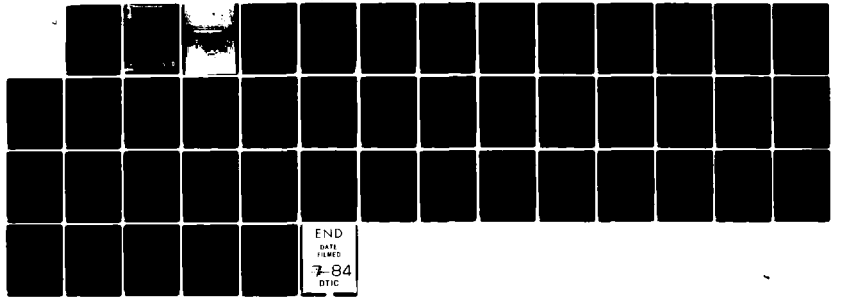
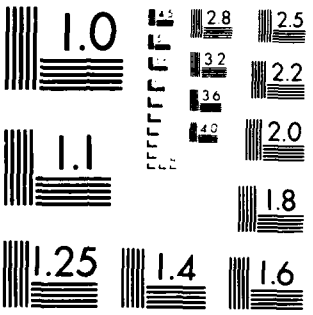
AD-A141 993 A FLOATING POINT PROCESSOR FOR INTEL 8080A MICROPROCESSOR SYSTEMS(U) RUTHERFORD AND APPLETON LABS CHILTON (ENGLAND) R BAIRSTOW ET AL. MAR 82 RL-82-020 F/G 9/2 NL

1/1

UNCLASSIFIED



END
DATE FILMED
7-84
DTIC



Rutherford Appleton Laboratory
CHILTON, DIDCOT, OXON, OX11 0QX

RL-82-020

A Floating Point Processor for INTEL 8080A Microprocessor Systems

R Bairstow, J Barlow, M Jires and M Waters.

March 1982

A FLOATING POINT PROCESSOR FOR INTEL 8080A MICRO PROCESSOR SYSTEMS

R. Bairstow, J. Barlow, M. Jires*, M. Waters

Summary

An A.M.D. 9511 Floating Point Processor[1] has been interfaced to the Rutherford Laboratory Bubble Chamber Group's micro computers[2]. These computers are based on the INTEL 8080A micro processor. The interface uses a memory mapped I/O technique to ensure rapid transfer of arguments between processors. The A.M.D. 9511 acts as a slave processor to the INTEL 8080A system. The 8080 processor is held in 'WAIT' status until completion of the A.M.D. operation.

A software Macro Processor has been written to effectively extend the basic INTEL 8080A instruction set to include the full range of A.M.D. 9511 instructions.

---

* Visitor from the Faculty of Mathematics and Physics, Charles University, PRAGUE.

CONTENTS

1. INTRODUCTION

At the present time, the largest sector of the micro computer market is that covered by the 8 bit micro-processors of which INTEL 8080A/8085/8048, MOTOROLA 6800 and FAIRCHILD F8 are well known examples. These micro-processors have found wide acceptance for many control and data handling tasks and indeed provide highly satisfactory solutions to many problems. They are particularly suited to many communication problems that are 8 bit orientated. However, their 8 bit architecture is somewhat of a limiting factor in applications involving numerical computation. Such computations requiring only single (16 bit) or perhaps even double (32 bit) precision integer or fixed point arithmetic can be handled without too much difficulty by software methods. These methods are slow to execute. More complex calculations, particularly those usually encountered in a scientific environment require the use of floating point arithmetic with 32 bit wide words. This can also be implemented via software but is very slow and inefficient. It is far preferable to provide such facilities via special high speed hardware.

This hardware now exists in the form of a single L.S.I. component; the AM 9511 produced by ADVANCED MICRO DEVICES[1]. This device provides the basic arithmetic operations of Add, Subtract, Multiply and Divide in Fixed Point single and double precision (16/32 bit) and Floating Point single precision (32 bit) formats. It also provides a set of derived functions based on 32 bit floating point format, namely:-

(i) Trigonometric and inverse trigonomentric functions.
(ii) Square roots.
(iii) Logarithms (common, naturel).
(iv) Exponentiation (e^x, y^x).
(v) Conversions from fixed point to floating point and vice versa.
(vi) Data Manipulation Instructions.

It has a general purpose bi-directional 8 bit data bus allowing simple interfacing to 8 bit micro-processor systems.

The overall system architecture of a micro-computer employing this device can be considered as that of a distributed system using two asynchronous CPU's operating in parallel. One CPU handles the basic 8 bit orientated problems such as computer/memory organisation, communication (I/O) etc., whilst the other is dedicated to handling the arithmetic operations.

This architecture makes very efficient usage of the basic component resources. In many micro-computer applications, 32 bit floating point operations are required relatively infrequently. It is therefore uneconomic to have to provide full 16/32 bit architecture with its associated wide address and data paths; far better to use an economical 8 bit architecture with a special purpose processor to handle the arithmetic requirements.

The speed with which the arithmetic operations are executed in a dual processor system is limited by:-

(i) The basic speed of the arithmetic processor.

(ii) The number and speed with which the basic arguments can be passed to the arithmetic processor.

The AM 9511 is currently available with a choice of two basic clock speeds, 2 and 4 M.Hz. The execution time for a floating point multiply operation with the 4 M.Hz. device is quoted as $\sim$ 40 μsecs. The speed with which arguments can be passed to the device depends upon the basic speed of the host micro-processor, its memory cycle time and the level of sophistication of its interface to the arithmetic processor. With the AM 9511, the arguments must be transferred in a byte-serial manner. As an example, a 32 bit floating point operation with two input and one output arguments requires the transfer of at least 9 bytes to the device and 4 bytes from the device. With an INTEL 8080A based host using a 2 M.Hz. clock and conventional programmed I/O, these transfer times are of the order 170 μsecs. The transfer time is small compared to the time taken by the AM 9511 to evaluate a trigonometric function, ($\sim$ 2 msec) but long compared to the time taken to perform a single precision floating point add operation (28 - 175 μsec).

The full instruction set of the AM 9511 comprises 41 instructions, 32 of which operate on full 32 bit word length variables. This instruction set together with execution times in terms of basic clock cycles can be found in Reference 1.

2. INTERFACING THE AM 9511 TO AN INTEL 8080A SYSTEM

2.1 The Hardware Interface

The simplest way to interface the AM 9511 device to an 8080 system is to treat the device as a peripheral I/O component. Conventional programmed I/O can then be used to output arguments, a byte at a time, to the AM 9511. These arguments are input by the AM 9511 and stored on its internal stack. This stack can be either 16 bits wide and 8 levels deep or 32 bits wide and 4 levels deep. Its organisation is illustrated in Figure 1. The entry of each new byte to the stack causes the stack to shift, the previously entered byte being pushed down and the new byte remaining on top of the stack. Single precision integer (16 bit) arguments require two entries to the stack. Double precision integer or real (32 bit) arguments require four entries to the stack.

A command instruction is output to the AM 9511 to initiate the required operation. The AM 9511 has a 'PAUSE' output line which it uses as a busy indicator. This can be used to hold the 8080 CPU in a 'WAIT' state until the AM 9511 is ready with the result of its operation. This result is available on the top of the stack and can be read by the 8080 system by a series of normal I/O read instructions. Special commands enable the stack to be manipulated independently of any arithmetic operation. Memory mapped I/O techniques can be used rather than conventional I/O. With memory mapping, the AM 9511 interface can be considered as analogous to a slow memory.

