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

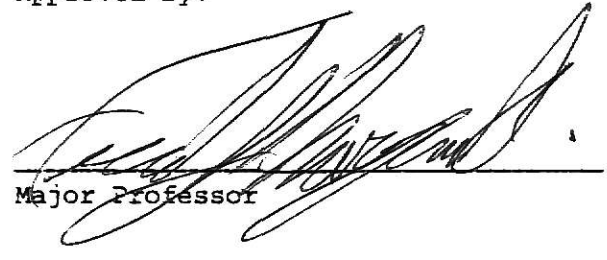
MASTER OF SCIENCE

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

Document
LD
2668
R4
1977
A42
C.2

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.

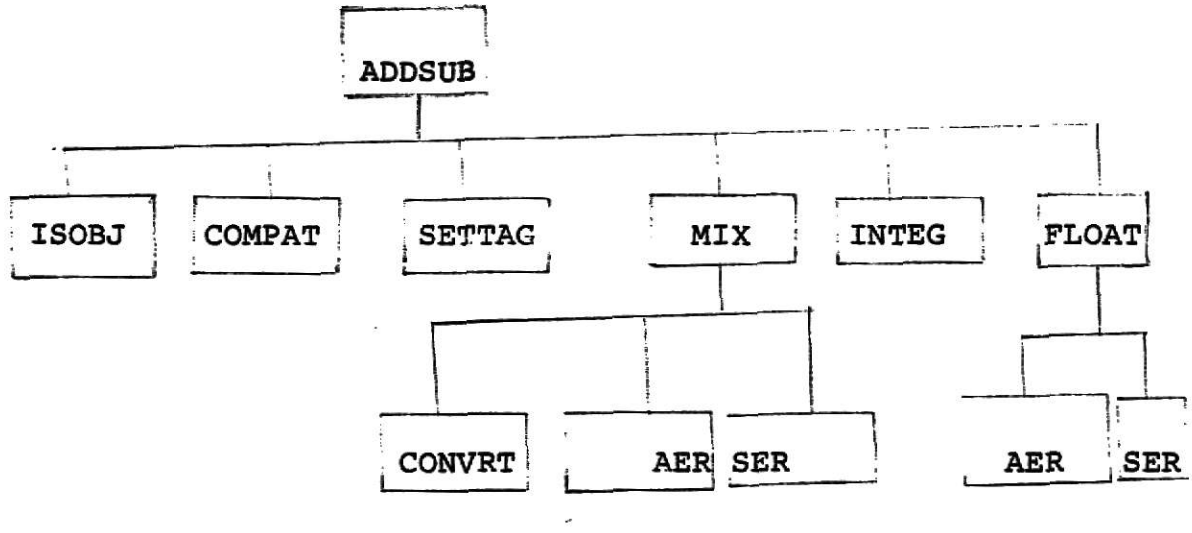


FIG. 1: ORGANIZATION DIAGRAM OF ADDSUB

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;

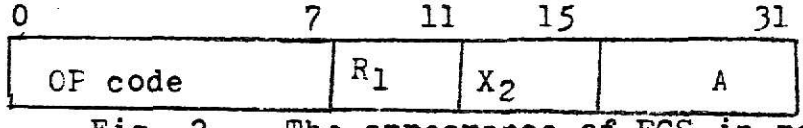


Fig. 2 The appearance of ECS in memory.

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

The AER and SER instructions are the floating-point add and subtract instructions. The RETURN instruction returns control to the routines which transferred control to AER and SER.

EXAMPLES-- The apperance of the tag for a matrix is described.

--A (2×2) matrix consisting of floating-point elements.

40	4	0	02	02
LENGTH	TYPE	ORG	NROW	NCOL

The maximum space allocated is assumed to be 40 bytes.

5.2 POINTERS IN CONTROL STORE

A number of pointers are maintained in Control Store. These are:

LOCAER-- The beginning location of EXEC.
LOCSER--The location EXEC + 8.
LOCRSV--The location of RSAVE, the register save area.

As the Control Store is only fullword addressable, and all of these pointers are halfword values, the actual values of the pointers are in the first halfword of the Control Store location.

5.3 CONVRT:

This routine converts a fixed-point number given in a micro-register (refered to as P) to floating-point form. The result is stored in a floating-point register whose address is contained in MAR (the Memory Address Register).

```
        *test for negative or positive number.

CONVRT  L       MDR,P,E         CV1
        BALL    NEG (MRØ)       CV2

        *This point is reached if the number ≥0. In this case, the
         number is simply converted to floating-point form as a power
         of 16.

POS     LI      MDR,4600        CV3
        L       NULL,NULL,DW    CV4
        L       MDR,P,DW2       CV5
        BAL     (RR) (MAR)      CV6

        *This point is reached if the number <0. In this case,
         first the number is converted to 2's complement. The
         complement is then stored as a power of 16. The comple-
         ment is obtained by taking an EXCLUSIVE OR with 'FFFF'
         and adding 1 to it.

NEG     LI      MDR,'C600'      CV7
        L       NULL,NULL,DW    CV8
        XI      MDR,P,'FFFF'    CV9
        AI      MDR,MDR,1       CV10
        L       NULL,NULL,DW2   CV11
        BAL     (RR) (MAR)      CV12
```

VALIDATION:

(a) Suppose that at the time that CONVRT IS CALLED, the register contents are:

P= -20 in hex=FFEC
MAR=0, the address of FR0.

Then a trace of CONVRT is shown. All numbers are in hex.

CV1 MDR = FFEC
E-flag = LT ∅

CV2 Branch to NEG

CV7 MDR = 'C600'

CV8 STOR (0) = 'C600'

STOR (N) indicates memory location N.

CV9 MDR = 0013

The Exclusive OR of FFFF with FFEC is 0013.

CV10 MDR = 0014
CV11 STOR (2) = 0014

Locations 0-3 (Floating Register 0) now contains.

C6000014

The mantissa M=000014.
Characteristic C=11000110=46=6 in excess 64 notation
and sign bit S=1, indicating negative number.

Thus the number in FR0 is:

$=16^{6} \times (0 \times 16^{-1} + 0 \times 16^{-2} + 0 \times 16^{-3} + 0 \times 16^{-4} + 1 \times 16^{-5} + 4 \times 16^{-6})$

= 000014

Since the sign bit is negative, this is the representation of -20.

