UT-MIX REFERENCE MANUAL

by

James L. Peterson, Editor

January, 1977 TR-64

Department of Computer Sciences

The University of Texas at Austin

ABSTRACT

This document describes the UT-MIX system used at the University of Texas. UT-MIX is a complete implementation of the MIX machine, as described by Knuth, and includes significant extensions. This document describes the differences and extensions to the basic MIX system defined by Knuth. It is intended primarily as a user's reference manual. The basic configuration, instruction set, assembly language, macro and conditional assembly instructions and input/output system are described.

This document is a revision of an earlier reference manual prepared and published by the Department of Computer Sciences at the University of Texas at Austin. The current revision was prepared and editted by Professor J. L. Peterson. The previous versions were not identified as to author. Any comments, corrections or suggestions concerning the present revision should be directed to:

James L. Peterson
Department of Computer Sciences
The University of Texas
Austin, Texas 78712

(512) 471-4353

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

MIX is a mythical computer, designed and used by Donald E. Knuth to give programming examples throughout his series of books, The Art of Computer Programming. UT-MIX is a complete system, designed for the CDC 6000-series computers at the University of Texas at Austin. It provides users of this facility the opportunity to write and run MIX programs.

UT-MIX has all of the features described by Knuth in references 1, 2, and 3 below, and includes several significant extensions. This paper describes the differences between UT-MIX and Knuth's original specifications.

Knuth's specifications for MIX appear in all three of the references below. This document assumes the reader has access to at least one of these books.

This document supersedes and replaces all earlier references on UT-MIX prepared at the University of Texas at Austin.

References

1. Knuth, Donald E., MIX, Addison-Wesley, 1970, 48 pp.

2. Knuth, Donald E., The Art of Computer Programming, Volume 1: Fundamental Algorithms, (Second Edition), Addison-Wesley, 1973, pp. 120-227.

3. Knuth, Donald E., The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Addison-Wesley, 1969, pp. 565-595.

CHAPTER 2

THE UT-MIX SYSTEM

UT-MIX includes a simulator for the MIX computer and an assembler for the MIXAL assembly language. This Chapter describes the basic features of the simulated MIX machine and its interface with the UT CDC 6000-series system.

2.1 ACCESS TO UT-MIX

2.1.1 Control Cards And Job Parameters

A field length of 40000 (octal) is required for the MIX system in all cases. This is both a minimum and a maximum. Larger field lengths are unnecessary and undesirable.

The MIX system is called by a MIX control card of the form,

```
    MIX.
or
    MIX, parameters.
```

This card calls the MIXAL assembler to translate a user program deck written in the MIXAL assembly language. After assembly, the program is executed by the simulator.

The possible parameters for the MIX control card are,

I = filename	Input file. This file is used for input to the MIX assembler.
O = filename	Output file. The assembly listing and MIX program output will be printed on this file.
X = filename	External text file. If specified, the MIXAL assembler first takes its input from this file. When an end-of-record is found on the X file, input is switched to the standard input file (the one specified by the I parameter).
D = filename	Data file. The assembled MIX program will use this file

for input when IN operations are done on the card reader (MIX I/O device 16).

The parameters may occur in any order. Defaults are INPUT for I and D, OUTPUT for O, and no X file. Thus a MIX. control card is equivalent to MIX,I=INPUT,O=OUTPUT,D=INPUT. If no D file is specified, the I file is used for input data. Thus, MIX,I=DMX. is equivalent to MIX,I=DMX,D=DMX. If program input should come from DMX, but data should come from INPUT, then either of the following two control cards could be used: MIX,I=DMX,D=INPUT. or MIX,X=DMX. This latter card uses the default values of INPUT for I and D.

2.1.2 Deck Structure

```
Sequence card.
Password card.
MIX.
7/8/9
MIXAL source program
7/8/9
data for MIX program
6/7/8/9
```

2.2 CENTRAL PROCESSOR AND MEMORY

2.2.1 Words

Each UT-MIX word consists of five bytes and a sign. The sign position has only two possible values, + and -. Each UT-MIX byte contains six bits, giving the limits:

```
    00 through 63   (00 to 77 octal)  for each byte;
    00 through 4095 (00 to 7777 octal) for two bytes;
 absolute value 1073741823 (7777777777 octal) for each word.
```

The partial fields of instruction words and their field specifications are: numbered 0 through 5, left to right, beginning with the sign.

Sign	AA-field	I-field	F-field	C-field
(0:0)	(1:2)	(3:3)	(4:4)	(5:5)

2.2.2 Registers

The UT-MIX central processor (CPU) has the following registers:

A register (accumulator): five bytes (30 bits), and sign.

X register	(extension): five bytes (30 bits), and sign.
I registers	(index):I1,I2,I3,I4,I5 and I6. Each index register has two bytes (12 bits) and sign.
J register	(jump address): two bytes (12 bits). Sign is +.

Additionally, there is an overflow toggle (one bit, either on or off) and a comparison indicator (three values: less, equal or greater). Both function exactly as specified by Knuth.

Arithmetic in all registers is done in integer mode. Numeric data are represented as a 30 (or 12) bit absolute value with a separate sign field. UT-MIX includes some boolean operations, which operate only on the 30 value bits of the A register.

2.2.3 MIX Memory

UT-MIX memory includes 4022 words of storage. Locations 0000 through 3999 (decimal) are available for normal use by a program for instructions and storage. Locations 4000 - 4021 have special uses and, while a program has access to them, their contents may be modified by activities of the simulator. (These locations are described in Sections 2.4, 2.5, and 6.5).

2.2.4 Indirect Addressing

UT-MIX provides for indirect addressing and double indexing as described by Knuth[2] pp. 248-249. The I-field of each instruction has the form 8*I1+I2, where $0 \leq I1 \leq 7$, $0 \leq I2 \leq 7$. In MIXAL this is written as:

```
        OP ADDRESS,I1:I2
    or
        OP ADDRESS,I2         if I1 = 0
```

When the instruction is executed, ADDRESS is modified by I1 then by I2 and OP is performed with the new address. If I1 and I2 are index register specifications (0<I1<7, 0<I2<7) the contents of the appropriate index registers are added to ADDRESS. If I1 = 7 the new address is taken from the 0:3 field of the location specified by ADDRESS (indirection). If I2 = 7 the new address is taken from the 0:3 field of the location specified by ADDRESS modified by I1. The case I1 = I2 = 7 is not allowed.

2.2.5 Instruction Timing

CPU instructions have the execution times prescribed by Knuth. UT-MIX specifies a MIX time unit to be equal to 1 microsecond. The behavior of input/output (I/O) devices is tied to this time scale.

--all load, store, compare and shift instructions:	2 units
--the ADD and SUB arithmetic operations:	2 units
--the MUL arithmetic operation:	10 units
--the DIV arithmetic operation:	12 units
--the MOVE operation, to MOVE n words in memory:	2n+1 units
--all other instructions:	1 unit

Note that I/O instructions (IN, OUT, IOC) require only one unit to start an I/O operation, but the operation may not be complete for several thousand units (milliseconds), depending on the characteristics of the I/O device and other I/O activity. A UT-MIX program may continue to execute CPU instructions of all kinds while I/O activities are in progress.

Each level of indirect addressing used by an instruction requires an additional time unit.

2.3 INPUT-OUTPUT DEVICES

The UT-MIX I/O subsystem provides some capabilities beyond those specified by Knuth. A full set of typical devices is available. Complete operating specifications for each device, and for the I/O subsystem, appear in Chapter 6 to supplement the brief description given here.

2.3.1 Magnetic Tapes -- Units 0,1

Two magnetic tape units are available for binary read/write operations. Each IN or OUT instruction transfers 100 full MIX words and requires 16 milliseconds to complete. Each tape may be spaced forward or backward, or rewound under program control.

2.3.2 Magnetic Disks -- Units 8,9

Two random-access, moving head disks are available for binary read/write operations. Each disk contains 64 tracks, with 64 100-word sectors on each track, giving each unit a total capacity of 4096 records (409,600 words). Each IN or OUT instruction transfers 100 full MIX words with a disk address specified by the contents of the X register. A transfer requires an average of 32 milliseconds for access, plus 32 milliseconds per track if head movement is required, plus 1 millisecond for actual data transfer.

2.3.3 Magnetic Drum -- Unit 10

One random-access, head-per-track drum is available for high speed binary read/write operations. Its organization and operation is similar to that of the disk units, but its fixed heads and higher rotation speed give it an average access time of only 16 milliseconds for any record. Each IN or OUT operation transfers 100 full MIX words with a drum address specified by the contents of the X register and requires one millisecond, after access delay. The capacity of the drum is 4096 100-word records.

2.3.4 Card Reader -- Unit 16

One high speed card reader is provided. The IN instruction transfers 80 characters into 16 words of MIX memory. The operation requires 50 milliseconds, simulating a rate of 1200 cards per minute.

2.3.5 Card Punch -- Unit 17

One low speed card punch is provided. For each OUT instruction, the contents of 16 words of MIX memory are transferred to 80 characters of punch output. The operation requires 200 milliseconds, simulating a rate of 300 cards per minute.

2.3.6 Line Printer -- Unit 18

One high speed printer is provided. Each OUT instruction transfers 120 characters (24 words) from MIX memory to the device at a simulated rate of 1200 lines per minute (50 milliseconds per operation). Page ejects, single and multiple line feeds may be performed under program control using the IOC instruction.

2.4 CLOCKS

Two internal clocks are used by the MIX simulator to record simulated CPU active time and simulated total elapsed time. These values are available to the user in a printed summary, and a running program may also access the current value of the CPU activity clock through MIX memory cell 4001.

2.4.1 User Interval Timer

The simulator automatically increments the value in MIX memory location 4001 as each instruction is executed, adding the number of time units (microseconds) required for the instruction. Using normal load and store instructions, the user may read this value, or preset or reset the value as desired.

2.4.2 Simulation Summary

The internal clock values are printed as the final lines of each simulation output. The summary shows:

--the number of units of MIX CPU active (time spent executing instructions)

--the number of units of MIX CPU idle time (usually awaiting completion of an I/O operation)

--the number of units of total simulated time (sum of active time and idle time).

All of the above are printed as decimal integers and should be interpreted as microseconds. The simulator also issues a dayfile message giving the number of seconds of CDC system TM (charge time) used by the assembler (excluding simulation) and by the simulator (excluding assembly of the MIXAL deck).

2.5 EXECUTION DIAGNOSTICS

UT-MIX provides two diagnostic features during execution of a MIX program: an execution trace and fatal error messages.

2.5.1 Trace

The simulator checks the contents of MIX memory location 4000 immediately after execution of each program instruction. If the contents of location 4000 are non-zero, a line of printed output is generated giving the location of the instruction just executed, the instruction, and the contents of all registers and toggles at the completion of the instruction.

