A DOCUMENTATION/DEVELOPMENT MODEL FOR EXTENDING THE INSTRUCTION SET OF A MINICOMPUTER

BY

SHAH FAROOQ ALAM

B. S. Aligarh University, 1968

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

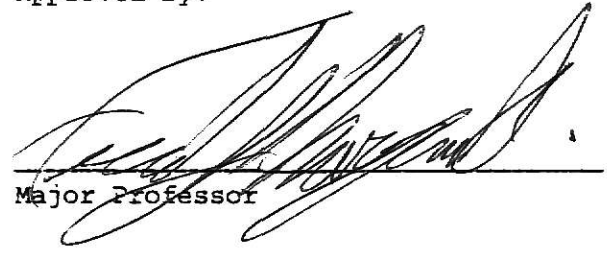
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1977

Approved by:



Major Professor

ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE

THIS BOOK CONTAINS NUMEROUS PAGES WITH THE ORIGINAL PRINTING BEING SKEWED DIFFERANTLY FROM THE TOP OF THE PAGE TO THE BOTTOM.

THIS IS AS RECEIVED FROM THE CUSTOMER.

ILLEGIBLE

THE FOLLOWING DOCUMENT (S) IS ILLEGIBLE DUE TO THE PRINTING ON THE ORIGINAL BEING CUT OFF

ILLEGIBLE

TABLE OF CONTENTS

REFERENCE GUIDE FOR MAJOR MODULES

Module Name	High-Level	Microcode	Verification
ADDSUB	3.2	5.24	5.29
CONVRT	---	5.3	5.4
MIX	3.5	5.8	5.11
INTEG	3.7	5.20	----
FLOAT	3.8	5.19	----
ISCBJ	3.6	5.17	----
COMPAT	3.4	5.22	----

LIST OF FIGURES

I. INTRODUCTION

The intent of this report is to provide a demonstration of a microprogrammed implementation of some array operations. The machine chosen for this purpose is an INTERDATA/85, owned by the Department of Computer Science at Kansas State University.

Array operations are desirable as most numerical and scientific computing requires extensive use of vectors and matrices. A number of languages, most notably APL, have been provided which support vector and matrix manipulation. It is clear that a language of the scope and power of APL would probably be difficult to implement on a minicomputer. However, it is felt that even a language instruction set that allowed some array operations would increase the usability of a conventional architechture minicomputer.

This report therefore, tries to develop an orderly methodology of implementation, rather than attempt an actual implementation. Operations are grouped into modules, and one of the modules is discussed. Later on, one of the operation (ADDSUB) is discussed in further detail. The intent is to provide a documentation/ developement model that can be easily extended.

The report contains the following sections:

1. Selection of instructions. The criteria for selections are discussed.

2. Specification. Complete specifications and error conditions are given for one group of instructions.

3. High-level algorithms. One of the instructions, ADDSUB, is chosen. High-level algorithms are given both for the mainline and for each called routine.

4. Introduction to Microcoding. A tutorial on micro programming for the INTERDATA/85.

5. Microcode. The actual microcode for routines in (3) is given. Assertions are in the text. These assertions surround a piece of code, and are the conditions which hold true after the code is executed. These assertions aid both in verification and in relating the code to the high-level algorithms.

The code is verified by means of a walk-through with some input data. The walk-through is performed for three of the major routines-as these routines contain enough operations of interest so that the reader should have no difficulty in grasping the remainder of the microcode, and convincing himself of its corectness.

It should be noted that a walk-through does not necessarily imply proof of correctness. However, it is a good technique, and has been demonstrated in practice to aid in understanding the code. It also demonstrates at least partial validity. In our case, for those routines for which a walk-through is performed, the parts that are not covered by a set of data will mostly be seen to be symmetric to the parts that are covered.

1.1 SELECTION OF OPERATIONS.

In this part, a number of primitives are proposed which support a language that can be used for operations on vectors and matrices. Some of the primitives that would be required can be grouped into three modules. (see (1) and (2) discussion of APL operations).

(i) A module consisting of a basic set of operations. These in our opinion are:

--Addition/subtraction of vectors and matrices.
--Dot product of vectors and inner product of matrices.
--A subscript capability for a vector or matrix. This can be generalized to taking a section of a matrix.
--Assignment of vectors/matrices.
--Type conversion, from floating to fixed point and from fixed to floating point.

(b) A module consisting of additional vector/matrix operations.