(b) Suppose the register contents are

MAR=0
P= +20=0014 in hex

```
CV1         MDR      =   0014
            E-flag   =   G
CV2         No branch
CV3         MDR      =   '4600'
CV4         STOR(0)  =   '4600'
CV5         MDR      =   '0014'
CV6         STOR(2)  =   '0014'
```

FRC now contains

46000014

Mantissa, M=000014
Characteristic, C=6 in excess 64 notation
Sign=0 or positive.
Hence the number is +20

5.4 CALL AND RETURN:

Routines MIX and FLOAT require the use of floating point addition and subtraction routines. These routines are already available as INTERDATA machine instructions, and are called AER and SER, respectively.

To use these routines, executable user code is placed in the EXEC data structure. A special macro instruction RETURN, is also provided which allows for return of control. The macro instructions in EXEC-AER or SER are invoked by means of a special subroutine, CALL.

```
CALL      LI          MAR,RSAVE
```

*RSAVE is the address of the register save area in memory.

```
          L           MDR,MR1,DW2
          L           MDR,MR2,DW2
          L           MDR,MR3,DW2
          L           MDR,MR4,DW2
          L           MDR,MR5,DW2
          L           MDR,MR6,DW2
          L           MDR,MR7,DW
```

*The CALL routine does not save LOC. The calling routine saves *LOC in the first two bytes of RSAVE.

RETURN restores the contents of registers, and then branches back.

```
RETURN    LI          MAR,LOCRS,I
          I           NULL,NULL,DR
          L           LOC,MDR,DR2
          L           MR0,MDR,DR2
          L           MR1,MDR,DR2
          L           MR2,MDR,DR2
          L           MR3,MDR,DR2
          L           MR4,MDR,DR2
          L           MR5,MDR,DR2
          L           MR6,MDR,DR2
          L           MR7,MDR
          BAL         (RR) (MAR)
```

*Branch to the address in RR-RR will be MR7 by convention

5.5 MIX:

This routine performs mixed-mode addition/subtraction. It calls on two sets of routines-CONVRT and AER/SER. CONVRT is called as an ordinary subroutine. The number to be converted is passed in a micro-register.

AER or SER are, on the other hand, executed by storing these macro-instructions as part of the data structure EXEC.

The location of the instruction in EXEC is placed in LCC and MAR. A branch to CALL occurs (see 5-4). Control is returned to MIX by means of the RETURN macro-instruction.

MIX is called with four parameter-A,B,C and P-A,B,and C are the same micro-registers used for the base addresses of tagged variables in the main routine, ADDSUB.P is used to indicate which of the operands are floating-point.

In addition, other registers and locations used are:

OP-contains a code which allows selection between Add/Subtract. This code is at the location of ADDSUB
N-contains the number of times the loop iterates. N is MR1.
X,Y-are working registers.

5.5.1 CODE FOR MIX:

```
*Assertion 1--The product NROW (A) * NCOL (A) is placed in MR1.
*
MIX       LI          MAR,4              M010
          A           MAR,MAR,A,DR       M020
          L           MR1,MDR,DR2        M030
          M           MR0,MR1,MDR        M040

*end assertion 1.
*
*
*Assertion 2--A,B, and C are set pointing to the start of the
*associated array values.
*
          AI          A,A,8              M050
          AI          B,B,8              M060
          AI          C,C,8              M070

*end assertion 2

*Assertion 3--after executing, M160 or M150, LOC contains
 either the address of AER or SER in EXEC.  The instruction
 pointed to is determined by OP, whose memory location
 is given by the present contents of LOC.
          L           MAR,LOC,DR         M080
          NI          MDR,MDR,'000F'     M090
          L           NULL,MDR,E         M100

*save the previous contents of LOC in RSAVE.

          LI          MAR,LOCRS , I      M110
          L           MDR,LOC, DW        M120
          BALNZ       MINUS (MAR)        M130

PLUS      L           LOC,LOCAER         M140
          BAL         MP (MAR)           M150

MINUS     L           LOC,LOCSER         M160

*end assertion 3.

MP        L           X,RR               M161

Save RR for later use.
```

```
*Assertion 4--Branch to FIRFL if P=1, to SECFL if P≠1.

          LI              MDR,1                 M170
          S               NULL,MDR,P,E          M180
          BALNZ           SECFL (RR)            M190

*end assertion 4.
```

```
*Assertion 5: If the first operand is floating-point, then
*all of the following conditions hold.

*(A) Number of iterations of loop labelled FIRFL=contents
 of MR1.

*(B) During the Ith iteration of loop I=1,---N; A,B, and C
 point to the Ith element of their associated matrices in
 row major order.

*(C) After each execution of FIRFL,

     STOR (C) ← CN (A) + CN (B);
```

```
*Assertion 6: FRØ contains fullword pointed to by A.

FIRFL     L               MAR,A,DR              M200
          LI              MAR,Ø                 M210
          L               NULL,NULL,DW          M220
          L               MAR,A,DR2             M230
          LI              MAR,Ø                 M240
          L               NULL,NULL,DW2         M250

*end assertion6.
```

```
*Assertion7: The corresponding element of B is converted
*floating-point and placed by FR2.

          L               MAR,B,DR              M260
          L               P,MDR                 M270
          LI              MAR,4                 M280
          BAL             CONVRT (RR)           M290

*end assertion 7.
```

```
*Assertion 8: FRØ contains the sum/difference of FR0 and
*FR2.

          BAL             CALL (RR)             M300

*end assertion 8.
```

```
*Assertion 9:  The result in FR0 is stored back in the
*corresponding element of C.

          LI        MAR,Ø                M301
          L         NULL,NULL,DR         M302
          L         MAR,C,DW             M303
          LI        MAR,Ø                M304
          L         NULL,NULL,DR2        M305
          L         MAR,C,DW2            M306

*end assertion 9.

          AI        A,A,4                M310
          AI        B,B,2                M320
          AI        C,C,4                M330

CONTF     LI        P,1                  M340
          S         MR1,MR1,P,E          M350
          BALNZ     FIRFL (MAR)          M360
          BAL       FIN (MAR)            M370

*end assertion 5.
```

*The second loop-SECFL is exactly complementary to FIRFL. In this case, the second operand, B, is floating-point instead of A, and FR2 is used instead of FRØ.

```
SECFL     L         MAR,B,DR             M390
          LI        MAR,4                M400
          L         NULL,NULL,DW         M410
          L         MAR,B,DR2            M420
          LI        MAR,4                M430
          L         NULL,NULL,DW2        M440
          L         MAR,A,DR             M450
          L         P,MDR                M460
          LI        MAR,0                M470
          BAL       CONVRT (RR)          M480
          BAL       CALL (RR)            M490
          LI        MAR,0                M500
          L         NULL,NULL,DR         M510
          L         MAR,C,DW             M520
          LI        MAR,0                M530
          L         NULL,NULL,DR2        M540
          L         MAR,C,DW2            M550

*

          AI        A,A,2                M560
          AI        B,B,4                M570
          AI        C,C,4                M580
```

```
CONTS     LI          P,1              M590
          S           MR1,MR1,P,E      M600
          BALNZ       SECFL (RR)       M610

*
FIN       L           RR,X             M620
          BAL         (RR) (MAR)       M630
```

5.5.2 VALIDATION OF MIX:

The validation will be done with the following data.
The instruction is at 100.
--OP=0, indicating addition
--TYPE (A) = FLOAT
--TYPE (B) = FIX
--P=1
--NROW (A) = NCOL (B) = 2
--NCOL (A) = NCOL (B) = 2
--ORG (A) = ORG (B) = 1, A and B are Matrices.

