Computer System, Organisation an Architecture

Alvin Sampson

COMPUTER SYSTEM, ORGANISATION AN ARCHITECTURE

COMPUTER SYSTEM, ORGANISATION AN ARCHITECTURE

Alvin Sampson

BIBLIOTEX
Digital Library

Computer System, Organisation an Architecture
by Alvin Sampson

www.bibliotex.com

Ebook ISBN: 9781984664334

Published by:
Bibliotex
Canada
Website: www.bibliotex.com

Contents

1

Working of Operating System

When you turn on your computer, it's nice to think that you're in control. There's the trusty computer mouse, which you can move anywhere on the screen, summoning up your music library or Internet browser at the slightest whim. Although it's easy to feel like a director in front of your desktop or laptop, there's a lot going on inside, and the real man behind the curtain handling the necessary tasks is the operating system.

Most desktop or laptop PCs come pre-loaded with Microsoft Windows. Macintosh computers come pre-loaded with Mac OS X. Many corporate servers use the Linux or UNIX operating systems. The operating system (OS) is the first thing loaded onto the computer — without the operating system, a computer is useless.

More recently, operating systems have started to pop up in smaller computers as well. If you like to tinker with electronic devices, you're probably pleased that operating

systems can now be found on many of the devices we use every day, from cell phones to wireless access points.

The computers used in these little devices have gotten so powerful that they can now actually run an operating system and applications. The computer in a typical modern cell phone is now more powerful than a desktop computer from 20 years ago, so this progression makes sense and is a natural development.

The purpose of an operating system is to organize and control hardware and software so that the device it lives in behaves in a flexible but predictable way. In this article, we'll tell you what a piece of software must do to be called an operating system, show you how the operating system in your desktop computer works and give you some examples of how to take control of the other operating systems around you.

Not all computers have operating systems. The computer that controls the microwave oven in your kitchen, for example, doesn't need an operating system. It has one set of tasks to perform, very straightforward input to expect (a numbered keypad and a few pre-set buttons) and simple, never-changing hardware to control. For a computer like this, an operating system would be unnecessary baggage, driving up the development and manufacturing costs significantly and adding complexity where none is required. Instead, the computer in a microwave oven simply runs a single hard-wired program all the time.

For other devices, an operating system creates the ability to:

- Serve a variety of purposes

- Interact with users in more complicated ways
- Keep up with needs that change over time

All desktop computers have operating systems. The most common are the Windows family of operating systems developed by Microsoft, the Macintosh operating systems developed by Apple and the UNIX family of operating systems (which have been developed by a whole history of individuals, corporations and collaborators). There are hundreds of other operating systems available for special-purpose applications, including specializations for mainframes, robotics, manufacturing, real-time control systems and so on.

In any device that has an operating system, there's usually a way to make changes to how the device works. This is far from a happy accident; one of the reasons operating systems are made out of portable code rather than permanent physical circuits is so that they can be changed or modified without having to scrap the whole device. For a desktop computer user, this means you can add a new security update, system patch, new application or even an entirely new operating system rather than junk your computer and start again with a new one when you need to make a change. As long as you understand how an operating system works and how to get at it, in many cases you can change some of the ways it behaves. The same thing goes for your phone, too.

FUNCTIONS OF OPERATING SYSTEM

The operating system provides several other functions which includes:

- System tools (programs) used to monitor computer performance, debug problems, or maintain parts of the system.

- A set of libraries or functions which programs may use to perform specific tasks especially relating to interfacing with computer system components.

The operating system makes these interfacing functions along with its other functions operate smoothly and these functions are mostly transparent to the user.

OBJECTIVES OF OPERATING SYSTEMS

Modern Operating systems generally have following three major goals. Operating systems generally accomplish these goals by running processes in low privilege and providing service calls that invoke the operating system kernel in high-privilege state.

TO HIDE DETAILS OF HARDWARE BY CREATING ABSTRACTION

An abstraction is software that hides lower level details and provides a set of higher-level functions. An operating system transforms the physical world of devices, instructions, memory, and time into virtual world that is the result of abstractions built by the operating system. There are several reasons for abstraction.

First, the code needed to control peripheral devices is not standardized. Operating systems provide subroutines called device drivers that perform operations on behalf of programs for example, input/output operations.

Second, the operating system introduces new functions as it abstracts the hardware. For instance, operating system introduces the file abstraction so that programs do not have to deal with disks.

Third, the operating system transforms the computer hardware into multiple virtual computers, each belonging to a different programme. Each programme that is running is called a process. Each process views the hardware through the lens of abstraction.

Fourth, the operating system can enforce security through abstraction.

TO ALLOCATE RESOURCES TO PROCESSES (MANAGE RESOURCES)

An operating system controls how processes (the active agents) may access resources (passive entities).

PROVIDE A PLEASANT AND EFFECTIVE USER INTERFACE

The user interacts with the operating systems through the user interface and usually interested in the "look and feel" of the operating system. The most important components of the user interface are the command interpreter, the file system, on-line help, and application integration. The recent trend has been toward increasingly integrated graphical user interfaces that encompass the activities of multiple processes on networks of computers.

One can view Operating Systems from two points of views: Resource manager and Extended machines. Form Resource manager point of view Operating Systems manage the different parts of the system efficiently and from extended machines point of view Operating Systems provide a virtual machine to users that is more convenient to use. The structurally Operating Systems can be design as a monolithic system, a hierarchy of layers, a virtual machine system, an

exokernel, or using the client-server model. The basic concepts of Operating Systems are processes, memory management, I/O management, the file systems, and security.

HISTORY

Historically operating systems have been tightly related to the computer architecture, it is good idea to study the history of operating systems from the architecture of the computers on which they run. Operating systems have evolved through a number of distinct phases or generations which corresponds roughly to the decades.

THE 1940'S - FIRST GENERATIONS

The earliest electronic digital computers had no operating systems. Machines of the time were so primitive that programs were often entered one bit at time on rows of mechanical switches (plug boards). Programming languages were unknown (not even assembly languages). Operating systems were unheard of.

THE 1950'S - SECOND GENERATION

By the early 1950's, the routine had improved somewhat with the introduction of punch cards. The General Motors Research Laboratories implemented the first operating systems in early 1950's for their IBM 701. The system of the 50's generally ran one job at a time. These were called single-stream batch processing systems because programs and data were submitted in groups or batches.

THE 1960'S - THIRD GENERATION

The systems of the 1960's were also batch processing systems, but they were able to take better advantage of the computer's resources by running several jobs at once. So operating systems designers developed the concept of multiprogramming in which several jobs are in main memory at once; a processor is switched from job to job as needed to keep several jobs advancing while keeping the peripheral devices in use.

For example, on the system with no multiprogramming, when the current job paused to wait for other I/O operation to complete, the CPU simply sat idle until the I/O finished. The solution for this problem that evolved was to partition memory into several pieces, with a different job in each partition. While one job was waiting for I/O to complete, another job could be using the CPU.

Another major feature in third-generation operating system was the technique called spooling (simultaneous peripheral operations on line). In spooling, a high-speed device like a disk interposed between a running programme and a low-speed device involved with the programme in input/output. Instead of writing directly to a printer, for example, outputs are written to the disk. Programs can run to completion faster, and other programs can be initiated sooner when the printer becomes available, the outputs may be printed.

Note that spooling technique is much like thread being spun to a spool so that it may be later be unwound as needed.

Another feature present in this generation was time-sharing technique, a variant of multiprogramming technique, in which each user has an on-line (i.e., directly connected)

terminal. Because the user is present and interacting with the computer, the computer system must respond quickly to user requests, otherwise user productivity could suffer. Timesharing systems were developed to multiprogram large number of simultaneous interactive users.

FOURTH GENERATION

With the development of LSI (Large Scale Integration) circuits, chips, operating system entered in the system entered in the personal computer and the workstation age. Microprocessor technology evolved to the point that it become possible to build desktop computers as powerful as the mainframes of the 1970s. Two operating systems have dominated the personal computer scene: MS-DOS, written by Microsoft, Inc. for the IBM PC and other machines using the Intel 8088 CPU and its successors, and UNIX, which is dominant on the large personal computers using the Motorola 6899 CPU family.

STRUCTURE OF OPERATING SYSTEMS

As modern operating systems are large and complex careful engineering is required. There are four different structures that have shown in this document in order to get some idea of the spectrum of possibilities. These are by no mean s exhaustive, but they give an idea of some designs that have been tried in practice.

MONOLITHIC SYSTEMS

This approach well known as “The Big Mess”. The structure is that there is no structure. The operating system is written

as a collection of procedures, each of which can call any of the other ones whenever it needs to. When this technique is used, each procedure in the system has a well-defined interface in terms of parameters and results, and each one is free to call any other one, if the latter provides some useful computation that the former needs.

For constructing the actual object programme of the operating system when this approach is used, one compiles all the individual procedures, or files containing the procedures, and then binds them all together into a single object file with the linker. In terms of information hiding, there is essentially none- every procedure is visible to every other one i.e. opposed to a structure containing modules or packages, in which much of the information is local to module, and only officially designated entry points can be called from outside the module.

However, even in Monolithic systems, it is possible to have at least a little structure. The services like system calls provide by the operating system are requested by putting the parameters in well-defined places, such as in registers or on the stack, and then executing a special trap instruction known as a kernel call or supervisor call.

LAYERED SYSTEM

A generalization of the approach as shown below in the figure for organizing the operating system as a hierarchy of layers, each one constructed upon the one below it. The system had 6 layers.

Layer 0 dealt with allocation of the processor, switching between processes when interrupts occurred or timers

expired. Above layer 0, the system consisted of sequential processes, each of which could be programmed without having to worry about the fact that multiple processes were running on a single processor.

In other words, layer 0 provided the basic multiprogramming of the CPU:

Layer 1: Did the memory management. It allocated space for processes in main memory and on a 512k word drum used for holding parts of processes (pages)for which there was no room in main memory. Above layer 1, processes did not have to worry about whether they were in memory or on the drum; the layer 1 software took care of making sure pages were brought into memory whenever they were needed.

Layer 2: Handled communication between each process and the operator console. Above this layer each process effectively had its own operator console. Layer 3 took care of managing the I/O devices and buffering the information streams to and from them. Above layer 3 each process could deal with abstract I/O devices with nice properties, instead of real devices with many peculiarities. Layer 4 was where the user programs were found. They did not have to worry about process, memory, console, or I/O management. The system operator process was located I layer 5.

VIRTUAL MACHINES

The heart of the system, known as the virtual machine monitor, runs on the bare hardware and does the multiprogramming, providing not one, but several virtual machines to the next layer up.

However, unlike all other operating systems, these virtual machines are not extended machines, with files and other nice features. Instead, they are exact copies of the bare hardware, including kernel/user mod, I/O, interrupts, and everything else the real machine has.

For reason of Each virtual machine is identical to the true hardware, each one can run any operating system that will run directly on the hard ware. Different virtual machines can, and usually do, run different operating systems. Some run one of the descendants of OF/360 for batch processing, while other ones run a single-user, interactive system called CMS (conversational Monitor System) fro timesharing users.

CLIENT-SERVER MODEL

A trend in modern operating systems is to take this idea of moving code up into higher layers even further, and remove as much as possible from the operating system, leaving a minimal kernel. The usual approach is to implement most of the operating system functions in user processes. To request a service, such as reading a block of a file, a user process (presently known as the client process) sends the request to a server process, which then does the work and sends back the answer.

In client-Server Model, all the kernel does is handle the communication between clients and servers. By splitting the

operating system up into parts, each of which only handles one fact of the system, such as file service, process service,

Terminal service, or memory service, each part becomes small and manageable; furthermore, because all the servers run as user-mode processes, and not in kernel mode, they do not have direct access to the hardware. As a consequence, if a bug in the file server is triggered, the file service may crash, but this will not usually bring the whole machine down.

Another advantage of the client-server model is its adaptability to use in distributed system. If a client communicates with a server by sending it messages, the client need not know whether the message is handled locally in its own machine, or whether it was sent across a network to a server on a remote machine. As far as the client is concerned, the same thing happens in both cases: a request was sent and a reply came back.

KINDS OF OPERATING SYSTEM

REAL-TIME OPERATING SYSTEM

It is a multitasking operating system that aims at executing real-time applications. Real-time operating systems often use specialized scheduling algorithms so that they can achieve a deterministic nature of behaviour. The main object of real-time operating systems is their quick and predictable response to events. They either have an event-driven or a time-sharing design. An event-driven system switches between tasks based of their priorities while time-sharing operating systems switch tasks based on clock interrupts.

MULTI-USER AND SINGLE-USER OPERATING SYSTEMS

The operating systems of this type allow a multiple users to access a computer system concurrently. Time-sharing system can be classified as multi-user systems as they enable a multiple user access to a computer through the sharing of time. Single-user operating systems, as opposed to a multi-user operating system, are usable by a single user at a time. Being able to have multiple accounts on a Windows operating system does not make it a multi-user system. Rather, only the network administrator is the real user. But for a Unix-like operating system, it is possible for two users to login at a time and this capability of the OS makes it a multi-user operating system.

MULTI-TASKING AND SINGLE-TASKING OPERATING SYSTEMS

When a single programme is allowed to run at a time, the system is grouped under a single-tasking system, while in case the operating system allows the execution of multiple tasks at one time, it is classified as a multi-tasking operating system. Multi-tasking can be of two types namely, pre-emptive or co-operative.

In pre-emptive multitasking, the operating system slices the CPU time and dedicates one slot to each of the programs. Unix-like operating systems such as Solaris and Linux support pre-emptive multitasking. Cooperative multitasking is achieved by relying on each process to give time to the other processes in a defined manner. MS Windows prior to Windows 95 used to support cooperative multitasking.

DISTRIBUTED OPERATING SYSTEM

An operating system that manages a group of independent computers and makes them appear to be a single computer is known as a distributed operating system. The development of networked computers that could be linked and communicate with each other, gave rise to distributed computing. Distributed computations are carried out on more than one machine. When computers in a group work in cooperation, they make a distributed system.

EMBEDDED SYSTEM

The operating systems designed for being used in embedded computer systems are known as embedded operating systems. They are designed to operate on small machines like PDAs with less autonomy. They are able to operate with a limited number of resources. They are very compact and extremely efficient by design. Windows CE, FreeBSD and Minix 3 are some examples of embedded operating systems. The most common is the Microsoft suite of operating systems.

They include from most recent to the oldest:

- *Windows XP Professional Edition*: A version used by many businesses on workstations. It has the ability to become a member of a corporate domain.
- *Windows XP Home Edition*: A lower cost version of Windows XP which is for home use only and should not be used at a business.
- *Windows* 2000: A better version of the Windows NT operating system which works well both at home and as a workstation at a business. It includes technologies which allow hardware to be automatically detected and other enhancements over Windows NT.

- *Windows ME*: A upgraded version from windows 98 but it has been historically plagued with programming errors which may be frustrating for home users.
- *Windows* 98: This was produced in two main versions. The first Windows 98 version was plagued with programming errors but the Windows 98 Second Edition which came out later was much better with many errors resolved.
- *Windows NT*: A version of Windows made specifically for businesses offering better control over workstation capabilities to help network administrators.
- *Windows* 95: The first version of Windows after the older Windows 3.x versions offering a better interface and better library functions for programs.

TYPES OF OPERATING SYSTEM

Microsoft Windows isn't the only operating system for personal computers, or even the best... it's just the best-distributed.

Its inconsistent behaviour and an interface that changes with every version are the main reasons people find computers difficult to use. Microsoft adds new bells and whistles in each release, and claims that this time they've solved the countless problems in the previous versions... but

the hype is never really fulfilled. Windows 7 offers little new: it's basically Vista without quite so many mistakes built into it. The upgrade prices serve primarily to keep the cash flowing to Microsoft, to subsidize their efforts to take over other markets. A slew of intrusive "features" in the recent versions benefit Microsoft at the expensive of both your privacy and your freedom. Switching to Windows Vista or Windows 7 requires buying new hardware and learning a new system, so instead consider switching to something *better*. More than 1 in 10 people on the web already have. There's an exciting array of interesting operating systems out there, and the overall quality of them is stronger than ever.

≅ ç If you can't say "no" to Windows (which is understandable in many cases), you can still say "no more". The simplest alternative to Windows Vista/7 is a previously-installed version of Windows.

Windows Vista/7 isn't a simple upgrade; it's a drastically different operating system, which may not even run your existing software, or work properly on hardware just a few years old, so installing the "upgrade" is a risk. Even if the package says "for Windows 7", that's mostly a Microsoft-directed marketing ploy; check the fine print to see if earlier versions of Windows are also supported (XP usually is).

The bottom line: if you already own Windows 98/2000/XP

and it works for you, you don't have to upgrade; you can continue using it without paying Microsoft another dime.

≅ ★ ⊘The Mac OS user interface inspired the creation of Windows, and is still the target Microsoft is trying to equal. As a popular consumer product, there's plenty of software available for it, and it's moving beyond its traditional niches of graphic design, education, and home use, into general business use (after all, Apple Corp. runs on it).

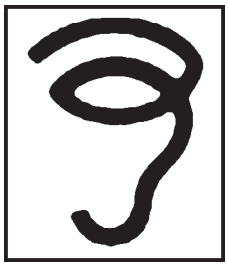
OS X (ten), uses Unix technology, which makes it more stable and secure than Windows. But the real star is OS X's visual interface, which shows the difference between Microsoft's guesswork in this area and Apple's innovative *design work*: it's both beautiful and easy to use.

e

ç⊘Linux ("LIH-nux") is a free Unix-like operating system, originally developed by programmers who who simply love the challenge of solving problems and producing quality software... even if that means giving the resulting product away.

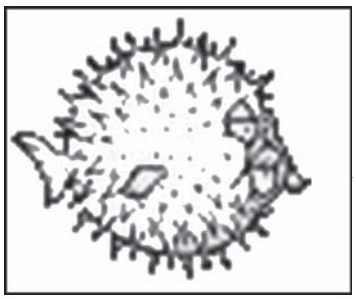
The main "negative" to Mac OS is that you need to buy an Apple computer to use it, but that's not much of a sacrifice: in addition to being stylish, they're top-notch in quality, and both faster and less expensive than you might expect. Apple has a section of their site for people wondering if they can switch to Mac OS. Not coincidentally, there's also a wealth of free software for it. Unlike proprietary operating systems, which are usually controlled in every detail by a single company, Linux has a standard consistent core (called the "kernel") around which many varieties (known as "distributions") have been produced by various companies and organisations. Some are aimed at geeks, some focus on the needs of business users, and some are designed with typical home users in mind. It has become a popular option for the makers of inexpensive "netbooks" and laptops to preinstall. You can test-drive Linux with versions such as Knoppix which runs directly from a CD without affecting the OS on your hard drive. Most individuals should start with one of the mainstream distributions, such as Ubuntu, Mandriva, or Linspire. Businesses might prefer RedHat/ Fedora, Novell/SUSE, or CentOS. Geeks should check out Debian, Slackware, and Gentoo. Linux is a first-rate choice for servers; this site is a Linux system

ç⊘ Google's Chrome OS is still vaporware so far, and it's arguably just another flavour of Linux, but it promises to be a viable alternative to Windows on small portable "netbooks" which will come with it preinstalled. The user interface is going to be based on Google's web browser of the same name, and take advantage of technology to make online apps like Google Docs work even if you're not online. It is open-source software, so netbook manufacturers won't have to pay to use it.



≅★ ç BeOS was designed with multimedia in mind, including the kinds of features that Microsoft is just recently tacking onto Windows. Although Microsoft successfully drove Be Corp. out of business through illegal interference with their marketing efforts, reports of BeOS's death are exaggerated: The source code for BeOS has been licensed to a European software firm whose Zeta is effectively the much-longed-for BeOS R6. The free BeOS R5 Personal Edition is still available to download, and has been packaged with all the latest drivers and free add-ons as BeOS Max Edition. And the Haiku project is creating an open duplicate of BeOS R5, which will then be enhanced

★ ç⃠ FreeBSD is commonly called “the free Unix”. It’s descended from the classic 1970’s Berkeley Software Distribution of Unix (from before the OS became “UNIX”®), making it one of the most mature and stable operating systems around. It’s “free” as in “free beer” (you can download it for nothing) and as in “free speech” (you can do pretty much whatever you like with it... like when Microsoft took code from it to add better networking to Windows NT). Unlike the plethora of Linuxes, there’s only one current version of FreeBSD, with a consistent structure and an easy-to-use “ports” system for installing software. It can also run most Linux binaries. Much of the Internet infrastructure was built on FreeBSD, due to its combination of quality and cost.



★

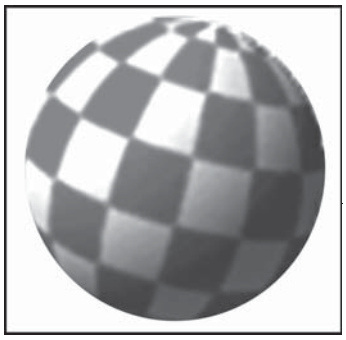
ç⃠ OpenBSD is “the other free Unix”. It’s similar to FreeBSD both in the Berkeley code it’s based on, and the licensing terms. One key advantage it has over its BSD siblings (and nearly any other OS) is that it’s incredibly secure from attack, as implied by its blowfish mascot, and made explicit by their boast of only one remotely-exploitable hole - ever - in their default installation. (Compare that to Windows’ hundreds.) “Open” is a reference to their code auditing process, not a welcome-mat forcrackers. It’s not as speedy as FreeBSD, but it’s safer. It’s also available for some hardware plaforms FreeBSD doesn’t support, including Mac 68K, PPC.

ç⊘NetBSD is "the *other* other free Unix". It's the work of another group of volunteer developers using the net to collaborate (hence the name of their product). Their mission is to get the OS to run - and run *well* - on hardware platforms no other Unix supports. In addition to most of the usual suspects above, it's been ported to run on the NeXT box, MIPSmachines, the good Atari computers, the BeBox, WinCE-compatible handhelds, ARM processors, and even game machines like the Playstation 2 or the orphaned Sega Dreamcast.

ç⊘Darwin is a cousin of Free/Open/NetBSD, and the free foundation on which the commercial Mac OS X is built. Although its development was originally managed rather tightly by Apple (understandable, because their business depends on it) they've loosened the leash, making participation in the development more open. Darwin is

making progress toward becoming an open-source OS in its own right. Any Darwin software will run on OS X, but software written specifically for OS X won't run on Darwin, because the Mac interface (and various other proprietary bits) are not part of Darwin itself. Instead, Darwin typically uses X11 with either TWM or KDE.

ç Syllable is a free alternative OS for standard PCs. It uses some of the better ideas from Unix, BeOS, AmigaOS, and others, and is compatible enough with portable software written for Unix that many have already been ported over to it. It's not a full-featured OS yet, but it's functional enough to be used with built-in web and e-mail clients, and media players.

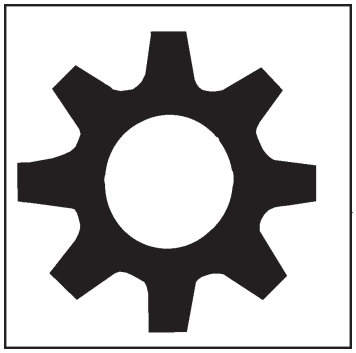


★

Amiga owners used to taunt PC and Mac users with their smoothly-multitasking graphical operating system, back when the Macs couldn't multitask, and PCs weren't even graphical. Even though the "classic" Amiga machines are no longer being produced, there's been a lot of activity in Amigaspace in the meantime: The OS has been updated to

support current technology with Amiga OS 4, emulation layers called AmigaOS XL and AMIthlon were created to run Amiga OS on modern PC hardware, Amiga Forever is an emulator for Windows and other operating systems, and a new hardware platform and OS called AmigaOne have been introduced to try to carry on the Amiga legacy.

MorphOS began as a project to port the Amiga OS to the then-new PowerPC architecture, but has since morphed into an OS in its own right. It runs on certain PowerPC/G3/G4-based systems, and has better-than-standard-emulation support for Amiga OS 3.1 applications as well as native apps built for MorphOS.



