPC
WAGS:=WAG
WAGSP + 1
WAGSP - 1
WAGSPC

As an example, suppose we are in group 3 and wish to work in group 8.

Before working in group 8 we want to save the unit which we are pointing to in group 3. This is done by executing

; WAUS:=WAU, WAG:=8 .

The microoperations associated with WAPS in table 2.4. can now be given their appropriate meaning in terms of the microoperations in Table 2.33. Thus we have,

```
WAPS:=WAP::=WAUS:=WAU and WAGS:=WAG
WAPSP + 1  ::=WAUSP + 1 and WAGSP + 1
WAPSP - 1  ::=WAUSP - 1 and WAGSP - 1
WAPSPC     ::=WAUSPC and WAGSPC.
```

There are a few additional conditions which can now be added to Table 2.29.

Table 2.34.

Additional WA and WB Conditions

Unit	Symbolic Notation	Condition
	WAUOV	WAU $\equiv$ 1111 (WAU overflow)
	WAGOV	WAG $\equiv$ 1111 (WAG overflow)
WA	WAUSPOV	WAUSP $\equiv$ 1111 (WAUSP overflow)
	WAGSPOV	WAGSP $\equiv$ 1111 (WAGSP overflow)
	WACS	WACS = 1 $\Rightarrow$ WAU and WAG are coupled
	WBUOV	WBU $\equiv$ 1111 (WBU overflow)
	WBGOV	WBG $\equiv$ 1111 (WBG overflow)
WB	WBUSPOV	WBUSP $\equiv$ 1111 (WBUSP overflow)
	WBGSPOV	WBGSP $\equiv$ 1111 (WBGSP overflow)
	WBCS	WBCS = 1 $\Rightarrow$ WBU and WBG are coupled

Thus we can deal with WA or WB as either 256 contiguous registers or 16 groups of 16 registers. We can switch back and forth between either interpretation in a relatively straightforward way.

2.28. An Alternate View of the Postshift Masks

The description of the Postshift Masks which was given in Section 2.7. was structured to make the Postshift Masks look as much like the Bus Masks as possible, to enchance the understanding of this unit. In fact, the output of the BS is masked during every bus transport by the mask which is specified to be

$$PA \vee PB \vee PG$$

where

PA = an element of a 16 bit wide, 16 element RG called the Postshift Mask A registers

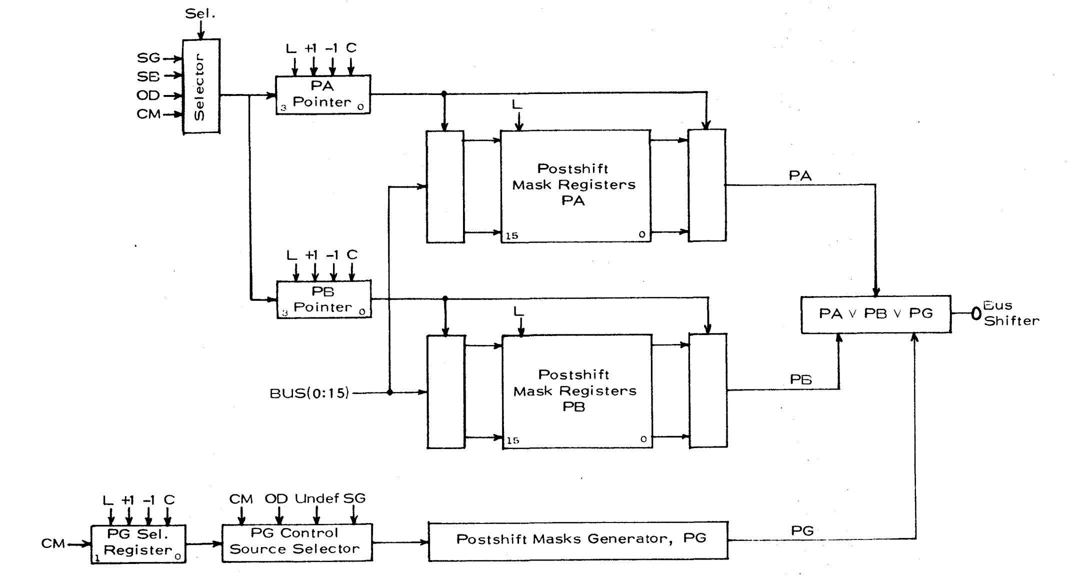
PB = an element of a 16-bit wide, 16 element RG called the Postshift Mask B registers

PG = the Postshift Mask Generator

V = logical "inclusive or".

In section 2.7. we had introduced the mask to be PAVPG; here we had merely assumed all elements of PB to contain all 0's. The actual situation is shown more clearly in Figure 2.37.

Figure 2.37.
Postshift Masks, PA, PB, and PG



The most important thing to note from this diagram is that the PA/PB structure is indeed the same as the MA/MB structure (see Figure 2.9.).

The microoperations associated with PB are then

Table 2.35.
Microoperations for control of PB

PB:=BUS
PBP:=CM|OD|SB|SG
PBP + 1
PBP - 1
PBPC

The name of the SG associated with the PA pointer and the PB pointer is the Postshift AB Pointer, PABP. The microoperations associated with this SG are given in Table 2.36.

Table 2.36.
Microoperations for control of PABP

PABP:=SB
PABPP:=CM|OD|S1|S2
PABPP + 1
PABPP - 1
PABPPC
PABPS1:=CM|OD|S1|S2
PABPS2:=PSBPP

We will assume that all elements of PB contain all 0's so that the effective mask is PA $\vee$ PG and all of out previous standardizations for the use of this facility are still valid.

