MTS on Wikipedia

Snapshot taken 9 January 2011

Contents

Articles

Michigan Terminal System 1
MTS system architecture 17
IBM System/360 Model 67 40
MAD programming language 46
UBC PLUS 55
Micro DBMS 57
Bruce Arden 58
Bernard Galler 59
TSS/360 60

References

Article Sources and Contributors 64
Image Sources, Licenses and Contributors 65

Article Licenses

License 66

Michigan Terminal System

The MTS welcome screen as seen through a 3270 terminal emulator.

Company / developer	University of Michigan and 7 other universities in the U.S., Canada, and the UK
Programmed in	various languages, mostly 360/370 Assembler
Working state	Historic
Initial release	1967
Latest stable release	6.0 / 1988 (final)
Available language(s)	English
Available programming languages(s)	Assembler, FORTRAN, PL/I, PLUS, ALGOL W, Pascal, C, LISP, SNOBOL4, COBOL, PL360, MAD/I, GOM (Good Old Mad), APL, and many more
Supported platforms	IBM S/360-67, IBM S/370 and successors

History of IBM mainframe operating systems

On early mainframe computers:

- GM OS & GM-NAA I/O 1955
- BESYS 1957
- UMES 1958
- SOS 1959
- IBSYS 1960
- CTSS 1961

On S/360 and successors:

- BOS/360 1965
- TOS/360 1965
- TSS/360 1967
- MTS 1967
- ORVYL 1967
- MUSIC 1972
  - MUSIC/SP 1985

- **DOS/360 and successors** 1966
  - DOS/VS 1972
  - DOS/VSE 1980s
  - VSE/SP late 1980s
  - VSE/ESA 1991
  - z/VSE 2005

- **OS/360 and successors** 1966
  - MFT 1966
  - MFT II 1968
    - OS/VS1 1972
    - OS/VS1 BPE
  - MVT 1967
    - 65MP
    - OS/VS2R1 (SVS) 1972
    - MVS (OS/VS2R2 and later) 1974
      - MVS/SE 1978
      - MVS/SE 2 1979
      - MVS/SP Version 1 1980
      - MVS/XA 1983
      - MVS/ESA 1988
      - OS/390 1995
      - z/OS 2000
- **VM line**
  - CP-40/CMS 1967
  - CP-67/CMS 1967
  - VP/CSS 1968
  - VM/370 1972
  - VM/BSE (BSEPP)
  - VM/SE (SEPP)
  - VM/SP 1980
  - VM/XA 1988
  - VM/ESA 1990
  - z/VM 2000
- **TPF line**
  - ACP 1967
  - TPF 1979
  - z/TPF 2005
- **UNIX and Unix-like**
  - UTS 1981
  - AIX/370 1990
  - AIX/ESA 1991
  - Linux 1999
  - OpenSolaris 2008
  - z/OS UNIX System Services

The **Michigan Terminal System** (**MTS**) is one of the first time-sharing computer operating systems.[1] Initially developed in 1967 at the University of Michigan for use on IBM S/360-67, S/370 and compatible mainframe computers, it was developed and used by a consortium of eight universities in the United States, Canada, and the United Kingdom over a period of 33 years (1967 to 1999).[2]

Overview

The software developed by the staff of the University of Michigan's academic Computing Center for the operation of the IBM S/360-67, S/370, and compatible computers can be described as a multiprogramming, multiprocessing, virtual memory, time-sharing supervisor (University of Michigan Multiprogramming Supervisor or UMMPS) that handles a number of resident, reentrant programs. Among them is a large subsystem, called MTS (Michigan Terminal System), for command interpretation, execution control, file management, and accounting. End-users interact with the computer's resources through MTS using terminal, batch, and server oriented facilities.[2]

The name MTS refers to:

- MTS the UMMPS Job Program with which most end-users interact;
- MTS the software system, including UMMPS, the MTS and other Job Programs, Command Language Subsystems (CLSs), public files (programs), and documentation; and
- MTS the time-sharing service offered at a particular site, including the MTS software system, the hardware used to run MTS, the staff that supported MTS and assisted end-users, and the associated administrative policies and procedures.

MTS was used on a production basis at 12 or 13 sites in the United States, Canada, the United Kingdom, Brazil, and possibly Yugoslavia and at several more sites on a trial or benchmarking basis. MTS was developed and maintained by a core group of eight universities that comprised the MTS Consortium.

The University of Michigan ceased operating MTS for end-users on June 30, 1996.[3] By that time, most services had moved to client/server-based computing systems, typically Unix for servers and various Mac, PC, and Unix flavors for clients. The University of Michigan shut down its MTS system for the last time on May 30, 1997.[4]

Rensselaer Polytechnic Institute (RPI) is believed to be the last site to use MTS in a production environment. RPI retired MTS in June 1999.[5]

Today MTS is not readily available, but still runs using IBM S/370 emulators such as Hercules, Sim390,[6] and FLEX-ES.[7]

Origins

In the mid-1960s, the University of Michigan was providing batch processing services on IBM 7090 hardware under the control of the University of Michigan Executive System (UMES), but was interested in offering interactive services using time-sharing.[8] At that time the work that computers could perform was limited by their lack of real memory storage capacity. When IBM introduced its System/360 family of computers in the mid-1960s, it did not provide a solution for this limitation and within IBM there were conflicting views about the importance of and need to support time-sharing.

A paper titled *Program and Addressing Structure in a Time-Sharing Environment* by Bruce Arden, Bernard Galler, Frank Westervelt (all associate directors at UM's academic Computing Center), and Tom O'Brian building upon some basic ideas developed at the Massachusetts Institute of Technology (MIT) was published in January 1966.[9] The paper outlined a virtual memory architecture using dynamic address translation (DAT) that could be used to implement time-sharing.

After a year of negotiations and design studies, IBM agreed to make a one-of-a-kind version of its S/360-65 mainframe computer with dynamic address translation (DAT) features that would support virtual memory and accommodate UM's desire to support time-sharing. The computer was dubbed the Model S/360-65M.[8] The "M" stood for Michigan. But IBM initially decided not to supply a time-sharing operating system for the machine. Meanwhile, a number of other institutions heard about the project, including General Motors, the Massachusetts Institute of Technology's (MIT) Lincoln Laboratory, Princeton University, and Carnegie Institute of Technology (later Carnegie Mellon University). They were all intrigued by the time-sharing idea and expressed interest in ordering the modified IBM S/360 series machines. With this demonstrated interest IBM changed the computer's model number to S/360-67 and made it a supported product.[1] With requests for over 100 new model S/360-67s IBM realized there was a market for time-sharing, and agreed to develop a new time-sharing operating system called TSS/360 (TSS stood for Time-sharing System) for delivery at roughly the same time as the first model S/360-67.

While waiting for the Model 65M to arrive, UM Computing Center personnel were able to perform early time-sharing experiments using an IBM S/360-50 that was funded by the ARPA CONCOMP (Conversational Use of Computers) Project[10] . The time-sharing experiment began as a "half-page of code written out on a kitchen table" combined with a small multi-programming system, LLMPS from MIT's Lincoln Laboratory,[1] which was modified

and became the UM Multi-Programming Supervisor (UMMPS) which in turn ran the MTS job program. This earliest incarnation of MTS was intended as a throw-away system used to gain experience with the new IBM S/360 hardware and which would be discarded when IBM's TSS/360 operating system became available.

Development of TSS took longer than anticipated, its delivery date was delayed, and it was not yet available when the S/360-67 (serial number 2) arrived at the Computing Center in January 1967.[11] At this time UM had to decide whether to return the Model 67 and select another mainframe or to develop MTS as an interim system for use until TSS was ready. The decision was to continue development of MTS and the staff moved their initial development work from the Model 50 to the Model 67. TSS development was eventually canceled by IBM, then reinstated, and then canceled again. But by this time UM liked the system they had developed, it was no longer considered interim, and MTS would be used at UM and other sites for 33 years.

MTS Consortium

MTS was developed, maintained, and used by a consortium of eight universities in the U.S., Canada, and the United Kingdom:[2] [12]

- University of Michigan (UM), 1967 to 1997,[13] USA
- University of British Columbia (UBC), 1968 to 1998, Canada
- NUMAC (University of Newcastle upon Tyne, University of Durham, and Newcastle Polytechnic),[14] 1969 to 1992, United Kingdom
- University of Alberta (UQV), 1971 to 1994,[15] Canada
- Wayne State University (WSU), 1971 to 1998, USA
- Rensselaer Polytechnic Institute (RPI), 1976 to 1999, USA
- Simon Fraser University (SFU), 1977 to 1992,[16] Canada
- University of Durham (NUMAC),[14] 1982 to 1992,[17] United Kingdom

Several sites ran more than one MTS system: NUMAC ran two (first at Newcastle and later at Durham), Michigan ran three in the mid-1980s (UM for Maize, UB for Blue, and HG at Human Genetics), UBC ran three or four at different times (MTS-G, MTS-L, MTS-A, and MTS-I for general, library, administration, and instruction).

Each of the MTS sites made contributions to the development of MTS, sometimes by taking the lead in the design and implementation of a new feature and at other times by refining, enhancing, and critiquing work done elsewhere. Many MTS components are the work of multiple people at multiple sites.[18]

In the early days collaboration between the MTS sites was accomplished through a combination of face-to-face site visits, phone calls, the exchange of documents and magnetic tapes by snail mail, and informal get-togethers at SHARE or other meetings. Later, e-mail, computer conferencing using Confer and Forum, network file transfer, and e-mail attachments supplemented and eventually largely replaced the earlier methods.

The members of the MTS Consortium produced a series of 82 *MTS Newsletters* between 1971 and 1982 to help coordinate MTS development.[19]

Starting at UBC in 1974 the MTS Consortium held annual *MTS Workshops* at one of the member sites. The workshops were informal, but included papers submitted in advance and *Proceedings* published after-the-fact that included session summaries.[20] In the mid-1980s several *Western Workshops* were held with participation by a subset of the MTS sites (UBC, SFU, UQV, UM, and possibly RPI).

The annual workshops continued even after MTS development work began to taper off. Called simply the "community workshop", they continued until the mid-1990s to share expertise and common experiences in providing computing services, even though MTS was no longer the primary source for computing on their campuses and some had stopped running MTS entirely.

Mugs from MTS Workshop VIII, Ann Arbor, July 1982

MTS sites

In addition to the eight MTS Consortium sites that were involved in its development, MTS was run at a number of other sites, including:[12]

- Centro Brasileiro de Pesquisas Fisicas (CBPF)[21] within the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq),[22] Brazil
- Empresa Brasileira de Pesquisa Agropecuária (EMBRAPA),[23] Brazil
- Hewlett-Packard (HP), USA
- Michigan State University (MSU), USA
- Goddard Space Flight Center, National Aeronautics and Space Administration (NASA), USA

A copy of MTS was also sent to the University of Sarajevo, Yugoslavia, though whether or not it was ever installed is not known.

INRIA, the French national institute for research in computer science and control in Grenoble, France ran MTS on a trial basis, as did Southern Illinois University, the Naval Postgraduate School in the United States, and a few other sites.

Hardware used

In theory MTS will run on the IBM S/360-67, any of the IBM S/370 series, and its successors. MTS has been run on the following computers in production, benchmarking, or trial configurations:[2]

- IBM: S/360-67, S/370-148, S/370-168, 3033U, 4341, 4361, 4381, 3081D, 3081GX, 3083B, 3090-200, 3090-400, 3090-600, and ES/9000-720
- Amdahl: 470V/6, 470V/7, 470V/8, 5860, 5870, 5990
- Hitachi: NAS 9060
- Various S/370 emulators


Computing Center staff member Mike Alexander sitting at the console of the IBM System 360 Model 67 Duplex at the University of Michigan, 1969

The University of Michigan installed and ran MTS on the first IBM S/360-67 outside of IBM (serial number 2) in 1967, the second Amdahl 470V/6 (serial number 2) in 1975,[24] [25] the first Amdahl 5860 (serial number 1) in 1982, and the first factory shipped IBM 3090-400 in 1986.[26] The University of British Columbia (UBC) took

the lead in converting MTS to run on the IBM S/370 series (an IBM S/370-168) in 1974. The University of Alberta installed the first Amdahl 470V/6 in Canada (serial number P5) in 1975.[15]

MTS was designed to support up to four processors on the IBM S/360-67, although IBM only produced one (simplex and half-duplex) and two (duplex) processor configurations of the Model 67. In 1984 RPI updated MTS to support up to 32 processors in the IBM S/370-XA (Extended Addressing) hardware series, although 6 processors is likely the largest configuration actually used.[27] MTS supports the IBM Vector Facility,[28] available as an option on the IBM 3090 and ES/9000 systems.

Amdahl 470V/6 P2 at the University of Michigan, 1975

In early 1967 running on the single processor IBM S/360-67 at UM without virtual memory support, MTS was typically supporting 5 simultaneous terminal sessions and one batch job.[2] In November 1967 after virtual memory support was added, MTS running on the same IBM S/360-67 was simultaneously supporting 50 terminal sessions and up to 5 batch jobs.[2] In August 1968 a dual processor IBM S/360-67 replaced the single processor system, supporting roughly 70 terminal and up to 8 batch jobs.[29] By late 1991 MTS at UM was running on an IBM ES/9000-720 supporting over 600 simultaneous terminal sessions and from 3 to 8 batch jobs.[2]

MTS can run under VM/CMS and some MTS sites did so, but most ran MTS on native hardware without using a virtual machine.

Features

Some of the notable features of MTS include:[30]

- The use of Virtual memory and Data Address Translation (DAT) on the IBM S/360-67 in 1967.[31]
- The use of multiprocessing on an IBM S/360-67 with two CPUs in 1968.
- Programs with access to (for the time) very large virtual address spaces.
- A straight forward command language that is the same for both terminal and batch jobs.
- A strong device independent input/output model that allows the same commands and programs to access terminals, disk files, printers, magnetic and paper tapes, card readers and punches, floppy disks, network hosts, and an audio response unit (ARU).
- A file system with support for "line files" where the line numbers and length of individual lines are stored as metadata separate from the data contents of the line and the ability to read, insert, replace, and delete individual lines anywhere in the file without the need to read or write the entire file.[37]
- Network host to host access from commands and programs as well as access to or from remote network printers, card readers and punches.[32]
- An e-mail system ($MESSAGE) that supports local and network mail with the ability to send to groups, to recall messages that haven't already been read, to add recipients to messages after they have been sent, and to display a history of messages in an e-mail chain without the need to include the text from older messages in each new message.[33]
- The ability to access tapes remotely, and to handle data sets that extend across multiple tapes efficiently.[34]
- The availability of a rich collection of well-documented subroutine libraries.[35] [19] [36]
- The ability for multiple users to quickly load and use a collection of common reentrant subroutines, which are available in shared virtual memory.
- The availability of compilers, assemblers, and a Symbolic Debugging System (SDS) that allow users to debug programs written in high-level languages such as FORTRAN, Pascal, PL/I, ... as well as in assembly language.

- A file editor ($EDIT) with both command line and "visual" interfaces and pattern matching based on SNOBOL4 patterns.[38]
- The ability to share files in controlled ways (read, write-change, write-expand, destroy, permit).[39]
- The ability to permit files, not just to other user IDs and projects (aka groups), but to specific commands or programs and combinations of user IDs, projects, commands and programs.[39]
- The ability for multiple users to manage simultaneous access to files with the ability to implicitly and explicitly lock and unlock files and to detect deadlocks.[37]
- A strong protection model that uses the virtual memory hardware and the S/360 and S/370 hardware's supervisor and problem states and via software divides problem state execution into system (privileged or unprotected) and user (protected or unprivileged) modes. Relatively little code runs in supervisor state. For example Device Support Routines (DSRs, aka device drivers) are not part of the supervisor and run in system mode in problem state rather than in supervisor state.[39] [40] [41]
- A simulated Branch on Program Interrupt (BPI) instruction.[42]

Programs developed for MTS

The following are some of the notable programs developed for MTS:[43]

- Awit, a computer chess program written in Algol W by Tony Marsland.[44]
- Chaos, one of the leading computer chess programs from 1973 through 1985. Written in FORTRAN Chaos started at RCA Systems Programming division in Cinnaminson, NJ with Fred Swartz and Victor Berman as first authors, Mike Alexander and others joined the team later and moved development to MTS at the UM Computing Center.[45]
- Confer II, one of the first computer conferencing systems. Confer was developed by Robert Parnes starting in 1975 while he was a graduate student and with support from the University of Michigan's Center for Research on Learning and Teaching (CRLT) [47] [48] [49].
- FakeOS, a simulator that allows object modules containing OS/360 SVCs, control blocks, and references to OS/360 access methods to execute under MTS.
- Forum, a computer conferencing system developed by staff of the Computing Centre at the University of British Columbia (UBC).
- GOM (Good Old Mad), a compiler for the 7090 MAD language converted to run under MTS by Don Boettner of the UM's Computing Center.[53]
- IF (Interactive Fortran), developed by the University of British Columbia Computing Centre.
- Micro, one of the earliest relational database management systems implemented in 1968 at the University of Michigan.
- MIDAS (Michigan Interactive Data Analysis System), an interactive statistical analysis package developed by Dan Fox and others at UM's Statistical Research Laboratory.[46]
- Plus, a programming language developed by Alan Ballard and Paul Whaley of the Computing Centre at the University of British Columbia (UBC).[50]
- TAXIR, an information storage and retrieval system designed for taxonomic data at the University of Colorado by David Rogers, Henry Fleming, Robert Brill, and George Estabrook and ported to MTS and enhanced by Brill at the University of Michigan.[51]
- Textform, a text-processing program developed at the University of Alberta's Computing Centre to support device independent output to a wide range of devices from line printers, to the Xerox 9700 page printers, to advanced phototypesetting equipment using fixed width and proportional fonts.[52]
- VSS, a simulator developed at the University of British Columbia's Computing Centre that makes it possible to run OS/MFT, OS/MVT, VS1, and MVS application programs under MTS.

Programs that run under MTS

The following are some of the notable programs ported to MTS from other systems:[43]

- APL VS, IBM's APL VS compiler program product.
- ASMH, a version of IBM's 370 assembler with enhancements from SLAC and MTS.
- COBOL VS, IBM's COBOL VS compiler program product.
- CSMP, IBM's Continuous System Modeling Program.[55]
- Fortran, the G, H, and VS compilers from IBM.
- GASP, a FORTRAN based discrete simulation package.[56]
- MPS, IBM's Mathematical Programming System/360.[58]
- NASTRAN, finite element analysis program originally developed by and for NASA.[59]
- OSIRIS (Organized Set of Integrated Routines for Investigations with Statistics), a collection of statistical analysis programs developed at the University of Michigan's Institute for Social Research [60] (ISR).[61]
- PascalSB, the Stony Brook Pascal compiler.
- Pascal/SLAC, the Pascal compiler from the Stanford Linear Accelerator Center.
- Pascal VS, IBM's Pascal VS compiler program product.
- PL/I Optimizing Compiler from IBM.
- REDUCE2, an algebraic language implemented in LISP.[54]
- SAS (Statistical Analysis System).
- SHAZAM, a package for estimating, testing, simulating and forecasting econometrics and statistical models
- SIMSCRIPT II.5, a free-form, English-like, general-purpose discrete event simulation language.[57]
- SPIRES (Stanford Public Information Retrieval System), a database management system.
- SPSS (Statistical Package for the Social Sciences)
- TELL-A-GRAPH, a proprietary conversational graphics program from ISSCO [62] of San Diego, CA.[63]
- TEX, Don Knuth's TeX text-processing program.[64]
- TROLL, econometric modeling and statistical analysis[65]

Programming languages available under MTS

MTS supports a rich set of programming languages, some developed for MTS and others ported from other systems:[43]

- ALGOL W[66]
- ALGOL 68[67]
- APL (IBM's VS APL)[70] [71] [72] [73]
- Assembler (360/370: G, H, Assist; DEC PDP-11)
- BASIC (BASICUM[78] , WBASIC[43] [79])
- BCPL[81] (Basic Combined Programming Language)
- C
- COBOL (ANSI[83] , VS[84] , WATBOL)
- EXPL[86] (Extended XPL)
- FORTRAN[87] (G, H, VS, WATFOR, WATFIV)
- GASP (A FORTRAN-based discrete simulation language)
- GOM (**G**ood **O**ld **M**ad, the 7090 Michigan Algorithm Decoder ported to the S/370 architecture)[53]
- MPS, IBM's Mathematical Programming System/360[58]
- MTS LISP 1.5[68] (a new implementation of LISP 1.5[69] developed at the UM's Mental Health Research Institute, MHRI)
- Pascal[74] (VS[75] [76] , JB)
- PIL, PIL/2 (Pitt Interpretive Language)[77]
- PL/I[80] (F and OPT from IBM, PL/C from Cornell University)
- PL/M
- PL360[82]
- Plus[50] (A "Pascal-like" system implementation language from the University of British Columbia (UBC) based on the SUE[85] system language developed at the University of Toronto, circa 1971)
- Prolog
- Simula[88]
- SUE[85]
- SNOBOL4[89] (String Oriented Symbolic Language[90])

- GPSS/H[91] (General Purpose Simulation System V)[92] [93]
- ICON[94]

- IF (Interactive FORTRAN, an incremental compiler and environment for executing and debugging FORTRAN programs, developed at the University of British Columbia)
- MAD/I (an expanded version of the Michigan Algorithm Decoder for the IBM S/360 architecture that is not compatible with the original 7090 version of MAD, see also GOM above)

- SPITBOL[89] (Speedy Implementation of SNOBOL)
- UMIST[95] (University of Michigan Interpretive String Translator, based on TRAC)[96]

System architecture

MTS Architecture[97]

	State	Mode[39]	VM	Interrupts
User programs	problem	user	on	on
Command Language Subsystems (CLSs), Device Support Routines (DSRs), System Subroutines		system		
Job programs (MTS, PDP, DMGR, RM or HASP, ...)			on or off	
Supervisor (UMMPS)	supervisor	n/a	off	off
S/360-67 or S/370 hardware				

UMMPS, the supervisor, has complete control of the hardware and manages a collection of job programs.[29] One of the job programs is MTS, the job program with which most users interact.[2] MTS operates as a collection of command language subsystems (CLSs). One of the CLSs allows for the execution of user programs. MTS provides a collection of system subroutines that are available to CLSs, user programs, and MTS itself.[35] Among other things these system subroutines provide standard access to Device Support Routines (DSRs), the components that perform device dependent input/output.

Manuals and documentation

The lists that follow are quite University of Michigan centric. Most other MTS sites used some of this material, but they also produced their own manuals, memos, reports, and newsletters tailored to the needs of their site.

End-user documentation

The manual series *MTS: The Michigan Terminal System*, was published from 1967 through 1991, in volumes 1 through 23, which were updated and reissued irregularly.[19] Initial releases of the volumes did not always occur in numeric order and volumes occasionally changed names when they were updated or republished. In general, the higher the number, the more specialized the volume.

