Microcomputers & Supercomputers

Nolan Franco
Shila Numbers

First Edition, 2012

ISBN 978-81-323-1288-8

Published by:
College Publishing House
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter 1

Introduction to Microcomputer

The Commodore 64 was one of the most popular microcomputers of its era, and is the best-selling model of home computer of all time.

A **microcomputer** is a computer with a microprocessor as its central processing unit. They are physically small compared to mainframe and minicomputers. Many microcomputers (when equipped with a keyboard and screen for input and output) are also personal computers (in the generic sense).

The abbreviation "**micro**" was common during the 1970s and 1980s, but has now fallen out of common usage.

Origins

The term "Microcomputer" came into popular use after the introduction of the minicomputer, although Isaac Asimov used the term microcomputer in his short story "The Dying Night" as early as 1956 (published in "The Magazine of Fantasy and Science Fiction" in July that year. Most notably, the microcomputer replaced the many separate components that made up the minicomputer's CPU with one integrated microprocessor chip. The earliest models such as the Altair 8800 were often sold as kits to be assembled by the user, and came with as little as 256 bytes of RAM, and no input/output devices other than indicator lights and switches, useful as a proof of concept to demonstrate what such a simple device could do. However, as microprocessors and semiconductor memory became less expensive, microcomputers in turn grew cheaper and easier to use:

- Increasingly inexpensive logic chips such as the 7400 series allowed cheap dedicated circuitry for improved user interfaces such as keyboard input, instead of simply a row of switches to toggle bits one at a time.
- Use of audio cassettes for inexpensive data storage replaced manual re-entry of a program every time the device was powered on.
- Large cheap arrays of silicon logic gates in the form of Read-only memory and EPROMs allowed utility programs and self-booting kernels to be stored within microcomputers. These stored programs could automatically load further more complex software from external storage devices without user intervention, to form an inexpensive turnkey system that does not require a computer expert to understand or to use the device.
- Random access memory became cheap enough to afford dedicating approximately 1-2 kilobytes of memory to a video display controller frame buffer, for a 40x25 or 80x25 text display or blocky color graphics on a common household television. This replaced the slow, complex, and expensive teletypewriter that was previously common as an interface to minicomputers and mainframes.

All these improvements in cost and usability resulted in an explosion in their popularity during the late 1970s and early 1980s. A large number of computer makers packaged microcomputers for use in small business applications. By 1979, many companies such as Cromemco, Processor Technology, IMSAI, Northstar, Southwest Technical Products Corporation, Ohio Scientific, Altos, Morrow Designs and others produced systems designed either for a resourceful end user or consulting firm to deliver business systems such as accounting, database management, and word processing to small businesses. This allowed businesses unable to afford leasing of a minicomputer or time-sharing service the opportunity to automate business functions, without (usually) hiring a full-time staff to operate the computers. A representative system of this era would have used an S100 bus, an 8-bit processor such as a Intel 8080 or Zilog Z80, and either CP/M or MP/M operating system. The increasing availability and power of desktop computers for personal use attracted the attention of more software developers. In time, and as the industry matured, the market for personal computers standardized around IBM PC compatibles running DOS, and later Windows. Modern desktop computers, video game consoles, laptops, tablet PCs, and many types of handheld devices, including mobile phones, pocket calculators, and industrial embedded systems, may all be considered examples of microcomputers according to the definition given above.

Colloquial use of the term

Everyday use of the expression "microcomputer" (and in particular the "micro" abbreviation) has declined significantly from the mid-1980s onwards, and is no longer commonplace. It is most commonly associated with the first wave of all-in-one 8-bit home computers and small business microcomputers (such as the Apple II, Commodore 64, BBC Micro, and TRS 80). Although, or perhaps because, an increasingly diverse range of modern microprocessor-based devices fit the definition of "microcomputer," they are no longer referred to as such in everyday speech.

In common usage, "microcomputer" has been largely supplanted by the description "personal computer" or "PC," which describes that it has been designed to be used by one person at a time. IBM first promoted the term "personal computer" to differentiate themselves from other microcomputers, often called "home computers", and also IBM's own mainframes and minicomputers. Unfortunately for IBM, the microcomputer itself was widely imitated, as well as the term. The component parts were commonly available to producers and the BIOS was reverse engineered through cleanroom design techniques. IBM PC compatible "clones" became commonplace, and the terms "personal computer," and especially "PC" stuck with the general public.

Since the advent of microcontrollers (monolithic integrated circuits containing RAM, ROM and CPU all onboard), the term "micro" is more commonly used to refer to that meaning.

Description

Monitors, keyboards and other devices for input and output may be integrated or separate. Computer memory in the form of RAM, and at least one other less volatile, memory storage device are usually combined with the CPU on a system bus in one unit. Other devices that make up a complete microcomputer system include batteries, a power supply unit, a keyboard and various input/output devices used to convey information to and from a human operator (printers, monitors, human interface devices). Microcomputers are designed to serve only one user at a time, although they can often be modified with software or hardware to concurrently serve more than one user. Microcomputers fit well on or under desks or tables, so that they are within easy access of users. Bigger computers like minicomputers, mainframes, and supercomputers take up large cabinets or even dedicated rooms.

A microcomputer comes equipped with at least one type of data storage, usually RAM. Although some microcomputers (particularly early 8-bit home micros) perform tasks using RAM alone, some form of secondary storage is normally desirable. In the early days of home micros, this was often a data cassette deck (in many cases as an external unit). Later, secondary storage (particularly in the form of floppy disk and hard disk drives) were built into the microcomputer case.

History

A collection of early microcomputers, including a Processor Technology SOL-20 (top shelf, right), an MITS Altair 8800 (second shelf, left), a TV Typewriter (third shelf, center), and an Apple I in the case at far right.

Although they contained no microprocessors but were built around transistor-transistor logic (TTL), Hewlett-Packard calculators as far back as 1968 had various levels of programmability such that they could be called microcomputers. The HP 9100B (1968) had rudimentary conditional (if) statements, statement line numbers, jump statements (go to), registers that could be used as variables, and primitive subroutines. The programming language resembled Assembly language in many ways. Later models incrementally added more features, including the BASIC programming language (HP 9830A in 1971). Some models had tape storage and small printers. However, displays were limited to one line at a time. The HP 9100A was referred to as a personal computer in an advertisement in a 1968 Science magazine but that advertisement was quickly dropped. It is suspected that HP was reluctant to call them "computers" because it would complicate government procurement and export procedures.

The Datapoint 2200, made by CTC in 1970, is perhaps the best candidate for the title of "first microcomputer". While it contains no microprocessor, it used the 4004 programming instruction set and its custom TTL was the basis for the Intel 8008, and for practical purposes the system behaves approximately as if it contains an 8008. This is because Intel was the contractor in charge of developing the Datapoint's CPU but ultimately CTC rejected the 8008 design because it needed 20 support chips.

Another early system, the Kenbak-1, was released in 1971. Like the Datapoint 2200, it used discrete transistor–transistor logic instead of a microprocessor, but functioned like a microcomputer in most ways. It was marketed as an educational and hobbyist tool, but was not a commercial success; production ceased shortly after introduction.

In 1972 a Sacramento State University team led by Bill Pentz built the Sac State 8008 computer, able to handle thousands of patients' medical records. The Sac State 8008 was designed with the Intel 8008 8-bit microprocessor. It had a full set of hardware and software components: a disk operating system included in a series of programmable read-only memory chips (PROMs); 8 Kilobytes of RAM; IBM's Basic Assembly Language (BAL); a hard drive; a color display; a printer output; a 150bps serial interface for connecting to a mainframe; and even the world's first microcomputer front panel.

Another system of note is the Micral-N, introduced in 1973 by a French company and powered by the 8008; it was the first microcomputer sold completely assembled and not as a construction kit.

Virtually all early microcomputers were essentially boxes with lights and switches; one had to read and understand binary numbers and machine language to program and use them (the Datapoint 2200 was a striking exception, bearing a modern design based on a monitor, keyboard, and tape and disk drives). Of the early "box of switches"-type microcomputers, the MITS Altair 8800 (1975) was arguably the most famous. Most of these simple, early microcomputers were sold as electronic kits--bags full of loose components which the buyer had to solder together before the system could be used.

The period from about 1971 to 1976 is sometimes called the first generation of microcomputers. These machines were for engineering development and hobbyist personal use. In 1975, the Processor Technology SOL-20 was designed, which consisted of one board which included all the parts of the computer system. The SOL-20 had built-in EPROM software which eliminated the need for rows of switches and lights. The MITS Altair just mentioned played an instrumental role in sparking significant hobbyist interest, which itself eventually led to the founding and success of many well-known personal computer hardware and software companies, such as Microsoft and Apple

Computer. Although the Altair itself was only a mild commercial success, it helped spark a huge industry.

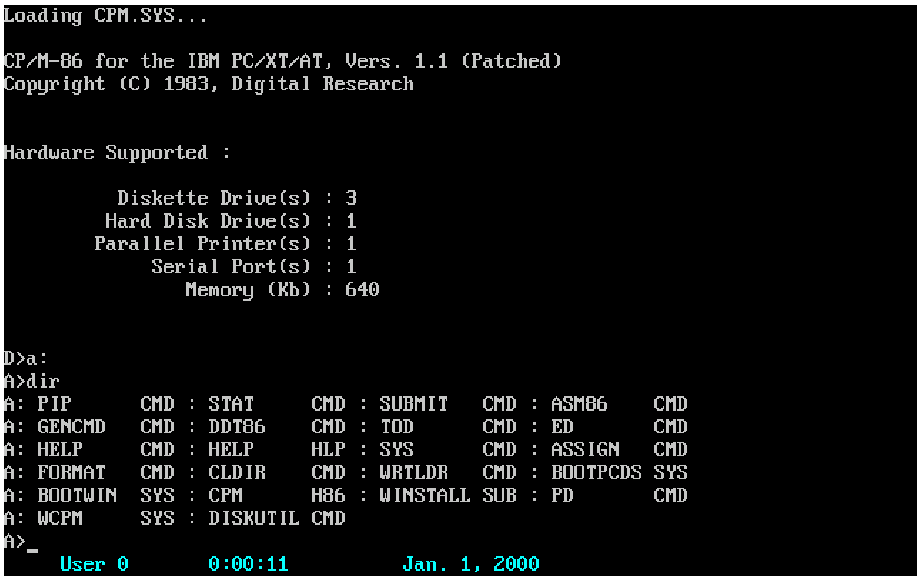
1977 saw the introduction of the second generation, known as home computers. These were considerably easier to use than their predecessors, which operation often demanded thorough familiarity with practical electronics. The ability to connect to a monitor (screen) or TV set allowed visual manipulation of text and numbers. The BASIC language, which was easier to learn and use than raw machine language, became a standard feature. These features were already common in minicomputers, with which many hobbyists and early produces were familiar.

1979 saw the launch of the VisiCalc spreadsheet (initially for the Apple II) that first turned the microcomputer from a hobby for computer enthusiasts into a business tool. After the 1981 release by IBM of their IBM PC, the term personal computer became generally used for microcomputers compatible with the IBM PC architecture (PC compatible).

Chapter 2

CP/M

CP/M



```
Loading CPM.SYS...

CP/M-86 for the IBM PC/XT/AT, Vers. 1.1 (Patched)
Copyright (C) 1983, Digital Research


Hardware Supported :

          Diskette Drive(s) : 3
         Hard Disk Drive(s) : 1
        Parallel Printer(s) : 1
             Serial Port(s) : 1
                Memory (Kb) : 640


D>a:
A>dir
A: PIP      CMD : STAT     CMD : SUBMIT   CMD : ASM86    CMD
A: GENCMD   CMD : DDT86    CMD : TOD      CMD : ED       CMD
A: HELP     CMD : HELP     HLP : SYS      CMD : ASSIGN   CMD
A: FORMAT   CMD : CLDIR    CMD : WRTLDR   CMD : BOOTPCDS SYS
A: BOOTWIN  SYS : CPM      H86 : WINSTALL SUB : PD       CMD
A: WCPM     SYS : DISKUTIL CMD
A>_
     User 0       0:00:11         Jan. 1, 2000
```

A screenshot of CP/M-86.

Company / developer	Digital Research, Inc. / Gary Kildall
Working state	Historic
Source model	Originally closed source, now open source
Latest stable release	3.1 / 1982
Available programming languages(s)	Assembler, BASIC, Modula-2, Pascal etc.
Supported platforms	Intel 8080, Intel 8085, Zilog Z80, Intel 8086, Motorola 68000

Kernel type	Monolithic kernel
Default user interface	Command line interface
License	Originally proprietary, now BSD-like

CP/M (Control Program for Microcomputers) is an operating system originally created for Intel 8080/85 based microcomputers by Gary Kildall of Digital Research, Inc. Initially confined to single-tasking on 8-bit processors and no more than 64 kilobytes of memory, later versions of CP/M added multi-user variations, and were migrated to 16-bit processors.

The combination of CP/M and S-100 bus computers patterned on the MITS Altair was an early "industry standard" for microcomputers, and this computer platform was widely used in business through the late 1970s and into the mid-1980s. By greatly reducing the amount of programming required to install an application on a new manufacturer's computer, CP/M increased the market size for both hardware and software.

Hardware model

A minimal 8-bit CP/M system would contain the following components:

- A computer terminal using the ASCII character set (very early systems used a teleprinter instead)
- An Intel 8080 (and later the 8085) or Zilog Z80 microprocessor
- At least 16 kilobytes of RAM
- A means to bootstrap the first sector of the diskette
- At least one floppy disk drive

The only hardware system that CP/M, as sold by Digital Research, would support was the Intel 8080 Development System. Manufacturers of CP/M compatible systems customized portions of the operating system for their own combination of installed memory, disk drives, and console devices. CP/M would also run on systems based on the Zilog Z80 processor since the Z80 was able to execute 8080 code. While the Digital Research distributed core of CP/M (BDOS, CCP, core transient commands; see below) did not use any of the Z80-specific instructions, many Z80 based systems used Z80 code in the system specific BIOS, and many applications were dedicated to Z80 based CP/M machines.

On most machines the "bootstrap" was a minimal bootloader in ROM; for others, this bootstrap had to be entered into memory using front panel controls each time the system was started.

CP/M used the 7-bit ASCII set. The other 128 characters made possible by the 8-bit byte were not standardized. For example, one Kaypro used them for Greek characters, and Osborne machines used the 8th bit set to indicate an underlined character. International

CP/M systems most commonly used the ISO 646 norm for localized character sets, replacing certain ASCII characters with localized characters rather than adding them beyond the 7-bit boundary.

Components of the operating system

In the 8-bit versions, while running, the CP/M operating system loaded into memory had three components:

- basic input/output system or BIOS,
- basic disk operating system or BDOS,
- console command processor or CCP.

The BIOS and BDOS were memory resident, while the CCP was memory resident unless overwritten by an application, in which case it was automatically reloaded after the application finished running. A number of transient commands for standard utilities were also provided. The transient commands resided in files with the extension .COM on disk.

The BIOS directly controlled hardware components other than the CPU and main memory. It contained functions such as character input and output and the reading and writing of disk sectors. The BDOS implemented the CP/M file system and some input/output abstractions (such as redirection) on top of the BIOS. The CCP took user commands and either executed them directly (internal commands such as DIR to show a directory or ERA to delete a file) or loaded and started an executable file of the given name (transient commands such as PIP.COM to copy files or STAT.COM to show various file and system information). Third-party applications for CP/M were also transient commands.

The BDOS, CCP and standard transient commands were (ideally) the same in all installations of a particular revision of CP/M, but the BIOS portion was always adapted to the particular hardware. Adding memory to a computer, for example, meant that the CP/M system had to be reinstalled. Once installed, the operating system (BIOS, BDOS and CCP) was stored in reserved areas at the beginning of any disk which would be used to boot the system. On start-up, the bootloader (usually contained in a ROM firmware chip) would load the operating system from the disk in drive `A:`.

By modern standards CP/M was primitive, owing to the extreme constraints on program size. With version 1.0 there was no provision for detecting a changed disk. If a user changed disks without manually rereading the disk directory the system would write on the new disk using the old disk's directory information, ruining the data stored on the disk. Starting with 1.1 or 1.2 this danger was reduced: if one changed disks without reading the new disk's directory, and tried to write to it, the operating system would signal a fatal error, avoiding overwriting but requiring a reboot (which took no more than a few seconds, but implied losing whatever data you were trying to save).

The majority of the complexity in CP/M was isolated in the BDOS, and to a lesser extent, the CCP and transient commands. This meant that by porting the limited number of simple routines in the BIOS to a particular hardware platform, the entire OS would work. This significantly reduced the development time needed to support new machines, and was one of the main reasons for CP/M's widespread use. Today this sort of abstraction is common to most OSs (a hardware abstraction layer), but at the time of CP/M's birth, OSs were typically intended to run on only one machine platform, and multilayer designs were considered unnecessary.

Command processor

The Console Command Processor, or CCP, accepted input from the keyboard and conveyed results to the terminal. All CP/M commands had to be typed in on the "command line" — the console would show most often `A>` and would await input from the user.

CP/M's command line interface was patterned after the operating systems from Digital Equipment, such as RSTS/E for the PDP-11.

Commands took the form of a keyword followed by a list of parameters separated by spaces or special characters. If an internal command was recognized, it was carried out by the CCP itself. Otherwise it would attempt to find an executable file on the currently logged disk drive and (in later versions) user area, load it, and pass it any additional parameters from the command line. These were referred to as "transient" programs. On completion, CP/M would reload the part of the CCP that had been overwritten by application programs — this allowed transient programs a larger memory space.

The commands themselves could sometimes be obscure. For instance, the command to duplicate files was named `PIP` (Peripheral-Interchange-Program), the name of the old DEC utility used for that purpose. The format of parameters given to a program was not standardized, so that there was no single "option character" that differentiated options from file names.

Basic disk operating system

The Basic Disk Operating System, or BDOS, provided access to such operations as opening a file, output to the console, or printing. Application programs would load processor registers with a function code for the operation, and addresses for parameters or memory buffers, and call a fixed address in memory. Since the address was the same independent of the amount of memory in the system, application programs would run the same way for any type or configuration of hardware.

Basic input output system

The Basic Input Output System, or BIOS, provided the lowest level functions required by the operating system. These included reading or writing single characters to the system consoles and reading or writing a sector of data from the disk. The BDOS handled some of the buffering of data from the diskette, but before CP/M 3.0 it assumed a fixed disk sector size of 128 Bytes, as used on single-density 8-inch floppy disks. Since most 5.25-inch disk formats used larger sectors, the blocking and deblocking and the management of a disk buffer area was handled by model-specific code in the BIOS.

File system

File names were specified as a string of up to eight characters, followed by a period, followed by a file name extension of up to three characters ("8.3" filename format). The extension usually identified the type of the file. For example, `.COM` indicated a binary executable program file, and `.TXT` indicated a file containing ASCII text.

Each disk drive was identified by a drive letter. Typically, dual floppy disk drives would be identified as drives `A` and `B`. To refer to a file on a specific drive, the drive letter was prepended to the file name, separated by a colon, e.g. `A:FILE.TXT`.With no drive letter prefixed, the default was to use files on the current drive (for example, a program running off a B drive would access files on the B drive unless the file names were prefixed with a drive letter).

File size was specified as the number of 128-byte *records* (directly corresponding to disk sectors on 8-inch drives) occupied by a file on the disk. There was no generally supported way of specifying byte-exact file sizes. The current size of a file was maintained in the file's file control block (FCB) by the operating system. Since many application programs (such as text editors) prefer to deal with files as sequences of characters rather than as sequences of records, by convention text files were terminated with a control-Z character (ASCII SUB, hexadecimal 1A). Determining the end of a text file therefore involved examining the last record of the file to locate the terminating control-Z. This also meant that inserting a control-Z character into the middle of a file usually had the effect of truncating the text contents of the file.

File modification times (timestamps) were not supported, although some later variants of CP/M added this feature as an extension.

CP/M 2.2 had no sub-directories in the file structure, but provided 16 "user areas" to organize files on a disk. The user area concept was to make the single-user version of CP/M somewhat compatible with multi-user MP/M systems. A common patch for the CP/M and derivative operating systems was to make one user area accessible to the user independent of the currently set user area. A supported command was "USER x" where x is a value from 0 to 15. User 0 was the default. If one changed to another user, such as USER 1, the material saved on the disk for this user would only be available to USER 1; USER 2 would not be able to see it or access it. Since CP/M was a single-user operating system, no security was provided for the user command; nothing would prevent any user from accessing any of the 16 user areas. The user area feature arguably had little utility on small floppy disks, though it was more useful for organizing files on machines equipped with large hard drives.

History

The beginning and CP/M's heyday

Gary Kildall originally developed CP/M during 1973-74, as an operating system to run on an Intel Intellec-8 development system, equipped with an Shugart Associates 8-inch floppy disk drive interfaced via a custom floppy disk controller. It was written in Kildall's own PL/M (*Programming Language for Microcomputers*). Various aspects of CP/M were influenced by the TOPS-10 operating system of the DECsystem-10 mainframe computer, which Kildall had used as a development environment.

The name

CP/M originally stood for "Control Program/Monitor". However, during the conversion of CP/M to a commercial product, trademark registration documents filed in November 1977 gave the product's name as "Control Program for Microcomputers". The CP/M name shows a prevailing naming scheme of the time, as in Kildall's PL/M language, and Prime Computer's PL/P (*Programming Language for Prime*), both suggesting IBM's PL/I; and IBM's CP/CMS operating system, which Kildall had used when working at the Naval Postgraduate School.

This renaming of CP/M was part of a larger effort by Kildall and his business-partner wife to convert Kildall's personal project of CP/M and the Intel-contracted PL/M compiler into a commercial enterprise. The Kildalls astutely intended to establish the Digital Research brand and its product lines as synonymous with "microcomputer" in the consumer's mind, similar to what IBM and Microsoft together later successfully accomplished in making "personal computer" synonymous with IBM and Microsoft product offerings. Intergalactic Digital Research, Inc. was later renamed via a corporation change-of-name filing to Digital Research, Inc.

Portability

CP/M was described as a "software bus", allowing multiple programs to interact with different hardware in a standardized way. Programs written for CP/M were typically portable between different machines, usually only requiring specification of the escape sequence for control of the screen and printer. This portability made CP/M popular, and much more software was written for CP/M than for operating systems that only ran on one brand of hardware. One restriction on portability was that certain programs used the extended instruction set of the Z80 processor and would not operate on an 8080 or 8085 processor.

Many different brands of machines ran CP/M, some notable examples being the Altair 8800, the IMSAI 8080, the Osborne 1 and Kaypro portables, and MSX computers. Even the Apple II could run CP/M when an extra Z80 card was installed. The best-selling CP/M-capable system of all time was probably the Commodore 128, although few people actually used its CP/M abilities. In the UK, CP/M was also available on Research

Machines educational computers (with the CP/M source code published as an educational resource), and for the BBC Micro when equipped with a Z80 co-processor. Furthermore, it powered the popular Amstrad PCW word-processing system and was available for the Amstrad CPC series and later models of the ZX Spectrum.

Applications

WordStar, one of the first widely used word processors, and dBASE, an early and popular database program for small computers, were originally written for CP/M. An early outliner, KAMAS (Knowledge and Mind Amplification System) was also written for CP/M, though later rewritten for MS-DOS. Turbo Pascal, the ancestor of Borland Delphi, and Multiplan, the ancestor of Microsoft Excel, also debuted on CP/M before MS-DOS versions became available. AutoCAD, a CAD application from Autodesk debuted on CP/M. A host of compilers and interpreters for popular programming languages of the time (such as BASIC and FORTRAN) were available, among them several of the earliest Microsoft products. The lack of standardized graphics support severely limited gaming, but various character and text-based games were ported, such as Hamurabi, Lunar Lander and Colossal Cave Adventure, along with other early interactive fiction. Lifeboat Associates started collecting and distributing user-written "free" software. One of the first was XMODEM, which allowed communication via modem and phone line.

Disk formats

While the 8-inch single density floppy disk format (so-called "distribution format") was standardized, various 5¼ inch formats were used depending on the characteristics of particular systems and to some degree the choices of the designers. CP/M supported options to control the size of reserved and directory areas on the disk, and the mapping between logical disk sectors (as seen by CP/M programs) and physical sectors as allocated on the disk. There were very many ways to customize these parameters for every system but once they had been set, no standardized way existed to for a system to load parameters from a disk formatted on another system. No single manufacturer prevailed in the 5¼ inch era of CP/M use, and disk formats were not portable between hardware manufacturers. A software manufacturer had to prepare a separate version of the program for each brand of hardware on which it was to run. With some manufacturers (Kaypro is an example), there was not even standardization across the company's different models. Because of this situation, disk format translation programs, which allowed a machine to read many different formats, became popular and reduced the confusion, as did programs like kermit which allowed transfer of data and programs from one machine to another using the serial ports that most CP/M machines had. The fragmented CP/M market, requiring distributors either to stock multiple formats of disks or to invest in multiformat duplication equipment, compared with the more standardized IBM PC disk formats, was a contributing factor to the rapid obsolescence of CP/M after 1981.

Graphics

Although graphics-capable S100 systems existed from the commercialization of the S100 bus, CP/M did not provide any standardized graphics support until the release of CP/M 3.0 with GSX (Graphic System eXtension). Owing to the small memory available, graphics was never a common feature associated with 8-bit CP/M operating systems. Most systems could only display rudimentary ascii art charts and diagrams in text mode or by using a custom character set.

The 16-bit world

Versions of CP/M were later completed for some 16-bit CPUs as well, although they required the application programs to be re-compiled for the new CPUs—or, if they were written in assembly language, to be largely rewritten from scratch.

One of the first was **CP/M-86** for the Intel 8086, which was soon followed by **CP/M-68k** for the Motorola 68000. At this point the original 8-bit CP/M became known by the retronym **CP/M-80** to avoid confusion. There was also a port to the Zilog Z8000, named **CP/M-8000**.

CP/M-68k was initially used in the Atari ST computer, but Atari decided to go with a newer DOS called GEMDOS. It also was used on the SORD M68 and M68MX computers. CP/M-86 was expected to be the standard operating system of the new IBM PCs, but DRI and IBM were unable to negotiate development and licensing terms. IBM turned to Microsoft instead, and Microsoft delivered PC-DOS based on a CP/M "clone," 86-DOS. Although CP/M-86 became an option for the IBM PC after DRI threatened legal action, it never overtook Microsoft's system.

When Digital Equipment Corporation put out the Rainbow 100 to compete with IBM, it came with CP/M-80 using a Z80 chip, and CP/M-86 or MS-DOS using an 8088 microprocessor. The Z80 and 8088 CPUs ran concurrently. A benefit of the Rainbow was that it could continue to run 8-bit CP/M software, preserving a user's possibly sizable investment as they moved into the 16-bit world of MS-DOS.

MS-DOS takes over

In 1980 IBM approached Digital Research to license a forthcoming version of CP/M for their new product, the IBM Personal Computer, but on their failure to obtain a signed non-disclosure agreement, the talks failed, and IBM instead used Microsoft to provide an operating system.

Many of the basic concepts and internal mechanisms of early versions of MS-DOS resembled those of CP/M. Internals like file-handling data structures were identical, and both referred to disk drives with a letter (`A:`, `B:`, etc.). MS-DOS's main innovation was its FAT file system. This similarity made it easier to port popular CP/M software like WordStar and dBase. However, CP/M's concept of separate user areas for files on the same disk was never ported to MS-DOS. Since MS-DOS had access to more memory (as few IBM PCs were sold with less than 64 KB of memory, while CP/M had to run in 16 KB if necessary), more commands were built in to the command-line user interface logic, making MS-DOS somewhat faster and easier to use on floppy-based computers.