3.0. Microinstruction Specification and Execution

We will in this section discuss the microinstruction format, the manner in which the instruction is executed, and then give a comprehensive table of all microoperations.

3.1. Microinstruction Format

Microinstructions are 64-bits wide. There are 4 major fields in a microinstruction. These fields specify

(a) bus transport
(b) microoperations and data
(c) microinstruction sequencing
(d) control of AS, VS, and DS

These fields are shown below with their sub-fields named and their actual bit location in the microinstruction.

(a) bus transport (7 bits)

BSE	BD	SOURCE
22	21 19	18 16
1	3	3

Bus Shifter Enable Bit

(b) microoperations and data (35 bits)

mops	mops/data	mops/data	mops/data
63 57	56 47	46 39	38 29
7	10	8	10

mops = microoperations

(c) microinstruction sequencing (16 bits)

BISB 15 14	CISB 13	Condition Selection 12 6	A_f 5 3	A_t 2 0
2	1	7	3	3

CISB → Carry-in selection bits

BISB → B-input selection bits

(d) AS, VS, and DS control (6 bits)

AS 28 27	VS 26 25	DS 24 23
2	2	2

Shift/Load Control for the Shifters

Let us discuss each of these in more detail.

(A) The Bus Transport Field

Table 3.1. shows the correspondence between the symbolic notation for SOURCE s and BD s and their binary representations.

Table 3.1.
Symbolic and Binary Notation for SOURCE s and BD s

SOURCE		BD	
Symbolic Notation	Binary Notation	Symbolic Notation	Binary Notation
LR	000	no destination	000
AL	001	MA	001
VS	010	MB	010
DS	011	LR	011
WA	100	WA	100
WB	101	WB	101
IA ≈	110	OA ≈	110
IB	111	OB	111

If the BS Enable bit = 0, no BS occurs; if the BS Enable bit = 1 a BS Shift occurs. The control source for BS control is given in the micro-operations and data field as in seen in (B) below. Thus the specification

BSE	BD	SOURCE
0	101	011

is the binary representation of our bus transport specification

WB:=DS .

We will show this symbolically as

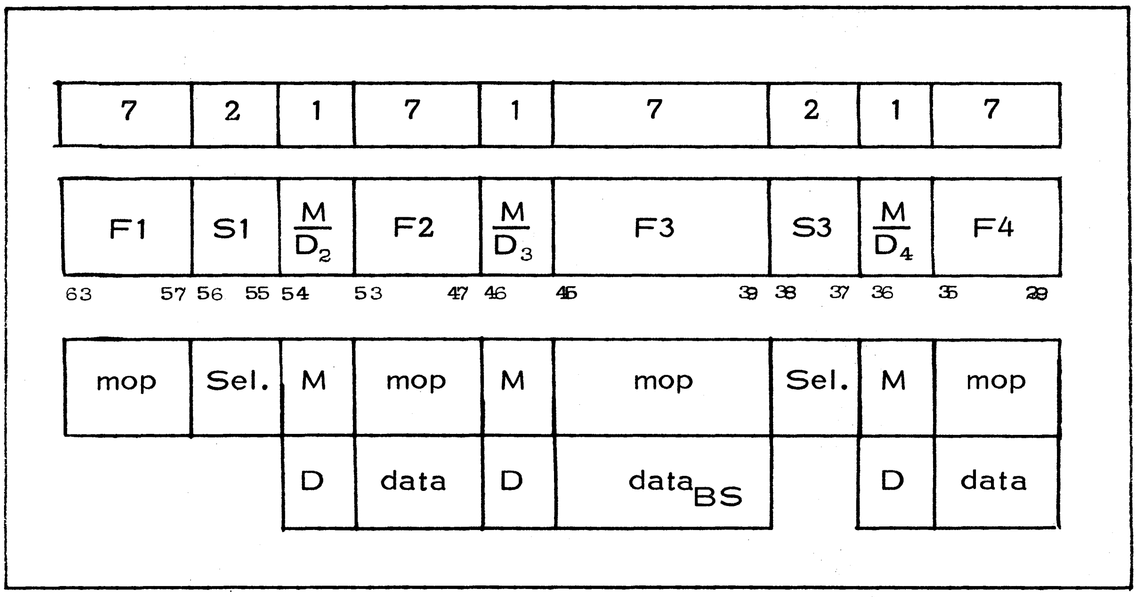
BSE	BD	SOURCE
0	WB	DS

as we have no need of binary representations in this report.

(B) The Microoperations and Data Field

The microoperations and data field can be considered to be made up of the following fields: F_1, S_1, $\frac{M}{D_2}$, F_2, $\frac{M}{D_3}$, F_3, S_3, $\frac{M}{D_4}$, F_4 as shown in Figure 3.1.

Figure 3.1.
Microoperation and Data Field



The following comments should assist in understanding this diagram.

B.1) Field F_1 always specifies a microoperation (1 of 128 mops).

if $\frac{M}{D_2} = 1$ then F_2 specifies a microoperation (1 of 128 mops).

if $\frac{M}{D_3} = 1$ then F_3 specifies a microoperation (1 of 128 mops).

if $\frac{M}{D_4} = 1$ then F_4 specifies a microoperation (1 of 128 mops).

Therefore up to 4 microoperations may be specified in this field; for example,

; BSP + 1, WBP + 1, MBP + 1, CA - 1;

B.2) We have seen that many microoperations concern the loading of a register from various sources, e.g.

MAP:=CM|OD|SB|SG .