The user may turn on the trace feature with a store instruction which places a non-zero value in location 4000 (e.g. STJ 4000), and turn off the feature with a store instruction which places a zero value in the cell (e.g. STZ 4000). Trace output is limited to a total of 500 lines for each program. The complete format of trace output is given in Appendix A.

2.5.2 Fatal Error Messages

As each MIX instruction is decoded prior to execution, it is checked for validity. Illegal instructions are trapped and cause the simulator to stop, printing an error message. The user has no control over this feature (unless an illegal instruction is deliberately issued).

The error termination process clears all I/O devices to assure that the last line of printed output (and the last card punched) are preserved, and generates one line of trace output. This trace differs from the normal trace in that the offending instruction and its address are printed, but all register values are as they were before the instruction was executed.

A complete list of fatal error conditions and associated messages is given in Appendix B.

CHAPTER 3

THE UT-MIX INSTRUCTION SET

All MIX instructions specified by Knuth have been implemented in UT-MIX except for the floating-point operations mentioned in references 2 and 3. With only minor exceptions noted below, all of these instructions have the exact formats and effects prescribed by Knuth. UT-MIX includes some extensions of the original instructions, and some additional instructions which are described in this Chapter. The MIXAL mnemonics are used in these description. See also appendices D and E. The symbol M stands for the calculated effective address (Section 2.2.4).

3.1 LOADING OPERATORS

LDA, LDX, LDi; $i = 1,2,...,6$.
LDAN, LDXN, LDiN; $i = 1,2,...,6$.

These instructions operate exactly as prescribed by Knuth. The M-value of the instruction must be in the range $0 \leq M \leq 4021$. An attempt to load an index register with a non-zero value exceeding 12 bits will stop the simulator with an error message.

3.2 STORING OPERATORS

STA, STX, STi; $i = 1,2,...,6$; STJ, STZ.

These instructions operate exactly as prescribed by Knuth. The M-value of the instruction must be in the range $0 \leq M \leq 4021$.

3.3 ARITHMETIC OPERATORS

ADD, SUB, MUL, DIV.

These instructions operate exactly as prescribed by Knuth. Overflow occurs for ADD and SUB if the absolute value of the result exceeds 1073741823 (7777777777 octal). If the DIV operation encounters a zero divisor or a quotient larger than 1073741823, the A and X registers are set to zero and the overflow toggle is turned on.

3.4 COMPARISON OPERATORS

CMPA, CMPX, CMPi; i = 1,2,...,6.

These instructions operate exactly as prescribed by Knuth. The M-value of the instruction must be in the range $0 \leq M \leq 4021$.

3.5 ADDRESS TRANSFER OPERATORS

```
INCA, INCX, INCi;  i = 1,2,...,6.  (increment)
DECA, DECX, DECi;  i = 1,2,...,6.  (decrement)
ENTA, ENTX, ENTi;  i = 1,2,...,6.  (enter)
ENNA, ENNX, ENNi;  i = 1,2,...,6.  (enter negative)
```

These instructions operate exactly as prescribed by Knuth. The overflow toggle is turned on for an A or X register result which exceeds an absolute value of 1073741823. Simulation is stopped with an error message if the absolute value of an index register result exceeds 4095 (7777 octal).

3.6 JUMP INSTRUCTIONS

```
JMP, JSJ, JOV, JNOV, JL, JE, JG, JGE, JNE, JLE.
JrN,  JrZ,  JrP,  JrN,  JrNZ,  JrNP;  r = A,1,2,...,6,X.
```

These instructions operate exactly as specified by Knuth. The M-value of the instruction must be in the range $0 \leq M \leq 4021$.

3.6.1 UT-MIX: Additional Jumps

JrE	r = A, 1, 2, 3, 4, 5, 6, X. Jump if the register is even. C = 40, 41, 42, 43, 44, 45, 46, 47; F = 6.
JrO	r = A, 1, 2, 3, 4, 5, 6, X. Jump if the register is odd. C = 40, 41, 42, 43, 44, 45, 46, 47; F = 7.

The least significant bit of the specified register is examined; all other bits, and the sign, are ignored. If the least significant bit is zero, the register is even. If the least significant bit is one, the register is odd.

3.6.2 Restriction On All Jumps

Simulation is stopped with an error message if an executed jump (in the case of a conditional jump, if the condition is satisfied) specifies its own location. This is done to avoid time-consuming non-productive loops.

3.7 SHIFT INSTRUCTIONS

SLA, SRA, SLAX, SRAX, SLC, SRC.

These instructions operate exactly as specified by Knuth, if $M \geq 0$.

3.7.1 UT-MIX Extensions

SLB, SRB	(shift left or right M bits). C = 6; F = 6 for SLB; C = 6; F = 7 for SRB.

These instructions produce the same results as SLAX, and SRAX, with both A and X registers participating in an end-off shift except that the value of M specifies the number of bits, instead of bytes, to be shifted.

The original shift instructions (SLA, SRA, SLAX, SRAX, SLC, SRC) are extended to permit bit shifts by using the value of M. If $M < 0$, it is interpreted to mean bits instead of bytes. (Note that the value of M does not in any way imply the direction of the shift).

3.8 INPUT-OUTPUT OPERATORS

IN, OUT, IOC, JBUS, JRED.

Basically, all of these instructions operate exactly as specified by Knuth, though there are some differences depending on the I/O unit referenced. Their exact UT-MIX implementation is described in Chapter 6. For all IN and OUT instructions, the value of M must be in the range $0 \leq M \leq (4022 - \text{record size for the unit})$. IN, OUT and IOC instructions which reference disk (units 8, 9) or drum (unit 10) compute the disk or drum address from the contents of the X register (bytes 4:5).

3.9 MISCELLANEOUS OPERATORS

NOP, MOVE, HLT

These instructions operate exactly as specified by Knuth. The HLT instruction stops the simulator. The simulator always clears all I/O operations in progress as part of its normal (or abnormal) termination procedure. For the MOVE instruction $0 \leq F \leq 63$ and the M-value must be in the range $0 \leq M \leq (4022 - F)$. The value of the I1 register, which specifies the destination address for MOVE, must satisfy the same restrictions as M.

3.10 CONVERSION OPERATORS

NUM, CHAR

NUM and CHAR operate exactly as specified by Knuth. Overflow will occur with the NUM instruction if the character code in the combined A and X registers represents a decimal number larger than 1073741823.

OCT

OCT is a UT-MIX extension (C = 5, F = 3) to provide binary-to-octal conversion of the value in the A register. The value in the A register is converted into a 10-byte octal number which is put into registers A and X in character code.

3.11 SPECIAL A-X REGISTER OPERATORS

UT-MIX implements an extended set of operations affecting the A and X registers. These include three groups: A-X register exchange, A register sign operations, and unary and binary A register boolean operations.

3.11.1 A-X Register Exchange

XCH	(exchange the A and X registers) C = 5; F = 9.

The contents of the A and X registers, and their signs, are exchanged.

3.11.2 A Register Sign Operations

SSP	(set sign positive)	C = 5; F = 4.
SSN	(set sign negative)	C = 5; F = 5.
CHS	(change sign)	C = 5; F = 6.

The indicated operation is performed on the sign of the A register. The absolute value of the A register is unchanged. M is ignored.

3.11.3 A Register Logical Operations

The following operations alter the 30-bit value field of the A register. The sign of the register is unchanged in all cases. The letter V below designates the specified field of the contents of M, right-justified with leading zeros in a 30-bit word. The M-value associated with OR, XOR and AND instructions must fall in the range of $0 \leq M \leq 4021$.

LNG (logical negate). C = 5; F = 8.

The value field of the A register is complemented, bit for bit. M is ignored.

MSK (mask). C = 5; F = 10.

M (modulo 30) bits are turned on in the A register; the rest are turned off. If M > 0, the consecutive on bits are left justified. If M < 0, the mask is right justified.

OR	(logical sum)	C = 1; F = 7.
XOR	(logical difference)	C = 2; F = 7.
AND	(logical product)	C = 3; F = 7.

V is combined with the 30-bit value field of the A register to form the logical sum, difference or product. Examples of the results:

```
...0101...   OR   ...1100...  =  ...1101...
...0101...  XOR   ...1100...  =  ...1001...
...0101...  AND   ...1100...  =  ...0100...
```

CHAPTER 4

THE MIXAL ASSEMBLER

4.1 INTRODUCTION

The MIXAL assembler is a comprehensive macro assembly system for the MIX computer. It provides a symbolic programming language for the effective and efficient utilization of the MIX hardware. This Chapter describes the basic assembler, while Chapter 5 describes the macro and conditional assembly features of the assembler.

4.2 PROGRAM STRUCTURE AND ORGANIZATION

Each set of statements submitted as input to the MIXAL assembler must constitute a complete, self-contained program with all code necessary to perform a specific task. No independent assembly of subprograms is provided. The MIXAL assembler processes all statements up to and including the first occurrence of an END pseudo-instruction, ignoring all other statements before the occurrence of a data separator (7/8/9 card). The MIXAL assembler stores the assembled information into the memory of the MIX machine directly. Areas of the memory not specified to receive information by the assembled program are preset to zero by the MIXAL assembler.

4.2.1 Location Counter

The MIXAL assembler maintains a location counter which always specifies the address of the location into which the next word of assembled information will be placed. Normally the location counter is started at 0000 (octal) and is incremented by 1 for each statement processed. The value of the location counter may be directly set by the programmer, however, by use of the ORIG pseudo-instruction (see Section 4.5.2 below). When the special element * appears in an operand expression on a MIXAL statement, the value of the location counter will be used.

4.3 THE MIXAL LANGUAGE

4.3.1 Coding Format

A MIXAL program consists of a sequence of symbolic statements. Each statement contains a maximum of four fields in the order listed below. The format is essentially free field.

1. Location field - must begin in column 1.

2. Operation field - may begin in any column from 2 to 16.

3. Operand field - must begin before column 18.

4. Comments field - may begin after termination of the operand field or no earlier than column 18 if the operand field is empty.

Fields on a line are separated by one or more blanks. Blanks are interpreted as field separators except when embedded in the comments field, in a character data string, or in a parenthesized macro parameter.

Columns 73-80 of the line may be used only for comment information. A statement extending beyond column 72 will be truncated to 72 characters. Continuation cards are not possible.

Comment lines may be included in programs, being denoted by the appearance of a * in column 1 or by columns 1-17 being blank. These lines appear in the program listing but otherwise do not affect the assembly process. Any other configuration of symbols on a line will be interpreted as a statement to be assembled.

The standard format for source statement lines is:

column	content
1-10	location field
11	blank
12-15	operation field
16	blank
17-72	operand and comments fields
73-80	statement sequencing information

4.3.2 Statement Types

The statements processed by the MIXAL assembler fall into three categories:

1. A normal statement which is assembled and may produce stored information.

2. A statement which is bypassed by the assembler because of a conditional instruction test which failed. (See Chapter 5).