The following conditions hold at the time that MIX is called:

--A = 200, B = 400 and C = 600. This means that the associated arrays are at locations 208, 408 and 608 respectively.

--A(1,1)=2·0 which appears as C6000002 in floating-point
B (1,1)=2

--The size of each element of the A-array is 4, that of each element of the B-array is 4, and for the C-array it is 4.

--NROW (A) is at location 204, NCOL (A) at location 206.

On return from MIX, the C-array contains the sum of the A and B arrays. The contents of registers A, B, and C are destroyed.

The validation will be done by walking thru each of the assertions in the text for the given input data.

ASSERTION 1:

```
M010        MAR  =  4
M020        MAR  =  204
            MDR  =  2 = NROW (A)
M030        MR1  =  2
            MDR  =  2 = NCOL (A)
M040        (MR0,MR1)  =  2x2=4.  This appears
```

as 00000004, so that the significant part is in MR1.

this completes the proof of assertion 1.

ASSERTION 2:

M050	A = 208
M060	B = 408
M070	C = 608

These are the start of the array values for A, B and C. Therefore, assertion 2 holds for this data.

ASSERTION 3:

M080	MAR = 100 MDR = XXX0

Where X indicates that the hex digit in this position does not matter, These are the contents of first two bytes of the ADDSUB instruction, with OP=0.

M090	MDR = 000C
M100	E-flag = Z
M110	MAR = LOCRS
M120	Contents of LOC are saved in memory location of RSAVE.
M130	NO branch.

PLUS is executed.

M140	LOC is loaded with memory location of AER.
M150	Branch to MP.

As LOC contains the address of AER and OP=C, assertion 3 holds for this data.

ASSERTION 4:

M170	MDR = 1
M180	(1—1) = 0 is formed. E-flag = Z
M190	No branch

As P=1, the branch does not occur. Instead control passes to the next sequential statement, labelled FIRFL. This satisfies the condition imposed by assertion 4.

ASSERTION 5:

The three assertion- 5 (a), 5 (b) and 5 (c) will each be proved in turn.

(5a) The number of iterations of the loop is controlled by the contents of MR1 Now the only instruction within the range of assertion 5 which changes the contents of MR1 in M350.

Suppose that MR1 is set to 4 by MØ4Ø. The contents of MR1 are shown each time that control passes to M350 and the statements following it.

ITERATION 1:

M350	MR1 = 4-1=3 E-flag = GTC
M360	Branch to FIRFL

ITERATION 2:

M350	MR1 = 3-1=2 E-flag = GTC
M360	Branch to FIRFL

ITERATION 3:

M350	MR1 = 2-1=1 E-flag = GTC
M360	Branch to FIRFL

ITERATION 4:

M350	MR1 = 1-1=0 E-flag = Z
M360	No branch
M370	Branch to FIN. This terminates the loop and the routine.

(5b) Initially, A=208, B=408 and C=608. These are the values of A, B, C during the first iteration, and are also the locations of A (1,1), B (1,1), C (1,1).

SECOND ITERATION:

M310	$A = 20C_{16}$	This is A(1,2)
M320	$B = 40A_{16}$	This is B(1,2)
M330	$C = 60C_{16}$	This is C(1,2)

THIRD ITERATION:

M310	$A = 210_{16}$	=A(2,1)
M320	$B = 40C_{16}$	=B(2,1)
M330	$C = 610_{16}$	=C(2,1)

FOURTH ITERATION:

M310	$A = 214_{16}$	=A(2,2)
	$B = 40E_{16}$	=B(2,2)
	$C = 614_{16}$	=C(2,2)

As these are the elements of A,B,C, in row major order, (5b) is verified.

(5c) This assertion will hold if assertions 6,7,8,and 9 hold. The result of executing M200-M250 is to place an element of A into FRØ. The result of executing M260-M29 is to convert an element of B, and put the converted result in FR2.

M300 results in adding/subtracting the contents of FRØ and FR2.

First, it will be shown that assertion 6 holds for this data. Suppose that at the time that control passes to M200, A=208, B=408, and C=608, then a trace of M200-M250 is shown.

M200	MAR = 208
	MDR = C600
M210	MAR = Ø
M220	STOR (0) = C600
M230	MAR = 208
	MDR = ØØØ2, the second half of the floating-point number.

M240	MAR = Ø
M250	STOR (2) = 0002.
Thus	FRØ contains the floating-point number at location 208 or A(1,1)

Next assertion 7 will be verfied.

M260	MAR = 408
	MDR = 2
M270	P = 2
M280	MAR = 4
M290	Branch to CONVRT. The branch occurs

with P containing 2 (in fixed point), and MAR the address of FR2. The contents of P will be converted to floating point and placed in FR2. See initial conditions.

ASSERTION 8:

A branch to CALL occurs in M300. CALL will transfer control to the location contained on LOC. This by virtue of MI40 and assertion3, is the location of AER-LOCAER. Hence, AER 0,2 is executed. This is part of the INTERDATA instruction set, and will result in adding the contents of FR0 and FR2 and placing the result in FR0.

Finally, Assertion 9 is verfied.

ASSERTION9:

M301	MAR = 0
M302	MDR = 1st halfword of result.
M303	MAR = 608 (for the first iteration of loop).
	STOR (608) = 1st halfword of result.
M304	MAR = 0
M305	MDR = 2nd halfword of result (DR2 results in incrementing MAR by 2
M306	MAR = 608
	STOR (60A) = 2nd halfword of result. (MAR is incremented by 2 because of DR2).

Thus the result is stored in FRØ.

After termination of the loop FIRFL, control passes to the statement labelled FIN. This statement restores RR to its previous contents (its contents at the time that MIX was called). M161 has saved these in another micro-register, X. The next instruction branches to the restored address in RR, thereby returning control to the routine which called MIX.

