MIX (abstract machine)

From Wikipedia, the free encyclopedia

This article **needs more citations**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.

Find sources: "MIX" abstract machine – news · newspapers · books · scholar · JSTOR

(August 2012) (Learn how and when to remove this message)

MIX is a hypothetical computer used in Donald Knuth's monograph, *The Art of Computer Programming* (*TAOCP*). MIX's model number is 1009, which was derived by combining the model numbers and names of several contemporaneous, commercial machines deemed significant by the author. Also, "MIX" read as a Roman numeral is 1009.

MIX

Designer	Donald Knuth
Bits	31-bit
Introduced	1968
Design	accumulator machine
Type	hypothetical
Encoding	Fixed
Branching	Condition code and register test
Endianness	Big
Open	Yes, and royalty free
Registers	
9 in total	

The 1960s-era MIX has since been superseded by a new (also hypothetical) computer architecture, MMIX, to be incorporated in forthcoming editions of *TAOCP*.

Software implementations for both the MIX and MMIX architectures have been developed by Knuth and made freely available (named "MIXware" and "MMIXware", respectively). Several derivatives of Knuth's MIX/MMIX emulators also exist. GNU MDK is one such software package; it is free and runs on a wide variety of platforms.

Their purpose for education is quite similar to John L. Hennessy's and David A. Patterson's DLX architecture, from *Computer Organization and Design - The Hardware Software Interface*.

Architecture [edit]

MIX is a hybrid binary–decimal computer. When programmed in binary, each byte has 6 bits (values range from 0 to 63). In decimal, each byte has 2 decimal digits (values range from 0 to 99). The two interpretions of a MIX byte (binary vs. decimal) represent different machine types and are not intended to be directly interchangeable:

> "MIX has a peculiar property in that it is both binary and decimal at the same time. MIX programmers don't actually know whether they are programming a machine with base 2 or base 10

arithmetic. Therefore algorithms written in MIX can be used on either type of machine with little change, and MIX can be simulated easily on either type of machine. Programmers who are accustomed to a binary machine can think of MIX as binary; those accustomed to decimal may regard MIX as decimal. Programmers from another planet might choose to think of MIX as a ternary computer."[1]

Bytes are grouped into words of five bytes plus a sign. Most programs written for MIX will work in either binary or decimal, so long as they do not try to store a value greater than 63 in a single byte when interpreted a binary, and 99 when interpreted as decimal.

A word has the range −1,073,741,823 to 1,073,741,823 (inclusive) in binary mode, and −9,999,999,999 to 9,999,999,999 (inclusive) in decimal mode. The sign-and-magnitude representation of integers in the MIX architecture distinguishes between “−0” and “+0.” This contrasts with modern computers, whose two's-complement representation of integer quantities includes a single representation for zero, but whose range for a given number of bits includes one more negative integer than the number of representable positive integers.

Registers [edit]

There are 9 registers in MIX:

- **rA**: Accumulator (full word, five bytes and a sign).
- **rX**: Extension (full word, five bytes and a sign).
- **rI1**, **rI2**, **rI3**, **rI4**, **rI5**, **rI6**: Index registers (two bytes and a sign).
- **rJ**: Jump address (two bytes, always positive).

A byte is assumed to be at least 6 bits. Most instructions can specify *which* of the "fields" (bytes) of a register are to be altered, using a suffix of the form *(first:last)*. The zeroth field is the one-bit sign.

MIX registers

$^{3}_{0}$	$^{2}_{9}$	$^{2}_{8}$	$^{2}_{7}$	$^{2}_{6}$	$^{2}_{5}$	$^{2}_{4}$	$^{2}_{3}$	$^{2}_{2}$	$^{2}_{1}$	$^{2}_{0}$	$^{1}_{9}$	$^{1}_{8}$	$^{1}_{7}$	$^{1}_{6}$	$^{1}_{5}$	$^{1}_{4}$	$^{1}_{3}$	$^{1}_{2}$	$^{1}_{1}$	$^{1}_{0}$	$^{0}_{9}$	$^{0}_{8}$	$^{0}_{7}$	$^{0}_{6}$	$^{0}_{5}$	$^{0}_{4}$	$^{0}_{3}$	$^{0}_{2}$	$^{0}_{1}$	$^{0}_{0}$	*(bit position)*	
Registers																																
±	A1					A2					A3					A4					A5					**rA**, Accumulator							
±	X1					X2					X3					X4					X5					**rX**, Extension							
Index registers																																
																±	I1.4				I1.5					**rI1**, Index 1							
																±	I2.4				I2.5					**rI2**, Index 2							
																±	I3.4				I3.5					**rI3**, Index 3							
																±	I4.4				I4.5					**rI4**, Index 4							
																±	I5.4				I5.5					**rI5**, Index 5							
																±	I6.4				I6.5					**rI6**, Index 6							
Program counter																																
																	J4				J5					**rJ**, Jump							
Condition code flags																																
																										O Overflow flag							
																										<=> Comparison flag							

MIX also records whether the previous operation overflowed, and has a one-trit comparison indicator (less than, equal to, or greater than).