CP/M rapidly lost market share as the microcomputing market moved to the PC platform, and it never regained its former popularity. Byte magazine, at the time one of the leading industry magazines for microcomputers, essentially ceased covering CP/M products within a few years of the introduction of the IBM PC. For example, in 1983 there were still a few advertisements for S100 boards and articles on CP/M software, but by 1987 these were no longer found in the magazine. InfoWorld magazine described the unsuccessful efforts at introducing CP/M based home computers in 1983 and in 1986 stated that the Kaypro corporation had stopped production of their 8-bit CP/M-based models to concentrate on sales of MS DOS compatible systems, long after most other vendors had ceased production of new equipment and software for CP/M.

Later versions of CP/M-86 made significant strides in performance and usability and were made compatible with MS-DOS. For some time in the 1980s, the resulting system was considered to be a better x86 OS than MS-DOS. To reflect this compatibility the name was changed, and CP/M-86 became DOS Plus, which in turn became DR-DOS.

ZCPR

ZCPR (the Z80 Command Processor Replacement) was introduced on February 2, 1982 as a drop-in replacement for the standard Digital Research console command processor (CCP) and was initially written by a group of computer hobbyists who called themselves "The CCP Group". They were Frank Wancho, Keith Petersen (the archivist behind Simtel at the time), Ron Fowler, Charlie Strom, Bob Mathias, and Richard Conn. Richard was, in fact, the driving force in this group (all of whom maintained contact through email).

ZCPR1 was released on a disk put out by SIG/M (Special Interest Group/Microcomputers), a part of the Amateur Computer Club of New Jersey.

ZCPR2 was released on February 14, 1983. It was released as a set of ten disks from SIG/M. ZCPR2 was upgraded to 2.3, and also was released in 8080 code, permitting the use of ZCPR2 on 8080 and 8085 systems.

ZCPR3 was released on Bastille Day, July 14, 1984, as a set of nine disks from SIG/M. The code for ZCPR3 could also be compiled (with reduced features) for the 8080 and would run on systems that did not have the requisite Z80 microprocessor.

In January 1987, Richard Conn decided to stop developing ZCPR, and Echelon asked Jay Sage (who already had a privately enhanced ZCPR 3.1) to continue work on ZCPR. Thus, ZCPR 3.3 was developed and released. ZCPR33 no longer supported the 8080 series of microprocessors, and added the most features of any upgrade in the ZCPR line.

Features of ZCPR as of version 3 included:

- shells
- aliases
- I/O redirection
- flow control
- named directories
- search paths
- custom menus
- passwords
- on line help

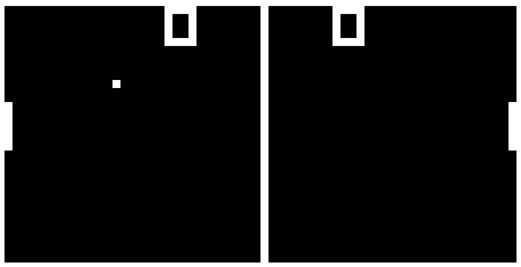
ZCPR3.3 also included a full complement of utilities with considerably extended capabilities. While enthusiastically supported by the CP/M user base of the time, ZCPR alone was insufficient to slow the demise of CP/M.

Legacy

A number of behaviors exhibited by modern versions of Microsoft Windows are a result of backwards compatibility to MS-DOS which attempted some backwards compatibility with CP/M. The 8.3 filename standard in MS-DOS and Windows 3.x was originally adopted by CP/M. The wildcard matching algorithm used by the Windows Command Prompt is based on that of CP/M, as are the reserved filenames used to redirect output to a printer ("PRN"), the console ("CON"), or a serial device ("COM1"). Also, the character marking the end of some text files and behavior exhibited by the copy command can also be attributed to CP/M.

Chapter 3

CHIP-8



Screenshot of Pong implemented in CHIP-8

CHIP-8 is an interpreted programming language, developed by the late Joseph Weisbecker. It was initially used on the COSMAC VIP and Telmac 1800 8-bit microcomputers in the mid-1970s. CHIP-8 programs are run on a CHIP-8 virtual machine. It was made to allow video games to be more easily programmed for said computers.

Roughly twenty years after CHIP-8 was introduced, derived interpreters appeared for some models of graphing calculators (from the late 1980s onward, these handheld devices in many ways have more computing power than most mid-1970s microcomputers for hobbyists).

An active community of users and developers existed in the late 1970s, beginning with ARESCO's "VIPer" newsletter whose first three issues revealed the machine code behind the CHIP-8 interpreter.

CHIP-8 applications

There are a number of classic video games ported to CHIP-8, such as Pong, Space Invaders, Tetris, and Pac-Man. There's also a random maze generator available. These programs are reportedly placed in the public domain, and can be easily found on the Internet.

CHIP-8 today

There is a CHIP-8 implementation for almost every platform imaginable, as well as some development tools. Despite this, there are only a small number of games for the CHIP-8.

CHIP-8 has a descendant called SCHIP (Super Chip), introduced by Erik Bryntse. In 1990, a CHIP-8 interpreter called CHIP-48 was made for HP-48 graphing calculators so that games could be programmed more easily. Its extensions to CHIP-8 are what became known as SCHIP. It features a larger resolution and several additional opcodes which make programming easier. If it were not for the development of the CHIP-48 interpreter, CHIP-8 would not be as well known today.

The next most influential developments (which popularized S/CHIP-8 on many other platforms) were David Winter's emulator, disassembler, and extended technical documentation. It laid out a complete list of undocumented opcodes and features, and was distributed across many hobbyist forums. Many of the emulators listed below had these works as a starting point.

Virtual machine description

Memory

CHIP-8's memory addresses range from 200h to FFFh, making for 3,584 bytes. The reason for the memory starting at 200h is that on the Cosmac VIP and Telmac 1800, the first 512 bytes are reserved for the interpreter. On those machines, the uppermost 256 bytes (F00h-FFFh on a 4K machine) were reserved for display refresh, and the 96 bytes below that (EA0h-EFFh) were reserved for the call stack, internal use, and the variables.

Registers

CHIP-8 has 16 8-bit data registers named from V0 to VF. The VF register doubles as a carry flag.

The address register, which is named I, is 16 bits wide and is used with several opcodes that involve memory operations.

The stack

The stack is only used to store return addresses when subroutines are called. The original 1802 version allocated 48 bytes for up to 12 levels of nesting; modern implementations normally have at least 16 levels.

Timers

CHIP-8 has two timers. They both count down at 60 hertz, until they reach 0.

- Delay timer: This timer is intended to be used for timing the events of games. Its value can be set and read.
- Sound timer: This timer is used for sound effects. When its value is nonzero, a beeping sound is made.

Input

Input is done with a hex keyboard that has 16 keys which range from 0 to F. The '8', '4', '6', and '2' keys are typically used for directional input. Three opcodes are used to detect input. One skips an instruction if a specific key is pressed, while another does the same if a specific key is *not* pressed. The third waits for a key press, and then stores it in one of the data registers.

Graphics and sound

Display resolution is 64×32 pixels, and color is monochrome. Graphics are drawn to the screen solely by drawing sprites, which are 8 pixels wide and may be from 1 to 15 pixels in height. Sprite pixels that are set flip the color of the corresponding screen pixel, while unset sprite pixels do nothing. The carry flag (VF) is set to 1 if any screen pixels are flipped from set to unset when a sprite is drawn and set to 0 otherwise.

As previously described, a beeping sound is played when the value of the sound timer is nonzero.

Opcode table

CHIP-8 has 35 opcodes, which are all two bytes long. They are listed below, in hexadecimal and with the following symbols:

- NNN: address
- NN: 8-bit constant
- N: 4-bit constant
- X and Y: 4-bit register identifier

Opcode	Explanation
0NNN	Calls RCA 1802 program at address NNN.
00E0	Clears the screen.
00EE	Returns from a subroutine.
1NNN	Jumps to address NNN.
2NNN	Calls subroutine at NNN.
3XNN	Skips the next instruction if VX equals NN.
4XNN	Skips the next instruction if VX doesn't equal NN.

5XY0	Skips the next instruction if VX equals VY.
6XNN	Sets VX to NN.
7XNN	Adds NN to VX.
8XY0	Sets VX to the value of VY.
8XY1	Sets VX to VX or VY.
8XY2	Sets VX to VX and VY.
8XY3	Sets VX to VX xor VY.
8XY4	Adds VY to VX. VF is set to 1 when there's a carry, and to 0 when there isn't.
8XY5	VY is subtracted from VX. VF is set to 0 when there's a borrow, and 1 when there isn't.
8XY6	Shifts VX right by one. VF is set to the value of the least significant bit of VX before the shift.
8XY7	Sets VX to VY minus VX. VF is set to 0 when there's a borrow, and 1 when there isn't.
8XYE	Shifts VX left by one. VF is set to the value of the most significant bit of VX before the shift.
9XY0	Skips the next instruction if VX doesn't equal VY.
ANNN	Sets I to the address NNN.
BNNN	Jumps to the address NNN plus V0.
CXNN	Sets VX to a random number and NN.
DXYN	Draws a sprite at coordinate (VX, VY) that has a width of 8 pixels and a height of N pixels. Each row of 8 pixels is read as bit-coded starting from memory location I; I value doesn't change after the execution of this instruction. As described above, VF is set to 1 if any screen pixels are flipped from set to unset when the sprite is drawn, and to 0 if that doesn't happen.
EX9E	Skips the next instruction if the key stored in VX is pressed.
EXA1	Skips the next instruction if the key stored in VX isn't pressed.
FX07	Sets VX to the value of the delay timer.
FX0A	A key press is awaited, and then stored in VX.
FX15	Sets the delay timer to VX.
FX18	Sets the sound timer to VX.
FX1E	Adds VX to I.
FX29	Sets I to the location of the sprite for the character in VX. Characters 0-F (in hexadecimal) are represented by a 4x5 font.
FX33	Stores the Binary-coded decimal representation of VX at the addresses I, I plus

	1, and I plus 2.
FX55	Stores V0 to VX in memory starting at address I.
FX65	Fills V0 to VX with values from memory starting at address I.

Chapter 4

Dbase

Vulcan dBase-II

Paradigm	Imperative, Declarative
Appeared in	1979
Developer	C. Wayne Ratliff
Influenced	Clipper, Paradox, Harbour

dBase II was the first widely used database management system (DBMS) for microcomputers. It was originally published by Ashton-Tate for CP/M, and later on ported to the Apple II and IBM PC under DOS. On the PC platform in particular it became one of the best-selling software titles for a number of years, propelling Ashton-Tate to become one of the "big three" software publishers in the early business software market. A major upgrade was released as dBASE III, and ported to a wider variety of platforms, adding UNIX, and VMS.

Starting in the mid 1980s many other companies produced their own dialects or variations on the product and language. These included FoxPro and Clipper/Harbour, together informally referred to as xBase. Many of these were technically stronger than dBase, but could not make a major dent in the market until dBase IV was introduced with many problems. This was coincident with an industry-wide switch to SQL and the client-server market, and the rapid introduction of Microsoft Windows in the business market. A combination of these factors led to rapid retrenchment in the xBase world, and the disappearance of Ashton-Tate with their sale to Borland in 1991. The rights to the dBase product line were sold in 1999 to the newly-formed dBase Inc. In 2004, dBase Inc. changed its name to dataBased Intelligence, Inc.

dBase's underlying file format, the `.dbf` file, is widely used in many other applications needing a simple format to store structured data.

Recent history

dBase has evolved into a modern object oriented language that runs on 32 bit Windows. It can be used to build a wide variety of applications including web apps hosted on a Windows server, Windows rich client applications, and middleware applications. dBase can access most modern database engines via ODBC drivers.

dBase features an IDE with a Command Window and Navigator, a just in time compiler, a preprocessor, a virtual machine interpreter, a linker for creating dBase application .exe's, a freely available runtime engine, and numerous two-way GUI design tools including a Form Designer, Report Designer, Menu Designer, Label Designer, Datamodule Designer, SQL Query Designer, and Table Designer. Two-way Tools refers to the ability to switch back and forth between using a GUI design tool and the Source Code Editor. Other tools include a Source Code Editor, a Project Manager that simplifies building and deploying a dBase application, and an integrated Debugger. dBase features structured exception handling and has many built-in classes that can be subclassed via single inheritance. There are visual classes, data classes, and many other supporting classes. Visual classes include Form, SubForm, Notebook, Container, Entryfield, RadioButton, SpinBox, ComboBox, ListBox, PushButton, Image, Grid, ScrollBar, ActiveX, Report, ReportViewer, Text, TextLabel and many others. Database classes include Session, Database, Query, Rowset, Field, StoredProc and Datamodule classes. Other classes include File, String, Math, Array, Date, Exception, Object and others. dBase objects can be dynamically subclassed by adding new properties to them at runtime.

The current version of dBase, dBase Plus, is not compatible with any previous DOS version, including the latest dBase V.

Origins

The original developer of dBase was C. Wayne Ratliff. In 1978, while working as a contractor at the Jet Propulsion Laboratory, Ratliff wrote a database program he called "Vulcan" (after Mr. Spock's race and homeworld on Star Trek) to help him win the office football pool. Written for his kit-built IMSAI 8080 microcomputer running PTDOS, he based the program on JPLDIS (Jet Propulsion Laboratory Display Information System), a mainframe (UNIVAC 1108) data base product developed by JPL's Jeb Long and Jack Hatfield. Long finished JPLDIS after Hatfield's departure from JPL.

According to Ratliff, the language in JPLDIS was a simple, command-driven language intended for interactive use on printing terminals. There is some evidence that JPLDIS was influenced by Tymshare Corporation's mainframe database product called RETRIEVE.

In early 1980, George Tate, of Ashton-Tate, entered into a marketing agreement with Ratliff. Vulcan was renamed dBase II, and the software quickly became a huge success.

dBase programming language

After writing Vulcan for the IMSAI 8080 and later porting it to CP/M and MS-DOS (as dBase), Ratliff added commands to accommodate the video screen interface as well as commands for improved control of flow (such as DO WHILE/ENDDO) and conditional logic (such as IF/ENDIF).

For handling data, dBase provided detailed procedural commands and functions to open and traverse records in data files (e.g., USE, SKIP, GO TOP, GO BOTTOM, and GO recno), manipulate field values (REPLACE and STORE), and manipulate text strings (e.g., STR() and SUBSTR()), numbers, and dates. Its ability to simultaneously open and manipulate multiple files containing related data led Ashton-Tate to label dBase a "relational database" although it did not meet the criteria defined by Dr. Edgar F. Codd's relational model; it could more accurately be called an application development language and integrated navigational database management system that is influenced by relational concepts.

dBase used a runtime interpreter architecture, which allowed the user to execute commands by typing them in a command line "dot prompt." Upon typing a command or function and pressing the return key, the interpreter would immediately execute or evaluate it. Similarly, program scripts (text files with PRG extensions) ran in the interpreter (with the DO command), where each command and variable was evaluated at runtime. This made dBase programs quick and easy to write and test because programmers didn't have to first compile and link them before running them. (For other languages, these steps were tedious in the days of single- and double-digit megahertz CPUs.) The interpreter also handled automatically and dynamically all memory management (i.e., no preallocating memory and no hexadecimal notation), which more than any other feature made it possible for a business person with no programming experience to develop applications.

Conversely, the ease and simplicity of dBase presented a challenge as its users became more expert and as professional programmers were drawn to it. More complex and more critical applications demanded professional programming features for greater reliability and performance, as well as greater developer productivity.

Over time, Ashton-Tate's competitors introduced so-called clone products and compilers that introduced more robust programming features such as user-defined functions (UDFs) to supplement the built-in function set, scoped variables for writing routines and functions that were less likely to be affected by external processes, arrays for complex data handling, packaging features for delivering applications as executable files without external runtime interpreters, object-oriented syntax, and interfaces for accessing data in remote database management systems. Ashton-Tate also implemented many of these features with varying degrees of success. Ashton-Tate and its competitors also began to incorporate SQL, the ANSI/ISO standard language for creating, modifying, and retrieving data stored in relational database management systems.

In the late 1980s, developer groups sought to create a dBase language standard (IEEE 1192). It was then that the language started being referred to as "Xbase" to distinguish it from the Ashton-Tate product. Hundreds of books have been written on dBase and Xbase programming.

In 1988 Ashton-Tate filed suit against Fox Software and Santa Cruz Operation (SCO) for copying dBase's "structure and sequence" in FoxBase+ (SCO marketed XENIX and UNIX versions of the Fox products). In December 1990, U.S. District judge Terry Hatter, Jr. dismissed Ashton-Tate's lawsuit and invalidated Ashton-Tate's copyrights for not disclosing that dBase had been based, in part, on the public domain JPLDIS. In October 1991, while the case was still under appeal, Borland International acquired Ashton-Tate, and as one of the merger's provisions the U.S. Justice Department required Borland to end the lawsuit against Fox and allow other companies to use the dBase language without the threat of legal action.

Today, implementations of the dBase language have expanded to include many features targeted for business applications, including object-oriented programming, manipulation of remote and distributed data via SQL, Internet functionality, and interaction with modern devices.

Programming examples

The following example opens an employee table ("empl"), gives every manager who supervises 1 or more employees a 10-percent raise, and then prints the names and salaries.

```
USE empl
REPLACE ALL salary WITH salary * 1.1 FOR supervises > 0
LIST ALL fname, lname, salary TO PRINT
* (comment: reserved words shown in CAPITALS for illustration purposes)
```

Note how one does not have to keep mentioning the table name. The assumed ("current") table stays the same until told otherwise. This is in contrast to SQL which almost always needs explicit tables. Because of its origins as an interpreted interactive language, dBase used a variety of contextual techniques to reduce the amount of typing needed. This facilitated incremental, interactive development but also made larger-scale modular programming difficult. A tenet of modular programming is that the correct execution of a program module must not be affected by external factors such as the state of memory variables or tables being manipulated in other program modules. Because dBase was not designed with this in mind, developers had to be careful about porting (borrowing) programming code that assumed a certain context and it would make writing larger-scale modular code difficult. Work-area-specific references were still possible using the arrow notation ("B->customer") so that multiple tables could be manipulated at the same time. In addition, if the developer had the foresight to name their tables appropriately, they could clearly refer to a large number of tables open at the same time by notation such as ("employee->salary") and ("vacation->start_date"). Alternatively, the alias command could be appended to the initial opening of a table statement which made referencing a table field unambiguous and simple. For example. one can open a table and assign an alias to it in this fashion, "use EMP alias Employee", and henceforth, refer to table variables as "Employee->Name".

Another notable feature is the re-use of the same clauses for different commands. For example, the FOR clause limits the scope of a given command. (It is somewhat comparable to SQL's WHERE clause). Different commands such as LIST, DELETE, REPLACE, BROWSE, etc. could all accept a FOR clause to limit (filter) the scope of their activity. This simplifies the learning of the language.

dBase was also one of the first business-oriented languages to implement string evaluation.

```
i = 2
myMacro = "i + 10"
i = &myMacro
* comment: i now has the value 12
```

Here the "&" tells the interpreter to evaluate the string stored in "myMacro" as if it were programming code. This is an example of a feature that made dBase programming flexible and dynamic, sometimes called "meta ability" in the profession. This could allow programming expressions to be placed inside tables, somewhat reminiscent of formulas in spreadsheet software.

However, it could also be problematic for pre-compiling and for making programming code secure from hacking. But, dBase tended to be used for custom internal applications for small and medium companies where the lack of protection against copying, as compared to compiled software, was often less of an issue.

Interactivity

In addition to the dot-prompt, dBase III, III+ and dBase IV came packaged with an ASSIST application to manipulate data and queries, as well as a APPSGEN application which allowed the user to generate applications without resorting to code writing like a 4GL. The dBASE IV APPSGEN tool was based largely on portions of an early CP/M product named Personal Pearl.

Niches

Although the language has fallen out of favor as a primary business language, some find dBase an excellent interactive ad-hoc data manipulation tool. Whereas SQL retrieves data sets from a relational database (RDBMS), with dBase one can more easily manipulate, format, analyze and perform calculations on individual records, strings, numbers, and so on in a step-by-step imperative (procedural) way instead of trying to figure out how to use SQL's declarative operations.

Its granularity of operations is generally smaller than SQL, making it easier to split querying and table processing into easy-to-understand and easy-to-test parts. For example, one could insert a BROWSE operation between the filtering and the aggregation step to study the intermediate table or view (applied filter) before the aggregation step is applied.

As an application development platform, dBase fills a gap between lower level languages such as C, C++, and Java and high-level proprietary 4GLs (fourth generation languages) and purely visual tools, providing relative ease-of-use for business people with less formal programming skill and high productivity for professional developers willing to trade off the low-level control.

dBase remained a popular teaching tool even after sales slowed because the text-oriented commands were easier to present in printed training material than the mouse-oriented competitors. (Mouse-oriented commands were added to the product over time, but the command language remained a popular de-facto standard while mousing commands tended to be vendor-specific.)

File formats

A major legacy of dBase is its `.dbf` file format, which has been adopted in a number of other applications. For example, the shapefile format developed by ESRI for spatial data in its PC ArcInfo geographic information system, uses .dbf files to store feature attribute data.

dBase's database system was one of the first to provide a header section for describing the structure of the data in the file. This meant that the program no longer required advance knowledge of the data structure, but rather could ask the data file how it was structured. Note that there are several variations on the .dbf file structure, and all dBase-related products and .dbf file structures are not necessarily compatible.

A second filetype is the `.dbt` file format for memo fields. While character fields are limited to 254 characters each, a memo field is a 10-byte pointer into a `.dbt` file which can include a much larger text field. dBase was very limited in its ability to process memo fields, but some other xBase languages such as Clipper treat memo fields as strings just like character fields for all purposes except permanent storage.

dBase uses `.ndx` files for single indexes, and `.mdx` *(multiple-index)* files for holding between 1 and 48 indexes. Some xBase languages include compatibility with `.ndx` files while others use different file formats such as `.ntx` used by Clipper and `.idx/.cdx` used by FoxPro or FlagShip. Later iterations of Clipper included drivers for `.ndx, .mdx, .idx and .cdx` indexes.

Chapter 5

Minicomputer

PDP 7

A **minicomputer** (colloquially, **mini**) is a class of multi-user computers that lies in the middle range of the computing spectrum, in between the largest multi-user systems (mainframe computers) and the smallest single-user systems (microcomputers or personal computers). The class at one time formed a distinct group with its own hardware and operating systems, but the contemporary term for this class of system is midrange computer, such as the higher-end SPARC, POWER and Itanium -based systems from Sun Microsystems, IBM and Hewlett-Packard.

History

1960s: Origin; 1970s: Market entrenchment

The term evolved in the 1960s to describe the "small" third generation computers that became possible with the use of integrated circuit and core memory technologies. They usually took up one or a few cabinets the size of a large refrigerator or two, compared with mainframes that would usually fill a room. The first successful minicomputer was Digital Equipment Corporation's 12-bit PDP-8, which cost from US$16,000 upwards when launched in 1964. The important precursors of the PDP-8 include the PDP-5, LINC, the TX-0, the TX-2, and the PDP-1. Digital Equipment gave rise to a number of minicomputer companies along Massachusetts Route 128, including Data General, Wang Laboratories, Apollo Computer, and Prime Computer.

Mini computers were also known as midrange computers. They had relatively high processing power and capacity that mostly fit the needs of mid range organizations. They were used in manufacturing processes or handling email that was sent and received by a company. In the 70's they were the hardware that was used to launch the computer aided design, CAD, industry and other similar industries where a smaller dedicated system was needed.

The 7400 series of TTL integrated circuits started appearing in minicomputers in the late 1960s. The 74181 arithmetic logic unit (ALU) was commonly used in the CPU data paths. Each 74181 had a bus width of four bits, hence the popularity of *bit-slice* architecture. The 7400 series offered data-selectors, multiplexers, three-state buffers, memories, etc. in dual in-line packages with one-tenth inch spacing, making major system components and architecture evident to the naked eye. Starting in the 1980s, many minicomputers used VLSI circuits.

As microcomputers developed in the 1970s and 80s, minicomputers filled the mid-range area between low powered microcomputers and high capacity mainframes. At the time microcomputers were single-user, relatively simple machines running simple program-launcher operating systems like CP/M or MS-DOS, while minis were much more powerful systems that ran full multi-user, multitasking operating systems like VMS and Unix, often with timesharing versions of BASIC for application development (MAI Basic Four systems being very popular in that regard). The classical mini was a 16-bit computer, while the emerging higher performance 32-bit minis were often referred to as superminis. At the launch of the MITS Altair 8800 in 1975, *Radio Electronics* magazine referred to the system as a "minicomputer", although it would properly be called a microcomputer; as it was the first commercially available personal computer based on the single-chip microprocessor from Intel.

Mid-1980s, 1990s: The minis give way to the micros

The decline of the minis happened due to the lower cost of microprocessor based hardware, the emergence of inexpensive and easily deployable local area network systems, the emergence of the 68020, 80286 and the 80386 microprocessors, and the desire of end-users to be less reliant on inflexible minicomputer manufacturers and IT departments/“data centers”—with the result that minicomputers and dumb terminals were replaced by networked workstations and servers and PCs in the latter half of the 1980s.

During the 1990s the change from minicomputers to inexpensive PC networks was cemented by the development of several versions of Unix to run on the Intel x86 microprocessor architecture, including Solaris, FreeBSD, NetBSD and OpenBSD. Also, the Microsoft Windows series of operating systems, beginning with Windows NT, now included server versions that supported preemptive multitasking and other features required for servers.

As microprocessors have become more powerful, CPUs built up from multiple components—once the distinguishing feature differentiating mainframes and midrange systems from microcomputers—have become increasingly obsolete, even in the largest mainframe computers.

Digital Equipment Corporation was the leading minicomputer manufacturer, at one time the second largest computer company after IBM. But as the minicomputer declined in the face of generic UNIX servers and Intel-based PCs, not only DEC, but almost every other minicomputer company including Data General, Prime, Computervision, Honeywell and Wang Laboratories, many based in New England also collapsed. DEC was sold to Compaq in 1998.

The minicomputer's industrial impact and heritage

Several pioneering computer companies first built minicomputers, such as DEC, Data General, and Hewlett-Packard (HP) (who now refers to its HP3000 minicomputers as “servers” rather than “minicomputers”). And although today’s PCs and servers are clearly microcomputers physically, architecturally their CPUs and operating systems have evolved largely by integrating features from minicomputers.

In the software context, the relatively simple OSs for early microcomputers were usually inspired by minicomputer OSs (such as CP/M's similarity to Digital's RSTS) and multiuser OSs of today are often either inspired by or directly descended from minicomputer OSs (UNIX was originally a minicomputer OS, while Windows NT—the foundation for all current versions of Microsoft Windows—borrowed design ideas liberally from VMS and UNIX). Many of the first generation of PC programmers were educated on minicomputer systems.