3. A statement which is part of a macro definition. (See Chapter 5).

4.3.3 Statement Fields

4.3.3.1 Location Field -

The location field may be blank or may contain one of:

local symbol
symbol

4.3.3.2 Operation Field -

The operation field may be blank or may contain one of:

MIX machine operation code mnemonic
pseudo-instruction
macro name (See Chapter 5)

4.3.3.3 Operand Field -

The content of the operand field is dictated by what is in the operation field.

For a MIX machine operation mnemonic the operand has the general form:

```
A-part,I-part(F-part)
```

The A-part corresponds to the sign and AA-field, the I-part to the I-field, and the F-part to the F-field of the instruction word as defined in Section 2.2.1. Each of A-part, I-part, and F-part may be an expression consisting of symbols and numeric constants combined by operators. Any or all of the parts may be absent, in which case zero values will be used for the A-part or I-part. The default F-part which is appropriate for this instruction (usually 0:5) will be used if the F-part is absent.

For a pseudo-instruction the content of the operand field will be determined by the operation field as described in Section 4.5 below.

The operand field for a macro name in the operation field is a sequence of character strings separated by commas. Further description is in Chapter 5.

4.3.3.4 Comments Field -

This field is completely optional and may contain any combination of characters.

4.3.3.5 Interpretation Of Special Cases -

When some or all of the statement fields are blank, the MIXAL assembler makes certain assumptions for their values. Given here is a summary of most of the special cases that may arise and the interpretation given to them.

If the location, operation, and operand fields are all blank, the line is treated as a comment, and generates no information.

If the operation and operand fields are blank but the location field is non-blank, a word containing zero will be assembled.

If the operation field is blank but the operand field is non-blank, the operand field will be assembled as for a MIX machine instruction and a word will be assembled with a zero operation code (no-operation).

If only the operand field is blank the operand of the operation will be treated as zero.

4.3.4 Local Symbols

A local symbol in the MIXAL language is any character string of the form:

```
<digit>H  or  <digit>B  or  <digit>F
```

where <digit> is any single decimal digit, 0-9. Local symbols play a special role in the MIXAL language, and may each appear as many times as desired. Local symbols represent values which are assigned to the local symbols according to usage as described below.

4.3.4.1 The <digit>H Symbol -

Local symbols of the form <digit>H may be used only in the location field of a MIXAL statement. If the operation field of the statement is EQU, the value assigned to the local symbol is the value of the expression in the operand field and may be a full MIX word in size. If the operation field is any machine operation mnemonic or any other pseudo-instruction, the value assigned to the local symbol is the current value of the location counter when the local symbol is encountered.

4.3.4.2 The <digit>B Symbol -

Local symbols of the form <digit>B may appear only in the operand field of a MIXAL statement. The value referred to by such usage is the value assigned to the most recent previous occurrence of the <digit>H local symbol with the same <digit>.

4.3.4.3 The <digit>F Local Symbol -

Local symbols of the form <digit>F may appear only in the operand field of a MIXAL statement. The value referred to by such usage is the value assigned to the next occurrence of the <digit>H local symbol with the same <digit>.

4.3.4.4 Examples Of MIXAL Local Symbols -

The following are examples of legal local symbols.

```
0H    2F    0B    2H    3H    3B    9F    9H    9B
```

4.3.5 Symbols

A symbol is a sequence of 1 to 10 letters and/or digits containing at least one letter, excluding those character sequences meeting the definition of a local symbol (see Section 4.3.4 above). Each such symbol represents a value which is assigned to the symbol according to usage as follows:

1. If the symbol appears in the location field of a machine instruction or of most pseudo-instructions the value assigned to the symbol is the current value of the location counter.

2. If the symbol appears in the location field of an EQU pseudo-instruction the value assigned to it is the value of the expression in the operand field.

3. If the symbol never appears in a location field, but does appear in an operand field, it will be defined by MIXAL with a value of the address of a location following the end of the program.

Any given symbol may appear in a location field only once in any MIXAL program.

4.3.5.1 Examples Of MIXAL Symbols -

The following are legal MIXAL symbols.

MIXAL	A1	1A
123456789Q	123S456	TABLE3

4.3.6 Forward References

The MIXAL assembler is a one-pass processor, that is, it examines each statement only once. Values are assigned to symbols when they appear in the location field of some statement or at the end of the program. Thus a symbol occurring in the operand field of some statement which has not previously appeared in a location field will be undefined at that point. If a symbol appears in such a circumstance, it is termed a forward reference as it refers to a value which will be determined later.

A forward reference may be used in only one way in a MIXAL program -- as the A-part of a machine operation. In such use, the forward reference must appear by itself with no sign or other operators.

Occurrences of a given symbol are forward references only until that symbol occurs in the location field of some statement.

4.3.7 Constants

A constant is a string of 10 or fewer digits specifying a decimal integer value. Constants are full-word values and are bounded in absolute magnitude by 1073741823. Any constant representing a value larger in magnitude than 1073741823 will be reduced modulo 1073741823 before use. Constants may be used in the operand fields of machine instructions and pseudo-instructions.

4.3.8 Special Element

The character * appearing in an operand field represents the value of the location counter when the * is encountered.

4.3.9 Expressions

Expressions enable the MIXAL programmer to compose values for the various parts of a machine operation and certain pseudo-instructions from symbols and constants. Elements of expressions may be:

element	meaning
symbol	the value assigned to the symbol is used in the evaluation of the expression.
constant	the value of the constant is used in the evaluation of the expression.
special element (*)	always stands for the current value of the location counter

Note well that forward references are not elements of expressions.

An expression may consist of:

1. An element, or
2. An element preceded by a plus or minus sign, or
3. An expression followed by a binary operator followed by an element.

The six admissible binary operators are

+, -, *, /, //, and :

Any of these operators may appear between any two elements of an expression.

4.3.9.1 Evaluation Of Expressions -

Expressions are evaluated as full-word (5 bytes plus sign) quantities. If the value of the expression is to be placed into a field that is smaller than a full word, it is truncated after evaluation.

Evaluation of expressions proceeds in a strictly left-to-right manner with no hierarchy of operations. Evaluation begins by evaluating the first element of the expression. If this element was preceded by a minus sign, the value is negated. This process defines the current value of the expression. If a binary operator follows this part of the expression, then the element following the operator is evaluated and its value is combined with the current value of the expression according to the meaning of the operator forming a new current value of the expression. Evaluation may then proceed using this current value as the left operand of the next binary operator until the expression is exhausted.

Meanings of the binary operators are given below:

expression	meaning
a+b	the sum of the two operands, a+b
a-b	the difference of the two operands, a-b
a*b	the least significant 5 bytes of the product of the two operands, a*b
a/b	the integer part of the quotient of the two operands, a/b
a//b	the fractional part of the quotient of the two operands, a/b, treated as an integer
a:b	is equivalent to: a*8+b

Some example expressions with their values:

expression	value
-1+3	2
-1+5*20/6	13
1//3	357913941
1:3	11
*+4	location counter + 4
***	location counter times location counter

4.3.10 W-values (Word Values)

A W-value is a MIXAL construct used to form data values with specific items in specific parts of a word. W-values are used in the operand field of a CON pseudo-instruction and in the specification of literal values (see Section 4.3.11 below). A W-value is,

1. An expression, or
2. An expression followed by a field specification in parentheses, or
3. A W-value followed by a comma followed by a W-value of the form specified by 1 or 2 above.

A W-value denotes the contents of one MIX word determined as follows:

Let the W-value have the form

```
e1(f1),e2(f2),...,en(fn)   where n > 0
```

The ei are expressions, and the fi are field specifications. The desired result is the final value which would appear in memory location CON if the following hypothetical program were executed:

```
STZ  CON
LDA  c1
STA  CON(f1)
LDA  c2
STA  CON(f2)
 .
 .
 .
LDA  cn
STA  CON(fn)
```

Here c1,c2,...,cn denote locations containing the values of expressions e1,e2,...,en respectively. Each fi must be of the form 8*Li+Ri where 0 ≤ Li ≤ Ri ≤ 5.

Examples:

```
                              -------------------
1              is the word    | +|              1|
                              -------------------
1,-1000(0:2)   is the word    | -|1000 |        1|
                              -------------------
-1000(0:2),1   is the word    | +|              1|
                              -------------------
```

4.3.11 Literals

A literal is a reference to a constant whose space allocation will be performed by the assembler. A literal may appear in the A-part of the operand field of a MIX machine operation. Every literal use is a forward reference to a word which will contain the data item specified in the literal. The word containing the data will be allocated by the MIXAL assembler at the end of the program.

A literal is composed of a W-value of less than 10 characters enclosed in = signs. Examples:

```
=2=    =1(0:3)=  =-2,1(2:3)=
```

4.4 OPERATION CODES

Refer to Chapter 3, Appendix D or Appendix E for descriptions of the legal MIX machine operation code mnemonics.

4.5 PSEUDO-INSTRUCTIONS

Pseudo-instructions provide the programmer with the ability to control certain operations of the assembler, to specify the physical layout of his program, and to create data items. All programs must contain at least an END pseudo-instruction.

4.5.1 Assembler Control - END

Format:

```
location  operation  operand
field      field     field
--------  ---------  -------

ignored   END        expression
```

Every program must have an END statement as its last card. The statement causes the assembler to stop assembling the program. The expression in the operand field is evaluated and its value specifies the location in the program at which execution of the program is to begin. If the operand field is empty, execution will begin at location 0.

4.5.2 Counter Control - ORIG

Format:

```
location  operation  operand
field      field     field
--------  ---------  -------

symbol    ORIG       expression
or empty
```

ORIG sets the location counter to the value of the expression in the operand field. If a symbol appears in the location field, it is assigned a value equal to the location counter before it is set by the ORIG pseudo-instruction.

ORIG is commonly used to allocate a block of memory words. The statement:

```
        X           ORIG *+10
```

for example, reserves a block of 10 words starting at location X.

4.5.3 Symbol Definition - EQU

Format:

location field	operation field	operand field
symbol	EQU	expression

There must be a symbol in the location field of an EQU statement. The EQU statement assigns the value of the expression in the operand field as the value of the symbol.

4.5.4 Data Generation - ALF

Format:

location field	operation field	operand field
symbol or empty	ALF	anything

The ALF pseudo-instruction reserves one memory word and fills it with a + sign and five character codes. The operand field of the ALF pseudo-instruction begins in the third column after the F of ALF and consists of the 5 characters starting in that column. (For example, if the ALF starts in column 12, its operand field will start in column 17 and extend through column 21). Any character may be in the operand of ALF and its code will be stored in the word.