--Input/output to or from an array.
--Multiplication/division of an array by a scalar.
--Inverse of a matrix.
--More specialized computations such as the eignvalves of a matrix.

(c) A Module consisting of supporting routines. Two routines would be particularly necessary.

--Allocation of array space.
--Deallocation.

It would be advantageous if at least part of these routines were providied at the level of microcode instead of software. The advantages of microcode as opposed to conventional software are:

(A) Using software routines involves considerable software support, in the form of linking programs. This can both complicate the execution of a program, and increase its overall execution time.

(B) Execution time is increased. The execution of a program coded in mocrocode is less than the execution time of a program coded in machine language. Each machine instruction, even if it maps onto one micro-instruction, requires extra processor time for instruction fetch and decode.

(C) Core storage is saved, since the micro executes out of control store. This can be an important consideration in minis.

The limiting factor in utilizing microcode is, of course, the limited amount of Control Store available. Thus the DCS on the Model 80 contains space for just 1000 instructions or data items.

Since space in control store is limited, it is necessary to be selective in the routines selected for implementation in microcode. Module (c) is already available to the user of the INTERDATA as part of most operating systems. Module (b) is too specialized, and does not have enough general utility. Module (a) on the other hand, provides a basic capability and is worth taking down into microcode.

However, if module (a) is in microcode and module (c) in machine code, then a special kind of enviroment is imposed on the user of the system. All array space must be allocated before an array instruction is used. Further, if the result of an instruction is too large to fit into the result array, then an error code is returned. In an interactive environment, the user would have the option of attempting the instruction again.

In the remainder of this report, specifications are given for the routines in module (a). A high-level algorithm and microprograms are given for one of the routines in module (a). It is expected that this report will provide enough information to aid in any further implementations of array operations.

II. SPECIFICATIONS FOR INSTRUCTIONS

In order to perform operations on arrays, we need to know some of their attributes. The attributes required are:

(1) TYPE-As the operation to be performed (floating or fixed-point add for example), depends on the type of the operand, hence the TYPE must be known.

(2) ORGANIZATION-The definition of an operation, depends on whether its operands are vectors or matrices. Hence the organization of operands needs to be known.

(3) MAXIMUM SIZE-This information is required in order to ensure that the result of an operation never exceeds the maximum space allocated for an operand.

(4) NUMBER OF ROWS & COLUMNS. This information is needed for all array operations.

There are two possible ways in which these attributes can be stored.

(i) As a table
(ii) As a tag which precedes the actual values in the array.

The second form has been chosen. There are two reasons for this choice.

(i) All the information required by the operation is provided along with the operand. No extraneous table lookups have to be performed.

(ii) Using a table would require space allocation for table entries. Since space allocation is not done at the level of microcode, this option would pose complex linkage problems between firmware and software.

2.1 TAGGED VARIABLES.

The instructions described below act on tagged variables, consisting of a 8-byte tag preceding the actual values of the array. The tag contains the following information.

1--Maximum length allocated for the data structure. This length includes the length of the tag.

2--Type. There are two types, TYPE=2 indicates integer, TYPE=4 indicates real or floating point.

3--Organization. There are two kinds of organization possible. These are ORG=0 indicating a vector, ORG=1 indicating a matrix.

4--The number of elements in a vector, (NEL). This field also contains the number of rows (NROW) for a matrix.

5--The number of columns in a matrux. This field is always set to one (1) for a vector.

The tag is followed by the set of values associated with the array. (see section 5-1 for more details).

2.2 NOTATION:

The following notation will be used throughout the report.

-An object, (the operand of an instruction) is indicated by a capital letter, A,B,C,...etc.

If X is an object,

Then BASE (X)-indicates the starting location of an object X.

LN (X)- is the maximum allocated length for an object X.

ORG (X)-is the organization of an object X.

TYPE (X)-is the type of an object, X.

NROW (X)-is the number of rows in a matrix, X.

NCOL(X)-is the number of columns in a matrix, X.

NEL(X)- is the number of elements in a vector X.

OH ( )- indicates "one of" the set within parenthesis.

If X contains address of a memory location then,

CN (X)-means contents of the memory location whose address is given by X.

STOR(X)-This always appears on the left-hand side of an assignment. It means that the value on the right-hand side is assigned to the location whose address is given by X.

2.3 SPECIFICATION OF ADD/SUBTRACT INSTRUCTION (ADDSUB)

The instruction has the form

ADDSUB (A,B,C,OP,RES)

where

A,B,C-are the beginning locations of three tagged variables. They contain absolute memory locations.