RISC OS is the operating system of the former Acorn line of computers (best known in the UK), which has been revived and updated for faster performance and to meet current OS standards (e.g. long filenames, large hard drives). It doesn't

run on standard PCs, but on systems specifically designed for it (such as the RiscPC and A7000), using the high-speed StrongARM processors. The OS itself is stored in electronic ROM rather than having to be loaded into RAM from a hard drive

ç⊘ GNU's Not Unix. In fact, that phrase is what G.N.U. is a (recursive) abbrevation for. It is a Unix-like operating system being developed as a long-term project by the Free Software Foundation to offer a fully-free alternative to the commercial and BSD versions of Unix. Although you'll find many key components of GNU used in Linux and BSD packages under the GNU General Public license (GPL), a fully GNU system will use the Hurd, GNU's own free-software kernel. The Hurd has some design advantages over the Linux kernel, but is still far from finished, and requires serious expertise with OS development to install.

ç⊘Minix is an open-source Unix-like operating system originally developed for educational purposes. Because of its relative simplicity and ample documentation, its creator says that a few months studying the source code should teach you most of how such things work. (It inspired Linus Torvalds to create Linux.) Versions 1 and 2 serve primarily as teaching examples, but version 3 has also become useful in its own right, intended for highly reliable uses on low-end 386-level hardware.

⊘There are also a bunch of commercial UNIX systems, which are typically customised to run on expensive, high-end, proprietary hardware sold by the same vendor. Most of them have names other than "Unix" due to old trademark issues. They're better as alternatives to the server versions of Windows, not the desktop versions of Windows such as 98/XP/Vista. They include Sun Solaris, HP-UX, IBM AIX, SGI IRIX, and Mac OS X Server

≅ ★IBM's OS/2warp was once supposed to replace MS Windows, back when Emperor IBM and Darth Microsoft were

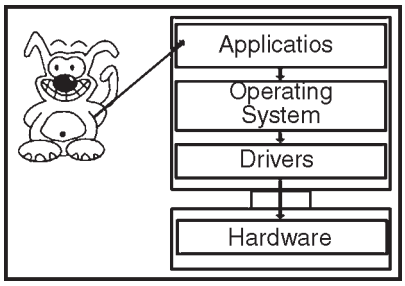
planning to rule the galaxy together. Then Darth decided he didn't need the Emperor, struck confidential deals with other hardware vendors and software developers, and made Windows (just barely) powerful enough to fill OS/2's intended role. Windows didn't really beat OS/2 technically, but it won the Marketing Wars, which is what mattered. Unfortunately, IBM has given up on OS/2's future. A third-party package called eComStation is a licensed effort to update and maintain OS/2.

ç Believe it or not, DOS (with or without Windows 3.1) is still a viable option for many uses. There was an incredible amount of software developed for it, and it still works. Plus, DOS runs like a champ, on old hardware that no one else wants. You can even fit it on a diskette, to boot it on nearly any PC anywhere.

DESCRIPTION OF OPERATING SYSTEM

For a computer to be able to operate a computer programme (sometimes known as application or software), the machine must be able to perform a certain number of preparatory operations to ensure exchange between the processor, the memory and the physical resources (peripherals).

The operating system (sometimes referred to by its abbreviation OS), is responsible for creating the link between the material resources, the user and the applications (word processor, video game, etc.). When a programme wants to access a material resource, it does not need to send specific information to the peripheral device but it simply sends the information to the operating system, which conveys it to the relevant peripheral via its driver. If there are no drivers, each programme has to recognise and take into account the communication with each type of peripheral!



The operating system thus allows the "dissociation" of programmes and hardware, mainly to simplify resource management and offer the user a simplified Man-machine interface (MMI) to overcome the complexity of the actual machine.

ROLES OF OPERATING SYSTEM

The operating system has various roles:

- *Management of the processor*: The operating system is responsible for managing allocation of the processor between the different programmes using ascheduling algorithm. The type of scheduler is totally dependent on the operating system, according to the desired objective.

- *Management of the random access memory*: The operating system is responsible for managing the memory space allocated to each application and, where relevant, to each user. If there is insufficient physical memory, the operating system can create a memory zone on the hard drive, known as "virtual memory". The virtual memory lets you run applications requiring more memory than there is available RAM on the system. However, this memory is a great deal slower.
- *Management of input/output*: The operating system allows unification and control of access of programmes to material resources via drivers (also known as peripheral administrators or input/output administrators).
- *Management of execution of applications*: The operating system is responsible for smooth execution of applications by allocating the resources required for them to operate. This means an application that is not responding correctly can be "killed".

COMPONENTS OF OPERATING SYSTEM

The operating system comprises a set of software packages that can be used to manage interactions with the hardware. The following elements are generally included in this set of software:

- The kernel, which represents the operating system's basic functions such as management of memory, processes, files, main inputs/outputs and communication functionalities.

- The shell, allowing communication with the operating system via a control language, letting the user control the peripherals without knowing the characteristics of the hardware used, management of physical addresses, etc.
- The file system, allowing files to be recorded in a tree structure.

OPERATING SYSTEMS SERVICES

Following are the five services provided by an operating systems to the convenience of the users.

PROGRAM EXECUTION

The purpose of a computer systems is to allow the user to execute programs.

So the operating systems provides an environment where the user can conveniently run programs. The user does not have to worry about the memory allocation or multitasking or anything. These things are taken care of by the operating systems.

Running a program involves the allocating and deallocating memory, CPU scheduling in case of multiprocess. These functions cannot be given to the user-level programs. So user-level programs cannot help the user to run programs independently without the help from operating systems.

I/O OPERATIONS

Each program requires an input and produces output. This involves the use of I/O. The operating systems hides the user the details of underlying hardware for the I/O. All the

user sees is that the I/O has been performed without any details. So the operating systems by providing I/O makes it convenient for the users to run programs.

For efficiently and protection users cannot control I/O so this service cannot be provided by user-level programs.

FILE SYSTEM MANIPULATION

The output of a program may need to be written into new files or input taken from some files. The operating systems provides this service. The user does not have to worry about secondary storage management. User gives a command for reading or writing to a file and sees his her task accomplished. Thus operating systems makes it easier for user programs to accomplished their task.

This service involves secondary storage management. The speed of I/O that depends on secondary storage management is critical to the speed of many programs and hence I think it is best relegated to the operating systems to manage it than giving individual users the control of it. It is not difficult for the user-level programs to provide these services but for above mentioned reasons it is best if this service s left with operating system.

COMMUNICATIONS

There are instances where processes need to communicate with each other to exchange information. It may be between processes running on the same computer or running on the different computers.

By providing this service the operating system relieves the user of the worry of passing messages between processes. In case where the messages need to be passed to processes on

the other computers through a network it can be done by the user programs. The user program may be customized to the specifics of the hardware through which the message transits and provides the service interface to the operating system.

ERROR DETECTION

An error is one part of the system may cause malfunctioning of the complete system. To avoid such a situation the operating system constantly monitors the system for detecting the errors. This relieves the user of the worry of errors propagating to various part of the system and causing malfunctioning.

This service cannot allowed to be handled by user programs because it involves monitoring and in cases altering area of memory or deallocation of memory for a faulty process. Or may be relinquishing the CPU of a process that goes into an infinite loop. These tasks are too critical to be handed over to the user programs. A user program if given these privileges can interfere with the correct (normal) operation of the operating systems.

SYSTEM CALLS AND SYSTEM PROGRAMS

System calls provide an interface between the process an the operating system. System calls allow user-level processes to request some services from the operating system which process itself is not allowed to do.

In handling the trap, the operating system will enter in the kernel mode, where it has access to privileged instructions, and can perform the desired service on the behalf of user-level process. It is because of the critical nature

of operations that the operating system itself does them every time they are needed. For example, for I/O a process involves a system call telling the operating system to read or write particular area and this request is satisfied by the operating system.

System programs provide basic functioning to users so that they do not need to write their own environment for program development (editors, compilers) and program execution (shells). In some sense, they are bundles of useful system calls.

LAYERED APPROACH DESIGN

In this case the system is easier to debug and modify, because changes affect only limited portions of the code, and programmer does not have to know the details of the other layers. Information is also kept only where it is needed and is accessible only in certain ways, so bugs affecting that data are limited to a specific module or layer.

MECHANISMS AND POLICIES

The policies what is to be done while the mechanism specifies how it is to be done. For instance, the timer construct for ensuring CPU protection is mechanism. On the other hand, the decision of how long the timer is set for a particular user is a policy decision.

The separation of mechanism and policy is important to provide flexibility to a system. If the interface between mechanism and policy is well defined, the change of policy may affect only a few parameters.

On the other hand, if interface between these two is vague or not well defined, it might involve much deeper change to

the system. Once the policy has been decided it gives the programmer the choice of using his/her own implementation. Also, the underlying implementation may be changed for a more efficient one without much trouble if the mechanism and policy are well defined.

Specifically, separating these two provides flexibility in a variety of ways. First, the same mechanism can be used to implement a variety of policies, so changing the policy might not require the development of a new mechanism, but just a change in parameters for that mechanism, but just a change in parameters for that mechanism from a library of mechanisms. Second, the mechanism can be changed for example, to increase its efficiency or to move to a new platform, without changing the overall policy.

FEATURES OF WINDOWS OPERATING SYSTEM

WINDOWS CE

This operating system is designed for embedded devices such as Personal Digital Assistants (PDA's) and desktop boxes such as Web-TV.

General Features of Windows CE:

- Requirements specified by OEM
- Major use is in PDA's and embedded systems

WINDOWS 95

This operating system is designed for use as a workstation client or desktop system, primarily for the home or mobile user. It is not intended to be used a server, but can be used in simple workgroups to share resources such as printers and files.

GENERAL FEATURES OF WINDOWS 95

- Easier to use and learn than Windows 3.1
- More reliable than Windows 3.1
- Supports all major networking protocols including Novell IPX and TCP/IP
- Network clients are faster, more reliable, and use no conventional memory
- Simplified user interface
- Automated installation for all users and custom installations
- Remote administration features built in
- Supports multiple users on a single PC with customized settings for each individual
- Pre-emptive multitasking and multi-threading
- Plug and play support for hardware devices
- Dial-up networking (remote access services)
- Supports existing MS-DOS and Windows drivers and programs

WINDOWS 98

This operating system is designed for use as a workstation client or desktop system, primarily for the home or remote user. It is not intended to be used a server, but can be used in simple workgroups to share resources such as printers and files. It is an upgraded enhanced version of Windows 95.

General Features of Windows 98:

- FAT32 enhanced file system
- Performance enhancements
- Fast startup and shutdown
- Intelligent update
- Wizards
- System file checker
- New hardware support, Universal Serial Bus
- Integrated browser/shell

WINDOWS NT V4 WORKSTATION

This operating system is designed for serious power users and desktop workstations, where users demand high reliability, pre-emptive multitasking of programs and support for OpenGL graphics applications. It can be used as a server in a workgroup, where the number of clients it supports is 10 or less.

GENERAL FEATURES OF WINDOWS NT V4 WORKSTATION

- Complete crash protection for 16- and 32-bit applications

- Built-in data protection
- Supports common networks and protocols
- Remote access service [client and/or server]
- Support for applications designed for MS-DOS®, Windows®, Windows 95, and other operating systems
- Preemptive multitasking
- OpenGL 3-D graphics
- Supports a wide range hardware devices
- Scalable [support for more than one processor]
- Multi-platform [support more than one processor type, eg, RISC]

WINDOWS NT 4 SERVER

This operating system is designed for robust scalable networks based on domains. It provides accounts for users and security logon using usernames and passwords.

Where servers are required to handle more than 10 clients, NT Server is the best choice of operating system. It has been especially optimized to give good performance as an application server, and has additional tools to ease network administration problems.

GENERAL FEATURES OF WINDOWS NT SERVER

- Exceptional file and print services
- Support for thousands of client/server applications
- Built-in Security and advanced fault tolerance
- Runs on your choice of scaleable hardware including Intel® x86, Pentium®, Alpha AXP™, MIPS® Rx400, and PowerPC™

- Supports MS-DOS®, Windows®, OS/2®, UNIX and Macintosh®
- Integrates with NetWare®, LAN Manager, UNIX, PATHWORKS™, SNA, and other network systems
- Built-in Migration Tool for NetWare
- File & Print Services for NetWare
- Instantly accessible and up-to-date information
- Hardware auto-detect and CD-ROM-based Express Setup
- Easy-to-use graphical environment
- Directory service that aids in the management and control of network resources
- TCP/IP, Macintosh support, and Remote Access Service at no extra charge

WINDOWS 2000 PROFESSIONAL

This is the replacement for Windows NT 4 workstation, designed for power users, remote users and high performance workstations. It has all the features of Windows NT 4 combined with the graphical desktop interface of Windows 98.

GENERAL FEATURES OF WINDOWS 2000 PROFESSIONAL

- *File protection*: Core files are protected from being overwritten by software applications
- *Microsoft Installer*: This service helps users install, configure, upgrade and remove software
- *System preparation*: Entire computers can be cloned

(copied) allowing multiple deployment of similar configurations to be done quickly and easily

- *Personalized menus*: The Start menu is adapted to the preferences of the user
- *Troubleshooters*: The troubleshooting wizards allow you to configure, optimize and troubleshoot Windows 2000
- *Encryption and security*: Files can be encrypted for greater protection. The standard security model for NT applies to all files, folders and system resources. Kerberos is also supported
- *Additional device support*: Support for USB, smart cards, remote notebooks (such as hot docking), IrDA, IEEE 1394, DVD and plug and plug devices
- *Networking*: Supports peer-to-peer communication with Windows NT and Windows 9x networks. In addition, UNIX services allow interoperability with UNIX networks.

WINDOWS 2000 SERVER

Organizations can use Windows 2000 server to build reliable scalable networks that support organizational requirements, such as file and application servers, Web and Intranet servers, e-mail and remote access servers and print servers.

GENERAL FEATURES OF WINDOWS 2000 SERVER

- *Internet information Services 5.0*: This provides the functionality to easily host and manage web sites. Active Server pages allow the creation of interactive

web-based applications and the media services allow the delivery of multimedia content across the Internet or corporate Intranet.

- *Active directory*: The domain database implemented in Windows NT is expanded to include additional information and provides a repository for software applications to store and retrieve information from.
- *Terminal services*: This allows users to run Windows based applications on a Terminal server, taking advantage of the extra processing power, from a remote PC or Windows-based terminal.
- Remote Access
- Virtual Private Networks
- *Disk quotas*: This allows administrators to set limits on disk space usage per user.

WINDOWS 2000 ADVANCED SERVER

Windows 2000 Advanced Server includes all the features of Windows 2000 Server. It is designed for large-scale networks requiring high reliability and for the provision of e-commerce and line-of-business applications.

GENERAL FEATURES OF WINDOWS 2000 ADVANCED SERVER

Includes all the features of Windows 2000 Server plus:

- Support for up to eight processors
- *Clustering*: Supports both hardware and software failures by mirroring servers
- *Load Balancing*: The load on servers can be dynamically redistributed in the event of server failure

OPERATING SYSTEMS WIDELY USED ON PERSONAL COMPUTERS

The Operating System of a computer must be considered when selecting Assistive Technology to be used to for accessing the computer, or accessing information displayed upon the computer monitor. It can be very frustrating to have purchased an application— and learn it is not compatible with the operating system of your computer.

The operating system of a computer performs basic tasks such as: recognizing information from the keyboard and mouse, sending information to the monitor, storing of information to the hard drive, and controlling device peripherals as printers and flatbed scanners.

Operating systems provide the basis for running common applications such as word processors and Internet browsers. Operating systems are also responsible for running Assistive Technology applications such as screen magnifiers, and applications that read text aloud.

INTERPROCESS COMMUNICATION

Since processes frequently needs to communicate with other processes therefore, there is a need for a well-structured communication, without using interrupts, among processes.

RACE CONDITIONS

In operating systems, processes that are working together share some common storage (main memory, file etc.) that each process can read and write. When two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called race conditions. Concurrently

executing threads that share data need to synchronize their operations and processing in order to avoid race condition on shared data. Only one 'customer' thread at a time should be allowed to examine and update the shared variable.

CRITICAL SECTION

AVOIDING RACE CONDITIONS

The key to preventing trouble involving shared storage is find some way to prohibit more than one process from reading and writing the shared data simultaneously.

That part of the programme where the shared memory is accessed is called the *Critical Section*. To avoid race conditions and flawed results, one must identify codes in *Critical Sections* in each thread.

The characteristic properties of the code that form a Critical Section are:

- Codes that reference one or more variables in a "read-update-write" fashion while any of those variables is possibly being altered by another thread.
- Codes that alter one or more variables that are possibly being referenced in "read-updata-write" fashion by another thread.
- Codes use a data structure while any part of it is possibly being altered by another thread.
- Codes alter any part of a data structure while it is possibly in use by another thread.

MUTUAL EXCLUSION

A way of making sure that if one process is using a shared modifiable data, the other processes will be excluded from

doing the same thing. Formally, while one process executes the shared variable, all other processes desiring to do so at the same time moment should be kept waiting; when that process has finished executing the shared variable, one of the processes waiting; while that process has finished executing the shared variable, one of the processes waiting to do so should be allowed to proceed. In this fashion, each process executing the shared data (variables) excludes all others from doing so simultaneously. This is called Mutual Exclusion.

MUTUAL EXCLUSION CONDITIONS

If we could arrange matters such that no two processes were ever in their critical sections simultaneously, we could avoid race conditions.

We need four conditions to hold to have a good solution for the critical section problem (mutual exclusion):

- No two processes may at the same moment inside their critical sections.
- No assumptions are made about relative speeds of processes or number of CPUs.
- No process should outside its critical section should block other processes.
- No process should wait arbitrary long to enter its critical section.

SEMAPHORES

DEFINITION

A semaphore is a protected variable whose value can be accessed and altered only by the operations P and V and

initialization operation called 'Semaphoiinitislize'. Binary Semaphores can assume only the value 0 or the value 1 counting semaphores also called general semaphores can assume only nonnegative values.

The P (or wait or sleep or down) operation on semaphores S, written as P(S) or wait (S), operates as follows:

- P(S): IF S > 0
- THEN S: = S-1
- ELSE (wait on S)

The V (or signal or wakeup or up) operation on semaphore S, written as V(S) or signal (S), operates as follows:

- V(S): IF (one or more process are waiting on S)
- THEN (let one of these processes proceed)
- ELSE S:= S +1

Operations P and V are done as single, indivisible, atomic action. It is guaranteed that once a semaphore operations has stared, no other process can access the semaphore until operation has completed. Mutual exclusion on the semaphore, S, is enforced within P(S) and V(S). If several processes attempt a P(S) simultaneously, only process will be allowed to proceed. The other processes will be kept waiting, but the implementation of P and V guarantees that processes will not suffer indefinite postponement.

DEADLOCK

A set of process is in a deadlock state if each process in the set is waiting for an event that can be caused by only another process in the set. In other words, each member of the set of deadlock processes is waiting for a resource that

can be released only by a deadlock process. None of the processes can run, none of them can release any resources, and none of them can be awakened. It is important to note that the number of processes and the number and kind of resources possessed and requested are unimportant. The resources may be either physical or logical. Examples of physical resources are Printers, Tape Drivers, Memory Space, and *CPU* Cycles. Examples of logical resources are Files, Semaphores, and Monitors.

PREEMPTABLE AND NONPREEMPTABLE RESOURCES

Resources come in two flavours: preemptable and nonpreemptable. A preemptable resource is one that can be taken away from the process with no ill effects. Memory is an example of a preemptable resource.

On the other hand, a nonpreemptable resource is one that cannot be taken away from process (without causing ill effect). For example, *CD* resources are not preemptable at an arbitrary moment.

Reallocating resources can resolve deadlocks that involve preemptable resources. Deadlocks that involve nonpreemptable resources are difficult to deal with.

NECESSARY AND SUFFICIENT DEADLOCK CONDITIONS

Coffman identified four (4) conditions that must hold simultaneously for there to be a deadlock:

- *Mutual Exclusion Condition*: The resources involved are non-shareable.

Explanation: At least one resource (thread) must be held in a non-shareable mode, that is, only one process at a time claims exclusive control of the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

- *Hold and Wait Condition*: Requesting process hold already, resources while waiting for requested resources.

  Explanation: There must exist a process that is holding a resource already allocated to it while waiting for additional resource that are currently being held by other processes.

- *No-Preemptive Condition*: Resources already allocated to a process cannot be preempted.

  Explanation: Resources cannot be removed from the processes are used to completion or released voluntarily by the process holding it.

- *Circular Wait Condition*: The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list. As an example, consider the traffic deadlock in the following figure

DEALING WITH DEADLOCK PROBLEM

In general, there are four strategies of dealing with deadlock problem:

- *The Ostrich Approach*: Just ignore the deadlock problem altogether.

- *Deadlock Detection and Recovery*: Detect deadlock and, when it occurs, take steps to recover.
- *Deadlock Avoidance*: Avoid deadlock by careful resource scheduling.
- *Deadlock Prevention*: Prevent deadlock by resource scheduling so as to negate at least one of the four conditions.

DEADLOCK DETECTION

Deadlock detection is the process of actually determining that a deadlock exists and identifying the processes and resources involved in the deadlock.

The basic idea is to check allocation against resource availability for all possible allocation sequences to determine if the system is in deadlocked state a. Of course, the deadlock detection algorithm is only half of this strategy. Once a deadlock is detected, there needs to be a way to recover several alternatives exists:

- Temporarily prevent resources from deadlocked processes.
- Back off a process to some check point allowing preemption of a needed resource and restarting the process at the checkpoint later.
- Successively kill processes until the system is deadlock free.

These methods are expensive in the sense that each iteration calls the detection algorithm until the system proves to be deadlock free. The complexity of algorithm is $O(N^2)$ where N is the number of proceeds. Another potential problem is starvation; same process killed repeatedly.

MEMORY MANAGEMENT

CONCEPT

The memory management subsystem is one of the most important parts of the operating system. Since the early days of computing, there has been a need for more memory than exists physically in a system. Strategies have been developed to overcome this limitation and the most successful of these is virtual memory. Virtual memory makes the system appear to have more memory than it actually has by sharing it between competing processes as they need it.

LARGE ADDRESS SPACES

The operating system makes the system appear as if it has a larger amount of memory than it actually has. The virtual memory can be many times larger than the physical memory in the system,

PROTECTION

Each process in the system has its own virtual address space. These virtual address spaces are completely separate from each other and so a process running one application cannot affect another. Also, the hardware virtual memory mechanisms allow areas of memory to be protected against writing. This protects code and data from being overwritten by rogue applications.

MEMORY MAPPING

Memory mapping is used to map image and data files into a processes address space. In memory mapping, the contents of a file are linked directly into the virtual address space of a process.

FAIR PHYSICAL MEMORY ALLOCATION

The memory management subsystem allows each running process in the system a fair share of the physical memory of the system,

SHARED VIRTUAL MEMORY

Although virtual memory allows processes to have separate (virtual) address spaces, there are times when you need processes to share memory. For example there could be several processes in the system running the bash command shell. Rather than have several copies of bash, one in each processes virtual address space, it is better to have only one copy in physical memory and all of the processes running bash share it. Dynamic libraries are another common example of executing code shared between several processes.

ABSTRACT MODEL OF VIRTUAL MEMORY

Before considering the methods that Linux uses to support virtual memory it is useful to consider an abstract model that is not cluttered by too much detail.

As the processor executes a programme it reads an instruction from memory and decodes it. In decoding the instruction it may need to fetch or store the contents of a location in memory. The processor then executes the instruction and moves onto the next instruction in the programme. In this way the processor is always accessing memory either to fetch instructions or to fetch and store data.

DEMAND PAGING

As there is much less physical memory than virtual memory the operating system must be careful that it does

not use the physical memory inefficiently. One way to save physical memory is to only load virtual pages that are currently being used by the executing programme.

For example, a database programme may be run to query a database. In this case not all of the database needs to be loaded into memory, just those data records that are being examined. If the database query is a search query then it does not make sense to load the code from the database programme that deals with adding new records. This technique of only loading virtual pages into memory as they are accessed is known as demand paging.

SWAPPING

If a process needs to bring a virtual page into physical memory and there are no free physical pages available, the operating system must make room for this page by discarding another page from physical memory. If the page to be discarded from physical memory came from an image or data file and has not been written to then the page does not need to be saved. Instead it can be discarded and if the process needs that page again it can be brought back into memory from the image or data file.

However, if the page has been modified, the operating system must preserve the contents of that page so that it can be accessed at a later time. This type of page is known as a *dirty* page and when it is removed from memory it is saved in a special sort of file called the swap file. Accesses to the swap file are very long relative to the speed of the processor and physical memory and the operating system must juggle the need to write pages to disk with the need to retain them in memory to be used again.