The earliest versions of *MTS Volume I and II* had a different organization and content from the MTS volumes that followed and included some internal as well as end user documentation. The second edition from December 1967 covered:

- *MTS Volume I*: Introduction; Concepts and facilities; Calling conventions; Batch, Terminal, Tape, and Data Concentrator user's guides; Description of UMMPS and MTS; Files and devices; Command language; User Programs; Subroutine and macro library descriptions; Public or library file descriptions; and Internal specifications: Dynamic loader (UMLOAD), File and Device Management (DSRI prefix and postfix), Device

Support Routines (DSRs), and File routines[98]

- *MTS Volume II*: Language processor descriptions: F-level assembler; FORTRAN G; IOH/360; PIL; SNOBOL4; UMIST; WATFOR; and 8ASS (PDP-8 assembler)[95]

The following *MTS Volumes* were published by the University of Michigan Computing Center[2] and most are available as PDFs:[99]

- MTS Volume 1: *The Michigan Terminal System*, 1991
- MTS Volume 2: *Public File Descriptions*, 1990
- MTS Volume 3: *Subroutine and Macro Descriptions*, 1989
- MTS Volume 4: *Terminals and Networks in MTS*, 1988 (earlier *Terminals and Tapes*)
- MTS Volume 5: *System Services*, 1985
- MTS Volume 6: *FORTRAN in MTS*, 1988
- MTS Volume 7: *PL/I in MTS*, 1985
- MTS Volume 8: *LISP and SLIP in MTS*, 1983
- MTS Volume 9: *SNOBOL4 in MTS*, 1983
- MTS Volume 10: *BASIC in MTS*, 1980
- MTS Volume 11: *Plot Description System*, 1985
- MTS Volume 12: *PIL/2 in MTS*, 1974
- MTS Volume 13: *The Symbolic Debugging System*, 1985 (earlier *Data Concentrator User's Guide*)
- MTS Volume 14: *360/370 Assemblers in MTS*, 1986
- MTS Volume 15: *FORMAT and TEXT360*, 1988
- MTS Volume 16: *ALGOL W in MTS*, 1980
- MTS Volume 17: *Integrated Graphics System*, 1984
- MTS Volume 18: *MTS File Editor*, 1988
- MTS Volume 19: *Tapes and Floppy Disks*, 1993
- MTS Volume 20: *PASCAL in MTS*, 1989
- MTS Volume 21: *MTS Command Extensions and Macros*, 1991
- MTS Volume 22: *Utilisp in MTS*, 1988
- MTS Volume 23: *Messaging and Conferencing in MTS*, 1991

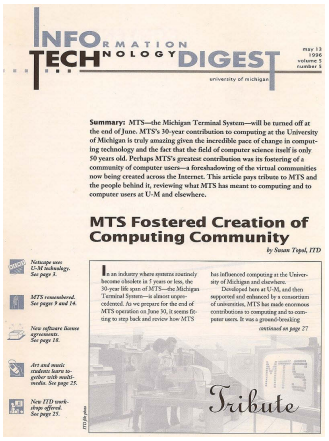
- *MTS Reference Summary*, a ~60 page, 3" x 7.5", pocket guide to MTS at the University of Michigan
- *Fundamental Use of the Michigan Terminal System*, Thomas J. Schriber, 5th Edition (revised), Ulrich's Books, Inc., Ann Arbor, MI, 1983, 376 pp.
- *Digital computing, FORTRAN IV, WATFIV, and MTS (with *FTN and *WATFIV)*, Brice Carnahan and James O Wilkes, University of Michgian, Ann Arbor, MI, 1968–1979, 1976 538 p.

Various aspects of MTS at the University of Michigan were documented in a series of *Computing Center Memos* (CCMemos)[100] which were published irregularly from 1967 through 1987, numbered 2 through 924, though not necessarily in chronological order. Numbers 2 through 599 are general memos about various software and hardware; the 600 series are the Consultant's Notes series—short memos for beginning to intermediate users; the 800 series covers issues relating to the Xerox 9700 printer, text processing, and typesetting; and the 900 series covers microcomputers. There was no 700 series. In 1989 this series continued as *Reference Memos* with less of a focus on MTS.[101] [102]

A long run of newsletters targeted to end-users at the University of Michigan with the titles *Computing Center News*, *Computing Center Newsletter*, *U-M Computing News*, and the *Information Technology Digest* were published starting in 1971.[100]

There was also introductory material presented in the *User Guide*, *MTS User Guide*, and *Tutorial* series, including:

- *Introduction to the Computing Center*
- *Introduction to MTS*
- *Introduction to Terminals*
- *Getting connected—Introduction to Terminals and Microcomputers*
- *Introduction to Magnetic Tapes*
- *Introduction to FORMAT*


INFORMATION TECHNOLOGY DIGEST

university of michigan

Summary: MTS—the Michigan Terminal System—will be turned off at the end of June. MTS's 30-year contribution to computing at the University of Michigan is truly amazing given the incredible pace of change in computing technology and the fact that the field of computer science itself is only 50 years old. Perhaps MTS's greatest contribution was its fostering of a community of computer users—a foreshadowing of the virtual communities now being created across the Internet. This article pays tribute to MTS and the people behind it, reviewing what MTS has meant to computing and to computer users at U-M and elsewhere.

MTS Fostered Creation of Computing Community

by Susan Topol, ITD

Tribute

Univ. of Michigan IT Digest, May 1996

Internals documentation

The following materials were not widely distributed, but were included in MTS Distributions and are available at the Bentley Historical Library[19] at the University of Michigan:

- MTS Operators Manual[103]
- MTS Message Manual
- MTS Volume *n*: Systems Edition[104]
- MTS Volume 99: Internals Documentation
- Supervisor Call Descriptions[105]
- Disk Disaster Recovery Procedures
- A series of lectures describing the architecture and internal organization of the Michigan Terminal System given by Mike Alexander, Don Boettner, Jim Hamilton, and Doug Smith (4 audio tapes, lecture notes, and transcriptions)

Distribution

The University of Michigan released MTS on magnetic tape on an irregular basis.[19] There were full and partial distributions, where *full distributions* (D1.0, D2.0, ...) included all of the MTS components and *partial distributions* (D1.1, D1.2, D2.1, D2.2, ...) included just the components that had changed since the last full or partial distribution.

MTS distributions included the updates needed to run licensed program products and other proprietary software under MTS, but not the base proprietary software itself, which had to be obtained separately from the owners. Except for IBM's Assembler H, none of the licensed programs were required to run MTS.

The last MTS distribution was D6.0 released in April 1988. It consisted of 10,003 files on six 6250 bpi magnetic tapes. After 1988 distribution of MTS components was done in an ad hoc fashion using network file transfer.

To allow new sites to get started from scratch two additional magnetic tapes were made available, an IPLable *boot tape* that contained a minimalist version of MTS plus the DASDI and DISKCOPY utilities that could be used to initialize and restore a *one disk pack starter version* of MTS from the second magnetic tape. In the earliest days of MTS the standalone TSS DASDI and DUMP/RESTORE utilities rather than MTS itself were used to create the one disk starter system.

There were also less formal *redistributions* where individual sites would send magnetic tapes containing new or updated work to a coordinating site. That site would copy the material to a common magnetic tape (RD1, RD2, ...), and send copies of the tape out to all of the sites. Sadly the contents of the redistribution tapes seem to have been

lost.

Licensing

MTS is no longer available for license.

In its earliest days MTS was made available for free without the need for a license to sites that were interested in running MTS and which seemed to have the knowledgeable staff required to support it.

In the mid-1980s licensing arrangements were formalized with the University of Michigan acting as agent for and granting licenses on behalf of the MTS Consortium.[106] MTS licenses were available to academic organizations for an annual fee of $5,000, to other non-profit organizations for $10,000, and to commercial organizations for $25,000. The license restricted MTS from being used to provide commercial computing services. The licensees received a copy of the full set of MTS distribution tapes, any incremental distributions prepared during the year, written installation instructions, two copies of the current user documentation, and a very limited amount of assistance.

Only a few organizations licensed MTS. Several licensed MTS in order to run a single program such as CONFER. The fees collected were used to offset some of the common expenses of the MTS Consortium.

References

[1] "In late 1968, MTS was the only large-scale timesharing system to be in regular, reliable operation in the US" in "The Life and Work of Bernard A. Galler (1928-2006) (http://muse.jhu.edu/journals/ieee_annals_of_the_history_of_computing/v030/30.1akera.pdf)", Atsushi Akera, *IEEE Annals of the History of Computing*, vol. 30, no. 1 (Jan-Mar 2008), p.8

[2] MTS Volume 1: *The Michigan Terminal System* (http://archive.michigan-terminal-system.org/documents/MTSVolume1-TheMichiganTerminalSystem-1991.pdf?attredirects=0&d=1), pages 3,11-14, May 1984 and Nov. 1991, University of Michigan Computing Center, Ann Arbor, Michigan

[3] "MTS Service to End", *Information Technology Digest*, Vol. 5, No. 5 (May 12, 1996), p.7

[4] "MTS Timeline" (http://www.clock.org/~jss/work/mts/timeline.html), *Information Technology Digest*, University of Michigan, pp.9-10, Volume 5, No. 5 (May 13, 1966)

[5] "MTS Timeline" (http://www.clock.org/~jss/work/mts/timeline.html), an after the fact one entry addition for 1999 to *Information Technology Digest*, University of Michigan, Volume 5, No. 5 (May 13, 1966)

[6] Sim390 (http://www.canpub.com/teammpg/de/sim390/), an ESA/390 emulator

[7] FLEX-ES (http://www.funsoft.com/mfos-body.html), a S/390 and z/Architecture emulator

[8] "A History of MTS—30 Years of Computing Service" (https://www.msu.edu/~mrr/mycomp/mts/others/feat02.htm), Susan Topol, *Information Technology Digest*, Volume 5, No. 5 (May 13, 1996), University of Michigan

[9] "Program and Addressing Structure in a Time-Sharing Environment" (http://portal.acm.org/citation.cfm?doid=321312.321313), B. W. Arden , B. A. Galler , T. C. O'Brien , F. H. Westervelt, *Journal of the ACM (JACM)*, v.13 n.1, p.1-16, Jan. 1966

[10] *CONCOMP : Research in Conversational Use of Computers : Final Report* (http://deepblue.lib.umich.edu/handle/2027.42/8249), Westervelt, F. H., University of Michigan Computing Center, 1970

[11] *The IBM 360/67 and CP/CMS* (http://www.multicians.org/thvv/360-67.html), Tom Van Vleck

[12] "How did sites learn about and make the decision to use MTS?" (http://archive.michigan-terminal-system.org/discussions/how-did-sites-learn-about-and-make-the-decision-to-use-mts), an item in the discussion section of the Michigan Terminal System Archive

[13] MTS at UM Retired (http://www.clock.org/~jss/work/mts/no-more-mts.html)

[14] Northumbrian Universities Multiple Access Computer (N.U.M.A.C.) (http://www.dur.ac.uk/its/news/archive/article/?itemno=427&rehref=/its/news/archive/issues/january2005/&resubj=Vol+12+No.+2+---+January+2005 Headlines), a collaboration between of the universities of Durham (DUR), Newcastle upon Tyne (UNE) and Newcastle Polytechnic that shared a S/360-67 at Newcastle starting in 1969

[15] *Timeline of Computing Services at the University of Alberta* (http://www.ualberta.ca/~vbowler/HyperDispatch19/timeline.html)

[16] *Dropping the Mainframe Without Crushing the Users: Mainframe to Distributed UNIX in Nine Months* (http://www.google.com/url?sa=t&source=web&cd=26&ved=0CCMQFjAFOBQ&url=http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.2631&rep=rep1&type=pdf&ei=LmAUTMrsC8mmnAfhxIWoDA&usg=AFQjCNG62jZ-BWj517MZLIud1u861XMakA) Peter Van Epp & Bill Baines, Simon Fraser University

[17] In 1982 NUMAC (http://www.dur.ac.uk/its/news/archive/article/?itemno=427&rehref=/its/news/archive/issues/january2005/&resubj=Vol+12+No.+2+---+January+2005 Headlines) installed a separate machine running MTS at the University of Durham, prior to that both DUR and UNE shared a single MTS system running at the University of Newcastle upon Tyne.

[18] It is difficult to properly give credit for all the work that was done, however, to avoid giving too little credit and at the risk of not giving proper credit to everyone that made contributions, an attempt is made to note the sites where a major feature or enhancement was initially developed

[19] Michigan Terminal System (MTS) subseries (http://quod.lib.umich.edu/cgi/f/findaid/findaid-idx?c=bhlead;idno=umich-bhl-0351;view=reslist;didno=umich-bhl-0351;subview=standard;focusrgn=C02;cc=bhlead;byte=24474999), Computing Center publications, 1965-1999, Bentley Historical Library, University of Michigan

[20] MTS (Michigan Terminal System) 1970-1986 series (http://quod.lib.umich.edu/cgi/f/findaid/findaid-idx?c=bhlead;idno=umich-bhl-9551;view=reslist;didno=umich-bhl-9551;subview=standard;focusrgn=C01;cc=bhlead;byte=24586272), Computing Center (University of Michigan) records, 1952-1996 and 1959-1987, Bentley Historical Library, University of Michigan

[21] CBPF is the Brazilian Center for Physics Research (http://portal.cbpf.br/index.php?page=home&lang=en)

[22] CNPq is the National Council of Scientific and Technological Development (http://www.cnpq.br/english/cnpq/index.htm)

[23] EMBRAPA is the Brazilian Enterprise for Agricultural Research (http://www.embrapa.br/english)

[24] Amdahl 470V/6 P2 at the Computing History Museum (http://www.computerhistory.org/collections/accession/X436.84A)

[25] "A performance Comparison of the Amdahl 470V/6 and the IBM 370/168", Allan R. Emery and M. T. Alexander, a paper read at the meeting of the Computer Measurement Group, October 1975, San Francisco

[26] Earlier 3090-400s were upgraded in the field from 3090-200s, "Installing the 3090", *UM Computing News*, vol 1, no. 8, 10 November 1986, p. 5

[27] MTS History at RPI (http://archive.michigan-terminal-system.org/documents/MTSHistoryAtRPI-1989.pdf?attredirects=0&d=1), 1989, 5p.

[28] "The IBM System/370 vector architecture" (http://domino.research.ibm.com/tchjr/journalindex.nsf/600cc5649e2871db852568150060213c/c8f541b78dae5a6485256bfa00685bb2!OpenDocument), W. Buchholz, *IBM Systems Journal*, Volume 25, No. 1 (1986), pp. 51-62

[29] "Organization and features of the Michigan Terminal System" (http://www.computer.org/portal/web/csdl/doi/10.1109/AFIPS.1972.80), M. T. Alexander, p. 586, *Proceedings of the May 1972 AFIPS Spring Joint Computer Conference*

[30] *MTS Innovations in A History of MTS: 30 Years of Computing Service* (http://www.clock.org/~jss/work/mts/30years.html), *Information Technology Digest*, Volume 5, No. 5 (May 13, 1966), University of Michigan

[31] Michigan Terminal System (http://www.eecis.udel.edu/~mills/gallery/gallery8.html) overview and photos by Dave Mills

[32] "A Chronicle of Merit's Early History" (http://www.merit.edu/about/history/article.php). Merit Network. 2008. . Retrieved 2008-09-15.—A university press release called a demonstration of the network (with a connection between UM and Wayne State University) on December 14, 1971, as "a milestone in higher education" and an "historic event."

[33] MTS Volume 23: *Messaging and Conferencing in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan

[34] MTS Volume 19: *Tapes and Floppy Disks* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan

[35] MTS Volume 3: *System Subroutine Descriptions* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), February 1983, University of Michigan Computing Center, Ann Arbor, Michigan

[36] "The Internal Design of the IG Routines, an Interactive Graphics System for a Large Timesharng Environment", James Blinn and Andrew Goodrich, *SIGGRAPH Proceedings*, 1976, pp. 229-234

[37] "A file system for a general-purpose time-sharing environment" (http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1451786), G. C. Pirkola, *Proceedings of the IEEE*, June 1975, volume 63 no. 6, pp. 918–924, ISSN 0018-9219

[38] MTS Volume 18: *MTS File Editor* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan

[39] "The Protection of Information in a General Purpose Time-Sharing Environment" (http://archive.michigan-terminal-system.org/documents/protection-1977.pdf?attredirects=0&d=1), Gary C. Pirkola and John Sanguinetti, *Proceedings of the IEEE Symposium on Trends and Applications 1977: Computer Security and Integrity*, vol. 10 no. 4, , pp. 106-114

[40] "The use of the monitor call instruction to implement domain switching in the IBM 370 architecture" (http://doi.acm.org/10.1145/850716.850719), John Sanguinetti, University of Michigan Computing Center, *ACM SIGOPS Operating Systems Review*, Volume 15, Issue 4 (October 1981), pp.55-61

[41] "A penetration analysis of the Michigan Terminal System" (http://doi.acm.org/10.1145/850693.850694), B. Hebbard, P. Grosso, et. al., *ACM SIGOPS Operating Systems Review*, Volume 14 , Issue 1 (January 1980), pp.7-20

[42] MTS Volume 14: *360/370 Assemblers in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), May 1983, University of Michigan Computing Center, Ann Arbor, Michigan

[43] MTS Volume 2: *Public File Descriptions* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan

[44] Awit article (http://chessprogramming.wikispaces.com/Awit) on chessprogramming.wikispaces.com (http://chessprogramming.wikispaces.com)

[45] Chaos article (http://chessprogramming.wikispaces.com/Chaos) on chessprogramming.wikispaces.com (http://chessprogramming.wikispaces.com)

[46] *Documentation for MIDAS: Michigan Interactive Data Analysis System*, by Daniel J. Fox and Kenneth E. Guire, 1974, Statistical Research Laboratory University of Michigan, Ann Arbor

[47] http://www.crlt.umich.edu/index.php

[48] "Computer-based educational communications at the University of Michigan" (http://portal.acm.org/citation.cfm?id=800191.805559), Karl L. Zinn, Robert Parnes, and Helen Hench, Center for Research on Learning and Teaching (CRLT) (http://www.crlt.umich.edu/index.php), University of Michigan, *Proceedings of the ACM Annual Conference/Meeting*, 1976, pages 150-154

[49] The History of the Student Conferencing Project (http://www.umich.edu/~umscp/history.html), University of Michigan, c. 1997

[50] *UBC PLUS: The Plus Programming Language* (http://archive.michigan-terminal-system.org/documents/Plus-1987.pdf?attredirects=0&d=1), Allan Ballard and Paul Whaley, October 1987, University of British Columbia Computing Centre, 198pp.

[51] *The Taxir Primer* (http://www.eric.ed.gov/ERICWebPortal/contentdelivery/servlet/ERICServlet?accno=ED064063), R. C. Brill, 1971, Colorado Univ., Boulder. Inst. of Arctic and Alpine Research

[52] *Textform* (http://www.worldcat.org/title/textform/oclc/020087914), Computing Services, University of Alberta, 1984, 216 p.

[53] *GOM: Good Old Mad* (http://archive.michigan-terminal-system.org/documents/GOMManual-June1989.pdf?attredirects=0&d=1), Donald Boettner, June 1989, University of Michigan Computing Center, 110p.

[54] "REDUCE 2: A system and language for algebraic manipulation" (http://doi.acm.org/10.1145/800204.806277), *Proceedings of the Second ACM Symposium on Symbolic and Algebraic Manipulation*, 1971, pages 128-133

[55] *Continuous-system simulation languages: A state-of-the-art survey* (http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V0T-4H3SBVM-3&_user=10&_coverDate=01/31/1974&_rdoc=1&_fmt=high&_orig=search&_sort=d&_docanchor=&view=c&_searchStrId=1368425567&_rerunOrigin=google&_acct=C000050221&_version=1&_urlVersion=0&_userid=10&md5=15e19c1ebccde8a454e7fcce5f8a8ea9), Ragnar N. Nilsen and Walter J. Karplus, Computer Science Department, UCLA

[56] *Simulation with GASP II*, A. A. B. Pritzker and Philip J. Kiviat, Prentice-Hall, 1969

[57] *Building Simulation models with SIMSCRIPT II.5* (http://www.caciasl.com/cust_center/ss3docs/zbuildin.pdf), Edward C. Russell, 1999, CACI, Los Angeles, CA

[58] *MPS/360 Version 2, Linear and Separable Programming User's Manual* (GH20-0476), 1971, IBM Corporation

[59] *MSC/NASTRAN at the University of Michigan* (http://www.mscsoftware.com/support/library/conf/wuc82/p00282.pdf), William J. Anderson and Robert E. Sandstorm, 1982, University of Michigan College of Engineering

[60] http://www.isr.umich.edu/home

[61] "Statistical Analysis and Data Management Highlights of OSIRIS IV" (http://www.jstor.org/stable/2684124), Neal A. Van Eck, *The American Statistician*, Vol. 34, No. 2 (May, 1980), pp. 119-121

[62] http://investing.businessweek.com/research/stocks/private/snapshot.asp?privcapId=30311

[63] *TELL-A-GRAF in MTS*, Computing Center Memo 450, University of Michigan

[64] *The Texbook* by Don Knuth, 1984, Addison-Wesley Publishing Company

[65] History of TROLL (http://www.intex.com/troll/history.html)

[66] MTS Volume 16: *ALGOL W in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan

[67] *Revised Report on the Algorithmic Language ALGOL 68 (PDF)* (http://vestein.arb-phys.uni-dortmund.de/~wb/RR/rr.pdf), A. van Wijngaarden, et al.