Such a microoperation must be places either in field F_1 or F_3 . If it is placed in F_1 then the 2 selection bits S_1 specify which source will be used. If the source specified is the CM then $\frac{M}{D_2}$ is set to D and F_2 is used as data (similarly $\frac{M}{D_4}$ and F_4 are used with F_3). For example

MAP:=7

could be symbolically represented

F_1	S_1	$\frac{M}{D_2}$	F_2
MAP:=	CM	D	7

Thus one sees that there can be at most 2 microoperations of this type in a microinstruction.

B. 3) Figure 3. 1. also shows that if the BS control data is to be taken from the CM then F_3 is used as data. If the BS has been enabled, the control source is selected via BSS. Thus the specification

WA:=AL, BS $\rightarrow$ 3

could be symbolically represented

$\frac{M}{D_3}$	F_3	BSE	BD	SOURCE
D	3	1	WA	DS

B. 4) All of the possible microoperations are not available in each field F_1 , F_2 , F_3 , and F_4 . The microoperations which can be specified in each field are given in Section 3. 3., the Comprehensive Tables of Micro-operations for Individual Functional Units.

C) The Microinstruction Sequencing Field

Table 3. 2. shows the correspondence between the symbolic notation for A_t and A_f and their binary representations.

Table 3. 2.

Symbolic and Binary Notations for A_t and A_f

A_t and A_f	
Symbolic Notation	Binary Notation
	000
AL	001
RB	010
RA	011
SA	100
A-1	101
A+1	110
A	111

A similar table can be given for the symbolic and binary notations for the conditions but is not given here because of its length. Tables 2.23. and 2.24. present this information for the CISB (Carry-in selection bit) and BISB (B-input selection bits) respectively. We will give all of our examples symbolically.

Example 1

if BUS ≡ 0 then HERE. could be represented

BISB	CISB	Condition Selection	A_f	A_t
0		BUS	A+1	A

Example 2

If ALOV then RA + 12. could be represented

BISB	CISB	Condition Selection	A_f	A_t
$t_{sign}t$		ALOV	A+1	RA+B

However, this is incomplete and immediately raises the question where do T and t come from? That is easily answered. T is always the least significant 6 bits of F_3 and t is always the least significant 6 bits of F_4 . BISB tells us, of course, how we will combine T and t (i.e., 0, Tt, $t_{sign}t$, or T0 , see Section 2.20.2). Thus, the complete specification would be

$\frac{M}{D_4}$	F_4	BISB	CISB	Condition Selection	A_f	A_t
D	12	$t_{sign}t$		ALOV	A+1	RA+B

D) AS, VS, and DS Control Field

The dedicated bits for shifter control are interpreted as shown in Table 3.3.

Table 3.3.
Shift/Load Control Bits

Binary Notation	Shift/Load Control
00	Do Nothing
01	Shift Right
10	Shift Left
11	Load

Thus, the specification

AS →, VS ←, DS ←

could be represented symbolically as

AS	VS	DS
→	←	←

The binary representation

AS	VS	DS
01	10	10

does not interest us here. The specification

AS, LR:=AL; DS← .

would be given by

main	AS	VS	DS	BSE	BD	SOURCE	BISB	CISB	Condition Selection	A_f	A_t
	L		←		LR	AL	0		TRUE	A+1	A+1

3.2. Microinstruction Execution

As introduced in Section 2.4 , and then explained in more detail in Section 2.24. the machine has both a long cycle and a short cycle. The result of that discussion, which is repeated here for convenience is that microinstructions can be thought of being executed in the following sequential way:

Short cycle:

a) Microinstruction fetch and saving of conditions
b) Bus-Transport
c) Execution of microoperations
d) Calculation of the address of next microinstruction based on saved conditions.

Long cycle:

a) Microinstruction fetch
b) Bus-Transport
c) Execution of microoperations
d) Calculation of the address of next microinstruction based on the actual state of machine (new value of conditions).

The difference between short and long cycle is that step d) is delayed in long cycle to wait for conditions affected by b) and c).

The above mentioned steps may be considered as being executed sequentially, (this implies that one step is completely finished before the next is entered), similarily each of the steps may be broken up in a number of sequential steps (each of which is completed before the next is initiated).

B) Bustransport

0) The SOURCE is selected.
1) The information of the SOURCE is masked by the BUS-masks and gated onto the BUS.
2) The BUS is shifted by the Bus-Shifter if this is enabled.
3) The output of BS is masked byt he Postshiftmask to yield the shifted Bus SB.
4) Loading of SB into the selected destination.

C) Execution of microoperations

0) Execute microoperations with $C_p = 1$.
1) Execute load/shift operations in AS, VS and DS.
2) Execute microoperations with $C_p = 2$.

D) Address calculation

0) Choose the selected condition and name it c (in short cycle a saved value, in long cycle the new state).
1) Select the carry-in and B-input into the CUAL, and the RA and RB adders.
2) Compute the result in the adders.
3) Select the next address using A_t if c = 1 or A_f if c = 0.
4) If RA and RB has been selected then pop the stack that was used
5) If an ~~[illegible]~~ interrupt has occured then load IRA and clear address-buffers

Notice that conflicts can occur between actions that take place within the same of the above mentioned sequential steps, and of course especially in those cases where more than one action refers to the same unit (ex.: count and clear of the same register) in which case the result is undefined.

Another source of conflicts is the case where an action in one step influences the information which is being gated onto some datapath, and the information is used (e. g. loaded) in a later step. On the other hand if no actions refer to the source nor to the datapath itself, the information on the path can be assumed to be stable in all the following steps.

*) The microoperations are divided into two classes those with Cp=1 and those with Cp=2. This defines exactly when the microoperation is initiated.

Although many conflicts are resolved by the sequential nature of the timing some will remain and will result in undefined situations some of which will be listed below:

a) WA:=WA and WB:=WB gives an undefined result.

b) LR:=LR and LRIP=LROP gives an undefined result.

c) Loading of a mask (MA, MB, PA, PB) in a bustransport where the same mask is being applied.