Chapter 6

ZX Spectrum

ZX Spectrum

An original 1982 ZX Spectrum

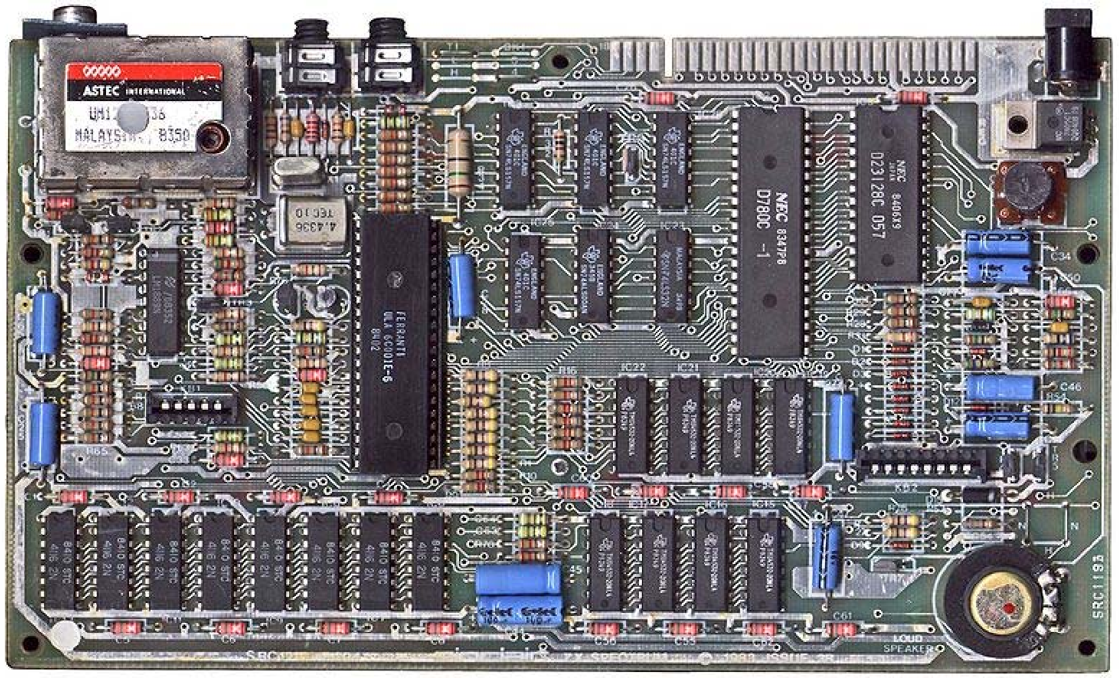
Type	8-bit Home computer
Release date	23 April 1982
Discontinued	1992
Media	Cassette tape
Operating system	Sinclair BASIC
CPU	Z80 @ 3.5 MHz and equivalent
Memory	16 KB / 48 KB / 128 KB

The **ZX Spectrum** *(Pronounced "Zed Ecks Spec-trum" in its original British English branding)* is an 8-bit personal home computer released in the United Kingdom in 1982 by Sinclair Research Ltd. Referred to during development as the *ZX81 Colour* and *ZX82*, the machine was launched as the *ZX Spectrum* by Sinclair to highlight the machine's colour display, compared with the black-and-white of its predecessor, the Sinclair ZX81. The Spectrum was released in eight different models, ranging from the entry level model with 16 KB RAM released in 1982 to the ZX Spectrum +3 with 128 KB RAM and built in floppy disk drive in 1987, together they sold in excess of 5 million units worldwide

The Spectrum was among the first mainstream audience home computers in the UK, similar in significance to the Commodore 64 in the USA. The introduction of the ZX Spectrum led to a boom in companies producing software and hardware for the machine, the effects of which are still seen; some credit it as the machine which launched the UK IT industry. Licensing deals and clones followed, and earned Clive Sinclair a knighthood for "services to British industry".

The Commodore 64, BBC Microcomputer and later the Amstrad CPC range were major rivals to the Spectrum in the UK market during the early 1980s. The ZX Spectrum has enjoyed a resurgence in popularity thanks to the accessibility of ZX Spectrum emulators, allowing 1980s video game enthusiasts to enjoy classic titles without the long loading times associated with data cassettes. Over 20,000 titles have been released since the Spectrum's launch and new titles continue to be released, with over 60 new ones in 2009.

Hardware



ZX Spectrum 48K motherboard (Issue 3B — 1983, heat sink removed)

The Spectrum is based on a Zilog Z80A CPU running at 3.5 MHz (or NEC D780C-1 clone). The original model Spectrum has 16 KB (16×1024 bytes) of ROM and either 16 KB or 48 KB of RAM. Hardware design was by Richard Altwasser of Sinclair Research, and the machine's outward appearance was designed by Sinclair's industrial designer Rick Dickinson.

Video output is through an RF modulator and was designed for use with contemporary portable television sets, for a simple colour graphic display. Text can be displayed using 32 columns × 24 rows of characters from the ZX Spectrum character set or from a set provided within an application, from a palette of 15 shades: seven colours at two levels of brightness each, plus black. The image resolution is 256×192 with the same colour limitations. To conserve memory, colour is stored separate from the pixel bitmap in a low resolution, 32×24 grid overlay, corresponding to the character cells. Altwasser received a patent for this design.

An "attribute" consists of a foreground and a background colour, a brightness level (normal or bright) and a flashing "flag" which, when set, causes the two colours to swap at regular intervals. Unfortunately, this scheme leads to what was dubbed *colour clash* or *attribute clash* with some bizarre effects in the animated graphics of arcade style games. This problem became a distinctive feature of the Spectrum and an in-joke among Spectrum users, as well as a point of derision by advocates of other systems. Other machines available around the same time, for example the Amstrad CPC, did not suffer from this limitation. The Commodore 64 used colour attributes in a similar way, but a special multicolour mode, hardware sprites and hardware scrolling were used to avoid attribute clash.

Sound output is through a beeper on the machine itself. This is capable of producing one channel with 10 octaves. The machine also includes an expansion bus edge connector and audio in/out ports for the connection of a cassette recorder for loading and saving programs and data.

Firmware

The machine's Sinclair BASIC interpreter is stored in ROM (along with fundamental system-routines) and was written by Steve Vickers on contract from Nine Tiles Ltd. The Spectrum's chiclet keyboard (on top of a membrane, similar to calculator keys) is marked with BASIC keywords, so that, for example, pressing "G" when in programming mode would insert the BASIC command GO TO.

The BASIC was developed from that used on the ZX81 and a ZX81 BASIC program can be typed into a Spectrum largely unmodified, but Spectrum BASIC included many extra features making it easier to use. It also featured a full ASCII character set, missing from the ZX81 which did not feature lower-case letters. Spectrum BASIC included extra keywords for the more advanced display and sound, and also supported multi-statement lines. The cassette interface was also much more advanced, saving and loading around four times faster than the ZX81, and much more reliably. As well as being able to save programs, the Spectrum could in addition save the contents of arrays, the contents of the screen memory, and the contents of any defined range of memory addresses.

Sinclair Research models

Pre-production designs

Rick Dickinson came up with a number of designs for the "ZX82" project before the final ZX Spectrum design. A number of the keyboard legends changed during the design phase including *ARC* becoming *CIRCLE*, *FORE* becoming *INK* and *BACK* becoming *PAPER*.

ZX Spectrum 16K/48K

ZX Spectrum 16K/48K (Dimensions (mm): 233x144x30 (WxHxD) @ ~552 grams).

The original ZX Spectrum is remembered for its rubber keyboard, diminutive size and distinctive rainbow motif. It was originally released in 1982 with 16 KB of RAM for £125 Sterling or with 48 KB for £175; these prices were later reduced to £99 and £129 respectively. Owners of the 16 KB model could purchase an internal 32 KB RAM upgrade, which for early "Issue 1" machines consisted of a daughterboard. Later issue machines required the fitting of 8 dynamic RAM chips and a few TTL chips. Users could mail their 16K Spectrums to Sinclair to be upgraded to 48 KB versions. To reduce the price, the 32 KB extension used eight faulty 64 kilobit chips with only one half of their capacity working and/or available. External 32 KB RAM packs that mounted in the rear expansion slot were also available from third parties. Both machines had 16 KB of onboard ROM.

About 60,000 "Issue 1" ZX Spectrums were manufactured; they can be distinguished from later models by the colour of the keys (light grey for Issue 1, blue-grey for later models).

ZX Spectrum+

ZX Spectrum+ (Dimensions (mm): 319x149x38 (WxHxD))

Planning of the **ZX Spectrum+** started in June 1984, and the machine was released in October the same year. This 48 KB Spectrum (development code-name *TB*) introduced a new QL-style case with an injection-moulded keyboard and a reset button. Electronically, it was identical to the previous 48 KB model. It retailed for £179.95. A DIY conversion-kit for older machines was also available. Early on, the machine outsold the rubber-key model 2:1; however, some retailers reported a failure rate of up to 30%, compared with a more usual 5-6%.

ZX Spectrum 128

ZX Spectrum 128

Sinclair developed the **ZX Spectrum 128** (code-named *Derby*) in conjunction with their Spanish distributor Investrónica. Investrónica had helped adapt the ZX Spectrum+ to the Spanish market after the Spanish government introduced a special tax on all computers with 64 KB RAM or less which did not support the Spanish alphabet (such as ñ) and show messages in Spanish.

New features included 128 KB RAM, three-channel audio via the AY-3-8912 chip, MIDI compatibility, an RS-232 serial port, an RGB monitor port, 32 KB of ROM including an improved BASIC editor, and an external keypad.

The machine was simultaneously presented for the first time and launched in September 1985 at the SIMO '85 trade show in Spain, with a price of 44,250 pesetas. Because of the large number of unsold Spectrum+ models, Sinclair decided not to start selling in the UK until January 1986 at a price of £179.95. No external keypad was available for the UK release, although the ROM routines to use it and the port itself, which was hastily renamed "AUX", remained.

The Z80 processor used in the Spectrum has a 16-bit address bus, which means only 64 KB of memory can be directly addressed. To facilitate the extra 80 KB of RAM the designers used bank switching so that the new memory would be available as eight pages of 16 KB at the top of the address space. The same technique was also used to page between the new 16 KB editor ROM and the original 16 KB BASIC ROM at the bottom of the address space.

The new sound chip and MIDI out abilities were exposed to the BASIC programming language with the command *PLAY* and a new command *SPECTRUM* was added to switch the machine into 48K mode, keeping the current BASIC program intact (although there is no way to switch back to 128K mode). To enable BASIC programmers to access the additional memory, a RAM disk was created where files could be stored in the additional 80 KB of RAM. The new commands took the place of two existing user-defined-character spaces causing compatibility problems with some BASIC programs.

The Spanish version had the "128K" logo in white while the English one had the same logo in red.

Amstrad models

ZX Spectrum +2

ZX Spectrum +2

The **ZX Spectrum +2** was Amstrad's first Spectrum, coming shortly after their purchase of the Spectrum range and "Sinclair" brand in 1986. The machine featured an all-new grey case featuring a spring-loaded keyboard, dual joystick ports, and a built-in cassette recorder dubbed the "Datacorder" (like the Amstrad CPC 464), but was in most respects identical to the ZX Spectrum 128. The main menu screen lacked the Spectrum 128's "Tape Test" option, and the ROM was altered to account for a new 1986 Amstrad copyright message. These changes resulted in minor incompatibility problems with software that accessed ROM routines at certain addresses. Production costs had been reduced and the retail price dropped to £139–£149.

The new keyboard did not include the BASIC keyword markings that were found on earlier Spectrums, except for the keywords *LOAD*, *CODE* and *RUN* which were useful for loading software. This was not a major issue however, as the +2 boasted a menu system, almost identical to the ZX Spectrum 128, where one could switch between 48k

BASIC programming with the keywords, and 128k BASIC programming in which all words (keywords and otherwise) must be typed out in full (although the keywords are still stored internally as one character each). Despite these changes, the layout remained identical to that of the 128.

ZX Spectrum +2A

ZX Spectrum +2A

The **ZX Spectrum +2A** was produced to homogenise Amstrad's range in 1987. Although the case reads "ZX Spectrum +2", the +2A/B is easily distinguishable from the original +2 as the case was restored to the standard Spectrum black.

The +2A was derived from Amstrad's +3 4.1 ROM model, using a new motherboard which vastly reduced the chip count, integrating many of them into a new ASIC. The +2A replaced the +3's disk drive and associated hardware with a tape drive, as in the original +2. Originally, Amstrad planned to introduce an additional disk interface, but this never appeared. If an external disk drive was added, the "+2A" on the system OS menu would change to a +3. As with the ZX Spectrum +3, some older 48K, and a few older 128K, games were incompatible with the machine.

ZX Spectrum +2B

The **ZX Spectrum +2B** signified a manufacturing move from Hong Kong to Taiwan later in 1987.

ZX Spectrum +3

ZX Spectrum +3

The **ZX Spectrum +3** looked similar to the +2 but featured a built-in 3-inch floppy disk drive (like the Amstrad CPC 6128) instead of the tape drive, and was in a black case. It was launched in 1987, initially retailed for £249 and then later £199 and was the only Spectrum capable of running the CP/M operating system without additional hardware.

The +3 saw the addition of two more 16 KB ROMs. One was home to the second part of the reorganised 128 ROM and the other hosted the +3's disk operating system. This was a modified version of Amstrad's AMSDOS, called +3DOS. These two new 16 KB ROMs and the original two 16 KB ROMs were now physically implemented together as two 32 KB chips. To be able to run CP/M, which requires RAM at the bottom of the address space, the bank-switching was further improved, allowing the ROM to be paged out for another 16 KB of RAM.

Such core changes brought incompatibilities:

- Removal of several lines on the expansion bus edge connector (video, power, and IORQGE); caused many external devices problems; some such as the VTX5000 modem could be used via the "FixIt" device
- Dividing ROMCS into 2 lines, to disable both ROMs
- Reading a non-existent I/O port no longer returned the last attribute; caused some games such as *Arkanoid* to be unplayable
- Memory timing changes; some of the RAM banks were now contended causing high-speed colour-changing effects to fail
- The keypad scanning routines from the ROM were removed
- move 1 byte address in ROM

Some older 48K, and a few older 128K, games were incompatible with the machine.

The +3 was the final official model of the Spectrum to be manufactured, remaining in production until December 1990. Although still accounting for one third of all home computer sales in the UK at the time, production of the model was ceased by Amstrad at that point.

Clones

Didaktik M

Sinclair licensed the Spectrum design to Timex Corporation in the United States. An enhanced version of the Spectrum with better sound, graphics and other modifications was marketed in the USA by Timex as the Timex Sinclair 2068. Timex's derivatives were largely incompatible with Sinclair systems. However, some of the Timex innovations were later adopted by Sinclair Research. A case in point was the abortive *Pandora* portable Spectrum, whose ULA had the high resolution video mode pioneered in the TS2068. *Pandora* had a flat-screen monitor and Microdrives and was intended to be Sinclair's business portable. When Alan Sugar bought the computer side of Sinclair it got ditched (a conversation with UK computer journalist Guy Kewney went thus: AS: "Have you seen it?" GK: "Yes" AS: "Well then.").

In the UK, Spectrum peripheral vendor Miles Gordon Technology (MGT) released the SAM Coupé as a potential successor with some Spectrum compatibility. However, by this point, the Commodore Amiga and Atari ST had taken hold of the market, leaving MGT in eventual receivership.

Many unofficial Spectrum clones were produced, especially in the former Eastern Bloc countries (e.g. in Romania, several models were produced (Tim-S, HC85, HC91, Cobra, Junior, CIP, CIP 3, Jet) , some featuring CP/M and a 5.25"/3.5" floppy disk) and South America (e.g. Microdigital TK 90X and TK 95). In the Soviet Union, ZX Spectrum clones were assembled by thousands of small start-ups and distributed though poster ads and street stalls. Over 50 such clone models existed. Some of them are still being produced, such as the *Pentagon* and *ATM Turbo*. In India, Decibells Electronics introduced a licensed version of the Spectrum+ in 1986. Dubbed the "db Spectrum+", it did reasonably well in the Indian market and sold quite a few thousand until 1990 when the market died away.

Peripherals

Several peripherals for the Spectrum were marketed by Sinclair: the ZX Printer was already on the market, as the ZX Spectrum expansion bus was backwards-compatible with that of the ZX81.

The ZX Interface 1 add-on module included 8 KB of ROM, an RS-232 serial port, a proprietary LAN interface (called ZX Net), and an interface for the connection of up to eight ZX Microdrives – somewhat unreliable but speedy tape-loop cartridge storage devices released in July 1983. These were later used in a revised version on the Sinclair QL, whose storage format was electrically compatible but logically incompatible with the Spectrum's. Sinclair also released the ZX Interface 2 which added two joystick ports and a ROM cartridge port.

There were also a plethora of third-party hardware addons. The better known of these included the Kempston joystick interface, the Morex Peripherals Centronics/RS-232 interface, the Currah Microspeech unit (speech synthesis), Videoface Digitiser, RAM pack, the Cheetah Marketing SpecDrum, a drum machine, and the Multiface, a snapshot and disassembly tool from Romantic Robot. Keyboards were especially popular in view of the original's notorious "dead flesh" feel.

ZX Interface 1

ZX Printer

ZX Interface 2

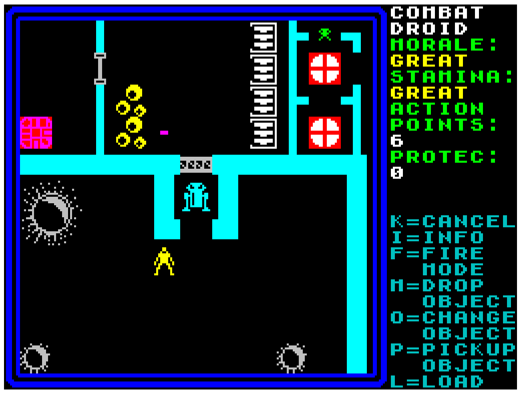
ZX Microdrive

Kempston joystick interface

There were numerous disk drive interfaces, including the Abbeydale Designers/Watford Electronics SPDOS, Abbeydale Designers/Kempston KDOS and Opus Discovery. The SPDOS and KDOS interfaces were the first to come bundled with Office productivity software (*Tasword* Word Processor, *Masterfile* database and *OmniCalc* spreadsheet). This bundle, together with OCP's Stock Control, Finance and Payroll systems, introduced many small businesses to a streamlined, computerised operation. The most popular floppy disk systems (except in East Europe) were the DISCiPLE and +D systems released by Miles Gordon Technology in 1987 and 1988 respectively. Both systems had the ability to store memory images onto disk *snapshots* could later be used to restore the Spectrum to its exact previous state. They were also both compatible with the Microdrive command syntax, which made porting existing software much simpler.

During the mid-1980s, Telemap Group Ltd launched a fee-based service allowing users to connect their ZX Spectrums via a Prism Micro Products VTX5000 modem to a viewdata service known as Micronet 800, hosted by Prestel. This service pre-dated the web, but offered many of the services now considered commonplace.

Software



A screenshot from *Rebelstar*, a well-known Spectrum game

The Spectrum enjoys a vibrant, dedicated fan-base. Since it was cheap and simple to learn to use and program, the Spectrum was the starting point for many programmers. The hardware limitations of the Spectrum imposed a special level of creativity on game designers, and so many Spectrum games are very creative and playable even by today's standards. The early Spectrum models' great success as a games platform came in spite of its lack of built-in joystick ports, primitive sound generation, and colour support that was optimised for text display.

The Spectrum family enjoys a very large software library of more than 20,000 titles which is still increasing. While most of these are games, the library is very diverse, including programming language implementations, databases (e.g. *VU-File*), word processors (e.g. *Tasword II*), spreadsheets (e.g. *VU-Calc*), drawing and painting tools (e.g. *OCP Art Studio*), and even 3D-modelling (e.g. *VU-3D*) and archaeology software amongst many other types.

Distribution

Most Spectrum software was originally distributed on audio cassette tapes. The Spectrum was intended to work with a normal domestic cassette recorder, and despite differences in audio reproduction fidelity, the software loading process was quite reliable, if somewhat slow (by today's standards).

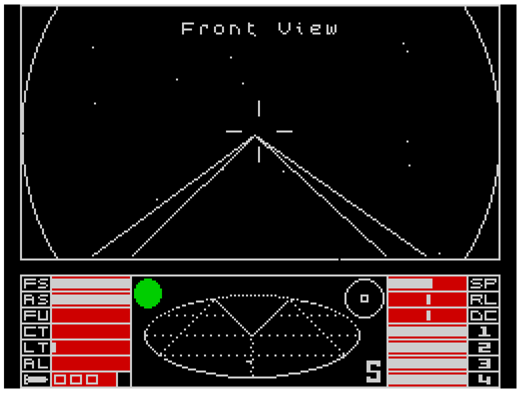
Although the ZX Microdrive was initially greeted with good reviews, it never took off as a distribution method due to worries about the quality of the cartridges and piracy. Hence the main use became to complement tape releases, usually utilities and niche products like the *Tasword* word processing software and *Trans Express*, (a tape to microdrive copying utility). No games are known to be exclusively released on Microdrive.

Despite the popularity of the DISCiPLE and +D systems, most software released for them took the form of utility software. The ZX Spectrum +3 enjoyed much more success when it came to commercial software releases on floppy disk. More than 700 titles were released on 3-inch disk from 1987 to 1997.

Software was also distributed through print media; magazines and books. The reader would type the Sinclair BASIC program listing into the computer by hand, run it, and could save it to tape for later use. The software distributed in this way was in general simpler and slower than its assembly language counterparts. Magazines also printed long lists of checksummed hexadecimal digits with machine code games or tools.

Another software distribution method was to broadcast the audio stream from the cassette on another medium and have users record it onto an audio cassette themselves. In radio or television shows in many European countries, the host would describe a program, instruct the audience to connect a cassette tape recorder to the radio or TV and then broadcast the program over the airwaves in audio format. Some magazines distributed 7" 33⅓ rpm *flexidisc* records, a variant of regular vinyl records which could be played on a standard record player. These disks were known as *floppy ROMs*.

Copying and backup software



A screenshot from *Elite* game

Many copiers—utilities to copy programs from audio tape to another tape, microdrive tapes, and later on diskettes—were available for the Spectrum. As a response to this, publishers introduced copy protection measures to their software, including different loading schemes. Other methods for copy prevention were also used including asking for a particular word from the documentation included with the game—often a novella like in Silicon Dreams trilogy—or another physical device distributed with the software—e.g. Lenslok as used in Elite. Special hardware, such as Romantic Robot's Multiface, was able to dump a copy of the ZX Spectrum RAM to disk/tape at the press of a button, entirely circumventing the copy protection systems.

Most Spectrum software has, in recent years, been converted to current media and is available for download. One popular program for converting Spectrum files from tape is *Taper*; it allows connecting a cassette tape player to the line in port of a sound card, or—through a simple home-built device—to the parallel port of a PC. Once in files on a host machine, the software can be executed on one of many emulators, on virtually any platform available today.

The largest on-line archive of ZX Spectrum software is World of Spectrum, with more than 21,000 titles. The legality of this practice is still in question and while a number of copyright holders have explicitly objected to the posting of their software, others have given their permission for their games to be archived as part of the preservation project.

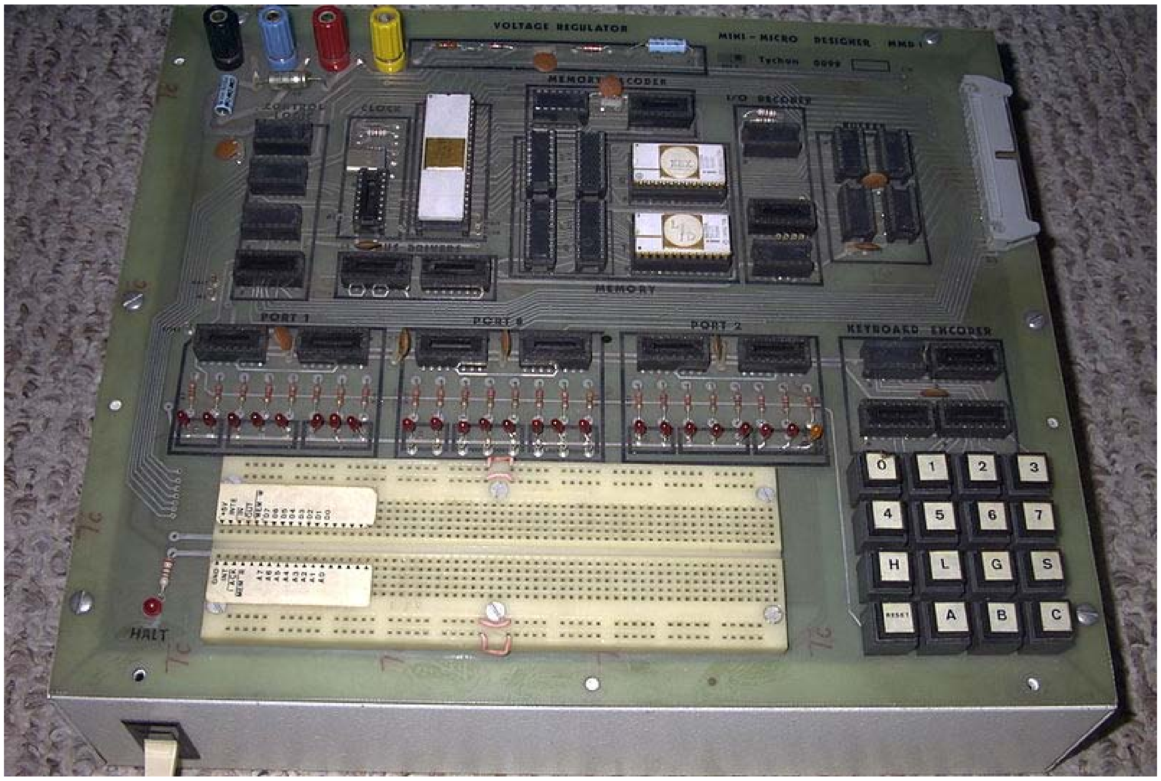
Notable developers

A number of current leading games developers and development companies began their careers on the ZX Spectrum, including David Perry of Shiny Entertainment, and Tim and Chris Stamper (founders of Ultimate Play The Game, now known as Rare, maker of many famous titles for Nintendo and Microsoft game consoles). Other prominent games developers include Julian Gollop (*Chaos*, *Rebelstar*, *X-COM* series), Matthew Smith (*Manic Miner*, *Jet Set Willy*), Jon Ritman (*Match Day*, *Head Over Heels*), The Oliver Twins (the *Dizzy* series), Clive Townsend (*Saboteur*), Pete Cooke (*Tau Ceti*), Mike Singleton (*The Lords of Midnight*,*War In Middle Earth*), and Alan Cox. Although the Spectrum's audio hardware was not as capable as that of the Commodore 64, computer musicians David Whittaker and Tim Follin produced notable multi-channel music for the machine.