[68] MTS Volume 8: *LISP and SLIP in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan

[69] *LISP 1.5 Programmer's Manual*, J. McCarthy, et al., 1966, MIT Press, Cambridge, MA

[70] CCMemo 435: *MTS VS APL User's Guide*, University of Michigan Computing Center, Ann Arbor, Michigan

[71] *A Programming Language*, K. E. Iverson, 1962, John Wiley & Sons

[72] *APL Language Reference*, IBM publication GC26-3874

[73] *APL/360 Primer*, IBM publication GH20-0689

[74] MTS Volume 20: *PASCAL in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan

[75] CCMemo 436: *Pascal VS in MTS"*

[76] *Pascal/VS Language Reference Manual*, IBM publication SH20-6168

[77] MTS Volume 12: *PIL/2 in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan

[78] MTS Volume 10: *Basic in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan

[79] *Waterloo BASIC - A Structured Programming Approach, Primer and Reference Manual*, J. W. Grahm, et al., 1980, WATFAC Publications Ltd., Waterloo, Ontario, Canada

[80] MTS Volume 7: *PL/I in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan

[81] *The BCPL Reference Manual* (http://www.fh-jena.de/~kleine/history/languages/Richards-BCPL-ReferenceManual.pdf), Memorandum M-352, Project MAC, Cambridge, July, 1967

[82] "PL360, a programming language for the 360 computers", Niklaus Wirth, *Journal of the ACM* (JACM), Volume 15, No. 1, January 1968, pp.37-74

[83] *IBM OS Full Americal National Standard COBOL System Library Manual*, IBM publication GC28-6396

[84] CCMemo 439: *IBM VS COBOL under MTS*, University of Michigan Computing Center

[85] "The System Language for Project SUE" (http://doi.acm.org/10.1145/800234.807062), B. L. Clark and J. J. Horning of the Computer Systems Research Group and Department of Computer Science, University of Toronto, *Proceedings of the SIGPLAN symposium on Languages for system implementation*, 1971, pp.79-88
[86] CCMemo 416: *Extended XPL", University of Michigan Computing Center*
[87] MTS Volume 6: *FORTRAN in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan
[88] "Compiling Simula: A historical study of technological genesis" (http://www.idi.ntnu.no/grupper/su/publ/simula/holmevik-simula-ieeeannals94.pdf,), Jan Rune Holmevik, *IEEE Annals in the History of Computing*, Volume 16 No. 4, 1994, pp.25-37
[89] MTS Volume 9: *SNOBOL4 in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan
[90] *The SNOBOL4 Programming Language*, Griswold, Ralph E., J. F. Poage, and I. P. Polonsky, Englewood Cliffs, NJ, 1968, Prentice Hall
[91] *GPSS/H User's Manual*, James O. Henriksen, Wolverine Software Corp.
[92] *IBM General Purpose Simulation System V User's Manual*, IBM publication SH20-0851
[93] *Simulation Using GPSS*, Thomas J. Schriber, 1974, John Wiley & Sons
[94] *The Icon Programming Language*, Ralph E. Griswold and Madge T. Griswold, 1983, Prentice-Hall, N.Y.
[95] MTS Volume II (http://www.bitsavers.org/pdf/univOfMichigan/mts/Michigan_Terminal_System_Second_Edition_Vol2_Dec67.pdf), second edition, December 1, 1967, University of Michigan Computing Center, Ann Arbor, Michigan, 415 p.
[96] "TRAC, A Procedure-Describing Language for the Reactive Typewriter" (http://doi.acm.org/10.1145/365230.365270), Calvin N. Mooers, *Communications of the ACM (CACM)*, Vol.9 No.3 (March 1966), pp.215-219, ISSN:0001-0782
[97] MTS Lecture 1 (http://archive.michigan-terminal-system.org/discussions/figure-1-from-mts-lecture-1/1original/mts-lecture-1-original), a transcription of the first in a series of lectures on the internals of the Michigan Terminal System given by Mike Alexander, Don Boettner, Jim Hamilton, and Doug Smith, c. 1972
[98] MTS Volume I (http://www.bitsavers.org/pdf/univOfMichigan/mts/Michigan_Terminal_System_Second_Edition_Vol1_Dec67.pdf), second edition, December 1, 1967, University of Michigan Computing Center, Ann Arbor, Michigan, 415 p.
[99] Manuals and Documentation section (http://archive.michigan-terminal-system.org/manuals-and-documentation) of the MTS Archive Web site (archive-Michigan-Terminal-System.org (http://archive.michigan-terminal-system.org))
[100] Unit Publications series (http://quod.lib.umich.edu/cgi/f/findaid/findaid-idx?c=bhlead;idno=umich-bhl-0351;view=reslist;didno=umich-bhl-0351;subview=standard;focusrgn=C01;cc=bhlead;byte=24453394), Computing Center publications, 1965-1999, Bentley Historical Library, University of Michigan
[101] Unit Publications series (http://quod.lib.umich.edu/cgi/f/findaid/findaid-idx?c=bhlead;idno=umich-bhl-9551;view=reslist;didno=umich-bhl-9551;subview=standard;focusrgn=C01;cc=bhlead;byte=24586272), Information Technology Division (University of Michigan) publications, 1971-1999, Bentley Historical Library, University of Michigan
[102] ITD Publications (http://archive.michigan-terminal-system.org/documents/ITDPublications-Nov1995.pdf?attredirects=0&d=1), University of Michigan, Ann Arbor, November 1995, 24 pages
[103] MTS Operators Manual (http://archive.michigan-terminal-system.org/manuals-and-documentation/additional-doumentation/MTSOperatorsManual-February1995.pdf?attredirects=0&d=1), February 1995, University of Michigan, 574p.
[104] MTS Volume 1: Systems Edition, Obsolete and Internal MTS Commands (http://archive.michigan-terminal-system.org/documents/MTSVolume1-SystemsEdition-1991.pdf), November 1991, University of Michigan, 60pp.
[105] UMMPS D6.0 Supervisor Call Descriptions (http://archive.michigan-terminal-system.org/manuals-and-documentation/additional-doumentation/SupervisorCallDescriptions-November1987.pdf?attredirects=0&d=1), November 1987, University of Michigan, 156p.
[106] MTS Licensing Statement, November 1986, Leonard J. Harding, MTS (Michigan Terminal System), 1968-1996 (http://quod.lib.umich.edu/cgi/f/findaid/findaid-idx?c=bhlead;idno=umich-bhl-9551;view=reslist;didno=umich-bhl-9551;subview=standard;focusrgn=C01;cc=bhlead;byte=24634866), Computing Center records 1952-1996, Bentley Historical Library, University of Michigan

External links

- *A Comparative Study of the Michigan Terminal System (MTS) with Other Time Sharing Systems for the IBM 360/67 Computer* (http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=AD0741109), Elvert F. Hinson, Master's thesis, Naval Postgraduate School, Monterey, CA., December 1971
- "Measurement and Performance of a Multiprogramming System" (http://doi.acm.org/10.1145/961053.961101), B. Arden and D. Boettner, *Proceedings of the 2nd ACM Symposium on Operating Systems Principles*, pp. 130–46, October 1969
- Merit Network History (http://www.merit.edu/about/history/index.php)

- MTS Bibliography (http://archive.michigan-terminal-system.org/bibliography), a list of published literature about MTS
- MTS PDF Document Archive (http://www.bitsavers.org/pdf/univOfMichigan/mts/) at Bitsavers'
- "MTS - Michigan Terminal System" (http://doi.acm.org/10.1145/1232909.1232910), Donald W. Boettner and Michael T. Alexander, *ACM SIGOPS Operating Systems Review*, Volume 4, Issue 4 (December 1970)
- "The Michigan Terminal System" (http://ieeexplore.ieee.org:80/Xplore/login.jsp?reload=true&url=http://ieeexplore.ieee.org/iel5/5/31193/01451785.pdf?arnumber=1451785&authDecision=-203), Donald W. Boettner and Michael T. Alexander, *Proceedings of the IEEE*, Volume 63, Issue 6 (June 1975), pp. 912–918
- MTS History (http://mtswiki.westwood-tech.com/), collected by former University of Michigan Computing Center staff member Tom Valerio
- MTS Archive (http://archive.michigan-terminal-system.org), a collection of documents, photographs, movies, and other materials related to MTS and the organizations and people that developed and used it
- *Network Models for Large-Scale Time-Sharing Systems* (http://deepblue.lib.umich.edu/bitstream/2027.42/6686/5/bad0316.0001.001.pdf), C. Moore, Ph.D. thesis, Department of Industrial Engineering, University of Michigan, 1971, 211 p., AAT:72-04937
- *Program behavior and control in virtual storage computer systems* (http://portal.acm.org/citation.cfm?id=905274&dl=GUIDE&coll=GUIDE&CFID=94132684&CFTOKEN=45694884), Tad Brian Pinkerton, Ph.D. thesis, Department of Computer and Communications Sciences, University of Michigan, 1968, 167 p., AAT: 68-13378
- *Performance Analysis of Multi-level Paging Hierarchies* (http://portal.acm.org/citation.cfm?id=907678), James Hamilton, Ph.D. thesis, Department of Computer and Communications Sciences, University of Michigan, 1976, 173 p., AAT:76-27489
- "Performance monitoring in a time-sharing system" (http://doi.acm.org/10.1145/363269.363279), Tad B. Pinkerton, University of Michigan Computing Center, *Communications of the ACM*, Volume 12, Issue 11 (November 1969), pp. 608–610, ISSN:0001-0782
- Personal perspective on MTS (http://www.everything2.com/index.pl?node_id=1404097&lastnode_id=0) by Dan Boulet a student and later Computing Services staff member at the University of Alberta
- Personal reflections on MTS (https://www.msu.edu/~mrr/mycomp/mts/mtsframe.htm) by Mark Riordan of Michigan State University's Computer Laboratory (http://computing.msu.edu/features/040406.php)
- Several articles (http://www.clock.org/~jss/work/mts/) from the May 13, 1996 issue of the University of Michigan Information Technology Digest, Volume 5, No. 5, giving the history of and reminiscences about MTS, Merit, and UMnet on the eve of MTS's retirement at the University of Michigan, preserved on Web pages created by Josh Simon

MTS system architecture

MTS System Architecture describes the software organization of the Michigan Terminal System, a time-sharing computer operating system in use from 1967 to 1999 on IBM S/360-67, IBM System/370, and compatible computers.

Overview

MTS Architecture[1]

	State	Mode[2]	VM	Interrupts
User programs	problem	user	on	on
Command Language Subsystems (CLSs), Device Support Routines (DSRs), System Subroutines		system		
Job programs (MTS, PDP, DMGR, RM or HASP, ...)			on or off	
Supervisor (UMMPS)	supervisor	n/a	off	off
S/360-67 or S/370 hardware				

UMMPS, the supervisor, has complete control of the hardware and manages a collection of job programs.[3] One of the job programs is MTS, the job program with which most users interact.[4] MTS operates as a collection of command language subsystems (CLSs). One of the CLSs allows for the execution of user programs. MTS provides a collection of system subroutines that are available to CLSs, user programs, and MTS itself.[5] [6] Among other things these system subroutines provide standard access to Device Support Routines (DSRs), the components that perform device dependent input/output.

Organization

The system is organized as a set of independent components with well-defined interfaces between them.[3] This idea is, of course, neither new nor unique; but MTS components are generally larger, interfaces between components more rigid, and a component communicates with fewer other components than in many systems. As a result, components are more independent of each other and it is easier to replace one component without affecting others.

The interface with the supervisor is the same for all components and very few special cases are allowed; for example, all input/output operations are done using the same supervisor facilities whether the input/output is for a card reader, a paging device, or any other device. Most access to supervisor services is via system subroutines that issue the necessary SVCs rather than by direct use of SVCs. Control blocks are accessed only indirectly by calls to subroutines within the component that "owns" the control block.

The interfaces used by user programs are the cleanest of all. User programs may never refer directly to any system control block (neither for reference or change), because the virtual memory segment(s) that contain system control blocks (the system segments) are removed from a job's virtual address space when a user mode program is running. The subroutine interfaces available to user programs are also used by most other parts of the system (system mode programs, CLSs, ...) even through components running in system mode do have access to the "system" virtual memory segment(s). Transitions from user mode to system mode and back are managed by a special protected set of subroutine interfaces known as "the gate" (initially developed at Wayne State University).[2]

The programming effort for MTS is divided vertically rather than horizontally. This means that one or two individuals are assigned responsibility for a component and then follow it from design through implementation and

maintenance. The responsible person has considerable freedom to design the internal structure of the component and even extend interfaces, so long as all appropriate existing interfaces are maintained unchanged.

Programming languages and system level debugging

The supervisor, most job programs, large parts of MTS including many DSRs and CLSs are written in 360/370 assembler language. A few job programs and portions of MTS including some DSRs and CLSs are written in higher level languages such as Plus or GOM. User programs are written in a wide range of languages from assembler to any of the higher level languages that are available.

Most components of the system, including user programs, CLSs, and subroutines loaded in shared virtual memory, can be debugged and new versions of many can be installed while the system is running without requiring a system shutdown. It is possible to substitute a private copy of all components except the supervisor and parts of some job programs. A "test" version of the MTS job program (TMTS) is available to allow testing in the regular production environment. SWAT is an interface that allows the Symbolic Debugging System, which is normally used to debug user programs, to be used to debug MTS.[7] $PEEK is a privileged MTS command that uses Programm Event Recording (PER) and other facilities to facilitate debugging one job program from another.[7] Components that cannot be debugged in this way can be debugged by running in an MTS virtual machine (a user program).

Supervisor

University of Michigan Multi-Programming Supervisor (UMMPS) is the name of the MTS supervisor.[8] UMMPS is the only portion of the system that runs in S/360 supervisor state. It runs with virtual memory (relocation) turned off and with hardware interrupts disabled. With multi-processor configurations it may be executing on more than one processor concurrently. UMMPS is what today would be called a microkernel, although UMMPS was developed long before that term was in common use.

To jobs UMMPS appears to be an extension of the S/360 or S/370 hardware and is responsible for:

- allocating all hardware resources (processors, real memory, input/output devices),
- scheduling I/O operations,
- processing all hardware interrupts including page-faults and program interrupts due to errors in job programs,
- implementing virtual memory including:
  - the allocation of VM addresses,
  - managing segment and page tables,
  - providing protected or read-only memory by setting storage keys,
  - managing memory reference and change bits,
  - managing named address spaces (NASs),
  - determining when and which pages should be moved between real memory and secondary storage to implement demand paging,
- providing services to job programs that issue Supervisor Call (SVC)[9] and Monitor call (MC) instructions, including:
  - starting and terminating jobs,
  - initiation of input/output operations (channel programs),
  - scheduling timer interrupts,
  - communication with the system operator,
  - providing inter-task communication services,
  - allowing jobs to acquire and release software locks,
  - allowing jobs to enter and leave user and system mode, where user mode programs do not have access to some virtual memory segments and the full range of SVCs,[2]

  - providing services to allow the synchronization of job programs,
  - providing shadow segment and page tables and other services that allow job programs to provide virtual machine services,
- simulating a few machine instructions that are present on some, but not all, models of the S/360 or S/370 computers,
- simulating the Branch on Program Interrupt (BPI) pseudo instructions,
- machine check error recovery,
- writing job dumps (making a snapshot of the current execution state of a job by writing out all real memory, all of the job's virtual memory, general registers, and program status word to magnetic tape),
- tracking the amount of processor time used and the number of page-ins for jobs,
- maintaining the time of day clock, and
- assisting in the creation of diagnostic trace tapes.

After initialization UMMPS is entirely interrupt driven. The interrupts may be due to supervisor (SVC)[9] or monitor (MC) call instructions issued by job programs to request services, page fault interrupts for virtual memory pages that are not in real memory when referenced by a job program, program interrupts caused by abnormal conditions in job programs, timer interrupts on behalf of job programs or used internally within the supervisor, interrupts from the input/output subsystem, machine check interrupts, external (operator initiated) interrupts, and interrupts from other processors in a multiprocessor configuration.

A program interrupt in supervisor state is a system failure that results in a supervisor dump (a Super Dump, where the machine state and the contents of all real memory is written to magnetic tape) followed by a system restart (re-IPL).

Branch on Program Interrupt (BPI)

The Branch on Program Interrupt (BPI) pseudo instruction provides a simple way for a sequence of code to retain control following a program interrupt. This can be useful to test for valid addresses in a parameter list, to catch overflow, underflow, and other exceptions during calculations, or really any situation where a program interrupt is possible. BPIs can be used at very low cost for the usually more common case where there is no program interrupt.

UMMPS implements the Branch on Program Interrupt (BPI) pseudo instruction using a special type of NOP instruction.[10] The form of the BPI instruction is:

```
BPI    M2,D2(B2)         [RX]
```

or

```
  BC     0,D2(M2,B2)       [RX]
```

```
   Op Code         Mask1    Mask2    Base    Displacement
  +--------------+-------+-------+-------+-------------+
  |    x'47'     |   0   |   M2  |   B2  |    D2       |
  +--------------+-------+-------+-------+-------------+
   0              8        12      16      20          31
```

Where $Mask_1$ is always zero, $Mask_2$ is a name or value as described in the table below, and the base and displacement specify a branch address. Several BPI instructions may be given in succession. The BPI instruction is available for use in problem-state as well as supervisor-state (that is, within UMMPS itself).

When an instruction causes a program interrupt, the following instruction is checked to determine if it is a BPI instruction. If it is, the type of program interrupt that occurred is compared with the type categories specified in the $Mask_2$ portion of the BPI instruction. If there is a match, the condition code is set to reflect the interrupt that occurred and the branch is taken. Otherwise, the next instruction is checked to determine if it is a BPI instruction,

etc. If there is no BPI transfer made (either because there was no BPI instruction or because the program interrupt type did not match the mask of any BPIs that were present), the normal processing of the program interrupt occurs.

When the BPI instruction is executed normally (when there is no program interrupt on the previous instruction), it is a NOP or "branch never" instruction.

BPI interrupt-type categories:

$Mask_2$ Name	$Mask_2$ Value	Interrupt Number	Interrupt Name	Condition Code on Branch
OPCD	8	1	Operation	1
		2	Privileged operation	2
		3	Execute	3
OPND	4	4	Protection	0
		5	Addressing	1
		6	Specification	2
		7	Data	3
OVDIV	2	8	Fixed overflow	0
		9	Fixed divide	1
		10	Decimal overflow	2
		11	Decimal divide	3
FP	1	12	Exponent overflow	0
		13	Exponent underflow	1
		14	Significance	2
		15	Floating-point divide	3

Job programs

All job programs run in S/360 problem state, may run with virtual addressing enabled or disabled, and may or may not be reentrant (more than one instance of the job program may or may not be allowed to execute). With multiprocessor configurations a single job will only execute on a single processor at a time, but the supervisor may assign a job to different processors at different times.[8]

The MTS job program is the one with which most users interact and provides command interpretation, execution control, file and device management, and accounting services.[4] Other job programs assist the supervisor (the Paging Device Processor or PDP, the OPERATOR console job, the Disk Manager or DMGR, ...), provide common or shared services (spooled local and remote batch services via HASP and the HASPlings or later the Resource Manager or RM which was developed at the University of British Columbia to replace HASP), or allow the system operators to display status and otherwise control the system (JOBS, UNITS, STOP, BLAST, GOOSE, STARTUP, SHUTDOWN, REW, WTM, ...).

New jobs, other than the very first job, are started by requests to UMMPS from other jobs, most often the OPERATOR job. The very first job, INIT, is started immediately after IPL and supervisor initialization.

24, 31, and 32-bit addressing

From their start and for much of their lifetime UMMPS and MTS operated using 24-bit addressing. UMMPS never used the 32-bit virtual memory addresses that were available on the IBM S/360-67.[11]

In August 1982 the University of Alberta changed UMMPS to operate in 31-bit addressing mode to allow more than 16 MB of real memory to be used, although real memory above 16 MB was only used for to hold virtual memory pages. Job programs and user programs continued to use 24-bit addresses.

In 1985 Rensselaer Polytechnic Institute (RPI) made changes to UMMPS to support S/370-XA which among other things allowed either 24 or 31-bit addressing for job programs and for user programs running under MTS. Changes were made at the University of Michigan in 1990 to allow user programs using 31-bit addresses to work smoothly: object modules could be flagged as supporting 31-bit addressing (or not), compilers and assemblers were changed to supply the correct flags, programs would switch between 24 and 31-bit addressing modes as needed when transitioning between system and user modes.

Protection

MTS has a strong protection model that uses the virtual memory hardware and the S/360 and S/370 hardware's supervisor and problem states and via software divides problem state execution into system (privileged or unprotected) and user (protected or unprivileged) modes. Relatively little code runs in supervisor state. For example Device Support Routines (DSRs, aka device drivers) are not part of the supervisor and run in system mode in problem state rather than in supervisor state.[2] [12] [13]

Virtual memory and paging

Virtual Memory (VM) and demand paging support were added to UMMPS in November 1967, making MTS the first operating system to use the Data Address Translation (DAT) features that were added to the IBM S/360-67.[3]

UMMPS uses 4096 byte virtual memory pages and 256 page virtual memory segments. UMMPS could be conditionally assembled to use the small (64 page) segments that were available on S/370 hardware, but job programs were always presented with what appeared to be large (256 page) segments. Both 2K and 4K block storage keys are supported.

There is a three level storage hierarchy: (1) real memory, (2) high speed paging devices, and (3) paging disks. High speed paging devices include the IBM 2301 Drum, IBM 2305 Fixed Head File, and various third party "solid-state" I/O devices such as the STC 4305 and Intel 3805 that simulate spinning disks or more often provide more efficient fixed block architecture (FBA) access to external RAM based storage.[14] The high speed paging devices are attached using "two-byte" I/O channels operating at up to 3.0 MB per second whenever possible. The paging disks are separate from the disks used for the file system and are used if the higher speed paging devices become full. Virtual memory pages migrate between real memory and the paging devices, but not between individual paging devices. Later in its life the system was changed to use IBM S/370-XA Extended Storage as part of the second level of the storage hierarchy.

Virtual memory is managed by UMMPS with assistance from the Paging Device Processor (PDP) job program. UMMPS responds to requests to allocate and free VM from job programs, allocates VM addresses, allocates real memory, manages segment and page tables, sets storage keys, manages reference and change bits, determines which virtual memory pages should be paged in or out, and communicates with the PDP. New virtual memory pages are initialized to a "core constant" value of x'81' on first reference.

The PDP is a real memory job program. It allocates space on the paging devices, initiates all I/O to the paging devices, is responsible for recovery from I/O errors, and communicates with UMMPS.

To reduce the likelihood of thrashing UMMPS uses a "big job mechanism" that identifies jobs with more real pages than a threshold, limits the number of these "big" jobs that are eligible to execute at a given time, and gives the big jobs an extended time slice when they do execute. This allows big jobs to accumulate more real memory pages and to make better use of those pages before they come to time slice end, but big jobs will wait longer between time slices when there are too many big jobs contending for limited real memory pages. The number of pages that a job can have before it is considered big (the big job threshold or BJT) and the number of big jobs (NBJ) that are eligible for execution are external parameters that are reevaluated and set outside of the supervisor every 20 seconds based on the overall system load.

Other than the big job mechanism, UMMPS storage, processor, and I/O scheduling are independent, with each area allowed to "take care of itself".

Virtual memory is divided into regions as follows:

- Segment 0: shared virtual equals real memory (read-only)
- Segments 1 to 4: shared virtual memory (read-only)
- Segment 5: private virtual memory (system segment, only available to system mode (unprotected) programs)[2]
- Segments 6 and up: private virtual memory (user segments, read-write to any program)

Different numbers of segments were assigned to the various regions over time and with the advent of 31-bit addressing and the ability to use VM segments larger than 16, the regions were expanded as follows:

- Segment 0: shared virtual equals real memory (read-only)
- Segments 1 to 5: shared virtual memory (read-only)
- Segments 6-7: private virtual memory (system segments, only available to system mode (unprotected) programs)
- Segment 8: shared virtual memory for attachment of named address spaces (NASs) (read-only)
- Segments 9-55: private virtual memory (user segments, read-write to any program)
- Segments 56-59: private virtual memory (system segments, only available to system mode (unprotected) programs)
- Segments 60-63: shared virtual memory for attachment of named address spaces (NASs) (read-only)

Some real memory is not addressable using virtual memory addresses and so is only available to UMMPS or real memory job programs. Read-only virtual memory may be changed by privileged programs that turn memory protection off (usually for very limited periods of time).

Named address spaces (NASs) allow the attachment of named segments of virtual memory. They are shared virtual memory spaces that may be attached and detached from a given job's virtual address space and the same addresses may have different contents depending on which named address spaces are attached. NAS support is mostly used by MTS to attach VM segments preloaded with system components as a way to extend shared virtual memory without using VM address space below the magic 16 MB line and thus keeping more of this valuable address space available for use by 24-bit user programs.

Signon and project IDs

Everybody who uses MTS is assigned a signon ID (also called userids or Computing Center IDs, CCIDs).[4] Signon IDs are always 4 characters long. If necessary shorter IDs are automatically padded on the right using the string ".$.". Thus the IDs "MTS.", "DAB.", "ME$." or "C.$." could be written as "MTS", "DAB", "ME" and "C", respectively.

Signon IDs are protected using passwords which must be given at the start of each session (as part of or more often immediately after the **$SIGNON** command). Exceptions are jobs submitted via ***BATCH*** that run under the same ID that submitted the new job, jobs scheduled to run repeatedly at a particular time or on a particular day when the jobs run under the same ID that scheduled them, or jobs initiated from the operator's console. Passwords are from 1 to 12 characters long, lower case letters are converted to uppercase, special characters other than comma and blank are allowed. Passwords can be changed using the **$SET PW** command. Changing a password from a terminal

session requires entering the original password, and the new password must be entered twice for verification.