3. 2. 1. Example of how to use Clock Pulse 1 and Clock Pulse 2

Recall that the RG is a basic building element used in the system. A very common operation is to load an RG and then change its pointer (e. g. this was done quite frequently in our examples). Often, one also wished to save the address of the current element pointed to before the pointer is changed. It was decided that this capability should be allowed in one microinstruction and, furthermore, <u>every</u> RG in the system should be treated in the same uniform way.

Example

The microinstruction

AS:=WA; WAPS:=WAP, WAP + 1 . ■

means: take the element of WA pointed to by WAP and store it in the AS; then store the WAP in the WAPS registers and then increment WAP by 1. It means this because the BD load and the microoperation bot occur before the microoperation WAP + 1 in the above mentioned sequential scheme. Thus, every RG in the system can be looked at in the following way:

a) it can be loaded or used as a source.

b) its current pointer can be saved, if it has a save capability.

c) its pointer can be changed after a) and b).

all with one microinstruction.

3. 3. Comprehensive Tables of Microoperations for Individual Functional Units

The following tables (presented in alphabetical order based on the abbreviations associated with the functional unit) show which microoperations can appear in which fields and at which clock pulse these microoperations are initiated.

Some particular points perhaps should be recalled and emphasized here:

a) use of these tables will show what space and time conflicts arise in the construction of a microinstruction. The reader is encouraged to review some of the examples of the earlier sections by constructing symbolic microinstructions similar to those presented in Section 3. 1.

b) t comes from field F_4, so if it is being used, for example in absolute addressing, a microinstruction should not be specified in F_3.

c) T comes from field F_3, so if T is being used, for example in absolute addressing, a microinstruction should not be specified in F_3.

d) Data for the BS, if the CM is the control source, comes from F_3.

e) Data for the PG, if the CM is the ocntrol source, comes from F_3.

MICPOOPERATIONS FOR /L

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
ALF:=	76	++					2	LOAD ALF FROM CM/OD/SB/SG
ALPC				63			2	CLEAR ALSG POINTER
ALP+1				61			2	INCREMENT ALP
ALP-1				62			2	DECREMENT ALP
ALP:=				60	++		2	LOAD ALSG PCINTER FROM CM/OD/S1/S2
ALSG:=SB			97				1	LOAD ALSG FROM SB(5:0)
ALS1:=			72	64	++		2	LOAD ALSG SAVE1 FROM CM/OD/S1/S2
ALS2:=ALP	121						1	LOAD ALS2 FROM ALP
SETALFB			74				2	SET ALF TO B
SETALF+1			91				2	SET ALF TO A+1
SETALF+			73				1	SET ALF TO A+B
SETALF-			90				2	SET ALF TO A-1

MICROOPERATIONS FOR AS

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
ASLL						79	2	SET THE AS TO LOGICAL LEFT SHIFT
ASLR						80	2	SET THE AS TO LOGICAL RIGHT SHIFT
AS(V)SC				67			2	CLEAR AS(V)S
AS(V)S+1				65			2	INCREMENT AS(V)S
AS(V)S-1				66			2	DECREMENT AS(V)S
AS(V)S:=	79	++				78	2	LOAD AS(V)S FROM CM/OD/SB/SG
AS(0)S:=	77	++				76	2	LOAD AS(0)S FROM CM/OD/SB/SG
AS(15)S:=	78	++				77	2	LOAD AS(15)S FROM CM/OD/SB/SG

MICROOPERATIONS FOR BS

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
BSPC	97						2	CLEAR BSSG POINTER
BSP+1	95						2	INCREMENT BSP
BSP-1	96						2	DECREMENT BSP
BSP:=	94	++					2	LOAD BSSG PCINTER FROM CM/OD/S1/S2
BSSG:=SB						105	1	LOAD BSSG FROM SB
BSS1:=	98	++					2	LOAD BSS1 FROM CM/OD/S1/S2
BSS2:=BSP				106			1	LOAD BSS2 FROM BSP
BSS:=	109		56	120	++		2	LOAD BSS FROM S3

MICROOPERATIONS FOR CS

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
CSLL						86	2	SET AS,DS AND VS TO LOGICAL LEFT SHIFT
CSLR						87	2	SET AS,DS AND VS TO LOGICAL RIGHT SHIFT
CSPC				74			2	CLEAR CSP
CSP+1				72			2	INCREMENT CSP
CSP-1				73			2	DECREMENT CSP
CSP:=				71	++		2	LOAD CSP FROM CM/OD/S1/S2
CSSG:=SB			98				1	LOAD CSSG FROM SB
CSS1:=			78	75	++		2	LOAD CSS1 FROM CM/OD/S1/S2
CSS2:=CSP	122			104			1	LOAD CSS2 FROM CSP
CS(V)SC				76			2	CLEAR AS(V)S,VS(V)S AND DS(V+1:V)S
CS(V)S:=	91	++					2	LOAD AS(V)S,VS(V)S,DS(V+1:V)S FROM CM/OD/SB/SG
CS(0)S:=	89	++					2	LOAD AS(0)S,VS(0S),DS(1:0)S FROM CM/OD/SB/SG
CS(15)S:=	90	++					2	LOAD AS(15)S,VS(15)S,DS(15:14)S FROM CM/OD/SB/SG