4.5.5 Data Generation - CON

Format:

location field	operation field	operand field
symbol or empty	CON	W-value

The W-value in the operand field of the statement is evaluated (see Section 4.3.10) and the resulting full-word value is stored in a memory word reserved by the CON pseudo-instruction. If a symbol appears in the location field, its value will be the address of the word reserved by the CON pseudo-instruction.

4.5.6 Listing Control - LIST

Format:

location field	operation field	operand field
ignored	LIST	sequence of options separated by commas

The LIST pseudo-instruction exerts control over the listing produced by the assembler. The available options are:

option	meaning
L	produce a listing of statements following
-L	suppress the listing of statements following
M	show the lines of macro bodies when they are expanded
-M	do not show the lines of macro bodies when they are expanded

The LIST pseudo-instruction may appear any number of times within a program with various combinations of options. The options in effect when the assembler begins are L and -M. No matter what the state of the listing options, all lines which contain erroneous statements will be listed.

4.5.7 Trace Limit Setting - TRLM

Format:

location field	operation field	operand field
ignored	TRLM	integer constant

The operand must be an unsigned integer constant. The operand value is taken as the maximum number of lines of trace output which are to be generated before the execution is terminated for exceeding the trace limit. The value may be any integer between 0 and 500. The default limit is 100. The last TRLM pseudo-instruction encountered during assembly defines the trace limit for execution.

4.6 THE ASSEMBLY LISTING

The MIXAL assembler produces a listing of the program as it is assembled. This listing shows the lines of the source program, the information resulting from the assembly of each line, and any errors detected in the form or meaning of the source program.

The first line of each page of the listing is a header giving the version number of the MIXAL assembler, the time and date of the beginning of assembly, and the listing page number. Subsequent lines of the listing consist of one printed line for each line of source program.

Each line consists of five (5) fields. From left to right these fields are:

field	content
errors	contains up to 4 single-character error codes. This field is blank if no errors were detected on the line.
location	address (in octal) of the word whose contents are specified by the present line.
value	contents (in octal) of the word filled by the present line. Words containing machine operations are shown in four separate pieces: the signed A-part, the I-part, the F-part, and the C-part. Source lines creating a full-word value contain a signed 10-digit octal number in this field.
line image	the 80-character image of the source program line.
line number	the number of the source card counted from the beginning of the program deck.

When the listing has been suppressed by use of the LIST pseudo-instruction, only lines containing error messages will be printed. In this case, the correct line number will be printed to enable the user to locate the erroneous line in his source deck.

The location and value fields will be blank on lines containing operations with no location or value is associated (comments, macro definitions, list and conditional pseudo-instructions). Certain pseudo-instructions (EQU, ORIG, END) which do not fill a memory word are listed with a blank location field and the value field shows the value of the operand. Literals are defined at the end of the program and each contains **LITERAL** in the line image field.

After the END pseudo-instruction of the program a line

```
n ERRORS IN MIXAL PROGRAM
```

is printed where n is the number of errors in the source program detected by the MIXAL assembler. If there were any errors detected, the listing is followed by one or more pages of error summary information. For each kind of error that occurred the error summary gives a message:

```
ERROR x OCCURRED ON LINE(S) y,y,...
```

where x is the error type and y,y,... are line numbers. After this message, a brief explanatory note is printed about the error.

4.7 ERROR CODES

The MIXAL assembler can detect a number of erroneous conditions occurring in the source statements of a program. The existence of an error is indicated by the appearance of a single-letter code in the errors field of the listing. The assembler may detect more than one error on any given line, and will report up to four errors. Note that the occurrence of strings such as 12 in the errors field does not denote the occurrence of error twelve, but rather the occurrence of both error 1 and error 2. The various error codes and their meanings are given below.

Code	meaning
C	error detected while filling-in forward references. The chain of forward references to a given symbol must be strictly forward-pointing. Probable cause of this error is improper use of the ORIG statement to position code.
D	the symbol in the label field is doubly-defined. This occurrence of the symbol is ignored and the first definition will be used.
E	the operand of an ORIG pseudo-instruction is negative. The absolute value of the operand is used.
F	a forward reference was used in an expression. The expression is given the value 0.
G	overflow of the internal stack used to process nested conditional assembly pseudo-instructions and macros. This error necessitates termination of the assembly. To correct, reduce the level of nesting of conditional assembly pseudo-instructions and/or the number of local symbols in macros.
H	the operand of the TRLM pseudo-instruction is not an integer constant or exceeds 500. The trace limit must be an integer

constant in the range 0 to 500. The incorrect limit is ignored and the previous limit remains in effect.

I	the operand field of a conditional assembly pseudo-instruction is formatted incorrectly.
K	incorrect nesting of conditional assembly pseudo-instructions. This error causes assembly to stop.
L	the label field of this line contains something other than a valid MIXAL symbol. It is ignored.
N	a number has exceeded the largest allowed magnitude of 1073741823. The number is truncated to 5 bytes.
O	the operation field of this line does not contain a valid MIXAL operation mnemonic or macro name. An illegal instruction is assembled.
P	format error in the parameter specifications on a MACR pseudo-instruction or a macro call.
Q	the EQU pseudo-instruction on this line does not have a label.
R	the A-part of a storage-referencing machine operation or the operand of an ORIG pseudo-instruction exceeded 4021. The value used is the generated value modulo 4022.
S	the F-part of the current instruction is larger than 45 (does not occur for a MOVE instruction). A zero F-part is substituted.
T	the transfer address in the operand field of the end statement is invalid. Either it is missing or is not in the range 0 to 3999. A transfer address of 0 is used.
U	a <digit>B symbol occurred in an expression for which no corresponding <digit>H symbol appears previously. The symbol is given the value zero.
W	a literal exceeds the maximum width of nine characters. The literal is truncated from the right to nine characters.
X	separator error. Fields of the source statement were not separated by blanks.
1	a symbol is longer than 10 characters. Only the first 10 characters of the symbol are used.
2	a number exceeds 10 decimal digits. Only the first 10 characters are used.
4	in the operand field of the statement a symbol or constant was expected in some position and was not found. The probable cause of this error is a mispunched character.

5 a character expected to be a binary operator in an expression was not one of the allowed operators.

6 the A-part of this machine operation is incorrect. It is not an expression nor is it vacuous. Check the keypunching.

7 the index part of this machine operation is incorrect. If no comma follows the A-part then next character should be a (or a blank. This error may occur improperly if there was something wrong with the A-part itself.

8 the F-part is incorrect due to a missing). This error may occur improperly if the F-part contains an illegal expression.

9 a W-value is incorrect. This error is usually caused by a missing comma, but can appear improperly if one of the expressions of the W-value itself is incorrect.

CHAPTER 5

MACRO ASSEMBLY AND CONDITIONAL ASSEMBLY

The UT-MIX assembler is a comprehensive macro assembly system for the MIX computer. The MIXAL assembly language has been extended from that defined by Knuth to include macros and conditional assembly, based loosely on the macro and conditional assembly capabilities of the CDC 6000 COMPASS assembler. This Chapter describes the pseudo-instructions which have been added to the MIXAL language for macros and conditional assembly, and indicates how they are to be used.

5.1 MACROS

A macro is a named sequence of statements. The use of a macro requires two steps: defining the macro and calling the macro. A statement which has a macro name in the operation field results in the sequence of statements identified by that name being assembled at that point in the program. Such a statement is termed a macro call. The macro call may also contain in the operand field a set of parameters which will be substituted for defined parameters in the statements of the macro.

5.1.1 Macro Definition

To define a macro, the programmer must specify the sequence of statements comprising the macro, identify the substitutable parameters of the macro, and name the macro. A macro definition consists of three parts:

macro heading	a MACR pseudo-instruction which states the name of the macro and identifies the substitutable parameters.
macro body	the sequence of statements which constitute the code to be generated by the macro.
macro terminator	an ENDM pseudo-instruction which terminates the macro definition.

A macro definition may appear anywhere in a program prior to the first call of that macro. A given macro may be redefined at any time with the latest definition applying to each macro call.

5.1.1.1 Macro Heading -

The macro heading line is a MACR pseudo-instruction statement and has the form:

```
location  operation  operand
field      field     field
--------  ---------  -------

macro     MACR       up to 10 parameter specifications
name                 or empty
```

The location field contains the name by which the macro is to be known. This name may be any legal symbol of 7 or fewer characters except that END, ENDM, and local symbols may not be used. The symbol used as a macro name stands for the macro only when used in the operation field of a subsequent statement. Other uses of the symbol stand for a value unrelated to the macro definition.

If a macro name is identical to a machine operation mnemonic or a MIXAL pseudo-instruction, that name is redefined as the name of the macro and the occurrence of that name in the operation field of a subsequent statement stands for the macro. In other words, once a machine operation mnemonic or pseudo-instruction is used as a macro name, that machine operation or pseudo-instruction is no longer available for use. The macro may be redefined in terms of the CON pseudo-instruction to give the same effect as a machine operation.

The operand field of the macro heading contains 0 to 10 parameter specifications. Each parameter specification contains a symbol which it identifies as substitutable. The substitutable symbol may occur in the sequence of statements making up the body of the macro. When the macro call is made the macro call statement specifies (possibly) a string of characters which will be substituted for every occurrence of the substitutable parameter in the body of the macro at that point. Thus general-purpose macros may be defined which can be specialized to a particular purpose at the point of use. Each parameter specification has the form:

```
        symbol
    or
        symbol=default value
```

Parameter specifications in the operand field are separated by commas. The symbol may be any legal MIXAL symbol except a local symbol or the symbols ENDM or END. A macro call may or may not give a string to be substituted for a parameter. For the case in which a string is not specified, the first form of parameter specification shown above indicates that the null (empty) string is to be used whereas the second form shown above gives a specific default string to be used. The default value must agree with the syntax for parameter strings given in Section 5.1.2.1 below.

5.1.1.1.1 Examples Of Macro Headings -

```
        ABC         MACR A,B,C
```

defines a macro named ABC with parameters A, B, and C

```
        SAVE3       MACR NAME=NONE,RETURN=I6
```

defines a macro named SAVE3 with parameters NAME and RETURN

```
        GENDAT      MACR STMT=(CON 0),LABEL,BRANCH=(JMP 2F)
```

defines a macro named GENDAT with parameters STMT, LABEL and BRANCH

5.1.1.2 Macro Body -

The macro body consists of a series of statements. Within these statements, in any field, may appear a substitutable parameter as defined in the macro heading. To be recognized as such, the parameter must be bounded on both sides by a character other than a letter or digit or by the beginning or end of the line.

The character → used as a parameter delimiter is treated specially. All occurrences of → are deleted from the macro body when the call is made and characters on either side of it appear adjacent in the resulting statement. For example, the macro definition:

```
        XYZ         MACR INDEX
                    LD→INDEX TABLE→INDEX
                    ENDM
```

when called by:

```
                    XYZ  4
```

becomes:

```
                    LD4  TABLE4
```

Comment statements within a macro definition are not reproduced when the macro is called.