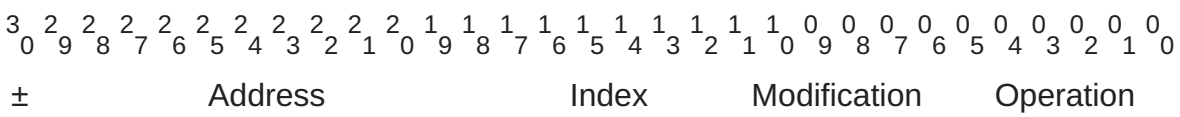
Memory and input/output [edit]

The MIX machine has 4000 words of memory (each with 5 bytes and a sign), addressed from 0 to 3999. A variety of input and output devices are also included:

- Tape units (devices 0...7).
- Disk or drum units (devices 8...15).
- Card reader (device 16).
- Card punch (device 17).
- Line printer (device 18).
- Typewriter terminal (device 19).
- Paper tape (device 20).

Instructions [edit]

Each machine instruction in memory occupies one word, and consists of 4 parts: the address (2 bytes and the sign of the word) in memory to read or write; an index specification (1 byte, describing which rI index register to use) to add to the address; a modification (1 byte) that specifies which parts of the register or memory location will be read or altered; and the operation code (1 byte). All operation codes have an associated mnemonic.



MIX programs frequently use self-modifying code, in particular to return from a subroutine, as MIX lacks an automatic subroutine return stack. Self-modifying code is facilitated by the modification byte, allowing the program to store data to, for example, the address part of the target instruction, leaving the rest of the instruction unmodified.

MIX programs are typically constructed using the MIXAL assembly language; for an example, see the list hello world programs page.

`LDA ADDR,i(0:5)`	`rA := memory[ADDR + rIi];`
`LDX ADDR,i(0:5)`	`rX := memory[ADDR + rIi];`
`LD? ADDR,i(0:5)`	`rI? := memory[ADDR + rIi];`
`LDAN ADDR,i(0:5)`	`rA := - memory[ADDR + rIi];`
`LDXN ADDR,i(0:5)`	`rX := - memory[ADDR + rIi];`
`LD?N ADDR,i(0:5)`	`rI? := - memory[ADDR + rIi];`
`STA ADDR,i(0:5)`	`memory[ADDR + rIi] := rA;`
`STX ADDR,i(0:5)`	`memory[ADDR + rIi] := rX;`
`ST? ADDR,i(0:5)`	`memory[ADDR + rIi] := rI?;`

`STJ ADDR,i(0:5)`	`memory[ADDR + rIi] := rJ;`
`STZ ADDR,i(0:5)`	`memory[ADDR + rIi] := 0;`
`ADD ADDR,i(0:5)`	`rA := rA + memory[ADDR + rIi];`
`SUB ADDR,i(0:5)`	`rA := rA - memory[ADDR + rIi];`
`MUL ADDR,i(0:5)`	`(rA,rX) := rA * memory[ADDR + rIi];`
`DIV ADDR,i(0:5)`	`rA := int( (rA,rX) / memory[ADDR + rIi] );` `rX := (rA,rX) % memory[ADDR + rIi];`
`ENTA ADDR,i`	`rA := ADDR + rIi;`
`ENTX ADDR,i`	`rX := ADDR + rIi;`
`ENT? ADDR,i`	`rI? := ADDR + rIi;`
`ENNA ADDR,i`	`rA := - ADDR - rIi;`
`ENNX ADDR,i`	`rX := - ADDR - rIi;`
`ENN? ADDR,i`	`rI? := - ADDR - rIi;`
`INCA ADDR,i`	`rA := rA + ADDR + rIi;`
`INCX ADDR,i`	`rX := rX + ADDR + rIi;`
`INC? ADDR,i`	`rI? := rI? + ADDR + rIi;`
`DECA ADDR,i`	`rA := rA - ADDR - rIi;`
`DECX ADDR,i`	`rX := rX - ADDR - rIi;`
`DEC? ADDR,i`	`rI? := rI? - ADDR - rIi;`
`CMPA ADDR,i(0:5)`	compare `rA` with `memory[ADDR + rIi]` and set comparison flag;
`CMPX ADDR,i(0:5)`	compare `rX` with `memory[ADDR + rIi]` and set comparison flag;
`CMP? ADDR,i(0:5)`	compare `rI?` with `memory[ADDR + rIi]` and set comparison flag;
`JMP ADDR,i`	`rJ := address of next instruction;` `goto ADDR + rIi;`
`JSJ ADDR,i`	`goto ADDR + rIi;`

`JOV ADDR,i`	`if (overflow) then`
	`  overflow := false;`
	`  goto ADDR + rIi;`

`JNOV ADDR,i`	`if (no overflow) then`
	`  goto ADDR + rIi;`
	`else`
	`  overflow := false;`