MICROOPERATIONS FOR BM

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
BMPPC				54			2	CLEAR BMPP
BMPP+1				52			2	INCREMENT BMPP
BMPP-1				53			2	DECREMENT BMPP
BMPP:=				51	++		2	LOAD BMPP FROM CM/OD/S1/S2
BMPS1:=			67	55	++		2	LOAD BMPS1 FROM CM/OD/S1/S2
BMPS2:=BMPP	120						1	LOAD BMPS2 FROM BMPP
BMP:=SB			96				1	LOAD BMP WITH SB(3:0)
MAPC	67		66			67	2	CLEAR MAP
MAP+1	65		64			65	2	INCREMENT MAP
MAP-1	66		65			66	2	DECREMENT MAP
MAP:=	64	++				64	2	LOAD MAP FROM CM/OD/SB
MBPC	71			50		71	2	CLEAR MBP
MBP+1	69			48		69	2	INCREMENT MBP
MBP-1	70			49		70	2	DECREMENT MBP
MBP:=	68	++				68	2	LOAD MBP FROM CM/OD/SB

MICROOPERATIONS FOR CA

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
CAC	35			34		43	2	CLEAR CA
CASPC			34				2	CLEAR CASP
CASP+1			32				2	INCREMENT CASP
CASP-1			33				2	DECREMENT CASP
CAS:=CA			51				1	LOAD CA SAVE RG FROM CA
CA+1	33			32		41	2	INCREMENT CA
CA-1	34			33		42	2	DECREMENT CA
CA:=	32	++				40	2	LOAD CA FROM CM/OD/SB/CAS

MICROOPERATIONS FOR CB

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
CBC	39		37			47	2	CLEAR CB
CBSPC				37			2	CLEAR CBSP
CBSP+1				35			2	INCREMENT CBSP
CBSP-1				36			2	DECREMENT CBSP
CBS:=CB				46			1	LOAD CB SAVE RG FROM CB
CB+1	37		35			45	2	INCREMENT CB
CB-1	38		36			46	2	DECREMENT CB
CB:=	36	++				44	2	LOAD CB FROM CM/SB/CBS

MICROOPERATIONS FOR DS

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
DSLL						84	2	SET DS TO LOGICAL LEFT SHIFT
DSLR						85	2	SET DS TO LOGICAL RIGHT SHIFT
DS(V)SC				70			2	CLEAR DS(V)S
DS(V)S+1				68			2	INCREMENT DS(V)S
DS(V)S-1				69			2	DECREMENT DS(V)S
DS(V)S:=	88	++					2	LOAD DS(V)S FROM CM/OD/SB/SG
DS(15:14)S:=	87	++					2	LOAD DS(15:14)S FROM CM/OD/SB/SG
DS(1:0)S:=	86	++					2	LOAD DS(1:0)S FROM CM/OD/SB/SG

MICROOPERATIONS FOR INPUT

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
IAA	107		86			98	2	ACTIVATE PORT A
IADC			108				1	CLEAR IAD
IAD+1			107				1	INCREMENT IAD
IAD-1			115				1	DECREMENT IAD
IAD:=	119	++					1	LOAD IAD FROM CM/OD/SB
IBA	108		87			99	2	ACTIVATE PORT B
IBDC			110				1	CLEAR IBD
IBD+1			109				1	INCREMENT IBD
IBD-1			116				1	DECREMENT IAD
IBD:=	118	++					1	LOAD IBD FROM CM/OD/SB

MICROOPERATIONS FOR KC AND KD

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
KCC			13				1	CLEAR KC
KC:=			11				1	LOAD KC WITH SELECTED CONDITION
KDC			6				1	CLEAR KD
KD:=						7	1	LOAD KD WITH SELECTED CONDITION
SETKC			2				1	SET KC IE KC:=1
SETKD						5	1	SET KD IE KD:=1

MICROOPERATIONS FOR LA

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
LAPC	30					34	2	CLEAR LAP
LAP+1	28		26			32	2	INCREMENT LAP
LAP-1	29		27			33	2	DECREMENT LAP
LAP:=	27	++					2	LOAD LAP FROM CM/OD/S1/S2
LAS1:=	31	++				35	1	LOAD LAS1 FROM CM/OD/S1/S2
LAS2:=LAP				45			1	LOAD LAS2 FROM LAP
LA:=¬SB						60	1	LOAD LA WITH INVERTED SB

MICROOPERATIONS FOR LB

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
LBPC				30		38	2	CLEAR LBP
LBP+1			28	28		36	2	INCREMENT LBP
LBP-1			29	29		37	2	DECREMENT LBP
LBP:=				27	++		2	LOAD LBP FROM CM/OD/S1/S2
LBS1:=			30	31	++		2	LOAD LBS1 FROM CM/OD/S1/S2
LBS2:=LBP	57						1	LOAD LBS2 FROM LBP
LB:=¬SB			50				2	LOAD LB WITH INVERTED SB

MICROOPERATIONS FOR LR

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
LRIPC	75						2	CLEAR LRIP
LRIP+1	73						2	INCREMENT LRIP
LRIP-1	74						2	DECREMENT LRIP
LRIP:=DS	72						2	LOAD LRIP WITH DS(V+1:V)
LROPC						75	2	CLEAR LROP
LROP+1						73	2	INCREMENT LROP
LROP-1						74	2	DECREMENT LRIP
LROP:=DS						72	2	LOAD LROP WITH DS(V+1:V)
LRPC			71	59			2	CLEAR BOTH LRIP AND LROP
LRP+1			69	57			2	INCREMENT BOTH LRIP AND LROP
LRP-1			70	58			2	DECREMENT BOTH LRIP AND LROP
LRP:=DS			68	56			2	LOAD LRIP AND LROP WITH DS(V+1:V)