Any type of MIXAL statement except END may appear in a macro body. In particular, macro definitions and macro calls may appear in a macro body. Macro definitions occurring in the body of another macro are not defined by MIXAL until the enclosing macro is called the first time. Therefore, the inner macro may not be called until after the outer one has been called.

5.1.1.2.1 The LOC Pseudo-instruction -

Sometimes it is desirable or necessary for a non-substitutable symbol to appear in the location field of a statement of a macro body. If that macro is called more than once in the program, a multiply-defined symbol error will result. The use of MIXAL local symbols can partially remedy this situation except in the case that the macro call lies in the range of a matching pair of references of the same local symbol. The LOC pseudo-instruction remedies this problem completely by allowing a set of symbols to be declared as defined only within the expansion of the macro. The macro can then be used any number of times and each time the symbol(s) will be redefined. The form of the LOC pseudo-instruction statement is:

```
location operation operand

     field     field     field
     --------  --------- -------

     ignored   LOC       a list of symbols separated by commas
```

The LOC pseudo-instruction may occur only within a macro definition.

5.1.1.3 Macro Terminator -

A macro definition is terminated by an ENDM pseudo-instruction statement of the form:

```
     location  operation  operand
     field      field     field
     --------  ---------  -------

     ignored   ENDM       ignored
```

Each ENDM pseudo-instruction encountered must match some MACR pseudo-instruction. Each macro definition must be terminated by its own ENDM.

5.1.2 Macro Call

When a macro is defined, its body of statements is stored by the MIXAL assembler for use when a macro call is made. When the name of a macro appears in the operation field of a MIXAL statement, the saved body of that macro is expanded at that point of the program as though the statements had been individually placed there by the programmer. The macro call statement may contain a symbol in the location field and a set of strings to be substituted for the substitutable parameters of the macro.

If the location field of the macro call statement contains a symbol the effect is as if the symbol had appeared on a ORIG * statement immediately followed by the macro call without the symbol in the location field. For example:

The macro call

```
        LABEL       MAC  X,Y
```

is equivalent to

```
        LABEL       ORIG *
                    MAC  X,Y
```

The operand field of the macro call statement may specify substitutable parameters in either of two ways as described in Sections 5.1.2.2 and 5.1.2.3 below. In either case, the strings to be substituted must follow the rules outlined in the next Section below.

5.1.2.1 Parameter Strings -

Parameter strings appearing in the operand field of macro call statements are just that; they are arbitrary strings which are substituted as a whole for the substitutable parameters. They are not interpreted in any way by the assembler at the time of the macro call, but will be interpreted in whatever way is appropriate when any statement they are substituted into is processed. If all is to be well, the result of substitution must be a set of legal MIXAL statements.

Essentially any sequence of characters may be used as a parameter string except it may not contain commas or blanks since commas are used to separate parameter strings in the operand field and a blank terminates the operand field.

If, as is often the case, it is desirable or necessary for a parameter string to contain commas or blanks, such a string may be used if it is enclosed in parentheses. When scanning the operand field of a macro call statement the assembler assumes that when the first character of a parameter is a left parenthesis all characters between it and the first matching right parenthesis are part of the parameter string. The parentheses themselves are not considered to be part of the string, and do not appear when substitution occurs.

5.1.2.1.1 Examples Of Parameter Strings -

legal	illegal
ABC	A,BC
1234	12 34
++--	, ,
(A B C)	(A B) C
(X,Y)	(X,Y))
(12,(Z,(345)))	((PDQ-456)

5.1.2.2 Positional Parameter Substitution -

One way to specify the correspondence between parameter strings and the substitutable parameters of the macro is positional. The substitutable parameters of the macro are defined in a particular order when the macro is defined. The macro call statement may give a set of parameter strings separated by commas in the operand field. These parameter strings are then substituted in order for the corresponding substitutable parameter of the macro. If no parameter string is to be specified for a particular parameter, then adjacent commas appear where that parameter string would normally be. These commas may be omitted from the right. Any substitutable parameter for which no parameter string is supplied will be replaced by its default value given in the macro definition.

For example, the macro defined by

```
XYZ         MACR A=ARRAY,B=(INC6 1),C,D=6
            ENTA C
            B
            STA  A,D
            ENDM
```

when called by

```
            XYZ  TABLE,(DEC5 2),,5
```

will expand as

```
            ENTA
            DEC5 2
            STA  TABLE,5
```

and when called by

```
            XYZ  TABLE2
```

will expand as

```
            ENTA
            INC6 1
            STA  TABLE2,6
```

This positional method of parameter specification may not be intermixed with the keyword method described in the next Section.

5.1.2.3 Keyword Parameter Substitution -

The second method available for parameter correspondence specification is the use of keywords. Each substitutable parameter of the macro is a symbol, or a keyword. In this method, the parameters which are to be substituted are given with the parameter string in the form

```
    keyword=parameter string
```

Several such parameter specifications may appear in the operand field of the macro call statement separated by commas. Since each specification gives the keyword, it is unnecessary for them to be in any particular order, and substitutable parameters which are to have their default values are simply not mentioned.

The two example macro calls shown in the previous Section may also be given by (respectively):

```
            XYZ  D=5,A=TABLE,B=(DEC5 2)
```

and

```
            XYZ  A=TABLE2
```

and will give exactly the same macro expansion in each case as was produced in the previous example.

This method of parameter specification may not be intermixed with the positional method previously described.

5.1.3 Operation Code Recognition

The MIXAL assembler keeps a table of all defined macro names and a table of all MIXAL pseudo-instructions and machine operation mnemonics. When processing the operation field of a MIXAL statement, the assembler always searches the table of macro names first. If the symbol in the operation field appears in that table, the statement is treated as a macro call. If the symbol is not in the table of macro names, the table of pseudo-instruction and machine operation mnemonics is searched. If the symbol is in that table,the statement is processed accordingly. Only if the symbol does not appear in either table is an illegal operation error message given.

5.2 CONDITIONAL ASSEMBLY

There are several pseudo-instructions that fall in the category of conditional assembly. They provide the programmer with the capability to detect certain conditions during the assembly process and optionally assemble or not assemble sequences of statements in response to those conditions. These pseudo-instructions are most often used within macro bodies to control the detailed expansion of the macro, but there is no requirement that they be used only within macros.

5.2.1 IF

Format:

```
location  operation  operand
field      field     field
--------  ---------  -------

ignored   IF         relation,expression-1,expression-2
```

The two expressions in the operand field are evaluated and their values are compared according to the specified relation. The relation may be chosen from the following table.

Relation	meaning
EQ	equal
NE	not equal
LT	less than
LE	less than or equal to
GT	greater than
GE	greater than or equal to

If "value-1 relation value-2" is true the statements following the IF pseudo-instruction up to the matching ELSE or ENDI statement will be assembled. If the relation is false those statements will be skipped by the assembler and will not even appear on the listing.

The idea of matching ELSE or ENDI statement is based on nesting of conditional assembly statements. If the assembler is skipping statements, it will skip other conditional assembly statements and their corresponding ELSE and ENDI statements until it sees the ELSE or ENDI which goes with the conditional assembly pseudo-instruction which caused the skipping to commence.

5.2.2 IFC

Format:

```
location  operation  operand
field      field     field
--------  ---------  -------

ignored   IFC        relation,/string-1/string-2/
```

The IFC pseudo-instruction compares two strings of characters. The relation may be one of EQ (for equal) or NE (for not equal). "/" stands for a character used to delimit the two strings, and it may be any character not contained in either string-1 or string-2.

If the two strings satisfy the relation then the statements following the IFC pseudo-instruction up to the matching ELSE or ENDI statement will be assembled. If the relation is not satisfied those statements will be skipped and will not even appear in the listing.

5.2.3 IFD

Format:

```
location  operation  operand
field      field     field
--------  ---------  -------

ignored   IFD        symbol
```

The IFD pseudo-instruction checks the definition status of the symbol in the operand field. If the symbol is defined already the statements following the IFD pseudo-instruction up to the matching ELSE or ENDI pseudo-instruction will be assembled. If the symbol is not already defined, those statements will be skipped and will not even appear in the listing.

5.2.4 ELSE

Format:

```
location  operation  operand
field      field     field
--------  ---------  -------

ignored   ELSE       ignored
```

The ELSE pseudo-instruction may appear only between a matching IF, IFC, or IFD and ENDI pair. If the assembler is assembling statements when the ELSE is encountered, then it will skip the following statements up to the matching ENDI statement. If the assembler is skipping statements when the else is encountered, then it will assemble the statements following.

5.2.5 ENDI

Format:

```
location  operation  operand
field      field     field
--------  ---------  -------

ignored   ENDI       ignored
```

The ENDI pseudo-instruction merely delimits the scope of the IF, IFC, or IFD pseudo-instruction it matches. Every ENDI must match some IF, IFC, or IFD, and conversely every IF, IFC, or IFD must have a matching ENDI.

CHAPTER 6

THE UT-MIX INPUT/OUTPUT SUBSYSTEM

UT-MIX provides a complete input/output subsystem simulation, with several capabilities beyond those proposed by Knuth. The MIX devices are described briefly in Chapter 2. This Chapter provides complete specifications for the performance of each device.

6.1 THE SIMULATED MIX I/O UNITS

UNITS	SIMULATED TYPE	MODES OF OPERATION	RECORD SIZE
0,1	magnetic tapes	series, binary I/O	100 MIX words
8,9	magnetic disks	random, binary I/O	100 MIX words
10	magnetic drum	random, binary I/O	100 MIX words
16	card reader	serial, alpha input	80 characters
17	card punch	serial, alpha output	80 characters
18	line printer	serial, alpha output	120 characters

6.2 CDC 6000-SERIES FILES

FILE NAME	MIX UNIT AND MODE	RECORD SIZE
TAPE0	0, binary I/O	100 CDC words per MIX record
TAPE1	1, binary I/O	100 CDC words per MIX record
DISK	8,9 and 10 combined	100 CDC words per MIX record
INPUT	16, alpha or binary	80 characters (alpha) per card
OUTPUT	18, alpha output	120 characters per printed line
PUNCH	17, alpha output	80 characters per card

6.3 DATA TRANSFER

UT-MIX handles all character conversion, packing and unpacking operations necessary for data transfers between the CDC files and a running MIX program.

6.3.1 Alpha (character) Input

The UT-MIX character set is slightly different from the set specified for MIX by Knuth, and the CDC display code values differ significantly from the MIX character values. The simulator accepts the punch characters shown in Appendix C, converting them to the numeric values shown. All other punch codes are converted to 00 (blank). Characters read from the input file are packed into MIX memory, five characters to each MIX word, with the sign of each word set to +.