Entering an incorrect password is counted and reported to the user at the next successful signon. Too many password failures without a successful entry are reported to the system operator and still more password failures without a successful entry will cause the signon ID to be "locked out" until it is reset by business office staff. A short delay is introduced between failed password entry attempts to prevent large numbers of password "guesses" from being made quickly.

Individuals can have multiple sign on IDs for use in different courses, different research projects, or with different funding sources (university, government, non-profit, industry, ...). The sharing of signon IDs by individuals is discouraged, but does occur.

Signon IDs are grouped into projects. Each signon ID is a member of one and only one project. Project IDs, like signon IDs, are 4 characters long. Many projects are controlled by a "Project Leader" signon ID which can allocate resources to the accounts that are members of the project (within the resource limits allocated to the project) using the **$ACCOUNTING MANAGEMENT** command.

Signon and project IDs are also used to control access to files and to send e-mail.

With one exception there are no signon IDs with "special" privileges by virtue of the ID itself. Instead flags can be set that allow specific signon IDs to:

- create public files and set public program keys,
- run with a zero or negative account balance,
- perform privileged operations,[2] including:
  - flagging files to run in system (unprotected) rather than user (protected) mode by default,
  - use the **PROT=OFF** options on the **$SET** and **$RUN** commands,
  - use the test command language subsystem (**$#CLS**),
  - use privileged options of the **$SYSTEMSTATUS** and other command language subsystems (CLSs).

The exception is the signon ID "MTS.", which can read, but not modify or permit, any file in the system regardless of ownership or permit status. The MTS. ID can also use the **$SET FILEREF=OFF** option, which prevents the file reference dates on files from being updated (useful when recovering from file system problems or investigating security issues).

There is no ability for a program or user to assume the privileges of a signon ID other than the one that was used to sign on to the current session. Instead, programs and files may be permitted to specific signon IDs, projects, and program keys or to combinations of signon IDs, projects, and program keys.

Terminal, batch, and server sessions

MTS supports terminal, batch, and server sessions.[4] All three use the same command language.

- Terminal sessions are interactive with the user able to respond to the output produced including error messages and prompts.
- Batch jobs are not interactive and so all input needs to be prepared in advance with little or no opportunity for the user to alter the input (at least not without programming) once the batch job starts to execute.
- Server sessions can support user to MTS or client to MTS interactions and while there may be interaction with the user, MTS commands are usually read from a command file and the user is not likely to have to know or enter MTS commands. Server sessions can be sponsored in which case they will appear to be free to the user, and do not require that the user enter an ID and password. Server sessions can also be charged for and require a valid ID and password. Server sessions can be initiated from the network or from within an MTS session using the **$MOUNT** command.

The University of Alberta developed a Student Oriented Batch Facility in 1971 to provide quick job turnaround for undergrad students learning to program in FORTRAN, ALGOL, PL/C, and 360 Assembler. It was a dedicated punch

card input, printer output system that provided 5 minute turn around and ran several thousands of jobs a week at a fixed cost per job (15 cents).

Command language

MTS reads commands from the ***SOURCE*** pseudo device, which is initially the user's terminal or the batch input stream.[4] Programs may execute MTS commands by calling the CMD, CMDNOE, and COMMAND subroutines.[5]

Leading and trailing blanks as well as null and all blank lines are ignored. Lines that start with an asterisk (***** or **$***) are treated as comments. Command lines that end with a continuation character (by default the minus-sign) are continued on the next line. Command lines may be up to 255 characters long.

MTS uses keyword oriented commands and command options. The command verb (**SIGNON**, **RUN**, **EDIT**, ...) is the first keyword on the command line. Commands may start with an optional dollar-sign (**$SIGNON**, **$RUN**, **$EDIT**, ...). In batch jobs, following invalid commands and some other errors, MTS looks for the next line that starts with a dollar-sign (**$**) in column 1 as the next command to execute. All commands and most command options allow initial sub-string abbreviations (**C** for **COPY**, **R** for **RUN**, **DEB** for **DEBUG**, ...). MTS commands and most command options are case insensitive.

MTS has "one-shot" commands (**CREATE**, **FILESTATUS**, **SIGNOFF**, ...) and commands that have sub-command modes (**EDIT**, **CALC**, **SYSTEMSTATUS**, ...). Most commands with sub-command modes can also be invoked as one-shot commands by giving one or more sub-commands on the command line.

All MTS jobs start with a **SIGNON** command and most end with a **SIGNOFF** command. Commands may be stored in files and executed using the **SOURCE** command. Commands may be stored in signon-files (sigfiles) or project-signon-files (projectsigfiles) that are always executed immediately after the **SIGNON** command. The execution of sigfiles may be required (**SIGFILEATTN=OFF**) or optional (**SIGFILEATTN=ON**, the default).

Global control

```
SIGNON  { ccid | * }  [ option ... ]  [ comment ]
SIGNOFF  [ SHORT | $ | LONG ]  [ RECEIPTS | NORECEIPTS ]
ACCOUNTING  [ option ... ]
ACCOUNTING  MANAGEMENT
COMMENT  [ text ]
DISPLAY  item  [ OUTPUT=FDname ]
SET  option ...
SINK  [ FDname | PREVIOUS ]
SOURCE  [ FDname | PREVIOUS ]
SYSTEMSTATUS [ option ]
#CLS  FDname  [ options ]  (privileged command that runs a test CLS)[7]
```

File management

```
CREATE  filename  [ SIZE={ n | nP } ]  [MAXSIZE={n | nP} ]  [TYPE={LINE | SEQ | SEQWL} ]
DESTROY  filelist  [ OK | ALLOK | PROMPT ]
DUPLICATE  oldname  [ AS | TO ]  newname  [ options ] [ OK | ALLOK | PROMPT ]
EDIT  [ filename ]  [ :edit-command ]
EMPTY [ filelist ]  [ OK | ALLOK | PROMPT ]
TRUNCATE  filelist  [ ALLOK | PROMPT ]
RENAME  oldname  [ AS ]  newname  [ OK | ALLOK | PROMPT ]
RENUMBER  filelist  [ first [ last [ begin [ increment ] ] ] ]  [ ALLOK | PROMPT ]
FILESTATUS  [ filelist ]  [ format ]  [ items ]
```

```
FILEMENU  [ filelist ]  [ items ]
FMENU  [ filelist ]  [ items ]
PERMIT filelist  [ access [ accessor ] ]
PERMIT filelist  LIKE  filelist2  [ EXCEPT access [ accessor ] ]
LOCK filename  [ how ]  [ WAIT | NOWAIT ]  [ QUIT | NOQUIT ]
UNLOCK  filename
LOCKSTATUS  [ filename | JOB nnnnnn ]  [ LOCK ]  [ WAIT ]
LSTATUS  [ filename | JOB nnnnnn ]  [ LOCK ]  [ WAIT ]
```

File and device management

```
COPY  [ FROM ]  { FDlist1  | 'string' }  [ [ TO ] [ FDlist2 ]
CREATE  *pdn*  TYPE={ PRINT | IMPORT | EXPORT | DUMMY }
DESTROY *pdn*  [ OK | ALLOK | PROMPT ]
LIST  FDlist  [ [ ON | TO ]  FDname ]  [ [ WITH ] option ... ]
LIST  FDlist WITH  options [ {ON | TO} FDname ]
LIST
MOUNT  [ request [; request ] ... ]
CANCEL  *...* [ [ JOB ] nnnnnn ]  [ {ID | CCID}=ccid ]
RELEASE  { *PRINT* | *PUNCH* | *BATCH* | *pdn* }
LOCATE  { SYSTEM | LOCAL | FULL | SHORT | HELP }
LOCATE  { jobnumber | jobname }  [ option  ...  ]
VIEW  [ jobnumber [ ; view-command ] ]
LOG  [ FDname1 ]  { [ ON ] FDname2 [ format ]  [ options ] | OFF }
FTP  [ hostname ]
GET  FDname  (old fashioned and obsolete, but sometimes still useful)[7]
NUMBER  (old fashioned and obsolete way to enter data into a file)[7]
```

User program execution and control

```
RUN  [ FDname ] [ I/Ounits ]  [ option ] ... [ PAR=parameters ]
RERUN  [ ECHO | NOECHO ]  [ I/Ounits ]  [ option ] ...  [ PAR=parameters ]
DEBUG [ FDname ] [ I/Ounits ]  [ option ] ... [ PAR=parameters ]
SDS  [ sds-command ]
LOAD  [ FDname ] [ I/Ounits ]  [ option ] ... [ PAR=parameters ]
START  [ [ AT ]  [ RF={hhhhhh | GRx} ] location ]  [ I/Ounits ]  [ option ] ...
RESTART  [ [ AT ] location ]  [ I/Ounits ]  [ option ] ...
UNLOAD  [ CLS=clsname ]
ALTER  location value ...  ...
DISPLAY  [ format ]  location  [ OUTPUT=FDname ]
DUMP  [ format ]  [ OUTPUT=FDname ]
IF  RUNRC  condition  integer , MTS-command
ERRORDUMP (obsolete command, causes an  automatic dump in batch mode following abnormal termination of a user program)[7]
```

Miscellaneous

```
CALC  [ expression ]
MESSAGESYSTEM  [ message-command ]
FSMESSAGE  [ FSMessage-command ]
NET  [ host | *pdn* ]  [ .network-command ]
HEXADD  [ hexnumber1 ]  [ hexnumber2 ]  (obsolete, replaced by $Calc)[7]
HEXSUB  [ hexnumber1 ]  [ hexnumber2 ]  (obsolete, replaced by $Calc)[7]
PASSWORD  (obsolete, removed, allowed changes to public files
           before true shared file access was available)
```

File-name patterns

Several MTS commands that use file names or lists of file names allow the use of file-name patterns: **COPY**, **DESTROY**, **DUPLICATE**, **EMPTY**, **EDIT**, **FILESTATUS**, **FILEMENU**, **LIST**, **LOCKSTATUS**, **PERMIT**, **RENAME**, **RENUMBER**, and **TRUNCATE**. A question-mark (**?**) is the pattern match character. A single question-mark used in a file-name will match zero or more characters. "**?**" matches all files for the current signon ID, "**?.S**" matches all files that end with "**.S**", "**A?B**" matches all files that begin with "**A**" and end with "**B**", "**A?B?C**" matches all files that start with "**A**", end with "**C**", and contain a "**B**". Two or more consecutive question-marks match "*n*-1" characters. "**???.S**" matches all four character file-names that end with "**.S**", and "**????**" matches all three character files names. "**W163:?**" matches all files under the signon ID "**W163**" to which the current user has some access.

Command Macros

The MTS command macro processor allows users to define their own MTS commands.[15] It provides a "scripting" language with conditional commands and is available for use with any lines read from ***SOURCE*** by user programs or command language sub-systems as well with MTS commands. Macro processor lines are usually prefixed with the greater than character (**>**). The command macro processor is controlled using the **$SET** command as well as by I/O modifiers on FDnames.

Prefix Characters

To help users keep track of what command, command subsystem, or program they are working with and when input is expected, MTS displays a prefix character or sometimes a prefix string at the front of each input and output line it writes to the user's terminal. The common prefixes are:

```
#      MTS command mode
#-     MTS command continuation mode
?      Prompts
>      COPY and LIST commands
.      Program loader
blank  User programs
:      Editor
+      Symbolic Debugging System (SDS)
@      Message System
ftp>   FTP (File-Transfer)
```

Command language subsystems

The MTS job program is always executing one of several command language subsystems or CLSs. Many of the MTS commands are "built-in" to MTS and execute as part of the MTS CLS. User programs execute as the USER CLS. The USER CLS has a special relationship to the Symbolic Debugging System (SDS CLS) when the debugger is active. Other MTS commands are implemented as separate modules, confusingly also named command language subsystems or CLSs, that may be executed from shared virtual memory or may be loaded from files.

These separate CLSs each have their own four character name and they execute as a separate CLS in the original sense of the term. Many, but not all, of these CLSs provide their own separate sub-command language. There are **$SET** command options to cause old or new versions of CLSs rather than the current versions to be used. There is an option on the **$UNLOAD** command to unload a CLS (free the virtual memory it is using, close any FDnames and release any devices or pseudo devices that it has open).[7]

Only one CLS is executing at a time, but one CLS of each type may be active and it is possible to switch from one CLS to another without exiting or unloading the original CLS and then to later return to the original CLS and continue working from where one left off. CLSs that have their own sub-commands usually support a **STOP** command to exit from the CLS, an **MTS** and/or a **RETURN** command to return to the calling CLS or MTS command mode, and commands that begin with a dollar-sign (**$**) are executed as MTS commands with an immediate return to the original CLS.

All CLSs except for the USER CLS execute in system mode in problem state.

Limited-service state

MTS sessions normally operate in "full-service state", but during times of extreme system overload terminal sessions may be placed into "limited-service state" (LSS).[4] The LSS mechanism is manually enabled by the system operator and is normally only used when the hardware system is operating at reduced capacity due to a malfunction.

A terminal session is placed into LSS if LSS has been enabled by the system operator and the system is overloaded at signon. LSS sessions may only issue MTS commands and run programs with a short local time limit. Rather than giving all users poor performance, LSS limits the size of the tasks that some users may perform to relatively small tasks such as editing of files and reading of messages in order to allow other users to receive reasonable performance on larger tasks. Users may request that their session be changed to full-service state (**$SET LSS=OFF**) and such requests are granted if the system is not overloaded at the time the request is made.

Command statistics

Each MTS command that is issued is recorded, first to a disk file and later to magnetic tape. This information is only available to staff and is used to investigate software problems, security problems, rebate requests, and to provide statistics about how the command language is used.

User programs

User program refers to a program run by the user and which is not necessarily a program that belongs to or that was created by a user. User programs may be supplied in public files, in files available under the OLD: or NEW: signon IDs, in files belonging to other users and permitted for use by others, or user programs may be developed by the current user in files that they own.

User programs are executed using the **$RUN**, **$RERUN**, and **$DEBUG** commands or less often using the **$LOAD** and **$START** commands. The **$RESTART** command may be used to restart execution of a program following an attention interrupt that wasn't handled by the program, a program interrupt that wasn't handled by the program (although restarting after a program interrupt usually does not work well), or following an explicit return to MTS from a call to the MTS subroutine.

MTS loads programs using a dynamic linking loader (UMLOAD) that reads loader records (ESD, TXT, CSI, RDL, LCS, END, ...) from the file or device specified by the user and will selectively include subroutines from libraries supplied by the user, from system subroutine libraries such as ***LIBRARY**, and from system subroutines pre-loaded in shared virtual memory. MTS uses standard OS/360 loader records which makes it fairly easy for MTS to use compilers developed for use under other IBM operating systems.[3]

When a program starts execution a number of logical I/O units will be set either explicitly on the **$RUN** or other command or by default. Any text string given following the PAR= keyword is passed to the program as a parameter.

By default user programs execute with the program key ***EXEC**, but a different program key may be set using the **$CONTROL** command.[4] Programs may call a system subroutine to shorten the program key they are using or switch to the ***EXEC** program key thus temporary giving themselves less access to files, devices, and other services controlled using program keys. Programs may also call a system subroutine to lengthen or restore their program key according to some pre-established rules.

MTS uses the standard S-type and, less often, R-type calling sequences used in OS/360.[5]

By default user programs execute in user mode in problem state.[2] User mode programs do not have access to the system virtual memory segment and therefore have no access to system control blocks, may not call privileged system subroutines, and may not issue privileged supervisor calls (SVCs). User mode programs can issue non-privileged SVCs, but few programs do so directly and instead call system subroutines to obtain system services. User mode programs may call system subroutines that switch to system mode after checking that the protected service is allowed for the particular caller, there is a return to user mode when the system subroutine returns.

Selected user programs can be flagged to run in system rather than user mode by staff with privileged signon IDs or staff with privileges can cause a user program to run in system mode using a keyword on the **$RUN** or **$SET** command.[7]

Device independent input/output

All input/output requests, whether by the MTS job program itself or by a program running under MTS, is done using a common set of subroutine calls (GETFD, FREEFD, READ, WRITE, CONTROL, GDINFO, ATTNTRP, ...). The same subroutines are used no matter what program is doing the I/O and no matter what type of file or device is being used (typewriter or graphics terminal, line printer, card punch, disk file, magnetic and paper tape, etc.). No knowledge of the format or contents of system control blocks is required to use these subroutines. Programs may use specific characteristics of a particular device, but such programs will be somewhat less device independent.

MTS input/output is record or line oriented. Programs read lines from a terminal, card reader, disk file, or tape and write lines to a terminal, printer, disk file, or tape. Conversion to and from ASCII/EBCDIC and end-of-line processing is usually done by a front end processor or Device Support Routine (DSR) and so is not a concern of most programs. While it is possible to do character I/O to a terminal by reading or writing single character lines, reading or writing many such very short lines is not very efficient.

Each line read or written consists of from 0 to 32,767 bytes of data and an associated line number (a signed integer number scaled by 1000) giving the line's location. The length of each line read or written is given explicitly, so programs do not need to do their own processing of line ending characters (CR/LF, NL) or other terminators (null). Some devices support zero length lines, while others do not. For many files and devices the line number is simply a sequential count of the lines read, while some file types explicitly associate a specific line number with each line of the file, and in other cases the line number is synthesized from data that appears at the start of an input line or the line number can be prepended to an output line.

File or device names

Input/output is done directly by referencing a file or device by its name (FDname) or indirectly by referencing a logical I/O unit (**SCARDS** or **IN**PUT, **SPRINT** or **PR**INT, **SPUNCH** or **OBJ**ECT, **GUSER**, **SERCOM**, **0** to **99**). FDnames are assigned to logical I/O units using keywords in the command language or by default.

FDnames can be a simple file name such as **MYFILE**, a simple device name prefixed with a greater than sign such as **>T901**, or a pseudo-device name such as ***PRINT***. All FDnames are converted to uppercase before they are used, so like MTS commands, FDnames are case independent.

I/O modifiers, line number ranges, and explicit concatenation can be used to create complex FDnames from simple FDnames. For example:

```
FILE1@-TRIM  (I/O modifier that retains trailing blanks)
FILE2(1,10)  (line number range that reads lines from 1 to 10 inclusive)
FILE3+*SOURCE*  (explicit concatenation)
FILE4(1,10)@-TRIM+*TAPE*@-TRIM (all of the above in a single complex FDname)
```

Pseudo device names

Pseudo device names (PDNs) begin and end with an asterisk (e.g., *name*). Common pseudo devices include:

- ***SOURCE*** standard input (normally either a terminal or for batch jobs, the input queue);
- ***SINK*** standard output (normally a terminal or for batch jobs, a printer);
- ***MSOURCE*** master source, not re-assignable, usually a terminal or a card reader;
- ***MSINK*** master sink, not re-assignable, usually a terminal or a printer;
- ***BATCH*** spooled input to a new batch job;
- ***PRINT*** spooled output to a printer, same as *MSINK* for batch jobs;
- ***PUNCH*** spooled output to a card punch (until card punches were retired); and
- ***DUMMY*** all data written is discarded and all reads return an End-of-File (much like /dev/null for UNIX); and
- ***AFD*** the active file or device as established using the **$GET** command.[7]

The **$SOURCE** and **$SINK** commands may be used to reassign the FDnames assigned to ***SOURCE*** and ***SINK***. The **$MOUNT** command assigns pseudo device names (e.g. *T22*, *NET*) to devices such as magnetic and paper tapes and network connections (including server connections). The **$CREATE** command can be used to create pseudo device names for use with BITNET import and export, for spooled print jobs, and for dummy devices.

I/O modifiers

I/O modifiers, possibly negated, may be associated with an FDname to modify default behaviors.

An I/O modifier is specified by appending an at-sign followed by the modifer's name to an FDname. For example ***SOURCE*@UC** would cause lines read from ***SOURCE*** to be converted to uppercase before they are presented to a program and **MYFILE@UC@-TRIM** would cause lines read from the file MYFILE to be converted to uppercase and any trailing spaces at the end of the line would be retained. Some of the commonly used I/O modifiers are: @S (sequential), @I (indexed), @FWD (forward), @BKWD (backward), @EBCD (EBCDIC), @BIN (binary), @UC (uppercase), @CC (logical carriage control), @MCC (machine carriage control), @NOCC (no carriage control), @TRIM (trim all but last training blank). Some I/O modifers are processed in a device independent fashion by MTS and others are device dependent and processed by the Device Support Routines (DSRs).

Not all files or devices support all I/O modifiers. Different files and devices have different default I/O modifiers and a few I/O modifier defaults can be changed using the **$SET** command.

Line number ranges

Specific parts of a file or device can be referenced by including starting and ending line numbers and possibly a line number increment in parenthesis separated by commas. The line numbers and increment are integers scaled by 1000 and can be positive or negative (±*nnnnn.nnn*). For example **SIMPLE.F(-35,197.5)** would open the file **SIMPLE.F**, starting at the first line number greater or equal to -35 and return an 'end of file' instead of the first line number greater than 197.5. One can also include line number increments—as an example: **SIMPLE.F(2,200,2)** would return all (and only) even line numbers between 2 and 200 (inclusive).

The symbolic line numbers **FIRST** or ***F**, **LAST** or ***L**, **MIN**, and **MAX** may be used to refer to the first, last, minimum possible, and maximum possible lines, respectively. For example **SIMPLE.F(*F,0)** would refer to the 'negative' lines of the file **SIMPLE.F**. This is where programmers might place self-documentation for a (often binary) file, actual data in the file would start at line number 1.

One can also do simple addition and subtraction with the symbolic line numbers: **FIRST±*m***, ***F±*m***, **LAST±*m***, ***L±*m***, **MIN+*m***, **MAX-*m***, where *m* is an integer with or without a decimal point scaled by 1000 (±*nnnnn.nnn*). So to add new lines to the end of an existing file one could use an FDname of the form **SIMPLE.F(LAST+1)**.

File or device concatenation

Explicit concatenation allows FDnames to be connected using a plus-sign, as *NAMEA*+*NAMEB*. In this case MTS transparently returns the contents of *NAMEA* followed by the contents of *NAMEB* or writes to *NAMEB* after writing to *NAMEA* reaches an end of file or other error condition.

Implicit concatenation occurs when an input line contains the string:

```
$CONTINUE WITH FDname
```

MTS will continue with the FDname given as the new source of data. Or, if a line of the form:

```
$CONTINUE WITH FDname RETURN
```

is read, MTS will return the contents of the new FDname until and End-of-File is reached and then return the next line of the original FDname (note that, a file that continues with itself causes an infinite loop, usually a mistake, but sometimes used to good effect).

While the line starts with a dollar-sign, **$CONTINUE WITH** is not an MTS command, but rather a delimiter.

The **@IC** I/O modifier and the command **$SET IC={ON | OFF}** can be used to control implicit concatenation.

$ENDFILE lines

If a line contains the string **$ENDFILE**, MTS returns a 'soft' end of file.

While the line starts with a dollar-sign, **$ENDFILE** is not an MTS command, but rather a delimiter.

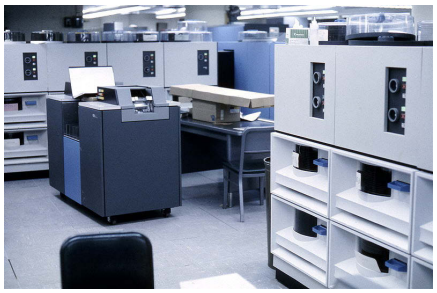
The **@ENDFILE** I/O modifier and the command **$SET ENDFILE={ALWAYS | SOURCE | NEVER}** can be used to control **$ENDFILE** processing.