This simple interface is easy to implement at the hardware level but has the disadvantage that the host CPU is inactive for the duration of the required operation. This could be for a time of the order of milli-seconds for some operations. For some time-critical environments, this dead time may be unacceptable. In such cases it is better to ignore the 'PAUSE' line and to use an alternative signal (END) issued by the AM 9511 device on completion of its operation as an interrupt to the host CPU. The interfacing is slightly more complicated, requiring the use of an interrupt controller, but has the advantage that the host CPU can service other tasks whilst the AM 9511 device is busy. There is a slight penalty in effective execution time with this method due to the overheads of context switching on receipt of the interrupt.

Another interface method for higher performance systems is to use a Direct Memory Access controller. A simple DMA system provides only a very slight decrease in transfer time for the arguments and unless care is taken can actually worsen the transfer time. This is due to the overheads in DMA systems of the need to pass the address and number of bytes for transfer to the DMA controller. Where only a small number of bytes have to be transferred, this overhead can be significant. The method can be useful in situations where many intermediate results have to be stored and a very high speed buffer or cache memory is used.

Figure 2 shows a block diagram of a simple programmed or memory mapped I/O interface.

2.2 The Software Interface

The function of the software interface is to isolate the programmer from the hardware. It ought not to be necessary for the programmer to have to know details of either AM 9511 device or hardware interface. The software interface can be provided by an extension of the host micro-computer instruction set in the form of additional macro instructions. Assemblers running on local micro-computer development systems can be modified to provide the necessary macro instruction translation. However, a more powerful and flexible approach is to use a Macro Processor written in a high level language and running as a cross product on a large host computer. This Macro Processor can run as a pre-step to a conventional cross assembler or can produce an output file of standard micro-computer instructions for use with local assemblers. The Macro Processor should be modifiable to handle different host micro-computers and different hardware interfacing techniques.

3. THE M9511 MACRO LANGUAGE

3.1 Basic Definitions and Comments

The syntax of all data sets and hardware files is described according to BNF (Backus Normal Form) production rules[3]. The hardware representations of all data sets (<M 9511 data set>) are called files.

Survey of all data sets (Software):

<M 9511 data set> ::= <mainstream data set>/<print data set>

<mainstream data set> ::= <input data set>/<output data set>

<input data set> ::= <input data set><input logical record>/
<input logical record>

<output data set>::= <output data set><output logical record>/
<output logical record>

Survey of all corresponding files (Hardware):

<input file> ::= <file>

<output file> ::= <file>

<print file> ::= <file>

<file> ::= <row of closed physical records> <EOF>

<EOF> means "end of file" or equivalent

<row of closed physical records> ::= <row of closed physical records>
<closed physical record>
<closed physical record>

<closed physical record> ::= <physical record> <IRG>

<IRG> means "inter-record gap" or equivalent (new line, new card)

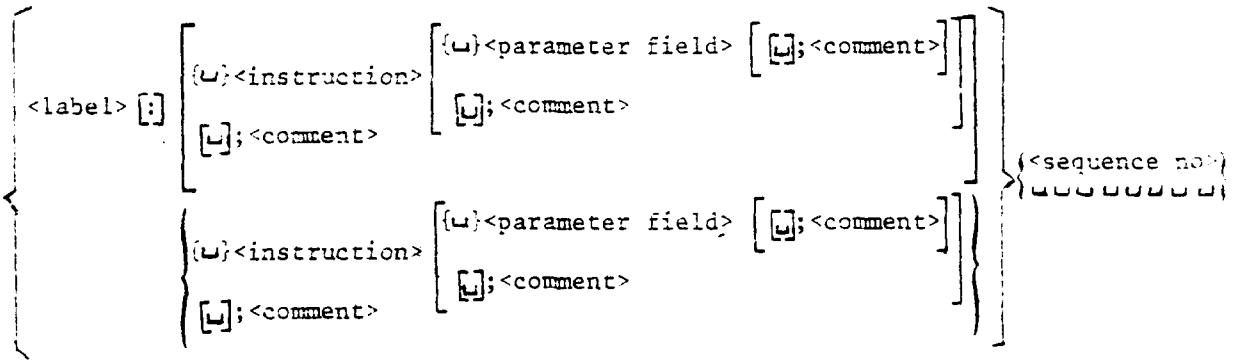
Each <physical record> is a hardware representation of exactly one <input logical record> or <output logical record>

The input file consists of card images. The input text contains 8080 instructions, macroinstructions and comments. All input cards are printed on the printer (print file) together with error or warning messages and description of the content and status of the M 9511 stack. The output file can be realised on any medium that can hold sequential data sets and consists of 8080 assembler instructions in card image format (fig. 3).

3.2 The Mainstream data set

The 8080 assembler instructions differ from the instructions of the AM 9511 (written in macro language M 9511) by the name of the instruction (mnemonics) and by the definition of the parameter subfields.

The general pattern of the statement of both kinds of instructions is as follows:-



where $\begin{bmatrix} a \\ b \end{bmatrix}$ means a or b or nothing

$\begin{Bmatrix} a \\ b \end{Bmatrix}$ means a or b

[␣] means nothing or one (or more) blank

{␣} means one (or more) blank

<a> means "a"

3.3 The Input data set

3.3.1 Description of the structure

The input data set consists of 8080 assembler language instructions, 8080 assembler language macro instructions (if any), M 9511 macro instructions and comments (if any).

The M 9511 Macro instructions do not differ syntactically from normal 8080A instructions. The only difference lies in the name of the instruction operation and in the meaning of the parameters. Names that are defined for the M 9511 Macro instruction set must be reserved exclusively for this set and cannot be duplicated for normal 8080A macros.

M 9511 Macro Instructions can be optionally labelled. Comments can optionally follow the parameter fields or instruction name in the absence of parameter fields.

3.3.2. M 9511 Syntax definition:

The Syntax of input logical records as described by BNF production rules is as follows:-

<input logical record>::= <logical record 8080>|<logical record M>
<logical record 8080>::= <statement 8080><sequence number field>
<logical record M>::= <statement M><sequence number field>
<sequence number field>[8]::= <blankety><sequence number>
<seq. of blanks>
<seq. of blanks>::= <seq. of blanks><blank>|<blank>
<sequence number>::= <sequence number><digit>|<digit>
<digit>::= 0|1|2|3|4|5|6|7|8|9
<blank>::= ␣
<blankety>::= <seq. of blanks>|<empty>
<empty>::=
<statement 8080> See appendix I

The definition of M 9511 macrolanguage now follows:-

<statement M>[72]::= <instruction field>|<label>:<instruction field>
<instruction field>::= <seq. of blanks><instruction><general parameter field>
<label>::= <label><letter>|<label><digit>|<letter>

Only the first five characters in the label are significant.

<letter>::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|?
<instruction>[4]::= <instruction><letter>|<letter>
<general parameter field>:: <seq. of blanks><parameter field><comment field>|<comment field>
<comment field>::= <seq. of blanks>|<blankety>;<comment>
[3][1]<parameter field>::= <parameter>|<parameter subfield><parameter field>
<parameter subfield>::= <parameter>|<empty>
<parameter>::= <shift>|<register pair>|<variable>

<variable>::= <label>

<register pair>::= B/D/H

<shift>::= S

<comment>::= <any char> | <comment><any char>

<any char>::= <digit>|<letter> <spec char> <blank>

<spec char>::= + - | * / ( ) ; ' : , $ ' = ( )

A restriction is that the single letters S, B, D, H cannot be used as labels since these are defined as register pairs and shifts.