MICROOPERATIONS FOR MS

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
MSAC	43		42			50	2	CLEAR MSA
MSAPC	47						2	CLEAR MSAP
MSAP+1	45						2	INCREMENT MSAP
MSAP-1	46						2	DECREMENT MSAP
MSAP:=	44	++					2	LOAD MSAP FROM CM/OD/S1/S2
MSASG:=MSA	58					61	1	LOAD MSASG FROM MSA
MSAS1:=	48	++				51	2	LOAD MSAS1 FROM CM/OD/S1/S2
MSA+1	41		40			48	2	INCREMENT MSA
MSA-1	42		41			49	2	DECREMENT MSA
MSA:=	40	++					2	LOAD MSA FROM CM/OD/SB/SG

MICROOPERATIONS FOR OUTPUT

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
OAA	52	++		102		100	2	ACTIVATE PORT A
OADC				109			1	CLEAR OAD
OAD+1				115			1	INCREMENT OAD
OAD-1				116			1	DECREMENT OAD
OAD:=				108	++		1	LOAD OAD FROM CM/OD/SB
OAR			92				2	DEACTIVATE PORT A
OBA	53	++	88			101	2	ACTIVATE PORT B
OBDC			111				1	CLEAR OBD
OBD+1			117				1	INCREMENT OBD
OBD-1			118				1	DECREMENT OBD
OBD:=				110	++		1	LOAD OBD FROM CM/OD/SB
OBR			93				2	DEACTIVATE PORT B
OCA	54	++		103		102	2	ACTIVATE PORT C
OCDC				112			1	CLEAR OCD
OCD+1				117			1	INCREMENT OCD
OCD-1				118			1	DECREMENT OCD
OCD:=				111	++		1	LOAD OCD FROM CM/OD/SB
OCR			94				2	DECATIVATE PORT C
OC:=BUS			112				1	LOAD OC FROM BUS(15:0)
MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
ODA	55	++	89			103	2	ACTIVATE PORT D
ODDC			113				1	CLEAR OCD
ODD+1			119				1	INCREMENT ODD
ODD-1			120				1	DECREMENT ODD
ODD:=				113	++		1	LOAD ODD FROM CM/ODSB
ODR			95				2	DEACTIVATE PORT D
OD:=BUS			114				1	LOAD OD FROM BUS(15:0)

MICROOPERATIONS FOR PM

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
PABC				91			2	CLEAR PA AND PB POINTER
PABPPC				95			2	CLEAR PABPP
PABPP+1				93			2	INCREMENT PABPP
PABPP-1				94			2	DECREMENT PABPP
PABPP:=				92	++		2	LOAD PABPP FROM CM/OD/S1/S2
PABPS1:=			84	96	++		2	LOAD PABPS1 FROM CM/OD/S1/S2
PABPS2:=PABPP	126						1	LOAD PABPS2 FROM PABPP
PABP:=SB			106				1	LOAD PABP FROM SB
PAB+1				89			2	INCREMENT PAP AND PBP
PAB-1				90			2	DECREMENT PAP AND PBP
PAPC	102					94	2	CLEAR PA POINTER
PAP+1	100					92	2	INCREMENT PAP
PAP-1	101					93	2	DECREMENT PAP
PAP:=	99	++					2	LOAD PA POINTER FROM CM/OD/SB
PA:=¬BUS				107			1	LOAD PA WITH THE INVERTED BUS(15:0)
PBPC	106					97	2	CLEAR PB POINTER
PBP+1	104					95	2	INCREMENT PBP
PBP-1	105					96	2	DECREMENT PBP
PBP:=	103	++					2	LOAD PB POINTER FROM CM/OD/SB
MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
PB:=¬BUS				114			1	LOAD PB WITH THE INVERTED BUS(15:0)
PGPC				87			2	CLEAR PGSG POINTER
PGP+1				85			2	INCREMENT PGSG POINTER
PGP-1				86			2	DECREMENT PGSG POINTER
PGP:=				84	++		2	LOAD PGSG POINTER FROM CM/OD/S1/S2
PGSC			82			91	2	CLEAR PGS
PGSG:=SB			105				1	LOAD PGSG FROM SB
PGS1:=			83	88	++		2	LOAD PGS1 FROM CM/OD/S1/S2
PGS2:=PGP	125						1	LOAD PGS2 FROM PGP
PGS+1			80			89	2	INCREMENT PGS
PGS-1			81			90	2	DECREMENT PGS
PGS:=				83	++		2	LOAD PGS WITH S3

MICROOPERATIONS FOR SA

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
SAC						3	1	CLEAR SAVE ADDRESS
SA+1						1	1	INCREMENT SA
SA-1						2	1	DECREMENT SA
SA:=SB			1				2	LOAD SA FROM SB(11:0)

MICROOPERATIONS FOR CON.STORE

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
CS LOAD			8				1	LOAD CONTROL STORE AND CHOOSE HERE+1 AS NEXT
LCC				5			1	CLEAR LC

MICROOPERATIONS FOR WC

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
WCGS				44			1	LOAD WAGS AND WBGS
WCUS			49				1	LOAD WAUS AND WBUS
WCU+1						28	2	INCREMENT WAU AND WBU
WCU-1						29	2	DECREMENT WAU AND WBU

MICROOPERATIONS FOR VS

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
VSLL			76				2	SET VS TO LOGICAL LEFT SHIFT
VSLR			77				2	SET VS TO LOGICAL RIGHT SHIFT
VS(V)SC	85						2	CLEAR VS(V)S
VS(V)S+1	83						2	INCREMENT VS(V)S
VS(V)S-1	84						2	DECREMENT VS(V)S
VS(V)S:=	82	++				83	2	LOAD VS(V)S FROM CM/OD/SB/SG
VS(0)S:=	80	++				81	2	LOAD VS(0)S FROM CM/OD/SB/SG
VS(15)S:=	81	++				82	2	LOAD VS(15)S FROM CM/OD/SB/SG