SHARED VIRTUAL MEMORY

Virtual memory makes it easy for several processes to share memory. All memory access are made via page tables and each process has its own separate page table. For two processes sharing a physical page of memory, its physical page frame number must appear in a page table entry in both of their page tables shows two processes that each share physical page frame number 4. For process *X* this is virtual page frame number 4 whereas for process *Y* this is virtual page frame number 6. This illustrates an interesting point about sharing pages: the shared physical page does not have to exist at the same place in virtual memory for any or all of the processes sharing it.

2

Computer Organization

MULTICORE DESIGNS

A different approach to improving a computer's performance is to add extra processors, as in symmetric multiprocessing designs which have been popular in servers and workstations since the early 1990s. Keeping up with Moore's Law is becoming increasingly challenging as chip-making technologies approach the physical limits of the technology. In response, the microprocessor manufacturers look for other ways to improve performance, in order to hold on to the momentum of constant upgrades in the market. A multi-core processor is simply a single chip containing more than one microprocessor core, effectively multiplying the potential performance with the number of cores (as long as the operating system and software is designed to take advantage of more than one processor). Some components,

such as bus interface and second level cache, may be shared between cores. Because the cores are physically very close they interface at much faster clock rates compared to discrete multiprocessor systems, improving overall system performance.

In 2005, the first personal computer dual-core processors were announced and as of 2009 dual-core and quad-core processors are widely used in servers, workstations and PCs while six and eight-core processors will be available for high-end applications in both the home and professional environments. Sun Microsystems has released the Niagara and Niagara 2 chips, both of which feature an eight-core design. The Niagara 2 supports more threads and operates at 1.6 GHz. High-end Intel Xeon processors that are on the LGA771 socket are DP (dual processor) capable, as well as the Intel Core 2 Extreme QX9775 also used in the Mac Pro by Apple and the Intel Skulltrail motherboard. With the transition to the LGA1366 socket and the Intel i7 chip quad core is now considered mainstream and the upcoming i9 chip will introduce six and possibly dual-die hex-core (12-cores), processors.

RISC

In the mid-1980s to early-1990s, a crop of new high-performance Reduced Instruction Set Computer (RISC) microprocessors appeared, influenced by discrete RISC-like CPU designs such as the IBM 801 and others. RISC microprocessors were initially used in special-purpose machines and Unix workstations, but then gained wide acceptance in other roles.

In 1986, HP released its first system with a PA-RISC CPU. The first commercial microprocessor design was released either by MIPS Computer Systems, the 32-bit R2000 (the R1000 was not released) or by Acorn computers, the 32-bit ARM2 in 1987. The R3000 made the design truly practical, and the R4000 introduced the world's first commercially available 64-bit RISC microprocessor. Competing projects would result in the IBM POWER and Sun SPARC architectures. Soon every major vendor was releasing a RISC design, including the AT&T CRISP, AMD 29000, Intel i860 and Intel i960, Motorola 88000, DEC Alpha. As of 2007, two 64-bit RISC architectures are still produced in volume for non-embedded applications: SPARC and Power ISA. The Itanium is produced in smaller quantities. The vast majority of 64-bit microprocessors are now x86-64 CISC designs from AMD and Intel.

SPECIAL-PURPOSE DESIGNS

Though the term "microprocessor" has traditionally referred to a single-or multi-chip CPU or system-on-a-chip (SoC), several types of specialized processing devices have followed from the technology. The most common examples are microcontrollers, digital signal processors (DSP) and graphics processing units (GPU). Many examples of these are either not programmable, or have limited programming facilities. For example, in general GPUs through the 1990s were mostly non-programmable and have only recently gained limited facilities like programmable vertex shaders. There is no universal consensus on what defines a "microprocessor", but it is usually safe to assume that the

term refers to a general-purpose CPU of some sort and not a special-purpose processor unless specifically noted.

MARKET STATISTICS

In 2003, about $44 billion (USD) worth of microprocessors were manufactured and sold. Although about half of that money was spent on CPUs used in desktop or laptop personal computers, those count for only about 0.2% of all CPUs sold.

About 55% of all CPUs sold in the world are 8-bit microcontrollers, over two billion of which were sold in 1997. As of 2002, less than 10% of all the CPUs sold in the world are 32-bit or more. Of all the 32-bit CPUs sold, about 2% are used in desktop or laptop personal computers. Most microprocessors are used in embedded control applications such as household appliances, automobiles, and computer peripherals. Taken as a whole, the average price for a microprocessor, microcontroller, or DSP is just over $6. About ten billion CPUs were manufactured in 2008. About 98% of new CPUs produced each year are embedded.

MICROCONTROLLER

A microcontroller (also microcontroller unit, MCU or μC) is a small computer on a single integrated circuit consisting of a relatively simple CPU combined with support functions such as a crystal oscillator, timers, watchdog timer, serial and analog I/O etc. Program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are

designed for small or dedicated applications. Thus, in contrast to the microprocessors used in personal computers and other high-performance or general purpose applications, simplicity is emphasized. Some microcontrollers may operate at clock rate frequencies as low as 4 kHz, as this is adequate for many typical applications, enabling low power consumption (milliwatts or microwatts). They will generally have the ability to retain functionality while waiting for an event such as a button press or other interrupt; power consumption while sleeping (CPU clock and most peripherals off) may be just nanowatts, making many of them well suited for long lasting battery applications. Other microcontrollers may serve performance-critical roles, where they may need to act more like a digital signal processor (DSP), with higher clock speeds and power consumption.

Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, remote controls, office machines, appliances, power tools, and toys. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digitally control even more devices and processes. Mixed signal microcontrollers are common, integrating analog components needed to control non-digital electronic systems.

EMBEDDED DESIGN

A microcontroller can be considered a self-contained system with a processor, memory and peripherals and can be used with an embedded system. (Only the software needs be added.) The majority of computer systems in use today

are embedded in other machinery, such as automobiles, telephones, appliances, and peripherals for computer systems. These are called embedded systems. While some embedded systems are very sophisticated, many have minimal requirements for memory and program length, with no operating system, and low software complexity. Typical input and output devices include switches, relays, solenoids, LEDs, small or custom LCD displays, radio frequency devices, and sensors for data such as temperature, humidity, light level etc. Embedded systems usually have no keyboard, screen, disks, printers, or other recognizable I/O devices of a personal computer, and may lack human interaction devices of any kind.

INTERRUPTS

Microcontrollers must provide real time (predictable, though not necessarily fast) response to events in the embedded system they are controlling. When certain events occur, an interrupt system can signal the processor to suspend processing the current instruction sequence and to begin an interrupt service routine (ISR, or "interrupt handler"). The ISR will perform any processing required based on the source of the interrupt before returning to the original instruction sequence. Possible interrupt sources are device dependent, and often include events such as an internal timer overflow, completing an analog to digital conversion, a logic level change on an input such as from a button being pressed, and data received on a communication link. Where power consumption is important as in battery operated devices, interrupts may also wake

a microcontroller from a low power sleep state where the processor is halted until required to do something by a peripheral event.

MULTIPROCESSING AND MULTITHREADING

Computer architects have become stymied by the growing mismatch in CPU operating frequencies and DRAM access times. None of the techniques that exploited instruction-level parallelism within one program could make up for the long stalls that occurred when data had to be fetched from main memory. Additionally, the large transistor counts and high operating frequencies needed for the more advanced ILP techniques required power dissipation levels that could no longer be cheaply cooled. For these reasons, newer generations of computers have started to exploit higher levels of parallelism that exist outside of a single program or program thread. This trend is sometimes known as *throughput computing*. This idea originated in the mainframe market where online transaction processing emphasized not just the execution speed of one transaction, but the capacity to deal with massive numbers of transactions. With transaction-based applications such as network routing and website serving greatly increasing in the last decade, the computer industry has re-emphasized capacity and throughput issues.

One technique of how this parallelism is achieved is through multiprocessing systems, computer systems with multiple CPUs. Once reserved for high-end mainframes and supercomputers, small scale (2-8) multiprocessors servers have become commonplace for the small business market.

For large corporations, large scale (16-256) multiprocessors are common. Even personal computers with multiple CPUs have appeared since the 1990s.

With further transistor size reductions made available with semiconductor technology advances, multicore CPUs have appeared where multiple CPUs are implemented on the same silicon chip. Initially used in chips targeting embedded markets, where simpler and smaller CPUs would allow multiple instantiations to fit on one piece of silicon. By 2005, semiconductor technology allowed dual high-end desktop CPUs *CMP* chips to be manufactured in volume. Some designs, such as Sun Microsystems' UltraSPARC T1 have reverted back to simpler (scalar, in-order) designs in order to fit more processors on one piece of silicon.

Another technique that has become more popular recently is multithreading. In multithreading, when the processor has to fetch data from slow system memory, instead of stalling for the data to arrive, the processor switches to another program or program thread which is ready to execute. Though this does not speed up a particular program/thread, it increases the overall system throughput by reducing the time the CPU is idle.

Conceptually, multithreading is equivalent to a context switch at the operating system level. The difference is that a multithreaded CPU can do a thread switch in one CPU cycle instead of the hundreds or thousands of CPU cycles a context switch normally requires. This is achieved by replicating the state hardware (such as the register file and program counter) for each active thread. A further

enhancement is simultaneous multithreading. This technique allows superscalar CPUs to execute instructions from different programs/threads simultaneously in the same cycle.

MICROPROCESSOR

A microprocessor incorporates most or all of the functions of a central processing unit (CPU) on a single integrated circuit (IC). The first microprocessors emerged in the early 1970s and were used for electronic calculators, using binary-coded decimal (BCD) arithmetic on 4-bit words. Other embedded uses of 4-and 8-bit microprocessors, such as terminals, printers, various kinds of automation etc, followed rather quickly. Affordable 8-bit microprocessors with 16-bit addressing also led to the first general purpose microcomputers in the mid-1970s.

Computer processors were for a long period constructed out of small and medium-scale ICs containing the equivalent of a few to a few hundred transistors. The integration of the whole CPU onto a single chip therefore greatly reduced the cost of processing capacity. From their humble beginnings, continued increases in microprocessor capacity have rendered other forms of computers almost completely obsolete, with one or more microprocessor as processing element in everything from the smallest embedded systems and handheld devices to the largest mainframes and supercomputers.

Since the early 1970s, the increase in capacity of microprocessors has been known to generally follow Moore's Law, which suggests that the complexity of an integrated

circuit, with respect to minimum component cost, doubles every two years. In the late 1990s, and in the high-performance microprocessor segment, heat generation (TDP), due to switching losses, static current leakage, and other factors, emerged as a leading developmental constraint.

HISTORY

FIRST TYPES

Three projects arguably delivered a complete microprocessor at about the same time, namely Intel's 4004, the Texas Instruments (TI) *TMS* 1000, and Garrett AiResearch's Central Air Data Computer (CADC). Intel's 4004 is considered the first microprocesor, and cost in the thousands of dollars. The first known advertisement for the 4004 is dated to November 1971; it appeared in Electronic News. The project that produced the 4004 originated in 1969, when Busicom, a Japanese calculator manufacturer, asked Intel to build a chipset for high-performance desktop calculators. Busicom's original design called for a dozen different logic and memory chips. Ted Hoff, the Intel engineer assigned to the project, believed the design was not cost effective. His solution was to simplify the design and produce a programmable processor capable of creating a set of complex special-purpose calculator chips. Together with Masatoshi Shima and Federico Faggin, later the founder of Zilog, Hoff came up with a four-chip design; a ROM for custom application programs, a RAM for processing data, an I/O device, and an unnamed 4-bit central processing unit which would become known as a "microprocessor."

The Smithsonian Institution says TI engineers Gary Boone and Michael Cochran succeeded in creating the first microcontroller (also called a microcomputer) in 1971. The result of their work was the TMS 1000 which went commercial in 1974.

In early 1971 Pico Electronics. and General Instrument introduced their first collaboration in ICs, a complete single chip calculator IC for the Monroe Royal Digital III calculator. This IC could also arguably lay claim to be one of the first microprocessors or microcontrollers having ROM, RAM and a RISC instruction set on-chip. Pico was a spinout by five GI design engineers whose vision was to create single chip calculator ICs. They had significant previous design experience on multiple calculator chipsets with both GI and Marconi-Elliott. Pico and GI went on to have significant success in the burgeoning handheld calculator market.

The design engineer Ray Holt, a graduate of California Polytechnical University in 1968, began his computer design career with the F14 CADC. The central air data computer was shrouded in secrecy for over 30 years from its creation (the year being 1968), it was not publicly known until 1998 at which time, at the request of Mr. Ray Holt, the US Navy allowed the documents into the public domain. Since then many debates have argued that this was, in fact, the first microprocessor. The scientific papers and literature published around 1971 reveal that the MP944 digital processor used for the F-14 Tomcat aircraft of the US Navy qualifies as the “first microprocessor”. Although interesting, it was not a single-chip processor, and was not general purpose – it was more like a set of parallel building blocks

you could use to make a special-purpose DSP form. It indicates that today's industry theme of converging DSP-microcontroller architectures was started in 1971. This convergence of DSP and microcontroller architectures is known as a Digital Signal Controller.

In 1968, Garrett AiResearch, with designer Ray Holt and Steve Geller, were invited to produce a digital computer to compete with electromechanical systems then under development for the main flight control computer in the US Navy's new F-14 Tomcat fighter. The design was complete by 1970, and used a MOS-based chipset as the core CPU. The design was significantly (approximately 20 times) smaller and much more reliable than the mechanical systems it competed against, and was used in all of the early Tomcat models. This system contained a "a 20-bit, pipelined, parallel multi-microprocessor". However, the system was considered so advanced that the Navy refused to allow publication of the design until 1997. For this reason the CADC, and the MP944 chipset it used, are fairly unknown even today. TI developed the 4-bit TMS 1000, and stressed pre-programmed embedded applications, introducing a version called the TMS1802NC on September 17, 1971, which implemented a calculator on a chip. The Intel chip was the 4-bit 4004, released on November 15, 1971, developed by Federico Faggin and Ted Hoff. The manager of the design team was Leslie L. Vadász.

TI filed for the patent on the microprocessor. Gary Boone was awarded U.S. Patent 3,757,306 for the single-chip microprocessor architecture on September 4, 1973. It may never be known which company actually had the first working

microprocessor running on the lab bench. In both 1971 and 1976, Intel and TI entered into broad patent cross-licensing agreements, with Intel paying royalties to TI for the microprocessor patent. A nice history of these events is contained in court documentation from a legal dispute between Cyrix and Intel, with TI as intervenor and owner of the microprocessor patent.

Interestingly, a third party (Gilbert Hyatt) was awarded a patent which might cover the "microprocessor". See a webpage claiming an invention pre-dating both TI and Intel, describing a "microcontroller". According to a rebuttal and a commentary, the patent was later invalidated, but not before substantial royalties were paid out.

A computer-on-a-chip is a variation of a microprocessor which combines the microprocessor core (CPU), some memory, and I/O (input/output) lines, all on one chip.It is also called as micro-controller. The computer-on-a-chip patent, called the "microcomputer patent" at the time, U.S. Patent 4,074,351, was awarded to Gary Boone and Michael J. Cochran of TI. Aside from this patent, the standard meaning of microcomputer is a computer using one or more microprocessors as its CPU(s), while the concept defined in the patent is perhaps more akin to a microcontroller. According to *A History of Modern Computing*, (MIT Press), pp. 220–21, Intel entered into a contract with Computer Terminals Corporation, later called Datapoint, of San Antonio TX, for a chip for a terminal they were designing. Datapoint later decided not to use the chip, and Intel marketed it as the 8008 in April, 1972. This was the world's first 8-bit microprocessor. It was the basis for the famous "Mark-8"

computer kit advertised in the magazine Radio-Electronics in 1974. The 8008 and its successor, the world-famous 8080, opened up the microprocessor component marketplace.

NOTABLE 8-BIT DESIGNS

The 4004 was later followed in 1972 by the 8008, the world's first 8-bit microprocessor. These processors are the precursors to the very successful Intel 8080 (1974), Zilog Z80 (1976), and derivative Intel 8-bit processors. The competing Motorola 6800 was released August 1974 and the similar MOS Technology 6502 in 1975 (designed largely by the same people). The 6502 rivaled the Z80 in popularity during the 1980s.

A low overall cost, small packaging, simple computer bus requirements, and sometimes circuitry otherwise provided by external hardware (the Z80 had a built in memory refresh) allowed the home computer "revolution" to accelerate sharply in the early 1980s, eventually delivering such inexpensive machines as the Sinclair ZX-81, which sold for US$99.

The Western Design Centre, Inc. (WDC) introduced the CMOS 65C02 in 1982 and licensed the design to several firms. It was used as the CPU in the Apple IIc and IIe personal computers as well as in medical implantable grade pacemakers and defibrilators, automotive, industrial and consumer devices. WDC pioneered the licensing of microprocessor designs, later followed by ARM and other microprocessor Intellectual Property (IP) providers in the 1990's.

Motorola introduced the MC6809 in 1978, an ambitious and thought through 8-bit design source compatible with the 6800 and implemented using purely hard-wired logic. (Subsequent 16-bit microprocessors typically used microcode to some extent, as design requirements were getting too complex for purely hard-wired logic only.)

Another early 8-bit microprocessor was the Signetics 2650, which enjoyed a brief surge of interest due to its innovative and powerful instruction set architecture.

A seminal microprocessor in the world of spaceflight was RCA's RCA 1802 (aka CDP1802, RCA COSMAC) (introduced in 1976) which was used in NASA's Voyager and Viking spaceprobes of the 1970s, and onboard the Galileo probe to Jupiter (launched 1989, arrived 1995). RCA COSMAC was the first to implement C-MOS technology. The CDP1802 was used because it could be run at very low power, and because its production process (Silicon on Sapphire) ensured much better protection against cosmic radiation and electrostatic discharges than that of any other processor of the era. Thus, the 1802 is said to be the first radiation-hardened microprocessor.

The RCA 1802 had what is called a *static design*, meaning that the clock frequency could be made arbitrarily low, even to 0 Hz, a total stop condition. This let the Voyager/Viking/Galileo spacecraft use minimum electric power for long uneventful stretches of a voyage. Timers and/or sensors would awaken/improve the performance of the processor in time for important tasks, such as navigation updates, attitude control, data acquisition, and radio communication.

16-BIT DESIGNS

The first multi-chip 16-bit microprocessor was the National Semiconductor IMP-16, introduced in early 1973. An 8-bit version of the chipset was introduced in 1974 as the IMP-8. During the same year, National introduced the first 16-bit single-chip microprocessor, the National Semiconductor PACE, which was later followed by an NMOS version, the INS8900.

Other early multi-chip 16-bit microprocessors include one used by Digital Equipment Corporation (DEC) in the LSI-11 OEM board set and the packaged PDP 11/03 minicomputer, and the Fairchild Semiconductor MicroFlame 9440, both of which were introduced in the 1975 to 1976 timeframe.

The first single-chip 16-bit microprocessor was TI's TMS 9900, which was also compatible with their TI-990 line of minicomputers. The 9900 was used in the TI 990/4 minicomputer, the TI-99/4A home computer, and the TM990 line of OEM microcomputer boards. The chip was packaged in a large ceramic 64-pin DIP package, while most 8-bit microprocessors such as the Intel 8080 used the more common, smaller, and less expensive plastic 40-pin DIP. A follow-on chip, the TMS 9980, was designed to compete with the Intel 8080, had the full TI 990 16-bit instruction set, used a plastic 40-pin package, moved data 8 bits at a time, but could only address 16 KB. A third chip, the TMS 9995, was a new design. The family later expanded to include the 99105 and 99110. The Western Design Centre, Inc. (WDC) introduced the CMOS 65816 16-bit upgrade of the WDC CMOS 65C02 in 1984. The 65816 16-bit

microprocessor was the core of the Apple IIgs and later the Super Nintendo Entertainment System, making it one of the most popular 16-bit designs of all time.

Intel followed a different path, having no minicomputers to emulate, and instead "upsized" their 8080 design into the 16-bit Intel 8086, the first member of the x86 family which powers most modern PC type computers. Intel introduced the 8086 as a cost effective way of porting software from the 8080 lines, and succeeded in winning much business on that premise. The 8088, a version of the 8086 that used an external 8-bit data bus, was the microprocessor in the first IBM PC, the model 5150. Following up their 8086 and 8088, Intel released the 80186, 80286 and, in 1985, the 32-bit 80386, cementing their PC market dominance with the processor family's backwards compatibility.

The integrated microprocessor memory management unit (MMU) was developed by Childs et al. of Intel, and awarded US patent number 4,442,484.

32-BIT DESIGNS

The most significant of the 32-bit designs is the MC68000, introduced in 1979. The 68K, as it was widely known, had 32-bit registers but used 16-bit internal data paths and a 16-bit external data bus to reduce pin count, and supported only 24-bit addresses. Motorola generally described it as a 16-bit processor, though it clearly has 32-bit architecture. The combination of high performance, large (16 megabytes or 2^{24} bytes) memory space and fairly low cost made it the most popular CPU design of its class. The Apple Lisa and Macintosh designs made use of the 68000, as did a host

of other designs in the mid-1980s, including the Atari ST and Commodore Amiga.

The world's first single-chip fully-32-bit microprocessor, with 32-bit data paths, 32-bit buses, and 32-bit addresses, was the AT&T Bell Labs BELLMAC-32A, with first samples in 1980, and general production in 1982. After the divestiture of AT&T in 1984, it was renamed the WE 32000 (WE for Western Electric), and had two follow-on generations, the WE 32100 and WE 32200. These microprocessors were used in the AT&T 3B5 and 3B15 minicomputers; in the 3B2, the world's first desktop supermicrocomputer; in the "Companion", the world's first 32-bit laptop computer; and in "Alexander", the world's first book-sized supermicrocomputer, featuring ROM-pack memory cartridges similar to today's gaming consoles. All these systems ran the UNIX System V operating system.

Intel's first 32-bit microprocessor was the iAPX 432, which was introduced in 1981 but was not a commercial success. It had an advanced capability-based object-oriented architecture, but poor performance compared to contemporary architectures such as Intel's own 80286 (introduced 1982), which was almost four times as fast on typical benchmark tests. However, the results for the iAPX432 was partly due to a rushed and therefore suboptimal Ada compiler.

Motorola's success with the 68000 led to the MC68010, which added virtual memory support. The MC68020, introduced in 1985 added full 32-bit data and address busses. The 68020 became hugely popular in the Unix supermicrocomputer market, and many small companies

(e.g., Altos, Charles River Data Systems) produced desktop-size systems. The MC68030 was introduced next, improving upon the previous design by integrating the MMU into the chip. The continued success led to the MC68040, which included an FPU for better math performance. A 68050 failed to achieve its performance goals and was not released, and the follow-up MC68060 was released into a market saturated by much faster RISC designs. The 68K family faded from the desktop in the early 1990s.

Other large companies designed the 68020 and follow-ons into embedded equipment. At one point, there were more 68020s in embedded equipment than there were Intel Pentiums in PCs. The ColdFire processor cores are derivatives of the venerable 68020.

During this time (early to mid 1980s), National Semiconductor introduced a very similar 16-bit pinout, 32-bit internal microprocessor called the NS 16032 (later renamed 32016), the full 32-bit version named the NS 32032, and a line of 32-bit industrial OEM microcomputers. By the mid-1980s, Sequent introduced the first symmetric multiprocessor (SMP) server-class computer using the NS 32032. This was one of the design's few wins, and it disappeared in the late 1980s.

The MIPS R2000 (1984) and R3000 (1989) were highly successful 32-bit RISC microprocessors. They were used in high-end workstations and servers by SGI, among others.

Other designs included the interesting Zilog Z8000, which arrived too late to market to stand a chance and disappeared quickly.

In the late 1980s, "microprocessor wars" started killing off some of the microprocessors. Apparently, with only one major design win, Sequent, the NS 32032 just faded out of existence, and Sequent switched to Intel microprocessors.

From 1985 to 2003, the 32-bit x86 architectures became increasingly dominant in desktop, laptop, and server markets, and these microprocessors became faster and more capable. Intel had licensed early versions of the architecture to other companies, but declined to license the Pentium, so AMD and Cyrix built later versions of the architecture based on their own designs. During this span, these processors increased in complexity (transistor count) and capability (instructions/second) by at least three orders of magnitude. Intel's Pentium line is probably the most famous and recognizable 32-bit processor model, at least with the public at large.

64-BIT DESIGNS IN PERSONAL COMPUTERS

While 64-bit microprocessor designs have been in use in several markets since the early 1990s, the early 1990s. saw the introduction of 64-bit microprocessors targeted at the PC market.