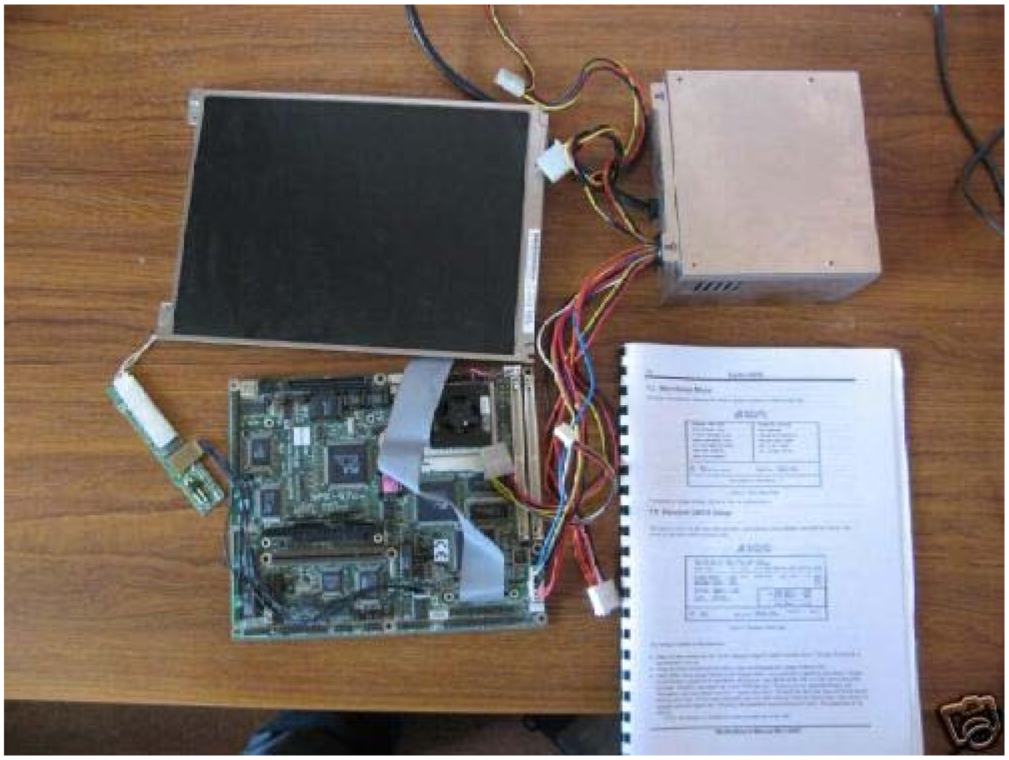
Jeff Minter ported some of his Commodore VIC-20 games for the ZX Spectrum.

Chapter 7

Single-board Computer



One of the first 10 MMD-1s, a prototype unit, produced by E&L Instruments in 1976. The "dyna-micro"/"MMD-1" was the world's first true single board computer. The MMD-1 had all components on a single printed circuit board, including memory, I/O, user input device, and a display. Nothing external to the single board except power was required to both program and run the MMD-1. The original design of the MMD-1 was called the "dyna-micro", but it was soon re-branded as the "MMD-1"



A 486/Pentium SBC with power supply and flatscreen

Close up of SBC

Single-board computers (**SBCs**) are complete computers built on a single circuit board. The design is centered on a single or dual microprocessor with RAM, IO and all other features needed to be a functional computer on the one board. The first true single-board computer called the "dyna-micro" was based on the Intel C8080A, and also used Intel's first EPROM, the C1702A. The dyna-micro was re-branded by E&L Instruments of Derby, CT in 1976 as the "MMD-1" (Mini-Micro Designer 1) and was made famous as the example microcomputer in the very popular 8080 "BugBook" series of the time. SBCs also figured heavily in the early history of home computers, for example in the Acorn Electron and the BBC Micro. Other typical early single board computers were often shipped without enclosure, which had to be added by the owner, examples are the Ferguson Big Board and the Nascom.

With the development of PCs there was a sharp shift away from SBC, with computers being constructed from a motherboard, with functions like serial ports, disk drive controller and graphics being provided on daughterboards. The recent availability of advanced chip sets providing most of the I/O features as embedded components allows motherboard manufacturers to offer motherboards with I/O traditionally provided by daughterboards. Most PC motherboards now offer on-board support for disk drives including IDE and SATA with RAID, graphics, Ethernet, and traditional I/O such as serial and parallel ports, USB, and keyboard/mouse support. Plug-in cards are now more commonly high performance graphics cards (really graphic co-processors), high end RAID controllers, and specialized I/O cards such as data acquisition and DSP (Digital Signal Processor) boards.

Applications

Single Board Computers are now commonly defined across two distinct architectures: no slots and slot support.

Embedded Single Board Computers are boards providing all the required I/O with no provision for plug-in cards. Applications are typically gaming (slot machines, video poker), kiosk, and machine control. Embedded Single Board Computers are much smaller than ATX motherboards, and provide an I/O mix more targeted to an industrial application such as on-board digital and analog I/O, on-board bootable flash so no hard drive is required, no on-board video, etc.

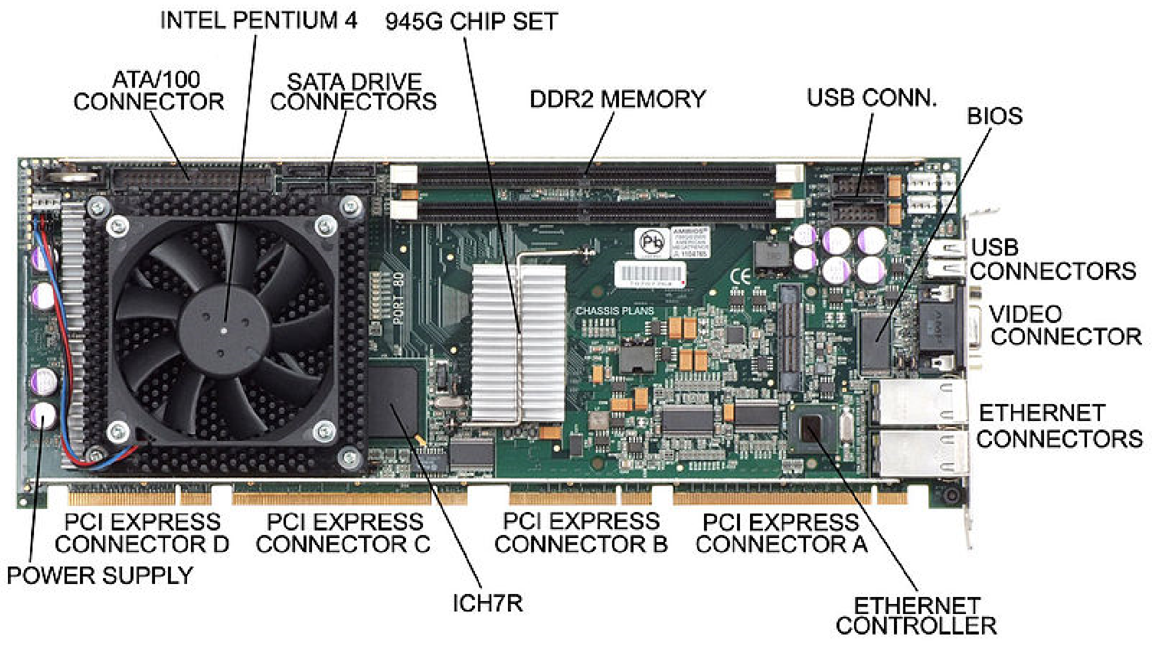
The term "Single Board Computer" now generally applies to an architecture where the Single Board Computer is plugged into a backplane to provide for I/O cards. In the case of PC104, the bus is not a backplane in the traditional sense but is a series of pin connectors allowing I/O boards to be stacked.

Single board computers are most commonly used in industrial situations where they are used in rackmount format for process control or embedded within other devices to provide control and interfacing. Because of the very high levels of integration, reduced component counts and reduced connector counts, SBCs are often smaller, lighter, more power efficient and more reliable than comparable multi-board computers.

The primary advantage of ATX motherboards versus Single Board Computers is cost. Motherboards are manufactured by the millions for the consumer and office markets allowing tremendous economies of scale. Single Board Computers, on the other hand, are in a specialized market niche and are manufactured in much smaller numbers with the resultant higher cost. Motherboards and Single Board Computers now offer similar levels of feature integration meaning that a motherboard failure in either standard will require equivalent replacement.

The primary advantage of a PICMG Single Board Computer is the availability of backplanes offering virtually any slot configuration including legacy ISA support. Motherboards tend to the latest slot technology such that PCI slots are becoming legacy support with PCI-Express becoming the standard. In addition, motherboards offer, at most, 7 slots while backplanes can offer up to 20 slots. In a backplane 12.3" wide, similar in size to an ATX motherboard at 12", a backplane with a Single Board Computer can offer 12 slots for I/O cards with virtually any mix of slot types.

Types, standards



Major Components on an PICMG 1.3 SBC

PICMG 1.3 Single Board Computer (SHB) and Backplane

Currently the most common variety of Single Board Computer in use is of a specific form factor similar to other full-size plug-in cards and is intended to be used in a backplane. Some architectures are dependent entirely on single-board computers, such as CompactPCI, PXI, VMEbus, VXI, PICMG architecture, etc. In the Intel PC world, the intelligence and interface/control circuitry is placed on a plug-in board that is then inserted into a passive (or active) backplane. The end result is similar to having a system built with a motherboard, except that the backplane determines the slot configuration. Backplanes are available with a mix of slots (ISA, PCI, PCIX, PCI-Express, etc), usually totaling 20 or less, meaning it will fit in a 19" rackmount enclosure (17" wide chassis).

Some single-board computers also exist as form factors that stack like building blocks, and do not have the form of a traditional backplane. Examples of stacking SBC form factors include PC/104, PC/104-*Plus*, PCI-104, EPIC, and EBX; these systems are commonly available for use in embedded control systems.

PICMG provides standards for the backplane interface: PICMG 1.0, 1.1 and 1.2 provide for ISA and PCI support with 1.2 adding PCIX support PICMG 1.3 provides for PCI-Express support. Single Board Computers meeting the PICMG 1.3 specification are referred to as a System Host Board.

Stack-type SBCs often have memory provided on plug-cards such as SIMMs and DIMMs, however they can still be regarded as SBCs because although the memory modules are technically additional circuit boards, they have no extra functionality beyond providing memory and are basically just carriers for the RAM chips. Hard drive circuit boards are also not counted for determining if a computer is an SBC or not for two reasons, firstly because the HDD is regarded as a single block storage unit, and secondly because the SBC may not require a hard drive at all as most can be booted from their network connections.

Chapter 8

AIM-65

AIM-65

Comelta Drac-1

Comelta Drac-1 and expansion box

back of expansion box

The Rockwell **AIM-65** computer was a development computer based on the MOS Technology 6502 microprocessor introduced in 1976. The AIM-65 was essentially an expanded KIM-1 computer. Available software included a monitor with line at a time assembler/disassembler, BASIC interpreter, assembler, Pascal, PL/65, and FORTH development system. Available hardware included a floppy disk controller and a backplane for expansion.

Features

Standard software included the system console monitor software in ROM, called Advanced Interactive Monitor. It featured line assembler, disassembler, setting and viewing memory and registers, starting execution of other programs and more. Single stepping was made possible using non-maskable interrupt (NMI). The command prompt was the less-than sign "<", and on receiving a single character command, it added this input character and the greater-than sign ">". If the thermal printer was turned on, this would be output on a single line. The monitor included a number of service routines that could be accessed and used by a user's program to control I/O and code execution, and was fully documented, including source code.

The machine featured dual cassette tape control. This made it possible to write large assembly programs using the two pass assembler ROM. Source code in text was written twice consecutively to the input tape, and then the assembler, which could start/stop the input cassette tape using motor control was invoked. During the first pass the symbol table was built and stored in RAM. During the second pass symbols would be translated and code written out to the second tape, also using start/stop motor control. Being able to avoid storing code in RAM made it possible to save much space. It was however, still important to keep the symbols list short since RAM size was often no more than 4 KB.

In 1981 Rockwell introduced an improved model with a 40 character display as the **AIM-65/40**. An industrial chassis version was known as the System 65 and included a PROM burner and floppy drives. Rockwell was also a pioneer in solid-state storage devices, introducing "bubble memory" non-volatile expansion boards about 1980.

MTU made a "Visible Memory" card in 1978 that worked with the KIM-1 and AIM-65 computers, providing raster graphics display capability. MTU also made the first real time music synthesizer for a microcomputer; it worked with the KIM-1 and AIM-65, and featured a DAC with software providing 4 voices of wavetable synthesis.

In Spain they were distributed by Comelta. This company made various card expansions:

- CR-106 8 Kbytes of RAM
- CR-119 RAM / ROM / PROM expansion
- CR-120 Universal programming
- CR-115 Microcassette controller(two units)
- CR-113 Video controller
- CR-401 Board Bus Extension(Standard S-64)

Comelta assembled all the options in a single box to produce a new computer, the Comelta Drac-1. The first prototype used microcassetes, but definitive versions have two 8" floppy disk drives.

In the late '70s, the Rockwell AIM-65, and successor System 65 became the first computers used onboard a float in the Tournament of Roses Parade. Cal Poly Universities wrote their own animation control language to control hydraulic and motor actuators on floats for many years. In 2003, some of these 27 year old computers were still in use controlling various displays and creatures at a high tech Halloween show near Alexandria, Virginia, U.S.A.

Technical specifications

- Built in full sized QWERTY keyboard
- 20 character alphanumeric LED display (16 segments)
- Integrated 20 character thermal printer
- RS-232 serial interface
- Expansion connector

- Application connector with 6522 VIA chip
- 4 KB RAM
- 5 sockets for 4 KB ROM/EPROM chips

Programming

PL/65 was a programming language designed and implemented by Rockwell International for the AIM-65. It is based on a mix of ALGOL and PL/I, simplified where possible in order to adapt to the limited processing environment afforded by the 6502 (64k memory for instance).

Chapter 9

Introduction to Supercomputer

The Columbia Supercomputer, located at the NASA Ames Research Center

A 1985 supercomputer Cray-2

A **supercomputer** is a computer that is at the frontline of current processing capacity, particularly speed of calculation. Supercomputers were introduced in the 1960s and were designed primarily by Seymour Cray at Control Data Corporation (CDC), which led the market into the 1970s until Cray left to form his own company, Cray Research. He then took over the supercomputer market with his new designs, holding the top spot in supercomputing for five years (1985–1990). In the 1980s a large number of smaller competitors entered the market, in parallel to the creation of the minicomputer market a decade earlier, but many of these disappeared in the mid-1990s "supercomputer market crash".

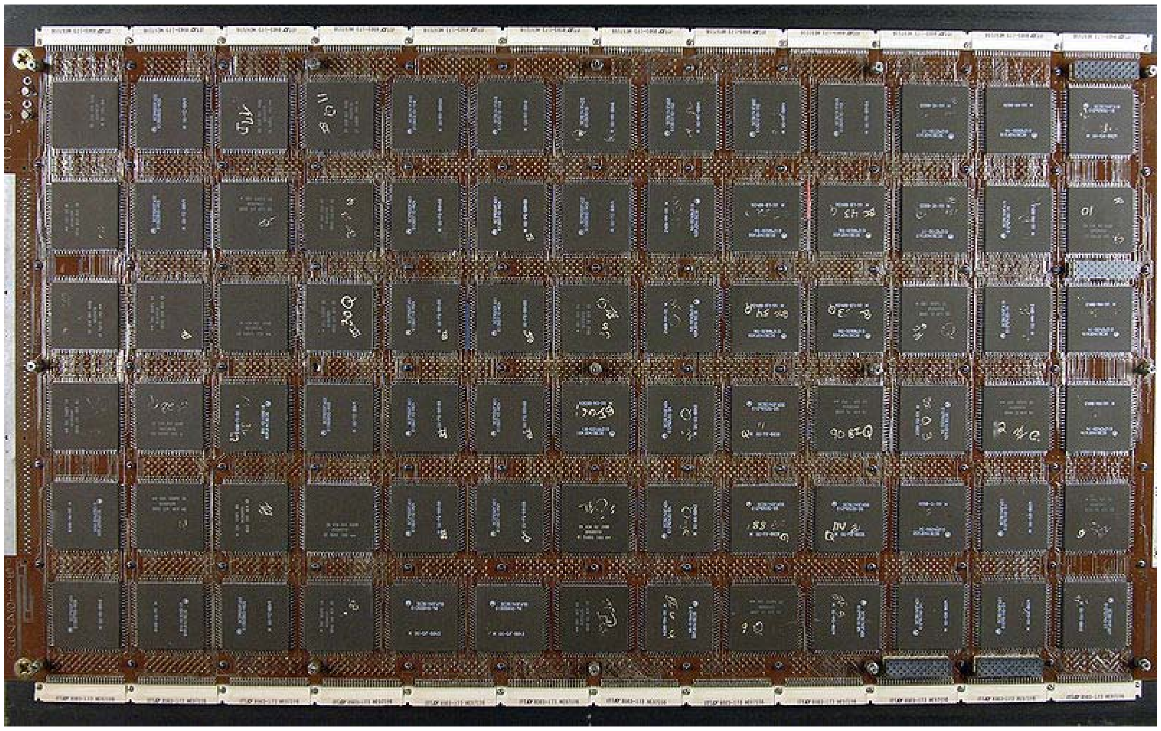
Today, supercomputers are typically one-of-a-kind custom designs produced by "traditional" companies such as Cray, IBM and Hewlett-Packard, who had purchased many of the 1980s companies to gain their experience. Since October 2010, the Tianhe-1A supercomputer has been the fastest in the world; it is located in China.

The term *supercomputer* itself is rather fluid, and today's supercomputer tends to become tomorrow's ordinary computer. CDC's early machines were simply very fast scalar processors, some ten times the speed of the fastest machines offered by other companies. In the 1970s most supercomputers were dedicated to running a vector processor, and many of the newer players developed their own such processors at a lower price to enter the market. The early and mid-1980s saw machines with a modest number of vector processors working in parallel to become the standard. Typical numbers of processors were in the range of four to sixteen. In the later 1980s and 1990s, attention turned from vector processors to massive parallel processing systems with thousands of "ordinary" CPUs, some being off the shelf units and others being custom designs. Today, parallel designs are based on "off the shelf" server-class microprocessors, such as the PowerPC, Opteron, or Xeon, and coprocessors like NVIDIA Tesla GPGPUs, AMD GPUs, IBM Cell, FPGAs. Most modern supercomputers are now highly-tuned computer clusters using commodity processors combined with custom interconnects.

Supercomputers are used for highly calculation-intensive tasks such as problems involving quantum physics, weather forecasting, climate research, molecular modeling (computing the structures and properties of chemical compounds, biological macromolecules, polymers, and crystals), physical simulations (such as simulation of airplanes in wind tunnels, simulation of the detonation of nuclear weapons, and research into nuclear fusion).

Relevant here is the distinction between capability computing and capacity computing, as defined by Graham et al. **Capability computing** is typically thought of as using the maximum computing power to solve a large problem in the shortest amount of time. Often a capability system is able to solve a problem of a size or complexity that no other computer can. **Capacity computing** in contrast is typically thought of as using efficient cost-effective computing power to solve somewhat large problems or many small problems or to prepare for a run on a capability system.

Hardware and software design



Processor board of a CRAY YMP vector computer

Supercomputers using custom CPUs traditionally gained their speed over conventional computers through the use of innovative designs that allow them to perform many tasks in parallel, as well as complex detail engineering. They tend to be specialized for certain types of computation, usually numerical calculations, and perform poorly at more general computing tasks. Their memory hierarchy is very carefully designed to ensure the processor is kept fed with data and instructions at all times — in fact, much of the performance difference between slower computers and supercomputers is due to the memory hierarchy. Their I/O systems tend to be designed to support high bandwidth, with latency less of an issue, because supercomputers are not used for transaction processing.

As with all highly parallel systems, Amdahl's law applies, and supercomputer designs devote great effort to eliminating software serialization, and using hardware to address the remaining bottlenecks.

Supercomputer challenges, technologies

- A supercomputer generates large amounts of heat and must be cooled. A typical TOP500 supercomputer consumes between 1 and 10 megawatt of electricity and converts all of it into heat. The cost to power and cool the system is usually one of the factors that limit the scalability of the system (for example, a high-end system such as Tianhe-1A could consume several million dollars worth of electricity per year).
- Information cannot move faster than the speed of light between two parts of a supercomputer. For this reason, a supercomputer that is many meters across must have latencies between its components measured at least in the tens of nanoseconds. Seymour Cray's supercomputer designs attempted to keep cable runs as short as possible for this reason, hence the cylindrical shape of his Cray range of computers. In modern supercomputers built of many conventional CPUs running in parallel, latencies of 1–5 microseconds to send a message between CPUs are typical.
- Supercomputers consume and produce massive amounts of data in a very short period of time. According to Ken Batcher, "A supercomputer is a device for turning compute-bound problems into I/O-bound problems." Much work on external storage bandwidth is needed to ensure that this information can be transferred quickly and stored/retrieved correctly.

Processing techniques

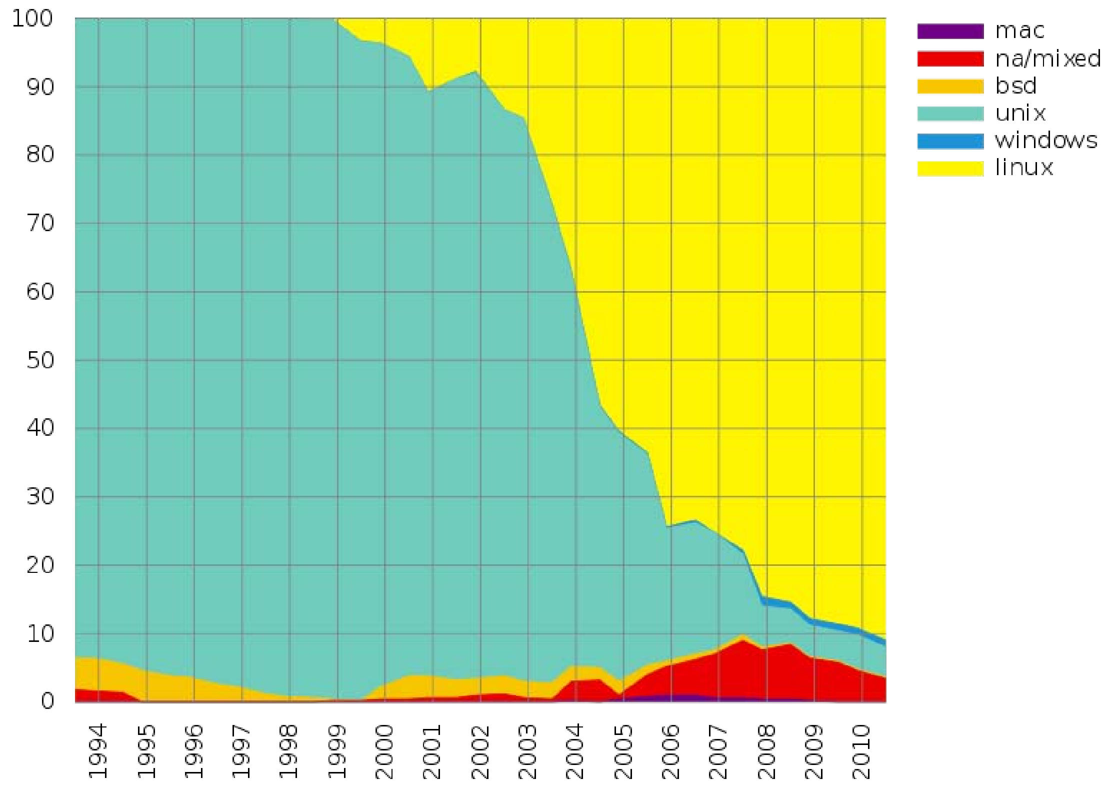
Vector processing techniques were first developed for supercomputers and continue to be used in specialist high-performance applications. Vector processing techniques have trickled down to the mass market in DSP architectures and SIMD (**S**ingle **I**nstruction **M**ultiple **D**ata) processing instructions for general-purpose computers.

Modern video game consoles in particular use SIMD extensively and this is the basis for some manufacturers' claim that their game machines are themselves supercomputers.

Indeed, some graphics cards have the computing power of several TeraFLOPS. The applications to which this power can be applied was limited by the special-purpose nature of early video processing. As video processing has become more sophisticated, graphics processing units (GPUs) have evolved to become more useful as general-purpose vector processors, and an entire computer science sub-discipline has arisen to exploit this capability: General-Purpose Computing on Graphics Processing Units (GPGPU).

The current Top500 list (from May 2010) has 3 supercomputers based on GPGPUs. In particular, the number 2 supercomputer, Nebulae built by Dawning in China, is based on GPGPUs.

Operating systems



Supercomputers predominantly run a variant of Linux,

Supercomputers today most often use variants of Linux. as shown by the graph to the right.

Until the early-to-mid-1980s, supercomputers usually sacrificed instruction set compatibility and code portability for performance (processing and memory access speed). For the most part, supercomputers to this time (unlike high-end mainframes) had vastly different operating systems. The Cray-1 alone had at least six different proprietary OSs largely unknown to the general computing community. In similar manner, different and incompatible vectorizing and parallelizing compilers for Fortran existed. This trend would have continued with the ETA-10 were it not for the initial instruction set compatibility between the Cray-1 and the Cray X-MP, and the adoption of computer systems such as Cray's Unicos, or Linux.

Programming

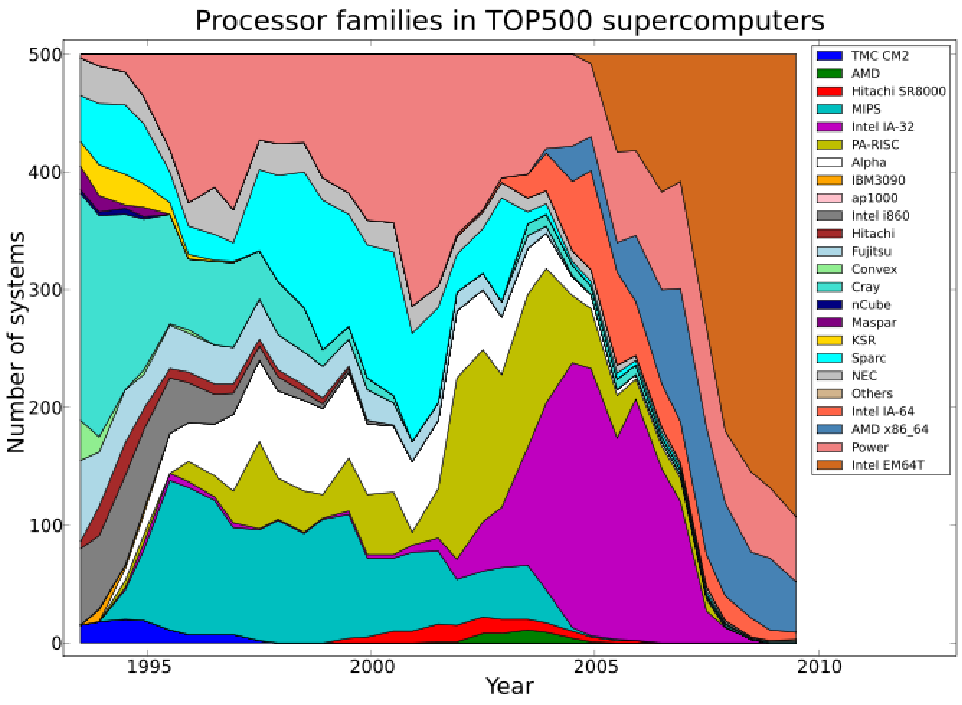
The parallel architectures of supercomputers often dictate the use of special programming techniques to exploit their speed. The base language of supercomputer code is, in general, Fortran or C, using special libraries to share data between nodes. In the most common scenario, environments such as PVM and MPI for loosely connected clusters and OpenMP for tightly coordinated shared memory machines are used. Significant effort is required to optimize a problem for the interconnect characteristics of the machine it will be run on; the aim is to prevent any of the CPUs from wasting time waiting on data from other nodes. The new massively parallel GPGPUs have hundreds of processor cores and are programmed using programming models such as CUDA and OpenCL.