6.3.2 Alpha (character) Output

The simulator prints or punches the characters shown in Appendix C. All other six-bit codes are printed (punched) as blanks. Five characters are transferred from each word in MIX memory to the output or punch file. The signs of affected words are ignored.

6.3.3 Binary Input And Output

Binary data transfer involves full (60-bit) CDC words, with the MIX value occupying the lower 36 bits of each word. The sign of a MIX word occupies the first six bits of the 36-bit field. (A sign byte value from 00 to 37 octal is +. A sign byte value from 40 octal to 77 octal is -.)

6.4 UT-MIX I/O INSTRUCTIONS

The reader is urged to review pages 17-19 of reference 1, and if possible, pages 132-134, 211-221 of reference 2 to achieve a complete understanding of I/O operations in MIX. Each I/O operation is completed in several timed steps, normally independent of central processor (CPU) activity once it has been started. In particular, a MIX program must be sure that the data transfer initiated by an I/O instruction is complete before further load or store operations refer to the affected area (buffer) within MIX memory.

6.4.1 The IN And OUT Instructions

The instruction is decoded, and its elements (memory address, unit number, and operation) are checked for validity. Any error will cause the simulator to stop immediately with an error message.

If a previous operation is still in progress on the unit, CPU activity is stopped until data transfer for that operation is complete.

An initial (access-time) delay is then started for the unit, and the CPU is permitted to continue with its next instruction.

At intervals thereafter, the CPU is interrupted for one memory cycle (one microsecond) while one word is transferred between the specified device and MIX memory. This process continues until all words have been transferred.

A final delay may then be imposed for the unit. At the end of this period, an internal register is set to indicate that the unit is ready. (This register is tested by the JBUS and JRED instructions; see below). The result of an input operation (IN instruction) is also recorded in memory location 4002 + n (n = unit number). The result codes are specified in Section 6.5, Exceptional Conditions.

6.4.2 The IOC Instruction

The IOC instruction permits certain special operations for the various I/O devices under control of the MIX program. The UT-MIX implementation of the IOC instruction is significantly different from the Knuth specifications, and its effect and timing varies depending on unit activity. The effect of this instruction for each unit is specified in Section 6.6, Input/Output Control.

The IOC instruction is decoded and checked for validity in all cases and, if the specified unit is busy, CPU activity ceases until the unit is ready. The desired control operation is then started, and the CPU resumed normal activity. The affected unit may remain busy for some time, depending on the instruction.

6.4.3 The JBUS Instruction

This instruction permits the MIX program to test the busy or ready status of a unit. Two forms are allowed, and produce different results.

```
JBUS *(unit)
```

This form stops the CPU if the unit is busy, and completes the I/O operation in progress on the unit. The CPU then proceeds normally with the next instruction.

```
JBUS to other addresses (unit)
```

This form of the instruction functions like any other conditional jump. If the specified unit is busy, a jump occurs; otherwise, processing continues with the next instruction. Activity on the I/O unit, if any, is unaffected.

6.4.4 The JRED Instruction

This instruction functions like any other conditional jump. If the specified unit is ready, a jump occurs; otherwise, processing continues with the next instruction. I/O activity is unaffected. JRED may not specify its own address.

6.5 EXCEPTIONAL CONDITIONS

One word in MIX memory is associated with each I/O unit to return an indication to the using program of the status of the unit after each I/O operation. The status-reply word associated with I/O unit n is at location 4002 + n. The simulator sets a non-zero value in a status-reply word if an exceptional condition is detected for the associated unit. If no exceptional condition is detected, the word is cleared to zero.

6.5.1 End Of Record

The card reader (unit 16) may sense an end-of-record (punch code 7/8/9) during input operations. A positive value will be placed in MIX memory word 4018. No data is transferred from the device, and the specified memory (buffer) area is unchanged.

6.5.2 End Of File

The card reader or magnetic tape units (units 0,1) may sense an end-of-file (punch code 6/7/8/9) during input operations. A negative value will be set in the associated status-reply word. An IOC instruction may be used to backspace or rewind a tape to clear the condition. The card reader cannot be rewound.

6.5.3 Trash Disk And Drum Records

If a program reads a record from one of the random-access devices (disk or drum), and if the record at that disk or drum address was not previously written by the program, the simulator will place the character string "THIS IS TRASH" in the specified memory (buffer) area. The exceptional-condition cells are not set in this case.

6.6 INPUT/OUTPUT CONTROL (IOC)

The interpretation of the IOC instruction is given below for each unit.

6.6.1 Magnetic Tape -- Units 0,1.

An end-of-file condition, if one exists, is cleared.

If M = 0, the tape is rewound.

If M < 0, the tape is backspaced -M records or to the beginning of the first record.

If M > 0, the tape is skipped forward M records or to the end-of-file mark, whichever occurs first. If end-of-file is encountered, the status-reply word is set to a negative value.

6.6.2 Magnetic Disks -- Units 8,9.

A seek operation is started to move the head to a new track, potentially saving some time when the next IN or OUT instruction is issued. Byte (4:4) of the X register is used as the track index. M is ignored. The seek operation requires 32 milliseconds per track.

6.6.3 Magnetic Drum -- Unit 10.

The IOC instruction has no effect on the drum and is ignored.

6.6.4 Card Reader -- Unit 16.

An IOC 0 instruction will clear an end-of-record setting in the status-reply word (location 4018).

6.6.5 Card Punch -- Unit 17

The IOC instruction should not be used with the card punch.

6.6.6 Line Printer -- Unit 18

The IOC instruction provides carriage control. No data is transferred from memory.

If M = 0, the printer skips to the top of the next page.

If M > 0, the printer skips M (modulo 64) lines, leaving them blank.

6.6.7 Multiple-record Input (card) Files

The IOC instruction permits a MIX program to use a card input file which consists of several CDC logical records separated by end-of-record markers (7/8/9 punch code). When an end-of-record condition is sensed, the status-reply cell 4018 will be set to a positive value, and no data are transferred. The next IN instruction will read the first card from the next logical record (following the 7/8/9 card) into memory. IOC may be used to reset the status-reply cell. End-of-file (6/7/8/9 punch code) cannot be reset, and any further in instructions will cause simulation to stop with an appropriate error message.

6.7 DISPOSITION OF I/O FILES

All I/O activity is allowed to complete within the simulator before a MIX program is terminated, regardless of the reason for termination. In the case of an error finish, this preserves the last line of printer output (and the last card punched) for the programmer's inspection. The last line is printed before the error termination message.

Final disposition of the CDC files produced is governed by the sequence of cards in the control deck which refer to them, if any. Files are not rewound by the UT-MIX simulator. All files may be printed, punched, dumped or released using standard CDC utility routines. The TAPE0 and TAPE1 files may be used by any 6000-series language program as ordinary serial binary records. The disk file may contain randomly-produced records for the three MIX units (8,9,10) intermixed and does not include a directory -- it is therefore of little value for further use.

6.8 FATAL I/O ERRORS

The following illegal I/O operations will cause simulation to stop with an appropriate error message.

6.8.1 Nonexistent Unit

An invalid unit number in the F-field of an I/O instruction. The current valid unit numbers are given in Section 6.1.

6.8.2 Illegal I/O Operation

--an attempt to write on a card reader, or to read the printer or punch.

--an attempt to read a tape unit, if any records have been written on the unit, before a backspace or rewind IOC instruction has been issued.

--an attempt to read a tape unit or card after end-of-file has been detected, if an IOC instruction has not been issued to clear the condition.

--an attempt to access a non-existent disk address (the X register is negative, or exceeds 4095).

APPENDIX A

SIMULATOR TRACE FEATURE

The user may turn on the trace feature by placing a nonzero value in MIX memory cell 4000 (decimal), and turn off the trace by storing a zero (STZ) in the cell. As long as memory cell 4000 contains a nonzero value, the execution of each instruction will cause one line to print on the output file showing the condition of the machine. The line has the following format:

```
P = a   IN = b   OT = c   CI = d   A = b  X = b   J = e   I1 = e ... I6 = e
```

a = unsigned 4-digit octal number, giving the P-register value (the location of the instruction just executed).

b = 10-digit signed octal number, to be interpreted as five bytes plus sign. This format is used to show the instruction just executed (IN), and the contents of the A and X registers after execution.

c = 0 or 1, showing the overflow toggle. 1 means overflow.

d = 0, -1, or +1 showing the comparison indicator. 0 means equal, -1 means less than, +1 means greater than.

e = 4-digit signed octal number, to be interpreted as two bytes plus sign. This format is used for the J, I1, I2, ..., I6 registers.

APPENDIX B

FATAL SIMULATOR ERRORS

Since MIX permits modification of any instruction or a jump to any legal address, many fatal errors detected by UT-MIX are the result of improper store operations which have destroyed program code or an unintended jump to an address containing data or garbage instead of a program instruction. Careful examination of the trace output accompanying the fatal message will help to isolate the problem.

UT-MIX checks all instructions for correct format and legal addresses. It also checks to see if a jump instruction results in a jump to itself (other than a JBUS) and detects these sorts of infinite loops. UT-MIX makes no attempt, however, to trap an infinite loop if it executes more than one instruction -- in this case, a time limit dump will occur.

All UT-MIX fatal error messages begin with the phrase **** EXECUTION STOPPED. The phrase which follows indicates the type of error detected. In all cases, simulation stops before any registers or memory values are updated. The one-line register dump accompanying the message will show the guilty instruction, its address, and all registers as they were before the instruction was executed.

B.1 ILLEGAL ADDRESS FIELD.

During instruction decoding, the value contained in the index register specified by the I-part of the instruction was added algebraically to the AA-field extracted by the instruction. The resulting absolute value exceeded 4095, and is thus an illegal M-value.

B.2 ILLEGAL ADDRESS FOR JUMP.

The M-value (see above) was either negative or exceeded 4021 at the time a jump instruction was to be executed. If the instruction was a conditional jump, the required condition was satisfied.

B.3 ILLEGAL ADDRESS FOR MOVE.

Either the M-value or the value of the I1 register was negative, or one of the values exceeded 4021 when added to the F-value of the instruction. In any event, the MOVE process would have accessed an invalid MIX memory location.

B.4 ILLEGAL INDEX SPECIFICATION.

During instruction decoding, the I-field of the instruction was found to contain the value 77 octal, specifying double indirect addressing.

B.5 ILLEGAL INDEX REGISTER LOAD.

A load or address transfer operation (ENTi, INCi, DECi, ENNi) generated an absolute value larger than 4095 to be placed in an index register.

B.6 ILLEGAL MEMORY REFERENCE.

The M-value associated with a memory-referencing instruction was either negative or exceeded the upper bound for the instruction. For a MOVE instruction, the upper bound is (4022-F-value). For I/O instructions, the upper bound is (4022 - record size). For all other instructions, the upper bound is 4021.