With AMD's introduction of a 64-bit architecture backwards-compatible with x86, x86-64 (now called AMD64), in September 2003, followed by Intel's near fully compatible 64-bit extensions (first called IA-32e or EM64T, later renamed Intel 64), the 64-bit desktop era began. Both versions can run 32-bit legacy applications without any performance penalty as well as new 64-bit software. With operating systems Windows XP x64, Windows Vista x64, Linux, BSD

and Mac OS X that run 64-bit native, the software is also geared to fully utilize the capabilities of such processors. The move to 64 bits is more than just an increase in register size from the IA-32 as it also doubles the number of general-purpose registers.

The move to 64 bits by PowerPC processors had been intended since the processors' design in the early 90s and was not a major cause of incompatibility. Existing integer registers are extended as are all related data pathways, but, as was the case with IA-32, both floating point and vector units had been operating at or above 64 bits for several years. Unlike what happened when IA-32 was extended to x86-64, no new general purpose registers were added in 64-bit PowerPC, so any performance gained when using the 64-bit mode for applications making no use of the larger address space is minimal.

RELATION TO INSTRUCTION SET ARCHITECTURE

The ISA is roughly the same as the programming model of a processor as seen by an assembly language programmer or compiler writer. The ISA includes the execution model, processor registers, address and data formats among other things. The microarchitecture includes the constituent parts of the processor and how these interconnect and interoperate to implement the ISA.

The microarchitecture of a machine is usually represented as (more or less detailed) diagrams that describe the interconnections of the various microarchitectual elements of the machine, which may be everything from single gates

and registers, to complete ALUs and even larger elements. These diagrams generally separate the data path (where data is placed) and the control path (which can be said to steer the data). Machines with different microarchitectures may have the same instruction set architecture, and thus be capable of executing the same programs. New microarchitectures and/or circuitry solutions, along with advances in semiconductor manufacturing, are what allows newer generations of processors to achieve higher performance while using the same ISA.

ASPECTS OF MICROARCHITECTURE

The pipelined datapath is the most commonly used datapath design in microarchitecture today. This technique is used in most modern microprocessors, microcontrollers, and DSPs. The pipelined architecture allows multiple instructions to overlap in execution, much like an assembly line. The pipeline includes several different stages which are fundamental in microarchitecture designs. Some of these stages include instruction fetch, instruction decode, execute, and write back. Some architectures include other stages such as memory access. The design of pipelines is one of the central microarchitectural tasks.

Execution units are also essential to microarchitecture. Execution units include arithmetic logic units (ALU), floating point units (FPU), load/store units, branch prediction, and SIMD. These units perform the operations or calculations of the processor. The choice of the number of execution units, their latency and throughput is a central microarchitectural design task. The size, latency, throughput

and connectivity of memories within the system are also microarchitectural decisions. System-level design decisions such as whether or not to include peripherals, such as memory controllers, can be considered part of the microarchitectural design process. This includes decisions on the performance-level and connectivity of these peripherals. Unlike architectural design, where achieving a specific performance level is the main goal, microarchitectural design pays closer attention to other constraints. Since microarchitecture design decisions directly affect what goes into a system, attention must be paid to such issues as:

- chip area/cost
- power consumption
- logic complexity
- ease of connectivity
- manufacturability
- ease of debugging
- testability.

MICROARCHITECTURAL CONCEPTS

In general, all CPUs, single-chip microprocessors or multi-chip implementations run programs by performing the following steps:

1. read an instruction and decode it
2. find any associated data that is needed to process the instruction
3. process the instruction
4. write the results out.

Complicating this simple-looking series of steps is the fact that the memory hierarchy, which includes caching, main memory and non-volatile storage like hard disks, (where the program instructions and data reside) has always been slower than the processor itself. Step (2) often introduces a lengthy (in CPU terms) delay while the data arrives over the computer bus. A considerable amount of research has been put into designs that avoid these delays as much as possible. Over the years, a central goal was to execute more instructions in parallel, thus increasing the effective execution speed of a program. These efforts introduced complicated logic and circuit structures. Initially these techniques could only be implemented on expensive mainframes or supercomputers due to the amount of circuitry needed for these techniques. As semiconductor manufacturing progressed, more and more of these techniques could be implemented on a single semiconductor chip.

What follows is a survey of micro-architectural techniques that are common in modern CPUs.

INSTRUCTION SET CHOICE

Instruction sets have shifted over the years, from originally very simple to sometimes very complex (in various respects). In recent years, load-store architectures, VLIW and EPIC types have been in fashion. Architectures that are dealing with data parallelism include SIMD and Vectors. Some labels used to denote classes of CPU architectures are not particularity descriptive, especially so the CISC label; many early designs, retroactively denoted “CISC” are in fact

significantly simpler than modern RISC processors (in several respects). However, the choice of instruction set architecture may greatly affect the complexity of implementing high performance devices. The prominent strategy, used to develop the first RISC processors, was to simplify instructions to a minimum of individual semantic complexity combined with high encoding regularity and simplicity. Such uniform instructions were easily fetched, decoded and executed in a pipelined fashion and a simple strategy to reduce the number of logic levels in order to reach high operating frequencies; instruction cache-memories compensated for the higher operating frequency and inherently low code density while large register sets were used to factor out as much of the (slow) memory accesses as possible.

INSTRUCTION PIPELINING

One of the first, and most powerful, techniques to improve performance is the use of the instruction pipeline. Early processor designs would carry out all of the steps above for one instruction before moving onto the next. Large portions of the circuitry were left idle at any one step; for instance, the instruction decoding circuitry would be idle during execution and so on.

Pipelines improve performance by allowing a number of instructions to work their way through the processor at the same time. In the same basic example, the processor would start to decode (step 1) a new instruction while the last one was waiting for results. This would allow up to four instructions to be "in flight" at one time, making the processor look four times as fast. Although any one instruction takes

just as long to complete (there are still four steps) the CPU as a whole "retires" instructions much faster and can be run at a much higher clock speed.

RISC make pipelines smaller and much easier to construct by cleanly separating each stage of the instruction process and making them take the same amount of time — one cycle. The processor as a whole operates in an assembly line fashion, with instructions coming in one side and results out the other. Due to the reduced complexity of the Classic RISC pipeline, the pipelined core and an instruction cache could be placed on the same size die that would otherwise fit the core alone on a CISC design. This was the real reason that RISC was faster. Early designs like the SPARC and MIPS often ran over 10 times as fast as Intel and Motorola CISC solutions at the same clock speed and price.

Pipelines are by no means limited to RISC designs. By 1986 the top-of-the-line VAX implementation (VAX 8800) was a heavily pipelined design, slightly predating the first commercial MIPS and SPARC designs. Most modern CPUs (even embedded CPUs) are now pipelined, and microcoded CPUs with no pipelining are seen only in the most area-constrained embedded processors. Large CISC machines, from the VAX 8800 to the modern Pentium 4 and Athlon, are implemented with both microcode and pipelines. Improvements in pipelining and caching are the two major microarchitectural advances that have enabled processor performance to keep pace with the circuit technology on which they are based.

CACHE

It was not long before improvements in chip manufacturing allowed for even more circuitry to be placed on the die, and designers started looking for ways to use it. One of the most common was to add an ever-increasing amount of cache memory on-die. Cache is simply very fast memory, memory that can be accessed in a few cycles as opposed to "many" needed to talk to main memory. The CPU includes a cache controller which automates reading and writing from the cache, if the data is already in the cache it simply "appears," whereas if it is not the processor is "stalled" while the cache controller reads it in.

RISC designs started adding cache in the mid-to-late 1980s, often only 4 KB in total. This number grew over time, and typical CPUs now have at least 512 KB, while more powerful CPUs come with 1 or 2 or even 4, 6, 8 or 12 MB, organized in multiple levels of a memory hierarchy. Generally speaking, more cache means more performance, thanks to reduced stalling. Caches and pipelines were a perfect match for each other. Previously, it didn't make much sense to build a pipeline that could run faster than the access latency of off-chip memory. Using on-chip cache memory instead, meant that a pipeline could run at the speed of the cache access latency, a much smaller length of time. This allowed the operating frequencies of processors to increase at a much faster rate than that of off-chip memory.

BRANCH PREDICTION

One barrier to achieving higher performance through instruction-level parallelism stems from pipeline stalls and

flushes due to branches. Normally, whether a conditional branch will be taken isn't known until late in the pipeline as conditional branches depend on results coming from a register. From the time that the processor's instruction decoder has figured out that it has encountered a conditional branch instruction to the time that the deciding register value can be read out, the pipeline needs to be stalled for several cycles, or if it's not and the branch is taken, the pipeline needs to be flushed. As clock speeds increase the depth of the pipeline increases with it, and some modern processors may have 20 stages or more. On average, every fifth instruction executed is a branch, so without any intervention, that's a high amount of stalling.

Techniques such as branch prediction and speculative execution are used to lessen these branch penalties. Branch prediction is where the hardware makes educated guesses on whether a particular branch will be taken. In reality one side or the other of the branch will be called much more often than the other. Modern designs have rather complex statistical prediction systems, which watch the results of past branches to predict the future with greater accuracy. The guess allows the hardware to prefect instructions without waiting for the register read. Speculative execution is a further enhancement in which the code along the predicted path is not just perfected but also executed before it is known whether the branch should be taken or not. This can yield better performance when the guess is good, with the risk of a huge penalty when the guess is bad because instructions need to be undone.

SUPERSCALAR

Even with all of the added complexity and gates needed to support the concepts outlined above, improvements in semiconductor manufacturing soon allowed even more logic gates to be used.

In the outline above the processor processes parts of a single instruction at a time. Computer programs could be executed faster if multiple instructions were processed simultaneously. This is what superscalar processors achieve, by replicating functional units such as ALUs. The replication of functional units was only made possible when the die area of a single-issue processor no longer stretched the limits of what could be reliably manufactured. By the late 1980s, superscalar designs started to enter the market place.

In modern designs it is common to find two load units, one store (many instructions have no results to store), two or more integer math units, two or more floating point units, and often a SIMD unit of some sort. The instruction issue logic grows in complexity by reading in a huge list of instructions from memory and handing them off to the different execution units that are idle at that point. The results are then collected and re-ordered at the end.

OUT-OF-ORDER EXECUTION

The addition of caches reduces the frequency or duration of stalls due to waiting for data to be fetched from the memory hierarchy, but does not get rid of these stalls entirely. In early designs a *cache miss* would force the cache controller to stall the processor and wait. Of course there

may be some other instruction in the program whose data *is* available in the cache at that point. Out-of-order execution allows that ready instruction to be processed while an older instruction waits on the cache, then re-orders the results to make it appear that everything happened in the programmed order. This technique is also used to avoid other operand dependency stalls, such as an instruction awaiting a result from a long latency floating-point operation or other multi-cycle operations.

REGISTER RENAMING

Register renaming refers to a technique used to avoid unnecessary serialized execution of program instructions because of the reuse of the same registers by those instructions. Suppose we have two groups of instruction that will use the same register, one set of instruction is executed first to leave the register to the other set, but if the other set is assigned to a different similar register both sets of instructions can be executed in parallel.

3

Interconnecting Networks Architecture

The authors created architecture for interconnecting independent networks that could then be federated into a seamless whole without changing any of the underlying networks. This was the genesis of the Internet, as we know it today. In order to work properly, the architecture required a global addressing mechanism (or Internet address) to enable computers on any network to reference and communicate with computers on any other network in the federation.

Internet addresses fill essentially the same role as telephone numbers do in telephone networks. The design of the Internet assumed first that the individual networks could not be changed to accommodate new architectural requirements; but this was largely a pragmatic assumption

to facilitate progress. The networks also had varying degrees of reliability and speed. Host computers would have to be able to put disordered packets back into the correct order and discard duplicate packets that had been generated along the way.

This was a major change from the virtual circuit-like service provided by ARPANET and by then contemporary commercial data networking services such as Tymnet and Telenet. In these networks, the underlying network took responsibility for keeping all information in order and for re-sending any data that might have been lost. The Internet design made the computers responsible for tending to these network problems.

A key architectural construct was the introduction of gateways (now called routers) between the networks to handle the disparities such as different data rates, packet sizes, error conditions, and interface specifications. The gateways would also check the destination Internet addresses of each packet to determine the gateway to which it should be forwarded.

These functions would be combined with certain end-end functions to produce the reliable communication from source to destination. A draft paper by the authors describing this approach was given at a meeting of the International Network Working Group in 1973 in Sussex, England and the final paper was subsequently published by the Institute for Electrical and Electronics Engineers, the leading professional society for the electrical engineering profession, in its Transactions on Communications in May, 1974. DARPA contracted with Cerf's group at Stanford to carry out the initial detailed design of the TCP software and, shortly

thereafter, with BBN and University College London to build independent implementations of the TCP protocol (as it was then called - it was later split into TCP and IP) for different machines.

BBN also had a contract to build a prototype version of the gateway. These three sites collaborated in the development and testing of the initial protocols on different machines. Cerf, then a professor at Stanford, provided the day-to-day leadership in the initial TCP software design and testing. BBN deployed the gateways between the ARPANET and the PRNET and also with SATNET. During this period, under Kahn's overall leadership at DARPA, the initial feasibility of the Internet Architecture was demonstrated.

The TCP/IP protocol suite was developed and refined over a period of four more years and, in 1980, it was adopted as a standard by the U.S. Department of Defence. On January 1, 1983 the ARPANET converted to TCP/IP as its standard host protocol. Gateways (or routers) were used to pass packets to and from host computers on "local area networks." Refinement and extension of these protocols and many others associated with them continues to this day by way of the Internet Engineering Task Force.

MODERN USES OF INTERNET

The Internet is allowing greater flexibility in working hours and location, especially with the spread of unmetreed high-speed connections and web applications. The Internet can now be accessed almost anywhere by numerous means, especially through mobile Internet devices.

Mobile phones, datacards, handheld game consoles and cellular routers allow users to connect to the Internet from anywhere there is a wireless network supporting that device's technology. Within the limitations imposed by small screens and other limited facilities of such pocket-sized devices, services of the Internet, including e-mail and the web, may be available.

Service providers may restrict the services offered and wireless data transmission charges may be significantly higher than other access methods. Educational material at all levels from pre-school to post-doctoral is available from websites. Examples range from CBeebies, through school and high-school revision guides, virtual universities, to access to top-end scholarly literature through the likes of Google Scholar. In distance education, help with homework and other assignments, self-guided learning, whiling away spare time, or just looking up more detail on an interesting fact, it has never been easier for people to access educational information at any level from anywhere.

The Internet in general and the World Wide Web in particular are important enablers of both formal and informal education. The low cost and nearly instantaneous sharing of ideas, knowledge, and skills has made collaborative work dramatically easier, with the help of collaborative software. Not only can a group cheaply communicate and share ideas, but the wide reach of the Internet allows such groups to easily form in the first place. An example of this is the free software movement, which has produced, among other programmes, Linux, Mozilla Firefox, and OpenOffice.org. Internet "chat", whether in the form of IRC chat rooms or

channels, or via instant messaging systems, allow colleagues to stay in touch in a very convenient way when working at their computers during the day.

Messages can be exchanged even more quickly and conveniently than via e-mail. Extensions to these systems may allow files to be exchanged, "whiteboard" drawings to be shared or voice and video contact between team members. Version control systems allow collaborating teams to work on shared sets of documents without either accidentally overwriting each other's work or having members wait until they get "sent" documents to be able to make their contributions.

Business and project teams can share calendars as well as documents and other information. Such collaboration occurs in a wide variety of areas including scientific research, software development, conference planning, political activism and creative writing. Social and political collaboration is also becoming more widespread as both Internet access and computer literacy grow. From the flash mob 'events' of the early 2000s to the use of social networking in the 2009 Iranian election protests, the Internet allows people to work together more effectively and in many more ways than was possible without it.

The Internet allows computer users to remotely access other computers and information stores easily, wherever they may be across the world. They may do this with or without the use of security, authentication and encryption technologies, depending on the requirements. This is encouraging new ways of working from home, collaboration and information sharing in many industries. An accountant

sitting at home can audit the books of a company based in another country, on a server situated in a third country that is remotely maintained by IT specialists in a fourth. These accounts could have been created by home-working bookkeepers, in other remote locations, based on information e-mailed to them from offices all over the world.

Some of these things were possible before the widespread use of the Internet, but the cost of private leased lines would have made many of them infeasible in practice. An office worker away from their desk, perhaps on the other side of the world on a business trip or a holiday, can open a remote desktop session into his normal office PC using a secure Virtual Private Network (VPN) connection via the Internet. This gives the worker complete access to all of his or her normal files and data, including e-mail and other applications, while away from the office.

This concept has been referred to among system administrators as the Virtual Private Nightmare, because it extends the secure perimetre of a corporate network into its employees' homes.

INTERNET SERVICES

INFORMATION

Many people use the terms Internet and World Wide Web, or just the Web, interchangeably, but the two terms are not synonymous. The World Wide Web is a global set of documents, images and other resources, logically interrelated by hyperlinks and referenced with Uniform Resource Identifiers (URIs).

URIs allow providers to symbolically identify services and clients to locate and address web servers, file servers, and other databases that store documents and provide resources and access them using the Hypertext Transfer Protocol (HTTP), the primary carrier protocol of the Web. HTTP is only one of the hundreds of communication protocols used on the Internet. Web services may also use HTTP to allow software systems to communicate in order to share and exchange business logic and data. World Wide Web browser software, such as Microsoft's Internet Explorer, Mozilla Firefox, Opera, Apple's Safari, and Google Chrome, let users navigate from one web page to another via hyperlinks embedded in the documents.

These documents may also contain any combination of computer data, including graphics, sounds, text, video, multimedia and interactive content including games, office applications and scientific demonstrations. Through keyword-driven Internet research using search engines like Yahoo! and Google, users worldwide have easy, instant access to a vast and diverse amount of online information.

Compared to printed encyclopedias and traditional libraries, the World Wide Web has enabled the decentralization of information. The Web has also enabled individuals and organizations to publish ideas and information to a potentially large audience online at greatly reduced expense and time delay. Publishing a web page, a blog, or building a website involves little initial cost and many cost-free services are available.

Publishing and maintaining large, professional web sites with attractive, diverse and up-to-date information is still a

difficult and expensive proposition, however. Many individuals and some companies and groups use web logs or blogs, which are largely used as easily updatable online diaries. Some commercial organizations encourage staff to communicate advice in their areas of specialization in the hope that visitors will be impressed by the expert knowledge and free information, and be attracted to the corporation as a result.

One example of this practice is Microsoft, whose product developers publish their personal blogs in order to pique the public's interest in their work. Collections of personal web pages published by large service providers remain popular, and have become increasingly sophisticated. Whereas operations such as Angelfire and GeoCities have existed since the early days of the Web, newer offerings from, for example, Facebook and MySpace currently have large followings. These operations often brand themselves as social network services rather than simply as web page hosts. Advertising on popular web pages can be lucrative, and e-commerce or the sale of products and services directly via the Web continues to grow. In the early days, web pages were usually created as sets of complete and isolated HTML text files stored on a web server. More recently, websites are more often created using content management or wiki software with, initially, very little content.

Contributors to these systems, who may be paid staff, members of a club or other organization or members of the public, fill underlying databases with content using editing pages designed for that purpose, while casual visitors view and read this content in its final HTML form. There may or

may not be editorial, approval and security systems built into the process of taking newly entered content and making it available to the target visitors.

COMMUNICATION

E-mail is an important communications service available on the Internet. The concept of sending electronic text messages between parties in a way analogous to mailing letters or memos predates the creation of the Internet. Today it can be important to distinguish between internet and internal e-mail systems. Internet e-mail may travel and be stored unencrypted on many other networks and machines out of both the sender's and the recipient's control. During this time it is quite possible for the content to be read and even tampered with by third parties, if anyone considers it important enough.

Purely internal or intranet mail systems, where the information never leaves the corporate or organization's network, are much more secure, although in any organization there will be IT and other personnel whose job may involve monitoring, and occasionally accessing, the e-mail of other employees not addressed to them. Pictures, documents and other files can be sent as e-mail attachments. E-mails can be cc-ed to multiple e-mail addresses. Internet telephony is another common communications service made possible by the creation of the Internet. VoIP stands for Voice-over-Internet Protocol, referring to the protocol that underlies all Internet communication. The idea began in the early 1990s with walkie-talkie-like voice applications for personal computers.

In recent years many VoIP systems have become as easy to use and as convenient as a normal telephone. The benefit is that, as the Internet carries the voice traffic, VoIP can be free or cost much less than a traditional telephone call, especially over long distances and especially for those with always-on Internet connections such as cable or ADSL. VoIP is maturing into a competitive alternative to traditional telephone service. Interoperability between different providers has improved and the ability to call or receive a call from a traditional telephone is available. Simple, inexpensive VoIP network adapters are available that eliminate the need for a personal computer.

Voice quality can still vary from call to call but is often equal to and can even exceed that of traditional calls. Remaining problems for VoIP include emergency telephone number dialling and reliability. Currently, a few VoIP providers provide an emergency service, but it is not universally available.

Traditional phones are line-powered and operate during a power failure; VoIP does not do so without a backup power source for the phone equipment and the Internet access devices. VoIP has also become increasingly popular for gaming applications, as a form of communication between players. Popular VoIP clients for gaming include Ventrilo and Teamspeak. Wii, PlayStation 3, and Xbox 360 also offer VoIP chat features.

DATA TRANSFER

File sharing is an example of transferring large amounts of data across the Internet. A computer file can be e-mailed

to customers, colleagues and friends as an attachment. It can be uploaded to a website or FTP server for easy download by others. It can be put into a "shared location" or onto a file server for instant use by colleagues.

The load of bulk downloads to many users can be eased by the use of "mirror" servers or peer-to-peer networks. In any of these cases, access to the file may be controlled by user authentication, the transit of the file over the Internet may be obscured by encryption, and money may change hands for access to the file. The price can be paid by the remote charging of funds from, for example, a credit card whose details are also passed—usually fully encrypted—across the Internet.

The origin and authenticity of the file received may be checked by digital signatures or by MD5 or other message digests. These simple features of the Internet, over a worldwide basis, are changing the production, sale, and distribution of anything that can be reduced to a computer file for transmission. This includes all manner of print publications, software products, news, music, film, video, photography, graphics and the other arts. This in turn has caused seismic shifts in each of the existing industries that previously controlled the production and distribution of these products.

Streaming media refers to the act that many existing radio and television broadcasters promote Internet "feeds" of their live audio and video streams (for example, the BBC). They may also allow time-shift viewing or listening such as Preview, Classic Clips and Listen Again features. These providers have been joined by a range of pure Internet

"broadcasters" who never had on-air licenses.

This means that an Internet-connected device, such as a computer or something more specific, can be used to access on-line media in much the same way as was previously possible only with a television or radio receiver. The range of available types of content is much wider, from specialized technical webcasts to on-demand popular multimedia services. Podcasting is a variation on this theme, where—usually audio—material is downloaded and played back on a computer or shifted to a portable media player to be listened to on the move. These techniques using simple equipment allow anybody, with little censorship or licensing control, to broadcast audio-visual material worldwide.

Webcams can be seen as an even lower-budget extension of this phenomenon. While some webcams can give full-frame-rate video, the picture is usually either small or updates slowly. Internet users can watch animals around an African waterhole, ships in the Panama Canal, traffic at a local roundabout or monitor their own premises, live and in real time.

Video chat rooms and video conferencing are also popular with many uses being found for personal webcams, with and without two-way sound. YouTube was founded on 15 February 2005 and is now the leading website for free streaming video with a vast number of users. It uses a flash-based web player to stream and show video files. Registered users may upload an unlimited amount of video and build their own personal profile. YouTube claims that its users watch hundreds of millions, and upload hundreds of thousands of videos daily.

PROCESS OF TROUBLESHOOTING

Outbound e-mail uses a process called SMTP (simple mail transport protocol). SMTP is the standard for e-mail transmissions across the Internet. SMTP is generally used to send messages from a mail client (Thunderbird) to a mail server. The e-mail is stored on the server until retrieved by another e-mail client using a POP3 or IMAP retrieval protocol. Outbound e-mail problems can be frustrating to deal with because you rarely see an indicator as to what might be causing the problem. However, there are fewer factors involved when setting up the SMTP section in Thunderbird, so isolating the problem can be much simpler. Unlike setting up an e-mail retrieval for inbound e-mail, with SMTP, a single SMTP setup can be used to cover all of your e-mail addresses, regardless of how many different ISPs are involved.