Software tools

Software tools for distributed processing include standard APIs such as MPI and PVM, VTL, and open source-based software solutions such as Beowulf, WareWulf, and openMosix, which facilitate the creation of a supercomputer from a collection of ordinary workstations or servers. Technology like ZeroConf (Rendezvous/Bonjour) can be used to create ad hoc computer clusters for specialized software such as Apple's Shake compositing application. An easy programming language for supercomputers remains an open research topic in computer science. Several utilities that would once have cost several thousands of dollars are now completely free thanks to the open source community that often creates disruptive technology.

Modern supercomputer architecture

IBM Roadrunner – LANL



The CPU Architecture Share of Top500 Rankings between 1993 and 2009

Supercomputers today often have a similar top-level architecture consisting of a cluster of MIMD multiprocessors, each processor of which is SIMD. The supercomputers vary radically with respect to the number of multiprocessors per cluster, the number of processors per multiprocessor, and the number of simultaneous instructions per SIMD processor. Within this hierarchy we have:

- A computer cluster is a collection of computers that are highly interconnected via a high-speed network or switching fabric. Each computer runs under a separate instance of an Operating System (OS).
- A multiprocessing computer is a computer, operating under a single OS and using more than one CPU, wherein the application-level software is indifferent to the number of processors. The processors share tasks using Symmetric multiprocessing (SMP) and Non-Uniform Memory Access (NUMA).
- A SIMD processor executes the same instruction on more than one set of data at the same time. The processor could be a general purpose commodity processor or special-purpose vector processor. It could also be high-performance processor or a low power processor. As of 2007, the processor executes several SIMD instructions per nanosecond.

As of October 2010 the fastest supercomputer in the world is the Tianhe-1A system at National University of Defense Technology with more than 21000 computers, it boasts a speed of 2.507 petaflops, over 30% faster than the world's next fastest computer, the Cray XT5 "Jaguar".

The third fastest supercomputer and the fastest heterogeneous (or hybrid) machine is Dawning Nebulae in China. This machine is a cluster of 4640 blade servers, each with 1 NVIDIA Tesla C2050 (Fermi) GPGPU and 2 Intel Westmere CPUs. The Tesla GPUs deliver most of the Linpack performance, since each Tesla C2050 GPU has 515 Gigaflops peak double precision performance. The most remarkable thing about the hybrid supercomputers like Nebulae and the IBM Roadrunner (uses IBM Cell as coprocessor) is the low power of these systems. Nebulae for example is 2.55 Megawatts power and delivers 1.271 Petaflops/s compared to the number 1 supercomputer Jaguar (made using AMD Opteron CPUs) that consumes 7 Megawatt power and delivers 1.759 Petaflops/s. This makes Nebulae two times higher performance per watt compared to Jaguar.

In February 2009, IBM also announced work on "Sequoia," which appears to be a 20 petaflops supercomputer. This will be equivalent to 2 million laptops (whereas Roadrunner is comparable to a mere 100,000 laptops). It is slated for deployment in late 2011. The Sequoia will be powered by 1.6 million cores (specific 45-nanometer chips in development) and 1.6 petabytes of memory. It will be housed in 96 refrigerators spanning roughly 3,000 square feet (280 m^2).

Moore's Law and economies of scale are the dominant factors in supercomputer design. The design concepts that allowed past supercomputers to out-perform desktop machines of the time tended to be gradually incorporated into commodity PCs. Furthermore, the costs of chip development and production make it uneconomical to design custom chips for a small run and favor mass-produced chips that have enough demand to recoup the cost of production. A current model quad-core Xeon workstation running at 2.66 GHz will outperform a multimillion dollar Cray C90 supercomputer used in the early 1990s; most workloads requiring such a supercomputer in the 1990s can be done on workstations costing less than 4,000 US dollars as of 2010. Supercomputing is taking a step of increasing density, allowing for desktop supercomputers to become available, offering the computer power that in 1998 required a large room to require less than a desktop footprint.

In addition, many problems carried out by supercomputers are particularly suitable for parallelization (in essence, splitting up into smaller parts to be worked on simultaneously) and, in particular, fairly coarse-grained parallelization that limits the amount of information that needs to be transferred between independent processing units. For this reason, traditional supercomputers can be replaced, for many applications, by "clusters" of computers of standard design, which can be programmed to act as one large computer.

The fastest supercomputers today

Measuring supercomputer speed

In general, the speed of a supercomputer is measured in "FLOPS" (***FL****oating Point* ***O****perations* ***P****er* ***S****econd*), commonly used with an SI prefix such as tera-, combined into the shorthand "TFLOPS" (10^{12} FLOPS, pronounced *teraflops*), or peta-, combined into the shorthand "PFLOPS" (10^{15} FLOPS, pronounced *petaflops*.) This measurement is based on a particular benchmark, which does LU decomposition of a large matrix. This mimics a class of real-world problems, but is significantly easier to compute than a majority of actual real-world problems.

"Petascale" supercomputers can process one quadrillion (10^{15}) (1000 trillion) FLOPS. Exascale is computing performance in the exaflops range. An exaflop is one quintillion (10^{18}) FLOPS (one million teraflops).

The TOP500 list

Since 1993, the fastest supercomputers have been ranked on the TOP500 list according to their LINPACK benchmark results. The list does not claim to be unbiased or definitive, but it is a widely cited current definition of the "fastest" supercomputer available at any given time.

Current fastest supercomputer system

Jack Dongarra has stated that the Tianhe-1A supercomputer in China at the National Supercomputing Center in Tianjin is 1.4 times as fast as the AMD Opteron-based Cray XT5 Jaguar at the Oak Ridge National Laboratory. According to Nvidia, Tianhe-1A has achieved a processing rate of 2.507 petaflops on the LINPACK benchmark. Tianhe-1A consists of 14,336 Intel Xeon CPUs and 7,168 Nvidia Tesla M2050 GPUs with a new interconnect fabric of Chinese origin, reportedly twice the speed of InfiniBand. Tianhe-1A spans 103 cabinets, weighs 155 tons, and consumes 4.04 megawatts of electricity. The dethroned Cray XT5 Jaguar has a sustained processing rate of 1.759 PFLOPS.

Quasi-supercomputing

A Blue Gene/P node card

Some types of large-scale distributed computing for embarrassingly parallel problems take the clustered supercomputing concept to an extreme.

The fastest cluster, Folding@home, reported over 7.8 petaflops of processing power as of December 2009. Of this, 2.3 petaflops of this processing power is contributed by clients running on NVIDIA GeForce GPUs, AMD GPUs, PlayStation 3 systems and another 5.1 petaflops is contributed by their newly released GPU2 client.

Another distributed computing project is the BOINC platform, which hosts a number of distributed computing projects. As of April 2010, BOINC recorded a processing power of over 5 petaflops through over 580,000 active computers on the network. The most active project (measured by computational power), MilkyWay@home, reports processing power of over 1.4 petaflops through over 30,000 active computers.

As of April 2010, GIMPS's distributed Mersenne Prime search currently achieves about 45 teraflops.

Also a “quasi-supercomputer” is Google's search engine system with estimated total processing power of between 126 and 316 teraflops, as of April 2004. In June 2006 the *New York Times* estimated that the Googleplex and its server farms contain 450,000 servers. According to recent estimations, the processing power of Google's cluster might reach from 20 to 100 petaflops.

The PlayStation 3 Gravity Grid uses a network of 16 machines, and exploits the Cell processor for the intended application, which is performing astrophysical simulations of large supermassive black holes capturing smaller compact objects. The Cell processor has a main CPU and 6 floating-point vector processors, giving the machine a net of 16 general-purpose machines and 96 vector processors. This cluster was built in 2007 by Dr. Gaurav Khanna, a professor in the Physics Department of the University of Massachusetts Dartmouth with support from Sony Computer Entertainment and is the first PS3 cluster that generated numerical results that were published the scientific research literature.

Other notable computer clusters are the flash mob cluster and the Beowulf cluster. The flash mob cluster allows the use of any computer in the network, while the Beowulf cluster still requires uniform architecture.

Research and development

IBM is developing the Cyclops64 architecture, intended to create a "supercomputer on a chip".

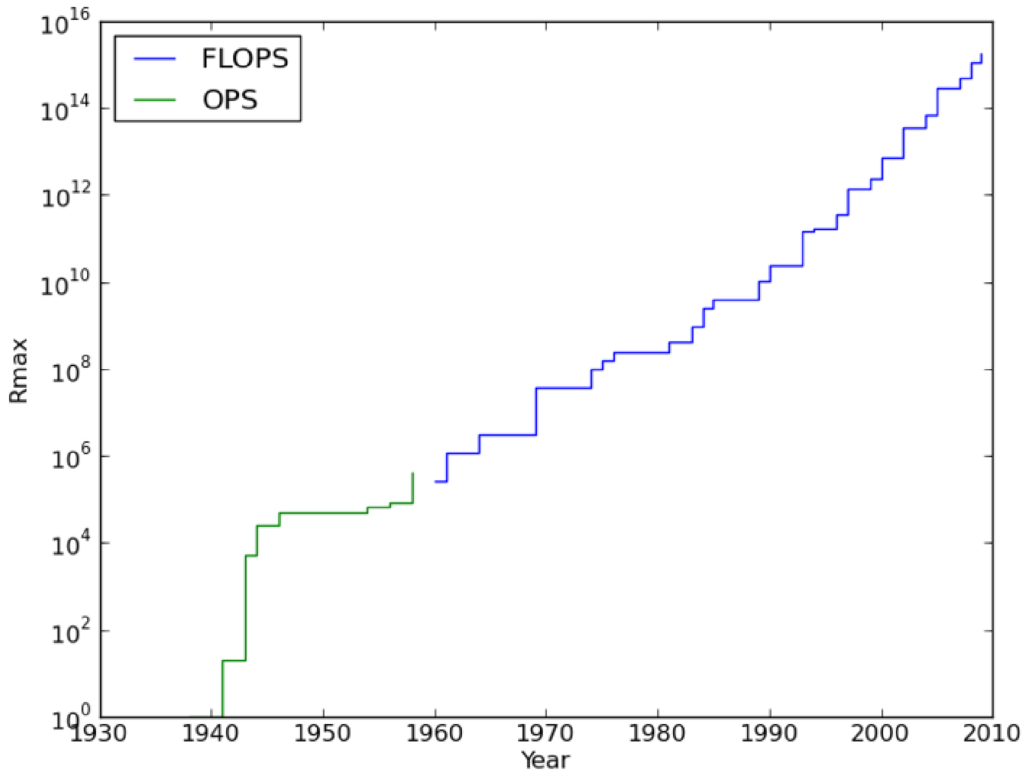
Other PFLOPS projects include one by Narendra Karmarkar in India, a C-DAC effort targeted for 2010, and the Blue Waters Petascale Computing System funded by the NSF ($200 million) that is being built by the NCSA at the University of Illinois at Urbana-Champaign (slated to be completed by 2011).

In May 2008 a collaboration was announced between NASA, SGI and Intel to build a 1 petaflops computer, Pleiades, in 2009, scaling up to 10 PFLOPs by 2012. Meanwhile, IBM is constructing a 20 PFLOPs supercomputer at Lawrence Livermore National Laboratory, named Sequoia, which is scheduled to go online in 2011.

Given the current speed of progress, supercomputers are projected to reach 1 exaflops (10^{18}) (one quintillion FLOPS) in 2019.

Erik P. DeBenedictis of Sandia National Laboratories theorizes that a zettaflops (10^{21}) (one sextillion FLOPS) computer is required to accomplish full weather modeling, which could cover a two week time span accurately. Such systems might be built around 2030.

Timeline of supercomputers



Fastest supercomputers: log speed vs. time

This is a list of the record-holders for fastest general-purpose supercomputer in the world, and the year each one set the record. For entries prior to 1993, this list refers to various sources. From 1993 to present, the list reflects the Top500 listing, and the "Peak speed" is given as the "Rmax" rating.

Chapter 10

Vector Processor

A **vector processor**, or **array processor**, is a central processing unit (CPU) that implements an instruction set containing instructions that operate on one-dimensional arrays of data called *vectors*. This is in contrast to a scalar processor, whose instructions operate on single data items. The vast majority of CPUs are scalar.

Vector processors first appeared in the 1970s, and formed the basis of most supercomputers through the 1980s and into the 1990s. Improvements in scalar processors, particularly microprocessors, resulted in the decline of traditional vector processors in supercomputers, and the appearance of vector processing techniques in mass market CPUs around the early 1990s. Today, most commodity CPUs implement architectures that feature instructions for some vector processing on multiple (vectorized) data sets, typically known as SIMD (**S**ingle **I**nstruction, **M**ultiple **D**ata). Common examples include MMX, SSE, and AltiVec. Vector processing techniques are also found in video game console hardware and graphics accelerators. In 2000, IBM, Toshiba and Sony collaborated to create the Cell processor, consisting of one scalar processor and eight vector processors, which found use in the Sony PlayStation 3 among other applications.

Other CPU designs may include some multiple instructions for vector processing on multiple (vectorised) data sets, typically known as MIMD (**M**ultiple **I**nstruction, **M**ultiple **D**ata). Such designs are usually dedicated to a particular application and not commonly marketed for general purpose computing.

History

Vector processing was first worked on in the early 1960s at Westinghouse in their **Solomon** project. Solomon's goal was to dramatically increase math performance by using a large number of simple math co-processors under the control of a single master CPU. The CPU fed a single common instruction to all of the arithmetic logic units (ALUs), one per "cycle", but with a different data point for each one to work on. This allowed the Solomon machine to apply a single algorithm to a large data set, fed in the form of an array. In 1962, Westinghouse cancelled the project, but the effort was re-started at the University of Illinois as the ILLIAC IV. Their version of the design originally called for a 1 GFLOPS machine with 256 ALUs, but, when it was finally delivered in 1972, it had only 64 ALUs and could reach only 100 to 150 MFLOPS.

Nevertheless it showed that the basic concept was sound, and, when used on data-intensive applications, such as computational fluid dynamics, the "failed" ILLIAC was the fastest machine in the world. The ILLIAC approach of using separate ALUs for each data element is not common to later designs, and is often referred to under a separate category, massively parallel computing.

The first *successful* implementation of vector processing appears to be the Control Data Corporation STAR-100 and the Texas Instruments Advanced Scientific Computer (ASC). The basic ASC (i.e., "one pipe") ALU used a pipeline architecture that supported both scalar and vector computations, with peak performance reaching approximately 20 MFLOPS, readily achieved when processing long vectors. Expanded ALU configurations supported "two pipes" or "four pipes" with a corresponding 2X or 4X performance gain. Memory bandwidth was sufficient to support these expanded modes. The STAR was otherwise slower than CDC's own supercomputers like the CDC 7600, but at data related tasks they could keep up while being much smaller and less expensive. However the machine also took considerable time decoding the vector instructions and getting ready to run the process, so it required very specific data sets to work on before it actually sped anything up.

The vector technique was first fully exploited in the famous Cray-1. Instead of leaving the data in memory like the STAR and ASC, the Cray design had eight "vector registers," which held sixty-four 64-bit words each. The vector instructions were applied between registers, which is much faster than talking to main memory. The Cray design used pipeline parallelism to implement vector instructions rather than multiple ALUs. In addition the design had completely separate pipelines for different instructions, for example, addition/subtraction was implemented in different hardware than multiplication. This allowed a batch of vector instructions themselves to be pipelined, a technique they called *vector chaining*. The Cray-1 normally had a performance of about 80 MFLOPS, but with up to three chains running it could peak at 240 MFLOPS – a respectable number even as of 2002.

Other examples followed. Control Data Corporation tried to re-enter the high-end market again with its ETA-10 machine, but it sold poorly and they took that as an opportunity to leave the supercomputing field entirely. In the early and mid-1980s Japanese companies (Fujitsu, Hitachi and Nippon Electric Corporation (NEC) introduced register-based vector machines similar to the Cray-1, typically being slightly faster and much smaller. Oregon-based Floating Point Systems (FPS) built add-on array processors for minicomputers, later building their own minisupercomputers. However Cray continued to be the performance leader, continually beating the competition with a series of machines that led to the Cray-2, Cray X-MP and Cray Y-MP. Since then, the supercomputer market has focused much more on massively parallel processing rather than better implementations of vector processors. However, recognising the benefits of vector processing IBM developed Virtual Vector Architecture for use in supercomputers coupling several scalar processors to act as a vector processor.

Vector processing techniques have since been added to almost all modern CPU designs, although they are typically referred to as SIMD. In these implementations, the vector unit runs beside the main scalar CPU, and is fed data from programs that know it is there.

Description

In general terms, CPUs are able to manipulate one or two pieces of data at a time. For instance, many CPUs have an instruction that essentially says "add A to B and put the result in C". The data for A, B and C could be—in theory at least—encoded directly into the instruction. However things are rarely that simple. In general the data is rarely sent in raw form, and is instead "pointed to" by passing in an address to a memory location that holds the data. Decoding this address and getting the data out of the memory takes some time. As CPU speeds have increased, this *memory latency* has historically become a large impediment to performance.

In order to reduce the amount of time this takes, most modern CPUs use a technique known as instruction pipelining in which the instructions pass through several sub-units in turn. The first sub-unit reads the address and decodes it, the next "fetches" the values at those addresses, and the next does the math itself. With pipelining the "trick" is to start decoding the next instruction even before the first has left the CPU, in the fashion of an assembly line, so the address decoder is constantly in use. Any particular instruction takes the same amount of time to complete, a time known as the *latency*, but the CPU can process an entire batch of operations much faster than if it did so one at a time.

Vector processors take this concept one step further. Instead of pipelining just the instructions, they also pipeline the data itself. They are fed instructions that say not just to add A to B, but to add all of the numbers "from here to here" to all of the numbers "from there to there". Instead of constantly having to decode instructions and then fetch the data needed to complete them, it reads a single instruction from memory, and "knows" that the next address will be one larger than the last. This allows for significant savings in decoding time.

To illustrate what a difference this can make, consider the simple task of adding two groups of 10 numbers together. In a normal programming language you would write a "loop" that picked up each of the pairs of numbers in turn, and then added them. To the CPU, this would look something like this:

```
execute this loop 10 times
  read the next instruction and decode it
  fetch this number
  fetch that number
  add them
  put the result here
end loop
```

But to a vector processor, this task looks considerably different:

```
read instruction and decode it
```

```
fetch these 10 numbers
fetch those 10 numbers
add them
put the results here
```

There are several savings inherent in this approach. For one, only two address translations are needed. Depending on the architecture, this can represent a significant savings by itself. Another saving is fetching and decoding the instruction itself, which has to be done only one time instead of ten. The code itself is also smaller, which can lead to more efficient memory use.

But more than that, a vector processor may have multiple functional units adding those numbers in parallel. The checking of dependencies between those numbers is not required as a vector instruction specifies multiple independent operations. This simplifies the control logic required, and can improve performance by avoiding stalls.

As mentioned earlier, the Cray implementations took this a step further, allowing several different types of operations to be carried out at the same time. Consider code that adds two numbers and then multiplies by a third; in the Cray, these would all be fetched at once, and both added and multiplied in a single operation. Using the pseudocode above, the Cray did:

```
read instruction and decode it
fetch these 10 numbers
fetch those 10 numbers
fetch another 10 numbers
add and multiply them
put the results here
```

The math operations thus completed far faster overall, the limiting factor being the time required to fetch the data from memory.

Not all problems can be attacked with this sort of solution. Adding these sorts of instructions necessarily adds complexity to the core CPU. That complexity typically makes *other* instructions run slower—i.e., whenever it is **not** adding up many numbers in a row. The more complex instructions also add to the complexity of the decoders, which might slow down the decoding of the more common instructions such as normal adding.

In fact, vector processors work best only when there are large amounts of data to be worked on. For this reason, these sorts of CPUs were found primarily in supercomputers, as the supercomputers themselves were, in general, found in places such as weather prediction centres and physics labs, where huge amounts of data are "crunched".

Real world example: vector instructions usage with the x86 architecture

Shown below is a actual x86 architecture example for vector instruction usage with the SSE instruction set. The example multiplies two arrays of single precision floating point numbers. It's written in the C language with inline assembly code parts for compilation with GCC (32bit).

```
//SSE simd function for vectorized multiplication of 2
arrays with single-precision floatingpoint numbers
//1st param pointer on source/destination array, 2nd param
2. source array, 3th param number of floats per array
 void mul_asm(float* out, float* in, unsigned int leng)
 {    unsigned int count, rest;

      //compute if array is big enough for vector operation
      rest  = (leng*4)%16;
      count = (leng*4)-rest;

     // vectorized part; 4 floats per loop iteration
      if (count>0){
      __asm __volatile__  (".intel_syntax noprefix\n\t"
      "loop:                 \n\t"
      "movups xmm0,[ebx+ecx] ;loads 4 floats in first
register (xmm0)\n\t"
      "movups xmm1,[eax+ecx] ;loads 4 floats in second
register (xmm1)\n\t"
      "mulps xmm0,xmm1       ;multiplies both vector
registers\n\t"
      "movups [eax+ecx],xmm0 ;write back the result to
memory\n\t"
      "sub ecx,16            ;increase address pointer by 4
floats\n\t"
      "jnz loop              \n\t"
      ".att_syntax prefix    \n\t"
        : : "a" (out), "b" (in), "c"(count), "d"(rest):
"xmm0","xmm1");
      }

      // scalar part; 1 float per loop iteration
      if (rest!=0)
      {
       __asm __volatile__  (".intel_syntax noprefix\n\t"
      "add eax,ecx           \n\t"
      "add ebx,ecx           \n\t"

      "rest:                 \n\t"
      "movss xmm0,[ebx+edx]  ;load 1 float in first
register (xmm0)\n\t"
      "movss xmm1,[eax+edx]  ;load 1 float in second
register (xmm1)\n\t"
      "mulss xmm0,xmm1       ;multiplies both scalar parts
of registers\n\t"
      "movss [eax+edx],xmm0  ;write back the result\n\t"
      "sub edx,4             \n\t"
      "jnz rest              \n\t"
      ".att_syntax prefix    \n\t"
        : : "a" (out), "b" (in), "c"(count), "d"(rest):
"xmm0","xmm1");
      }
      return;
 }
```

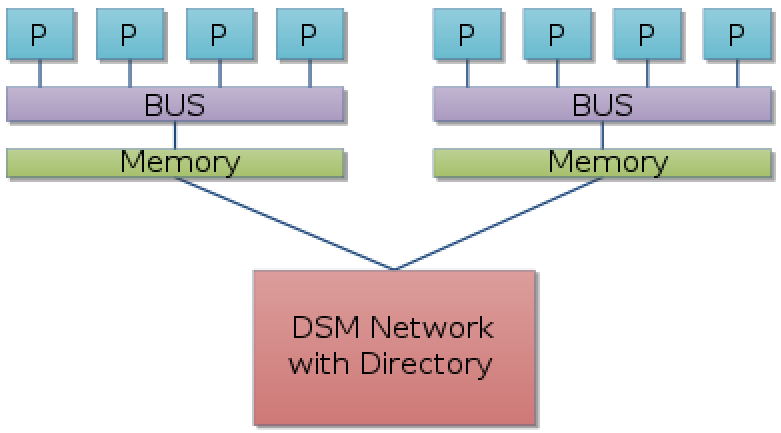
Chapter 11

Non-Uniform Memory Access

Non-Uniform Memory Access or **Non-Uniform Memory Architecture** (**NUMA**) is a computer memory design used in multiprocessors, where the memory access time depends on the memory location relative to a processor. Under NUMA, a processor can access its own local memory faster than non-local memory, that is, memory local to another processor or memory shared between processors.

NUMA architectures logically follow in scaling from symmetric multiprocessing (SMP) architectures. Their commercial development came in work by Burroughs (later Unisys), Convex Computer (later Hewlett-Packard), Silicon Graphics, Sequent Computer Systems, Data General and Digital during the 1990s. Techniques developed by these companies later featured in a variety of Unix-like operating systems, and somewhat in Windows NT.

Basic concept



One possible architecture of a NUMA system. Notice that the processors are connected to the bus or crossbar by connections of varying thickness/number. This shows that different CPUs have different priorities to memory access based on their location.

Modern CPUs operate considerably faster than the main memory to which they are attached. In the early days of computing and data processing the CPU generally ran slower than its memory. The performance lines crossed in the 1960s with the advent of the first supercomputers and high-speed computing. Since then, CPUs, increasingly *starved for data*, have had to stall while they wait for memory accesses to complete. Many supercomputer designs of the 1980s and 90s focused on providing high-speed memory access as opposed to faster processors, allowing them to work on large data sets at speeds other systems could not approach.

Limiting the number of memory accesses provided the key to extracting high performance from a modern computer. For commodity processors, this means installing an ever-increasing amount of high-speed cache memory and using increasingly sophisticated algorithms to avoid "cache misses". But the dramatic increase in size of the operating systems and of the applications run on them has generally overwhelmed these cache-processing improvements. Multi-processor systems make the problem considerably worse. Now a system can starve several processors at the same time, notably because only one processor can access memory at a time.

NUMA attempts to address this problem by providing separate memory for each processor, avoiding the performance hit when several processors attempt to address the same memory. For problems involving spread data (common for servers and similar applications), NUMA can improve the performance over a single shared memory by a factor of roughly the number of processors (or separate memory banks).

Of course, not all data ends up confined to a single task, which means that more than one processor may require the same data. To handle these cases, NUMA systems include additional hardware or software to move data between banks. This operation has the effect of slowing down the processors attached to those banks, so the overall speed increase due to NUMA will depend heavily on the exact nature of the tasks run on the system at any given time.

Cache coherent NUMA (ccNUMA)

Nearly all CPU architectures use a small amount of very fast non-shared memory known as cache to exploit locality of reference in memory accesses. With NUMA, maintaining cache coherence across shared memory has a significant overhead.

Although simpler to design and build, non-cache-coherent NUMA systems become prohibitively complex to program in the standard von Neumann architecture programming model. As a result, all NUMA computers sold to the market use special-purpose hardware to maintain cache coherence, and thus class as "cache-coherent NUMA", or **ccNUMA**.

Typically, this takes place by using inter-processor communication between cache controllers to keep a consistent memory image when more than one cache stores the same memory location. For this reason, ccNUMA may perform poorly when multiple processors attempt to access the same memory area in rapid succession. Operating-system support for NUMA attempts to reduce the frequency of this kind of access by allocating processors and memory in NUMA-friendly ways and by avoiding scheduling and locking algorithms that make NUMA-unfriendly accesses necessary. Alternatively, cache coherency protocols such as the MESIF protocol attempt to reduce the communication required to maintain cache coherency. Scalable Coherent Interface (SCI) is an IEEE standard defining a directory based cache coherency protocol to avoid scalability limitations found in earlier multiprocessor systems. SCI is used as basis for the Numascale NumaConnect technology.

Current ccNUMA systems are multiprocessor systems based on the AMD Opteron, which can be implemented without external logic, and Intel Itanium, which requires the chipset to support NUMA. Examples of ccNUMA enabled chipsets are the SGI Shub (Super hub), the Intel E8870, the HP sx2000 (used in the Integrity and Superdome servers), and those found in recent NEC Itanium-based systems. Earlier ccNUMA systems such as those from Silicon Graphics were based on MIPS processors and the DEC Alpha 21364 (EV7) processor.

Intel announced NUMA Template:Unclear - ccNUMA? introduction to its x86 and Itanium servers in late 2007 with Nehalem and Tukwila CPUs. Both CPU families will share a common chipset; the interconnection is called Intel Quick Path Interconnect (QPI).