5.6 CODE FOR ISCBJ:

```
*If X > Ø, then branch to ISCB1.

          L              MDR,X,E
          BALG           ISCB1 (MAR)

*Otherwise, return error code of 11.

          LI             X,11
          BAL            (RR) (MAR)

*Test if X + Ln (X) < UPL.  If so branch to ISCB2.

ISCB1     L              MAR,X,DR
          A              MDR,X,MDR
          LI             MAR,UPL
          S              NULL,MDR,MAR,E
          BALL           ISCB2 (MAR)

*Otherwise, return error code of 12.

          LI             X,12
          BAL            (RR) (MAR)

*Test if Ln (X) > Ø.  If so, branch to ISCB3

ISCB2     L              MAR,X,DR
          L              NULL,MDR,E
          BALG           ISCB3 (MAR)
          LI             X,13
          BAL            (RR) (MAR)

*Test if P=1.  If so, return code of Ø.  Otherwise, go on to
ISCB4.

ISCB3     LI             MDR,1
          S              MDR,MDR,P,E
          BALNZ          ISCB4 (MAR)
          LI             X,Ø
          BAL            (RR) (MAR)

*Test if type (X)≠2 or 4.  If so, return error code of 14,
*Otherwise, go on to ISCB5.

ISCB4     L              MAR,X,DR2
          NI             MDR,MDR,'FFØØ'
          SRI            MDR,MDR,8
          LI             MAR,2
          S              MDR,MDR,MAR,E
```

```
          BALZ        ISOB5(MAR)
          LI          MAR,4
          S           MDR,MDR,MAR,E
          BALZ        ISOB5 (MAR)
          LI          X,13
          BAL         (RR) (MAR)

*Test if ORG (X)=0 or 1.  If not, return error code of 15.
*Otherwise, go to ISOB6

ISOB5     L           MAR,X,DR2
          NI          MDR,MDR,'ØØFF'
          L           NULL,MDR,E
          BALZ        ISOB6 (MAR)
          LI          MAR,1
          S           NULL,MDR,MAR,E
          BALZ        ISOB6 (MAR)
          LI          X,15
          BAL         (RR) (MAR)

*Test if NROW (X) > Ø and NCOL (X) > Ø.  If so, go to ISOB7.
*Otherwise, return error code of 16.

ISOB6     AI          MAR,X,4
          L           NULL,NULL,DR
          L           NULL,MDR,E
          BALG        ISOB7 (MAR)
          L           NULL,NULL,DR2
          L           NULL,MDR,E
          BALG        ISOB7 (MAR)
          LI          X,16
          BAL         (RR) (MAR)

*Test if space allocated (Ln (X)) is greater than NROW (X)*
*NCOL (X)+8.  If so, go on to ISOB8, Otherwise, return error
*code of 17.

ISOB7     L           MAR,X,DR
          AI          MAR,MAR,4
          L           MR1,MDR,DR
          L           MR1,MDR,DR2
          M           MRØ,MR1,MDR
          AI          MR1,MR1,8
          S           NULL,P,MR1,E
          BALG        ISOB8 (MAR)
          LI          X,18
          BAL         (RR) (MAR)

*Return code of Ø.

ISOB8     LI          X,Ø
          BAL         (RR) (MAR)
```

5.7 CODE FOR FLOAT:

```
*N = NROW (A) * NCOL (A)

          AI          MAR,A,4
          L           NULL,NULL,DR
          LI          MR1,MDR,DR2
          M           MRØ,MR1,MDR
          L           MR2,MR1

*A = A+8;   B = B+8;    C = C+8.

          AI          A,A,8
          AI          B,B,8
          AI          C,C,8

*Save RR in preparation for CALL to AER or SER.

          L           MRØ,MR7

*FRØ = CN(A (I))

FLOOP     L           MAR,A,DR
          LI          MAR,Ø
          L           NULL,NULL,DW
          L           MAR,A,DR2
          LI          MAR,Ø
          L           NULL,NULL,DW2

*FR2 = CN(B (I))

          LI          MAR,B,DR
          LI          MAR,4
          L           NULL,NULL,DW
          L           MAR,B,DR2
          LI          MAR,4
          L           NULL,NULL,DW2

*Branch to CALL, which passes control to the
*Macro instruction whose address is in LOC
*The address is placed there by the routine which calls FLOAT.

          BAL         CALL(RR)

*On return from CALL, FRØ contains the sum or difference
*STOR (C (I)) = FRØ

          LI          MAR,Ø
          L           NULL,NULL,DR
          L           MAR,C,DW
          LI          MAR,Ø
          L           NULL,NULL,DR2
          L           MAR,C,DW2
```

```
*Test for termination of loop.

          LI              MDR,1
          S               NULL,N,MDR,E
          BALNZ           FLOOP (MAR)

*Return

          BAL             (MRØ) (MAR)
```

5.8 CODE FOR INTEG

```
*N = NROW (A) * NCOL (A)

          AI              MAR,A,4
          L               NULL,NULL,DR
          L               MR1,MDR,DR2
          M               MRØ,MR1,MDR
          L               N,MDR

*A = A+8;    B = B+8;    C = C+8;

          AI              A,A,8
          AI              B,B,8
          AI              C,C,8

*Test OP.  If OP=Ø then addition, otherwise, subtraction

          L               MAR,LOC,DR
          NI              MDR,MDR,'ØØØF'
          L               NULL,MDR,E
          BALZ            INTAD (MAR)

*Perform subtraction

INTSUB    L               MAR,A,DR
          L               MRØ,MDR
          L               MAR,B,DR
          L               MR1,MDR
          S               MDR,MRØ,MR1

*Store in C.

          L               MAR,C,DW

*Increment A,B,and C

          AI              A,A,2
          AI              B,B,2
          AI              C,C,2
```

```
*Test for loop termination.

          LI          MAR,1
          S           N,N,MAR,E
          BALNZ       INTSUB (MAR)
          BAL         (RR) (MAR)

*Perform addition

INTAD     L           MAR,A,DR
          L           MRØ,MDR
          L           MAR,B,DR
          L           MR1,MDR
          A           MDR,MRØ,MR1
          L           MAR,C,DW
          AI          A,A,2
          AI          B,B,2
          AI          C,C,2
          LI          MAR,1
          S           NY,N,MAR,E
          BALNZ       INTAD (MAR)
          BAL         (RR) (MAR)
```