`JL, JE, JG ADDR,i` `JGE, JNE, JLE ADDR,i`	`if (less, equal, greater) then goto ADDR + rIi;` `if (no less, unequal, no greater) then goto ADDR + rIi;`
`JAN/JAZ/JAP ADDR,i` `JANN/JANZ/JANP ADDR,i`	`if (rA<0 or rA==0 or rA>0) then goto ADDR + rIi;` `if (rA>=0 or rA!=0 or rA<=0) then goto ADDR + rIi;`
`JXN/JXZ/JXP ADDR,i` `JXNN/JXNZ/JXNP ADDR,i`	`if (rX<0 or rX==0 or rX>0) then goto ADDR + rIi;` `if (rX>=0 or rX!=0 or rX<=0) then goto ADDR + rIi;`
`J?N/J?Z/J?P ADDR,i` `J?NN/J?NZ/J?NP ADDR,i`	`if (rI?<0 or rI?==0 or rI?>0) then goto ADDR + rIi;` `if (rI?>=0 or rI?!=0 or rI?<=0) then goto ADDR + rIi;`
`MOVE ADDR,i(F)`	`for (n = 0; n < F; n++, rI1++)` `  memory[rI1] := memory[ADDR+rIi+n];`

`SLA/SRA ADDR,i`
`SLAX/SRAX ADDR,i`
`SLC/SRC ADDR,i`
	shift `rA` to the left/right by `ADDR+rIi` bytes
	shift `(rA,rX)` to the left/right by `ADDR+rIi` bytes
	rotate `(rA,rX)` to the left/right by `ADDR+rIi` bytes

`NOP`	do nothing;
`HLT`	halt execution;

`IN ADDR,i(F)`	read in one block from input unit `F` into `memory[ADDR + rIi]` onwards;
`OUT ADDR,i(F)`	output one block to unit `F` from `memory[ADDR + rIi]` onwards;
`IOC ADDR,i(F)`	send control instruction to i/o unit `F`;
`JRED ADDR,i(F)`	`if (i/o unit F is ready) then goto ADDR + rIi;`
`JBUS ADDR,i(F)`	`if (i/o unit F is busy) then goto ADDR + rIi;`
`NUM`	`rA := numerical value of characters in (rA,rX);`
`CHAR`	`(rA,rX) := character codes representing value of rA;`

Implementations [edit]

MIX has been implemented in software by:

- Knuth's *MIXWare* and the derived GNU MDK;
- 9front's mix(1);[2] and
- Hardware::Simulator::MIX on CPAN.[3]

An implementation of MIX was created for the iCE40HX8K FPGA board in 2021.[4]

See also [edit]

- Educational programming language
- DLX
- LC-3
- Little man computer
- MMIX
- MikroSim

References [edit]

1. ^ Knuth, Donald (May 1997). *The Art of Computer Programming, Volume 1: Fundamental Algorithms* (3 ed.). Addison Wesley Longman. p. 124. ISBN 0-201-89683-4.
2. ^ `mix(1)` – 9front manual page
3. ^ Hardware::Simulator::MIX Perl module from CPAN
4. ^ "Michael Schröder / mix-fgpa" . *GitLab*.

External links [edit]

- MMIX 2009: A RISC Computer for the Third Millennium Knuth's official MIX page
- MMIX News Knuth's official MIX news

- MIX: the design of a typical computer and its assembly language on Open Library at the Internet Archive, Knuth's original 1970 official MIX book, with Tom Mix on the cover.
- MMIXware: A RISC Computer for the Third Millennium Knuth's official MIX book

V · T · E	**Donald Knuth**	[hide]
Publications	*The Art of Computer Programming* · "The Complexity of Songs" · *Computers and Typesetting* · *Concrete Mathematics* · *Surreal Numbers* · *Things a Computer Scientist Rarely Talks About* · *Selected papers series*	
Software	TeX · Metafont · MIXAL (**MIX** · MMIX)	
Fonts	AMS Euler · Computer Modern · Concrete Roman	
Literate programming	WEB · CWEB	
Algorithms	Knuth's Algorithm X · Knuth–Bendix completion algorithm · Knuth–Morris–Pratt algorithm · Knuth shuffle · Robinson–Schensted–Knuth correspondence · Trabb Pardo–Knuth algorithm · Generalization of Dijkstra's algorithm · Knuth's Simpath algorithm	
Other	Dancing Links · Knuth reward check · Knuth Prize · Knuth's up-arrow notation · Man or boy test · Quater-imaginary base · -yllion · Potrzebie system of weights and measures	

Categories: Educational abstract machines | Donald Knuth

This page was last edited on 11 October 2025, at 21:56 (UTC). Page was rendered with Parsoid.

Text is available under the Creative Commons Attribution-ShareAlike 4.0 License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

Privacy policy About Wikipedia Disclaimers Contact Wikipedia Legal & safety contacts Code of Conduct Developers Statistics

Cookie statement Mobile view