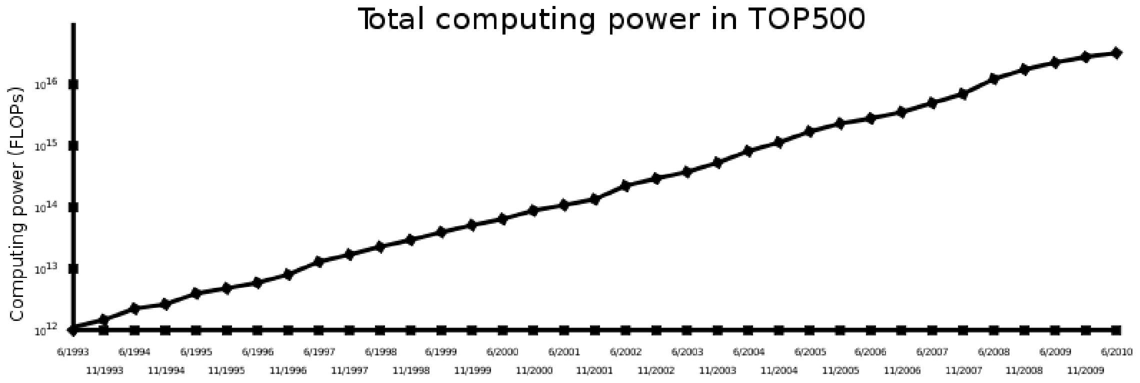
NUMA vs. cluster computing

One can view NUMA as a very tightly coupled form of cluster computing. The addition of virtual memory paging to a cluster architecture can allow the implementation of NUMA entirely in software where no NUMA hardware exists. However, the inter-node latency of software-based NUMA remains several orders of magnitude greater than that of hardware-based NUMA.

Chapter 12

TOP500 and Minisupercomputer

TOP500



Total computational power of the 500 most powerful computer systems in the world, 1993–2010. Note the logarithmic scale.

The **TOP500** project ranks and details the 500 (non-distributed) most powerful known computer systems in the world. The project was started in 1993 and publishes an updated list of the supercomputers twice a year. The first of these updates always coincides with the International Supercomputing Conference in June, the second one is presented in November at the ACM/IEEE Supercomputing Conference. The project aims to provide a reliable basis for tracking and detecting trends in high-performance computing and bases rankings on HPL, a portable implementation of the High-Performance LINPACK benchmark written in Fortran for distributed-memory computers.

The TOP500 list is compiled by Hans Meuer of the University of Mannheim, Germany, Jack Dongarra of the University of Tennessee, Knoxville, Erich Strohmaier and Horst Simon of NERSC/Lawrence Berkeley National Laboratory.

Project history

In the early 1990s, a new definition of supercomputer was needed to produce meaningful statistics. After experimenting with metrics based on processor count in 1992, the idea was born at the University of Mannheim to use a detailed listing of installed systems as the basis. Early 1993 Jack Dongarra was persuaded to join the project with his Linpack benchmark. A first test version was produced in May 1993, partially based on data available on the Internet, including the following sources:

- "List of the World's Most Powerful Computing Sites" maintained by Gunter Ahrendt
- David Kahaner, who had an immense amount of data.

The information from those sources was used for the first two lists. Since June 1993 the TOP500 is produced bi-annually based on site and vendor submissions only.

Since 1993, performance of the #1 ranked position steadily grew in agreement with Moore's law, doubling roughly every 14 months. The fastest system as of November 2010 is roughly 36 thousand times faster (in terms of peak Tflops) than the fastest system as of June 1993.

The systems ranked #1 since 1993

- NUDT Tianhe-1A (China, since November 2010)
- Cray Jaguar (United States, November 2009 - November 2010)
- IBM Roadrunner (United States, June 2008 – November 2009)
- IBM Blue Gene/L (United States, November 2004 – June 2008)
- NEC Earth Simulator (Japan, June 2002 – November 2004)
- IBM ASCI White (United States, November 2000 – June 2002)
- Intel ASCI Red (United States, June 1997 – November 2000)
- Hitachi CP-PACS (Japan, November 1996 – June 1997)
- Hitachi SR2201 (Japan, June 1996 – November 1996)
- Fujitsu Numerical Wind Tunnel (Japan, November 1994 – June 1996)
- Intel Paragon XP/S140 (United States, June 1994 – November 1994)
- Fujitsu Numerical Wind Tunnel (Japan, November 1993 – June 1994)
- TMC CM-5 (United States, June 1993 – November 1993)

November 2010 list

The following table gives the Top 10 positions of the 36th TOP500 List released on November 14, 2010 during SC10 in New Orleans, LA.

Rank	**Rmax Rpeak** (Tflops)	**Name**	**Computer Processor cores**	**Vendor**	**Site Country, Year**	**Operating System**
1	2566.00 4701.00	*Tianhe-1A*	**NUDT YH Cluster** 14,336x6 Xeon +	NUDT	National Supercomputing Center in Tianjin China, 2010	Linux

			7168×14 Fermi, Proprietary			
2	1759.00 2331.00	*Jaguar*	**Cray XT5** 224,162 Opteron	Cray	Oak Ridge National Laboratory United States, 2009	Linux (CLE)
3	1271.00 2984.30	*Nebulae*	**Dawning TC3600 Blade** 55,680 Xeon + 64,960 Tesla, InfiniBand	Dawning	National Supercomputing Center in Shenzhen (NSCS) China, 2010	Linux
4	1192.00 2287.63	*TSUBAME 2.0*	**HP Cluster Platform 3000SL** 73,278 Xeon, Fermi	NEC/HP	GSIC Center, Tokyo Institute of Technology Japan, 2010	Linux (SLES 11)
5	1054.00 1288.63	*Hopper*	**Cray XE6** 153,408 Opteron	Cray	DOE/SC/LBNL/NERSC United States, 2010	Linux (CLE)
6	1050.00 1254.55	*Tera 100*	**Bull Bullx** 138,368 Xeon, InfiniBand	Bull SA	Commissariat à l'énergie atomique (CEA) France, 2010	Linux
7	1042.00 1375.78	*Roadrunner*	**BladeCenter QS22/LS21** 122,400 Cell/Opteron	IBM	Los Alamos National Laboratory United States, 2009	Linux
8	831.70 1028.85	*Kraken*	**Cray XT5** 98,928 Opteron	Cray	National Institute for Computational Sciences United States, 2009	Linux (CLE)
9	825.50 1002.70	*JUGENE*	**Blue Gene/P Solution** 294,912 Power	IBM	Forschungszentrum Jülich Germany, 2009	Linux (SLES 11)
10	816.60 1028.66	*Cielo*	**Cray XE6** 107,152 Opteron	Cray	DOE/NNSA/LANL/SNL United States, 2010	Linux (CLE)

Legend

- **Rank** – Position within the TOP500 ranking. In the TOP500 List table, the computers are ordered first by their Rmax value. In the case of equal performances (Rmax value) for different computers, we have chosen to order by Rpeak. For sites that have the same computer, the order is by memory size and then alphabetically.

- **Rmax** – The highest score measured using the LINPACK benchmark suite. This is the number that is used to rank the computers. Measured in trillions of floating point operations per second, i.e. Teraflops.
- **Rpeak** – This is the theoretical peak performance of the system. Measured in Tflops.
- **Name** – Some supercomputers are unique, at least on its location, and are therefore christened by its owner.
- **Computer** – The computing platform as it is marketed.
- **Processor cores** – The number of active processor cores actively used running Linpack. After this figure is the processor architecture of the cores named. If the interconnect between computing nodes is of interest, it's also included here.
- **Vendor** – The manufacturer of the platform and hardware.
- **Site** – The name of the facility operating the supercomputer.
- **Country** – The country in which the computer is situated.
- **Year** – The year of installation/last major update.
- **Operating System** – The operating system that the computer uses.

Minisupercomputer

Minisupercomputers constituted a short-lived class of computers that emerged in the mid-1980s. As scientific computing using vector processors became more popular, the need for lower-cost systems that might be used at the departmental level instead of the corporate level created an opportunity for new computer vendors to enter the market. As a generalization, the price targets for these smaller computers were one-tenth of the larger supercomputers. These computer systems were characterized by the combination of vector processing and small-scale multiprocessing.

The appearance of even lower-priced scientific workstations based on microprocessors with high performance floating point units (FPUs) during the 1990s (such as the MIPS R8000, IBM POWER2), and Weitek eroded the demand for this class of computer.

The industry magazine Datamation coined the term "crayette" which in short order meant instruction set compatible to Cray Research, Inc.

Notable minisupercomputer companies (alphabetically)

- Ametek Another Caltech/Intel based hypercube
- Alliant Computer Systems (founded 1982 as Dataflow Systems; went bankrupt in 1992)

- American Supercomputer (founded by Mike Flynn, failed 2nd round funding)
- Astronautics (Division founded by Jim Smith, U. Wisc.)
- BBN
- Convex Computer (founded 1982 as Parsec; acquired by Hewlett-Packard in 1995)
- Culler Harris (CHI)
- Culler Scientific
- Cydrome (founded 1984)
- DEC (VAX 9000)
- Elxsi Corporation (founded 1979)
- Encore Computer (founded 1983; acquired by Systems Engineering Laboratories)
- Evans & Sutherland (Computer Division, Mountain View, CA)
- Flexible Computer (FLEX) (founded in TX)
- Floating Point Systems (founded 1970; acquired by Cray Research in 1991)
- Guiltech/SAXPY
- HAL Computer Systems
- ICL (DAP)
- Kendall Square Research
- Key Laboratories
- MasPar
- Meiko Scientific
- Multiflow Computer (founded 1984; ceased operation in 1990)
- Myrias
- Prisma
- Pyramid Technology
- Scientific Computer Systems (founded 1983; switched to high-speed network development in 1989; now defunct)
- Sequent (if Encore is regarded, then, so too these next two firms)
- Solbourne
- SUPRENUM requires German translation
- Supertek Computers (Founded 1985; acquired by Cray Research 1990)
- Thinking Machines Corporation

Chapter 13

CDC Cyber

The **CDC Cyber** range of mainframe-class supercomputers were the primary products of Control Data Corporation (CDC) during the 1970s and 1980s. In their day, they were the computer architecture of choice for scientific and mathematically intensive computing. They were used for modeling fluid flow, material science stress analysis, electrochemical machining analysis, probabilistic analysis, energy and academic computing, radiation shielding modeling, and other applications.

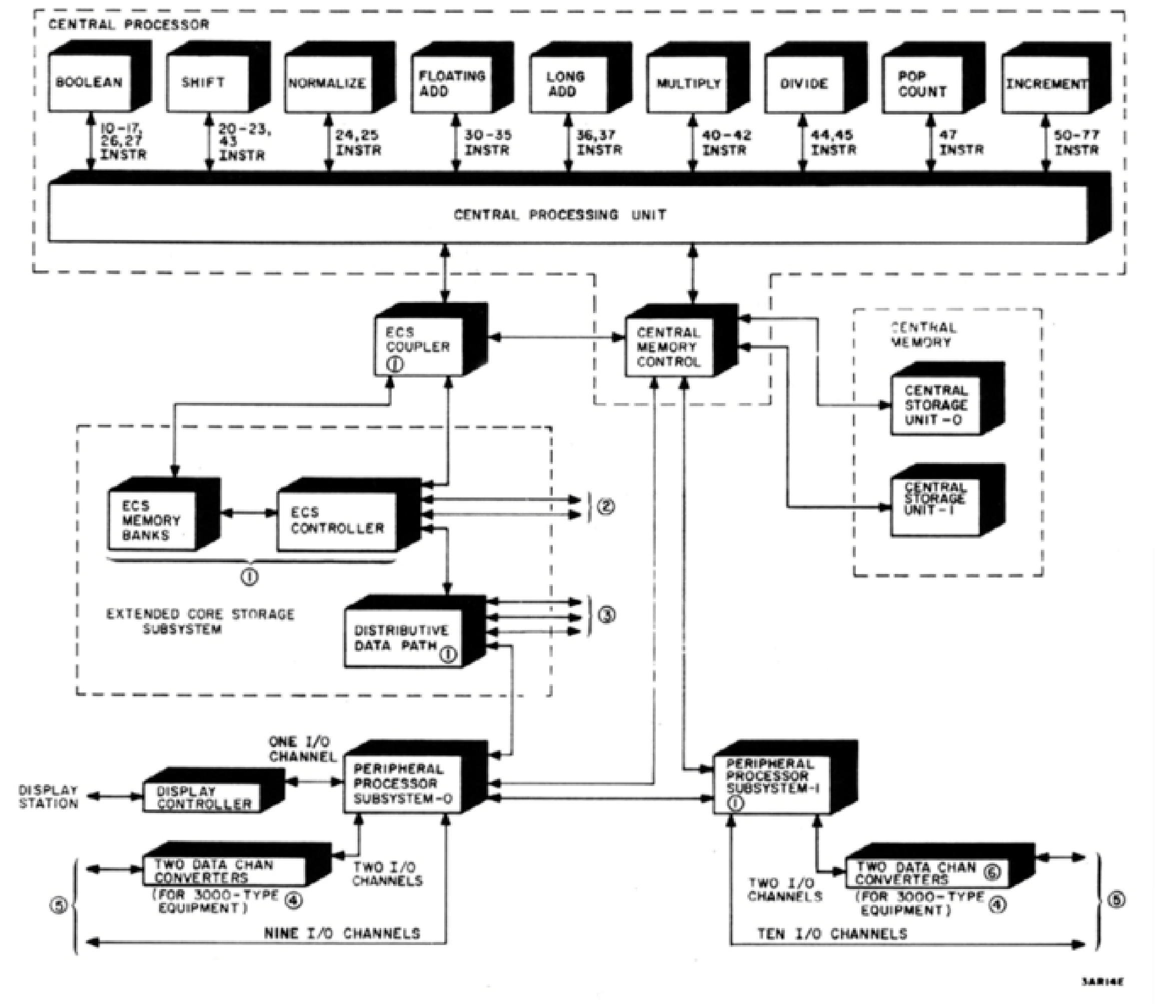
Models

The Cyber line included five very different models of computer:

- The 70 and 170 series based on the architecture of the CDC 6600 and CDC 7600
- The 180 series developed by a team in Canada
- The 200 series based on the CDC STAR-100
- The CYBERPLUS or Advanced Flexible Processor (AFP)
- The Cyber-18 minicomputer based on the CDC 1700

Primarily aimed at large office applications instead of the traditional supercomputer tasks, some of the Cyber machines nevertheless included basic vector instructions for added performance in "traditional" CDC roles.

Cyber 70 and 170 series



Hardware architecture of the CDC Cyber 170 series computer

The Cyber 70 and 170 architectures were successors to the earlier CDC 6600 and CDC 7600 series and therefore shared almost all of the earlier architecture's characteristics. The Cyber-70 series was a minor upgrade from the earlier systems. The Cyber-170 series represented CDCs move from discrete electronic components and core memory to integrated circuits and semiconductor memory. The Cyber-170/700 series was a late-1970s refresh of the Cyber-170 line.

The central processor (CPU) and central memory (CM) operated in units of 60-bit words. In CDC lingo, the term "byte" referred to 12-bit entities (which coincided with the word size used by the peripheral processors). Characters were six bits, operation codes were six bits, and central memory addresses were 18 bits. Central processor instructions were either 15 bits or 30 bits. The 18-bit addressing inherent to the Cyber 170 series imposed a limit of 262,144 (256K) words of main memory, which was semiconductor memory in this series. The central processor had no I/O instructions, relying upon the peripheral processor (PP) units to do I/O.

A Cyber 170-series system consisted of one or two CPUs that ran at either 25 or 40 MHz, and was equipped with 10, 14, 17, or 20 peripheral processors (PP), and up to 24 high-performance channels for high-speed I/O. Due to the relatively slow memory reference times of the CPU (in some models, memory reference instructions were slower than floating point divides), the higher end CPUs (e.g., Cyber-74, Cyber-76, Cyber-175, and Cyber-176) were equipped with 8 or 12 words of high-speed memory used as an instruction cache. Any loop that fit into the cache (which was usually called *in-stack*) would run without referencing main memory for instruction fetch. The lower-end models did not contain an instruction stack. However since up to four instructions were packed into each 60-bit word, some degree of prefetching was inherent in the design.

As with predecessor systems, the Cyber 170 series had eight 18-bit address registers (A0 through A7), eight 18-bit index registers (B0 through B7), and eight 60-bit operand registers (X0 through X7). Seven of the A registers were tied to their corresponding X register. Setting A1 through A5 read that address and fetched it into the corresponding X1 through X5 register. Likewise, setting register A6 or A7 wrote the corresponding X6 or X7 register to central memory at the address written to the A register. A0 was effectively a scratch register.

The higher end CPUs consisted of multiple functional units (e.g., shift, increment, floating add) which allowed some degree of parallel execution of instructions. This parallelism allowed assembly programmers to minimize the effects of the system's slow memory fetch time by *pre-fetching* data from central memory well before that data was needed. By interleaving independent instructions between the memory fetch instruction and the instructions manipulating the fetched operand, the time occupied by the memory fetch could be used for other computation. With this technique, coupled with the handcrafting of tight loops that fit within the instruction stack, a skilled Cyber assembly programmer could write extremely efficient code that made the most of the power of the hardware.

The peripheral processor subsystem used a technique known as *barrel and slot* to share the execution unit; each PP had its own memory and registers, but the processor (the slot) itself executed one instruction from each PP in turn (the barrel). This is a crude form of hardware multiprogramming. The peripheral processors had 4096 bytes of 12-bit memory words and an 18-bit accumulator register. Each PP had access to all I/O channels and all of the system's central memory (CM) in addition to the PP's own memory. The PP instruction set lacked, for example, extensive arithmetic capabilities and did not run user code; the peripheral processor subsystem's purpose was to process I/O and thereby free the more powerful central processor unit(s) to running user computations.

A feature of the 'lower Cyber' CPUs was the *Compare Move Unit* (CMU). It provided four additional instructions intended to aid text processing applications. In an unusual departure from the rest of the 15- and 30-bit instructions, these were 60-bit instructions (3 actually used all 60 bits, the other used 30 bits, but its alignment required 60 bits to be used). The instructions were move a short string, move a long string, compare strings, and compare a collated string. They operated on 6-bit fields (numbered 1 through 10) in central memory. For example, a single instruction could specify "move the 72 character string starting at word 1000 character 3 to location 2000 character 9". The CMU hardware was not included in the higher-end Cyber CPUs, because handcoded loops could run as fast or faster than the CMU instructions.

Later systems typically ran CDC's *NOS* (Network Operating System). Version 1 of NOS continued to be updated until about 1981; NOS version 2 was released early 1982. Besides NOS, the only other operating systems commonly used on the 170 series was *NOS/BE* or its predecessor *SCOPE*, a product of CDC's Sunnyvale division. These operating systems provided time sharing of batch and interactive applications. The predecessor to NOS was *KRONOS* which was in common use up until 1975 or so. Due to the strong dependency of developed applications on the particular installation's character set, many installations chose to run the older operating systems than convert their applications. Other installations would patch newer versions of the operating system to use the older character set to maintain application compatibility. Desktop CYBER emulates CDC Cyber 70 and 170 series mainframes in software running on modern desktop PCs.

Cyber 180 series

As the computing world standardized to an eight-bit byte size, CDC customers started pushing for the Cyber machines to do the same. The result was a new series of systems that could operate in both 60- and 64-bit modes. The 64-bit operating system was called NOS/VE, and supported the virtual memory capabilities of the hardware. The older 60-bit operating systems, NOS and NOS/BE, could run in a special address space for compatibility with the older systems.

The true 180-mode machines were microcoded processors that could, and did, support both instruction sets simultaneously. Their hardware was completely different from the earlier 6000/70/170 machines. The small 170-mode exchange package was mapped into the much larger 180-mode exchange package; within the 180-mode exchange package, there was a VMID—virtual machine identifier—that determined whether the 8/16/64-bit twos complement 180 instruction set or the 12/60-bit ones complement 170 instruction set was executed.

There were 3 true 180s in the initial lineup, codenamed P1, P2, P3. P2 & P3 were larger water-cooled designs. The P2 was designed in Mississauga, by the same team who later designed the smaller P1, and the P3 was designed in Arden Hills, Minnesota. The P1 was a novel air-cooled, 60-board cabinet designed by a group in Mississauga, Ontario; the P1 ran on 60 Hz current (no motor-generator sets needed). A fourth high-end 180 codenamed THETA was also under development in Arden Hills.

The 180s were initially marketed as 170/8xx machines with no mention of the new 8/64-bit system inside. However, the primary control program was a 180-mode program known as EI (Environmental Interface). The 170 operating system (NOS) utilized a single, large, fixed page within the main memory. There were a few clues that an alert user could pick up on, such as the "building page tables" message that flashed on the operator's console at startup and deadstart panels with 16 (instead of 12) toggle switches per PP word on the P2 & P3.

The peripheral processors in the true 180s were always 16-bit machines with the sign bit determining whether a 16/64 bit or 12/60 bit PP instruction was being executed. The single word I/O instructions in the PPs were always 16-bit instructions, so at deadstart the PPs could set up the proper environment to run both EI plus NOS and the customer's existing 170-mode software. To hide this process from the customer, earlier in the 1980s CDC had ceased distribution of the source code for its DDS (Deadstart Diagnostic Sequence) package and turned it into the proprietary CTI (Common Tests & Initialization) package.

The initial 170/800 lineup was: 170/825 (P1), 170/835 (P2), 170/855 (P3), 170/865 and 170/875. The 825 was released initially after some delay loops had been added to its microcode; it seemed the design folks in Toronto had done a little too well and it was too close to the P2 in performance. The 865 and 875 models were revamped 170/760 heads (1 or 2 processors with 6600/7600-style parallel functional units) with larger memories. The 865 used normal 170 memory; the 875 took its faster main processor memory from the *Cyber 205* line.

A year or two after the initial release, CDC announced the 800-series' true capabilities to its customers, and the true 180s were relabeled as the 180/825 (P1), 180/835 (P2), and 180/855 (P3). At some point the model 815 was introduced with the delayed microcode and the faster microcode was restored to the model 825. Eventually the THETA was released as the *Cyber 990*.

Cyber 200 series

In 1974 CDC introduced the STAR architecture. The STAR was an entirely new 64-bit design with virtual memory and vector processing instructions added for high performance on a certain class of math tasks. The STAR's vector pipeline was a *memory to memory* pipe, which supported vector lengths of up to 65,536 elements. Unfortunately, the latencies of the vector pipeline were very long, so peak speed was approached only when very long vectors were used. The scalar processor was relatively slow in comparison to the CDC 7600. As such, the original STAR proved to be a great disappointment when it was released. However many of its problems seemed solvable. Best estimates claim that three STAR-100 systems were delivered.

In the late 1970s, CDC addressed some of these issues with the *Cyber 203*. The new name kept with their new branding, and perhaps to distance itself from the STAR's failure. The Cyber 203 contained redesigned scalar processing and *loosely coupled* I/O design, but retained the STAR's vector pipeline. Best estimates claim that two Cyber 203s were delivered and/or upgraded from STAR-100s.

In 1980, the successor to the Cyber 203, the *Cyber 205* was announced. The UK Meteorological Office at Bracknell, England was the first customer and they received their Cyber 205 in 1981. The Cyber 205 replaced the STAR vector pipeline with redesigned vector pipelines: both scalar and vector units utilized ECL gate array ICs and were cooled with Freon. Cyber 205 systems were available with two or four vector pipelines, with the four-pipe version theoretically delivering 400 64-bit MFLOPs and 800 32-bit MFLOPs. These speeds were rarely seen in practice other than by handcrafted assembly language. The ECL gate array ICs contained 168 logic gates each, with the clock tree networks being tuned by hand-crafted coax length adjustment. It is worth noting that the instruction set would be considered V-CISC (very complex instruction set) among modern processors. Many specialized operations facilitated hardware searches, matrix mathematics, and special instructions that would enable decryption. The original Cyber 205 was renamed to *Cyber 205 Series 400* in 1983 when the Cyber 205 Series 600 was introduced. The Series 600 differed in memory technology and packaging but was otherwise the same. The Cyber 205 architecture evolved into the ETA10 as the design team spun off into ETA Systems in September 1983. A single four-pipe Cyber 205 was installed. All other sites appear to be two-pipe installations with final count to be determined.

Also there was a Cyber 250 which was scheduled for release in 1987 priced at $20 million; it was later renamed the ETA30 after ETA Systems was absorbed back into CDC.

Cyberplus/AFP

At least 21 CYBERPLUS (aka Advanced Flexible Processor, AFP) multiprocessor installations were operational in 1986. These Parallel Processing Systems include from 1 to 256 CYBERPLUS processors providing 250 MFLOPS each, which are connected to an existing CYBER system via a direct memory interconnect architecture (MIA), this was available on NOS 2.2 for the CYBER 170/835, 845, 855 and 180/990 models. Each CYBERPLUS is a 16-bit processor with optional 64-bit floating point capabilities and has 256 K or 512 K words of 64-bit memory.

Each physical CYBERPLUS processor unit was

- 348 cm wide (465 cm with floating point unit)
- 161 cm deep
- 490 cm high
- 1000 kg weight

Software that was bundled with the CYBERPLUS was

- system software

- FORTRAN cross compiler
- MICA (Machine Instruction Cross Assember)
- Load File Builder Utility
- ECHOS (simulator)
- Debug facility
- Dump utility
- Dump analyzer utility
- Maintenance software

Some sites using the CYBERPLUS were the University of Georgia and Gesellschaft fur Trendanalysen (GFTA) in Germany.

A fully configured 256 processor CYBERPLUS system would have a theoretical performance of 64 GFLOPS and would weigh 256 tonnes.

Cyber 18

The Cyber 18 was a 16-bit minicomputer which was a successor to the CDC 1700 minicomputer. It was mostly used in real-time environments. One noteworthy application is as the basis of the 2550 - a communications processor used by CDC 6000 series and Cyber 70/Cyber 170 mainframes. The 2550 was a product of CDC's Communications Systems Division, in Santa Ana, California (STAOPS). STAOPS also produced another communication processor (CP), used in networks hosted by IBM mainframes. This M1000 CP, later renamed C1000, came from an acquisition of Marshall MDM Communications. A 3-board set was added to the Cyber 18 to create the 2550.

The Cyber 18 was generally programmed in Pascal and assembly language; FORTRAN, BASIC, and RPG II were also available. Operating systems included RTOS (Real-Time Operating System), MSOS 5 (Mass Storage Operating System), and TIMESHARE 3 (time-sharing system).

"Cyber 18-17" was just a new name for the System 17, based on the 1784 processor. Other Cyber 18s (Cyber 18-05, 18-10, 18-20, and 18-30) had microprogrammable processors with up to 128K words of memory, four additional general registers, and an enhanced instruction set. The Cyber 18-30 had dual processors. A special version of the Cyber 18, know as the MP32, that was 32-bit instead of 16-bit was created for the National Security Agency for crypto-analysis work. The Soviet Union tried to buy several of these systems and they were being built when the US Government cancelled the order. The parts for the MP32 were absorbed into the Cyber 18 production. One of the uses of the Cyber 18 was monitoring the Alaskan Pipeline.