N.B.: Comment on BNF production rules:

The length of the character strings produced by production rules or the number of recursive repetitions is described by these rules:

$<\ >_N^M$ means the minimal N and maximal M length of the string

$<\ >^L$ means that the length L of the string is mandatory

${}_N^M<\ >$ means the minimal number N and maximal number M of recursive repetitions of the production rule

3.3.2. M 9511 Semantics:

All M 9511 macro instructions are expanded into the sequences of 8080 assembler instructions in order to provide the required operations.

The meaning of some syntactical symbols is as follows:-

<label> means label

<instruction> corresponds to a specific instruction in the AM 9511 instruction set

<parameter> represents a Direct Address or Indirect Address and/or an AM 9511 Stack Operation

a) Instructions

A full description of M 9511 Macro Instructions is provided in Appendix II. To a large extent the same mnemonics as are used in Reference 1 are repeated in the Macro Instruction set. The following list of instructions however differ:-

M 9511	AM 9511 (Original Set)
NOPM	NOP
NOPS	
NOPD	
NOPF	
DSTM	

The NOPM instruction is so named as to distinguish it from the INTEL 8080 instruction NOP. The remaining NOP ($\frac{S}{D}$) instructions are used to read/write parameters into/out of the AM 9511 stack without performing any operation. The DSTM instruction is used to read the AM 9511 device status information (see Ref. 1). This instruction affects the 8080 condition bits accordingly.

b) Parameters

Apart from the S parameter that will be described later, the presence of a parameter in the parameter list implies the reading/writing of

information to the M 9511 stack. The number of bytes read/written to the stack depends upon the instruction (see Appendix II). The first two parameters refer to the sending of information to the stack and the third parameter refers to reading information from the stack.

The parameters can be specified in four different ways:-

1. <empty> means that the parameter is not used and any operation relevant to this parameter is not performed.

2. <variable> means the 'direct address' of the value that is to be sent/read from the top of the stack.

3. <register pair> means the register pair containing the address of the value that is to be sent/read from the top of the stack.

4. <shift> The meaning depends upon the position in the parameter list.

   a) For the first two positions, the stack is shifted down, the TOS (top of stack) being copied to the NOS (next to top of stack). The bottom of the stack is lost.

   b) For the third position, the stack is shifted up in circular manner (i.e. rotated).

   For both cases a) and b) the number of shifts is determined by the instruction (See Appendix II).

3.4 The Output File

3.4.1 Description of Structure

The structure of the <output data set> is the same as that for the <input data set> except that each <logical record M> from the <input data set> is expanded by the M 9511 Macro processor into <logical record M expanded> of the <output data set>. This <logical record M expanded> consists of Intel 8080 assembler instructions only.

The original <logical record M> is added as a comment to the <output data set> before the expansion is performed. A comment containing blanks is added to the end of the <output data set> after the expansion

The expansion is described in Appendix III.

3.4.2 Syntax of M 9511 Expansion

<output logical record>::= <logical record 8080> <logical record M expanded>

<logical record M expanded>::= <comment head logical record>
<row of logical records 8080>
<comment tail logical record>

$^{26}_{2}$<row of logical records 8080> ::= <row of logical records 8080>
<logical record 8080>
<logical record 8080>

<comment head logical record>::= <comment head><sequence number field>

<comment tail logical record>::= <comment tail><sequence number field>

<comment head>^{72} ::= ;00.<statement M*>

<statement M*>is the truncated <statement M> from the right to the 60th character.

<comment tail>^{72} ::= ; <seq. of blanks>

3.5 The Print File

3.5.1 Description of Structure

The <print data set> consists of a listing of each instruction from the <input data set>. For M 9511 Macro instructions, the contents of the AM 9511 stack are simulated and printed for each step in the instructions execution. Warning and error messages are added as necessary. Each instruction in the listing is numbered.

The print file is usually printed on the system printer. The records in the file are grouped into numbered pages.

a) The Syntax

<print dat >::= <file heading><print data set M><file ending>

<print da M>::= <print data set M><printed page> <printed page>

<printed ew page action><page heading><printed page M>

$^{65}_{1}$<printed pa M>::= <printed page M><printed line> <printed line>

<printed line>::= <new line action><print record>

<print record>^{130} ::= <print record><any char> <any char>

b) The Semantic

The <file heading> consists of a title and date

The <file ending> is a summary of totals for numbers of input, output, print records and various error messages and warnings.

The <page heading> is a page number

The <new page action> and <new line action> are appropriate actions taken by hardware to display new page or new line.

The <print record> consists of five types of messages:

1. All input logical records are listed step by step, always preceeded by the print record number
2. Error messages see 3.5.2
3. Error warnings see 3.5.2
4. After each M 9511 Macro Instruction, the status of the AM 9511 internal stack is described (four lines) together with warnings concerning any meaningless information on the stack. (This check is only formal) see 3.5.3.
5. Blank lines

3.5.2 Error and Warnings

There are 15 possible error/warning messages. Error messages are issued wherever an error occurs during card input and syntax checking (label and parameter fields).

Warning messages are concerned mainly with possible inconsistencies in the state of the AM 9511 internal stack.

Checks and appropriate error messages are also issued concerning error conditions arising through possible but hopefully very rare errors in the basic programme logic.

The number of error/warning conditions arising during the macro-processors operation is counted and a summary printed at the end of the run. Each error condition has attributed to it an appropriate severity code. The highest severity code arising during the run is issued as an IBM 360 completion code for the 360 job step.

3.5.3 AM 9511 Stack Simulation

It is not obligatory that the results of macro instruction execution be returned to the external memory. Intermediate results can be left in the AM 9511 internal stack ready for use with a subsequent instruction. This technique can be used for program optimisation in situations where speed of execution is critical. However, with this

technique it is essential that the exact content of the stack be known at all stages of the execution of consecutive instructions. Since certain instructions operate on single precision word lengths (16 bits) whilst others operate on double precision word lengths (32 bits) and since some instructions cause movements of the stack, it can become very difficult to be certain as to the content of the stack at any given stage.

Therefore, for ease of verification and as an aid to optimisation the stack behaviour is simulated by the macro processor. The results of the simulation are displayed as four lines in the print file and appear after the macro instruction itself.

The first line shows the contents of the stack after the input of first parameter, the second line the contents after input of the second parameter, the third line the contents after execution of the macro instruction and the fourth line the contents after output of the third parameter.

The format of each line starts with the work STACK and is followed by four fields separated by 'dot' characters. Each field represents one double word (32 bits) of information on the stack with the 'Top of Stack' to the left. The name appearing in each field is either a parameter name or the name of an instruction, implying the result following the execution of the instruction. For double precision parameters, the field is left extended with '*' characters. For single precision parameters, two such parameters occupy one field. A totally blank field or half field in the case of single precision characters implies an undefined stack content.

Warning messages concerning possible inconsistencies are issued as necessary between lines.