B.7 ILLEGAL (SAME ADDRESS) JUMP.

The M-value was the same as the address of a jump instruction at the time the instruction was to be executed. If the instruction was a conditional jump, the condition was satisfied. (Note: this restriction is designed primarily to kill non-productive infinite loops. It can be used to advantage, however, during debugging if the user wishes to stop simulation immediately upon occurrence of an event which can be tested by a conditional jump -- e.g., JAZ *).

B.8 ILLEGAL DISK ADDRESS.

The value of the X-register exceeded 4095 at the time an IN, OUT or IOC instruction was issued for unit 8, 9, or 10.

B.9 ILLEGAL I/O OPERATION.

An IN instruction has been issued for card punch or printer, or an OUT instruction has been issued for the card reader, or an IN instruction immediately follows an OUT instruction on a tape unit.

B.10 ILLEGAL I/O AFTER END OF FILE.

An end-of-file condition was detected for a magnetic tape or card unit, and a subsequent IN instruction has been issued. (For tape unit end-of-file handling, see Section 6.6.1).

B.11 F-VALUE ERRORS

The F-value specifies the type of shift, jump, special, address transfer or miscellaneous instruction to be executed, and the I/O unit for I/O instructions. It is the first value checked when one of these instructions (denoted by the C-value) is to be executed. For most other instructions, the F-value must satisfy certain fairly strict restrictions. As a result, any one of the following errors may frequently be reported when an attempt has been made by a MIX program to execute data or garbage through an unintentional jump to a wrong (but legal) address, or improper use of a store instruction which has destroyed program code.

B.11.1 Illegal Field Specification.

The instruction references memory, and contains an F-value illegal for the operation being performed. Only the MOVE instruction may have an F-value exceeding 45 (55 octal). All others require $L \leq R$ -- the first octal digit of the F-value must be less than or equal to the second.

B.11.2 Illegal Special Instruction.

$C = 5$, and the F-value specifies a non-existent instruction type. The present simulator accepts $0 \leq F \leq 10$ (decimal).

B.11.3 Illegal Shift Type.

$C = 6$, and the F-value specifies a non-existent shift type. The present simulator accepts $0 \leq F \leq 7$.

B.11.4 Illegal Jump Type.

$C = 39$ and $F > 9$, or $40 \leq C \leq 47$ (decimal) and $F > 7$.

B.11.5 Illegal Address Transfer Type.

$48 \leq C \leq 55$ (decimal), and $F > 3$.

B.12 TRACE TERMINATION

To conserve time and printer paper, the simulator limits trace output to a total of 500 lines (approximately 10 pages) for a single run. The trace feature of UT-MIX is valuable if used sparingly and analyzed carefully. If an attempt is made to produce more than 500 lines of trace output, the simulator terminates with the message EXECUTION STOPPED -- EXCESSIVE TRACE OUTPUT. This message does not indicate any error in the MIX program code.

APPENDIX C

THE UT-MIX CHARACTER SET

Any punched character not included in the set below will be read as a blank (00). Any numeric code not included in the set below will be printed as a blank.

code			code			code		
dec.	octal	char.	dec.	octal	char.	dec.	octal	char.
00	00	blank	21	25	^	42	52	(
01	01	A	22	26	S	43	53	)
02	02	B	23	27	T	44	54	+
03	03	C	24	30	U	45	55	-
04	04	D	25	31	V	46	56	*
05	05	E	26	32	W	47	57	/
06	06	F	27	33	X	48	60	=
07	07	G	28	34	Y	49	61	$
08	10	H	29	35	Z	50	62	<
09	11	I	30	36	0	51	63	>
10	12	%	31	37	1	52	64	→
11	13	J	32	40	2	53	65	;
12	14	K	33	41	3	54	66	:
13	15	L	34	42	4	55	67	↑
14	16	M	35	43	5	56	70	[
15	17	N	36	44	6	57	71	∨

16	20	O	37	45	7	58	72	]
17	21	P	38	46	8	59	73	≠
18	22	Q	39	47	9	60	73	≤
19	23	R	40	50	.	61	74	¬
20	24	≡	41	51	,	62	75	≥

APPENDIX D

MIX SYMBOLIC OPCODES - ALPHABETIC ORDER

Notation:

M is the computed effective address
(M) is the contents of location M
* in the field specification means L:R

code	field	symbol	instruction
01	*	ADD	add (M) to register A
03	07	AND	logical and (M) into A
05	01	CHAR	A is converted to 10-byte decimal characters in AX
05	06	CHS	change the sign of A
70	*	CMPA	compare A and (M), set comparison indicator
77	*	CMPX	compare X and (M), set comparison indicator
71	*	CMP1	compare I1 and (M), set comparison indicator
72	*	CMP2	compare I2 and (M), set comparison indicator
73	*	CMP3	compare I3 and (M), set comparison indicator
74	*	CMP4	compare I4 and (M), set comparison indicator
75	*	CMP5	compare I5 and (M), set comparison indicator
76	*	CMP6	compare I6 and (M), set comparison indicator
60	01	DECA	decrement A by M
67	01	DECX	decrement X by M
61	01	DEC1	decrement I1 by M
62	01	DEC2	decrement I2 by M
63	01	DEC3	decrement I3 by M
64	01	DEC4	decrement I4 by M
65	01	DEC5	decrement I5 by M
66	01	DEC6	decrement I6 by M
04	*	DIV	divide (M) into AX giving A (quotient) and X (remainder)
60	03	ENNA	enter negative of M into A
67	03	ENNX	enter negative of M into X
61	03	ENN1	enter negative of M into I1
62	03	ENN2	enter negative of M into I2
63	03	ENN3	enter negative of M into I3
64	03	ENN4	enter negative of M into I4
65	03	ENN5	enter negative of M into I5
66	03	ENN6	enter negative of M into I6
60	02	ENTA	enter M into A
67	02	ENTX	enter M into X
61	02	ENT1	enter M into I1
62	02	ENT2	enter M into I2

63	02	ENT3	enter M into I3
64	02	ENT4	enter M into I4
65	02	ENT5	enter M into I5
66	02	ENT6	enter M into I6
05	02	HLT	halt the MIX machine
44	N	IN	start input transfer from unit N
60	00	INCA	increment A by M
67	00	INCX	increment X by M
61	00	INC1	increment I1 by M
62	00	INC2	increment I2 by M
63	00	INC3	increment I3 by M
64	00	INC4	increment I4 by M
65	00	INC5	increment I5 by M
66	00	INC6	increment I6 by M
43	N	IOC	issue I/O control signal to unit N
50	06	JAE	jump to M if A is even
50	00	JAN	jump to M if A is negative
50	03	JANN	jump to M if A is non-negative
50	05	JANP	jump to M if A is non-positive
50	04	JANZ	jump to M if A is non-zero
50	07	JAO	jump to M if A is odd
50	02	JAP	jump to M if A is positive
50	01	JAZ	jump to M if A is zero
42	N	JBUS	jump to location M if unit N is busy
47	05	JE	jump to M if comparison indicator is equal
47	06	JG	jump to M if comparison indicator is greater
47	07	JGE	jump to M if comparison indicator is greater or equal
47	04	JL	jump to M if comparison indicator is less
47	11	JLE	jump to M if comparison indicator is less or equal
47	00	JMP	jump to M
47	10	JNE	jump to M if comparison indicator is not equal
47	03	JNOV	jump to M if overflow off, turn overflow off anyway
47	02	JOV	jump to M if overflow on, turn overflow off
46	N	JRED	jump to location M if unit N is ready
47	01	JSJ	jump to M (but do not change register J)
57	06	JXE	jump to M if X is even
57	00	JXN	jump to M if X is negative
57	03	JXNN	jump to M if X is non-negative
57	05	JXNP	jump to M if X is non-positive
57	04	JXNZ	jump to M if X is non-zero
57	07	JXO	jump to M if X is odd
57	02	JXP	jump to M if X is positive
57	01	JXZ	jump to M if X is zero
51	06	J1E	jump to M if I1 is even
51	00	J1N	jump to M if I1 is negative
51	03	J1NN	jump to M if I1 is non-negative
51	05	J1NP	jump to M if I1 is non-positive
51	04	J1NZ	jump to M if I1 is non-zero
51	07	J1O	jump to M if I1 is odd
51	02	J1P	jump to M if I1 is positive
51	01	J1Z	jump to M if I1 is zero
52	06	J2E	jump to M if I2 is even
52	00	J2N	jump to M if I2 is negative
52	03	J2NN	jump to M if I2 is non-negative
52	05	J2NP	jump to M if I2 is non-positive

52	04	J2NZ	jump to M if I2 is non-zero
52	07	J2O	jump to M if I2 is odd
52	02	J2P	jump to M if I2 is positive
52	01	J2Z	jump to M if I2 is zero
53	06	J3E	jump to M if I3 is even
53	00	J3N	jump to M if I3 is negative
53	03	J3NN	jump to M if I3 is non-negative
53	05	J3NP	jump to M if I3 is non-positive
53	04	J3NZ	jump to M if I3 is non-zero
53	07	J3O	jump to M if I3 is odd
53	02	J3P	jump to M if I3 is positive
53	01	J3Z	jump to M if I3 is zero
54	06	J4E	jump to M if I4 is even
54	00	J4N	jump to M if I4 is negative
54	03	J4NN	jump to M if I4 is non-negative
54	05	J4NP	jump to M if I4 is non-positive
54	04	J4NZ	jump to M if I4 is non-zero
54	07	J4O	jump to M if I4 is odd
54	02	J4P	jump to M if I4 is positive
54	01	J4Z	jump to M if I4 is zero
55	06	J5E	jump to M if I5 is even
55	00	J5N	jump to M if I5 is negative
55	03	J5NN	jump to M if I5 is non-negative
55	05	J5NP	jump to M if I5 is non-positive
55	04	J5NZ	jump to M if I5 is non-zero
55	07	J5O	jump to M if I5 is odd
55	02	J5P	jump to M if I5 is positive
55	01	J5Z	jump to M if I5 is zero
56	06	J6E	jump to M if I6 is even
56	00	J6N	jump to M if I6 is negative
56	03	J6NN	jump to M if I6 is non-negative
56	05	J6NP	jump to M if I6 is non-positive
56	04	J6NZ	jump to M if I6 is non-zero
56	07	J6O	jump to M if I6 is odd
56	02	J6P	jump to M if I6 is positive
56	01	J6Z	jump to M if I6 is zero
10	*	LDA	load A with (M)
20	*	LDAN	load A with negative of (M)
17	*	LDX	load X with (M)
27	*	LDXN	load X with negative of (M)
11	*	LD1	load I1 with (M)
21	*	LD1N	load I1 with negative of (M)
12	*	LD2	load I2 with (M)
22	*	LD2N	load I2 with negative of (M)
13	*	LD3	load I3 with (M)
23	*	LD3N	load I3 with negative of (M)
14	*	LD4	load I4 with (M)
24	*	LD4N	load I4 with negative of (M)
15	*	LD5	load I5 with (M)
25	*	LD5N	load I5 with negative of (M)
16	*	LD6	load I6 with (M)
26	*	LD6N	load I6 with negative of (M)
05	10	LNG	complement (bitwise) bytes 1 to 5 of A
07	N	MOVE	move N words starting from M to (I1), add N to I1
05	12	MSK	create A mask of M bits in A (+ = left-justified, - = right)