Most SMTP mail servers are set up to allow e-mail accounts from other ISPs to be run through them. Thunderbird identifies the first SMTP account that you set up as being the default account. That is because in most cases it can be shared by all of your e-mail accounts.

Most of the configuration issues with sending e-mail can be resolved in the Server Settings section in Thunderbird.

- Select Account Settings from the Tools menu.
- Scroll down to the bottom of the list if e-mail accounts.
- Select Outgoing Server (SMTP).

When you are sure that this information is correct, click the OK button, shut down Thunderbird and then start it up again to make sure that any new settings are loaded. If you

have multiple e-mail accounts and you know that inbound e-mail is working, try sending a test message to another of your e-mail addresses. If it still does not work or you see an error, make sure that you Internet access is active. If you can access the Internet using your browser, you should be able to send e-mail messages once the SMTP information is correct. If you are still having trouble connecting to the SMTP server, Thunderbird will usually display a message, but depending on the type of error, sometimes it does not. Most of the time the User Name is just the account name, but some configurations may require a complete e-mail address. If one way does not work, try the other. If your SMTP server requires a secure connection, you may have to check the SSL box. I sometimes use the SMTP server at my AT&T account. AT&T connections usually require that you check the SSL box. If you are having connections problems, post your issues in the Comments below. Try to give us as much information as you can and we will see if we can help you to get connected.

INTERNET EVOLVED AS AN EXPERIMENTAL SYSTEM

THE DOMAIN NAME SYSTEM

The Internet evolved as an experimental system during the 1970s and early 1980s. It then flourished after the TCP/IP protocols were made mandatory on the ARPANET and other networks in January 1983; these protocols thus became the standard for many other networks as well.

Indeed, the Internet grew so rapidly that the existing mechanisms for associating the names of host computers (e.g. UCLA, USC-ISI) to Internet addresses (known as IP addresses) were about to be stretched beyond acceptable engineering limits. Most of the applications in the Internet referred to the target computers by name.

These names had to be translated into Internet addresses before the lower level protocols could be activated to support the application. For a time, a group at SRI International in Menlo Park, CA, called the Network Information Centre (NIC), maintained a simple, machine-readable list of names and associated Internet addresses which was made available on the net. Hosts on the Internet would simply copy this list, usually daily, so as to maintain a local copy of the table. This list was called the "host.txt" file (since it was simply a text file). The list served the function in the Internet that directory services (e.g. 411 or 703-555-1212) do in the US telephone system - the translation of a name into an address.

As the Internet grew, it became harder and harder for the NIC to keep the list current. Anticipating that this problem would only get worse as the network expanded, researchers at USC Information Sciences Institute launched an effort to design a more distributed way of providing this same information. The end result was the Domain Name System (DNS) which allowed hundreds of thousands of "name servers" to maintain small portions of a global database of information associating IP addresses with the names of computers on the Internet.

The naming structure was hierarchical in character. For example, all host computers associated with educational

institutions would have names like "stanford.edu" or "ucla.edu". Specific hosts would have names like "cs.ucla.edu" to refer to a computer in the computer science department of UCLA, for example. A special set of computers called "root servers" maintained information about the names and addresses of other servers that contained more detailed name/address associations. The designers of the DNS also developed seven generic "top level" domains, as follows:

- *Education*: EDU
- *Government*: GOV
- *Military*: MIL
- *International*: INT
- *Network*: NET
- *(non-profit) Organization*: ORG
- *Commercial*: COM

Under this system, for example, the host name "UCLA" became "UCLA.EDU" because it was operated by an educational institution, while the host computer for "BBN" became "BBN.COM" because it was a commercial organization. Top-level domain names also were created for every country: United Kingdom names would end in ".UK," while the ending ".FR" was created for the names of France.

The Domain Name System (DNS) was and continues to be a major element of the Internet architecture, which contributes to its scalability. It also contributes to controversy over trademarks and general rules for the creation and use of domain names, creation of new top-level domains and the like. At the same time, other resolution schemes exist as well. One of the authors (Kahn) has been involved in the development of a different kind of standard

identification and resolution scheme that, for example, is being used as the base technology by book publishers to identify books on the Internet by adapting various identification schemes for use in the Internet environment.

For example, International Standard Book Numbers (ISBNs) can be used as part of the identifiers. The identifiers then resolve to state information about the referenced books, such as location information (e.g. multiple sites) on the Internet that is used to access the books or to order them. These developments are taking place in parallel with the more traditional means of managing Internet resources. They offer an alternative to the existing Domain Name System with enhanced functionality.

The growth of Web servers and users of the Web has been remarkable, but some people are confused about the relationship between the World Wide Web and the Internet. The Internet is the global information system that includes communication capabilities and many high level applications. The Web is one such application.

The existing connectivity of the Internet made it possible for users and servers all over the world to participate in this activity. Electronic mail is another important application. As of today, over 60 million computers take part in the Internet and about 3.6 million web sites were estimated to be accessible on the net. Virtually every user of the net has access to electronic mail and web browsing capability. Email remains a critically important application for most users of the Internet, and these two functions largely dominate the use of the Internet for most users.

PROCESS OF INTERNET STANDARDS

Internet standards were once the output of research activity sponsored by DARPA. The principal investigators on the internetting research effort essentially determined what technical features of the TCP/IP protocols would become common. The initial work in this area started with the joint effort of the two authors, continued in Cerf's group at Stanford, and soon thereafter was joined by engineers and scientists at BBN and University College London. This informal arrangement has changed with time and details can be found elsewhere. At present, the standards efforts for Internet are carried out primarily under the auspices of the Internet Society (ISOC). The Internet Engineering Task Force (IETF) operates under the leadership of its Internet Engineering Steering Group (IESG), which is populated by appointees approved by the Internet Architecture Board (IAB) which is, itself, now part of the Internet Society.

The IETF comprises over one hundred working groups categorized and managed by Area Directors specializing in specific categories. There are other bodies with considerable interest in Internet standards or in standards that must interwork with the Internet.

Examples include the International Telecommunications Union Telecommunicationsstandards group (ITU-T), the International Institute of Electrical and Electronic Engineers (IEEE) local area network standards group (IEEE 801), the Organization for International Standardization (ISO), the American National Standards Institute (ANSI), the World Wide Web Consortium (W3C), and many others.

As Internet access and services are provided by existing media such as telephone, cable and broadcast, interactions with standards bodies and legal structures formed to deal with these media will become an increasingly complex matter. The intertwining of interests is simultaneously fascinating and complicated, and has increased the need for thoughtful cooperation among many interested parties.

MANAGING THE INTERNET

Perhaps the least understood aspect of the Internet is its management. In recent years, this subject has become the subject of intense commercial and international interest, involving multiple governments and commercial organizations, and recently congressional hearings. At issue is how the Internet will be managed in the future, and, in the process, what oversight mechanisms will insure that the public interest is adequately served. In the 1970s, managing the Internet was easy. Since few people knew about the Internet, decisions about almost everything of real policy concern were made in the offices of DARPA. It became clear in the late 1970s, however, that more community involvement in the decision-making processes was essential.

In 1979, DARPA formed the Internet Configuration Control Board (ICCB) to insure that knowledgeable members of the technical community discussed critical issues, educated people outside of DARPA about the issues, and helped others to implement the TCP/IP protocols and gateway functions. At the time, there were no companies that offered turnkey solutions to getting on the Internet. It would be another five years or so before companies like Cisco

Systems were formed, and while there were no PCs yet, the only workstations available were specially built and their software was not generally configured for use with external networks; they were certainly considered expensive at the time.

In 1983, the small group of roughly twelve ICCB members was reconstituted (with some substitutions) as the Internet Activities Board (IAB), and about ten "Task Forces" were established under it to address issues in specific technical areas. The attendees at Internet Working Group meetings were invited to become members of as many of the task forces as they wished.

The management of the Domain Name System offers a kind of microcosm of issues now frequently associated with overall management of the Internet's operation and evolution. Someone had to take responsibility for overseeing the system's general operation. In particular, top-level domain names had to be selected, along with persons or organizations to manage each of them. Rules for the allocation of Internet addresses had to be established. DARPA had previously asked the late Jon Postel of the USC Information Sciences Institute to take on numerous functions related to administration of names, addresses and protocol related matters.

With time, Postel assumed further responsibilities in this general area on his own, and DARPA, which was supporting the effort, gave its tacit approval. This activity was generally referred to as the Internet Assigned Numbers Authority (IANA). In time, Postel became the arbitrator of all controversial matters concerning names and addresses until

his untimely death in October 1998. It is helpful to consider separately the problem of managing the domain name space and the Internet address space. These two vital elements of the Internet architecture have rather different characteristics that colour the manage-ment problems they generate.

Domain names have semantics that numbers may not imply; and thus a means of determining who can use what names is needed. As a result, speculators on Internet names often claim large numbers of them without intent to use them other than to resell them later. Alternate resolution mechanisms, if widely adopted, could significantly change the landscape here.

The rapid growth of the Internet has triggered the design of a new and larger address space (the so-called IP version 6 address space); today's Internet uses IP version 4. However, little momentum has yet developed to deploy IPv6 widely. Despite concerns to the contrary, the IPv4 address space will not be depleted for some time.

Further, the use of Dynamic Host Configuration Protocol (DHCP) to dynamically assign IP addresses has also cut down on demand for dedicated IP addresses. Nevertheless, there is growing recognition in the Internet technical community that expansion of the address space is needed, as is the development of transition schemes that allow interoperation between IPv4 and IPv6 while migrating to IPv6.

In 1998, the Internet Corporation for Assigned Names and Numbers (ICANN) was formed as a private sector, non-profit, organization to oversee the orderly progression in use of Internet names and numbers, as well as certain protocol related matters that required oversight.

The birth of this organization, which was selected by the Department of Commerce for this function, has been difficult, embodying as it does many of the inherent conflicts in resolving discrepancies in this arena. However, there is a clear need for an oversight mechanism for Internet domain names and numbers, separate from their day-to-day management. Many questions about Internet management remain.

They may also prove difficult to resolve quickly. Of specific concern is what role the U.S. government and indeed governments around the world need to play in its continuing operation and evolution. This is clearly a subject for another time.

ADVANTAGES OF INTERNET

There are many advantages of using the Internet, such as:

GLOBAL AUDIENCE

Content published on the World Wide Web is immediately available to a global audience of users. This makes the World Wide Web a very cost-effective medium to publish information. Reaching more than 190 countries.

OPERATES 24 HOURS AND 365 DAYS

You don't need to wait until resources are available to conduct business. From a consumer's perspective as well as a provider's business can be consummated at any time. The fact that the Internet is operational at all times makes it the most efficient business machine to date.

RELATIVELY INEXPENSIVE

It is relatively inexpensive to publish information on the Internet. At a fraction of the cost to publish information by traditional methods, various organizations and individuals can now distribute information to millions of users. It costs only a few thousand dollars to establish an Internet presence and publish content on the Internet.

PRODUCT ADVERTISING

You can use the World Wide Web to advertise various products. Before purchasing a product, customers will be able to look up various product specification sheets and find out additional information. You can use the multimedia capabilities of the World Wide Web to make available not only various product specification sheets but also audio files, images, and even video clips of products in action. The beauty of the Web is that it allows customers to explore products in as much detail as they desire.

If the client just wants a general overview, he or she can look at the advertising information. For those wanting more in depth information, you can provide white papers and product descriptions for download. The Web allows a business to provide timely information, you can simply place the information on the Web page and it is available immediately for your customers.

DISTRIBUTE PRODUCT CATALOGS

The World Wide Web is a very effective medium for distributing product catalogs. In the old days, putting together a product catalog used to be very costly in terms of time and money needed to publish and distribute it.

The World Wide Web changes all this by allowing content developers to put together a sales catalog and make it available to millions of users immediately. Furthermore, unlike printed product catalogs that are usually updated around once a month, product catalogs on the World Wide Web can be updated as needed to respond to various changing market conditions.

ONLINE SURVEYS

Traditional methods of performing surveys are often relatively slow and expensive compared to online surveys conducted on the Internet. For example, in order to fill out various needs of customers or what they would like to see in a future product, it's often necessary to compile a list of address and mail a questionnaire to many customers.

The success of such an attempt is not always guaranteed and can be very costly in terms of mailing the questionnaires and entering responses to a databases and analyzing it. On the other hand, you can use the World Wide Web to automate the whole process. For example, you can set up a CGI script to conduct online surveys. Results of such a survey can be automatically updated to a database. This database can then be used to keep a pulse on various opinions and needs of customers.

ANNOUNCEMENTS

With the World Wide Web, you can distribute various announcements to millions of users in a timely manner. Because there is virtually no time lag from the time it takes to publish information to making the information available to users, the Web is an ideal medium to publicize announce-

ments. As more people discover the virtues of the Web and get connected to the Internet, the Web will become the medium of choice for many organizations and individuals to publicize various announcements.

PROVIDE TECHNICAL SUPPORT

You can also use Web site to provide technical support to users. Because Web pages can be updated immediately with new information, various technical support literature can be immediately modified in light of new findings and developments. This can be accomplished without having to distribute changes to all users affected by any changes using traditional mediums of information distribution, which are often quite costly compared to the World Wide Web.

CREATE ONLINE DISCUSSION FORUMS

By using applications such as WebBoard, it's possible to set up online discussion forums on the Web.

OBTAIN CUSTOMER FEEDBACK

The interactive nature of the World Wide Web is ideal for obtaining customer feedback. You can easily set up a CGI script to obtain customer feedback about a product or service. Because customer feedback submitted by customers can be read immediately, it's possible to respond to various customer concerns in a timely manner, increasing customer satisfaction and quality of customer service.

IMMEDIATE DISTRIBUTION OF INFORMATION

When information is added to a Web site, it's immediately available for browsing by millions of Internet users. The World

Wide Web is an ideal medium of information distribution because it takes away the time lag associated with publishing content and actually making it available to users

EASY INTEGRATION WITH INTERNAL INFORMATION SYSTEMS

Internet information systems deployed on the Internet can be easily integrated with internal information systems managed with office productivity applications such as Microsoft Office.

POWERFUL CONTENT PUBLISHING TOOLS

A new breed of Internet aware applications will start emerging in software stores by the time you read this. These applications will enable users to develop content for the World Wide Web by simply saving as an HTML file.

In addition to software developers making existing applications Internet aware, various new, powerful, and easy-to use Internet content publishing applications are also being developed. These applications will make the task of publishing content on the Internet even easier. Most of these applications are developed for Windows users.

MULTIMEDIA

The capability to incorporate multimedia into Web pages is a major advantage of using World Wide Web to publish information. For example, many Web sites use sounds and video clips to make the content easier and more interesting to browse.

FORMATTING CAPABILITIES

Content published on the World Wide Web can be richly formatted by using various HTML tags and graphic formats.

The capability to do this is a major reason for the success of the World Wide Web. In addition to using HTML tags and various multimedia formats in Web pages, various interactive controls can also be added to a web page.

This capability allows Web site content developers to create "active" Web sites. For example, before a user sends some information to a Web server for processing, a VBScript or JavaScript subroutine can be used to verify information typed in by the user. Various formatting capabilities, along with technologies such as Java and VBScript, make the World Wide Web a richly interactive medium that you can use to distribute information to millions of users.

DISADVANTAGES OF INTERNET

Many fear the Internet because of its disadvantages. They claim to not use the Internet because they are afraid of the possible consequences or are simply not interested. People who have yet connected to the Internet claim they are not missing anything . Today's technological society must realise, it is up to them to protect themselves on the Internet .

Children using the Internet has become a big concern. Most parents do not realise the dangers involved when their children log onto the Internet. When children are online, they can easily be lured into something dangerous. When children talk to others online, they do not realise they could actually be talking to a harmful person. In addition, children may also receive pornography online by mistake; therefore, causing concern among parents everywhere.

Whether surfing the Web, reading newsgroups, or using email, children can be exposed to extremely inappropriate

material. Pornographic sites tend to make sure they are the first sites to be listed in any search area. Some critics say that parents are responsible for their own children on the Internet because there are available services to protect children.

To keep children safe, parents and teachers must be aware of the dangers. They must actively guide and guard their children online.There are a number of tools available today that may help keep the Internet environment safer for children.

Musicians are also concerned with disadvantages to the Net such as, accessibility and freedom. They are upset because the Internet provides their music online at no charge to consumers. File-sharing software, such Kazaa, Emule and may others provides copyrighted songs to all Internet users. The main concern is - the music is free! Musicians feel "they are not getting paid for their work". It is almost impossible to close down all file-sharing services; there are too many of them to count. Another major disadvantage of the Internet is privacy. Electronic messages sent over the Internet can be easily snooped and tracked, revealing who is talking to whom and what they are talking about.

As people surf the Internet, they are constantly giving information to web sites. People should become aware that the collection, selling, or sharing of the information they provide online increases the chances that their information will fall into the wrong hands. When giving personal information on the Internet, people should make sure the Web site is protected with a recognizable security symbol. On the other hand, this does not mean they are fully

protected because anyone may obtain a user's information. Today, not only are humans getting viruses, but computers are also.

Computers are mainly getting these viruses from the Internet; yet, viruses may also be transmitted through floppy disks. However, people should mainly be concerned about receiving viruses from the Internet. Some of these dangerous viruses destroy the computer's entire hard drive, meaning that the user can no longer access the computer. Virus protection is highly recommended. In conclusion, today's society is in the middle of a technological boom. People can either choose to take advantage of this era, or simply let it pass them by. The Internet is a very powerful tool. It has many advantages; however, people need to be extremely aware of the disadvantages as well.

4

Computer Process Architecture

Process architecture is the structural design of general process systems and applies to fields such as computers (software, hardware, networks, etc.), business processes (enterprize architecture, policy and procedures, logistics, project management, etc.), and any other process system of varying degrees of complexity. Processes are defined as having inputs, outputs and the energy required to transform inputs to outputs. Use of energy during transformation also implies a passage of time: a process takes real time to perform its associated action. A process also requires space for input/output objects and transforming objects to exist: a process uses real space. A process system is a specialized system of processes. Processes are composed of processes. Complex processes are made up of several processes that are in turn made up of several processes. This results in an overall structural hierarchy of abstraction. If the process

system is studied hierarchically, it is easier to understand and manage; therefore, process architecture requires the ability to consider process systems hierarchically.

Leading examples of such process architectures include CCS and the ð-calculus. Process systems are a dualistic phenomenon of change/no-change or form/transform and as such, are well-suited to being modelled by the bipartite Petri Nets modelling system and in particular, process-class Dualistic Petri nets where processes can be simulated and studied hierarchically.

SOFTWARE ARCHITECTURE

The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both. The term also refers to documentation of a system's software architecture. Documenting software architecture facilitates communication between stakeholders, documents early decisions about high-level design, and allows reuse of design components and patterns between projects.

OVERVIEW

The field of computer science has come across problems associated with complexity since its formation. Earlier problems of complexity were solved by developers by choosing the right data structures, developing algorithms, and by applying the concept of separation of concerns. Although the term "software architecture" is relatively new to the industry, the fundamental principles of the field have been

applied sporadically by software engineering pioneers since the mid 1980s. Early attempts to capture and explain software architecture of a system were imprecise and disorganized, often characterized by a set of box-and-line diagrams. During the 1990s there was a concentrated effort to define and codify fundamental aspects of the discipline. Initial sets of design patterns, styles, best practices, description languages, and formal logic were developed during that time. The software architecture discipline is centered on the idea of reducing complexity through abstraction and separation of concerns. To date there is still no agreement on the precise definition of the term "software architecture". As a maturing discipline with no clear rules on the right way to build a system, designing software architecture is still a mix of art and science. The "art" aspect of software architecture is because a commercial software system supports some aspect of a business or a mission. How a system supports key business drivers is described via scenarios as non-functional requirements of a system, also known as quality attributes, determine how a system will behave.

Every system is unique due to the nature of the business drivers it supports, as such the degree of quality attributes exhibited by a system such as fault-tolerance, backward compatibility, extensibility, reliability, maintainability, availability, security, usability, and such other –ilities will vary with each implementation. To bring a software architecture user's perspective into the software architecture, it can be said that software architecture gives the direction to take steps and do the tasks involved in each such user's

speciality area and interest e.g. the stakeholders of software systems, the software developer, the software system operational support group, the software maintenance specialists, the deployer, the tester and also the business end user. In this sense software architecture is really the amalgamation of the multiple perspectives a system always embodies. The fact that those several different perspectives can be put together into a software architecture stands as the vindication of the need and justification of creation of software architecture before the software development in a project attains maturity.

HISTORY

The origin of software architecture as a concept was first identified in the research work of Edsger Dijkstra in 1968 and David Parnas in the early 1970s. These scientists emphasized that the structure of a software system matters and getting the structure right is critical. The study of the field increased in popularity since the early 1990s with research work concentrating on architectural styles (patterns), architecture description languages, architecture documentation, and formal methods. Research institutions have played a prominent role in furthering software architecture as a discipline. Mary Shaw and David Garlan of Carnegie Mellon wrote a book titled *Software Architecture: Perspectives on an Emerging Discipline* in 1996, which brought forward the concepts in Software Architecture, such as components, connectors, styles and so on. The University of California, Irvine's Institute for Software Research's efforts in software architecture research is

directed primarily in architectural styles, architecture description languages, and dynamic architectures. The IEEE 1471: ANSI/IEEE 1471-2000: Recommended Practice for Architecture Description of Software-Intensive Systems is the first formal standard in the area of software architecture, and was adopted in 2007 by ISO as *ISO/IEC 42010:2007.*

SOFTWARE ARCHITECTURE TOPICS

ARCHITECTURE DESCRIPTION LANGUAGES

Architecture description languages (ADLs) are used to describe a Software Architecture. Several different ADLs have been developed by different organizations, including AADL (SAE standard), Wright (developed by Carnegie Mellon), Acme (developed by Carnegie Mellon), xADL (developed by UCI), Darwin (developed by Imperial College London), DAOP-ADL (developed by University of Málaga), and ByADL (University of L'Aquila, Italy). Common elements of an ADL are component, connector and configuration.

VIEWS

Software architecture is commonly organized in views, which are analogous to the different types of blueprints made in building architecture. A view is a representation of a set of system components and relationships among them. Within the ontology established by ANSI/IEEE 1471-2000, *views* are responses to *viewpoints*, where a viewpoint is a specification that describes the architecture in question from the perspective of a given set of stakeholders and their concerns. The viewpoint specifies not only the concerns

addressed but the presentation, model kinds used, conventions used and any consistency (correspondence) rules to keep a view consistent with other views. Some possible views (actually, *viewpoints* in the 1471 ontology) are:

- Functional/logic view
- Code/module view
- Development/structural view
- Concurrency/process/runtime/thread view
- Physical/deployment/install view
- User action/feedback view
- Data view/data model

Several languages for describing software architectures ('architecture description language' in ISO/IEC 42010 / IEEE-1471 terminology) have been devised, but no consensus exists on which symbol-set or language should be used to describe each architecture view. The UML is a standard that can be used *"for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes."* Thus, the UML is a visual language that can be used to create software architecture views.

ARCHITECTURE FRAMEWORKS

Frameworks related to the domain of software architecture are:

- 4+1
- RM-ODP (Reference Model of Open Distributed Processing)
- Service-Oriented Modeling Framework (SOMF)

Other architectures such as the Zachman Framework, DODAF, and TOGAF relate to the field of Enterprize architecture.

THE DISTINCTION FROM FUNCTIONAL DESIGN

The IEEE Std 610.12-1990 Standard Glossary of Software Engineering Terminology defines the following distinctions:

- Architectural Design: the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.
- Detailed Design: the process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to begin implementation.
- Functional Design: the process of defining the working relationships among the components of a system.
- Preliminary Design: the process of analyzing design alternatives and defining the architecture, components, interfaces, and timing/sizing estimates for a system or components.