4. THE CROSS-MACROPROCESSOR (TRANSLATOR)

4.1 Basic Structure

The translator is composed of three main parts: syntactical analyser, synthesiser of the new structure and code generator. The syntactical analyser evaluates each statement step by step starting with the label followed by the instruction, operands and comment. Internal codes are generated and stored for each field indicating its presence or absence and its type and content.

The synthesiser uses the above codes to establish the basic structure of calls to the code generators. These in turn create groups of Intel 8080 instructions to access parameters and initiate operations.

Any future change in method of interfacing the AM 9511 to the 8080 system can be easily accommodated by changes to the code generators and possibly synthesiser.

4.2 Coding Method

A top down structured programming approach was rigourously adhered to at all stages in the coding of the macro processor. This approach enabled both the coding and de-bugging stages to be completed quickly with a minimum of errors being encountered during the work and afterwards when in production. The final programme is also easy to understand and can be modified if necessary with relative ease.

The processor was coded in Fortran which although not being ideal for structured programming methods is nevertheless widely used and understood by the user community for whose use the programme was originally intended. Special sequences of comment statements and 'GO TO' instructions were created to simulate the required structures i.e. SEQUENCE, IF-THEN, IF-THEN-ELSE, WHILE-DO, REPEAT-UNTIL, LOOP-EXITIF-ENDLOOP, SELECT-CASE.

Each programme module was divided into three parts, declaration, main text and formats. The programme was extensively commented which considerably assists its legibility.

It is intended that the macro processor be used as the first step of a two step job, the second step being a conventional assembler. Step Number 1 (Pre-Processor) produces a file containing conventional 8080 assembler instructions which are then used as input to Step Number 2 (assembler).

5. Conclusion

The system as described in the previous sections is operational and in use by the Rutherford Appleton Laboratory Bubble Chamber Group in connection with its micro-computer based film digitising table system. The cross macro-assembler runs on the Group's VAX 11/780 computer, the final object code being loaded into a selected micro-computer via a local communication network.

Some examples illustrating the usage of the macro-assembler and the macro expansions produced are given in Appendix IV.

APPENDIX I

The partial description of syntax of 8080 assembler statements as described by BNF production rules is as follows:-

<statement 8080>[72] ::= <instrcom field>|<label>:<instrcom field>|<label> <instruction 8080 field>

<instrcom field>::= <blankety>;<comment>|<instruction 8080 field>

<instruction 8080 field>::= <seq. of blanks><instruction 8080><general 8080 parameter field>

<instruction 8080>[1]::= <instruction 8080><letter>|<letter>

<general 8080 parameter field>::= <seq. of blanks><8080 parameter field>| <blankety>; <comment>|<seq. of blanks>

<8080 parameter field>::= <8080 parameter subfield>|<8080 parameter subfield>, <8080 parameter field>

<8080 parameter subfield> for definition see [4].

APPENDIX II

M9511 Instruction Set

Instruction Number (MNM)	NAME	CODE 1234	DESCRIPTION		
1	SADD	S2SY	NOS + TOS,	POP up stack, result to TOS	All operations are made with version I stack organisation (two bytes times eight) see Figure 1.
2	SSUB		NOS - TOS,		
3	SMUL		NOS * TOS,		
4	SDIV		NOS /TOS ,		For details see also (1).
5	DADD	D2DY	NOS + TOS,	POP up stack, result to TOS	All operations are made with version II stack organisation (four bytes times four) see Figure 1.
6	DSUB		NOS - TOS,		
7	DMUL		NOS * TOS,		
8	DDIV		NOS / TOS,		For details see also (1).
9	FADD	F2FY	NOS + TOS,	POP up stack, result to TOS	All operations are made with version II stack organisation (four bytes times four) see Figure 1.
10	FSUB		NOS - TOS,		
11	FMUL		NOS * TOS,		
12	FDIV		NOS / TOS,		For details see also (1).
13	SQRT	F1FY	$\sqrt{TOS}$,	Result to TOS	Version II stack organisation.
14	SIN		sin(TOS),		
15	COS		cos(TOS),		
16	TAN		tg(TOS),		
17	ASIN		arc sin(TOS),		
18	ACOS		arc cos(TOS),		
19	ATAN		arc tg(TOS),		
20	LOG		log(TOS),		
21	LN		ln(TOS),		
22	EXP		exp(TOS),		
23	PWR	F2FY	NOS raised to power TOS - for details see (1).		

APPENDIX II (Contd.)

Instruction Number (MNM)	NAME	CODE	DESCRIPTION
24	NOPM	OOON	no-operation, no transfer of parameters allowed
25	FIXS	F1SY	conversion functions
26	FIXD	F1DY	
27	FLTS	S1FY	
28	FLTD	D1FY	
29	CHSS	S1SY	change sign operations
30	CHSD	D1DY	
31	CHSF	F1FY	
32	PTOS	OOSY	push down and pop up operations
33	PTOD	OODY	
34	PTOF	OOFY	
35	POPS	OOSY	
36	POPD	OODY	
37	POPF	OOFY	
38	XCHS	OOSY	exchange operations
39	XCHD	OODY	
40	XCHF	OOFY	
41	PUPI	OOFY	π to TOS operation: PUSH down stack, π to TOS (version II stack operation)
42	NOPS	S2SY	no-operations with allowed transfer of parameters
43	NOPD	D2DY	
44	NOPF	F2FY	
45	DSTM	OOSY	reading of AM 9511 Device Status, see Appendix III section D

APPENDIX III

CODE GENERATORS

These are the sets of 8080 instructions that are used for developing the M 9511 Macro Instructions.

For all cases:

```
STACK   EQU   OFFFOH   (Hardware Dependent)
CONTR   EQU   OFFFCH   (Hardware Dependent).
```

Code Section A (Sending information to the top of stack)

A1) Parameter is a Label (i.e. Direct Address)

```
LDA   3 + OP
STA   STACK
LDA   2 + OP
STA   STACK
LDA   1 + OP   } For single precision only
STA   STACK    }
LDA   0 + OP   } For single precision only
STA   STACK    }
```

} Double Precision and Floating

OP is replaced by actual Label Name.

Sequence takes 104 Clock Cycles (Double P)
52 Clock Cycles (Singe P)

A2) Parameter is a Register Pair (BC, DE)

```
INX   B or D
INX   B or D
INX   B or D   } For single precision only
LDAX  B or D   } For single precision only
STA   STACK    } For single precision only
DCX   B or D   } For single precision only
LDAX  B or D   } For single precision only
STA   STACK    } For single precision only
DCX   B or D
LDAX  B or D
STA   STACK
DCX   B or D
LDAX  B or D
STA   STACK
```

} Double Precision and Floating

Sequence takes 110 Clock Cycles (Double P)
50 Clock Cycles (Single P)

Explanation of Code:-

Column number of code:

1 :- Type of Input Parameter

2 :- Maximum Number of Parameters allowed

3 :- Type of Output Parameter

4 :- Third Parameter Allowed

Type of code:

S :- Single Precision Integer (16 bit)

D :- Double Precision Integer (32 bit)

F :- Floating Point (32 bits)

Y :- Yes

N :- No

A3) Parameter is a Register Pair (H, L)

```
INX     H
INX     H
INX     H
MOV     A, M
STA     STACK      } For single precision only
DCX     H
MOV     A, M
STA     STACK
DCX     H
MOV     A, M
STA     STACK
DCX     H
MOV     A, M
STA     STACK
```

Double Precision and Floating

Sequence takes 110 Clock Cycles (Double P)
50 Clock Cycles (Single P)