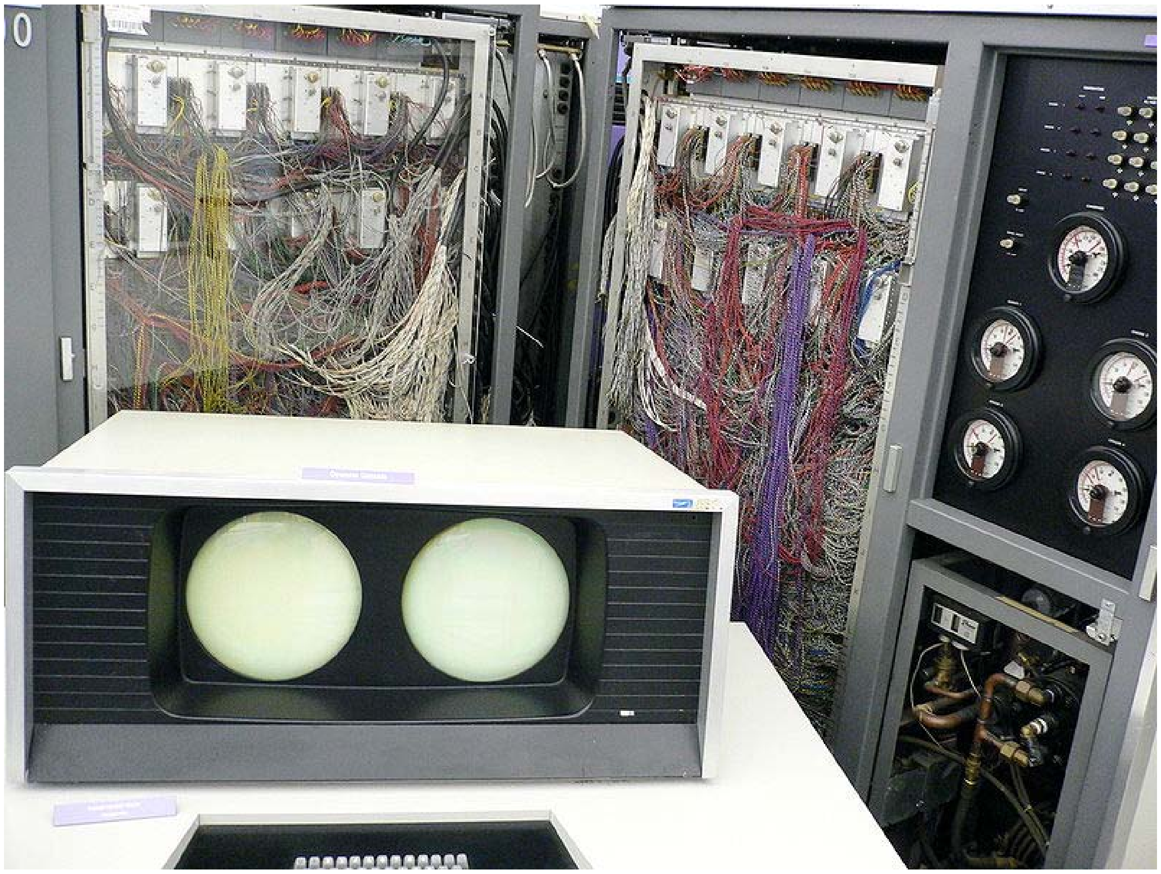
M1000 / C1000 later renamed Cyber 1000 was used as a message store and forward system used by the Federal Reserve System. A version of the Cyber 1000 with its hard drive removed was used by Bell Telephone. This was a RISC processor (Reduced Instruction Set Computer). An improved version know as the Cyber 1000-2 with the Line Termination Sub-System added 256 Zilog Z80 microprocessors. The Bell Operating

Companies purchased large numbers of these systems in the mid to late 1980s for data communications. In the late 1980s the XN10 was released with an improved processor (a direct memory access instruction was added) as well as a size reduction from two cabinets to one. The XN20 was an improved version of the XN10 with a much smaller footprint. The Line Termination Sub-System was redesigned to use the improved Z180 microprocessor (the Buffer Controller card, Programmable Line Controller card and two Communication Line Interface cards were incorporated on to a single card). The XN20 was in pre-production stage when the Communication Systems Division was shut down in 1992.

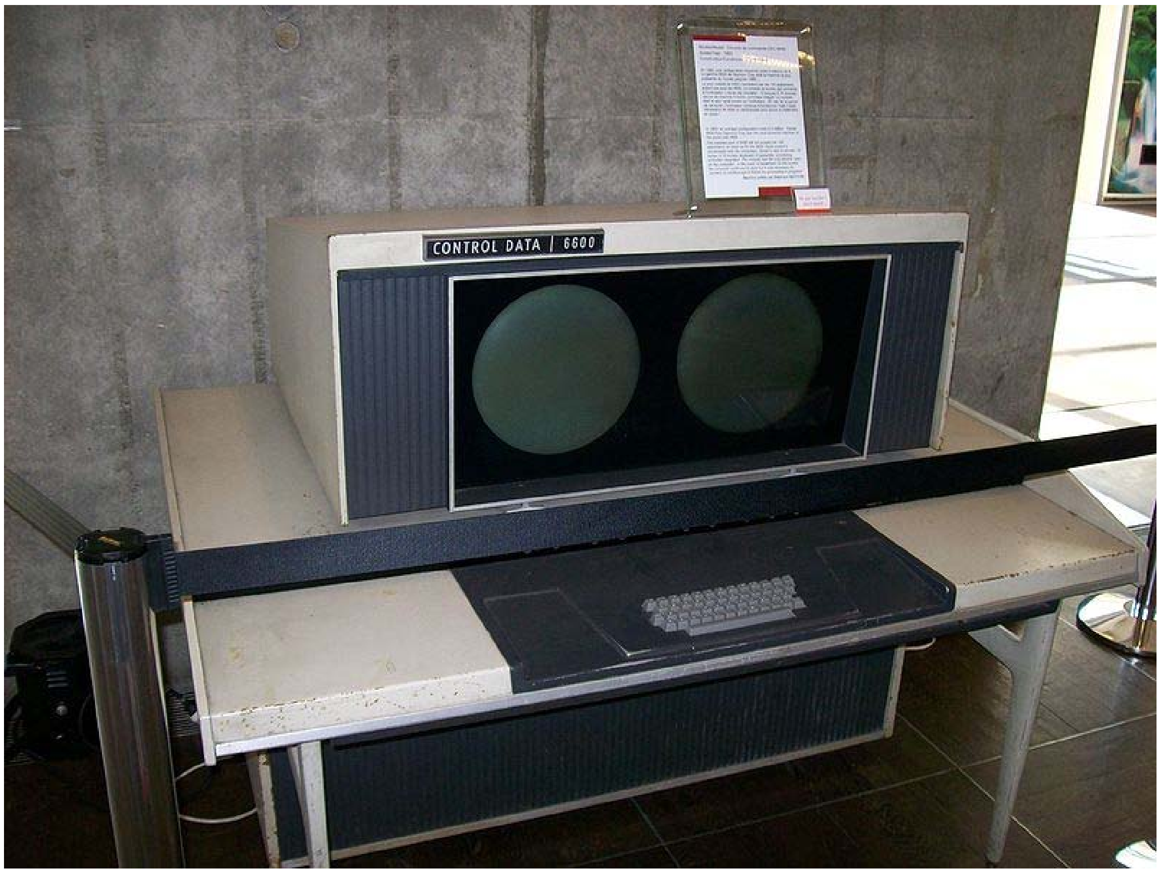
Jack Ralph was the chief architect of the Cyber 1000-2, XN-10 and XN-20 systems. Dan Nay was the chief engineer of the XN-20.

Chapter 14

CDC 6600



The CDC 6600. Behind the system console are two of the "arms" of the plus-sign shaped cabinet with the covers opened. Individual modules can be seen inside. The racks holding the modules are hinged to give access to the racks behind them. Each arm of the machine had up to four such racks. On the right is the cooling system.



A CDC 6600 system console. The displays were driven through software, primarily to provide text display (in a choice of three sizes). It also provided a way to draw simple graphics, one dot at a time.

The **CDC 6600** was a mainframe computer from Control Data Corporation, first delivered in 1964. It is generally considered to be the first successful supercomputer, outperforming its fastest predecessor, IBM 7030 Stretch, by about three times. With performance of about 1 MFLOP, it remained the world's fastest computer from 1964–1969, when it relinquished that status to its successor, the CDC 7600.

The system organization of the CDC 6600 was used for the simpler (and slower) CDC 6400, and later a version containing two 6400 processors known as the CDC 6500. These machines were instruction-compatible with the 6600, but ran slower due to a much simpler and more sequential processor design. The entire family is now referred to as the CDC 6000 series. The CDC 7600 was originally to be compatible as well, starting its life as the CDC 6800, but during the design compatibility was dropped in favor of outright performance. While the 7600 CPU remained compatible with the 6600, allowing portable user code, the PPUs were different, requiring a different operating system.

A CDC 6600 is on display at the Computer History Museum in Mountain View, California.

History and impact

CDC's first products were based on the machines designed at ERA, which Seymour Cray had been asked to update after moving to CDC. After an experimental machine known as the *Little Character*, they delivered the CDC 1604, one of the first commercial transistor-based computers, and one of the fastest machines on the market. Management was delighted, and made plans for a new series of machines that were more tailored to business use; they would include instructions for character handling and record keeping for instance. Cray was not interested in such a project, and set himself the goal of producing a new machine that would be 50 times faster than the 1604. When asked to complete a detailed report on future plans at one and five years into the future, he wrote back that his five year goal was "to produce the largest computer in the world", "large" at that time being synonymous with "fast", and that his one year plan was "to be one-fifth of the way".

Taking his core team to new offices nearby the original CDC headquarters, they started to experiment with higher quality versions of the "cheap" transistors Cray had used in the 1604. After much experimentation, they found that there was simply no way the germanium-based transistors could be run much faster than the 1604. The "business machine" that management had originally wanted, now forming as the CDC 3000 series, pushed them about as far as they could go. Cray then decided the solution was to work with the then-new silicon-based transistors from Fairchild Semiconductor, which were just coming onto the market and offered dramatically improved switching performance.

During this period, CDC grew from a startup to a large company and Cray became increasingly frustrated with what he saw as ridiculous management requirements. Things became considerably more tense in 1962 when the new CDC 3600 started to near production quality, and appeared to be exactly what management wanted, when they wanted it. Cray eventually told CDC's CEO, William Norris that something had to change, or he would leave the company. Norris felt he was too important to lose, and gave Cray the green light to set up a new lab wherever he wanted.

After a short search, Cray decided to return to his home town of Chippewa Falls, WI, where he purchased a block of land and started up a new lab. Although this process introduced a fairly lengthy delay in the design of his new machine, once in the new lab things started to progress quickly. By this time, the new transistors were becoming quite reliable, and modules built with them tended to work properly on the first try. Working with Jim Thornton, who was the system architect and the 'hidden genius' behind the 6600, the machine soon took form.

More than 100 CDC 6600s were sold over the machine's lifetime. Many of these went to various nuclear bomb-related labs, and quite a few found their way into university computing labs. Cray immediately turned his attention to its replacement, this time setting a goal of 10 times the performance of the 6600, delivered as the CDC 7600. The later CDC Cyber 70 and 170 computers were very similar to the CDC 6600 in overall design.

Description

Typical machines of the era used a single complex CPU to drive the entire system. A typical program would first load data into memory (often using pre-rolled library code), process it, and then write it back out. This required the CPUs to be fairly complex in order to handle the complete set of instructions they would be called on to perform. A complex CPU implied a large CPU, introducing signalling delays while information flowed between the individual modules making it up. These delays set a maximum upper limit on performance, the machine could only operate at a cycle speed that allowed the signals time to arrive at the next module.

Cray took another approach. At the time, CPUs generally ran slower than the main memory they were attached to. For instance, a processor might take 15 cycles to multiply two numbers, while each memory access took only one or two. This meant there was a significant time where the main memory was idle. It was this idle time that the 6600 exploited.

Instead of trying to make the CPU handle all the tasks, the 6600s handled arithmetic and logic only. This resulted in a much smaller CPU which could operate at a higher clock speed. Combined with the faster switching speeds of the silicon transistors, the new CPU design easily outperformed everything then available. The new design ran at 10 MHz (100 ns cycle), about ten times faster than other machines on the market. In addition to the clock being faster, the simple processor executed instructions in fewer clock cycles; for instance, the CPU could complete a multiplication in ten cycles.

However, the CPU could only execute a limited number of simple instructions. A typical CPU of the era had a complex instruction set, which included instructions to handle all the normal "housekeeping" tasks such as memory access and input/output. Cray instead implemented these instructions in separate, simpler processors dedicated solely to these tasks, leaving the CPU with a many smaller instruction set. (This was the first of what later came to be called reduced instruction set computer (RISC) design.) By allowing the CPU, peripheral processors (PPs) and I/O to operate in parallel, the design considerably improved the performance of the machine. Under normal conditions a machine with several processors would also cost a great deal more. Key to the 6600's design was to make the I/O processors, known as *peripheral processors* (PPs), as simple as possible. The PPs were based on the simple 12-bit CDC 160A, which ran much slower than the CPU, gathering up data and "squirting" it into main memory at high speed via dedicated hardware.

The machine as a whole operated in a fashion known as *barrel and slot*, the "barrel" referring to the ten PPs, and the "slot" the main CPU. For any given slice of time, one PP was given control of the CPU, asking it to complete some task (if required). Control was then handed off to the next PP in the barrel. Programs were written, with some difficulty, to take advantage of the exact timing of the machine to avoid any "dead time" on the CPU. With the CPU running much faster than on other computers, each memory access took ten CPU clock cycles to complete, so by using ten PPs, each PP was guaranteed one memory access per machine cycle.

The 10 PPs were implemented virtually; there was CPU hardware only for a single PP. This CPU hardware was shared and operated on 10 PP register sets which represented each of the 10 PP *states* (similar to modern multithreading processors). The PP *register barrel* would "rotate", with each PP register set presented to the "slot" which the actual PP CPU occupied. The shared CPU would execute all or some portion of a PP's instruction whereupon the barrel would "rotate" again, presenting the next PP's register set (state). Multiple "rotations" of the barrel were needed to complete an instruction. A complete barrel "rotation" occurred in 1000 nanoseconds (100 nanoseconds per PP), and an instruction could take from 1 to 5 "rotations" of the barrel to be completed, or more if it was a data transfer instruction.

The basis for the 6600 CPU is what would today be referred to as a RISC system, one in which the processor is tuned to do instructions which are comparatively simple and have limited and well-defined access to memory. The philosophy of many other machines was toward using instructions which were complicated — for example, a single instruction which would fetch an operand from memory and add it to a value in a register. In the 6600, loading the value from memory would require one instruction, and adding it would require a second. While slower in theory due to the additional memory accesses, the PPs offloaded this expense. This simplification also forced programmers to be very aware of their memory accesses, and therefore code deliberately to reduce them as much as possible.

The Central Processor (CP)

The Central Processor (CP) and main memory of the 6400, 6500, and 6600 machines had a 60-bit word length. The Central Processor had eight general purpose 60-bit registers X0 through X7, eight 18-bit address registers A0 through A7, and eight 18-bit scratchpad registers B0 through B7. B0 was held permanently at zero by the hardware; many programmers found it useful to set B1 to 1 and then treat it as similarly inviolate.

The CP had no instructions for input and output, which are accomplished through Peripheral Processors (below). No opcodes were specifically dedicated to loading or storing memory; this occurred as a side effect of assignment to certain A registers. Setting A1 through A5 loaded the word at that address into X1 through X5 respectively; setting A6 or A7 stored a word from X6 or X7. No side effects were associated with A0. A separate hardware load/store unit handled the actual data movement independently of the operation of the instruction stream, allowing other operations to complete while memory was being accessed, which required (best case) eight cycles.

The 6600 CP included 10 parallel functional units, allowing multiple instructions to be worked on at the same time. Today, this is known as a superscalar design, but it was unique for its time. Unlike most modern CPU designs, functional units were not pipelined; the functional unit would become busy when an instruction was "issued" to it and would remain busy for the entire time required to execute that instruction. (By contrast, the CDC 7600 introduced pipelining into its functional units.) In the best case, an instruction could be issued to a functional unit every 100 ns clock cycle. The system read and decoded instructions from memory as fast as possible, generally faster than they could be completed, and fed them off to the units for processing. The units were:

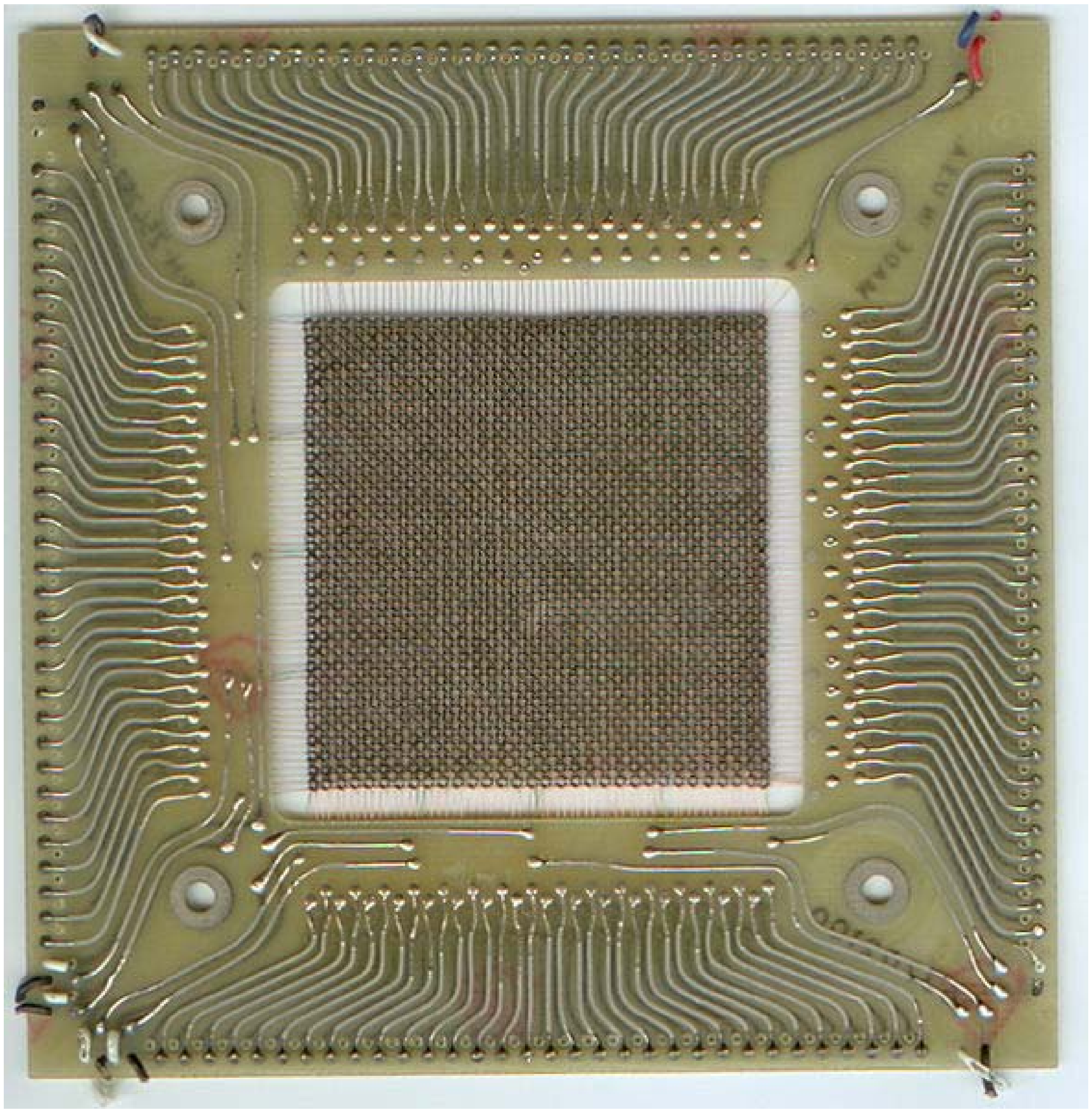
- floating point multiply (2 copies)
- floating point divide
- floating point add
- "long" integer add
- incrementers (2 copies; performed memory load/store)
- shift
- boolean logic
- branch

Floating-point operations were given pride of place in this architecture: the CDC 6600 (and kin) stand virtually alone in being able to execute a 60-bit floating point multiplication in time comparable to that for a program branch.

Previously executed instructions were saved in an eight-word cache, called the "stack". In-stack jumps were quicker than out-of-stack jumps because no memory fetch was required. The stack was flushed by an unconditional jump instruction, so unconditional jumps at the ends of loops were conventionally written as conditional jumps that would always succeed.

The system used a 10-megahertz clock, but used a four-phase signal, so the system could at times effectively operate at 40 MHz. A floating-point multiplication took ten cycles, a division took 29, and the overall performance, taking into account memory delays and other issues, was about 3 MFLOPS. Using the best available compilers, late in the machine's history, FORTRAN programs could expect to maintain about 0.5 MFLOPS.

Memory organization



CDC-6600 core memory

User programs are restricted to use only a contiguous area of main memory. The portion of memory to which an executing program has access is controlled by the *RA* (Relative Address) and *FL* (Field Length) registers which are not accessible to the user program. When a user program tries to read or write a word in central memory at address *a*, the processor will first verify that a is between 0 and FL-1. If it is, the processor accesses the word in central memory at address RA+a. This process is known as base-bound relocation; each user program sees core memory as a contiguous block words with length FL, starting with address 0; in fact the program may be anywhere in the physical memory. Using this technique, each user program can be moved ("relocated") in main memory by the operating system, as long as the RA register reflects its position in memory. A user program which attempts to access memory outside the allowed range (that is, with an address which is not less than FL) will trigger an interrupt, and will be terminated by the operating system. When this happens, the operating system may create a core dump which records the contents of the program's memory and registers in a file, allowing the developer of the program a means to know what happened. Note the distinction with virtual memory systems; in this case, the entirety of a process's addressable space must be in core memory, must be contiguous, and its size cannot be larger than the real memory capacity.

All but the first seven CDC 6000 series machines could be configured with an optional Extended Core Storage (ECS) system. ECS was built from a different variety of core memory than was used in the central memory. This made it economical for it to be both larger and slower. The primary reason was that ECS memory was wired with only two wires per core (contrast with 5 for central memory), Because it performed very wide transfers, its sequential transfer rate was the same as that of the small core memory. A 6000 CPU could directly perform block memory transfers between a user's program (or operating system) and the ECS unit. Wide data paths were used, so this was a very fast operation. Memory bounds were maintained in a similar manner as central memory — with an RA/FL mechanism maintained by the operating system. ECS could be used for a variety of purposes, including containing user data arrays that were too large for central memory, holding often-used files, swapping, and even as a communication path in a multi-mainframe complex.

Peripheral Processors (PPs)

To handle the 'household' tasks which other designs put in the CPU, Cray included ten other processors, based partly on his earlier computer, the CDC 160A. These machines, called Peripheral Processors, or PPs, were full computers in their own right, but were tuned to performing I/O tasks and running the operating system. One of the PPs was in overall control of the machine, including control of the program running on the main CPU, while the others would be dedicated to various I/O tasks — quite similarly to I/O channels in IBM mainframes of the time. When the program needed to perform some sort of I/O, it instead loaded a small program into one of these other machines and let it do the work. The PP would then inform the CPU when the task was complete with an interrupt.

Each PP included its own memory of 4096 12-bit words. This memory served for both for I/O buffering and program storage, but the execution units were shared by 10 PPs, in a configration called the Barrel and slot. This meant that the execution units (the "slot") would execute one instruction cycle from the first PP, then one instruction cycle from the second PP, etc. in a round robin fashion. This was done both to reduce costs, and because access to CP memory required 10 PP clock cycles: when a PP accesses CP memory, the data is available next time the PP receives its slot time.

Wordlengths, characters

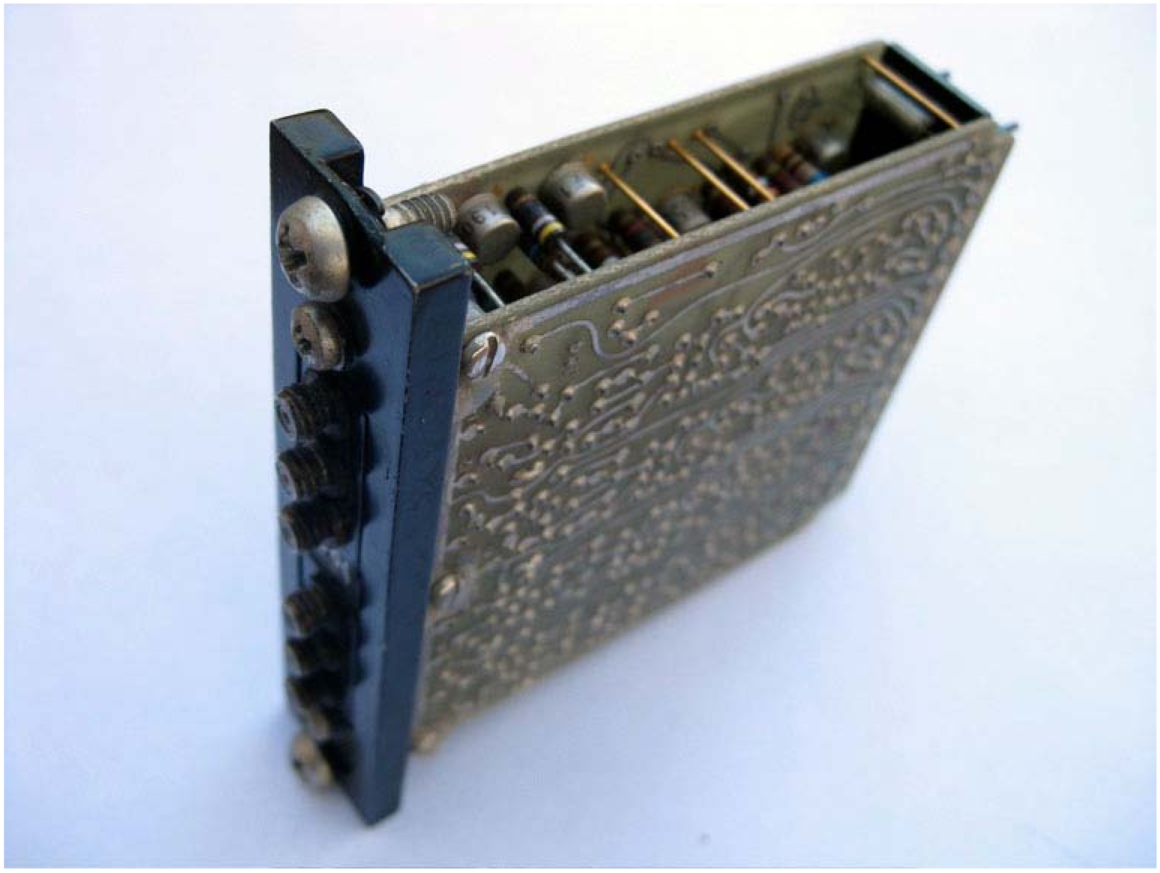
The central processor had 60-bit words, whilst the peripheral processors had 12-bit words. CDC used the term "byte" to refer to 12-bit entities used by peripheral processors; characters were 6-bit, and central processor instructions were either 15 bits, or 30 bits with a signed 18-bit address field, the latter allowing for a directly addressable memory space of 128K words of central memory (converted to modern terms, with 8-bit bytes, this is 0.94 MB). The signed nature of the address registers limited an individual program to 128K words. (Later CDC 6000-compatible machines could have 256K or more words of central memory, budget permitting, but individual user programs were still limited to 128K words of CM.) Central processor instructions started on a word boundary when they were the target of a jump statement or subroutine return jump instruction, so no-operations were sometimes required to fill out the last 15, 30 or 45 bits of a word.