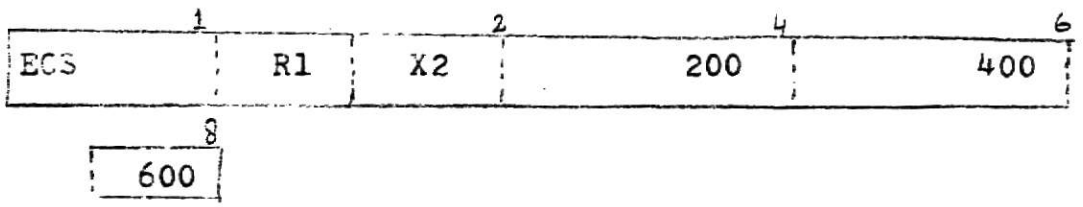
5.9 CODE FOR COMPAT

```
      *Read ORG(A) and ORG(B) into X and Y and compare.

           L          MAR,A,DR2
           L          X,MDR
           L          MAR,B,DR2
           L          Y,MDR
           S          MDR,X,Y,E
           BALZ       CM1 (MAR)

      *If ORG (X) ≠ ORG (Y), then load error code and return

           L          R,2
           BAL        (RR) (MAR)

      *Read NROW (A) and NROW (B) into X and Y and compare.

CM1        LI         MAR,4
           A          MAR,MAR,A,DR
           L          X,MDR
           LI         MAR,4
           A          MAR,MAR,B,DR
           L          Y,MDR
           S          MDR,X,Y,E
           BALZ       CM2 (RR)

      *If NROW (A) ≠ NROW (B), then return with error code

           L          R,3
           BAL        (RR) (MAR)

      *Read NCOL (A) and NCOL (B) and compare

CM2        LI         MAR,6
           A          MAR,MAR,A,DR
           L          X,MDR
           LI         MAR,6
           A          MAR,MAR,B,DR
           L          Y,MDR
           S          MDR,X,Y,E
           BALZ       CM3 (MAR)

      *If NCOL (A) ≠ NCOL (B), return with error code.

           L          R,3
           BAL        (RR) (MAR)

CM#        L          R,Ø
           BAL        (RR) (MAR)
```

5.10 ADDSUB:

The parameters to this routine are A, B, C, OP. A, B, C are memory addresses that require two bytes each, OP requires only 1 byte.

The instruction implementing the ADDSUB instruction is the ECS (Enter Control Store) instruction. The ECS instruction for ADDSUB is as follows:



The three addresses are in bytes 3-6. The X2 field is used to store OP when ADDSUB is called. On return from ADDSUB, the result is stored in bits 0-15 of the instruction.

On entering the ADDSUB instruction, MAR and LOC both contain the location of the instruction. MDR contains the second halfword of the instruction; i-e. bytes 3-4 in the figure above.