$9700 and $9700CONTROL lines

Lines that begin with the strings "**$9700**" or "**$9700CONTROL**" may be copied or written to ***PRINT*** to control print options on the Xerox 9700 page printer. **$9700** lines take effect at the point where they occur, while **$9700CONTROL** lines apply to the entire print job in which they occur. While these lines have a form similar to MTS commands, they are really device commands and not true MTS commands.

Files

MTS files are stored as 4096 byte "pages" on one or more public or private disk volumes.[16] Volumes have volume labels, volume numbers, and volume names (usually MTS001, MTS002, ..., MTSnnn). Disk volumes are stored on traditional cylinder-track-record and fixed block architecture (FBA) disk drives or at one time on the IBM 2321 Data Cell.



IBM 2314 disk drives and IBM 2540 card reader/punch at the University of Michigan Computing Center, c. 1968

Individual files do not span disk volumes. The maximum size of a file is limited to the free space available on the disk volume where it resides. By default files are created one page in size, but a larger size as well as a maximum size may be specified (**$CREATE *name* SIZE=*n*P MAXSIZE=*n*P**). Files will automatically expand until they reach their maximum size or the disk space limit for the owner's signon ID is exceeded. Users may request that a file be created on a specific disk volume (**$CREATE *name* VOLUME=*name***).

IBM 2321 data cell at the University of Michigan Computing Center, c. 1968

MTS files fall into one of three categories: **public files**, **user files**, and **temporary files**:

- **Public files** are files whose names begin, but do not end, with an asterisk (e.g., ***LIBRARY**, ***USERDIRECTORY**). Public files, often called 'star files', are publicly available files that contain programs and data that are widely available to all users. For example, ***LIBRARY** is a library of commonly used system subroutines. In the earliest days of MTS public files were the only files that could be shared and then only as read-only files. Later, public files could be permitted and shared in the same fashion as any other files.
- **User files** are files whose names do not begin with an asterisk or a minus sign. They must be explicitly created (**$CREATE**) and destroyed (**$DESTROY**). They are owned by and initially permitted to just the userID that creates them, but they can be permitted for use by other userIDs using the **$PERMIT** command. To reference a file belonging to another user, the file name is prefixed with the owner's userID followed by a colon (e.g., **W163:MYPROGRAM**). There are charges for the amount of disk space used and most signon IDs have a maximum disk space limit.
- **Temporary files** are files whose names begin with a minus sign (e.g., **-TEMP**). Their names are unique within a single session. They are created implicitly on first use, are not charged for, do not count against a signon ID's disk space limit, and are automatically destroyed when the terminal or batch session ends.

MTS doesn't implement directories, but there is a *de facto* two-tier grouping of files owing to the inclusion in a file's name of its owner's four-character MTS user ID.

File names, like all FDnames, are converted to uppercase before use and so are case insensitive.

File types

MTS supports three types of file, line files, sequential files, and sequential with line number files, but line files were by far the most common:

Line files

Line files (**$CREATE *name*** or **$CREATE *name* TYPE=LINE**) are line-oriented files which are indexed (and randomly accessible) by line number. Allowed line numbers are ±2147483.647—essentially a signed integer value divided by 1000, but command line references were limited to ±99999,999. Regular writes to a file increase the line number by 1. Lines are variable length, and a line can be rewritten to any length between 1 and the line length limit (originally 256, but later changed to 32767) without affecting the lines around it. Rewriting a pre-existing line to a length of zero deletes that line without affecting surrounding lines.

By default the first line number written to an empty file is 1, and is incremented by 1 with each subsequent write. By default, reading a file starts with the first line number at or above 1 and continues by reading each line in order of increasing line numbers. This means that negative line numbers are 'invisible' parts of a file, which require specific references to read.

There are commands (and system subroutines) to renumber lines. A contiguous set of lines can be renumbered to any combination of start and increment as long as lines of the file are not re-ordered. For example, if a file consists of lines 10, 20, 30, 40 and 50, lines 30–40 can be renumbered as 35,36, but not as 135,136, as that would change the sequence of lines.

The line index and data are stored on separate disk pages with the exception of the smallest (one page) files, where the line index and data are stored together.

The **$CREATE** command creates line files by default.

A side-effect of the line-based file system is that programs can read and write individual lines incrementally. If one edits a file (usually a text file) with the MTS file editor (**$EDIT**), any changes made to lines are written immediately, as are insertions and deletions of specific lines. This makes it quite different than most (byte-oriented) file systems where a file is usually read into and changed in memory, and then saved to disk in bulk.

Due to hardware or software problems, line files can become corrupt. The program ***VALIDATEFILE** checks the structure of line files.

Sequential files

Sequential files (**$CREATE *name* TYPE=SEQ**) are line-oriented files with the first line number being implicitly 1 and incremented by 1 for each line. Once written the length of a line (other than the last line of a file) can not be changed, although any line can be replaced by a line of the same length. Sequential files are generally only readable sequentially from start to end, or written by appending to the end. One can, however, request a reference for the current line of a sequential file, and use that reference to jump to that specific location again.

Sequential files are somewhat more efficient in terms of space than line files and can be more efficient in terms of CPU time too when compared with large disorganized line files. But the main reason for the existence of SEQ files is that they supported long lines (up to 32767 characters) before line files did. Sequential files were less common once line files could support long lines. Sequential files are also used to force new lines to be appended to the end of the file without the need to give the line number range (LAST+1).

Sequential with line number files

Sequential With Line Number files (**$CREATE *name* TYPE=SEQWL**) are similar to Sequential Files, except that their line numbers were explicitly stored. They have all of the restrictions of Sequential Files, except that the line number could be specifically supplied when writing to a file (as long as it is greater than the last line number written to the file). Unlike Line Files, the first read of an SEQWL file returns the first line of the file, even if it was negative.

SEQWL files were rarely used, and were not officially supported or were removed from the documentation by some MTS sites. Because line files did not work well with the Data Cell, SEQWL files were implemented as a way to allow the Data Cell to be used for longer term less expensive storage of files while still preserving line numbers.

File save and restore

Files are regularly backed up to tape unless they have been marked as **NOSAVE**. The file save process includes full and partial backups. Full saves are typically done once a week with no users signed on to the system. Partial saves save just the files that have changed since the last full or partial save and are typically done once each day in the late evening or early morning during normal operation with users signed on to the system.

At the University of Michigan two copies of the full save tapes were made and one copy was stored "off-site". Save tapes were kept for six weeks and then reused. The tapes from every sixth full save were kept "forever".

Files are saved to allow recovery from "disk disasters" in which the file system becomes damaged or corrupt, usually due to a hardware failure. But users could restore individual files as well using the program ***RESTORE**.

Terminal support

At its peak, MTS at the University of Michigan simultaneously supported more than 600 terminal sessions as well as several batch jobs.[4]

An early computer terminal, the Teletype Model ASR-33 with attached paper tape reader/punch

Terminals are attached to MTS over dial-in modems, leased or dedicated data circuits, and network connections. The Michigan Communications Protocol (MCP), a simple framing protocol for use with asynchronous connections that provides error detection and retransmission, was developed to improve the reliability of terminal to MTS and computer to MTS connections.[17]

A very wide range of terminals are supported including the 10 character per second (cps) Teletype, the 30 cps LA-36 and 120 cps LA-120 DECWriter, the 14 cps IBM 2741, and at ever increasing speeds up to 56,000 bits per second, the VT100 display, the Visual 550 display, the Ontel OP-1 and OP-1/R displays, Tektronix 4000 series of graphic displays, and personal computers from Apple (AMIE for the Apple][), IBM (PCTie for DOS), and others running terminal emulation programs, including some specifically developed for use with MTS. Most terminals that are compatible with any of these models are also supported.

A DEC VT100 display terminal

MTS also supports access from 10- or 12-button touch-tone telephones via the IBM 7772 Audio Response Unit[18] [19] and later the Votrax Audio Response Unit,[20] [21] IBM 1052 consoles, IBM 3066 console displays, and IBM 3270 family of locally attached displays (IBM 3272 and 3274 control units, but not remote 3270 displays).

PDP-8 Data Concentrator at the University of Michigan, c. 1971

A Tektronix 4014 display terminal

Touch-tone Telephone

Front-end communication processors

MTS can and does use communication controllers such as the IBM 2703 and the Memorex 1270 to support dial-in terminals and remote batch stations over dial-in and dedicated data circuits, but these controllers proved to be fairly inflexible and unsatisfactory for connecting large numbers of diverse terminals and later personal computers running terminal emulation software at ever higher data rates. Most MTS sites choose to build their own front-end processors or to use a front-end processor developed by one of the other MTS sites to provide terminal support.

These front-end processors, usually DEC PDP-8, PDP-11, or LSI-11 based with locally developed custom hardware and software, would act as IBM control units attached to the IBM input/output channels on one side and to modems and phone lines on the other. At the University of Michigan the front-end processor was known as the Data Concentrator (DC).[22] The DC was developed as part of the CONCOMP project by Dave Mills and others and was the first non-IBM device developed for attachment to an IBM I/O Channel.[23] Initially a PDP-8 based system, the DC was upgraded to use PDP-11 hardware and a Remote Data Concentrator (RDC) was developed that used LSI-11 hardware that connected back to a DC over a synchronous data circuit. The University of British Columbia (UBC) developed two PDP-11 based systems: the Host Interface Machine (HIM) and the Network Interface Machine (NIM). The University of Alberta used a PDP-11 based Front-end processor.

These front-end systems support their own command language of "device commands", usually lines prefixed with a special character such as a percent-sign (%), to allow the user to configure and control the connections.[24] The **$CONTROL** command and programs running on MTS can use the CONTROL subroutine to issue device commands to front-end and network control units.

Network support

Over time some of the front-ends evolved to provide true network support rather than just providing support for connections to MTS.

At the University of Michigan (UM) and Wayne State University (WSU) there was a parallel development effort by the Merit Network to develop network support. The Merit nodes were PDP-11 based and used custom hardware and software to provide host to host interactive connections between MTS systems and between MTS and the CDC SCOPE/HUSTLER system at Michigan State University (MSU). The

Merit PDP-11 based Primary Communications Processor (PCP) at the University of Michigan, c. 1975

IBM 3279 display terminal

Merit nodes were known as Communication Computers (CCs) and acted as IBM Control Units on the one side while providing links to other CCs on the other side. The initial host to host interactive connections were supplemented a bit later by terminal to host (TL) connections, and later still by host to host batch connections which allowed remote jobs submitted from one system to be executed (EX) on another with printed (PR) and punched card output (PU) returned to the submitting system or to another host on the network. The remote batch jobs could be submitted from a real card reader or via ***BATCH*** using a **#NET** "card" at the front of the job.

Merit renamed its Communication Computers to be Primary Communication Processors (PCPs) and created LSI-11 based Secondary Communication Processors (SCPs). PCPs formed the core of the network and were attached to each other over Ethernet and dedicated synchronous data circuits. SCPs were attached to PCPs over synchronous data circuits. PCPs and SCPs would eventually include Ethernet interfaces and support local area network (LAN) attachments. PCPs would also serve as gateways to commercial networks such as GTE's Telenet (later SprintNet), Tymnet, and ADP's Autonet, providing national and international network access to MTS. Later still the PCPs provided gateway services to the TCP/IP networks that became today's Internet.

The Merit PCPs and SCPs eventually replaced the Data Concentrators and Remote Data Concentrators at the University of Michigan. At their peak there were more than 300 Merit PCPs and SCPs installed, supporting more than 10,000 terminal ports.

Virtual environments

UMMPS provides facilities that allow the creation of virtual environments, either virtual machines or virtual operating systems. Both are implemented as user programs that run under MTS.

The initial work on the first MTS virtual machine was done at the University of Michigan to simulate the IBM S/360-67 and allow debugging of UMMPS and MTS. Later the University of British Columbia did the initial work to create a S/370 MTS virtual machine. In theory these virtual machines could be used to run any S/360 or S/370 system, but in practice the virtual machines were only used to debug MTS and so there may be subtle features that are not used by MTS that are not completely or correctly implemented. The MTS virtual machine was never updated to support the S/370-XA architecture (instead other tools such as SWAT and PEEK were used to debug MTS and IBM's VM/XA or VM/ESA were used to debug UMMPS).

In the early 1970s work was done at Wayne State University to run a version of OS/MVT in a modified virtual machine (VOS) under MTS as a production service.[25]

"Student" virtual machines in MTS have also been created as teaching tools. Here the OS running in the virtual machine (written by the student) uses simulated devices and has no connection to the "real" outside world at all (except possibly a console).

In addition to virtual machines, MTS provides two programs that implement virtual operating system environments.[26] ***FAKEOS**, developed at the University of Michigan, allows programs from OS/360 to run as user programs in MTS. ***VSS**, developed at the University of British Columbia, allows programs from OS/VS1 and MVS/370 to run as user programs in MTS.[27] Neither program actually runs the IBM operating system, instead they simulate enough of the operating environment to allow individual programs developed for those operating systems to run. Both programs can be run directly, but often they are run from driver files that give an end user the impression that they are running a regular MTS user program.

Electronic mail

At least three different implementations of e-mail were available under MTS at different times:

- ***MAIL** from NUMAC, but not available at all MTS sites;
- **CONFER**, the computer conferencing system written by Robert Parnes at UM; and
- **$MESSAGESYSTEM** from the University of Michigan Computing Center.[4] [28]

CONFER and ***MAIL** only sent and received mail to and from "local" users.

Available to users in July 1981,[29] **$MESSAGESYSTEM** is the last of the three systems to be implemented and became the most widely used. Between 1981 and 1993 it was used to send and receive more than 18 million messages at the University of Michigan.[30] It can send:

- local and network e-mail messages,
- dispatches (immediate messages displayed at another user's terminal unless dispatches were blocked by the other user),
- bulletins (messages sent by the system operator to particular users delivered automatically at the beginning of an MTS session), and
- signon messages (messages sent by the system operator to all users delivered automatically before the start of an MTS session).

Some notable features of **$MESSAGESYSTEM** include the ability:

- to send to individuals by signon ID or name, to groups of individuals by signon ID, project ID, or group name, or to the system operator;
- to send to a list stored in a file;
- to use the program ***USERDIRECTORY** to create and maintain a database of e-mail names for individuals and for groups including names and groups that include remote or network users;
- to recall/delete messages that hadn't already been read;
- to add or remove recipients to messages after they had been sent;
- to display a history of messages in an e-mail chain without the need to include the text from older messages in each new message;
- to set expiration and hold until dates and times for e-mail messages;
- to display the status of incoming and outgoing messages;
- to retrieve incoming and outgoing messages using a database model (incoming, outgoing, new, old/seen, to *recipients*, from *recipients*, message number, date sent, expiration date, ...);
- to permit a mailbox allowing uses by signon IDs other than the mailbox owner's;
- to automatically forward messages from one mailbox to another;
- to archive older messages, and
- send and receive messages using a subroutine interface in addition to commands.

An application for the Apple Macintosh, **InfoX** (aka MacHost), was developed to provide a modern interface to the MTS Message System and ***USERDIRECTORY**.

In 1984 MTS could be used to send and receive remote e-mail to and from over 300 sites around the world.[4]

The first ability to send and receive e-mail messages to and from users on remote systems (remote messages or network mail) was implemented in 1982[29] as part of the MAILNET project, a joint effort of 16 universities and EDUCOM (later EDUCAUSE) supported with funding from the Carnegie Corporation. MIT served as a relay hub between the MAILNET sites and as a gateway to CSNET, ARPANET, and BITNET. MTS at the University of Michigan used its connections to the Merit Network and through Merit to GTE's commercial X.25 network, Telenet (later SprintNet), to communicate with MIT. MTS at the University of Michigan served as a relay site for other sites on the UM campus and for other MTS sites that did not have direct access to the MAILNET relay at MIT.

The remote e-mail addresses for an MTS user at the University of Michigan were:

- *name*@UMich-MTS.Mailnet (from MAILNET and BITNET sites)
- *name*%UMich-MTS.Mailnet@MIT-MULTICS.ARPA (from CSNET and ARPANET sites)
- *name*@UM (from other UM or MTS sites)

To send e-mail to a remote site, MTS users at the University of Michigan used addresses of the form:

- *name*@CARNEGIE (to Carnegie-Mellion University a MAILNET site)
- *name*@CARNEGIE.MAILNET (the more official, but longer name for CMU)
- *name*@WSU (to Wayne State University an MTS site)
- *name*@Wayne-MTS.Mailnet (the more official but longer name for WSU)
- *name*%brown@CSNET-RELAY.ARPA (to Brown University a CSNET Phonenet site)
- *name*@cornell.ARPA (to Cornell University a CSNET or ARPANET site)
- *name*@STANFORD.BITNET (to Stanford University a BITNET site)

Over time as more and more computers had direct connections to the Internet the MAILNET relay approach was replaced with the more direct and more reliable peer to peer e-mail delivery and Internet domain style of e-mail addresses in use today (*name*@um.cc.umich.edu).

Accounting and charging

Each signon ID is allocated resource limits (money, disk space, connect time, ...) which control the amount and types of work that can be done by the ID.[4] IDs can be limited to using just terminal sessions or just batch jobs or restricted to working during times of the day or days of the week when the rates charged are lower. Each signon ID is assigned an expiration date.

Resources that can be charged for include:

- CPU time—charged in seconds of CPU time
- Memory usage—charged as CPU-VM integral ... e.g. 40 pages of virtual memory used for 10 seconds is charged as 400 page-seconds
- Printer usage—charged as pages of paper and lines of output (for line printers) or pages and sheets (for page printers)
- Disk space used—charged in page-months (one page=4096 bytes)
- Terminal or network connect time-charged in minutes
- Cards read and punched-charged by the card
- Paper tape punched-charged by the foot
- Tapes mounted and tape drive usage time-charged by number of tapes mounted and minutes of usage
- Program product surcharges (charged on a program by program basis for certain licensed program products)
- Other resources (e.g. plotters, photo-typesetters, etc.)

Note that while there is a charge for virtual memory used, there is no charge for real memory used. Note too that there is no change for page-in operations, although they are included in the session summary information that is reported at sign off.

Different rates can be changed for different classes of projects (internal, external, commercial, ...) and for different times of the day or days of the week. Depending on the policies and practices at different sites, charges can be for "real money" or "soft money" (soft money is sometimes called "funny money", although just how funny it is usually depends on who is or isn't paying the bills).

Users can display the cost of a session using the **$DISPLAY COST** command, can display their account balances using the **$ACCOUNTING** command, and the costs of a session and the account's remaining balance are displayed the job or session ends. There is also an option (**$SET COST=ON**) that will cause the incremental and cumulative session cost to be displayed after each MTS command is executed.

To prevent a user from overdrawing their account, the money limit is checked when the user attempts to sign on. If the account balance is zero or negative, the sign on is not allowed. For batch jobs, if the account balance is not sufficient to cover the charges estimated for the job, the job is not run. For terminal sessions, when an account's balance falls below one dollar, a warning "You have run out of money" followed by the current balance is printed. This "out of money" message is repeated at regular intervals until the user signs off. Signon IDs can run a negative balance, but usually not a large one or by accident. Depending on the administrative policies at a particular site, projects often have to pay for resources used even if they are beyond the amount authorized.

To provide additional protection against accidents that might quickly use more resources than desired, users may also set global and local limits on CPU time usage. Global time limits (**$SIGNON *ccid* T=*maxtime***) apply to an entire job or session. Local time limits apply to running individual programs (**$RUN *program* T=*maxtime***). Global and local limits on the number of pages to be printed and the number of cards to be punched can also be set (**$SIGNON *ccid* P=*maxpages* C=*maxcards*** and **$RUN *program* P=*maxpages* C=*maxcards***). A default local CPU time limit can be established using the **$SET TIME=*maxtime*** command.

References

[1] MTS Lecture 1 (http://archive.michigan-terminal-system.org/discussions/figure-1-from-mts-lecture-1/1original/mts-lecture-1-original), a transcription of the first in a series of lectures on the internals of the Michigan Terminal System given by Mike Alexander, Don Boettner, Jim Hamilton, and Doug Smith, c. 1972

[2] "The Protection of Information in a General Purpose Time-Sharing Environment" (http://archive.michigan-terminal-system.org/documents/protection-1977.pdf?attredirects=0&d=1), Gary C. Pirkola and John Sanguinetti, *Proceedings of the IEEE Symposium on Trends and Applications 1977: Computer Security and Integrity*, vol. 10 no. 4, , pp. 106-114

[3] "Organization and features of the Michigan Terminal System" (http://www.computer.org/portal/web/csdl/doi/10.1109/AFIPS.1972.80), M. T. Alexander, p. 586, *Proceedings of the May 1972 AFIPS Spring Joint Computer Conference*

[4] MTS Volume 1: *The Michigan Terminal System* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), pages 3,11-14, May 1984 and Nov. 1991, University of Michigan Computing Center, Ann Arbor, Michigan

[5] MTS Volume 3: *System Subroutine Descriptions* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), February 1983, University of Michigan Computing Center, Ann Arbor, Michigan

[6] Michigan Terminal System (MTS) subseries (http://quod.lib.umich.edu/cgi/f/findaid/findaid-idx?c=bhlead;idno=umich-bhl-0351;view=reslist;didno=umich-bhl-0351;subview=standard;focusrgn=C02;cc=bhlead;byte=24474999), Computing Center publications, 1965-1999, Bentley Historical Library, University of Michigan

[7] *MTS Volume 1: Systems Edition, Obsolete and Internal MTS Commands* (http://archive.michigan-terminal-system.org/documents/MTSVolume1-SystemsEdition-1991.pdf), November 1991, University of Michigan, 60pp.

[8] "Time Sharing Supervisor Programs" (http://archive.michigan-terminal-system.org/documents/timeSharingSupervisorPrograms-1971.pdf?attredirects=0&d=1), notes comparing the supervisor programs of CP-67, TSS/360, MTS, and Multics by Michael T. Alexander, *Advanced Topics in Systems Programming* (1970, revised 1971), University of Michigan Engineering Summer Conference

[9] UMMPS D6.0 Supervisor Call Descriptions (http://archive.michigan-terminal-system.org/manuals-and-documentation/5-internal-system-documentation/SupervisorCallDescriptions-November1987.pdf?attredirects=0&d=1), November 1987, University of Michigan, 156p.