The 6-bit characters, in an encoding called display code, could be used to store up to 10 characters in a word. They permitted a character set of 64 characters, which is enough for all upper case letters, digits, and some punctuation. Certainly, enough to write FORTRAN, or print financial or scientific reports. There were actually two variations of the display code character sets in use, 64-character and 63-character. The 64-character set had the disadvantage that two consecutive ':' (colon) characters might be interpreted as the end of a line if they fell at the end of a 10-byte word. A later variant, called 6/12 display code, was also used in the KRONOS and NOS timesharing systems to allow full use of the ASCII character set in a manner somewhat compatible with older software.

With no byte addressing instructions at all, code had to be written to pack and shift characters into words. The very large words, and comparatively small amount of memory, meant that programmers would frequently economize on memory by packing data into words at the bit level.

It is interesting to note that due to the large word size, and with 10 characters per word, it was often faster to process words full of characters at a time — rather than unpacking/processing/repacking them. For example, the CDC COBOL compiler was actually quite good at processing decimal fields using this technique. These sorts of techniques are now commonly used in the 'multi-media' instructions of current processors.

Physical design



A CDC 6600 cordwood logic module. The coaxial connectors are test points. The module is cooled conductively via the front panel.

The machine was built in a plus-sign-shaped cabinet with a pump and heat exchanger in the outermost 18 in (46 cm) of each of the four arms. Cooling was done with Freon circulating within the machine and exchanging heat to an external chilled water supply. Each arm could hold four chassis, each about 8 in (20 cm) thick, hinged near the center, and opening a bit like a book. The intersection of the "plus" was filled with cables which interconnected the chassis. The chassis were numbered from 1 (containing all 10 PPUs and their memories, as well as the 12 rather minimal I/O channels) to 16. The main memory for the CPU was spread over many of the chassis. In a system with only 64K words of main memory, one of the arms of the "plus" was omitted.

The logic of the machine was packaged into modules about 2.5 in (64 mm) square and about 1 in (2.5 cm) thick. Each module had a connector (30 pins, 2 vertical rows of 15) on one edge, and six test points on the opposite edge. The module was placed between two aluminum cold plates to remove heat. The module itself consisted of two parallel printed circuit boards, with components mounted either on one of the boards or between the two boards. This provided a very dense, if somewhat difficult to repair, package with good heat removal that was known as cordwood construction.

Operating system and programming

There was a sore point with the 6600 operating system support — slipping timelines. The machines originally ran a very simple job-control system known as *COS* (Chippewa Operating System), which was quickly "thrown together" based on the earlier CDC 3000 operating system in order to have something running to test the systems for delivery. However the machines were intended to be delivered with a much more powerful system known as *SIPROS* (for Simultaneous Processing Operating System), which was being developed at the company's System Sciences Division in Los Angeles. Customers were impressed with SIPROS's feature list, and many had SIPROS written into their delivery contracts.

SIPROS turned out to be a major fiasco. Development timelines continued to slip, costing CDC major amounts of profit in the form of delivery delay penalties. After several months of waiting with the machines ready to be shipped, the project was eventually cancelled. The programmers who had worked on COS had little faith in SIPROS (likely due largely to not invented here syndrome) and had continued working on improving COS.

Operating system development then split into two camps. The CDC-sanctioned evolution of COS was undertaken at the Sunnyvale (California) software development lab. Many customers eventually took delivery of their systems with this software, then known as *SCOPE* (Supervisory Control Of Program Execution). (Some Control Data Field Engineers used to refer to SCOPE as *Sunnyvale's Collection Of Programming Errors*). SCOPE version 1 was, essentially, dis-assembled COS; SCOPE version 2 included new device and file system support; SCOPE version 3 included permanent file support, EI/200 remote batch support, and INTERCOM time sharing support. SCOPE always had significant reliability and maintainability issues.

The underground evolution of COS took place at the Arden Hills, Minnesota assembly plant. *MACE* ([Greg] Mansfield And [Dave] Cahlander Executive) was written largely by a single programmer in the off-hours when machines were available. Its feature set was essentially the same as COS and SCOPE 1. It retained the earlier COS file system, but made significant advances in code modularity to improve system reliability and adaptiveness to new storage devices. MACE was never an official product, although many customers were able to wrangle a copy from CDC.

MACE was later used as the basis of *KRONOS*, named after the Greek god of time. The main marketing reason for its adoption was the development of its TELEX time sharing feature and its BATCHIO remote batch feature. KRONOS continued to use the COS/SCOPE 1 file system with the addition of a permanent file feature.

An attempt to unify the SCOPE and KRONOS operating system products produced *NOS*, (Network Operating System). NOS was intended to be the sole operating system for all CDC machines, a fact CDC promoted heavily. Many SCOPE customers remained software-dependent on the SCOPE architecture, so CDC simply renamed it *NOS/BE*

(Batch Environment), and were able to claim that everyone was thus running NOS. In practice, it was far easier to modify the KRONOS code base to add SCOPE features than the reverse.

The assembly plant environment also produced other operating systems which were never intended for customer use. These included the engineering tools SMM for hardware testing, and KALEIDOSCOPE, for software smoke testing. Another commonly used tool for CDC Field Engineers during testing was MALET (Maintenance Application Language for Equipment Testing), which was used to stress test components and devices after repairs and/or servicing by engineers. Testing conditions often used hard disk packs and magnetic tapes which were deliberately marked with errors to determine if the errors would be detected by MALET and the engineer.

Chapter 15

CDC 7600 and CDC 8600

CDC 7600

The **CDC 7600** was the Seymour Cray-designed successor to the CDC 6600, extending Control Data's dominance of the supercomputer field into the 1970s. The 7600 ran at 36.4 MHz (27.5 ns clock cycle) and had a 65 Kword primary memory using core and variable-size (up to 512 Kword) secondary memory (depending on site). It was generally about ten times as fast as the CDC 6600, and could deliver about 10 MFLOPS on hand-compiled code, with a peak of 36 MFLOPS. When the system was released in 1969, it carried a price tag around $5 million, more as options and features were added.

Although the 7600 shared many features of the 6600, including hardware, instructions, and its 60-bit word size, it was not object-code compatible to the CDC 6600. In addition, it was not entirely source-code (COMPASS) compatible, as some instructions in the 7600 did not exist in the 6600, and vice-versa. It had originally been named the CDC 6800, but was changed to 7600 when Cray decided it could not be completely compatible.

After the 6600 started to near production quality, Cray lost interest in it and turned to designing its replacement. Making a machine "somewhat" faster would not be too difficult in the late 1960s; the introduction of integrated circuits allowed for denser packing of components, and in turn a high clock speed. Transistors in general were also getting somewhat faster as the production processes and quality improved. However these sorts of improvements might be expected to make a machine twice as fast, perhaps as much as five times, but not the tenfold increase he demanded. Likewise the 6600 already had a hard time filling its existing ten functional units, so simply adding more parallelism wouldn't help all that much.

In order to solve this problem, Cray turned to the concept of an instruction pipeline. While the 6600 could work on several instructions at once, it had to wait for any one to complete its trip through a functional unit before moving on to the next. For some period of time, the majority of the circuitry in any one unit was not being used. A pipeline improves on this by feeding in the next instruction before the first has completed, thereby having each unit effectively work in "parallel", as well as the machine as a whole. The improvement in performance generally depends on the number of steps the unit takes to complete, for instance, the 6600's divide unit took 10 cycles to complete an instruction, so by pipelining the units it could be expected to gain about 10 times the speed.

Things are never that simple, however. Pipelining requires that the unit's internals can be effectively separated to the point where each step of the operation is running on completely separate circuitry. This is rarely achievable in the real world. Nevertheless, the use of pipelining on the 7600 improved performance over the 6600 by a factor of about 3.

As always, Cray's design also focussed on packaging to reduce size, shorten signal paths, and thereby increase operating frequency. For the 7600 each circuit module actually consisted of up to six PC boards, each one stuffed with subminiature resistors, diodes, and transistors. The six boards were stacked up and then interconnected along their edges, making for a very compact, but basically unrepairable module.

However the same dense packing also led to the machine's biggest problem—heat. For the 7600, Cray once again turned to his refrigeration engineer, Dean Roush, formerly of the Amana company. Roush added an aluminum plate to the back of each side of the cordwood stack, which were in turn cooled by a liquid freon system running through the core of the machine. Since the system was mechanical and therefore prone to failure, the 7600 was redesigned into a large "C" shape to allow access to the modules on either side of the cooling piping by walking into the inside of the C and opening the cabinet.

From a high-level perspective, the 7600 was quite similar to the 6600. At the time computer memory could be arranged in blocks with independent access paths, and Cray's designs used this to their advantage. While most machines would use a single CPU to run all the functionality of the system, Cray realized that this meant each memory block spent a considerable amount of time idle while the CPU was processing instructions and accessing other blocks. In order to take advantage of this, the 6600 and 7600 left mundane housekeeping tasks, printing output or reading punched cards for instance, to a series of ten smaller 12-bit machines based on the CDC 160A known as *Peripheral Processor*s or PP's. For any given cycle of the machine one of the PP's was in control, feeding data into the memory while the main processor was crunching numbers. When the cycle completed, the next PP was given control. In this way the memory always held up-to-date information for the main processor to work on (barring delays in the external devices themselves), eliminating delays on data as well as allowing the CPU to be built for mathematical performance and nothing else. The PPU could have been called a very smart "communications channel".

Like the 6600, the 7600 used 60-bit words with instructions that were generally 15-bits in length (although there were longer versions). However the instruction set itself had changed to reflect the new internal memory layout, thereby rendering it incompatible with the earlier 6600. The machines were similar enough to make porting of compilers and operating systems possible without too much trouble. The machine initially did not come with software; sites had to be willing to write their own operating system, like

NLTSS, NCAROS, and others; and compilers like LRLTRAN [Livermore's version of Fortran with dynamic memory management and other non-standard features].

From the period from about 1969 to 1975, the CDC 7600 was generally regarded as the fastest computer in the world at the time except specialized units. However, even with the advanced mechanicals and cooling the 7600 was prone to failure. Both LLNL and NCAR reported that the machine would break down at least once a day, often 4 or 5 times. Acceptance at installation sites took years while the bugs were worked out, and while the machine generally sold well enough given its "high end" niche, it is unlikely the machine generated any sort of real profits for CDC. The successor CDC 8600 was never completed, and Seymour Cray went on to form his own company, Cray Research.

One surviving 7600 is partially on display at the Computer History Museum. Its sheer size allows only 2 corner units to be shown. The rest is in storage.

Another 7600 is on display at the Chippewa Falls Museum of Industry and Technology, along with its console and a tape controller.

CDC 8600

The **CDC 8600** was the last of Seymour Cray's supercomputer designs while working for the Control Data Corporation. The "natural successor" to the CDC 6600 and CDC 7600, the 8600 was intended to be about 10 times as fast as the 7600, already the fastest computer on the market.

Development started in 1968, shortly after the release of the 7600, but the project soon started to bog down. By 1971 CDC was having cash flow problems and the design was still not coming together, prompting Cray to leave the company in 1972. The 8600 design effort was eventually cancelled in 1974, and Control Data moved on to the CDC STAR-100 series instead.

Design

In the 1960s computer design was based on mounting electronic components (transistors, resistors, etc.) on circuit boards. Several boards would be use to make a discrete logic element of the machine, known as a "module". As computer power increased the complexity of the modules did too, and even a single faulty part or solder joint would render the entire module inoperative. Cray was well known in the industry for making seemingly impossibly complex modules work.

Overall machine "cycle speed" is strongly related to the signal path – the length of the wiring – forcing high speed computers to make their modules as small as possible. This was at odds with the need to make the modules themselves more complex to increase computing power. By the late 1960s individual components had stopped getting much smaller, and although integrated circuits addressed the size issue, their simple MOSFET-based technology didn't have the performance needed for high-speed applications. So in order to increase the complexity of the machines, the modules would have to grow.

Cray aimed to solve these contradictory problems by doing both; making each module larger and crammed with many more components, while at the same time making the computer as a whole smaller by packing the modules closer together inside the machine. In the case of the 8600, this led to modules containing eight four-layer circuit boards about 8" by 6", resulting in a stack the size of a large textbook and using up about 3 kilowatts of power.

Cooling the modules proved to be a major problem. Cray's refrigeration engineer, Dean Roush, formerly of Amana, placed a sheet of copper inside each of the circuit boards, removing the heat to a copper block on one end where it was cooled by a freon system. This further increased the weight and complexity of the modules, to the point where each one weighed about 15 pounds. The modules were then packed into a mainframe chassis that was comparatively tiny, a 16-sided cylinder about one meter across and high, sitting on top of a ring of power supplies. The external cooling system was considerably larger than the machine itself. Perhaps unsurprisingly, the 8600 bears a strong resemblance to the later Cray-2. (prototype photo)

The components themselves were likewise improved over previous designs. The main CPU circuits moved to ECL-based logic, allowing the clock speed to be increased to 8 ns (125 MHz) from the 7600's 27.5 ns – an increase of about four times. Main memory was also moved to an ECL implementation and the machine was equipped with a whopping 256k-words (2 megabytes) standard. The memory was spread across 64 banks, thereby allowing fast access at about 8 ns/word even though the cycle time of any one bank was about 250 ns. A high-speed core memory with a 20 ns access (overall) was also designed as a backup to the semiconductor version.

Cray decided that the 8600 would include four complete CPUs sharing the main memory. In order to improve overall throughput, the machine could be operated in a special mode in which a single instruction was sent to all four processors with different data. This technique, today known as SIMD, reduced the total number of memory accesses because the instruction was only read once, instead of four times. Each processor was about 2.5 times as fast as a 7600, so with all four running the machine as a whole would be about 10 times as fast, at about 100 MFLOPS.

The 8600 was the first CDC design to move to ASCII-based processing, and therefore used a 64-bit word (eight bytes) instead of the earlier 60-bit word (ten 6-bit characters) used on the 6600 and 7600. As in prior designs, instructions were "stuffed" into words, with each instruction taking up either 16 or 32 bits (up from 15/30). The 8600 no longer used the A or B registers as in previous designs, and included a set of 16 general-purpose

X registers instead. A 6600/7600 Peripheral Processor system was used for I/O, largely unchanged.

Company problems

In 1971 Control Data was undergoing a "belt tightening" due to the cost of an ongoing lawsuit against IBM, and all divisions were asked to reduce their payroll by 10%. Cray begged to be exempted in order to get the 8600 shipping, and when this request was refused he instead had his own pay cut to minimum wage to solve the problem.

By 1972 it appeared that even Cray's legendary module design abilities were failing him in the case of the 8600. Reliability was so poor that it was appeared impossible to get a whole machine working. This was not the first time this had happened, on the 6600 project Cray had to start over from scratch, and the 7600 was in production for some time before it started working reliably. In this case Cray decided the current design was a dead-end, and told William Norris (CDC's CEO) that the only way forward was to redesign the machine from scratch. The finances of the company were dangerous, and Norris decided that he couldn't take the risk; Cray would have to continue with the current design.

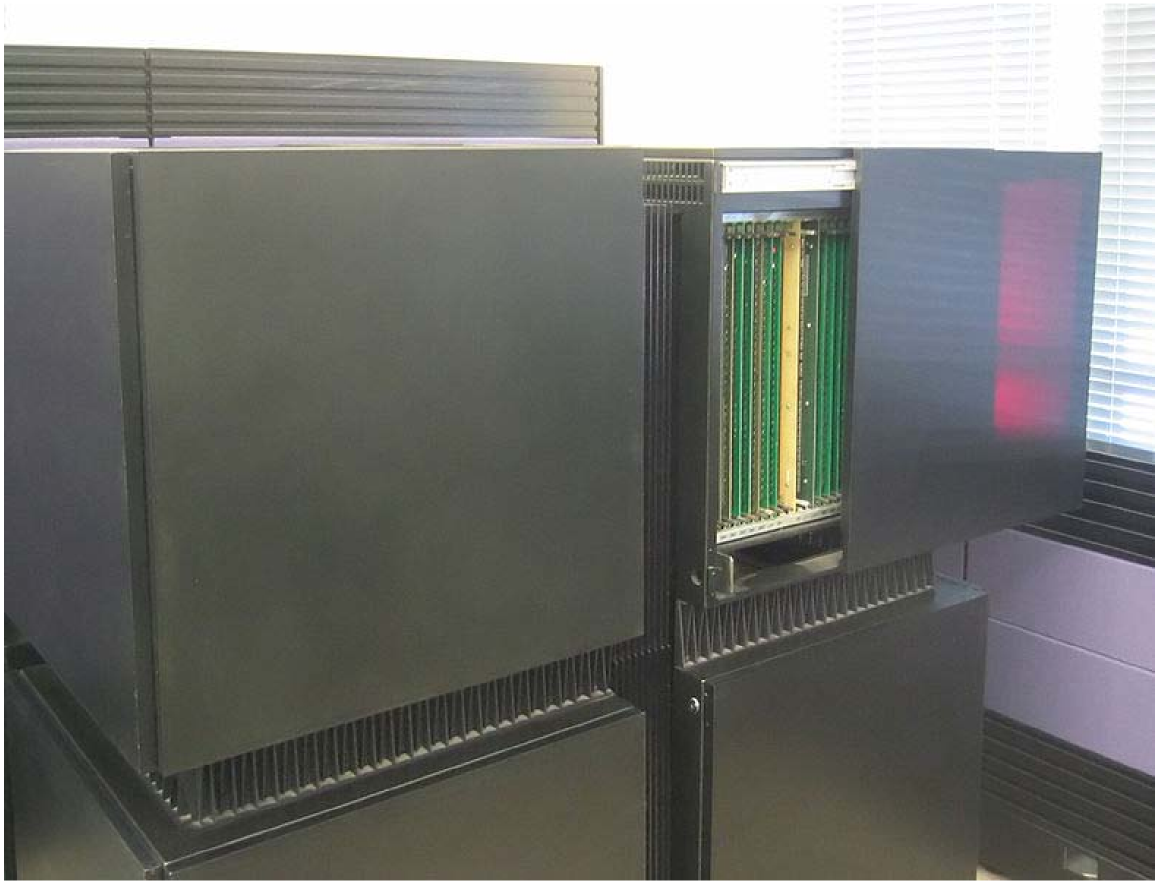
In 1972 Cray decided that he couldn't work under such conditions, and left CDC to form Cray Research. For his new work he abandoned the multiprocessor concept, concerned that software of the era would be unable to take full advantage of the CPUs. He may have come to this conclusion after the ILLIAC IV finally entered operation at about the same time, and proved to have disappointing performance.

Team members convinced Norris that the 8600 could be completed even without Cray, and work continued at the Chippewa Lab. By 1974 the machine still didn't work correctly. Jim Thornton's competing STAR design had reached production quality at this point, and the 8600 project was then cancelled. In service STAR proved to have poor real-world performance, and when the Cray-1 entered the market in 1976, CDC was quickly pushed from the supercomputer market. An effort was made to re-enter the market in the 1980s with the ETA-10, but this ended poorly.

Chapter 16

Connection Machine and Cray MTA

Connection Machine



Thinking Machines CM-1 at the Computer History Museum in Mountain View. One of the face plates has been partially removed to show the circuit boards inside.

The **Connection Machine** was a series of supercomputers that grew out of Danny Hillis' research in the early 1980s at MIT on alternatives to the traditional von Neumann architecture of computation. The Connection Machine was originally intended for applications in artificial intelligence and symbolic processing, but later versions found greater success in the field of computational science.

Basis

Danny Hillis' original thesis paper on which the **CM-1** Connection Machine was based is *The Connection Machine (MIT Press Series in Artificial Intelligence)* (ISBN 0-262-08157-1). The title is out of print as of 2005. The book provides an overview of the philosophy, architecture and software for the Connection Machine, including data routing between CPU nodes, memory handling, Lisp programming for parallel machines, etc.

History

Danny Hillis and Sheryl Handler founded Thinking Machines in Waltham, Massachusetts (it was later moved to Cambridge, Massachusetts) in 1983 and assembled a team to develop the CM-1 Connection Machine. This was a "massively parallel" hypercubic arrangement of thousands of microprocessors, each with its own 4 kbits of RAM, which together executed in a SIMD fashion. The CM-1, depending on the configuration, had as many as 65,536 processors. The individual processors were extremely simple, processing one bit at a time.

The CM-1 and CM-2 took the form of a cube 1.5 meters on a side, divided equally into eight smaller cubes. Each sub-cube contained 16 printed circuit boards and a main processor called a sequencer. Each printed circuit board contained 32 chips. Each chip contained a communication channel called a router, 16 processors, 16 RAMs. The CM-1 as a whole had a hypercubic routing network, a main RAM, and an input/output processor. It was connected to a switching device called a nexus.

In order to improve its commercial viability, the **CM-2**, launched in 1987, added Weitek 3132 floating-point numeric co-processors and more RAM to the system. 32 of the original one-bit processors shared each numeric processor. The CM-2 could be configured with up to 512 MB of RAM, and a RAID hard disk array, called a **DataVault**, of up to 25 GB.

Two later variants of the CM-2 were also produced, the smaller **CM-2a** with either 4096 or 8192 single-bit processors, and the faster **CM-200**.

The light panels of FROSTBURG, a CM-5, on display at the National Cryptologic Museum. The panels were used to check the usage of the processing nodes, and to run diagnostics.

Due to its origins in AI research, the software for the CM-1/2/200 single-bit processor was influenced by the Lisp programming language and a version of Common Lisp, *Lisp (spoken: "Star-Lisp"), was implemented on the CM-1. Other early languages included Karl Sims' IK and Cliff Lasser's URDU. Much system utility software for the CM-1/2 was written in *Lisp.

With the **CM-5**, announced in 1991, Thinking Machines switched from the CM-2's hypercubic architecture of simple processors to an entirely new MIMD architecture based on a fat tree network of SPARC RISC processors. The later **CM-5E** replaced the SPARC processors with faster SuperSPARCs.

Visual Design

Connection Machines were noted for their (intentionally) striking visual design. The CM-1 and CM-2 design teams were led by Tamiko Thiel. The physical form of the CM-1, CM-2, and CM-200 chassis was a cube-of-cubes, referencing the machine's internal 12-dimensional hypercube network, with the red blinking LEDs of the processor status lights visible through the doors of each cube.

The CM-5, in plain view, had a "staircase"-like shape, and also had large panels of red blinking LEDs. Perhaps because of its design, a CM-5 was featured in the movie *Jurassic Park* in the control room for the island (instead of a Cray X-MP supercomputer as in the novel). Prominent sculptor/architect Maya Lin contributed to the CM-5 design.

Cray MTA

The **Cray MTA**, formerly known as the Tera MTA, is a supercomputer architecture based on thousands of independent threads, fine-grain communication and synchronization between threads, and latency tolerance for irregular computations.

Each MTA processor (CPU) has a high-performance ALU with many independent register sets, each running an independent thread. For example, the Cray MTA-2 uses 128 register sets and thus 128 threads per CPU/ALU. All MTAs to date use a barrel processor arrangement, with a thread switch on every cycle, with blocked (stalled) threads skipped to avoid wasting ALU cycles. When a thread performs a memory read, execution blocks until data returns; meanwhile, other threads continue executing. With enough threads (concurrency), there are nearly always runable threads to "cover" for blocked threads, and the ALUs stay busy. The memory system uses full/empty bits to ensure correct ordering. For example, an array `A` is initially written with "empty" bits, and any thread reading a value from `A` blocks until another thread writes a value. This ensures correct ordering, but allows fine-grained interleaving and provides a simple programming model. The memory system is also "randomized", with adjacent physical addresses going to different memory banks. Thus, when two threads access memory simultaneously, they rarely conflict unless they are accessing the same location.

A goal of the MTA is that porting codes from other machines is straightforward, but gives good performance. A parallelizing FORTRAN compiler can produce high performance for some codes with little manual intervention. Where manual porting is required, the simple and fine-grained synchronization model often allows programmers to write code the "obvious" way yet achieve good performance. A further goal is that programs for the MTA will be scalability -- that is, when run on an MTA with twice as many CPUs, the same program will have nearly twice the performance. Both of these are challenges for many other high-performance computer systems.

An uncommon feature of the MTA is several workloads can be interleaved with good performance. Typically, supercomputers are dedicated to a task at a time. The MTA allows idle threads to be allocated to other tasks with very little effect on the main calculations.

Implementations

There have been three MTA implementations and as of 2009 a fourth is planned. The implementations are:

- **MTA-1** The MTA-1 uses a GaAs processor and was installed at the San Diego Supercomputer Center. It used four processors (512 threads)
- **MTA-2** The MTA-2 uses a CMOS processor and was installed at the Naval Research Laboratory. It was reportedly unstable, but being inside a secure facility was not available for debugging or repair.
- **MTA-3** The MTA-3 uses the same CPU as the MTA-2 but a dramatically cheaper and slower network interface. About six Cray XMT systems have been sold (2009) using the MTA-3.
- **MTA-4** The MTA-4 is a planned system (2009) that is architecturally similar but will use limited data caching and a faster network interface than the MTA-3.

Performance

Only a few systems have been deployed, and only MTA-2 benchmarks have been reported widely, making performance comparisons difficult.

Across several benchmarks, a 2-CPU MTA-2 shows performance similar to a 2-processor Cray T90. For the specific application of ray tracing, a 4-CPU MTA-2 was about 5x faster than a 4-CPU Cray T3E, and in scaling from 1 CPU to 4 CPUs the Tera performance improved by 3.8x, while the T3E going from 1 to 4 CPUs improved by only 3.0x.

Architectural Considerations

Another way to compare systems is by inherent overheads and bottlenecks of the design. Some considerations:

- The MTA uses many register sets, thus each register access is slow. Although concurrency (running other threads) typically hides latency, slow register file access limits performance when there are few runable threads. In existing MTA implementations, single-thread performance is 21 cycles per instruction, so performance suffers when there are fewer than 21 threads per CPU.

- The MTA-1, -2, and -3 use no data caches. This reduces CPU complexity and avoids cache coherency problems. However, no data caching introduces two performance problems. First, the memory system must support the full data access bandwidth of all threads, even for unshared and thus cacheable data. Thus, good system performance requires very high memory bandwidth. Second, memory references take 150-170 cycles, a much higher latency than even a slow cache, thus increasing the number of runable threads required to keep the ALU busy. The MTA-4 will have a non-coherent cache, which can be used for read-only and unshared data (such as non-shared stack frames), but which requires software coherency e.g., if a thread is migrated between CPUs. Data cache competition is often a performance bottleneck for highly-concurrent processors, and sometimes even for 2-core systems; however, by using the cache for data that is either highly shared or has very high locality (stack frames), competition between threads can be kept low.

- Full/empty status changes use polling, with a timeout for threads that poll too long. A timed-out thread may be descheduled and the hardware context used to run another thread; the OS scheduler sets a "trap on write" bit so the waited-for write will trap and put the descheduled thread back in the run queue. Where the descheduled thread is on the critical path, performance may suffer substantially.

- The MTA is latency-tolerant, including irregular latency, giving good performance on irregular computations if there is enough concurrency to "cover" delays. The latency-tolerance hardware may be wasted on regular calculations, including those with latency that is high but which can be scheduled easily.