MICROOPERATIONS FOR WAU

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
WAUC	19					18	2	CLEAR WAU POINTER
WAUSPC						21	2	CLEAR WAUSG POINTER
WAUSP+1						19	2	INCREMENT WAUSP
WAUSP-1						20	2	DECREMENT WAUSP
WAUS:=WAU						58	1	LOAD WAUS FROM WAU
WAU+1	17					16	2	INCREMENT WAU
WAU-1	18					17	2	DECREMENT WAU
WAU:=	16	++					2	LOAD WAU FROM CM/OD/SB/US

MICROOPERATIONS FOR WAG

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
WAGC				19			2	CLEAR WAG POINTER
WAGSPC	24						2	CLEAR WAGSG POINTER
WAGSP+1	22						2	INCREMENT WAGSP
WAGSP-1	23						2	DECREMENT WAGSP
WAGS:=WAG	56						1	LOAD WAGS FROM WAG
WAG+1				17			2	INCREMENT WAG
WAG-1				18			2	DECREMENT WAG
WAG:=				16	++		2	LOAD WAG POINTER FROM CM/OD/SB/SG

MICROOPERATIONS FOR WA COUPLED

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
CPL A						56	1	COUPLE WAU AND WAG
UNCPL A						57	1	UNCOUPLE WAU AND WAG
WAPC	21						2	CLEAR WAU POINTER AND WAG POINTER
WAPSPC						24	2	CLEAR WAP AND WAG
WAPSP+1						22	2	INCREMENT WAGSP AND WAUSP
WAPSP-1						23	2	DECREMENT WAGSP AND WAUSP
WAPS:=WAP						59	1	LOAD WAGS AND WAUS FROM WAG AND WAU RESPECTIVELY
WAP:=	20	++			++		2	LOAD WAU FROM CM/OD/SB/US,WAG FROM CM/OD/SB/GS

MICROOPERATIONS FOR WBU

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
WBUC			18			27	2	CLEAR WBU POINTER
WBUSPC			25				2	CLEAR WBUSG POINTER
WBUSP+1			23				2	INCREMENT WBUSP
WBUSP-1			24				2	DECREMENT WBUSP
WBUS:=WBU			48				1	LOAD WBUS FROM WBU
WBU+1			16			25	2	INCREMENT WAU
WBU-1			17			26	2	DECREMENT WAU
WBU:=				20	++		2	LOAD WBU POINTER FROM CM/OD/SB/US

MICROOPERATIONS FOR WBG

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
WBGC			21				2	CLEAR WBG POINTER
WBGSPC				23			2	CLEAR WBGSG POINTER
WBGSP+1				21			2	INCREMENT WBGSP
WBGSP-1				22			2	DECREMENT WBGSP
WBGS:=WBG				42			1	LOAD WBGS FROM WBG
WBG+1			19				2	INCREMENT WBG
WBG-1			20				2	DECREMENT WBG
WBG:=	25	++					2	LOAD WAG POINTER FROM CM/OD/SB/GS

MICROOPERATIONS FOR WB COUPLED

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
CPL B				40			1	COUPLE WBU AND WBG
UNCPL B				41			1	UNCOUPLE WBU AND WBG
WBPC			22				2	CLEAR WBP POINTER AND WBG POINTER
WBPSPC				26			2	CLEAR WBP AND WBG
WBPSP+1				24			2	INCREMENT WBGSP AND WBUSP
WBPSP-1				25			2	DECREMENT WBGSP AND WBUSP
WBPS:=WBP				43			1	LOAD WBUS AND WBGS FROM WBG AND WBU RESPECTIVELY
WBP:=	26	++			++		2	LOAD WBU FROM CM/OD/SB/US,WBG FROM CM/OD/SB/GS

MICROOPERATIONS FOR CUAL

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
CUALF:=				1			1	LOAD CUALF WITH DATA FROM F4
SCUALFB	1						1	SET CUALF TO B
SCUALF+				2			1	SET CUALF TO A+B

MICROOPERATIONS FOR RA

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
RAPC			4				1	CLEAR RA POINTER
RA↑			2				1	DECREMENT RA POINTER
RA↓	2		3	3			1	INCREMENT RA POINTER AND THEN LOAD RA

MICROOPERATIONS FOR RB

MICROOPERATION	F1	S1	F2	F3	S3	F4	CP	
RBPC	5						1	CLEAR RB POINTER
RB↑	3						1	DECREMENT RB POINTER
RB↓	4		5	4			1	INCREMENT RB POINTER AND THEN LOAD RB

Table of First Occurrance of Abbreviations and Symbols
(not including conditions or microoperations)

Abbreviation	Interpretation	Page
A_t , A_f	Address Specifications	15
AL	Arithmetical Logical Unit	35
ALF	AL Function and Carry-in Register	37
ALP	ALRG Pointer	37
ALSG	AL Standard Group	37
ALS1	ALSG Save1 Pointer	37
ALS2	ALSG Save2 Pointer	37
AS	Accumulator Shifter	40
BD	Bus Destination	11
BISB	B-Input Selection Bits	76
BM	Bus Masks	24
BMP	Bus Mask Pointer Standard Group	27
BMPP	BMP Pointer	28
BMPS1	BMP Save1 Register	28
BMPS2	BMP Save2 Register	28
BS	Bus Shifter	10
BSP	BS Standard Group Pointer	21
BSSG	Bus Shifter Standard Group	20
BSS	Bus Shifter Selection Register	20
BSS1	BS Save1 Register	21
BSS2	BS Save2 Register	22
BUS	the BUS	10
CA	Counter A	8
CAS	Counter A Save Registers	9
CASP	Counter A Save Register Pointer	9
CB	Counter B	92
CBS	Counter B Save Regsiters	92
CBSP	Counter B Save Register Pointer	93