Software architecture, also described as strategic design, is an activity concerned with global requirements governing *how* a solution is implemented such as programming paradigms, architectural styles, component-based software engineering standards, architectural patterns, security, scale, integration, and law-governed regularities. Functional design, also described as tactical design, is an activity concerned with local requirements governing *what* a solution does such as algorithms, design patterns, programming idioms,

refactorings, and low-level implemenation. According to the Intension/Locality Hypothesis, the distinction between architectural and detailed design is defined by the Locality Criterion, according to which a statement about software design is non-local (architectural) if and only if a programme that satisfies it can be expanded into a programme which does not. For example, the client–server style is architectural (strategic) because a programme that is built on this principle can be expanded into a programme which is not client–server; for example, by adding peer-to-peer nodes. Architecture is design but not all design is architectural. In practice, the architect is the one who draws the line between software architecture (architectural design) and detailed design (non-architectural design). There aren't rules or guidelines that fit all cases. Examples of rules or heuristics that architects (or organizations) can establish when they want to distinguish between architecture and detailed design include:

- Architecture is driven by non-functional requirements, while functional design is driven by functional requirements.
- Pseudo-code belongs in the detailed design document.
- UML component, deployment, and package diagrams generally appear in software architecture documents; UML class, object, and behaviour diagrams appear in detailed functional design documents.

EXAMPLES OF ARCHITECTURAL STYLES AND PATTERNS

There are many common ways of designing computer software modules and their communications, among them:

- Blackboard
- Client–server model (2-tier, n-tier, peer-to-peer, cloud computing all use this model)
- Database-centric architecture (broad division can be made for programmes which have database at its center and applications which don't have to rely on databases, E.g. desktop application programmes, utility programmes etc.)
- Distributed computing
- Event-driven architecture
- Front end and back end
- Implicit invocation
- Monolithic application
- Peer-to-peer
- Pipes and filters
- Plug-in (computing)
- Representational State Transfer
- Rule evaluation
- Search-oriented architecture (A pure SOA implements a service for every data access point.)
- Service-oriented architecture
- Shared nothing architecture
- Software componentry
- Space based architecture
- Structured (module-based but usually monolithic within modules)
- Three-tier model (An architecture with Presentation, Business Logic and Database tiers)

SOFTWARE ENGINEERING

Software engineering (SE) is a profession dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build. It is a "systematic approach to the analysis, design, assessment, implementation, test, maintenance and reengineering of software, that is, the application of engineering to software." The term *software engineering* first appeared in the 1968 NATO Software Engineering Conference, and was meant to provoke thought regarding the perceived "software crisis" at the time. The IEEE Computer Society's *Software Engineering Body of Knowledge* defines "software engineering" as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software. It is the application of Engineering to software because it integrates significant mathematics, computer science and practices whose origins are in Engineering. *Software development*, a much used and more generic term, does not necessarily subsume the engineering paradigm. Although it is questionable what impact it has had on actual software development over the last more than 40 years, the field's future looks bright according to Money Magazine and Salary.com, which rated "software engineer" as the best job in the United States in 2006.

HISTORY

When the first modern digital computers appeared in the early 1940s, the instructions to make them operate were

wired into the machine. Practitioners quickly realized that this design was not flexible and came up with the "stored programme architecture" or von Neumann architecture. Thus the first division between "hardware" and "software" began with abstraction being used to deal with the complexity of computing. Programming languages started to appear in the 1950s and this was also another major step in abstraction. Major languages such as Fortran, ALGOL, and COBOL were released in the late 1950s to deal with scientific, algorithmic, and business problems respectively. E.W. Dijkstra wrote his seminal paper, "Go To Statement Considered Harmful", in 1968 and David Parnas introduced the key concept of modularity and information hiding in 1972 to help programmers deal with the ever increasing complexity of software systems. A software system for managing the hardware called an operating system was also introduced, most notably by Unix in 1969. In 1967, the Simula language introduced the object-oriented programming paradigm. These advances in software were met with more advances in computer hardware. In the mid 1970s, the microcomputer was introduced, making it economical for hobbyists to obtain a computer and write software for it.

This in turn led to the now famous Personal Computer (PC) and Microsoft Windows. The Software Development Life Cycle or SDLC was also starting to appear as a consensus for centralized construction of software in the mid 1980s. The late 1970s and early 1980s saw the introduction of several new Simula-inspired object-oriented programming languages, including Smalltalk, Objective-C, and C++. Open-

source software started to appear in the early 90s in the form of Linux and other software introducing the "bazaar" or decentralized style of constructing software. Then the World Wide Web and the popularization of the Internet hit in the mid 90s, changing the engineering of software once again. Distributed systems gained sway as a way to design systems, and the Java programming language was introduced with its own virtual machine as another step in abstraction. Programmers collaborated and wrote the Agile Manifesto, which favoured more lightweight processes to create cheaper and more timely software. The current definition of *software engineering* is still being debated by practitioners today as they struggle to come up with ways to produce software that is "cheaper, better, faster". Cost reduction has been a primary focus of the IT industry since the 1990s. Total cost of ownership represents the costs of more than just acquisition. It includes things like productivity impediments, upkeep efforts, and resources needed to support infrastructure.

PROFESSION

Legal requirements for the licensing or certification of professional software engineers vary around the world. In the UK, the British Computer Society licenses software engineers and members of the society can also become Chartered Engineers (CEng), while in some areas of Canada, such as Alberta, Ontario, and Quebec, software engineers can hold the Professional Engineer (P.Eng)designation and/or the Information Systems Professional (I.S.P.) designation; however, there is no legal requirement to have these

qualifications. The IEEE Computer Society and the ACM, the two main professional organizations of software engineering, publish guides to the profession of software engineering. The IEEE's *Guide to the Software Engineering Body of Knowledge - 2004 Version*, or SWEBOK, defines the field and describes the knowledge the IEEE expects a practicing software engineer to have. The IEEE also promulgates a "Software Engineering Code of Ethics".

EMPLOYMENT

In 2004, the U. S. Bureau of Labor Statistics counted 760,840 software engineers holding jobs in the U.S.; in the same time period there were some 1.4 million practitioners employed in the U.S. in all other engineering disciplines combined. Due to its relative newness as a field of study, formal education in software engineering is often taught as part of a computer science curriculum, and many software engineers hold computer science degrees. Many software engineers work as employees or contractors. Software engineers work with businesses, government agencies (civilian or military), and non-profit organizations. Some software engineers work for themselves as freelancers. Some organizations have specialists to perform each of the tasks in the software development process. Other organizations require software engineers to do many or all of them. In large projects, people may specialize in only one role. In small projects, people may fill several or all roles at the same time. Specializations include: in industry (analysts, architects, developers, testers, technical support, middleware analysts, managers) and in academia (educators,

researchers). Most software engineers and programmers work 40 hours a week, but about 15 percent of software engineers and 11 percent of programmers worked more than 50 hours a week in 2008. Injuries in these occupations are rare. However, like other workers who spend long periods in front of a computer terminal typing at a keyboard, engineers and programmers are susceptible to eyestrain, back discomfort, and hand and wrist problems such as carpal tunnel syndrome.

CERTIFICATION

The Software Engineering Institute offers certifications on specific topics like Security, Process improvement and Software architecture. Apple, IBM, Microsoft and other companies also sponsor their own certification examinations. Many IT certification programmes are oriented toward specific technologies, and managed by the vendors of these technologies. These certification programmes are tailored to the institutions that would employ people who use these technologies. Broader certification of general software engineering skills is available through various professional societies. As of 2006, the IEEE had certified over 575 software professionals as a Certified Software Development Professional (CSDP). In 2008 they added an entry-level certification known as the Certified Software Development Associate (CSDA). In the U.K. the British Computer Society has developed a legally recognized professional certification called *Chartered IT Professional (CITP)*, available to fully qualified Members (*MBCS*). In Canada the Canadian Information Processing Society has developed a legally

recognized professional certification called *Information Systems Professional (ISP)*. The ACM had a professional certification programme in the early 1980s, which was discontinued due to lack of interest. The ACM examined the possibility of professional certification of software engineers in the late 1990s, but eventually decided that such certification was inappropriate for the professional industrial practice of software engineering.

IMPACT OF GLOBALIZATION

The initial impact of outsourcing, and the relatively lower cost of international human resources in developing third world countries led to the dot com bubble burst of the 1990s. This had a negative impact on many aspects of the software engineering profession. For example, some students in the developed world avoid education related to software engineering because of the fear of offshore outsourcing (importing software products or services from other countries) and of being displaced by foreign visa workers. Although statistics do not currently show a threat to software engineering itself; a related career, computer programming does appear to have been affected. Nevertheless, the ability to smartly leverage offshore and near-shore resources via the [follow-the-sun] workflow has improved the overall operational capability of many organizations. When North Americans are leaving work, Asians are just arriving to work. When Asians are leaving work, Europeans are arriving to work. This provides a continuous ability to have human oversight on business-critical processes 24 hours per day, without paying overtime compensation or disrupting key human resource sleep patterns.

EDUCATION

A knowledge of programming is a pre-requisite to becoming a software engineer. In 2004 the IEEE Computer Society produced the SWEBOK, which has been published as ISO/IEC Technical Report 19759:2004, describing the body of knowledge that they believe should be mastered by a graduate software engineer with four years of experience. Many software engineers enter the profession by obtaining a university degree or training at a vocational school. One standard international curriculum for undergraduate software engineering degrees was defined by the CCSE, and updated in 2004. A number of universities have Software Engineering degree programmes; as of 2010, there were 244 Campus programmes, 70 Online programmes, 230 Masters-level programmes, 41 Doctorate-level programmes, and 69 Certificate-level programmes in the United States. In addition to university education, many companies sponsor internships for students wishing to pursue careers in information technology. These internships can introduce the student to interesting real-world tasks that typical software engineers encounter every day. Similar experience can be gained through military service in software engineering.

COMPARISON WITH OTHER DISCIPLINES

Major differences between software engineering and other engineering disciplines, according to some researchers, result from the costs of fabrication.

SUB-DISCIPLINES

Software engineering can be divided into ten subdisciplines. They are:

- Software requirements: The elicitation, analysis, specification, and validation of requirements for software.
- Software architecture: The elicitation, analysis, specification, definition and design, and validation and control of software architecture requirements.
- Software design: The design of software is usually done with Computer-Aided Software Engineering (CASE) tools and use standards for the format, such as the Unified Modeling Language (UML).
- Software development: The construction of software through the use of programming languages.
- Software testing
- Software maintenance: Software systems often have problems and need enhancements for a long time after they are first completed. This subfield deals with those problems.
- Software configuration management: Since software systems are very complex, their configuration (such as versioning and source control) have to be managed in a standardized and structured method.
- Software engineering management: The management of software systems borrows heavily from project management, but there are nuances encountered in software not seen in other management disciplines.
- Software development process: The process of building software is hotly debated among practitioners; some of the better-known processes are the Waterfall Model, the Spiral Model, Iterative and Incremental Development, and Agile Development.

- Software engineering tools, see Computer Aided Software Engineering
- Software quality

RELATED DISCIPLINES

Software engineering is a direct subfield of computer science and has some relations with management science. It is also considered a part of overall systems engineering.

SYSTEMS ENGINEERING

Systems engineers deal primarily with the overall system design, specifically dealing more with physical aspects which include hardware design. Those who choose to specialize in computer hardware engineering may have some training in software engineering.

COMPUTER SOFTWARE ENGINEERS

Computer Software Engineers are usually systems level (software engineering, information systems) computer science or software level computer engineering graduates. This term also includes general computer science graduates with a few years of practical on the job experience involving software engineering.

5

CPU Instructions Design and Architecture

INSTRUCTION

As you can see from the processor block diagrams, the next stage, once an instruction is decoded, is in Athlon's case the Instruction Control Unit. This Unit can hold up to 72 MOps (because a MOp can equal an x86 instruction, this means Athlon can have up to 72 in-flight instructions) before they're dispatched to the schedulers. This is a lot more than the 20 μ-Ops (if you take an average of say 1.5 uOPs per instructions, then the P6 archtecture has approximately 13 in-flight instructions) that can be held in Intel's Reservation Station, which is already the next advantage of Athlon over PIII, but let's not even count that. The next step is where it gets really interesting.

EXECUTION PORTS

You certainly agree that the most important thing a microprocessor has to do is to actually execute the instructions of the software it's running. Thus it's about time that we are getting to this stage.

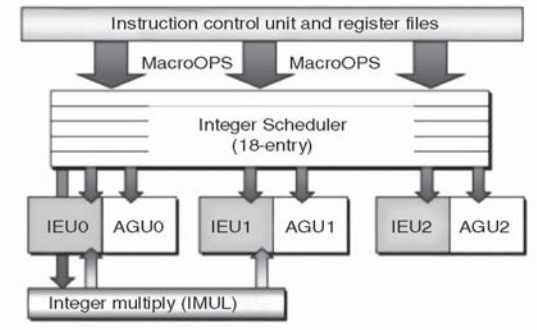


Fig. Athlon's Integer Execution Path.

You cannot really see it in the block diagrams, but Pentium III has 11 (+1) parallel execution units, Athlon has even more. Those units are executing the OPs, and since it's so many in parallel, you can imagine why we are talking of 'out-of-order' execution here. Executing one OP after another would obviously not make any use of parallel execution units. To make sure that the out-of-order execution is actually working, Intel is using the 'Renamer & Allocator' as well as the 'Reorder-Unit'. The 'Integer/FP Renamer/Allocator' is found before the Reservation Station, and as the name already says it, this unit is responsible for integer as well as FP and multimedia OPs. Athlon does this work a bit more sophisticated. The units that take care of the out-of-order execution are the Integer Scheduler and the FP Scheduler, both able to hold a quite impressive number of OPs (18/36).

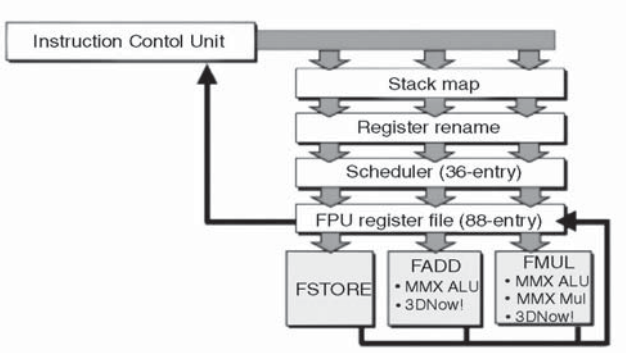
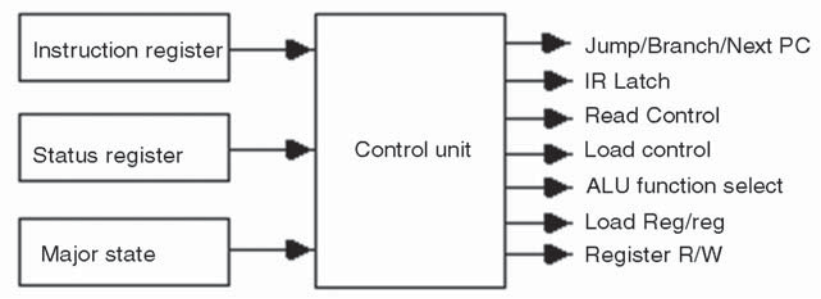


Fig. Athlon's Floating Point and Multimedia Execution Path.

The control unit is the part of a computer that controls the computer's operation. Basically, each part of the computer requires control signals to arrive at particular times for each instruction of the computer's software. The control unit provides those control signals. In modern computers, each of these subsystems may have its own subsidiary controller, but the control unit is the top dog that controls the computer overall.

A control unit for a CISC ISA can ... have to deal with instructions that involve tens and even hundreds of minor states. There would thus be thousands of logic expressions to generate the control signals. CAD tools can simplify and minimize these and even lay them out on the silicon automatically, but it is still a large block of irregular logic on the chip. The control unit is a finite state machine that takes as its inputs the IR, the status register (which is partly filled by the status output from the ALU), and the current major state of the cycle. Its rules are encoded either in random logic, a Programmable Logic Array (PLA), or Read-Only Memory (ROM), and its outputs are sent across the processor to each point requiring coordination or direction for the control unit.

For example the outputs needed for the portion of the instruction/data path shown in Discussion 13 are Jump/ Branch/NextPC, IR Latch, Read Control, Load Control, ALU Function Select, Load/Reg-Reg, Reg R/W.



The ALU function select takes the instruction op code and translates it into a given function of the ALU (either one line per ALU function or a compact binary code for the function). The Jump/Branch/PC depends on the instruction type and in a RISC architecture these may be directly coded in the op code. Read control occurs at the start of an instruction cycle. IR latch and occurs at the end of the fetch state. Load control happens at the end of the data fetch state of a load instruction. Load Reg/Reg again depends on the op-code. Register R/W is in the start of the data fetch stage and at the write back stage of an operation. It thus depends on the major state and the instruction.

A CISC architecture typically uses a more complex control unit. As we've noted before, the IR is often multiple words, and the control unit has to look at different parts of the IR at different stages of execution. In fact, the entire IR may not be available at once, requiring interlocks with fetch logic to ensure the contents of the IR are valid.

There are many more control signals coming out of a CISC control unit, partly to control the more complex addressing

logic, but also to directly connect to the many special purpose registers. In a RISC architecture, the registers are accessed uniformly in a block so a simple decoder in the register file can select the particular register. In a CISC architecture, there are restrictions on the particular registers that can be used by a given instruction and these are enforced by the control unit.

To begin the design of a control unit, we start by listing every control signal in the instruction/data path of the processor. This becomes a list of the control unit's outputs. As input, it has the instruction register, any status information (such as branch flags, interrupts, *etc.*) from the processor, and a "major state" which simply keeps track of where we are in the execution of an instruction. We always begin an instruction with state 0, which corresponds to Fetch. During that state, the control unit outputs the necessary signals to route the contents of the PC to the memory address port, to select and clock the memory until it responds with data from that location, and then cause this data to be latched into the IR.

In a CISC architecture, the Fetch may only retrieve the first part of an instruction, and (depending on bits in the IR that are then decoded by the control unit) more words may need to be fetched. In a RISC architecture, a single Fetch retrieves a complete instruction, so we may proceed to the next major state, which is usually to begin fetching data from the registers, while we decode the instruction.

In a RISC architecture, "decoding an instruction" mainly means that the instruction type field determines what the control unit will do for the remainder of the instructions. If

you think of the CU as a finite state machine, the bits in the type field select the next state following the decode.

In terms of a program's logic, this is like selecting a branch in a Switch statement – each branch of the Switch contains the series of steps to be performed for one type of instruction. For example, after decoding a Jump instruction, the control unit outputs the signals required to combine the address portion of the instruction with the upper bits of the PC and load the result back into the PC. The CU then returns to the Fetch step. Thus, a Jump has three major states (Fetch, Decode, Complete). For a memory Load instruction, the CU first sends one of the selected register values (the address) to the address port of the memory (via a multiplexer) and signals the memory to fetch this location. When the memory returns the value, then the CU sends signals to the necessary multiplexer(s) and the register file so that the memory data goes over the Dest bus and is stored in the designated register. Thus, a Load has four major states (Fetch, Decode, Memory, Write Back).

So, for each type of instruction, and for each major state in each type of instruction, we look at the list of control signals and decide what value each signal must have. In some cases, the value doesn't matter (*e.g.*, if memory isn't selected, it doesn't matter whether it is set to read or write, because it simply won't do anything in either case). You can think of this as a large 2-dimensional table indexed by instruction type and major state. Within each cell of the table is a list of the control signal and their values.

One last bit of control output that we've neglected is the control of the major state itself. This is usually a register, as

shown above, that is input to the CU. But it also receives its next value on each clock from the CU. In the above example, the Jump proceeds from State 0 (Fetch) to State 1 (Decode) to State 3 (Complete) and then goes back to State 0. While a Load adds a State 4. In some designs, the state register also encodes the instruction type. Thus, it is really referring to the different states of the finite state machine (FSM) rather than the major steps of the instructions. So, for example, the FSM states for a Load might be the sequence 0, 1, 12, 13. The latter two distinguish Memory and Write Back from the Complete stage of the Jump. In other designs, we might see Jump going through states 0, 1, 2, and Load going through 0, 1, 2, 3, with the type field used to distinguish the different behaviour of the latter states. This is all just a matter of using somewhat different ways of naming the same things. The important point is just that the CU has the inputs it needs to know what it is supposed to be doing on the present clock and what it will do next. In the CU design process, this translate to ensuring that one of the control signals on the list is the "next state" signal, and that we always specify this in every cell of our table.

CONTROL FOR MULTI CYCLE DATA PATH

The text shows how this data path can be controlled by a finite state machine with just nine states. Unlike the way this is drawn in the text, it is quite obvious here that each level of the controller's finite state machine corresponds to a clock cycle (or major state). This view clearly shows the commonality of the first two major states, prior to the decoding of the instruction. The third stage is a typical

fanning out of the finite state machine to deal with the different cases of the instruction types. Thus, we can see that the machine is arranged in a table where the rows are major states, and the columns are major instruction types. Because the new PC values have been computed, the Jump and Branch types can be finished early. The memory read takes the longest (5 major states).

Each of these states produces a set of control signals that cause the multiplexers to select the appropriate inputs, and the various registers to latch at the proper time. This is determined by looking at each device requiring control and determining for each state whether it is necessary to issue a signal on that particular control line.

Note that in the text, the circuitry controlled by the status from the ALU is shown external to the control for simplicity. But in a more general controller that must handle multiple conditions, this logic would not be separated. Because there are ten distinct states, and the book chooses to encode the instruction type in the major state (really the FSM state number) the state register for this machine must have four bits.

One common approach to implementing a controller is to have the major state be implemented by a counter register and within each of these states the different columns are represented by the state register (which in a RISC architecture could just be the instruction type code). The major state counter simply increments on each clock cycle, and when a column of states finishes early, its last state generates a reset signal to the major state counter. Another issue that is not addressed here is what happens when there is a memory

delay. We are still assuming that memory returns in a single cycle. The simplest case is to stall the finite state machine until the fetch is complete. This is done by having a memory wait signal that is input to the finite state machine. Each memory fetch state has a next state arc that loops back to itself whenever memory wait is asserted.

SIMPLE DATA PATH EXAMPLE: PDP-8

The DEC PDP-8 is a frequently cited example of an almost trivial datapath. We'll quickly take a look at it and note some differences in the implementation approach. The PDP-8 is a 12-bit word machine with a single register (called the Accumulator). It thus falls into the class of single address computers. The majority of the instructions are specified by the upper 3 bits of a 12-bit word.

There are thus 8 major instruction types:

0xxx Logical AND location xxx with Accumulator

1xxx Add location xxx to Accumulator

2xxx Increment location xxx and skip next instruction if the result is 0

3xxx Store Accumulator in location xxx and clear Accumulator

4xxx Subroutine jump to xxx + 1 and store return address in xxx

5xxx Unconditional jump to xxx

6xxx I/O value in Accumulator with device according to xxx

7xxx Subinstruction code specified by xxx

In the DEC scheme, the high order bit is number 0 and the low order bit is 11. Operations 0 through 5 use an

addressing scheme in which bit 3 determines whether the address is direct or indirect, and bit 4 determines whether it refers to an address in the current 128-word "page" or the page that starts at address 0.

The processor has three major states: Fetch, Defer (Indirect address fetch), and Execute. Here is an example of an instruction execution:

1077 — add location 77 directly to the accumulator

Major State 1 (Fetch)

Minor State 1 MAR <- PC

Minor State 2 MBR <- Mem(MAR), PC <- PC + 1

Minor State 3 Latch MBR into IR

Minor State 4 Decode Instruction = Add, Mode = Direct (no defer), Page = 0, Max St, No defer

Major State 2 (Execute)

Minor State 1 MAR <- 00000 + IR6..IR11

Minor State 2 MBR <- Mem(MAR), ALU Function = Add

Minor State 3 Acc <- ALU result, Max St

Now let's look at some of these control lines to see what logic expressions drive them:

No defer = (Major State = 1) AND (IR5 = 0)

Latch MBR = (Major State = 1) AND (Minor State = 3)

Increment PC = (Major State = 1) AND (Minor State = 2)

IR address/MBR = (IR0..IR2 = 4 or 5) AND (IR3 = 1)

Load PC = (Major State = 3) AND (Minor State = 3) AND (IR0..IR2 = 4 or 5)

Memory Data/ALU = (Major State = 3) AND (IR0..IR2 = 3)

MBR Latch = ((Major State = 1) AND (Minor State = 2)(OR)(Major State = 2) AND (Minor State = 2)(OR ((Major State = 3) AND (Minor State = 2) AND (IR0..IR2 < 4))

And so on. Essentially for each of the control signals we identify all of the conditions that could cause it to be asserted and add them to the expression for the given signal.

MICROCODE

In many CISC architectures the control unit can feed back to the major (and minor) states and has internal registers and a ROM. It can thus cause portions of an instruction to be extended or repeated as necessary. The minor states, registers. additional control logic, and ROM form a finite state machine called a microcode engine. In the microcode engine, the op-code from the IR becomes a jump address into the ROM. A micro-PC can be used to step through a series of fetches from the ROM starting at that point, with each fetch resulting in control signals being sent out and providing feedback to the major and minor state values.