[10] MTS Volume 14: *360/370 Assemblers in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), May 1983, University of Michigan Computing Center, Ann Arbor, Michigan

[11] Recollections of MTS developers bolstered by reviews of the update logs included in the MTS distributions, in MTS (Michigan Terminal System), 1968-1996 (http://quod.lib.umich.edu/cgi/f/findaid/findaid-idx?c=bhlead;idno=umich-bhl-9551;view=reslist;didno=umich-bhl-9551;subview=standard;focusrgn=C01;cc=bhlead;byte=24634866), Computing Center (University of Michigan) records 1952-1996, Bentley Historical Library, University of Michigan

[12] "The use of the monitor call instruction to implement domain switching in the IBM 370 architecture" (http://doi.acm.org/10.1145/850716.850719), John Sanguinetti, University of Michigan Computing Center, *ACM SIGOPS Operating Systems Review*, Volume 15, Issue 4 (October 1981), pp.55-61

[13] "A penetration analysis of the Michigan Terminal System" (http://doi.acm.org/10.1145/850693.850694), B. Hebbard, P. Grosso, et. al., *ACM SIGOPS Operating Systems Review*, Volume 14 , Issue 1 (January 1980), pp.7-20

[14] "The effects of solid state paging devices in a large time-sharing system" (http://doi.acm.org/10.1145/1010629.805484), John Sanguinetti, University of Michigan Computing Center, *ACM SIGMETRICS Performance Evaluation Review*, Volume 10, Issue 3 (Fall 1981), pp. 136–153, ISSN 0163-5999

[15] MTS Volume 21: *MTS Command Extensions and Macros* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), February 1991, University of Michigan Computing Center, Ann Arbor, Michigan

[16] "A file system for a general-purpose time-sharing environment" (http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1451786), G. C. Pirkola, *Proceedings of the IEEE*, June 1975, volume 63 no. 6, pp. 918–924, ISSN 0018-9219

[17] MTS Volume 4: *Terminals and Networks in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center

[18] *The audio response unit user's guide* (http://deepblue.lib.umich.edu/bitstream/2027.42/7659/5/bad2224.0001.001.pdf), Douglas B. Smith, CONCOMP Project, University of Michigan, 1970

[19] "Voice Output from IBM System/360" (http://doi.ieeecomputersociety.org/10.1109/AFIPS.1965.120), A. B. Urquhart, IBM, afips, pp.857, *Proceedings of the Fall Joint Computer Conference*, 1965

[20] "The University of Michigan Audio Response System and Speech Synthesis Facility", Edward J. Fronczak, *Second USA Japan Computer Conference, Proceedings, pp. 380-84, 1975*

[21] "Internal Design of the University of Michigan Audio Response System for the Generation of Segmental Phonemes from Text", Edward J. Fronczak and James F. Blinn, *Proceedings of the International Computer Symposium 1975*, pp. 404-10, Vol. 1, August 1975

[22] The Data Concentrator (http://www.eecis.udel.edu/~mills/gallery/gallery7.html), a special-purpose peripheral device for the attachment of interactive terminals to the System/360 Model 67, overview and photos from Dave Mills, the project leader and chief designer during its development

[23] *The Data Concentrator* (http://deepblue.lib.umich.edu/bitstream/2027.42/6635/5/bac9430.0001.001.pdf), David L. Mills, CONCOMP Project, University of Michigan, 1968

[24] *Data Concentrator User's Guide* (http://deepblue.lib.umich.edu/bitstream/2027.42/6644/5/bac9432.0001.001.pdf), David L. Mills, Jack L. Di Giuseppe, and W. Scott Gerstenberger, CONCOMP Project, University of Michigan, April 1970

[25] "An efficient virtual machine implementation" (http://doi.ieeecomputersociety.org/10.1109/AFIPS.1973.57), R. J. Srodawa and L. A. Bates, Wayne State University, afips, pp. 301, *Proceedings of the National Computer Conference*, 1973

[26] MTS Volume 2: *Public File Descriptions* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan

[27] MSC/NASTRAN is an early example, perhaps too early, of the use of *VSS, see *MSC/NASTRAN at the University of Michigan* (http://www.mscsoftware.com/support/library/conf/wuc82/p00282.pdf), William J. Anderson and Robert E. Sandstorm, 1982, University of Michigan College of Engineering

[28] MTS Volume 23: *Messaging and Conferencing in MTS* (http://archive.michigan-terminal-system.org/manuals-and-documentation/documentation), University of Michigan Computing Center, Ann Arbor, Michigan

[29] "MTS Timeline" (http://www.clock.org/~jss/work/mts/timeline.html), *Information Technology Digest*, University of Michigan, pp.9-10, Volume 5, No. 5 (May 13, 1966)

[30] Letter dated January 12, 1993 from James J. Duderstadt, President of the University of Michigan, to Jim Sterken, former UM Computing Center staff member and primary author of the MTS Message System (http://archive.michigan-terminal-system.org/documents/JJDThankYouToJJS-12Jan1993.pdf?attredirects=0&d=1)

IBM System/360 Model 67

Designer	IBM
Bits	32-bit
Introduced	1967
Design	CISC
Type	Register-Register, Register-Memory, Memory-Memory
Encoding	Variable (2, 4, or 6 bytes long)
Branching	Condition code
Endianness	Big
Page size	4K byte pages
Extensions	Data Address Translation, 24 or 32-bit addresses, memory reference and change bits
Registers	
General purpose	16 32-bit
Floating point	4 64-bit

The **IBM System/360 Model 67** (**S/360-67**) was an important IBM mainframe model in the late 1960s.[1] Unlike the rest of the S/360 series, it included features to facilitate time-sharing applications, notably a DAT box to support virtual memory and 32-bit addressing. The S/360-67 was otherwise compatible with the rest of the S/360 series.

IBM System/360 Model 67-2 (duplex) at the University of Michigan, c. 1969

Origins

The S/360-67 was intended to satisfy the needs of key time-sharing customers, notably MIT (where Project MAC had become a notorious IBM sales failure), the University of Michigan, General Motors, Bell Labs, Princeton University, and the Carnegie Institute of Technology (later Carnegie Mellon University).[2]

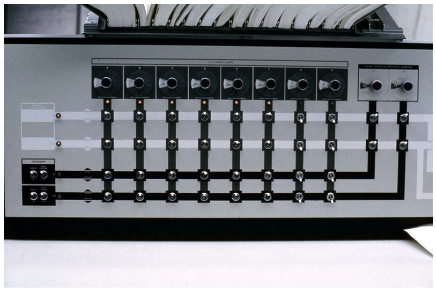
2167 configuration console for the IBM System/360 Model 67-2 (duplex) at the University of Michigan, c. 1969

In the mid-1960s a number of organizations were interested in offering interactive computing services using time-sharing.[3] At that time the work that computers could perform was limited by their lack of real memory storage capacity. When IBM introduced its System/360 family of computers in the mid-1960s, it did not provide a solution for this limitation and within IBM there were conflicting views about the importance of and need to support time-sharing.

A paper titled *Program and Addressing Structure in a Time-Sharing Environment* by Bruce Arden, Bernard Galler, Frank Westervelt (all associate directors at UM's academic Computing Center), and Tom O'Brian building upon some basic ideas developed at the

Massachusetts Institute of Technology (MIT) was published in January 1966.[4] The paper outlined a virtual memory architecture using dynamic address translation (DAT) that could be used to implement time-sharing.

After a year of negotiations and design studies, IBM agreed to make a one-of-a-kind version of its S/360-65 mainframe computer for the University of Michigan. The S/360-65M[3] would include dynamic address translation (DAT) features that would support virtual memory and allow support for time-sharing. Initially IBM decided not to supply a time-sharing operating system for the new machine.



Left side, 2167 configuration console for the IBM/System 360 Model 67-2 (duplex) at the University of Michigan, c. 1969

As other organizations heard about the project they were intrigued by the time-sharing idea and expressed interest in ordering the modified IBM S/360 series machines. With this demonstrated interest IBM changed the computer's model number to S/360-67 and made it a supported product. When IBM realized there was a market for time-sharing, it agreed to develop a new time-sharing operating system called TSS/360 (TSS stood for Time-sharing System) for delivery at roughly the same time as the first model S/360-67.

The first S/360-67 was shipped in May 1966. The S/360-67 was withdrawn on March 15, 1977.[5]

Before the announcement of the Model 67, IBM had announced models 64 and 66, DAT versions of its 60 and 62 models, but they were almost immediately replaced by the 67 at the same time that the 60 and 62 were replaced by the 65.[6]

Announcement

IBM announced the S/360-67 in its August 16, 1965 "blue letters" (a standard mechanism used by IBM to make product announcements). IBM stated that:[7]

- "Special bid restrictions have been removed from the System/360 Model 67" (i.e., it was now generally available)
- It included "multiprocessor configurations, with a high degree of system availability", with up to four processing units [while configurations with up to four processors were announced, only one and two processors configurations were actually built][1]
- It had "its own powerful operating system...[the] Time Sharing System monitor (TSS)" offering "virtually instantaneous access to and response from the computer" to "take advantage of the unique capabilities of a multiprocessor system"
- It offered "dynamic relocation of problem programs using the dynamic address translation facilities of the 2067 Processing Unit, permitting response, within seconds, to many simultaneous users"

Virtual memory

The S/360-67 design included a radical new component for implementing virtual memory, the "DAT box" (Data Address Translation box). DAT on the 360/67 was based on the architecture outlined in a 1966 JACM paper by Arden, Galler, Westervelt, and O'Brien[4] and included both segment and page tables. The Model 67's virtual memory support was very similar to the virtual memory support that eventually became standard on the entire System/370 line.

The S/360-67 provided a 24- or 32-bit address space[1] – unlike the strictly 24-bit address space of other S/360 and early S/370 systems, and the 31-bit address space of S/370-XA available on later S/370s. The S/360-67 virtual address space was divided into *pages* (of 4096 bytes)[1] grouped into *segments* (of 1 million bytes); pages were dynamically mapped onto the processor's real memory. These S/360-67 features plus reference and change bits as part of the storage key enabled operating systems to implement demand paging: referencing a page that was not in memory caused a page fault, which in turn could be intercepted and processed by an operating system interrupt handler.

The S/360-67's virtual memory system was capable of meeting three distinct goals:

- **Large address space.** It mapped physical memory onto a larger pool of virtual memory, which could be dynamically swapped in and out of real memory as needed from random-access storage (typically: disk or drum storage).
- **Isolated OS components.** It made it possible to remove most of the operating system's memory footprint from the user's environment, thereby increasing the memory available for application use, and reducing the risk of applications intruding into or corrupting operating system data and programs.
- **Multiple address spaces.** By implementing multiple virtual address spaces, each for a different user, each user could potentially have a private virtual machine.

The first goal removed (for decades, at least) a crushing limitation of earlier machines: running out of physical storage. The second enabled substantial improvements in security and reliability. The third enabled the implementation of true virtual machines. It is important to note that full hardware virtualization and virtual machines were *not* original design goals for the S/360-67 (contemporary documents and observers make this clear, despite revisionist claims to the contrary).

Features

The S/360-67 included the following extensions in addition to the standard and optional features available on all S/360 systems:[1]

- Dynamic Address Translation (DAT) with support for 24 or 32-bit virtual addresses using segment and page tables (up to 16 segments each containing up to 256 4096 byte pages)
- Extended PSW Mode that enables additional interrupt masking and additional control registers
- High Resolution Interval Timer with a resolution of approximately 13-microseconds
- Reference and change bits as part of storage protection keys
- Extended Direct Control allowing the processors in a duplex configuration to present an external interrupt to the other processor
- Partitioning of the processors, processor storage, and I/O channels in a duplex configuration into two separate subsystems
- Floating Addressing to allow processor storage in a partitioned duplex configuration to be assigned consecutive real memory addresses
- A Channel Controller that allows both processors in a duplex configuration to access all of the I/O channels and that allows I/O interrupts to be presented to either processor independent of what processor initiated the I/O operation

- Simplex configurations can include 7 I/O channels, while duplex configurations can include 14 I/O channels
- Three new supervisor-state instructions: Load Multiple Control (LMC), Store Multiple Control (SMC), Load Real Address (LRA)
- Two new problem-state instructions: Branch and Store Register (BASR), and Branch and Store (BAS)
- Two new program interruptions: Segment translation exception (16) and page translation exception (17)

The S/360-67 operated with a basic internal cycle time of 200 nanoseconds and a basic 750 nanosecond magnetic core storage cycle, the same as the S/360-65.[1] The 200 ns cycle time put the S/360-67 in the middle of the S/360 line in terms of processor speed (3.9 times faster than the Model 30 at 750 ns and 3.7 times slower than the Model 195 at 54 ns). From 1 to 8 bytes (8 data bits and 1 parity bit per byte) could be read or written to processor storage in a single cycle. A 60-bit parallel adder facilitated handling of long fractions in floating-point operations. An 8-bit serial adder enabled simultaneous execution of floating point exponent arithmetic, and also handled decimal arithmetic and variable field length (VFL) instructions.

New Components

Four new components were part of the S/360-67:

- 2067 Processing Unit Models 1 and 2,
- 2365 Processor Storage Model 12,
- 2846 Channel Controller, and
- 2167 Configuration Unit.

These components, together with the 2365 Processor Storage Model 2, 2860 Selector Channel, 2870 Multiplexer Channel, and other System/360 control units and devices were available for use with the S/360-67.

Note that while Carnegie Tech had a 360/67 with an IBM 2361 LCS, that option was not listed in the price book and may not have worked in a duplex configuration.

Basic Configurations

Three basic configurations were available for the IBM System/360 model 67:

- Simplex—one IBM 2067-1 processor, two to four IBM 2365-2 Processor Storage components (512K to 1M bytes), up to seven data channels, and other peripherals. This system was called the IBM System/360 model 67-1.
- Half-duplex—one IBM 2067-2 processor, two to four IBM 2365-12 Processor Storage components (512K to 1M bytes), one IBM 2167 Configuration Unit, one or two IBM 2846 Channel Controllers, up to fourteen data channels, and other peripherals.
- Duplex—two IBM 2067-2 processors, three to eight IBM 2365-12 Processor Storage components (768K to 2M bytes), one IBM 2167 Configuration Unit, one or two IBM 2846 Channel Controllers, up to fourteen data channels, and other peripherals.

A half-duplex system could be upgraded in the field to a duplex system by adding one IBM 2067-2 processor and the third IBM 2365-12 Processor Storage, unless the half-duplex system already had three or more. The half-duplex and duplex configurations were called the IBM System/360 model 67-2.

Operating systems

When the S/360-67 was announced in August 1965, IBM also announced TSS/360, an ill-fated time-sharing operating system project that was canceled in 1971 (having also been canceled in 1968, but reprieved in 1969).

IBM's failure to deliver TSS/360 as promised opened the door for others to develop operating systems that would use the unique features of the S/360-67:

- MTS, the Michigan Terminal System, was the time-sharing operating system developed at the University of Michigan and first used on the Model 67 in January 1967. Virtual memory support was added to MTS in October 1967. Multi-processor support for a duplex S/360-67 was added in October 1968.[8]
- CP/CMS was the first virtual machine operating system. Developed at IBM's Cambridge Scientific Center (CSC) near MIT. CP/CMS was essentially an unsupported research system, built away from IBM's mainstream product organizations, with active involvement of outside researchers. Over time it evolved into a fully supported IBM operating system (VM/370 and today's z/VM).
- VP/CSS was developed by National CSS to provide commercial time-sharing services. It was based upon CP/CMS.

Legacy

The S/360-67 had an important legacy. After the failure of TSS/360, IBM was surprised by the blossoming of a time-sharing community on the S/360-67 platform (CP/CMS, MTS, MUSIC). A large number of commercial, academic, and service bureau sites installed the system. By taking advantage of IBM's lukewarm support for time-sharing, and by sharing information and resources (including source code modifications), they built and supported a generation of time-sharing centers.

The unique features of the S/360-67 were initially *not* carried into IBM's next product series, the System/370, although the 370/145 had an associative memory that appeared more useful for paging than for its ostensible purpose.[9] This was largely fallout from a bitter and highly-visible political battle within IBM over the merits of time-sharing versus batch processing. Initially at least, time-sharing lost.

However, IBM faced increasing customer demand for time-sharing and virtual memory capabilities. IBM also could not ignore the large number of S/360-67 time-sharing installations – including the new industry of time-sharing vendors, such as National CSS[10] [11] and Interactive Data Corporation (IDC),[12] that were quickly achieving commercial success.

In 1972, IBM added virtual memory features to the entire S/370 series, a move seen by many as a vindication of work done on the S/360-67 project. The survival and success of IBM's VM family, and of virtualization technology in general, also owe much to the S/360-67.

In 2010, in the technical description of its latest mainframe, the z196, IBM stated that its software virtualization started with the System/360 model 67[13] .

References

- A. Padegs, "System/360 and Beyond" [14], *IBM Journal of Research & Development, vol. 25 no. 5, pp.* 377-390, September 1981
- E.W. Pugh, L.R. Johnson, and John H. Palmer, *IBM's 360 and early 370 systems,* MIT Press, Cambridge MA and London, ISBN 0-262-16123-0, includes extensive (819 *pp.*) treatment of IBM's offerings during this period
- *IBM System/360 System Summary* [15], thirteenth edition, January 1974, IBM publication GA22-6810-12, pages 6–13 to 6-15 describe the model 67
- IBM System/360 Model 67 Reference Data (Blue card) [16]
- Melinda Varian, *VM and the VM community, past present, and future* [17]*, SHARE 89 Sessions 9059-9061, 1997*

[1] *IBM System/360 Model 67 Functional Characteristics* (http://www.bitsavers.org/pdf/ibm/360/funcChar/GA27-2719-2_360-67_funcChar.pdf), Third Edition (February 1972), IBM publication GA27-2719-2

[2] *The IBM 360/67 and CP/CMS* (http://www.multicians.org/thvv/360-67.html), Tom Van Vleck, 1995, 1997, 2005, 2009

[3] "A History of MTS—30 Years of Computing Service" (https://www.msu.edu/~mrr/mycomp/mts/others/feat02.htm), Susan Topol, *Information Technology Digest*, Volume 5, No. 5 (May 13, 1996), University of Michigan

[4] "Program and Addressing Structure in a Time-Sharing Environment" (http://portal.acm.org/citation.cfm?doid=321312.321313), B. W. Arden , B. A. Galler , T. C. O'Brien , F. H. Westervelt, *Journal of the ACM (JACM)*, v.13 n.1, p.1-16, Jan. 1966

[5] "System/360 Dates and characteristics" (http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_FS360.html) at IBM Archives > Exhibits > IBM Mainframes > Mainframes reference room > Mainframes basic information sources

[6] DIGITAL COMPUTER NEWSLETTER (http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=AD0694645), Office of Naval Research, Mathematical Sciences Division, July 1965--pages 5-6: IBM System/360 time-sharing computers

[7] Varian, *op. cit., p.* 17 (Note 54) – S/360-67 announcement

[8] Pugh, *op. cit., p.* 364 – MTS on dual processor S/360-67 in 1968

[9] IBM. *IBM Maintenance Library 3145 Processing Unit Theory - Maintenance*. SY24-3581-2.:CPU 117-129

[10] "A technical history of National CSS" (http://www.computerhistory.org/corphist/documents/doc-42ae226a5a4a1.pdf), Harold Feinleib, Computer History Museum (March 2005)

[11] "From the very beginning… from my vantage point — early history of National CSS" (http://www.computerhistory.org/corphist/documents/doc-422fd82791f26.pdf), Dick Orenstein, Computer History Museum (January 2005)

[12] Varian, op. cit., pp. 24, Note 76 – IDC systems (quoting Dick Bayles)

[13] SG24-7832-00: IBM zEnterprise System Technical Introduction (http://www.redbooks.ibm.com/redpieces/pdfs/sg247832.pdf), page 55: “Starting in 1967, IBM has continuously provided software virtualization in its mainframe servers.”

[14] http://www.research.ibm.com/journal/rd/255/ibmrd2505D.pdf

[15] http://bitsavers.informatik.uni-stuttgart.de/pdf/ibm/360/systemSummary/GA22-6810-12_360sysSumJan74.pdf

[16] http://archive.michigan-terminal-system.org/documents/IBM360-67RefCard.pdf?attredirects=0&d=1

[17] http://www.princeton.edu/~melinda/25paper.pdf

MAD programming language

Paradigm	imperative
Appeared in	1959
Developer	Galler, Arden, and Graham
Major implementations	IBM 704, IBM 7090, UNIVAC 1108, Philco 210-211, IBM S/360, and IBM S/370
Dialects	MAD, MAD/I, GOM
Influenced by	IAL, ALGOL 58
OS	UMES, MTS, CTSS, others

MAD (short for **M**ichigan **A**lgorithm **D**ecoder), is a programming language and compiler developed at the University of Michigan by Bernard Galler, Bruce Arden and Robert M. Graham.[1]

MAD, MAD/I, and GOM

There are three MAD compilers:

1. Original **MAD**, the compiler developed in 1959 at the University of Michigan for the IBM 704 and later the IBM 709 and IBM 7090 mainframe computers running the University of Michigan Executive System (UMES) and the Compatible Time-Sharing System (CTSS) operating systems.[2] [3] In the mid-1960s MAD was ported at the University of Maryland to the UNIVAC 1108.[4] Versions of MAD were also available for the Philco 210-211 and UNIVAC 1107.[4]
2. **MAD/I**, an "extended" version of MAD for the IBM S/360 series of computers running under the Michigan Terminal System (MTS). Work on the new compiler started in 1965 as part of the ARPA sponsored CONCOMP project at the University of Michigan. As work progressed it gradually became clear that MAD/I was a new language independent of the original 7090 version of MAD.[5]
3. **GOM** (**G**ood **O**ld **M**AD), a reimplementation of the original 7090 MAD for the IBM S/370 series of mainframe computers running the Michigan Terminal System (MTS). GOM was created in the early 1980s by Don Boettner at the University of Michigan Computing Center.[6] [7]

History

MAD is a variant of the International algorithmic language (IAL). IAL is the original name for what eventually became ALGOL 58. While MAD was motivated by ALGOL 58, it does not resemble ALGOL 58 in any significant way.[8] [9]

Programs written in MAD included MAIL[10] , RUNOFF, one of the first text processing systems, and several other utilities all under Compatible_Time-Sharing_System(CTSS)[11] . Work was done on a design for a MAD compiler for Multics, but it was never implemented.[12]

The following is an interesting quote from *An Interview with Brian Kernighan*[13] *when he was asked "What hooked you on programming?":*

> I think that the most fun I had programming was a summer job at Project MAC at MIT in the summer of 1966, where I worked on a program that created a job tape for the brand new GE 645 in the earliest days of Multics. I was writing in MAD, which was much easier and more pleasant than the FORTRAN and COBOL that I had written earlier, and I was using CTSS, the first time-sharing system, which was infinitely easier and more pleasant than punch cards.

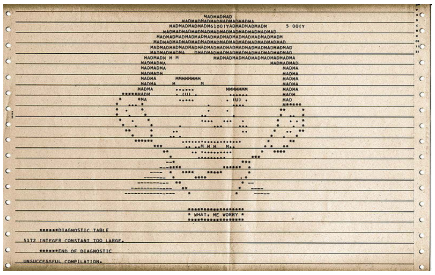
MAD was quite fast compared to some of the other compilers of its day. Because a number of people were interested in using the FORTRAN language and yet wanted to obtain the speed of the MAD compiler, a system called MADTRAN (written in MAD) was developed. MADTRAN was simply a translator from FORTRAN to MAD, which then produced machine code. MADTRAN was distributed through SHARE.[8]

MAD/I has a syntactic structure similar to ALGOL 60 together with important features from the original MAD and from PL/I.[5] MAD/I was designed as an extensible language. It was available for use under MTS and provided many new ideas which made their way into other languages, but MAD/I compilations were slow and MAD/I never extended itself into widespread use when compared to the original 7090 MAD.[7]

GOM is essentially the 7090 MAD language modified and extended for the 360/370 architecture with some judicious tailoring to better fit current programming practices and problems.[7] The MTS Message System was written in GOM.

MAD, MAD Magazine, and Alfred E. Neuman

In a pre-release version of the original MAD, as a reference to MAD's namesake, MAD magazine, when a program contained too many compile time errors the compiler would print a full-page picture of Alfred E. Neuman using ASCII art. The caption read, "See this man about your program--He might want to publish it. He never worries--but from the looks of your program, you should."[4] This feature was not included in the final official version.[14]



Line printer output following a MAD compiler error on an IBM 704 computer at the University of Michigan, c. 1960

And Bernie Galler remembers:

> By the time we designed the language that we thought would be worth doing and for which we could do a compiler, we couldn't call it Algol any more; it really was different. That's when we adopted the name MAD, for the Michigan Algorithm Decoder. We had some funny interaction with the Mad Magazine people, when we asked for permission to use the name MAD. In a very funny letter, they told us that they would take us to court and everything else, but ended the threat with a P.S. at the bottom - "Sure, go ahead." Unfortunately, that letter is lost.[11]

"Hello, world" example

The "hello, world" example program prints the string "Hello, world" to a terminal or screen display.

```
PRINT FORMAT HELLOW
VECTOR VALUES HELLOW=$13h0Hello, world*$
END OF PROGRAM
```

The first character of the line is treated as logical carriage control, in this example the character zero which causes a double spaced line to be printed.