OP-is a switch that selects between addition and subtraction. OP=0 indicates addition, OP=1 subtraction.

RES-is the result of the operation. If RES=0, then normal termination is indicated. Otherwise, an error has occured, and RES contains a code giving the type of error.

The instruction checks for a number of error condition. These are listed below.

1--If OP$\neq$0 H(0,1) , RES = 1.

2--Test for a valid tagged variable. This test will be referred to as ISOBJ, and is performed for A and B. Only the first part of this test, the length check, is performed for C. If the object being tested is a valid object, then the test returns a zero (0), Otherwise, it returns an error code > 0.

Test of addition (subtraction) compatible operands.

3a. If org (A) $\neq$ org (B), return 2.
3b. Otherwise, if nrow (A) $\neq$ nrow (B), or ncol (A) $\neq$ ncol (B), return 3.

Note that (3b) applies to vectors as well as matrices. In the case of vectors, NROW is the same as NEL, and NCOL is always set to 1.

The type, and size of C is determined entirely by the type of A and B.

The type of C is given by the following rules.

A	B	C
Float	float	float
Float	fix	float
Fix	fix	fix

In the case of mixed operands, conversion of the fixed-point operand to floating-point occurs before addition/subtraction is carried out. This conversion is done on an element-by-element basis.

The organization of C=ORG (A) or ORG (B). The size of C is equal to the size of the floating-point operand (in the case of mixed or floating operand). It is equal to the fixed-point operand for fixed point operands.

A final error check also occurs at this point.

If size (C) > Ln (C), then RES = 3 and Ln (C) is set to the amount of space required. The user of the system has the choice of reallocating and retrying the instruction.

The contents of C are:

Ci=Ai±Bi i=1,-----NEL (A), if A and B are vectors.

Ci,j=Ai,j±Bi,j i=1------NROW (A)
j=1------NCOL (A)

if A and B are matrices.

2.4 SPECIFICATION OF ISOBJ:

This test is performed as a part of most of the instructions in this section. The test, ISOBJ for an object, X, consists of the following steps.

*Length check

a. If X $\leq$ 0, return 11
b. If (base (X) + Ln (X)) > Upl, return 12.
c. If Ln (X) $\leq$ 0, return 13.

Base (X) is the starting location of the object X. Ln (X) is the maximum space allocated to the object (see description of tag). Upl is the upper limit of memory on the INTERDATA.

*end length check.

*Type Check

d. If TYPE (X)$\neq$OH(fix, float), return 14.

*organization check

e. If ORG (X) $\neq$ OH(Vector, matrix) return 15.
f. If NEL (X) = 0 (or NROW (X) = 0) or NCOL (X) = 0, then return 16. NEL and NROW occupy the same field. The test applies to both vectors and matrices, since for vectors NCOL (X)=1.
g. If (TYPE (X) * NROW (X) * NCOL (X) + 8) $\geq$ Ln (X), then return 17.

If the actual array length is larger then the maximum space allocated to the array, an error occurs. The array length includes 8 bytes for the tag.

2.5 ASSIGNMENT

The form of the instruction is.

ASSIGN (A,B,RES)

A,B-are location of tagged variable
RES-is the result of the operation. A RES=0 indicates normal termination, and a RES > 0 indicates an error.

ERROR CHECKING:

(1) Test for a valid object, ISOBJ, is performed for B. If the test returns a value > 0, then RES is set to this value, and an error exit occurs.

(2) The first part of the test ISOBJ (length check) is performed for A. If the test returns a value > 0, then RES is set to this value, and an error exit occurs.

The result of the operation is to assign the type, size and organization of B to A. If A is not large enough, then RES is set to and error code of 2 and Ln (A) is set to the amount required. An error exit occurs.

2.6 INNER PRODUCTS.

The form of the instruction is:

INPROD (A,B,C,RES)

A and B are the location of tagged variables.

C--is the location of the result, If A and B are both vectors, then C is an untagged scalar location of size 4, (i.e. large enough to hold either a fixed or floating-point result). If either A or B are matrices, then C is a tagged array location.

RES--is the result of the operation. RES=0 indicates normal termination. RES > 0 indicates an error.

The instruction checks for the following error conditions:

1--The test for a valid object is performed for A and B. If the value returned by the test is 0, an error exit occurs, with RES set to the returned value.

2--The LENGTH-CHECK part of ISOBJ is performed for C if ORG(A)=matrix or ORG(B) = matrix.