A4) Parameter is S

```
Single Precision:-  MVI   A , 77H
                    STA   CONTR
Double Precision:-  MVI   A , 37H
                    STA   CONTR
Floating:-          MVI   A , 17H
                    STA   CONTR
```

In all cases Sequence takes 20 Clock Cycles

Code Section B (Retrieving information from TOS)

B1) Parameter is a Label (i.e. Direct Address)

```
LDA     STACK
STA     0 + OP     } For single precision only
LDA     STACK
STA     1 + OP
LDA     STACK
STA     2 + OP
LDA     STACK
STA     3 + OP
```

Double Precision and Floating

Sequence takes 104 Clock Cycles for Double P
" " 52 " " " Single P

B2) Parameter is a Register Pair (BC, DE)

```
LDA     STACK
STAX    B or D
INX     B or D
LDA     STACK
STAX    B or D
INX     B or D
LDA     STACK
STAX    B or D
INX     B or D
LDA     STACK
STAX    B or D
DCX     B or D
DCX     B or D
DCX     B or D
```

For single precision only

Double Precision and Floating

Sequence takes 110 Clock Cycles for Double P
" " 90 " " for Single P

B3) Parameter is a Register Pair (HL)

```
LDA     STACK
MOV     M, A
INX     H
LDA     STACK
MOV     M, A
INX     H
LDA     STACK
MOV     M, A
INX     H
LDA     STACK
MOV     M, A
DCX     H
DCX     H
DCX     H
```

For single precision only

Double Precision and Floating

Sequence takes 110 Clock Cycles for Double P
" " 90 " " " Single P

B-) Parameter is S

Single Precision:-	MVI	A, 78H
	STA	CONTR
Double Precision:-	MVI	A, 38H
	STA	CONTR
Floating:-	MVI	A, 18H
	STA	CONTR

In all cases sequence takes 20 Clock Cycles

Code Section C (Sending Instruction into AM 9511)

```
MVI     A. INSTR
STA     CONTR

INSTR   EQU     Binary Code of AM 9511 instruction as defined in
                Command Summary (Ref. 1)
```

Sequence takes 20 Clock Cycles to send Instruction to AM 9511

N.B. Number of Clock Cycles required for AM 9511 to execute given instruction can be found in Ref. 1.

Code Section D (Reading AM 9511 Device Status)

```
LDA             CONTR
RAL
JC              $-4
XRI             40H
ANI             0C0H
LDA             CONTR
RAL
JNC             $+4
CMC
RAL
JNC             $+4
CMC
RAL
JNC             $+4
CMC
RAR
RAR
RAR
RAR
```

Sequence takes at least 114 Clock Cycles. More cycles may be required if device is busy when request is made.

This sequence corresponds to the Macro Instruction RSTM. Error Code Field from Status Register is transferred to Bits (0-4) of the 8080A Accumulator. Sign, Zero and Carry replace the corresponding 8080A Flags.

8080A Parity Flag is not affected.

The above code sections are assembled together according to the M 9511 macro instruction in the following order:-

1. Code relevant to first parameter (if any) is selected from Group A.
2. Code relevant to second parameter (if any) follows and is also selected from Group A.
3. Code for instruction execution follows and is Group C code.
4. Code relevant to third parameter (if any) follows and is selected from Group B.

APPENDIX IV

EXAMPLES

A. Parameter Usage

1. Binary Functions

a)
```
        FADD    AM, ALI, VT
AM:     DS      4; floating point number (2.0)
ALI:    DS      4; floating point number (3.0)
VT:     DS      4; floating point result (5.0)
```

Whenever an external address is specified for the third parameter the result is copied from the TOS to the external address and the stack is popped up circular (the result of operation is now on the bottom of the stack.

b)
```
LHLD    AM
MOV     B, H        ; Address of AM in BC
MOV     C, L
LHLD    ALI
XCHG                ; Address of ALI in DE
FADD    B, D, B     ; Result back in AM
```

c)
```
LHLD    AM
MOV     B, H        ; Address of AM in BC
MOV     C, L
FADD    ALI, B      ; Result stays in the TOS
```

d)
```
FADD    AM,, VT
FADD    , AM, VT
```

These instructions add floating point number expected in TOS to floating point number stored at AM. The results is sent to location address VT. Both these instructions produce the same result.

e)
```
FADD    ,,VT
```

This instruction adds floating point number expected in TOS to floating point number expected in NOS. The result is sent to location address VT.

f) FADD

This instruction adds floating point number expected in TOS to floating point number expected in NOS. The result stays in the TOS.

g) FADD AM, S, VT

This instruction first sends the floating point number expected as a content of AM to TOS. The second parameter(S) causes the stack to be pushed down whilst leaving the original TOS intact. The instruction then adds TOS and NOS the result being placed first in TOS and then sent to VT.

The above result could be obtained in a less efficient manner by using the following instruction: FADD AM, AM, VT.

h) FADD S,, VT

The expected floating point number in TOS is doubled.

i) FADD S, S
FADD ,, VT

The expected floating point number in TOS is tripled and sent to VT.

This example emphasises the importance of understanding the stack behaviour.

2. Unary Functions

SIN AM,, VT

sinus of AM is sent to VT

but: SIN AM, ALI, VT

is flagged as an error 009 and parameter ALI is not used and sinus of AM is sent to VT

SIN ,, VT

sinus of TOS is sent to VT

SIN

sinus of TOS replaces TOS

3. Functions that do not require any input parameters

PUPI ,, VT

π is sent to VT

PUPI

π is in TOS

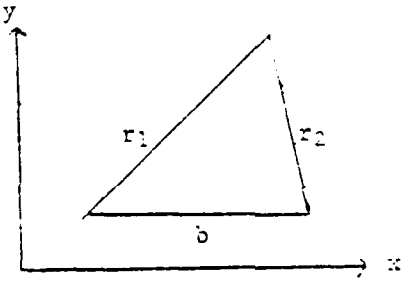
PUPI ,, S

π is in bottom of the stack

B Part Program Example

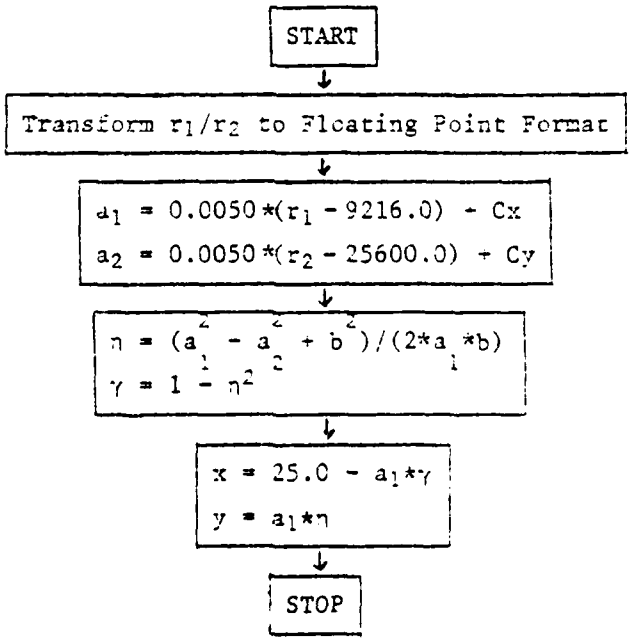
Given a triangular coordinate system with fixed base length b.