Or, if entering all of the keywords at your keypunch is too much work, you can use contractions and the compiler will expand them in the listing:

```
P'T HELLOW
V'S HELLOW=$13h0Hello, world*$
E'M
```

Language elements

MAD and GOM, but not MAD/I, are composed of the following elements:[3] [7] [8]

Input format

MAD programs are a series of statements written on punched cards, generally one statement per card, although a statement can be continued to multiple cards.

- Columns 1-10 contains an optional statement label
- Column 11 is the remark and continuation column
- Columns 73-80 are unused and could contain a sequence identifier

The first card for a statement has a blank or no punch in card column 11 and continuation cards contain the digits 0 through 9 in column 11. Comments or remarks are flagged using the letter "R" in column 11.

The character set supported includes the upper case letters A-Z, the digits 0-9, and the characters , (comma), + (plus), - (minus), ' (apostrophe), =, *, /, (,), &, and blank or space. Lowercase letters and " (double quote) are not supported.

Spaces are not significant anywhere other than within character constants.

Few key words in the language are reserved words since since most are longer than six letters or are surrounded by periods. There is a standard set of abbreviations which can be used to replace the longer words. These consist of the first and last letters of the key words with an apostrophe between them, such as W'R for WHENEVER and D'N for DIMENSION.

For GOM:

- input is free form,
- there is no sequence field,
- input lines may be up to 255 characters long,
- lines that start with an asterisk (*) are comments,
- lines that start with a plus-sign (+) are continuation lines,
- lines that start with a letter (A-Z) have a statement label that continues to the first blank character,
- multiple-statements separated by semi-colons (;) may be placed on a line,
- the characters "1" (new page), "-" (triple space), and "0" (double space) in column 1 control spacing of the source listing,
- the underscore (_) and double-quote (") characters are added, and
- input may include upper or lower case letters, but case is ignored except in character constants.

Data types

MAD uses the term "mode" for its data types. Five basic modes are supported:

1. **Integer** (-9,999,999,999 to +9,999,999,999)
    1. integer constants (1, +1, -1)
    2. octal constants (to 7777777777777K)
    3. treated as integers, written with or without a scale factor (1K10, 1K)
    4. alphabetic or character constants (1 to 6 characters delimited using the dollar sign ($ABCDEF$), double dollar-signs are used to enter a true dollar sign ($$$.56$ is 56 cents).
    5. treated as integers, stored left justified and padded on the right with from 0 to 5 blanks
2. **Floating Point** (10^{-38} to 10^{38})
    1. written with a decimal point without an exponent (0., 1.5, -0.05, +100.4, -4.); or
    2. written with an exponent with or without a decimal point (.05E-2, -.05E2, 5E02, 5.E2)
3. **Boolean** (1B for true and 0B for false)

4. **Statement Label**
5. **Function Name**
    1. written as a name followed by a period (SQRT.), where names are from one to six alphabetic characters or decimal digits long and must start with an alphabetic character

Strings longer than six characters may be represented using arrays.

The mode of a constant can be redefined by adding the character M followed by a single digit at the end of the constant, where 0 indicates floating point, 1 integer, 2 boolean, 3 function name, and 4 statement label.

Input-output lists, VECTOR VALUES statements, and some subroutines allow the use of block notation, which has the form A,...,B or A...B, which is a reference to the entire region from A to B, inclusive. In terms of a vector, A(1)...A(N) would be A(1), A(2), A(3), ..., A(N).

For GOM:

- 4 byte words are used for the original five modes;
- six additional modes are added: CHARACTER, SHORT INTEGER, BYTE INTEGER, LONG INTEGER, POINTER, and DYNAMIC RECORD;
- constants may be qualified by appending an at-sign (@) followed by one or more characters: @X (hexadecimal), @L*n* (length of *n* bytes), @M*n* (mode *n*), @F or @FT (FLOATING POINT), @I or @IR (INTEGER), @BN (BOOLEAN), @C, @CH, or @CR (CHARACTER), @SI, @SR, or @H (SHORT INTEGER), @BI or @BR (BYTE INTEGER), @LI or @LR (LONG INTEGER), and @P (POINTER);
- character constants may be delimited using either the dollar-sign ($) or the double-quote ("); and
- uninitialized variables are not preset to a known value.

Names

- Variable names, function names, and statement labels have the same form, a letter followed by zero to five letters or digits. Function names end with a period.
- The same name cannot be used for more than one purpose.
- All names can be subscripted (the name followed by parentheses, with multiple subscripts separated by commas). There is no limit on the number of subscripts.

For GOM:

- names may be up to 24 characters long, and
- names may include the underscore (_) character.

Arrays

- Arrays of one dimension are called vectors and multiple dimension arrays are called matrices.
- Arrays must be declared using the DIMENSION statement, by including dimension information on another declaration statement, or can be implicitly declared using the VECTOR VALUES statement.
- Negative and zero subscripts are allowed.
- Matrices may be referenced using a subscript for each dimension, NAME(s_1,s_2,s_3), or using a single subscript, NAME(s_1).
- Matrices are storied in consecutive memory locations in the order determined by varying the rightmost subscript first.
- There are facilities that allow changing dimensions at run-time; permitting the programmer to vary the location of the initial element in an array within the overall block which has been set aside for the array; and allowing an arbitrary storage mapping to be specified.
- In GOM character arrays may be referenced using the following notation:
    - *array*(*first ... last*)

- *array*(... *last*)
- *array*(*first* ...)
- *array*(...)
- *array*(*first* | *length*)

Operators

Arithmetic operators

- .ABS. (unary absolute value)
- + (unary identity)
- - (unary negation)
- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- .P. (exponentiation)
- .N. (bitwise negation)
- .A. (bitwise and)
- .V. (bitwise or)
- .EV. (bitwise exclusive or)
- .LS. (left shift)
- .RS. (right shift)
- .REM. (remainder, GOM only)

Relational operators

- .L. (less than)
- .LE. (less than or equal)
- .E. (equal)
- .NE. (not equal)
- .G. (greater than)
- .GE. (greater than or equal)

Boolean operators

- .NOT. (unary logical not)
- .OR. (logical or)
- .EXOR. (logical exclusive or)
- .AND. (logical and)
- .THEN. (implies)
- .EQV. (equivalence)

Pointer operators (GOM only)

- : (selection)
- .LOC. (location)
- .IND. (indirection)

Bit operators (GOM only)

- .SETBIT. (set bit to 1)
- .RESETBIT. (reset bit to 0)
- .BIT. (test bit)

Expressions

Integer, floating point, and boolean expressions are provided. Integer and floating point variables and constants can be combined in a single expression with floating point as the result (parts of such computation may be performed using integer arithmetic and the final result may differ from those where the entire computation is done in floating point).

The left and right sides of an assignment must be of the same type, except that an integer or floating point expression on the right will be converted to a floating point or integer expression on the left if necessary.

Declaration statements

Variables may be implicitly or explicitly declared. By default all implicitly declared variables are assumed to be floating point. The NORMAL MODE IS statement may be used to change this default.

- FLOATING POINT var_1, var_2, ... (may include dimension information)
- INTEGER var_1, var_2, ... (may include dimension information)
- BOOLEAN var_1, var_2, ... (may include dimension information)
- FUNCTION NAME *name*$_1$, *name*$_2$, ... (may include dimension information)
- STATEMENT LABEL *label*$_1$, *label*$_2$, ... (may include dimension information)
- MODE NUMBER *n*, var_1, var_2, ... (may include dimension information)
- NORMAL MODE IS *type-name* (INTEGER, BOOLEAN, FLOATING POINT, STATEMENT LABEL, or FUNCTION NAME)
- NORMAL MODE IS MODE NUMBER *n*
- DIMENSION *variable*(*max-dimension*) (declares an array from 0...*max-dimension*)
- DIMENSION *variable*(*from...to*)
- DIMENSION *variable*((*subscript*$_1$, *subscript*$_2$, ..., *subscript*$_n$) (declares a multidimensional array)
- VECTOR VALUES *array*(*n*) = c_1, c_2, c_3, ...
- VECTOR VALUES *array*(*m*) ... *array*(*n*) = constant
- DOUBLE STORAGE MODE *mode-list* (doubles the amount of storage allocated for the modes listed)
- EQUIVALENCE (a_1, a_2, ..., a_m), ...
- PROGRAM COMMON *a*, *b*, *c*, ... (may include dimension information)
- ERASABLE *a*, *b*, *c*, ... (may include dimension information)
- PARAMETER $A_{1<(B}$1/sub>), $A_2(B_2)$, ..., $A_n(B_n)$
- SYMBOL TABLE VECTOR *variable*
- FULL SYMBOL TABLE VECTOR *variable*
- LISTING ON (the default)
- LISTING OFF
- REFERENCES ON
- REFERENCES OFF (the default)

Executable statements

- *variable* = *expression* (assignment)
- TRANSFER TO *statement-label*
- WHENEVER *boolean-expression*, *executable-statement* (simple conditional)
- WHENEVER *boolean-expression* (compound conditional)
  - OR WHENEVER *boolean-expression*
  - OTHERWISE
  - END OF CONDITIONAL
- CONTINUE (do nothing statement, usually used to carry a statement label)
- THROUGH *statement-label*, FOR VALUES OF *variable* = *expression-list* (iteration)
- SET LIST TO *array-element*, [*expression*]
- SAVE DATA *list*
- RESTORE DATA *list*
- PAUSE NO. *octal-integer* (stop execution, print an octal number on the operators console, allow manual restart)
- END OF PROGRAM (the last statement in all MAD programs)

Input and output statements

- READ DATA (reads data using a self-defining format, *var1*=*value1*, *var2*=*value2*, ..., *varN*=*valueN*
- READ AND PRINT DATA (similar to READ DATA, but data read is echoed to the printer)
- READ FORMAT *format*, *list*
- READ BCD TAPE *n*, *format*, *list*
- READ BINARY TAPE *n*, *list*
- PRINT RESULTS *list*
- PRINT BCD RESULTS *list*
- PRINT OCTAL RESULTS *list*
- PRINT COMMENT $*string*$ (first character of string is carriage control)
- PRINT FORMAT *format*, *list*
- PUNCH FORMAT *format*, *list*
- LOOK AT FORMAT *format*, *list* (read data without advancing to next record)
- REWIND TAPE *n*
- END OF FILE TAPE *n*
- BACKSPACE RECORD OF TAPE *n*
- BACKSPACE RECORD OF TAPE *n*, IF LOAD POINT TRANSFER TO *statement*
- BACKSPACE FILE OF TAPE *n*
- BACKSPACE FILE OF TAPE *n*, IF LOAD POINT TRANSFER TO *statement*
- SET LOW DENSITY TAPE *n*
- SET HIGH DENSITY TABLE *n*

- PRINT ON LINE FORMAT *format*, *list* (display a message for the machine operator)
- WRITE BCD TAPE *n*, *format*, *list*
- WRITE BINARY TAPE *n*, *list*
- REWIND TAPE *n*
- UNLOAD TAPE *n*
- FORMAT VARIABLE *list* (declaration, may include dimension information)

Functions

Internal and external functions are supported. Internal functions are compiled as part of the program in which they are used and share declarations and variables with the main program. External functions are compiled separately and do not share declarations and variables.

Function names end with a period. Singled-valued functions are called when they are referenced by name as part of an expression, while non-single valued functions (procedures) are called using the EXECUTE statement.

Functions are passed arguments by value. Arguments can be expressions, statement labels, and function names.

A one statement definition of internal functions is permitted.

Recursive functions are permitted, although the function must do some of the required saving and restoring work itself.

- INTERNAL FUNCTION *function-name*.(*argument-list*) = *expression* (single statement definition)
- INTERNAL FUNCTION *function-name*.(*argument-list*)
- EXTERNAL FUNCTION *function-name*.(*argument-list*)
- ENTRY TO NAME *name*.
- END OF FUNCTION (last statement in a multiple line definition)
- FUNCTION RETURN [*expression*]
- ERROR RETURN (force an error return to a statement or to the operating system, if no error statement is given as last argument of the call)
- SAVE RETURN
- RESTORE DATA
- RESTORE RETURN
- EXECUTE *procedure*.(*argument-list*) (call a non-single valued function)

Operator definition and redefinition

One of the most interesting features in MAD is the ability to extend the language by redefining existing operators, defining new operators, or defining new data types (modes). The definitions are made using MAD declaration statements and assembly language mnemonics included following the declaration up to the END pseudo-instruction that implement the operation.

- DEFINE BINARY OPERATOR *defined-op*, PRECEDENCE *rank existing-op* MODE STRUCTURE *mode-options*
- DEFINE UNARY OPERATOR *defined-op*, PRECEDENCE *rank existing-op* MODE STRUCTURE *mode-options*
- MODE STRUCTURE *mode-no* = *mode-no existing-op mode-no*
- MODE STRUCTURE *mode-no* = *mode-no existing-op mode-no* SAME SEQUENCE AS *mode-no existing-op mode-no*

where:

- *rank* is one of SAME AS, LOWER THAN, or HIGHER THAN; and
- *mode-options* are the options that appear on the MODE STRUCTURE statement.

Three pre-defined packages of definitions (MATRIX, DOUBLE PRECISION, and COMPLEX) are available for inclusion in MAD source programs using the INCLUDE statement.

- INCLUDE *package*

See also

• ALGOL 58 • ALGOL 60

References

- *An Abbreviated description of the MAD compiler language*, Fernando J. Corbató, Jerome H. Saltzer, Neil Barta, and Thomas N. Hastings, M.I.T. Computation Center Memorandum CC-213, June, 1963.
- Bernie Galler talks "a little about the history of MAD"[11]
- *CLSYS, a program to facilitate the use of the MAD translator for large (class-size) batches*, Jerome H. Saltzer, M.I.T. Computation Center Memorandum CC-204. February, 1963.
- *A Computer Primer for the Mad Language*, Elliott Irving Organick, 1961.
- *Internal organization of the MAD translator* [15], Arden, B. W., Galler, B. A. and Graham, R. M., pp. 28–31, CACM Volume 4 No. 1 (Jan 1961)
- *An Introduction To Algorithmic Methods Using The MAD Language*, Alan B. Marcovitz and Earl J. Schweppe, Macmillan, 1966.
- *An Introduction to Digital Computers and the MAD Language* [16], Brice Carnahan, University of Michigan.
- *The Language of Computers* [17], Bernard A. Galler, University of Michigan, McGraw-Hill, 1962.
- *MAD at Michigan: its function & features* [15], Arden, B. W., Galler, B. A., and Graham, R. M., pp27–28, Datamation, Volume 7 No. 12 (Dec 1961)
- The archives at the Bentley Historical Library of the University of Michigan contain reference materials on the development of MAD and MAD/I, including three linear feet of printouts with hand-written notations and original printed manuals.[18] [19] [20] [21]

[1] Alt, Franz (1967). *Advances in Computers*. Academic Press. pp. 143. ISBN 0120121042.

[2] *A User's Reference Manual For The Michigan Algorithm Decoder (MAD) For the IBM 7090* (http://www.bitsavers.org/pdf/univOfMichigan/mad/L2-UOI-MAD1-2-RX_MADum_62.pdf), Digital Computer Laboratory, Graduate College, University of Illinois, 1962, 221 pages

[3] *The Michigan Algorithm Decoder (The MAD Manual)* (http://deepblue.lib.umich.edu/bitstream/2027.42/3292/5/bab2545.0001.001.pdf), Bruce W. Arden, Revised Edition 1966

[4] UNIVAC and ALGOL by George Gray in *Unisys History Newsletter* (https://wiki.cc.gatech.edu/folklore/index.php/UNIVAC_and_ALGOL#MAD), Volume 6, Number 2 (June 2002), Georgia Tech

[5] *The MAD/I Manual* (http://deepblue.lib.umich.edu/bitstream/2027.42/3709/5/abu9341.0001.001.pdf), Bolas, Springer, and Srodawa, CONCOMP Technical Report 32, 1970, University of Michigan, Ann Arbor, 194 pages

[6] *MTS Volume 2: Public File Descriptions*, University of Michigan Computing Center

[7] *GOM Manual* (http://archive.michigan-terminal-system.org/documents/GOMManual-June1989.pdf?attredirects=0&d=1), Don Boettner, University of Michigan Computing Center, Ann Arbor, June 1989

[8] *Computer Languages - Principles and History* (http://hopl.murdoch.edu.au/showlanguage.prx?exp=92&language=MAD), Sammet, Jean E., Englewood Cliffs, N.J., Prentice-Hall 1969, p. 205

[9] In August 2010 when asked about Jean's Sammet's statement that "MAD does not resemble ALGOL 58 in any significant way", Bruce Arden wrote: "Regarding Jean Sammet, she may have conflated the two versions of IAL (58 and 60). Unlike the later version, the 58 version said nothing about what words (or language) should be used to identify conditional and transfer statements, which led for parsing reasons to words like WHENEVER. Also there were some additional features in MAD that went beyond the 58 specs."

[10] Documentation and Source for Early Electronic Mail and Messaging (http://www.multicians.org/thvv/mail-details.html), Tom Van Vleck

[11] "A Career Interview with Bernie Galler" (http://hopl.murdoch.edu.au/showlanguage.prx?exp=92&language=MAD), Galler and Galler, pp 22-33, IEEE Annals of the History of Computing, 23(1) January 2001

[12] Glossary of Multics acronyms and terms (http://www.multicians.org/mgm.html) by Tom Van Vleck

[13] An Interview with Brian Kernighan: Breeding Little Languages (http://broadcast.oreilly.com/2009/04/an-interview-with-brian-kernig.html), by Allen Noren, 2009

[14] Shneiderman, Ben; Plaisant, Catherine (2004-05-07). *Designing the user interface* (4th edition ed.). Addison Wesley. ISBN 978-0321197863.

[15] http://hopl.murdoch.edu.au/showlanguage.prx?exp=92&language=MAD

[16] http://deepblue.lib.umich.edu/bitstream/2027.42/4023/5/aca1743.0001.001.pdf

[17] http://www.bitsavers.org/pdf/univOfMichigan/mad/Galler_TheLangOfComps_1962.pdf
[18] Technical Memos (http://quod.lib.umich.edu/cgi/f/findaid/findaid-idx?c=bhlead;cc=bhlead;type=simple;rgn=Entire Finding Aid;q1=MAD;view=reslist;subview=standard;sort=occur;start=1;size=25;didno=umich-bhl-0351;focusrgn=C01;byte=24555661), University of Michigan Computing Center publications, 1965-1999
[19] Technical Reports (http://quod.lib.umich.edu/cgi/f/findaid/findaid-idx?c=bhlead;cc=bhlead;type=simple;rgn=Entire Finding Aid;q1=MAD;view=reslist;subview=standard;sort=occur;start=1;size=25;didno=umich-bhl-0351;focusrgn=C01;byte=24564725), University of Michigan Computing Center publications, 1965-1999
[20] Topical File 1960-1986 (http://quod.lib.umich.edu/cgi/f/findaid/findaid-idx?c=bhlead;cc=bhlead;type=simple;rgn=Entire Finding Aid;q1=MAD;view=reslist;subview=standard;sort=occur;start=1;size=25;didno=umich-bhl-9551;focusrgn=C01;byte=24604876), University of Michigan Computing Center records, 1952-1996
[21] MAD (Michigan Algorithm Decoder) 1960-1979 (http://quod.lib.umich.edu/cgi/f/findaid/findaid-idx?c=bhlead;cc=bhlead;type=simple;rgn=Entire Finding Aid;q1=MAD;view=reslist;subview=standard;sort=occur;start=1;size=25;didno=umich-bhl-9551;focusrgn=C01;byte=24695986), University of Michigan Computing Center records, 1952-1996

External links

- Eric Raymond's retrocompiler for MAD (http://www.catb.org/~esr/mad/)
- A trivial example of a MAD program (http://99-bottles-of-beer.net/language-michigan-algorithm-decoder-391.html)
- Dave Pitts' IBM 7094 support (http://www.cozx.com/~dpitts/ibm7090.html) – Has a CTSS environment that includes the MIT version of MAD.

UBC PLUS

Paradigm	imperative, structured
Appeared in	1976
Developer	Alan Ballard and Paul Whaley at UBC
Typing discipline	static, strong, safe
Major implementations	IBM System/370, DEC PDP-11, and Motorola 68000
Influenced by	SUE, Pascal
OS	Michigan Terminal System (MTS), OS/VS1

Plus is a "Pascal-like" system implementation language from the University of British Columbia (UBC), Canada, based on the SUE[1] system language developed at the University of Toronto, circa 1971.

Description

Plus was developed at the University of British Columbia (UBC) Computing Centre by Alan Ballard and Paul Whaley for use with and for the development of the Michigan Terminal System (MTS), but the code generated by the compiler is not operating system dependent and so is not limited to use with or the development of MTS.

There is another programming language named PLUS, developed at Sperry Univac in RoseVille, Minnesota,[2] , but the Univac PLUS is not the subject of this article.

The UBC Plus compiler is written largely in Plus, runs under the Michigan Terminal System (MTS) on IBM S/370 or compatible hardware or under IBM's OS/VS1, and generates code for the IBM S/370, the DEC PDP-11, or the Motorola 68000 architectures.

Plus is based to a large extent on the SUE System Language[1] developed at the University of Toronto, circa 1971. The SUE language was derived, particularly in its data structure facilities from Pascal.[3]

Plus is superficially quite different from SUE or Pascal; however the underlying language semantics are really quite similar. Users familiar with the C programming language will also recognize much of its structure and semantics in PLUS.

Goals for the compiler and the Plus language include:[4]

1. Allow and encourage reasonable program structures
2. Provide problem-oriented data structures
3. Allow and encourage readable and understandable source code
4. Allow for parametrization using symbolic constants
5. Actively assist in the detection and isolation of errors, at compile-time if possible and optionally at run-time where necessary
6. Generate efficient code
7. Provide facilities necessary for systems programming
8. Provide reasonably efficient compilation including separate compilation of different parts of a program
9. Optionally produce symbol (SYM) information allowing programs to be debugged using a Symbolic Debugging System such as SDS under MTS

The manual, *UBC PLUS: The Plus Programming Language*,[4] is available. A description of the source and object libraries available for use with Plus, *PLUS Source Library Definitions*, is also available.[5]

"Hello, world" example

The "hello, world" example program prints the string "Hello, world!" to a terminal or screen display.

```
%Title := "Hello world";
%Include(Pluslist);
%Subtitle := "Definitions";
%Lower_Case := True;

/* Definitions that everyone needs */
%Include(Boolean, Numeric_Types, More_Numeric_Types, String_Types,
   More_String_Types);

/* A tasteful subset of procedure definitions */
%Include(Main);

/* Message routine definitions */
%Include(Message_Initialize, Message, Message_Terminate);

%Subtitle := "Local Procedure Definitions";
%Eject();
definition Main

  variable Mcb is pointer to Stream_Type;

  Mcb := Message_Initialize();
  Message("Hello, world!");
  Message_Terminate(Mcb);
  Mcb := Null;

end Main;
```

See also

- Michigan Terminal System (MTS)

References

[1] *The System Language for Project SUE* (http://doi.acm.org/10.1145/800234.807062), B. L. Clark and J. J. Horning of the Computer Systems Research Group and Department of Computer Science, University of Toronto, Proceedings of the SIGPLAN symposium on Languages for system implementation, 1971, pages 79-88

[2] The PLUS Programming Language (http://portal.acm.org/citation.cfm?doid=954127.954144), Frank W. Stodola, Sperry Univac, Roseville, Minnesota, *ACM SIGPLAN Notices*, Volume 15, Issue 1 (January 1980), pp. 146-155

[3] MTS Volume 2: Public file Descriptions, University of Michigan Computing Center, Ann Arbor, Michigan, 1984, pp. 350.1, 350.2

[4] *The PLUS Programming Language* (http://archive.michigan-terminal-system.org/documents/Plus-1987.pdf?attredirects=0&d=1), Allan Ballard and Paul Whaley, pp. 2-5, revised 1987, Computing Centre, University of British Columbia

[5] *PLUS Source Library Definitions* (http://archive.michigan-terminal-system.org/documents/PlusLib-1983.pdf?attredirects=0&d=1), Alan Ballard, 1983, University of British Columbia Computing Centre, 139pp.