Test for compatible operands

Case 1:

if ORG(A)=vector and ORG(B)=matrix, then if NEL (A)≠ NROW(B), return 1.

Case 11:

if ORG(A)=matrix and ORG(B)=vector, then if NCOL(A) ≠ NEL(B), return 2.

Case 111:

if ORG(A)=matrix and ORG(B)=matrix, then if NCOL(A) ≠ NROW(B), return 3.

Case 1V:

If ORG (A)=vector and ORG (B)=vector, then
if NROW (A)≠ NROW (B), then return 4.*

If none of Cases1-1V apply, return zero (0)

If the value returned is > 0, an error exit occurs and RES is set to the value returned.

The treatment of operands, and the type and size of C is the same as in the case of ADDSUB.

The instruction obtains either the dot product of two vectors, the inner product of two matrices, or the product of a vector and a matrix.

The treatment of operands, and the type and size of C is handled in the same way as in ADDSUB. If C does not have sufficient space allocated to it, an error code is returned.

*for a vector, NROW is the number of elements and NCOL is 1.

III. HIGH-LEVEL ALGORITHMS

High-level algorithms are given for ADDSUB and the routines called by it. The algorithms are given in a PL/I-like language. Explanatory comments are given in the text where required. An organization chart for ADDSUB is also given.

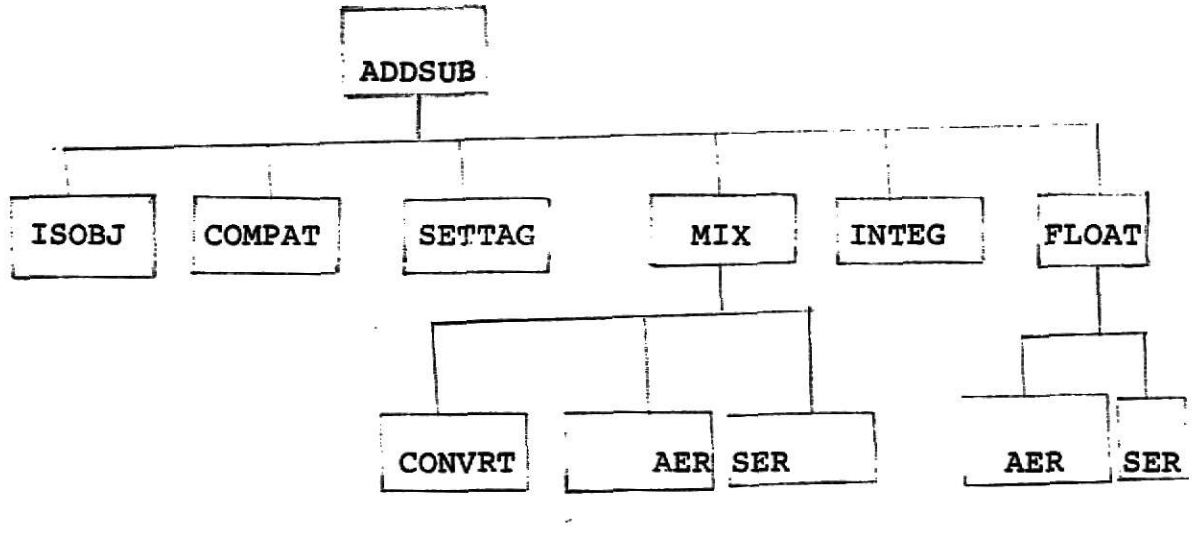


FIG. 1: ORGANIZATION DIAGRAM OF ADDSUB

However, note that all of the routines given in the figure above do not appear as high-level algorithms. AER & SER are user instructions for the INTERDATA. SETTAG & CONVRT are so machine-dependent that they are given only in microcode.