To convert to Cartesian coordinates x, y



Triangular Coordinates are r_1, r_2 (Integer 4 bytes each)

Basic Flow Chart



Program Section:—

```
CR1:    DS   4   ;   r1 Coordinate (Triangular)
CR2:    DS   4   ;   r2    "            "
CK:     DS   4   ;   Cx          Fl. Pt Format
CY:     DS   4   ;   Cy          (  "   "   "  )
CK1:    DS   4   ;   9216.0      (  "   "   "  )
CK2:    DS   4   ;   25600.0     (  "   "   "  )
CPAD:   DS   4   ;   0.005       (  "   "   "  )
CB:     DS   4   ;   b=137.45    (  "   "   "  )
CETA:   DS   4   ;   η           (  "   "   "  )
CGAM:   DS   4   ;   γ
CDVA:   DS   4   ;   25.0        (  "   "   "  )
CONE:   DS   4   ;   1.0         (  "   "   "  )
CTWO:   DS   4   ;   2.0         (  "   "   "  )
CA1:    DS   4   ;   a1
CA2:    DS   4   ;   a2
X:      DS   4   ;   x Coordinate (Cartesian)
Y:      DS   4   ;   y     "           "
:
```

Inst No;

```
1.  START:  FLTD   CR1          ;   r1
2.          FSUB   CK1          ;   r1 - 9216.0
3.          FMUL   CPAD         ;   0.0050*(r1 - 9216.0)
4.          FADD   CK,, CA1     :   a1
5.          FLTD   CR2          ;   r2
6.          FSUB   CK2          ;   r2 - 25600.0
7.          FMUL   CPAD         ;   0.0050*(r2 - 25600.0)
8.          FADD   CY,, CA2     ;   a2
9.          FMUL   CA1, S       ;   a1^2
10.         FMUL   CA2, S       ;   a2^2
11.         FSUB                ;   a1^2 - a2^2
12.         FMUL   CB, S        ;   b^2
13.         FADD                ;   a1^2 - a2^2 - b^2
14.         FDIV   CTWO         ;   (a1^2 - a2^2 + b^2)/2.0
15.         FDIV   CA1          ;   (a1^2 - a2^2 + b^2)/(2.0 * a1)
16.         FDIV   CB,, CETA    :   η
17.         NOPF   CDVA, CONE   ;
```

Program Section (Contd.)

```
18.                FMUL        CETA, S          ;      η²
19.                FSUB                         ;      1 - η²
20.                FMUL        CA1              ;      a₁*γ
21.                FSUB        ,,X              ;      X
22.                FMUL        CETA, CA1, Y     ;      Y
```

Stack Simulation (for Part Program above)

Inst. No.		TOS	NOS	BOS
1.	STACK:	xxxxx CR1		
	STACK:	xxxxx CR1		
	STACK:	xxxxx FLTD		
	STACK:	xxxxx FLTD		
2.	STACK:	xxxxx CK1	xxxxx FLTD	
	STACK:	xxxxx CK1	xxxxx FLTD	
	STACK:	xxxxx FSUB		
	STACK:	xxxxx FSUB		
3.	STACK:	xxxxx CPAD	xxxxx FSUB	
	STACK:	xxxxx CPAD	xxxxx FSUB	
	STACK:	xxxxx FMUL		
	STACK:	xxxxx FMUL		
4.	STACK:	xxxxx CX	xxxxx FMUL	
	STACK:	xxxxx CX	xxxxx FMUL	
	STACK:	xxxxx FADD		
	STACK:			xxxxx CA1
5.	STACK:	xxxxx CR2		
	STACK:	xxxxx CR2		
	STACK:	xxxxx FLTD		
	STACK:	xxxxx FLTD		
6.	STACK:	xxxxx CK2	xxxxx FLTD	
	STACK:	xxxxx CK2	xxxxx FLTD	
	STACK:	xxxxx FSUB		
	STACK:	xxxxx FSUB		
7.	STACK:	xxxxx CPAD	xxxxx FSUB	
	STACK:	xxxxx CPAD	xxxxx FSUB	
	STACK:	xxxxx FMUL		
	STACK:	xxxxx FMUL		
8.	STACK:	xxxxx CY	xxxxx FMUL	
	STACK:	xxxxx CY	xxxxx FMUL	
	STACK:	xxxxx FADD		
	STACK:			xxxxx CA2
9.	STACK:	xxxxx CA1		
	STACK:	xxxxx CA1	xxxxx CA1	
	STACK:	xxxxx FMUL		
	STACK:	xxxxx FMUL		

Stack Simulation (Contd.)

Inst. No.		TOS	NOS		BCS
10.	STACK:	xxxxx CA2	xxxxx FMUL		
	STACK:	xxxxx CA2	xxxxx CA2	xxxxx FMUL	
	STACK:	xxxxx FMUL	xxxxx FMUL		
	STACK:	xxxxx FMUL	xxxxx FMUL		
11.	STACK:	xxxxx FMUL	xxxxx FMUL		
	STACK:	xxxxx FMUL	xxxxx FMUL		
	STACK:	xxxxx FSUB			
	STACK:	xxxxx FSUB			
12.	STACK:	xxxxx CB	xxxxx FSUB		
	STACK:	xxxxx CB	xxxxx CB	xxxxx FSUB	
	STACK:	xxxxx FMUL	xxxxx FSUB		
	STACK:	xxxxx FMUL	xxxxx FSUB		
13.	STACK:	xxxxx FMUL	xxxxx FSUB		
	STACK:	xxxxx FMUL	xxxxx FSUB		
	STACK:	xxxxx FADD			
	STACK:	xxxxx FADD			
14.	STACK:	xxxxx CTWO	xxxxx FADD		
	STACK:	xxxxx CTWO	xxxxx FADD		
	STACK:	xxxxx FDIV			
	STACK:	xxxxx FDIV			
15.	STACK:	xxxxx CA1	xxxxx FDIV		
	STACK:	xxxxx CA1	xxxxx FDIV		
	STACK:	xxxxx FDIV			
	STACK:	xxxxx FDIV			
16.	STACK:	xxxxx CB	xxxxx FDIV		
	STACK:	xxxxx CB	xxxxx FDIV		
	STACK:	xxxxx FDIV			
	STACK:				xxxxx CETA
17.	STACK:	xxxxx CDVA			
	STACK:	xxxxx CONE	xxxxx CDVA		
	STACK:	xxxxx CONE	xxxxx CDVA		
	STACK:	xxxxx CONE	xxxxx CDVA		
18.	STACK:	xxxxx CETA	xxxxx CONE	xxxxx CDVA	
	STACK:	xxxxx CETA	xxxxx CETA	xxxxx CONE	xxxxx CDVA
	STACK:	xxxxx FMUL	xxxxx CONE	xxxxx CDVA	
	STACK:	xxxxx FMUL	xxxxx CONE	xxxxx CDVA	

Stack Simulation (Contd.)