5.10.1 CODE FOR ADDSUB:

```
*Assertion 1: Branch to NXT1 if OP=0 or 1. Otherwise,
*an error code of 1 is returned in the first 2 bytes of
*the instruction.

*Assertion 2: A, B, & C are set to base locations of arrays.
*Note that at the start of the instruction, both MAR and
*LOC contain the location of the ADDSUB instruction.

ADDSUB    L       NULL, NULL, DR2      AS010
          L       A,MDR,DR2            AS020
          L       B,MDR,DR2            AS030
          L       C,MDR                AS040

*end assertion 2.

*Obtain OP and test. OP is in the second byte of the
instruction ADDSUB, whose address is in LOC.

          L       MAR,LOC,DR           AS050
          NI      MDR,MDR,'00CF'       AS060
          L       NULL,MDR,E           AS070
          BALZ    NXT1 (RR)            AS080
          L       W,1                  AS090
          S       NULL,MDR,W,E         AS100
          BALZ    NXT1 (RR)            AS110

*If OP≠0 or 1, return error code.

          L       MDR,1                AS120
          L       MAR,LOC,DW           AS130
          L       NULL,NULL,IR4,D      AS140

*end assertion 1.

*Assertion 3: Branch to NXT2 if A is a valid object.
Otherwise, return error code returned by ISOBJ.

*ISOBJ is called to test A. It returns 0 or error code in
*Register X.

NXT1      L       X,A                  AS150
          LI      P,0                  AS160
          BAL     ISOBJ (RR)           AS180
          L       MDR,X,E              AS190
          BALZ    NXT2 (RR)            AS200
          L       MAR,LOC,DW           AS210
          L       NULL,NULL,IR4,D      AS220

*end assertion 3.
```

```
*If the test for A is passed, then B is tested.  The test for
*B is identical to that for A.  A branch to NXT3 occurs if
*B is a valid object.

NXT2      L          X,B                  AS230
          LI         P,C                  AS240
          BAL        ISOBJ (RR)           AS250
          L          MDR,X,E              AS260
          BALZ       NXT3 (RR)            AS270
          L          MAR,LOC,DW           AS280
          L          NULL,NULL,IR4,D      AS300

*Assertion 4:  Branch to NXT4 if the length-check part of
*ISOBJ is valid for C.  Otherwise, return error code.
*ISOBJ returns either Ø or an error code.

NXT3      L          X,C                  AS310
          LI         P,1                  AS320
          BAL        ISOBJ (RR)           AS330
          L          MDR,X,E              AS340
          BALZ       NXT4 (RR)            AS350
          L          MAR,LOC,DW           AS360
          L          NULL,NULL,IR4,D      AS370

*end assertion 4.

*ASSERTION 5:  If A and B are compatible operands, then
*branch to NXT5, Otherwise, return the code returned by COMPAT.

NXT4      BAL        COMPAT (RR)          AS400
          L          MDR,R,E              AS410
          BALZ       NXT5 (RR)            AS420
          L          MAR,LOC,DW           AS430
          L          NULL,NULL,IR4,D      AS440

*end assertion5.

*ASSERTION6:  Branch to AFX if type (A) = FIX, Otherwise,
*branch to AFL
*AFX is the start of the part of the program that is executed
*when A is fixed-point.
*AFL is the start of the part executed when A is floating-
*point.
*AS450-470 extract the type from the tag.  The organization
*byte is zeroed out.
```

```
NXT5        LI          MAR,2                   AS450
            A           MAR,MAR,A,DR            AS460
            NI          MDR,MDR,'FF00'          AS470
            SRI         MDR,MDR,8               AS471
            L           MR1,MDR                 AS480
            LI          X,2                     AS490
            S           MDR,MDR,X,E             AS500
            BALZ        AFX (RR)                AS510

*end assertion 6.

*ASSERTION 7:  If Ln (C) ≤ TYPE (A) * NROW (A) * NCOL (A),
*branch to NXT6.  Otherwise, return an error code of 3.

*The value of TYPE (A) has been placed in MR1 by AS470-471.
*AS 530-540 extract  NROW (A) and multiply MR1 by it, to
*leave the product in MR1.  AS550-560 extract NCOL (A) and
*multiply it by the product TYPE (A) * NROW (A) in MR1.

AFL         LI          MAR,4                   AS520
            A           MAR,MAR,A,DR            AS530
            M           MR0,MR1,MDR             AS540
            L           NULL,NULL,DR2           AS550
            M           MR0,MR1,MDR             AS560

*The value of Ln (C) is placed in MDR and compared against
*the value in MR1.

            L           MAR,C,DR                AS570
            S           MDR,MDR,MR1,E           AS580
            BALG        NXT6 (RR)               AS590
            LI          MDR,3                   AS600
            L           MAR,LOC,DW              AS610
            L           NULL,NULL,IR4,D         AS615

*end assertion 7.

*ASSERTION 8:  Branch to ABFL if type (B)=float, to ALBX,
*Otherwise, ABFL corresponds to the case when A & B are both
*floating-point.
*ALBX to the case when A is floating but B is fixed

NXT6        L           MAR,B,DR2               AS620
            L           MDR,MDR,'FF00'          AS630
            SRI         MDR,MDR,8               AS631
            LI          X,2                     AS640
            S           MDR,MDR,X,E             AS650
            BALZ        ALBX (RR)               AS660

*end assertion 8.
```

```
ASSERTION 9: set  TYPE (C)  =   TYPE (A)
    *             ORG (C)   =   ORG (A)
    *             NROW (C)  =   NROW (A)
    *             NCOL (C)  =   NCOL (A)
    *Produce the sum/difference of A and B, and store in C.

    ABFL                      L X,A              AS670
                              L Y,C              AS680
                              BAL SETTAG (RR)    AS700
                              BAL FLOAT (RR)     AS720
                              BAL WRAP (RR)      AS730

    *This corresponds to the case when A & B are both floating
    *point.

    *end assertion 9.
```

```
ASSERTION 10: set Type (C)  =   type (A)
    *             ORG  (C)  =   ORG  (A)
    *             NROW (C)  =   NROW (A)
    *             NCOL (C)  =   NCOL (A)
    *The sum of arrays of mixed type with the first operand
    *floating, and the second fixed is formed and stored in C-array

    *ALBX corresponds to the case when A is floating-point but
    *B is fixed-point.

    ALBX      L               X,A                AS740
              L               Y,C                AS750
              BAL             SETTAG (RR)        AS760
              LI              P,1                AS770
              BAL             MIX (RR)           AS780

    *MIX is called with the first operand floating-point.
    *P=1 indicates this.

              BAL             WRAP (RR)          AS790

    *end assertion 10.
```

```
    *The remainder of the code corresponds to a complementary
    *treatment for the case when A is fixed point.  A test for
    *Ln (C) ≤ TYPE (B) * NROW (A) * NCOL (A) is carried out.
    *If the test is passed, then integer addition/subtraction
    *(INTEG) is done if B is fixed-point, mixed mode addition/
    *subtraction (MIX) is done if B is floating-point.
    *
```

```
ASSERTION 11: Branch to NXT7 if Ln (C) > TYPE (B) * NROW (A) *
    *NCOL (A).  Otherwise, return an error code of 3.
AFX         L       MAR,B,DR2           AS800
            NI      MR1, MDR,'FF00'     AS810
            SRI     MDR,MDR,8           AS881
            L       NULL,NULL,DR2       AS820
            M       MR0,MR1,MDR         AS830
            L       NULL,NULL,DR2       AS840
            M       MR0,MR1,MDR         AS850
            L       MAR,C,DR            AS860
            S       NULL,MR1,MDR,E      AS870
            BALL    NXT7 (RR)           AS880
            LI      MDR,3               AS890
            L       LOC,MAR,DW          AS900
            L       NULL, NULL,IR4,D    AS910

ASSERTION 12: If type (B)=FIX, then branch to ABFX, otherwise,
    *to AXBL.
 NXT7       L       MAR,B,DR2,'         AS920
            NI      MDR,MDR,'FF00'      AS930
            SRI     MDR,MDR,8           AS931
            LI      X,2                 AS940
            S       MDR,MDR,X,E         AS950
            BALZ    ABFX (RR)           AS960

    AXBL    L       X,A                 AS970
            L       Y,C                 AS980
            BAL     SETTAG (RR)         AS990
            LI      P,2                 AS991
            BAL     MIX (RR)            AS992
            BAL     WRAP (RR)           AS993

    ABFX    BAL     FIX (RR)            AS994

    WRAP    L       NULL,NULL,IR4,D     AS995


    *Register Assignments.

            X       MR0
            N       MR1
            P       MR2
            Y       MR3
            A       MR4
            B       MR5
            C       MR6
            RR      MR7
            LOCRS   A (RSAVE)
            R       MR2
            LOCAER  A (AER)
            LOCSER  A (SER)
```

5.10.2 VERIFICATION

The routine ADDSUB will be verfied by means of a structured walk through with data which is described below.

(A) INPUT DATA:

OP=1 indicating Subtraction
The A-field points to $(200)_{16}$
The B-field points to $(400)_{16}$
The C-field points to $(600)_{16}$

The tags at locations 200, 400 and 600 are:

Ln (A)=28_{16}=40_{10}

ORG (A) =1, A points to a matrix
TYPE (A) =4, A points to a floating point array
Ln (B) =24_{16}
ORG (B) =1, B-array is a matrix
TYPE (B)=4, B-array is fixed-point.
Ln (C)=32_{16}

NROW (A) = NROW (B) = NCOL (A) = NCOL (B) = 2

The instruction ADDSUB is located at memory location $(100)_{16}$

(B) INITIAL CONDITIONS. Before control is passed to ADDSUB, the following conditions hold:

--MAR,LOC = Location of ADDSUB in memory=100_{16}.

--MDR = Contents of 102-103, the second halfword of the instruction. This contains 200.

--OP=0.

The fields in the tags of A,B,C are as follows.

LOC (LN (A))=200
LOC (TYPE (A))=202
LOC (ORG (A)) =203
LOC (NROW (A))=204
LOC (NCOL (A))=206

The array values associated with A start at location 208. The corresponding fields of B and C are at the same displacement with respect to locations 400 and 600 respectively.

The tag for A is shown below.

200	202	203	204	206
28	4	1	2	2

Finally, the TYPE, ORG and NROW and NCOL for C are undetermined at the start. They are set to their values during the execution of ADDSUB.

Result:

The result of the instruction is to form the sum/difference of A and B arrays and place it in C-array. The sum is formed if CP=0, the difference if CP=1.