3.1 ADDSUB: This routine adds or subtracts arrays whose beginning addresses are given as its first three parameters--A,B, and C. The selection between addition and subtraction is by means of the fourth parameter--OP. The result indicated by the fifth parameter RES.

```
PROC  ADDSUB  (A, B, C, OP, RES):

*check for OP.

     if OP≠0 or 1 then do;
     RES = 1;
     Return;
     end;

*test to see if A is a valid object.

     P = 0;
     Call ISOBJ  (A,P);
     if P ≠ 0 then do;
     RES = P;
     Return;
     end;

*if P=0, then the entire test is performed.  Otherwise
*only the length-check.  The result is also returned in
*P.  If P=0 on return, A is a valid object.

*test B

     P=0;
     Call ISOBJ  (B,P);
     If P ≠ 0 then do;
     RES=P;
     Return;
     end;

*test C.  Only length-check.

     P=1;
     Call ISOBJ  (C,P);
     If P ≠ 0 then do;
     RES=P;
     Return;
     end;
```

```
*test for Compatibility of operands for addition.

     Call COMPAT (A,B,R);
     If R ≠ 0 then do;
     RES=R;
     Return;
     end;

*Compat test for compatibility

     AFL:  If TYPE (A) = FLOAT then do;
           L=TYPE (A) * NROW (A) * NCOL (A);

     *L is the actual length required of C.
           If L > Ln (C) then do;
           RES=3;
           Return;
           end;

           RES=Ø;
           Call SETTAG (C,A);

    ABFX:  If TYPE (B) = FIX then do;
           P=1;
           Call MIX (A,B,C,P);

     *MIX performs mixed mode addition/subtraction
     *P=1 indicates that the first parameter is
     *floating-point.  P=2, the second parameter.

           Return;
           End;
           Else do;

    ABFL:  Call FLOAT (A,B,C);

     *Float performs floating-point addition/subtraction.

           Return;
           End;

     End;

     AFX:  Else do;

           L=TYPE (B) * NROW (B) * NCOL (B) + 8;
           If L > Ln (C) then do;
           RES=3;
           Return;
           End;
```

```
                RES=Ø
                Call SETTAG (C,B);
                If TYPE (B) = FIX then do;
                Call FIX (A,B,C);
                End;
                Else do;

                P=2;
                Call MIX (A,B,C,P);
                Return;
                End;

End:
End ADDSUB;
```

3.2 PROC COMPAT (A,B,R);

*This procedure tests A and B for compatibility for addition.

```
     if ORG (A) ≠ ORG (B) then do;

          R=2;
          Return;
          End;

     if NROW (A) ≠ NROW (B) or NCOL (A) ≠ NCOL (B) then do;

          R=3;
          Return;
          End;
```

*This test applies to both vectors and matrices. In the *case of vectors, NROW is the same as NEL and *NCOL is always 1.

```
          R=0; Return;

End COMPAT;
```

The procedures MIX follows on the next page. This procedure calls on INTERDATA floating-point addition and subtraction routines by means of two special locations-LOCAER and LOCSER. These locations contain executable code, in the form of floating point addition subtraction macro instructions. This can be loaded into LOC and directly executed. This point is covered in additional detail in Chapter 5.

IV. MICROPROGRAMMING FOR INTERDATA/85

This chapter provides a small tutorial on Microprogramming. The purpose of this tutorial is to prepare the reader for reading the code given in the next chapter.

Before reading this tutorial, it is necessary to review the material given in the Model 80 Micro-instruction Reference Manual (4). In particular, it is necessary that the reader review Fig. 2 and the description on page 4, which have been reproduced as Appendix A of this report, (see also (5) and for a discussion of assembly version of micro-instructions).

The tutorial itself consists of three major parts:

--Register Usage.

--Memory Control Options.

--Small examples.

--A method of adding new instructions to the instructions set of the INTERDATA.

4.1 REGISTER USAGE

RR-is a Return Register. It is used to call subroutines, and contains the Return Address, which is placed in RR by a BAL instruction as follows:

```
BAL    SUB (RR)
```

This instruction results in depositing the address of the next fullword into RR, and then branches to the location (symbolically) given by SUB.

MR7 will always be assigned as RR.

MAR-Memory Address Register. A data read (DR) results in reading the contents of the location pointed to by MAR. The value read is placed in MDR (see below). MAR is also used for write operations. A data write (DW) results in writing the contents of MDR into the location pointed to by MAR.

MDR-Memory Data Register. This holds the contents of the last location read. On a data write, the location pointed to by MAR receives the contents of MDR. MDR is used in all transfers to/from main memory.

LOC-The LOC register always points to the address of the next machine instruction to be executed.

S bus, A bus, B bus--The S,A,B, busses contain the contents of the registers whose names appear in the S,A,B,fields of a micro instruction (see page 4 of Reference manual).

YD-The YD register is the register whose name appears in the YD field (bits 12-15) of the machine instruction being executed. Thus a register need not be named explicitly. (See page 11 of Reference Manual).

FR0, FR2-These are the floating point registers of the machine. They are actually implemented as fixed fullword locations in memory.