Abbreviation	Interpretation	Page
CISB	Carry-in Selection Bit	74
CS	Common Shifter	54
CSB	Condition Selection Bits	89
CSP	CSSG Pointer	55
CSSG	Common Shifter Standard Group	54
CSS1	CSSG Save1 Register	55
CSS2	CSSG Save2 Register	55
CU	Control Unit	70
CUAL	Control Unit Arithmetical Logical Unit	72
CUALF	CUAL Function Register	73
DESTINA-TION	Bus Destination, BD	11
DS	Double Shifter	48
IA	Input Port A	61
IAD	IA Device Register	61
IB	Input Port B	61
IBD	IB Device Register	62
KA	Control Panel Switch KA	84
KB	Control Panel Switch KB	84
KC	Internal Flag KC	84
KD	Internal Flag KD	84
LA	Loading Masks A	56
LAP	LA Pointer	56
LAS1	LA Save1 Register	58
LAS2	LA Save 2 Register	58
LB	Loading Masks B	56
LBP	LB Pointer	58
LBS1	LB Save1 Register	58
LBS2	LB Save2 Register	58
LR	Local Registers	38

Abbreviation	Interpretation	Page
LRIP	Local Registers Input Pointer	38
LROP	Local Registers Output Pointer	38
LRP	LRIP and LROP	40
MA	Mask A Registers	25
MAP	MA Pointer	26
MB	Mask B Registers	25
MBP	MB Pointer	27
MS	Main Store	93
MSA	Main Store Address	93
MSAP	MSASG Pointer	94
MSASG	Main Store Address Standard Group	94
MSA S1	MSASG Save1 Register	95
MSA S2	MSASG Save2 Register	95
OA	Output Port A	64
OAD	OA Device Register	64
OB	Output Port B	64
OBD	OB Device Register	65
OC	Output Port C	64
OCD	OC Device Register	65
OD	Output Port D	64
ODD	OD Device Register	65
PA	Postshift Mask A Registers	30
PABP	Postshift AB Pointer	33
PAP	PA Pointer	33
PB	Postshift Mask B Registers	99
PBP	PB Pointer	100
PG	Postshift Mask Generator	30
PGP	PGSG Pointer	33
PGSG	Postshift Mask Generator Standard Group	33
PGS	Postshift Mask Generation Selection Reg.	32
PGS1	PGSG Save1 Register	33
PGS2	PGSG Save2 Register	33
PM	Postshift Masks	28

Abbreviation	Interpretation	Page
RA	Return Jump Stack A	77
RAP	Return Jump Stack A Pointer	78
RB	Return Jump Stack B	77
RBP	Return Jump Stack B Pointer	81
RG	Register Group	6
RGP	Register Group Pointer	6
SA	Save Address Register	70
SB	Shifted Bus	10
SG	Standard Group	22
"Shifters"	AS, VS, and DS	50
SOURCE	the input to the BUS	11
V	The Variable Bit	41
VS	Variable Width Shifter	46
WA	Working Registers A	10
WAG	Working Registers A Group Pointer	97
WAGS	WAG Save Registers	97
WAP	WA Pointer	12
WAPS	WA Pointer Save Registers	13
WAPSP	WAPS Pointer	14
WAU	Working Registers A Unit Pointer	97
WAUS	WAU Save Registers	97
WB	Working Registers B	10
WBP	WB Pointer	14
WBPS	WB Pointer Save Registers	14
WBPSP	WBPS Pointer	14

List of Figures

List of Figures
(Continued)

List of Tables

List of Tables
(Continued)

References

[1] "BPL - a hardware and software description language", by Ole Brun Madsen, RECAU, University of Aarhus, Aarhus, Denmark, 1972.

[2] "KAROLINE, a network computer project", by Ole Brun Madsen, RECAU, University of Aarhus, Aarhus, Denmark, 1972.

[3] "Microprogramming and Numerical Analysis", by Bruce D. Shriver, IEEE Transactions on Electronic Computers, Special Issue on Microprogramming, July 1971.

[4] "A Small Group of Research Projects in Machine Design for Scientific Computation", by Bruce D. Shriver, Department of Computer Science Report No. 14, University of Aarhus, Aarhus, Denmark, April 1973.

[5] "The Significance of Microprogramming", by R. F. Rosin, to be presented at the International Computing Symposium 1973 in Davos, Switzerland.

[6] "A Viable Host Machine for Research in Emulation", by Robert Dorin, Department of Computer Science Report 39-72-mu, State University of New York at Buffalo Amherst, New York, 1972.

[7] "A description of the Mathilda System", by Bruce D. Shriver, Department of Computer Science Report No. 13, University of Aarhus, Aarhus, Denmark, April 1973.

[8] "A Users Manual for the Simulated Rikke-Mathilda System on the CDC-6400", by Ejvind Lynning, Eric Kressel, Hans Ole Sandberg Andersen, Ib Holm Sørensen.

[9] "The RIKKE-1 Reference Manual",
by Eric Kressel and Jørgen Staunstrup, Department of Computer Science Manual No. 7, University of Aarhus, Aarhus, Denmark, April 1974.

Micro
Archives
6-32

Staunstrup, Jørgen.
A description of the RIKKE 1 system / by Jørgen Staunstrup.-- Aarhus, Denmark: Department of Computer Science, Institute of Mathematics, University of Aarhus, 1974.
(DAIMI PB-29)

I. Title.