An alternative to using a micro-PC is to have each instruction explicitly specify the address of its successor. Thus, one of the fields in the micro-op may be the address of the next instruction. This allows jumps to be used anywhere in the microcode with no time penalty — consider that if a separate instruction had to be employed for a jump, it would add a cycle to the execution of the ISA instruction.

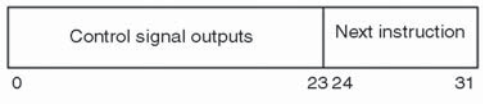
The microcode instruction set can contain a subroutine jump so that common sequences of control outputs can be reused. This is typically present in systems that employ a micro-PC rather than a next instruction field. There is typically just one subroutine return register, so nesting of subroutines is not allowed. Thus, the subroutine jump may be implemented as a normal jump with a signal issued that stores the current micro-PC value into the return register.

And the subroutine ends with a Return operation rather than a jump to another location. The Return operation issues a signal that loads the return register back into the micro-PC. In a system with an explicit next instruction field, the current address plus one is stored into the return register and it is implicit that the next location holds the next instruction following the subroutine call. (An alternative is to have both a subroutine jump address and a next instruction address, and simply return to the same instruction, but this requires the instruction format to be larger than necessary for most operations.

MICROINSTRUCTION FORMAT DESIGN

From the preceding discussion of how sequential execution, jumps, and calls are executed we can gather that the microinstruction needs to include control information for the microengine itself. But what else does it store?

In the simplest microinstruction formats, each bit represents a control signal that is sent out to the datapath. Thus, when a microinstruction is fetched, the bits in the instruction are connected to control wires and cause actions to occur in the datapath. For example, in our PDP-8 example, there are 14 control signals that are driven by a single bit, and three others that each require multiple bits, for a total of 23 bits. Thus, we might use a microinstruction format such as:



Each of the first 20 bits of this microinstruction corresponds to a control signal in the PDP-8. For example,

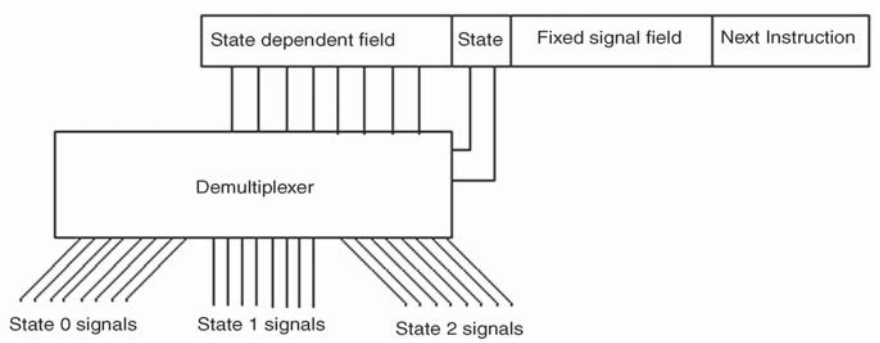
bit 0 might be the Halt, bit 1 the No defer, bit 2 the Max St., bit 3 Acc. Load, bit 4 Acc. clear, *etc.* This simple representation is effective but inefficient. For example, it uses 4 bits to select the ALU function but only one of those bits should be asserted at any time. Thus, we can save some memory by storing the number of the active bit and using an external "decoder" circuit to translate the two bits into the four lines that control the ALU.

In a simple design such as the PDP-8, it may seem that this (two bit) savings is trivial, but in a CISC ISA, the number of control signals can be quite large and it is important to minimize the number of bits in the microinstruction format. Every location in the microcode memory has to store the same set of bits, so any waste of bits is multiplied by the number of words.

Microinstruction format designs are often classified by the width of the word employed. One approach is to have a very wide word that contains all of the control signals necessary to drive the system. Such a design is referred to as a "horizontal" microinstruction format. Another approach is to use a narrower word, with a sequence of microinstructions being required to drive all of the control signals. This is called a "vertical" microinstruction format.

At first glance, it appears that a vertical microinstruction is inherently slower — it takes a sequence of operations to accomplish what the horizontal microinstruction can do in a single cycle. But consider that in many cases, individual control lines are asserted only in certain minor states. If all of these are grouped together by minor state, then they can reuse some of the bits of the microword by having their

outputs first fed to a "demultiplexer" that steers them to the proper signal lines according to the current minor state (which may itself be part of the instruction). In effect the microinstruction format is using multiple instruction formats to reduce redundancy.



Horizontal microcode has been employed in massively parallel array processors where every processor in the system shares a single controller. Often the controller is itself a full-fledged computer, and so the microinstruction both contains traditional machine code for the controller itself as well as the control signals that are distributed to the processors that make up the array. A typical horizontal microinstruction for this type of machine is 128 bits wide (16 bytes). Thus, every effort is made to reduce the number of words required. It is important to note, however, that this is an unusual application of microcode.

One other problem with horizontal microcode is that it is difficult to drive such a large number of signals simultaneously. The switching of so many drivers at once can cause the power supply voltage to sag momentarily (the same as when your lights dim as you turn on a big appliance). This in turn causes noise to appear on signal lines that can cause erroneous behaviour in other parts of the computer.

Avoiding this requires careful circuit design and sometimes clever tricks, such as ensuring that the signals are asserted in a series of slightly offset time steps.

MICROCODE

Of course, the whole reason for using microcode is to manage the complexity of a CISC ISA's control unit. Most RISC designs, even those that have fairly complex implementations, are still sufficiently regular that their control units can be directly constructed from a FSM built with combinational logic. (The advantage of using combinational logic is that it is easier to build a fast decoder with it than with a microcode ROM.) However, for a CISC ISA, the speed decrease resulting from the use of microcode is often outweighted by the need to manage the complexity of controlling the architecture (a slow processor is, after all, more useful than a faster one that doesn't work). In a CISC architecture there may be a large number of instruction types, each with different fields referring to a wide range of registers that have asymmetrical functions, or referring to one or more memory operands with as many as 20 different addressing modes. (The DEC VAX, Intel 80X86/Pentium, and Intel iAPX 432 are prime examples).

A control unit for a CISC ISA can thus have to deal with instructions that involve tens and even hundreds of minor states. There would thus be thousands of logic expressions to generate the control signals. CAD tools can simplify and minimize these and even lay them out on the silicon automatically, but it is still a large block of irregular logic on the chip.

More importantly, if a mistake is discovered later (*i.e.*, one of the logic expressions is wrong), then it may be necessary to resimplify the entire design and lay it out again, which could mean a redesign of the rest of the chip to accommodate a change in the size of the control unit. This is obviously a very costly error. Unfortunately, it is also common. Even with the best design and simulation tools, several commercial chips have gone into production with errors that were discovered later. An early 68000 design had a bug that would cause the processor to hang in certain cases, Intel shipped half a million Pentium processors with an error in the floating-point division instruction before it was caught, and has had bugs in earlier processors.

Since errors do occur, manufacturers of CISC processors use microcode to reduce the cost of correcting the errors (and to help simplify the initial design, which in itself helps to reduce errors). A bug-fix in a microcoded controller is just a matter of changing the ROM, which does not affect the size of the controller at all. It is a very low-cost correction. In addition, a microcoded design is easier to enhance because unused op-codes can be turned into new machine instructions by simply extending the microcode. This simplicity of extension may be another factor that has lead to the increasingly complex ISAs produced by CISC manufacturers.

At one point, it was even thought that allowing the user to add to the microcode was a good idea. If a user has a particular operation that they want to accelerate, they can code it up as a new instruction in the microcode. Such machines were said to have "writeable control stores." The

VAX 11/780, the first model in the VAX line, was one of the most widely sold of these machines. However, the feature was rarely used for two reasons: first, the compilers could not take advantage of the custom instructions so the user had to program in assembly language; second, the microcode is tied to the machine implementation (it refers to the particular control signals in the design) so it is not even portable to another model of the same architecture.

NANOCODE

In some cases, such as the Motorola 68000, there is also a nanocode engine. The 68000 uses 544 17-bit words in its microengine and 336 68-bit words in its nanocode engine. It thus has 32,096 bits of ROM. If everything had been done with 68-bit words, it would have required 36,992 bits. The M68000 microcode is very unusual in that the microcode implicitly calls the nanocode. Each microcode instruction causes a corresponding nanocode engine instruction to be fetched automatically. The nanocode bits are actually the control signals that get distributed across the machine. The microcode instructions thus have only to determine what the next instruction will be. They have two formats, one for an unconditional jump (perhaps just to the next location) or a conditional jump (two bits of the jump address are reserved for the result of the conditional test). This would seem to imply that there are as many nanocode instructions as microcode instructions. Yet we can see that there are 208 fewer.

This is accomplished by carefully assigning the addresses so that common nanocode operations can have multiple

microcode locations corresponding to them. The address space allows for 1024 instructions, and they are arranged so that if a bit (or several) is ignored, the same nanocode address is produced. Essentially the engineers mapped the microcode operations into locations so that certain of the address bits are "don't cares" and all of those locations are then mapped to the same nanocode address. The don't cares are achieved by removing selected transistors in the address decoders of the nanocode ROM.

COMPARISON OF SOME MICROCODE ENGINES

- Motorola 68000: 544 17-bit microwords, plus 336 68-bit nanowords
- DEC LSI-11: 2048 22-bit microwords
- IBM 3033 Mainframe: 2048 108-bit microwords plus 2048 126-bit microwords
- Texas Instruments 8800: 32K 128-bit microwords in user-programmable RAM
- UMass/Hughes IUA-2: 64K 128-bit microwords in RAM.

The STARTECH SD6 tuning kit consists of a computer-controlled auxiliary control unit. Thanks to an included model-specific wire harness and detailed installation instructions the plug-and-play module can be installed in less than 30 minutes by any authorized Chrysler/Jeep dealer.

At one time control units were ad-hoc logic, and they were difficult to design. Now they are designed as a microprogram that is stored in a control store. Words of the microprogram are selected by a sequencer and the bits from those words directly control the different parts of the computer, including

the registers, arithmetic and logic unit, instruction register, bus, input/output and computer storage.

FORMATS

CLAIMS

1. A disk controller for a disk having spirally formed tracks, comprising: Buffer memory for temporarily storing write data to be recorded on said disk or read data derived from said disk, a first register for temporarily storing current sector information representative of a sector currently accessible, said current sector information being changed each time a currently accessible sector varies in accordance with rotation of said disk, a second register for temporarily storing target sector information representative of a sector from which a data read/write operation starts, comparator means for comparing said current sector information with said target sector information and for producing a coincident signal when said current sector information coincides with said target sector information, monitor means coupled to said buffer memory for producing a ready signal when said write data is stored in said buffer memory or when said buffer memory has a vacancy for accepting said read data, and control means coupled to said comparator means and said monitor means for generating a jump back signal when said coincident signal is produced while said ready signal is not being produced, said control means further generating said jump back

signal when said monitor means stops producing said ready signal after said data read/write operation on said disk starts.

2. The disk controller as claimed comprises of: Third register loaded with the current sector information stored in said first register each time said jump back signal is generated, the current sector information stored in said first register being thereafter changed, and means for generating an additional coincident signal when the contents of said first and third registers become equal to each other, said control means generating said jump back signal again when said additional coincident signal is generated while said ready signal is not being produced.
3. The disk controller as claimed in claim 2, further comprising selector means responsive to said control means for selecting the contents of said second or third register for comparing with the contents of said first register, the contents of said first register being selected prior to said jump back signal being generated and the contents of said third register being selected after such time as said jump back signal is generated.
4. The disk controller as claimed in claim 2, wherein said disk controller performs a write-verify operation in which data written on the disk in response to a write command is thereafter read from the disk and the data read out is checked for errors, said control means comprising counter means for causing said jump back signal to be generated a number of times to return the read/write head to a track at which

data recorded in response to said write command began to be recorded.

5. The disk controller as claimed in claim 4, wherein said counter means comprises: Sector register means for temporarily storing a number corresponding to a number of sectors to be subjected to data recording in response to said write command; first down counter means initially loaded with the number temporarily stored in said sector register means for counting down to zero in response to data being recorded in a sector; track register means for temporarily storing a number corresponding to a number of tracks to be backed up upon completion of data recording for a write command; and second down counter means initially loaded with the number temporarily stored in said track register means for counting down to zero in response to said jump back signal being repeatedly generated.
6. In a disk controller for a disk having spirally formed tracks and driven by a servo controller which moves a read/write head to one of said tracks in response to control information supplied by a drive controller, said drive controller being responsive to seek information generated by a system controller from data read and write commands, said disk controller comprising: Buffer memory means for temporarily storing write data to be recorded on said disk or read data derived from said disk, First register means for temporarily storing current sector information representative of a sector currently accessible, said

current sector information being changed each time a currently accessible sector varies in accordance with rotation of said disk, Second register means for temporarily storing target sector information representative of a sector from which a data read/ write operation starts, Comparator means for comparing said current sector information with said target sector information and for producing a coincident output signal taking an active level when said current sector information coincides with said target sector information, Monitor means coupled to said buffer memory for producing a ready signal when said write data is stored in said buffer memory means or when said buffer memory means has a vacancy for accepting said read data.

Control means coupled to said comparator means and said monitor means for generating a jump back signal when said coincident signal is produced while said ready signal is not being produced, said control means further generating said jump back signal when said monitor means stops producing said ready signal after said data read/write operation on said disk starts, said jump back signal being supplied directly to said servo controller without intervention of the system controller and the drive controller, said servo controller responding directly to the jump back signal to perform a jump back operation.

7. A disk controller for a disk apparatus which includes a disk having spirally formed tracks each containing a plurality of sectors, head circuit means for tracing

each track to perform a data recording operation for recording write data supplied thereto to each sector or a data reading-out operation for reproducing read data from each sector, said head circuit means further producing current sector information indicative of a sector currently accessible and updating said current sector information each time the sector currently accessible varies, and a servo control unit for controlling, in response to a jump back signal supplied thereto, said head circuit means such that said head circuit means jumps back from a track currently being traced to a track precedent thereto, said disk controller comprising: Buffer memory means for temporarily storing said write data to be supplied to said head circuit means or said read data reproduced from each sector, memory control unit coupled to said buffer memory for producing a ready signal when said write data is actually stored in said buffer memory or when said buffer memory has a vacancy for accepting said read data, register for temporarily storing target sector information indicative of a target sector from which said data recording operation or said data reading-out operation starts, and a control circuit coupled to said head circuit means, said memory control unit and said register for allowing said head circuit means to start said data recording operation or said data reading-out operation when the current sector information from said head circuit means becomes equal to said target sector information while said ready signal is being produced and for generating and

supplying said jump back signal to said servo control unit to thereby inhibit said head circuit means from starting said data recording operation or said data reading-out operation when the current sector information from said head circuit means becomes equal to said target sector information while said ready signal is not being produced.

8. The disk controller wherein said control circuit includes an additional register loaded with and temporarily storing the current sector information each time said head circuit means updates the current sector information and a comparator for comparing the current sector information stored in said additional register with said target sector information to detect that the current sector information becomes equal to said target sector information.
9. A disk controller for a disk apparatus which includes a disk having spirally formed tracks each containing a plurality of sectors, head circuit means for tracing each track to perform a data recording operation for recording write data supplied thereto on each sector or a data reading-out operation for reproducing read data from each sector, said head circuit means producing current sector information indicative of a sector currently accessible and updating said current sector information each time the sector currently accessible varies, and a servo control unit for controlling, in response to a jump back signal supplied thereto, said head circuit means such that said head circuit means jumps back from a track currently being

traced to a track precedent thereto, said disk controller comprising: A buffer memory for temporarily storing said write data to be supplied to said head circuit means or said read data reproduced from said disk, a memory control unit coupled to said buffer memory for producing a ready signal when said buffer memory actually stores said write data or when said buffer memory has a vacancy for accepting said read data, a register for temporarily storing suspended sector information indicative of a suspended sector at which said data recording operation or said data reading-out operation is suspended, and a control circuit coupled to said head circuit means, said memory control unit and said register for allowing said head circuit means to resume the suspended data recording operation or the suspended reading-out operation when the current sector information from said head circuit means becomes equal to said suspended sector information while said ready signal is being produced and for generating and supplying said jump back signal to said servo control means to thereby inhibit said head circuit means from resuming the suspended data recording operation or the suspended data reading-out operation when the current sector information from said head circuit mean becomes equal to said suspended sector information while said ready signal is not being produced.

10. The disk controller as claimed in claim 9, wherein said control circuit includes an additional register loaded with and temporarily storing the current sector

information each time said head circuit means updates the current sector information and a comparator comparing the current sector information stored in said additional register with said suspended sector information to detect that the current sector information becomes equal to said suspended sector information.

DISK CONTROLLER INCLUDING FORMAT

The present invention relates to a disk controller and, more particularly, to an optical disk controller for an optical disk having a spirally formed track. An optical disk is employed as one of the data storage units in an information processing system. An optical disk controller performs a data transfer operation between the disk and a host processor.

The optical disk controller includes in general a buffer memory for temporarily storing write data from the host processor and read data from the disk, a format control unit for converting the write data from the buffer memory into data to be recorded on the disk and the data reproduced from the disk into the read data, and a system controller for responding to commands from the host processor to control the data transfer flow. Further included in the disk controller are a servo controller for controlling the focus and tracking of an optical beam on the disk and a drive controller for ordering, under the control of the system controller, the servo controller to perform a seek operation in which the optical beam moves to a target track and a jump operation in which the optical beam jumps to the adjacent track.

When the system controller receives a data transfer command from the host processor, it requests the seek

operation of the drive controller, so that the optical beam moves rapidly to the target track. At a time when a target sector on that track is searched, the format controller starts to operate in a data write mode to record the write data from the buffer memory on the disk and in a data read mode to supply the buffer memory with the read data responsive to the data recorded on the disk.

In the data write mode, the write data is transferred from the host processor to the buffer memory, and in the data read mode, the read data is transferred from the buffer memory to the host processor. Thus, the data transfer is executed between the disk and the host processor.

However, sometimes in the data write mode no write data has been transferred to the buffer memory at a time the target sector is searched and that the write data transfer to the buffer memory is suspended. Also in the data read mode, the buffer memory is often filled with the read data which are not transferred to the host processor yet.

In such cases, the data read/write operation is of course suspended until the write data arrives in the buffer memory or until the buffer memory has a vacancy for accepting the read data. On the other hand, the disk continues to rotate. Since the track on the disk is formed spirally, therefore, the accessible sector advances in sequence, so that the target sector is not searched again even when the write data is transferred to the buffer memory or the vacancy for the read data is formed in the buffer memory. Therefore, the format controller informs the system controller of a fact that the data read/write operation is suspended due to the above reason. In response thereto, the system controller requests

the jump operation of the drive controller to back the optical beam up by one track. To back the optical beam up is called hereinafter "jump back". However, this jump back operation is performed after the response time of the system controller and the drive controller has elapsed, resulting in lowering in an access speed. Moreover, the system controller must be designed to handle the request from the format controller, and hence the load thereof is made large.

INVENTION

Therefore, an object of the present invention is to provide an improved disk controller. Another object of the present invention is to provide a disk controller which can prevent lowering the access speed even when a data read/write operation is suspended. Still another object of the present invention is to provide a disk controller which can achieve a jump back processing operation without increasing the loading of the system controller. A disk controller according to the present invention is characterized in that a format control unit generates directly a jump back signal and a servo controller responds directly to that signal to perform a jump back operation.

More specifically, the format control unit comprises a first register for temporarily storing current sector information which indicates a currently accessible sector on a disk and is changed in accordance with the location of the disk, a second register for temporarily storing target sector information indicative of a target sector from which a data read/write operation starts, comparator means for comparing the information stored in the first register with the

information stored in the second register to produce a comparison output signal taking an active level when the former information coincides with the latter information, and a sequence controller receiving the comparison output signal and a ready signal which takes an active level when write data is already stored in a buffer memory or when the buffer memory has a vacancy for accepting read data and generating the jump back signal when the ready signal is in an inactive level when the comparison output signal takes the active level, the sequence controller further generating, after the comparison output signal takes the active level, the jump back signal when the ready signal is in the inactive level when the information stored in the first register is changed.

Thus, the format control unit initiates the jump back operation to prevent lowering the access speed due to the delay of system and drive controllers. When the sequence controller generates the jump back signal, the sector information at that time is temporarily retained, and the comparator means thereafter compares the current sector information with the retained sector information. The generation of the jump back signal is then controlled in response to the comparison output signal and the ready signal.

INSTRUCTION CYCLES AND SUB CYCLES

In order to program at an elementary level, it is not necessary to understand in detail the internal structure of the processor that one is using. However, in order to do efficient programming, such an understanding is required. The purpose of this chapter is to present the basic hardware

concepts necessary for understanding the operation of the Z80 system. The complete microcomputer system includes not only the microprocessor unit (here the Z80), but also other components.

We will review here the basic architecture of the microcomputer system, then study more closely the internal organization of the Z80. We will examine, in particular, the various registers.

We will then study the program execution and sequencing mechanism. From a hardware standpoint, this chapter is only a simplified presentation. The Z80 was designed as a replacement for the Intel 8080, and to offer additional capabilities. A number of references will be made in this chapter to the 8080 design.

SYSTEM ARCHITECTURE

The architecture of the microcomputer system appears in Figure. The microprocessor unit (MPU), which will be a Z80 here, appears on the left of the illustration. It implements the functions of a *central-processing unit* (CPU) within one chip: it includes an *arithmetic-logical unit* (ALU), plus its internal registers, and a*control unit* (CU), in charge of sequencing the system. Its operation will be explained in this chapter.

The MPU creates three *buses*: An 8-bit bidirectional *data bus*, which appears at the top of the illustration, a 16-bit unidirectional *address bus*, and a *control bus*, which appears at the bottom of the illustration. Let us describe the function of each of the buses. The *data bus* carries the data being exchanged by the various elements of the system. Typically, it will carry data from the memory to the MPU or from the

MPU to the memory or from the MPU to an input/output chip. (An input/output chip is a component in charge of communicating with an external device.)

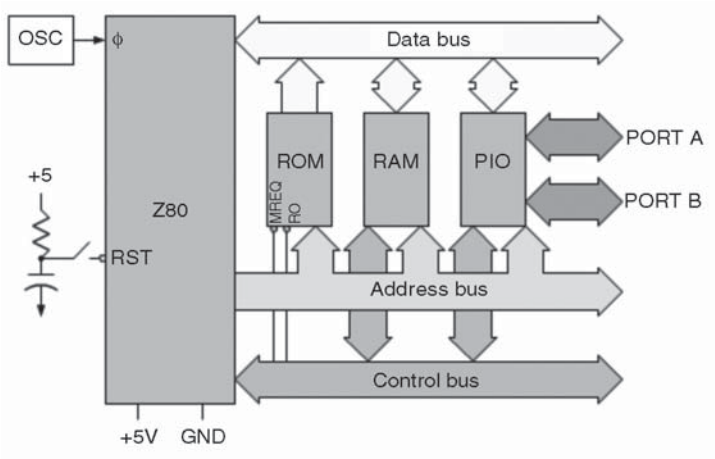


Fig. Standard Z80 System.

The *address bus* carries an address generated by the MPU, which will select one internal register within one of the chips attached to the system. This address specifies the source, or the destination, of the data which will transit along the data bus.

The *control bus* carries the various synchronization signals required by the system. Having described the purpose of the buses, let us now connect the additional components required for a complete system.

Every MPU requires a precise timing reference, which is supplied by a *clock* and a *crystal*. In most "older" microprocessors, the clock-oscilator is external to the MPU and requires an extra chip. In most recent microprocessors, the clock-oscilator is usually incorporated within the MPU. The quartz crystal, however, because of its bulk, is always external to the system. The crystal and the clock appear on the left of the MPU box in Figure.

Let us now turn our attention to the other elements of the system. Going from the left to right on the illustration, we distinguish:

The ROM is the *read-only memory* and contains the *program* for the system. The advantage of the ROM memory is that its contents are permanent and do not disappear whenever the system is turned off. The ROM, therefore, always contains a *bootstrap* or a *monitor* program (their function will be explained later) to permit initial system operation. In a process-control environment, nearly all the programs will reside in ROM, as they will probably never be changed. In such a case, the industrial user has to protect the system against power failure; programs must not be volatile. They must be in ROM.