LOC 0 is FR0, LOC4 is FR2 etc.

E flag-This flag is set to indicate the result of the last arithmetic operation. The only settings we will be concerned with are:

G--Greater than zero
Z--Equal to zero
L--Less than zero
(see page 8 of Reference Manual).

It should be noted that all of the registers described above are all program addressable, and can be used in any way by a user program.

4.2 MEMORY CONTROL OPTIONS:

There are number of memory control options available with RR Control instruction (see page 7 and 8 of Reference Manual). They are summarised as bit patterns on Page 10 of Reference Manual. Their symbolic equivalents are given in the MICROCODE ASSEMBLER REFERENCE MANUAL. (5).

The options used are:

DR--reads the contents of location in memory pointed to by MDR into MDR.

DR2--increments MAR by 2, and then does DR.

DW--writes the contents of MDR into location (in memory) pointed to by MAR.

DW2--increments MAR by 2 and then does DW.

IR--An instruction read from location pointed to by MAR and LOC. A fullword is read. The first halfword is placed in IR (Instruction Register), the second halfword in MDR.

IR4--Increments MAR,LOC by 4; and then does an IR.

D--Decode. Interprets the machine instruction in IR.

4.3 EXAMPLES:

Suppose that P,Q,R are registers

$C(P)=200_{16}$.	C(200)=0002.
$C(Q)=400_{16}$.	C(400)=0004.
$C(R)=600_{16}$.	C(600)=0006.

then:

(a) L MAR,P,DR

Results in
MAR=200
MDR=0002

(b) A MAR,P,Q,DW

Results in

MDR=0002
MAR=200 + 400=600_{16}.
C (600) = 0002

(c) S NULL,P,Q,E
BALL PI (RR)

NXT (next statement)

First S bus=P-Q= -200.
E is set to LT ∅

The branch is taken to PI. RR contains address of NXT. The next micro instruction is taken from PI.

4.4 ADDING NEW INSTRUCTIONS:

The instruction will be implemented by means of the ECS machine instruction.

The ECS machine instruction has the form;

ECS R_1, A (X_2)

and appears in storage as;

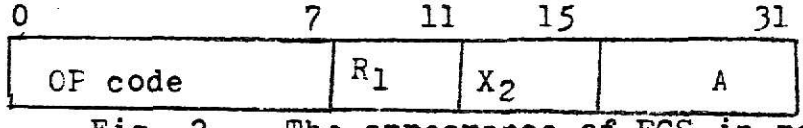


Fig. 2 The appearance of ECS in memory.

The instruction is executed as follows:

--The expression (BL+R_1) is formed. BL is the beginning location of writeable Control Store--which is available in a Machine Control Register (MCR). R_1 is the contents of bits 7-11.

Thus 16 different location are addressable by setting R, to different values. These contain a transfer vector which transfers control to different emulation sequences of (extended) machine instructions.

e.g. suppose the name of the Add/subtraction is ADDSUB. Suppose location BL in Control Store contains:

BAL ADDSUB (RR)

and the machine instruction

ECS	C	X_2	A

is executed.

Control would transfer to location BL+C=BL. The BAL instruction would be executed, and control would go to ADDSUB. In the actual microcode, the reference to ADDSUB in the BAL instruction would, of course, be replaced by the actual Control Store Address of ADDSUB. (see (6) for a discussion of the ECS instruction).

V. MICROCODE

The microcode for implementing the high-level algorithms in chapter III is given. First, the data structures are described from an implementation viewpoint. Then the code for ADDSUB and each of its subroutines follow. The code is followed by verification with input data.

5.1 DATA STRUCTURES IN MEMORY:

TAGGED VARIABLES

Each tagged variable consists of a tag followed by its associated values. The tag contains the following information.

--LN. The maximum space allocated for the variables. This field is 2 BYTES
--TYPE. The type of the variable, 1 BYTE.
--ORG. The organization of the variable, 1 BYTE.
--NROW. The number of rows, 2 BYTES.
--NCOL. The number of columns, 2 BYTES.

This tag is followed by the values of the array. Each fixed-point entry takes 2 BYTES and each floating-point entry takes 4 BYTES.

RSAV--a register save area, 20 bytes in length. Micro registers are saved in this area if required.

EXEC --This is an area, 16 bytes long which contains executable code. The contents of EXEC are:

AER	0	2	RETURN	SER	0	2	RETURN

Fig 3 The contents of EXEC