03	*	MUL	multiply (M) by A giving AX
00	00	NOP	no operation
05	00	NUM	10-byte decimal in AX converted to binary in A
05	03	OCT	A is converted to 10-byte octal characters in AX
01	07	OR	inclusive or of (M) with A
45	N	OUT	start output transfer from unit N
06	00	SLA	shift A M bytes (bits if M negative) left, end-off
06	02	SLAX	shift AX M bytes (bits if M negative) left, end-off
06	06	SLB	shift AX M bits left, end-off
06	04	SLC	shift AX M bytes (bits if M negative) left, circular
06	01	SRA	shift A M bytes (bits if M negative) right, end-off
06	03	SRAX	shift AX M bytes (bits if M negative) right, end-off
06	07	SRB	shift AX M bits right, end-off
06	05	SRC	shift AX M bytes (bits if M negative) right, circular
05	05	SSN	set sign of A negative
05	04	SSP	set sign of A positive
30	*	STA	store A into location M
40	*	STJ	store J register into location M
37	*	STX	store X into location M
41	*	STZ	store zero into location M
31	*	ST1	store I1 into location M
32	*	ST2	store I2 into location M
33	*	ST3	store I3 into location M
34	*	ST4	store I4 into location M
35	*	ST5	store I5 into location M
36	*	ST6	store I6 into location M
02	*	SUB	subtract (M) from A
05	11	XCH	exchange registers A and X
02	07	XOR	exclusive or of (M) with A

APPENDIX E

MIX SYMBOLIC OPCODES - NUMERIC ORDER

Notation:

M is the computed effective address
(M) is the contents of location M
* in the field specification means L:R

code	field	symbol	instruction
00	00	NOP	no operation
01	*	ADD	add (M) to register A
01	07	OR	inclusive or of (M) with A
02	*	SUB	subtract (M) from A
02	07	XOR	exclusive or of (M) with A
03	*	MUL	multiply (M) by A giving AX
03	07	AND	logical and (M) into A
04	*	DIV	divide (M) into AX giving A (quotient) and X (remainder)
05	00	NUM	10-byte decimal in AX converted to binary in A
05	01	CHAR	A is converted to 10-byte decimal characters in AX
05	02	HLT	halt the MIX machine
05	03	OCT	A is converted to 10-byte octal characters in AX
05	04	SSP	set sign of A positive
05	05	SSN	set sign of A negative
05	06	CHS	change the sign of A
05	10	LNG	complement (bitwise) bytes 1 to 5 of A
05	11	XCH	exchange registers A and X
05	12	MSK	create A mask of M bits in A (+ = left-justified, - = right)
06	00	SLA	shift A M bytes (bits if M negative) left, end-off
06	01	SRA	shift A M bytes (bits if M negative) right, end-off
06	02	SLAX	shift AX M bytes (bits if M negative) left, end-off
06	03	SRAX	shift AX M bytes (bits if M negative) right, end-off
06	04	SLC	shift AX M bytes (bits if M negative) left, circular
06	05	SRC	shift AX M bytes (bits if M negative) right, circular
06	06	SLB	shift AX M bits left, end-off
06	07	SRB	shift AX M bits right, end-off
07	N	MOVE	move N words starting from M to (I1), add N to I1
10	*	LDA	load A with (M)
11	*	LD1	load I1 with (M)
12	*	LD2	load I2 with (M)
13	*	LD3	load I3 with (M)
14	*	LD4	load I4 with (M)
15	*	LD5	load I5 with (M)

16	*	LD6	load I6 with (M)
17	*	LDX	load X with (M)
20	*	LDAN	load A with negative of (M)
21	*	LD1N	load I1 with negative of (M)
22	*	LD2N	load I2 with negative of (M)
23	*	LD3N	load I3 with negative of (M)
24	*	LD4N	load I4 with negative of (M)
25	*	LD5N	load I5 with negative of (M)
26	*	LD6N	load I6 with negative of (M)
27	*	LDXN	load X with negative of (M)
30	*	STA	store A into location M
31	*	ST1	store I1 into location M
32	*	ST2	store I2 into location M
33	*	ST3	store I3 into location M
34	*	ST4	store I4 into location M
35	*	ST5	store I5 into location M
36	*	ST6	store I6 into location M
37	*	STX	store X into location M
40	*	STJ	store J register into location M
41	*	STZ	store zero into location M
42	N	JBUS	jump to location M if unit N is busy
43	N	IOC	issue I/O control signal to unit N
44	N	IN	start input transfer from unit N
45	N	OUT	start output transfer from unit N
46	N	JRED	jump to location M if unit N is ready
47	00	JMP	jump to M
47	01	JSJ	jump to M (but do not change register J)
47	02	JOV	jump to M if overflow on, turn overflow off
47	03	JNOV	jump to M if overflow off, turn overflow off anyway
47	04	JL	jump to M if comparison indicator is less
47	05	JE	jump to M if comparison indicator is equal
47	06	JG	jump to M if comparison indicator is greater
47	07	JGE	jump to M if comparison indicator is greater or equal
47	10	JNE	jump to M if comparison indicator is not equal
47	11	JLE	jump to M if comparison indicator is less or equal
50	00	JAN	jump to M if A is negative
50	01	JAZ	jump to M if A is zero
50	02	JAP	jump to M if A is positive
50	03	JANN	jump to M if A is non-negative
50	04	JANZ	jump to M if A is non-zero
50	05	JANP	jump to M if A is non-positive
50	06	JAE	jump to M if A is even
50	07	JAO	jump to M if A is odd
51	00	J1N	jump to M if I1 is negative
51	01	J1Z	jump to M if I1 is zero
51	02	J1P	jump to M if I1 is positive
51	03	J1NN	jump to M if I1 is non-negative
51	04	J1NZ	jump to M if I1 is non-zero
51	05	J1NP	jump to M if I1 is non-positive
51	06	J1E	jump to M if I1 is even
51	07	J1O	jump to M if I1 is odd
52	00	J2N	jump to M if I2 is negative
52	01	J2Z	jump to M if I2 is zero
52	02	J2P	jump to M if I2 is positive
52	03	J2NN	jump to M if I2 is non-negative

52	04	J2NZ	jump to M if I2 is non-zero
52	05	J2NP	jump to M if I2 is non-positive
52	06	J2E	jump to M if I2 is even
52	07	J2O	jump to M if I2 is odd
53	00	J3N	jump to M if I3 is negative
53	01	J3Z	jump to M if I3 is zero
53	02	J3P	jump to M if I3 is positive
53	03	J3NN	jump to M if I3 is non-negative
53	04	J3NZ	jump to M if I3 is non-zero
53	05	J3NP	jump to M if I3 is non-positive
53	06	J3E	jump to M if I3 is even
53	07	J3O	jump to M if I3 is odd
54	00	J4N	jump to M if I4 is negative
54	01	J4Z	jump to M if I4 is zero
54	02	J4P	jump to M if I4 is positive
54	03	J4NN	jump to M if I4 is non-negative
54	04	J4NZ	jump to M if I4 is non-zero
54	05	J4NP	jump to M if I4 is non-positive
54	06	J4E	jump to M if I4 is even
54	07	J4O	jump to M if I4 is odd
55	00	J5N	jump to M if I5 is negative
55	01	J5Z	jump to M if I5 is zero
55	02	J5P	jump to M if I5 is positive
55	03	J5NN	jump to M if I5 is non-negative
55	04	J5NZ	jump to M if I5 is non-zero
55	05	J5NP	jump to M if I5 is non-positive
55	06	J5E	jump to M if I5 is even
55	07	J5O	jump to M if I5 is odd
56	00	J6N	jump to M if I6 is negative
56	01	J6Z	jump to M if I6 is zero
56	02	J6P	jump to M if I6 is positive
56	03	J6NN	jump to M if I6 is non-negative
56	04	J6NZ	jump to M if I6 is non-zero
56	05	J6NP	jump to M if I6 is non-positive
56	06	J6E	jump to M if I6 is even
56	07	J6O	jump to M if I6 is odd
57	00	JXN	jump to M if X is negative
57	01	JXZ	jump to M if X is zero
57	02	JXP	jump to M if X is positive
57	03	JXNN	jump to M if X is non-negative
57	04	JXNZ	jump to M if X is non-zero
57	05	JXNP	jump to M if X is non-positive
57	06	JXE	jump to M if X is even
57	07	JXO	jump to M if X is odd
60	00	INCA	increment A by M
60	01	DECA	decrement A by M
60	02	ENTA	enter M into A
60	03	ENNA	enter negative of M into A
61	00	INC1	increment I1 by M
61	01	DEC1	decrement I1 by M
61	02	ENT1	enter M into I1
61	03	ENN1	enter negative of M into I1
62	00	INC2	increment I2 by M
62	01	DEC2	decrement I2 by M
62	02	ENT2	enter M into I2

62	03	ENN2	enter negative of M into I2
63	00	INC3	increment I3 by M
63	01	DEC3	decrement I3 by M
63	02	ENT3	enter M into I3
63	03	ENN3	enter negative of M into I3
64	00	INC4	increment I4 by M
64	01	DEC4	decrement I4 by M
64	02	ENT4	enter M into I4
64	03	ENN4	enter negative of M into I4
65	00	INC5	increment I5 by M
65	01	DEC5	decrement I5 by M
65	02	ENT5	enter M into I5
65	03	ENN5	enter negative of M into I5
66	00	INC6	increment I6 by M
66	01	DEC6	decrement I6 by M
66	02	ENT6	enter M into I6
66	03	ENN6	enter negative of M into I6
67	00	INCX	increment X by M
67	01	DECX	decrement X by M
67	02	ENTX	enter M into X
67	03	ENNX	enter negative of M into X
70	*	CMPA	compare A and (M), set comparison indicator
71	*	CMP1	compare I1 and (M), set comparison indicator
72	*	CMP2	compare I2 and (M), set comparison indicator
73	*	CMP3	compare I3 and (M), set comparison indicator
74	*	CMP4	compare I4 and (M), set comparison indicator
75	*	CMP5	compare I5 and (M), set comparison indicator
76	*	CMP6	compare I6 and (M), set comparison indicator
77	*	CMPX	compare X and (M), set comparison indicator