Micro DBMS

Micro was one of the earliest relational database management systems.[1] It combined the relational model later made famous by Edgar F. Codd and Michael Stonebraker of the University of Michigan's Database Research Group[2] with a natural language interface which allowed non-programmers to use the system.[3]

Micro permitted users with little programming experience to define, enter, interrogate, manipulate and update collections of data in a relatively unstructured and unconstrained environment. An interactive system, Micro was powerful in terms of the complexity of requests which could be made by users without prior programming language experience.[4] Micro includes basic statistical computations such as mean, variance, frequency, median, etc. If more rigorous statistical analysis were desired, the data from a Micro database could be used with Michigan Interactive Data Analysis System (MIDAS), a statistical analysis package available under the Michigan Terminal System (MTS).[5]

Micro was originally implemented in 1968 at the University of Michigan's Institute of Labor and Industrial Relations (ILIR) and ran under the Michigan Terminal System, the time-sharing system developed at U-M. It became the first large scale relational database management system to be used in production. Organizations such as the US Department of Labor, the US Environmental Protection Agency and researchers from University of Alberta, the University of Michigan, and Wayne State University used it to manage very large scale databases. Micro continued to run in production until 1998.

References

[1] "A set theoretic data structure and retrieval language" (http://doi.acm.org/10.1145/1095495.1095500), William R. Hershey and Carol H. Easthope, Papers from the Session on Data Structures, Spring Joint Computer Conference, May 1972 in *ACM SIGIR Forum*, Volume 7, Issue 4 (December 1972), pp. 45-55, DOI=10.1145/1095495.1095500, Special Interest Group on Information Retrieval, Association for Computing Machinery, New York, NY, USA

[2] "A Relational Model of Data for Large Shared Data Banks" (http://dx.doi.org/10.1145/362384.362685), E.F. Codd, Communications of the ACM, volume 13, issue 6 (June 1970), pp.77–387, doi= 10.1145/362384.362685

[3] MICRO: Information Management System (Version 5.0) Reference Manual, M.A. Kahn, D.L. Rumelhart, and B.L. Bronson, October 1977, Institute of Labor and Industrial Relations (ILIR), University of Michigan

[4] " Use of a Relational Database to Support Clinical Research: Application in a Diabetes Program (http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2581360/pdf/procascamc00021-0314.pdf)", Diane Lomatch, M.P.H., Terry Truax, M.S., Peter Savage, M.D., Diabetes Center Unit, MDRTC, University of Michigan, 1981

[5] " Converting from Traditional File Structures to Database Management Systems: A Powerful Tool for Nursing Management" (http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2245120/pdf/procascamc00019-0674.pdf), Yvonne Marie Abdoo, Ph.D., R.N, Wayne State University College of Nursing, 1987

External links

University of Michigan Database Research Group (http://www.eecs.umich.edu/db/)

Bruce Arden

Bruce W. Arden, 2004

Bruce W. Arden (born in 1927 in Minneapolis, Minnesota) is an American computer scientist.

He graduated from Purdue University with a BS(EE) in 1949 and started his computing career in 1950 with the wiring and programming of IBM's hybrid (mechanical and electronic) Card Programmed Computer/Calculator [1] at the Allison Division of General Motors. Next he spent a short period as a programmer for computations being done at the University of Michigan's Willow Run Laboratory using the Standards Eastern Automatic Computer.

He then became a Research Associate at the University of Michigan's Statistical Research Laboratory and later an Associate Director of the University's Computing Center after its establishment in 1959. While at Michigan he co-authored two compilers, GAT[2] for the IBM 650 and MAD[3] for the IBM 704/709/7090, was involved in the design of the architecture and negotiations with IBM over the virtual memory features that would be included in what became the IBM S/360 Model 67 computer, and in the initial design of the Michigan Terminal System (MTS) time-sharing operating system.[4]

His increasing interest in academic computer science and engineering motivated Arden to complete a doctoral program in Electrical Engineering in 1965. He was subsequently a professor in, and ultimately Chairman of, the Computer and Communication Sciences department at Michigan.[4] In 1973 he accepted a professorship at Princeton University and chaired the department of Electrical Engineering and Computer Science.[5] In 1986, then Princeton's Alexander Doty Professor of Engineering, he went to the University of Rochester as its Dean of the College of Engineering and Applied Science. In the three years preceding the addition of "Emeritus" to his academic title (William May Professor of Engineering) in 1995, he also served as Rochester's Vice Provost for Telecommunications and Computing.[6]

During his academic career, Bruce Arden wrote two books on numerical computation and edited another on computer science and engineering research. He wrote many papers in the areas of compilers, operating systems, computer logic and networks.[7] In addition, he supervised many students, both undergraduate and graduate, in their studies of the various aspects of computing, and he served as a consultant to government agencies and several major computer companies at various times during those years.

He is retired and lives in Michigan and Maine.

References

[1] http://www.columbia.edu/acis/history/cpc.html

[2] On GAT and the Construction of Translators (http://www.informatik.uni-trier.de/~ley/db/journals/cacm/cacm2.html), B. Arden and R. Graham; *CACM* Volume 2, Number 7, July 1959, pp. 24-26.

[3] *MAD at Michigan: its function & features* (http://hopl.murdoch.edu.au/showlanguage.prx?exp=92&language=MAD), Arden, B. W., Galler, B. A., and Graham, R. M., pp27–28, Datamation, Volume 7 No. 12 (Dec 1961)

[4] Computing at the University of Michigan: The Early Years through the 1960s (http://www.google.com/url?sa=t&source=web&cd=11&ved=0CEQQFjAK&url=http://www.eecs.umich.edu/eecs/cse/Publications/CSE_Booklet.pdf&ei=E8Y7TPXzCcP_nQffg5jgAw&usg=AFQjCNGMjZsS4B2KoAt-LvLhfPbDO1VNyQ), Norman R. Scott, p. 12

[5] Department of Electrical Engineering (http://etcweb.princeton.edu/CampusWWW/Companion/electrical_engineering_department.html) from *A Princeton Companion* by Alexander Leitch, Princeton University Press (1978)

[6] University of Rochester News Release (http://www.rochester.edu/news//show.php?id=1083), January 24, 1995

[7] ACM Author Portal for Brue W. Arden (http://portal.acm.org/results.cfm?query=bruce arden&querydisp=bruce arden&source_query=&start=1&srt=score dsc&short=0&source_disp=&since_month=&since_year=&before_month=&before_year=&coll=Portal&

dl=GUIDE&termshow=matchall&range_query=&CFID=96692958&CFTOKEN=53769413)

Bernard Galler

Bernie Galler, 2004

Bernard A. Galler (October 3, 1928, Chicago – September 4, 2006, Ann Arbor, Michigan) was an American mathematician and computer scientist at the University of Michigan who was involved in the development of large-scale operating systems and computer languages including the MAD programming language and the Michigan Terminal System operating system.[1] [2]

He attended the University of Chicago where he earned a B.Sc. in mathematics at the University of Chicago (1947), followed by a M.Sc. from UCLA and a Ph.D. from the University of Chicago (1955), advised by Paul Halmos and Marshall Stone. He joined the mathematics department at the University of Michigan (1955) where he taught the first programming course (1956) using an IBM 704. Galler helped to develop the computer language called the Michigan Algorithm Decoder (1959-) in use at several universities. He formed the Communication Sciences dept (1965), renamed Computer Sciences (CS), which became the Computer and Communications (CCS) dept (1984), and Computer Science Department in the 70s, from which he retired in 1994. His class developed the realtime course scheduling program called Computer Registration Involving Student Participation (CRISP) which allowed students to register for courses without waiting in long lines.[3] The University used the CRISP application for over fifteen years.

From 1968 to 1970, Prof. Galler was the President of the Association for Computing Machinery (ACM). In 1994 he was inducted as a Fellow of the Association for Computing Machinery. He was the founding editor of the journal *IEEE Annals of the History of Computing* (1979-87). He was also the President of the Software Patent Institute (1992). For fifteen years, he served as an expert witness in numerous important legal cases around the country involving computer software issues.

He was married to Enid Harris, played violin in several orchestras and chamber groups, co-founded the Ypsilanti Youth Orchestra (2001) for children whose schools did not have string music education. He was president of the Orchestra Board at the University of Michigan and a member of the Ann Arbor chapter of Rotary International. He died from pulmonary embolism.[4]

The Bernard A. Galler Fellowship Fund [5] has been established at the University of Michigan Department of Electrical Engineering and Computer Science to "attract and support outstanding graduate students pursuing an advanced degree in computer science."

References

[1] "A career interview with Bernard Galler" (http://ieeexplore.ieee.org/iel5/85/19650/00910847.pdf?arnumber=910847) in IEEE Annals of the History of Computing, Jan-Feb, pp. 22-33, 2001

[2] Atsushi Akera, "The Life and Work of Bernard A. Galler (1928-2006) (http://muse.jhu.edu/journals/ieee_annals_of_the_history_of_computing/v030/30.1akera.pdf)," IEEE Annals of the History of Computing, vol. 30, no. 1, pp. 4-14, Jan-Mar, 2008

[3] "CRISP: An Interactive Student Registration System" (http://archive.michigan-terminal-system.org/michiganterminalsystem/documents/CRISP-ED084839-1973.pdf?attredirects=0&d=1), B.A. Galler, et.al., University of Michigan, Ann Arbor, 1973, Note: Paper presented at the ACM Annual Conference (Atlanta, Georgia, August 1973)

[4] In memory of Bernard Galler (http://vielmetti.typepad.com/vacuum/2006/09/in_memory_of_be.html)

[5] Bernard A. Galler Fellowship Fund (http://www.eecs.umich.edu/eecs/about/giving/Galler.html)

External links

- Oral history interview with Bernard A. Galler (http://www.cbi.umn.edu/oh/display.phtml?sub=891) Charles Babbage Institute, University of Minnesota, Minneapolis. Galler describes the development of computer science at the University of Michigan from the 1950s through the 1980s and discusses his own work in computer science. Galler also discusses Michigan's relationship with ARPANET, CSNET, and BITNET. He describes the atmosphere on campus in the 1960s and early 1970s and his various administrative roles at the university. Galler discusses his involvement with the Association for Computing Machinery, the American Federation of Information Processing Societies, the founding of the Charles Babbage Institute, and his work with the Annals of the History of Computing.
- A Day in the Life of Bernard Galler (http://www.acm.org/crossroads/dayinlife/bios/bernard_galler.html), ACM Crossroads (no date)
- The Mathematics Genealogy Project: Bernard Galler (http://genealogy.math.ndsu.nodak.edu/id.php?id=6448)
- Bernard A. Galler (http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/g/Galler:Bernard_A=.html), DBLP Computer Science Bibliography
- Bernard Galler's obituary (http://www.ur.umich.edu/0607/Nov06_06/obits.shtml) in the University Record, University of Michigan, Ann Arbor.

TSS/360

The IBM Time Sharing System **TSS/360** was an early time-sharing operating system designed exclusively for a special model of the System/360 line of mainframes, the Model 67. Made available on a trial basis to a limited set of customers in 1967, it was never officially released as a supported product by IBM. TSS implemented a number of novel features, some of which eventually saw daylight in more popular systems such as Multics and VM/CMS. Despite its many advances and novel capabilities, the TSS/360 project was a costly and embarrassing failure for IBM.

History of IBM mainframe operating systems

On early mainframe computers:

- GM OS & GM-NAA I/O 1955
- BESYS 1957
- UMES 1958
- SOS 1959
- IBSYS 1960
- CTSS 1961

On S/360 and successors:

- BOS/360 1965
- TOS/360 1965
- TSS/360 1967
- MTS 1967
- ORVYL 1967
- MUSIC 1972
  - MUSIC/SP 1985

- **DOS/360 and successors** 1966
  - DOS/VS 1972
  - DOS/VSE 1980s
  - VSE/SP late 1980s
  - VSE/ESA 1991
  - z/VSE 2005
- **OS/360 and successors** 1966
  - MFT 1966
  - MFT II 1968
    - OS/VS1 1972
    - OS/VS1 BPE
  - MVT 1967
    - 65MP
    - OS/VS2R1 (SVS) 1972
    - MVS (OS/VS2R2 and later) 1974
      - MVS/SE 1978
      - MVS/SE 2 1979
      - MVS/SP Version 1 1980
      - MVS/XA 1983
      - MVS/ESA 1988
      - OS/390 1995
      - z/OS 2000
- **VM line**
  - CP-40/CMS 1967
  - CP-67/CMS 1967
  - VP/CSS 1968
  - VM/370 1972
  - VM/BSE (BSEPP)
  - VM/SE (SEPP)
  - VM/SP 1980
  - VM/XA 1988
  - VM/ESA 1990
  - z/VM 2000
- **TPF line**
  - ACP 1967
  - TPF 1979
  - z/TPF 2005
- **UNIX and Unix-like**
  - UTS 1981
  - AIX/370 1990
  - AIX/ESA 1991
  - Linux 1999
  - OpenSolaris 2008
  - z/OS UNIX System Services

Novel characteristics

TSS/360 was one of the first implementations of tightly-coupled symmetric mainframe multiprocessing. A pair of Model 67 mainframes shared a common physical memory space, and ran a single copy of the kernel (and application) code. An I/O operation launched by one processor could end and cause an interrupt in the other. The Model 67 used a standard 360 instruction called Test and Set to implement locks on code critical sections.

TSS/360 also implemented Virtual Memory and Virtual Machines using position-independent code.[1] [2]

TSS/360 was also unique in implementing a Table Driven Scheduler — a user-configured table whose columns were parameters such as current priority, working set size, and number of timeslices used to date. The kernel would refer to this table when calculating the new priority of a thread.

As was standard with operating system software at the time, TSS/360 customers (such as General Motors Research Laboratories) were given full access to the entire corpus of Operating System code and development tools. User-developed improvements and patches were frequently incorporated into the official source code.

Failure of TSS

TSS/360 failed primarily due to performance and reliability problems, and lack of compatibility with OS/360, although those issues were eventually addressed. IBM attempted to develop it on a very aggressive schedule with a large staff of programmers to compete with Multics. By 1967, it had become evident that TSS/360 was suffering from the same kinds of delays as OS/360. In February 1968, at the time of SHARE 30, there were eighteen S/360-67 sites attempting to run TSS. During the conference, IBM announced via "blue letter" that TSS/360 was being decommitted — a great blow to the time-sharing community. This decision was temporarily reversed, and TSS/360 was not officially scrapped until 1971. However, TSS/360 continued to be quietly available for a time to existing TSS/360 customers, as an interim measure.

After TSS/360 was scrapped, IBM put its primary efforts into the Time Sharing Option (TSO), a time-sharing monitor for OS/360. Several other groups developed less ambitious, more successful time sharing systems for the S/360-67, notably CP-67 at IBM's Cambridge Scientific Center, an early virtual machine monitor which evolved into VM/370, MTS at the University of Michigan, and ORVYL at Stanford University. IBM also provided the TSS/370 PRPQ as a migration path for existing TSS/360 customers, which went through multiple releases.

See also

- History of IBM mainframe operating systems
- History of operating systems
- Operating systems timeline

References

[1] John R. Levine (October 1999). "Chapter 8: Loading and overlays" (http://darcs.olsner.se/Linker/linker-book/linker08.html). *Linkers and Loaders*. San Francisco: Morgan-Kauffman. pp. 170–171. ISBN 1-55860-496-0. .

[2] TSS code control sections (CSECT′s) were position independent, but the associated prototype sections (PSECT′s) were not.

Further reading

- Pugh, Emerson; Lyle R. Johnson, John H. Palmer (1991). *IBM's 360 and Early 370 Systems* (http://books.google.com/books?id=MFGj_PT_clIC&printsec=frontcover&dq=IBM's+360). Cambridge MA: MIT Press. pp. 362–265, 596. ISBN 0-262-16123-0. Describes the origin and schedule problems of TSS.
- Brooks, Frederick P. (1995). *The Mythical Man-Month*. Reading MA: Addison-Wesley. ISBN 0-201-83595-5. Describes the "second system syndrome" that affected TSS.

External links

- Public domain software archive (http://www.ibiblio.org/jmaynard), includes TSS/370 source and binary archives
- TSS/360 manual archive at BitSavers.org (http://www.bitsavers.org/pdf/ibm/360/tss/), contains PDFs for a large number of TSS manuals from IBM

Article Sources and Contributors

Michigan Terminal System *Source*: http://en.wikipedia.org/w/index.php?oldid=406153707 *Contributors*: Bkonrad, Blaxthos, Bressan, Darkonc, Dmsar, Emre D., Ghettoblaster, L Kensington, LilHelpa, MTA1309, Mje, N8chz, Richard k smith, Sarrica, Suruena, TheParanoidOne, W163, Whiner01, Yworo, 21 anonymous edits

MTS system architecture *Source*: http://en.wikipedia.org/w/index.php?oldid=404102650 *Contributors*: Mild Bill Hiccup, W163, Yworo, 1 anonymous edits

IBM System/360 Model 67 *Source*: http://en.wikipedia.org/w/index.php?oldid=397448993 *Contributors*: Buster7, Chatul, EagleOne, Guy Harris, John Sauter, John W. Kennedy, John of Reading, Pcap, Rwwww, SHOlafsson, SciCorrector, Spinality, Tatrgel, UncleDouggie, W163, 12 anonymous edits

MAD programming language *Source*: http://en.wikipedia.org/w/index.php?oldid=344744711 *Contributors*: Brownsteve, CRGreathouse, Danakil, Davepitts, Ebyabe, Edward, Edward Vielmetti, Eric S. Raymond, HappyDog, Hughcharlesparker, Imran, John Vandenberg, John of Reading, JohnJHenderson, Madmardigan53, NevilleDNZ, Norm mit, Serein (renamed because of SUL), The Wild Falcon, W163, 13 anonymous edits

UBC PLUS *Source*: http://en.wikipedia.org/w/index.php?oldid=371603071 *Contributors*: W163, Wavelength, Yworo

Micro DBMS *Source*: http://en.wikipedia.org/w/index.php?oldid=403474471 *Contributors*: Bressan, Cander0000, Midnightcomm, Twp, W163

Bruce Arden *Source*: http://en.wikipedia.org/w/index.php?oldid=384445345 *Contributors*: Magioladitis, W163

Bernard Galler *Source*: http://en.wikipedia.org/w/index.php?oldid=403545534 *Contributors*: Bobtheman1234, Edward Vielmetti, Esmito, HOT L Baltimore, Janm67, Ketil3, Luk, Rich Farmbrough, W163, 6 anonymous edits

TSS/360 *Source*: http://en.wikipedia.org/w/index.php?oldid=393596583 *Contributors*: Blaxthos, Chatul, CliffBamford, Ghettoblaster, Guy Harris, Jmaynard, John W. Kennedy, Jrlevine, Kubanczyk, LilHelpa, Pigsonthewing, Robert Brockway, Saper, SciCorrector, Smallpond, Suruena, W163, Wikiklrsc, 3 anonymous edits

Image Sources, Licenses and Contributors

Image:MTS signon screenshot.png *Source*: http://en.wikipedia.org/w/index.php?title=File:MTS_signon_screenshot.png *License*: Free Art License *Contributors*: Gavin Eadie

Image:MTSWorkshipVIIIMugs-1972.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:MTSWorkshipVIIIMugs-1972.jpg *License*: Public Domain *Contributors*: User:W163

Image:IBM360-67AtUmichWithMikeAlexander.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:IBM360-67AtUmichWithMikeAlexander.jpg *License*: Public Domain *Contributors*: Dave Mills

Image:DSCN1079.jpeg *Source*: http://en.wikipedia.org/w/index.php?title=File:DSCN1079.jpeg *License*: Creative Commons Zero *Contributors*: Jeff Ogden, original photographer unknown

File:MTSVol1Cover.png *Source*: http://en.wikipedia.org/w/index.php?title=File:MTSVol1Cover.png *License*: Creative Commons Zero *Contributors*: Computing Center University of Michigan

File:ITDDigestCover.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:ITDDigestCover.jpg *License*: Creative Commons Zero *Contributors*: University of Michigan

Image:IBM2314DiskDrivesAndIBM2540CardReaderPunch.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:IBM2314DiskDrivesAndIBM2540CardReaderPunch.jpg *License*: Public Domain *Contributors*: Scott Gerstenberger

Image:IBM2321DataCellAtUMich.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:IBM2321DataCellAtUMich.jpg *License*: Public Domain *Contributors*: Scott Gerstenberger

File:ASR-33 1.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:ASR-33_1.jpg *License*: Creative Commons Attribution 2.0 *Contributors*: Dominic Alves from Brighton, England

File:Terminal-dec-vt100.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Terminal-dec-vt100.jpg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: User:ClickRick

File:DataConcentrator-PDP8-UMich-DaveMills.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:DataConcentrator-PDP8-UMich-DaveMills.jpg *License*: Public Domain *Contributors*: Dave Mills

File:Tektronix 4014.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Tektronix_4014.jpg *License*: Creative Commons Attribution-Sharealike 2.5 *Contributors*: Original uploader was Rees11 at en.wikipedia

File:ATTtelephone-large.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:ATTtelephone-large.jpg *License*: Trademarked *Contributors*: User:Dante Alighieri

File:80 early pcp 2.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:80_early_pcp_2.jpg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: Merit Network, Inc.

File:IBM-3279.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:IBM-3279.jpg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: User:Shieldforyoureyes

Image:IBM360-67ConfigConsoleAtUmich.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:IBM360-67ConfigConsoleAtUmich.jpg *License*: Public Domain *Contributors*: Dave Mills

Image:IBM-S360-67ConfigurationConsoleCloseup.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:IBM-S360-67ConfigurationConsoleCloseup.jpg *License*: Public Domain *Contributors*: Scott Gerstenberger

Image:MAD-alfie-1960.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:MAD-alfie-1960.jpg *License*: Public Domain *Contributors*: Don Boettner, University of Michigan

Image:ArdenBruceW-10Sep2004.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:ArdenBruceW-10Sep2004.jpg *License*: Creative Commons Zero *Contributors*: Jeff Ogden

Image:GallerBernard-10Sep2004.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:GallerBernard-10Sep2004.jpg *License*: Creative Commons Zero *Contributors*: Jeff Ogden

License