(C) WALK THROUGH

The walk-through will consist of showing that each of the assertions inserted in the text hold for this data.

ASSERTIONS 1 and 2:

AS010	MDR =	CN (102) = 200
AS020	A =	200
	MDR =	CN(104)=400
AS030	B =	400
	MAR =	106
	MDR =	600
AS040	C =	600

Since A,B,C, contain 200, 400, 600 this proves assertion 2.

AS050	MAR =	100
	MDR =	XXX1. The X indicates that the hex digit in that position does not matter.
AS060	MDR =	0001
AS070	E-flag =	C
AS080	No branch, since E-flag ≠ Z.	
AS090	W =	1.
AS100	The difference (1-1) is formed. As this is zero, E-flag = Z.	
AS110	Branch to NXT1.	

As CP=1 and the branch to NXT1 occurs, this proves assertion 1 for this data.

ASSERTION 3:

AS150	X =	200
AS160	P =	0
AS180	A branch to ISCBJ with RR set to the return value. As A is a valid object, X will contain a zero on return.	
AS190	MDR =	0
	E-flag =	Z
AS200	Branch to NXT2.	

As A points to a valid object, and the branch to NXT2 occurs, this proves assertion 3 for this data.

NXT2	AS230-AS300 result in a branch to NXT3, since B is a valid object. The code in these statements is identical to that of the previous test.

ASSERTION 4:

NXT3 is executed.

AS310	X = 600
AS320	P = 1
AS330	IS-OBJ is called. As C=0 and Ln (C) = 0 the test return 0 in X. Since P=1, the test IS-OBJ returns after completing the length check.
AS340	MDR = Ø E-flag = Z
AS350	Branch to NXT4

Since C has valid length attributes, and the branch to NXT4 occurs, assertion 4 holds for this data.

ASSERTION 5:

NXT4 is executed.

AS400	Branch to COMPAT, Since org (A)= org (B) and NROW (A)=NROW (B), and NCOL (A)=NCOL (B), this routine returns Ø in R.
AS410	MDR = Ø E = Z
AS420	Branch to NXT5.

As A and B point to addition-compatible operands, and the branch to NXT5 occurs, assertion 5 holds for this data.

ASSERTION 6:

NXT5 is executed.

AS450	MAR = 2
AS460	MAR = 202 MDR = 0401, i.e. the type and organization of A.
AS470 & AS471	MDR = 0004. This is precisly TYPE(A).
AS480	MR1 = 0004. The type is stored in MR1 for later use.
AS490	X = 2
AS500	MDR = (4-2) E = G
AS510	No branch.

As the next statement is labelled AFL, contrl passes to this point. Since type (A)=4, i.e. floating-point, assertion 6 holds for this data.

ASSERTION 7:

AFL is executed

AS520	MAR = 4
AS530	MAR = 204 MDR = CN(204). This is NROW(A)=2.
AS540	(MR0,MR1) = 4x2=8. This appears as 00000008, i.e. the significant part is in MR1.
AS550	MAR = 206. MDR = CN(206)=2. This is NCOL(A).
AS560	(MR0,MR1) = 8x2=16. This appears as 00000010 (in hex).
AS570	MAR = 600 MDR = Ln(C)=32
AS580	MDR = (32-10)=22 E = G.
AS590	Branch to NXT6.

X=type(A) * NROW (A) * NCOL (A)=10 and Y=Ln (C)=32. Since X < Y and the branch to NXT6 does occurr, assertion 7 does hold for this data.

ASSERTION 8:

NXT6 is executed

AS620	MAR = 400 MDR = 0201, i.e. the type and organization of B.
AS630 & AS631	MDR = 0002, the type of B.
AS640	X = 2
AS650	MDR = (2-2)=0 E-flag = Z.
AS660	Branch to ALBX.

As type (B)=fix, and the branch to ALBX does occur, assertion 8 holds for this data. Note that ALBX is the part executed when A if floating-point & B is fixed-point.

ASSERTION 10:

ALBX is executed

AS740	X = 200
AS750	Y = 600
AS760	Branch to SETTAG. SETTAG copies the tag pointed to by X into the tag pointed to by Y, (all except for Ln). Return from SETTAG occurs by means of the contents of RR which is loaded with the return address at the time that the call occurs.

As SETTAG sets all the fields required in assertion 10, the first part of assertion 10 holds for this data.

AS770	P = 1
AS780	Branch to MIX. P=1 indicates first operand is floatin-point. MIX produces the difference since OP=1, and stores in C. The correctness of MIX has already been demonstrated. Thus Assertion 10 holds for this data.

Finally, control goes to WRAP, which results in executing the next macro-instruction.

5.11 CODE FOR SETTAG

```
*
        L       MAR,X,DR2
        L       MAR,Y,DW2
*
        AI      MAR,X,4
        L       NULL,NULL,DR
        AI      MAR,Y,4
        L       NULL,NULL,DW
*
        AI      MAR,X,6
        L       NULL,NULL,DR
        AI      MAR,Y,6
        L       NULL,NULL,DW
```

CONCLUSIONS

This experience with microprogramming has led to the conclusion that there are certain features that the microcode must have if it is to be useful in language implementation. These are:

(a) Provide a number of simple arithmetic statements with no memory control or other options in them. Most programs contain a number of very simple arithmetic operations such as adding the contents of registers. If the instructions for these operations also include memory control options, then a lot of the power of the instruction is wasted.

(b) Provide the capability of addressing variable length fields.

(c) Provide variable--sized instruction, so that Control Store space is not wasted. Obviously this is only possible if the micro instruction set contains instructions with varying capabilities. This refers back to (a).

(d) Provide a better means of implementing subroutine calls. The micro-instruction should include the capability of saving registers, and an EXECUTE instruction whose range is a group of instructions.

The Model 80 is not a good machine in this regard. Its micro instruction set is too specialized towards one application; and that is the emulation of its own instruction set.

APPENDIX A:

Two pages from the Micro-instruction Reference Manual (4) have been reproduced. These should give the reader enough background information to be able to follow up IV and V.

The 16-bit Instruction Register (IR) holds the instruction word of the current user instruction. The complete IR may only be loaded directly from the main memory. Bits 8 through 11 of the IR (YD) may be loaded from Bits 12 through 15 of the S Bus and unloaded to the first operand A Bus, Bits 12 through 15. Bits 12 through 15 of the IR (YS) may be unloaded to the second operand B Bus, Bits 12 through 15. The YD field of IR is ANDed with the Condition Code field of PSW to aid the emulation of user Branch instructions.