Inst. No.		TOS	NOS		BCS
19.	STACK:	xxxxxx FMUL	xxxxxx CONE	xxxxxx CDVA	
	STACK:	xxxxxx FMUL	xxxxxx CONE	xxxxxx CDVA	
	STACK:	xxxxxx FSUB	xxxxxx CDVA		
	STACK:	xxxxxx FSUB	xxxxxx CDVA		
20.	STACK:	xxxxxx CA1	xxxxxx FSUB	xxxxxx CDVA	
	STACK:	xxxxxx CA1	xxxxxx FSUB	xxxxxx CDVA	
	STACK:	xxxxxx FMUL	xxxxxx CDVA		
	STACK:	xxxxxx FMUL	xxxxxx CDVA		
21.	STACK:	xxxxxx FMUL	xxxxxx CDVA		
	STACK:	xxxxxx FMUL	xxxxxx CDVA		
	STACK:	xxxxxx FSUB			
	STACK:				xxxxxx X
22.	STACK:	xxxxxx CETA			
	STACK:	xxxxxx CA1	xxxxxx CETA		
	STACK:	xxxxxx FMUL			
	STACK:				xxxxxx Y

Appendix V

Using the M9511 Macro Processor on the Bubble Chamber Groups VAX 11/780 computer.

INTEL 8080 source code is assembled on the Bubble Chamber Groups VAX 11/780 computer using a cross assembler written in ECPL code.

As this cross assembler is a CPU intensive process then access to it is via a BATCH job running in the SYS$ASSEM batch queue.

This appendix describes the action of the batch job and how the M9511 Macro Processor is used, along with the commands needed to execute the job.

(A) BATCH JOB.

The batch job has seven input parameters (P1 to P7) which control the flow of the job. They are as follows:-

P1	Input file name	(If type is .ADD then P1 becomes a concatenation file containing a list of input files)
P2	Output file name	(4-character name of program)
P3	Default directory	(directory where P1 exists)
P4	Username	(Used by job to keep user informed of its progress)
P5	Title	(String of characters)
P6	Cross reference flag	(YES/NO)
P7	M9511 flag	(YES/NO)

The flow of execution through the batch job is as follows.-

1) Check parameters.

2) Run pre-processor
where,

Input file name	P1
Output file name	WORK:P2.TEM

3) If P7 is "YES" then run M9511 Macro Processor
where,

Input file name	WORK:P2.TEM;n (n is vers no.)
Output file name	WORK:P2.TEM;n+1
List file name	WORK:AM9511.LIS

4) Run Cross assembler.
where,

Input file name	WORK:P2.TEM
Output file name	GO.DAT
List file name	WORK.P2.LIS

5) If Program name (P2) is "NULL" then terminate

6) Run Loader.
where,

Input file name	GO.DAT
Output file name	SYS$EXE:[AP04]P2.LOD

(B) BATCH JOB EXECUTION

The batch job is executed by means of one of the following commands:-

1) ASSEM Run cross assembler
2) ASSFP Run cross assembler with M9511 macro processor

Both commands will prompt for the following set of parameters :-

File name
Program name (4 chars)
Title
Cross reference (YES/NO)

The only difference between the two commands is the value of P7 which is passed to the batch job, i.e.:-

ASSEM P7 = NO
ASSFP P7 = YES

The value of P7 is automatically set up by the respective command.

ACKNOWLEDGEMENTS

The authors acknowledge the help provided by P. Mace of the Track Analysis Maintenance Section in the construction of the hardware. Dr. Jiras also thanks Dr. Stafford and the Science Research Council for the hospitality provided by Rutherford Laboratory and Dr. Kalmus for the opportunity to work on this project.

REFERENCES:

(1) AM 9511 Arithmetic Processing Unit
Advanced Micro Devices, 901 Thompson Place, Sunnyvale, California [illegible].

(2) Rutherford Laboratory Report RL-82-019: Bubble Chamber Research Group Microcomputer Unit, R Bairstow, J Barlow, P R Mace, P Seller, M Waters, J G Watson.

(3) J W Backus: The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference, ICIP Paris June 19[illegible].

(4) INTEL 8080/8085 Assembly Language
Programming Manual 98-940.