Bits 0 through 7 of the IR, the user's operation code, are used to address the Privileged/Illegal ROM and, via the op-code to the Address Translator, to address the first micro-instruction of an emulation routine. The Privileged/Illegal ROM is a separate Read-Only-Memory that contains 256 four bit words. This ROM is interrogated prior to entering the micro-subroutine that executes a user instruction. If the op-code in the IR is illegal, or is that of a Privileged instruction and PSW Bit 7 is set, the illegal instruction interrupt is generated. If the user's instruction passes the Privileged/Illegal ROM, the op-code is translated into the corresponding Control Store address. Table 1 shows the standard Model 80 user's repertoire and the resultant Control Store address. A strap option disables the op-code to the Address Translator and makes the op-code itself the ROM address times two. The option also adds 100_{16} to the TRAP Locations when no Translator is used.

The user's sixteen 16-bit General Registers (GR0 thru GR15) may be directly addressed by the micro-program or indirectly addressed by specifying the field, YD or YS, of the IR that contains the appropriate General Register number.

The eight 16-bit Micro-Registers (MR0 through MR7) are available to the micro-program as general purpose registers.

The 16-bit A Bus holds the first operand for arithmetic and logical operations. The 16-bit B Bus holds the second operand. The A and B Busses are input to the Arithmetic Logic Unit (ALU). The ALU performs Addition, Subtraction, Multiplication, Division, Shifting, and Boolean connect functions. The output of the ALU is the 16-bit S Bus.

Input/Output operations are accomplished by gating data from the A and/or B Busses onto the 16-bit I/O Bus and gating data from the I/O Bus onto the S Bus.

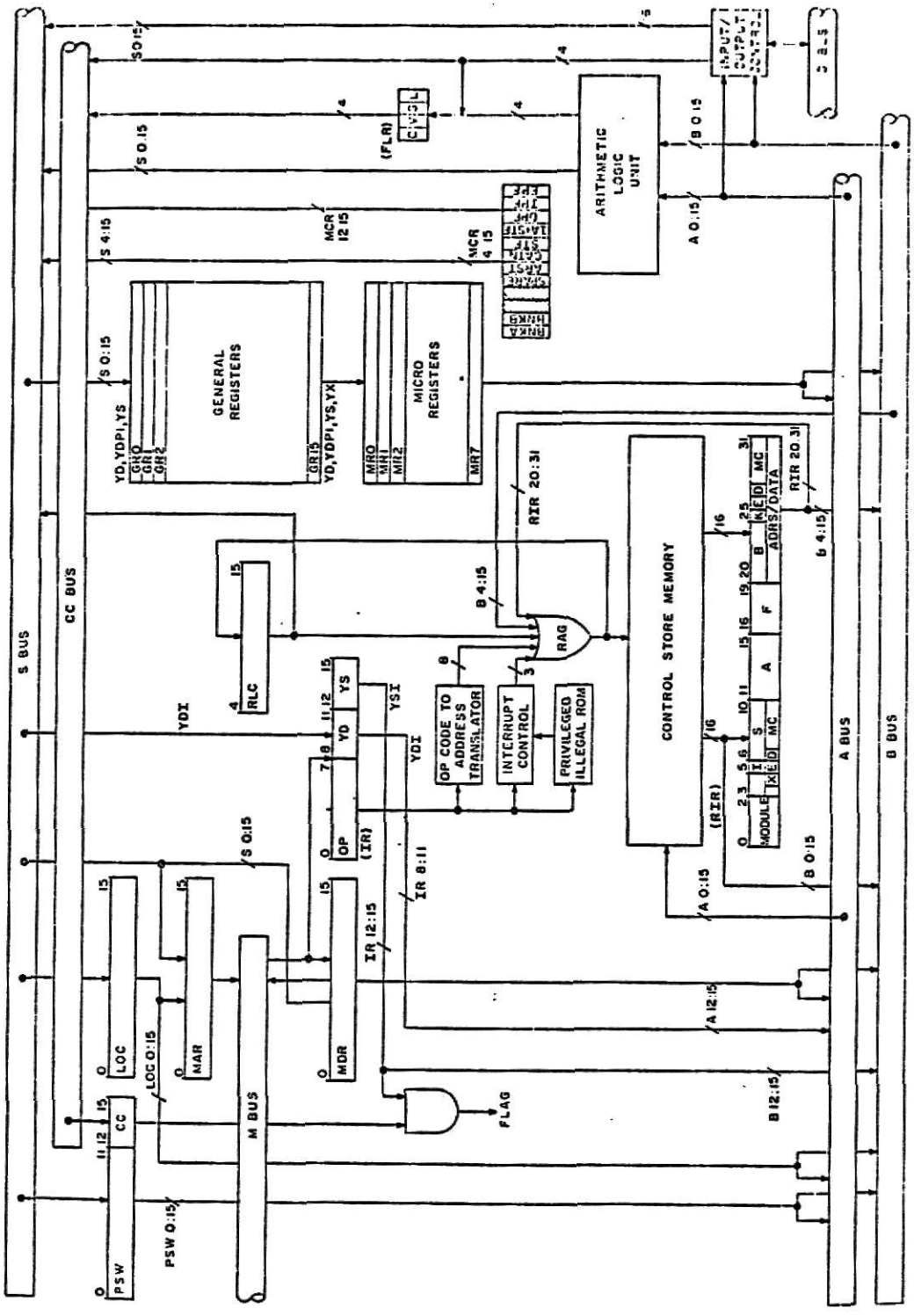


Figure 3. Model 80 Hardware Block Diagram

BIBLIOGRAPHY

1. P.J. Malcolm: A machine-independent design for an APL Translator. The Australian Computer Journal, Vol. 5, #1, Feb. 1973.

2. K.E. Iverson: A Programming Language, John Wiley & Sons, Inc., New York, 1962.

3. D.E. Knuth: The Art of Computer Programming, Addison-Wesley, Reading, Mass., 1973.

4. Model 80 Micro-instruction Reference Manual Publication No. 29-28R01, Interdata Inc., Oceanport, N.J.

5. Model 80 Microcode Assembler. Publication No. 03-041R01A13. Interdata Inc., Oceanport, N.J.

6. Model 80 DCS Control Program. Publication No. 03-051A15. Interdata Inc., Oceanport, N.J.

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

by

SHAH FAROOQ ALAM

B.S., Aligarh University, 1968

AN ABSTRACT OF 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

ABSTRACT

This report deals with a documentation/development model for adding new instructions to the instruction set of a microprogrammed mini-computer--the INTERDATA/85 owned by Kansas State University. Specifications are given for a number of vector/matrix operations. One of the instructions, ADDSUB, is then developed in detail. High-Level algorithms and micro-programs are given for this instruction.