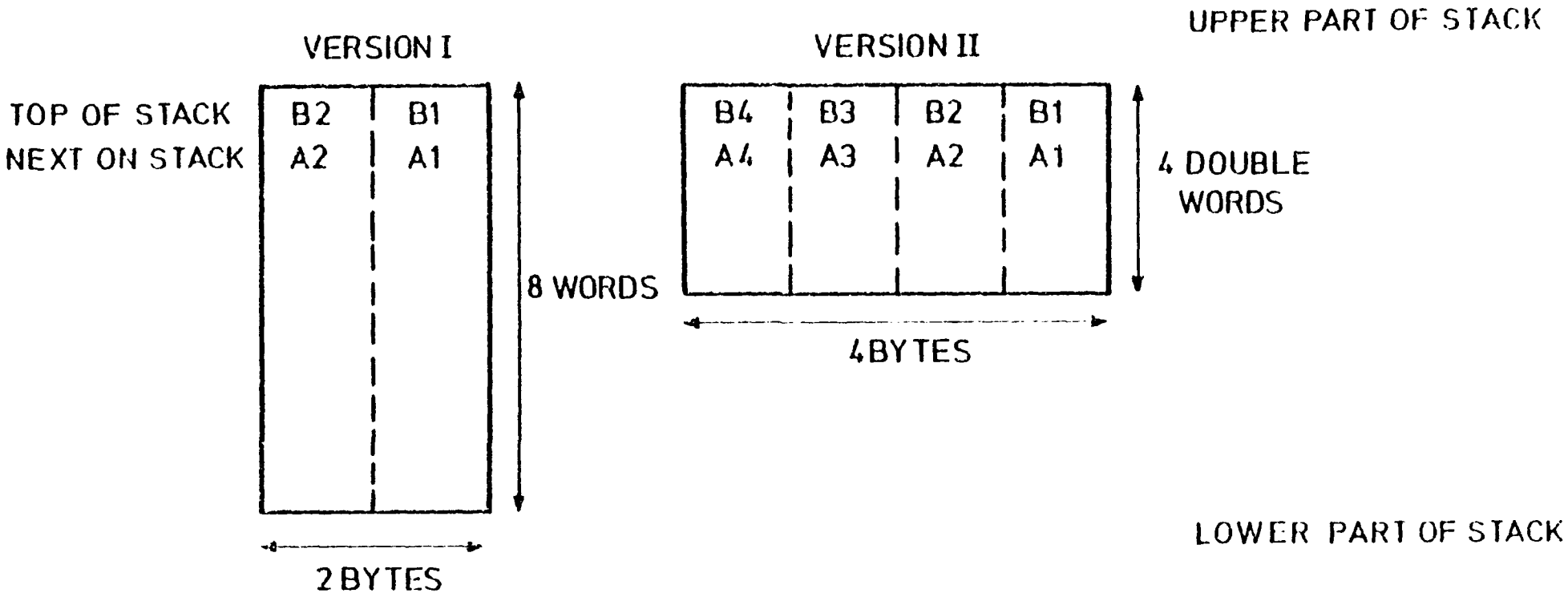


FIGURE 1. THE AM9511 STACK ORGANISATION

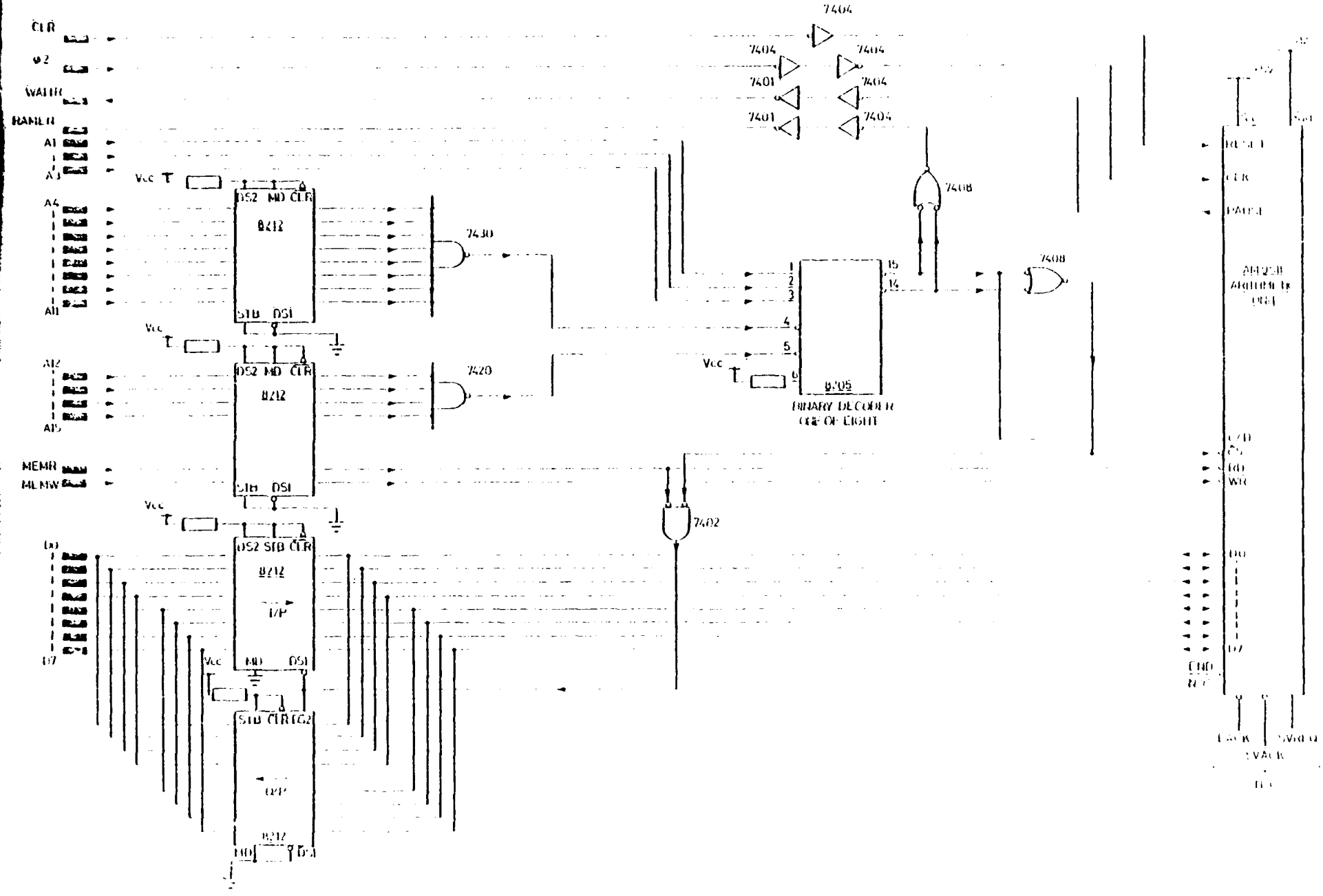


FIGURE 2 A SIMPLE MEMORY MAPPED HARDWARE INTERFACE

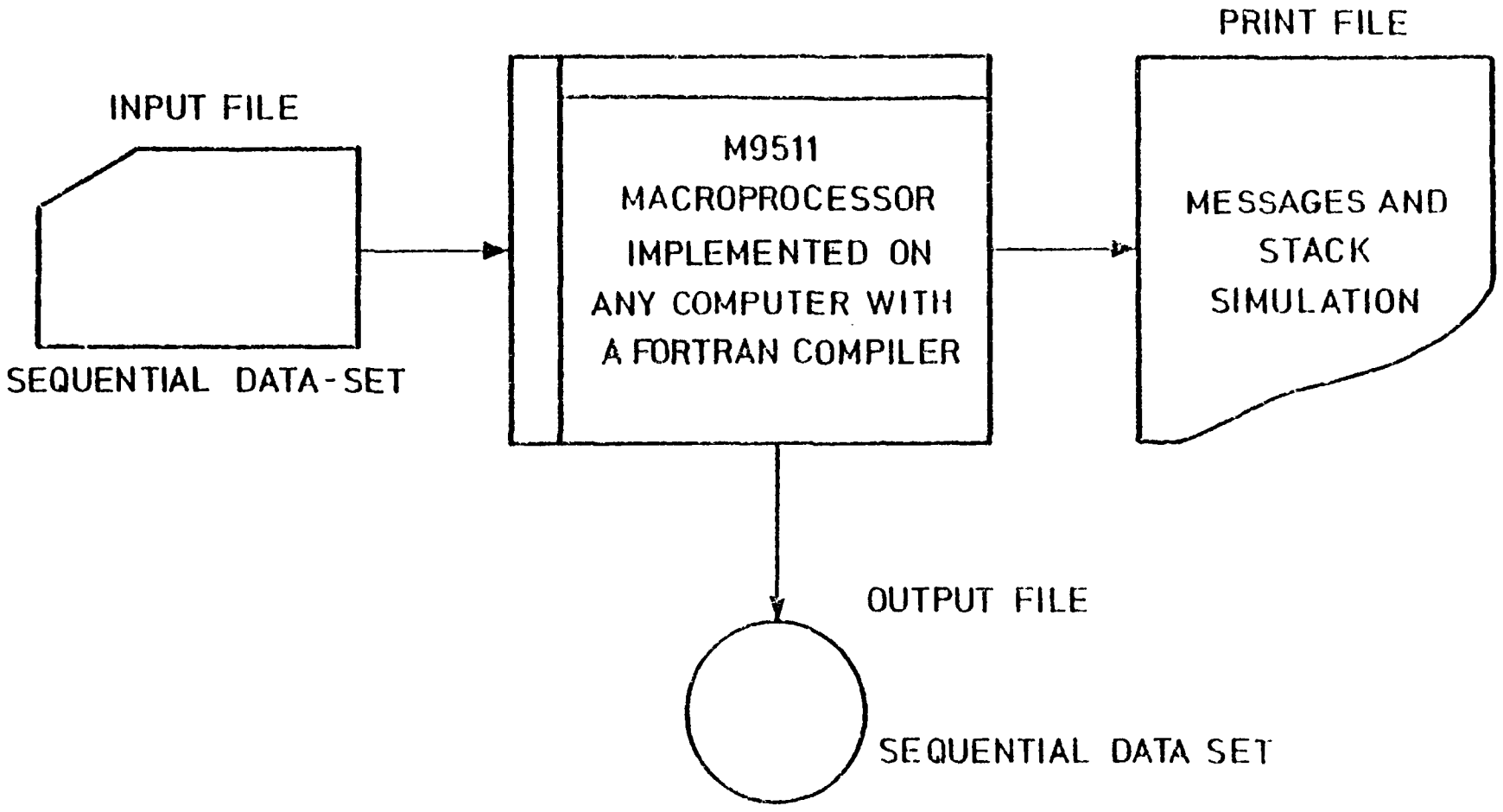


FIGURE 3. M9511 DATA FLOW