However, in a hobbyist environment, or in a program-development environment (when the programmer tests his program), most of the programs will reside in RAM, so that they can be easily changed. Later, they may remain in RAM, or be transferred into ROM, if desired. RAM, however, is volatile. Its contents are lost when power is turned off.

The RAM (*random-access memory*) is the read/write memory for the system. In the case of a control system, the amount of RAM will typically be small (for data only). On the other hand, in a program development environment, the amount of RAM will be large, as if will contain programs plus development software. All RAM contents must be loaded prior to use from an external device.

Finally the system will contain one or more interface chips so that it may communicate with the external world. The most frequently used interface chip is the PIO or *parallel*

input/output chip. It is the one shown on the illustration. This PIO, like all other chips in the system, connects to all three buses and provides at least two 8-bit ports for communication with the outside world. All the chips are connected to all three buses, including the control bus. The functional modules which have been described need not necessarily reside on a single LSI chip. In fact, we could use *combination chips*, which may include both PIO and a limited amount of ROM or RAM. Still more components will be required to build a real system. In particular, the buses need to be *buffered*. Also *decoding logic* may be used for the memory RAM chips, and, finally, some signals may need to be amplified by *drivers*. These auxiliary circuits will not be described here as they are not relevant to programming.

INSIDE A MICROPROCESSOR

The large majority of all microprocessor chips on the market today implement the same architecture. This "standard" architecture will be described here. It is shown in Figure below. The modules of this standard microprocessor will now be detailed, from right to left.

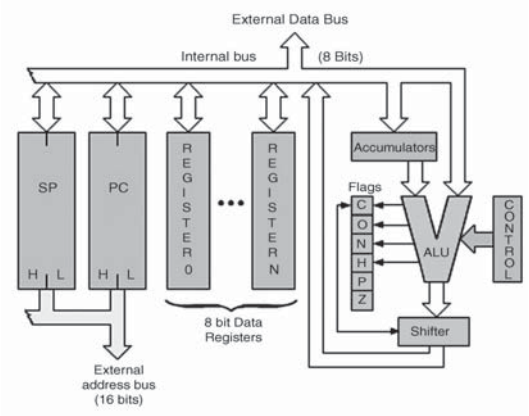


Fig. "Standard" Microprocessor Architecture.

The *control box* on the right represents the control unit which synchronizes the entire system. Its role will be clarified within the remainder of this chapter. The *ALU* performs arithmetic and logic operations. A special register equips one of the inputs of the ALU, the left input here. It is called the accumulator. (Several accumulators may be provided.) The accumulator may be referenced as input and output (source and destination) within the same instruction.

The ALU must also provide *shift* and *rotate* facilities.

A shift operation consists of moving the contents of a byte by one or more positions to the left or to the right. This is illustrated in Figure. Each bit has been moved to the left by one position. The details of shifts and rotations will be presented in the next chapter.

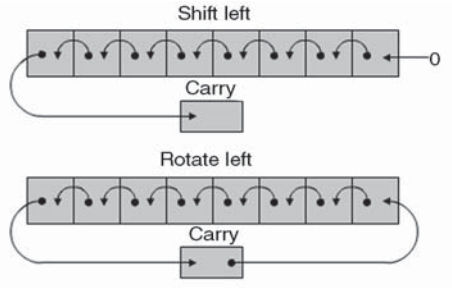


Fig. Shift and Rotate.

The shifter may be on the ALU output, as illustrated in Figure, or may be on the accumulator input. To the left of the ALU, the *flags* or *status register* appear. Their role is to store exceptional conditions within the microprocessor. The contents of the flags registers may be tested by specialized instructions, or may be read on the internal data bus. A *conditional* instruction will cause the execution of a new program, depending on the value of one of these bits. The role of the status bits in the Z80 will be examined later in this chapter.

SETTING FLAGS

Most of the instructions executed by the processor will modify some or all of the flags. It is important to always refer to the chart provided by the manufacturer listing which bits will be modified by the instructions. This is essential in understanding the way a program is being executed. Such a chart for the Z80 is shown in Figure.

THE REGISTERS

Let us look now at above Figure. On the left of the illustration, the registers of the microprocessor appear. Conceptually, one can distinguish the *general-purpose registers* and the *address registers*.

THE GENERAL-PURPOSE REGISTERS

General-purpose registers must be provided in order for the ALU to manipulate data at high speed. Because of restrictions on the number of bits which is reasonable to provide within an instruction, the number of (directly addressable) registers is usually limited to fewer than eight. Each of these registers is a set of eight flip-flops, connected to the bidirectional internal data bus. These eight bits can be transferred simultaneously to or from the data bus. The implementation of these registers in MOS flip-flops provide the fastest level of memory available, and their contents can be accessed within tens of nanoseconds.

Internal registers are usually labeled from 0 to n. The role of these registers is not defined in advance: they are said to be "general-purpose." They may contain any data used by the program.

These general-purpose registers will normally be used to store eight-bit data. On some microprocessors, facilities exist to manipulate two of these registers at a time. They are then called "register pairs." This arrangement facilitates the storage of 16-bit quantities, whether data or addresses.

THE ADDRESS REGISTERS

Address registers are 16-bit registers intended for the storage of addresses. They are also often called *data counters* or *pointers*. They are double registers, *i.e.,* two eight-bit registers. Their essential characteristic is to be connected to the address bus. The address registers create the address bus. The address bus appears on the left and the bottom part of the illustration in Figure below.

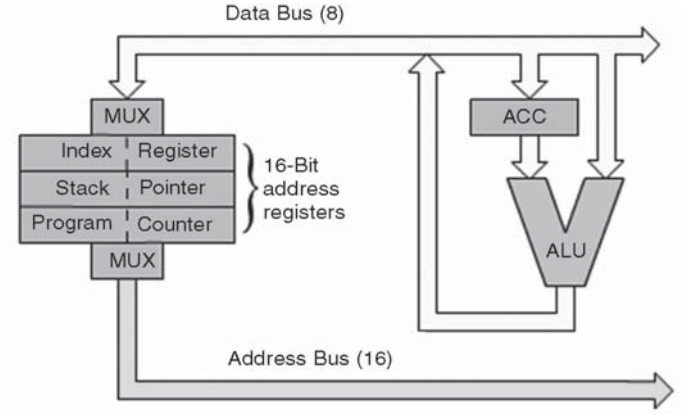


Fig. The 16-bit Address Registers Create the Address Bus.

The only way to load the contents of these 16-bit registers is via the data bus. Two transfers will be necessary along the data bus in order to transfer 16 bits. In order to differentiate between the lower half and the higher half of each register, they are usually labelled as L (low) or H (high), denoting bits 0 through 7, and 8 through 15, respectively. This label is used whenever it is necessary to differentiate the halves of

these registers. At least two address registers are present within most microprocessors. "MUX" in Figure stands for multiplexer.

PROGRAM COUNTER (PC)

The *program counter* must be present in any processor. It contains the address of the next instruction to be executed. The presence of the program counter is indispensable and fundamental to program execution. The mechanism of program execution and the automatic sequencing implemented with the program counter will be described in the next section. Briefly, execution of a program is normally sequential. In order to access the next instruction, it is necessary to bring it from the memory into the microprocessor. The contents of the PC will be deposited on the address bus, and transmitted towards the memory. The memory will then read the contents specified by this address and send back the corresponding word to the MPU. This is the instruction. In a few exceptional microprocessors, such as the two-chip F8, there is no PC on the microprocessor. This does not mean that the system does not have a program counter. The PC happens to be implemented directly on the memory chip, for reasons of efficiency.

STACK POINTER (SP)

The *stack* has not been introduced yet and will be described in the next section. In most powerful, general-purpose microprocessors, the stack is implemented in "software", *i.e.,* within the memory. In order to keep track of the top of this stack within the memory, a 16-bit register is dedicated to the *stack pointer* or *SP*. The SP contains the address of the

top of the stack within the memory. It will be shown that the stack is indispensable for interrupts and for subroutines.

INDEX REGISTER

Indexing is a memory-addressing facility which is not always provided in microprocessors. Indexing is a facility for accessing blocks of data in the memory with a single instruction. An *index register* will typically contain a displacement which will be automatically added to a base (or it might contain a base which would be added to a displacement). In short, indexing is used to access any word within a block of data.

THE STACK

A *stack* is formally called an LIFO structure (last-in, first-out). A stack is a set of registers, or memory locations, allocated to this data structure. The essential characteristic of this structure is that it is a *chronological* structure. This first element introduced into the stack is always at the bottom of the stack. The element most recently deposited in the stack is on top of the stack. The analogy can be drawn with a stack of plates on a restaurant counter. There is a hole in the counter with a spring in the bottom. Plates are piled up in the hole. With this organization, it is guaranteed that the plate which has been put first in the stack (the oldest) is always at the bottom.

The one that has been placed most recently on the stack is the one which is on top of it. This example also illustrates another characteristic of the stack. In normal use, a stack is only accessible via two instructions: “push” and “pop” (or “pull”). The *push* operation results in depositing one element

on top of the stack (two in case of the Z80). The *pull* operation consists of removing one element from the stack. In the case of a microprocessor, it is the *accumulator* that will be deposited on top of the stack. The *pop* will result in a transfer of the top element of the stack into the accumulator. Other specialized instructions may exist to transfer the top of the stack between other specialized registers, such as the status register. The Z80 is more versatile than most in this respect.

The availability of a stack is required to implement three programming facilities within the computer system: subroutines, interrupts, and temporary data storage. Finally, the role of the stack in saving data at high speed will be explained during specific application programs. We will simply assume at this point that the stack is a required facility in every computer system.

A stack may be implemented in two ways:

1. A fixed number of registers may be provided within the microprocessor itself. This is a "hardware stack." It has the advantage of high speed. However, it has the disadvantage of a limited number of registers.
2. Most general-purpose microprocessors choose another approach, the software stack, in order not to restrict the stack to a very small number of registers. This is the approach chosen in the Z80. In the software approach, a dedicated register within the microprocessor, here register SP, stores the stack pointer, *i.e.,* the address of the top element of the stack (or, sometimes, the address of the top element of the stack plus one). The stack is then implemented as an area of memory. The stack pointer will therefore require 16 bits to point anywhere in the memory.

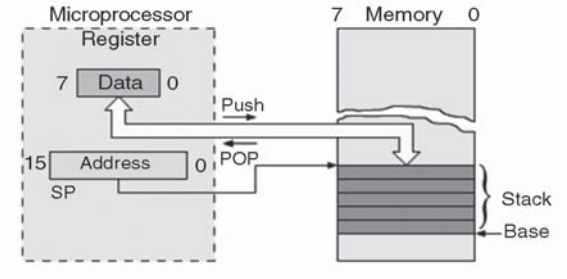


Fig. The Two Stack-Manipulation Instructions.

INSTRUCTION EXECUTION CYCLE

Let us now refer to Figure below:

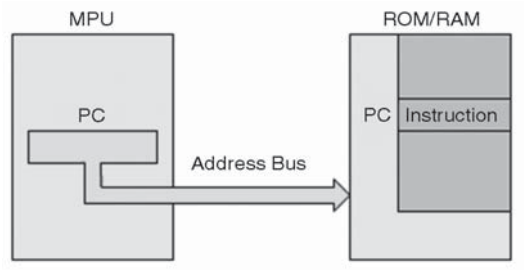


Fig. Fetching an Instruction from the Memory.

The microprocessor unit appears on the left, and the memory appears on the right. The memory chip may be a ROM or a RAM, or any other chip which happens to contain memory. The memory is used to store instructions and data. Here, we will fetch one instruction from the memory to illustrate the role of the program counter. We assume that the program counter has valid contents. It now holds a 16-bit address which is the address of the next instruction to fetch in the memory. Every processor proceeds in three cycles.

FETCH

Let us now follow the sequence. In the first cycle, the contents of the program counter are deposited on the address bus and gated to the memory (on the address bus). Simultaneously, a read signal may be issued on the control

bus of the system, if required. The memory will receive the address. This address is used to specify one location within the memory. Upon receiving the read signal, the memory will decode the address it has received, through internal decoders, and will select the location specified by the address. A few hundred nanoseconds later, the memory will deposit the eight-bit data corresponding to the specified address on its data bus. This eight-bit word is the instruction that we want to fetch. In our illustration, this instruction will be deposited on the data bus on top of the MPU box.

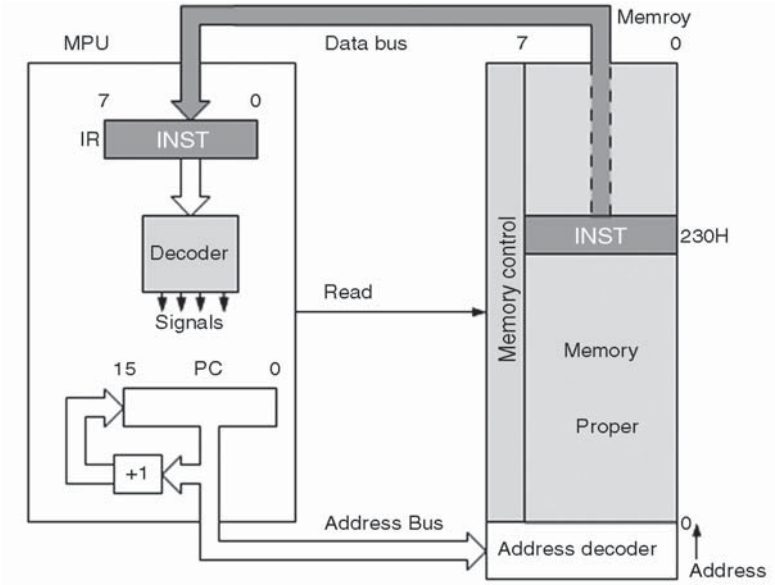


Fig. Automatic Sequencing.

Let us briefly summarize the sequencing: the contents of the program counter are output on the address bus. A read signal is generated. *The memory cycles*, and perhaps 300 nanoseconds later, the instruction at the specified address is deposited on the data bus (assuming a single byte instruction). The microprocessor then reads the data bus and deposits its contents into a specialized internal register, the IR register. The IR is the *instruction register*: It is eight-

bits wide and is used to contain the instruction just fetched from the memory. The fetch cycle is now completed. The 8 bits of the instruction are now physically in the special internal register of the MPU, the IR register. The IR appears on the left of Figure above. It is not accessible to the programmer.

DECODING AND EXECUTION

Once the instruction is contained in IR, the control unit of the microprocessor will decode the contents and will be able to generate the correct sequence of internal and external signals for the execution of the specified instruction. There is, therefore, a short decoding delay followed by an execution phase, the length of which depends on the nature of the instruction specified. Some instructions will execute entirely within the MPU.

Other instructions will fetch or deposit data from or into the memory. This is why the various instructions of the MPU require various length of time to execute. This duration is expressed as a number of (clock) cycles. Since various clock rates may be used, speed of execution is normally expressed in number of cycles rather than in number of nanoseconds.

FETCHING THE NEXT INSTRUCTION

We have described now, using the program counter, an instruction can be fetched from the memory. During the execution of a program, instructions are fetched *in sequence* from the memory. An automatic mechanism must therefore be provided by a simple incrementer attached to the program counter. Every time that the contents of the program counter (at the bottom of the illustration) are placed on the address

bus, its contents will be incremented and written back into the program counter.

As an example, if the program counter contained the value "0", the value "0" would be output on the address bus. Then the contents of the program counter would be incremented and the value "1" would be written back into the program counter. In this way, the next time that the program counter is used, it is the instruction at address 1 that will be fetched. We have just implemented an *automatic mechanism for sequencing instructions*. It must be stressed that the above descriptions are simplified. In reality, some instructions may be two- or even three-bytes long, so that successive bytes will be fetched in this manner from memory. However, the mechanism is identical. The program counter is used to fetch successive bytes of an instruction as well as to fetch successive instructions themselves. The program counter, together with its incrementer, provides an automatic mechanism for pointing to successive memory locations.

We will now execute an instruction within the MPU. A typical instruction will be, for example: R0 = R0 + R1. This means: -ADD the contents of R0 and R1, and store the results in R0." To perform this operation, the contents of R0 will be read from register R0, carried via the single bus to the left input of the ALU, and stored in the buffer register there.

R1 then will be selected and its contents will be read onto the bus, then transferred to the right input of the ALU. This sequence is illustrated in Figures. At this point, the right input of the ALU is conditioned by R1, and the left input of the ALU is conditioned by the buffer register, containing the previous value of R0.

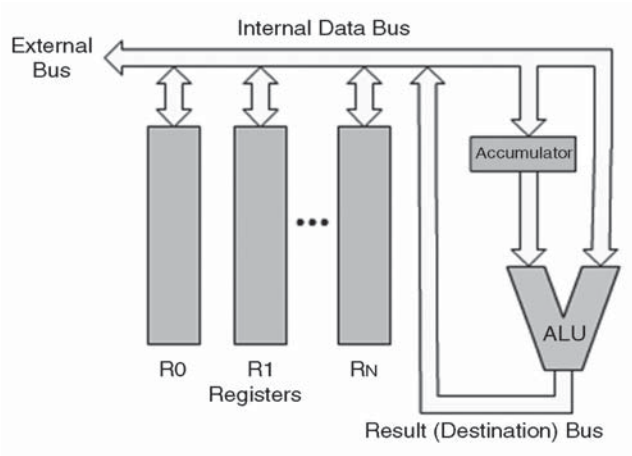


Fig. Single-Bus Architecture.

The operation can be performed. The addition is performed by the ALU, and the result appears on the ALU output, in the lower right-hand corner of Figure. The result will be deposited on the single bus, and will be propagated back to R0. This means, in practice, that the input latch of R0 will be enabled, so that data can be written into it. Execution of the instruction is now complete. The results of the addition are in R0. It should be noted that the contents of R1 have not been modified by this operation. This is general principle: the contents of a register, or any read/write memory, are not modified by a read operation.

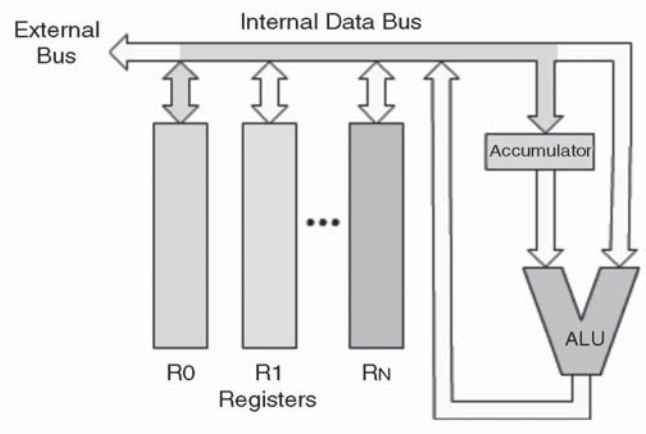


Fig. Execution of an Addition - R0 into ACC.

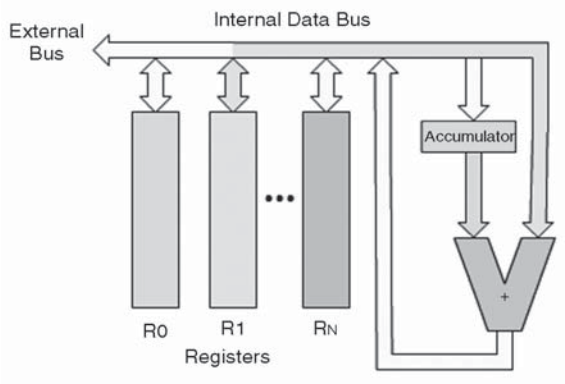


Fig. Addition - Second Register R1 into ALU.

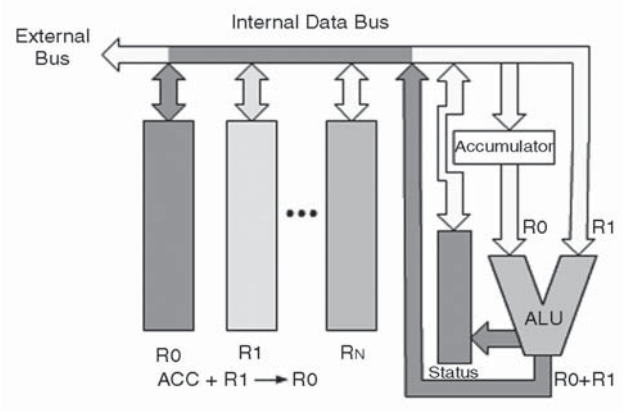


Fig. Result is Generated and Goes into R0.

The buffer register on the left input of the ALU was necessary in order to *memorize* the contents of R0, so that the single bus could be used again for another transfer. However, a problem remains.

CRITICAL RACE PROBLEM

The simple organization shown in Figure above will not function correctly.

Question: What is the timing problem?

Answer: The problem is that the result which will be propagated out of the ALU will be deposited back on the single bus. It will not just propagate in the direction of R0, but along all of the bus. In particular, it will recondition the

right input of the ALU, changing the result coming out of it a few nanoseconds later. This is acritical race. The output of the ALU must be isolated from its input.

Several solutions are possible which will isolate the input of the ALU from the output. A buffer register must be used. The buffer register could be placed on the output of the ALU, or on its input. It is usually placed on the input of the ALU. Here it would be placed on its right input. The buffering of the system is now sufficient for a correct operation. It will be shown later in this chapter that if the left register which appears in this illustration is to be used as an accumulator (permitting the use of one-byte long instructions), then the accumulator will require a buffer too, as shown.

INTERNAL ORGANIZATION OF Z80

The terms necessary in order to understand the internal elements of the microprocessor have been defined. We will now examine in more detail the Z80 itself, and describe its capabilities. The internal organization of the Z80 is shown in figure below. This diagram presents a logical description of the device. Additional interconnections may exist but are not shown. Let us examine the diagram from right to left.

On the right part of the illustration, the *arithmetic-logical unit* (the ALU) may be recognized by its characteristic "V" shape. The accumulator register, which has been described in the previous section, is identified as A on the right input path of the ALU. It has been shown in the previous section that the accumulator should be equipped with a *buffer register*. This is the register labeled ACT (temporary accumulator). Here, the left input of the ALU is also equipped

with a *temporary register*, called TMP. The operation of the ALU will become clear in the next section, where we will describe the execution of actual instructions.

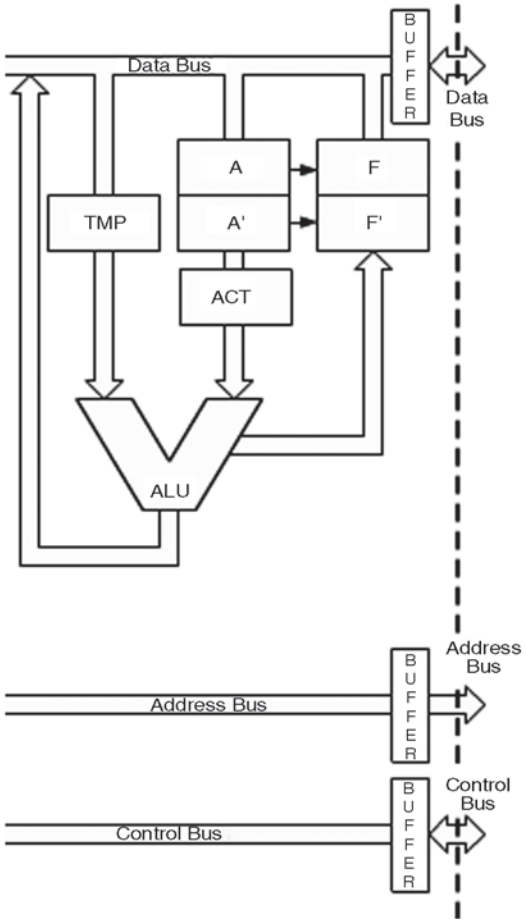


Fig. Right side: Internal Z80 Organization (ALU and connection to the outside world)

The *flags register* is called "F" in the Z80, and is shown on the right of the accumulator register. The contents of the flags register are essentially conditioned by the ALU, but it will be shown that some of its bits may also be conditioned by other modules or events. The accumulator and the flags registers are shown as double registers labeled respectively A, A' and F, F'. This is because the Z80 is equipped internally with two sets of registers A + F, and A' + F'.

However, only *one* set of these registers may be used at any one time. A special instruction is provided to exchange the contents of A and F with A' and F'. In order to simplify the explanations, only A and F will be shown on most of the diagrams which follow. The reader should remember that he has the option of switching to the alternate register set A' and F' if desired.

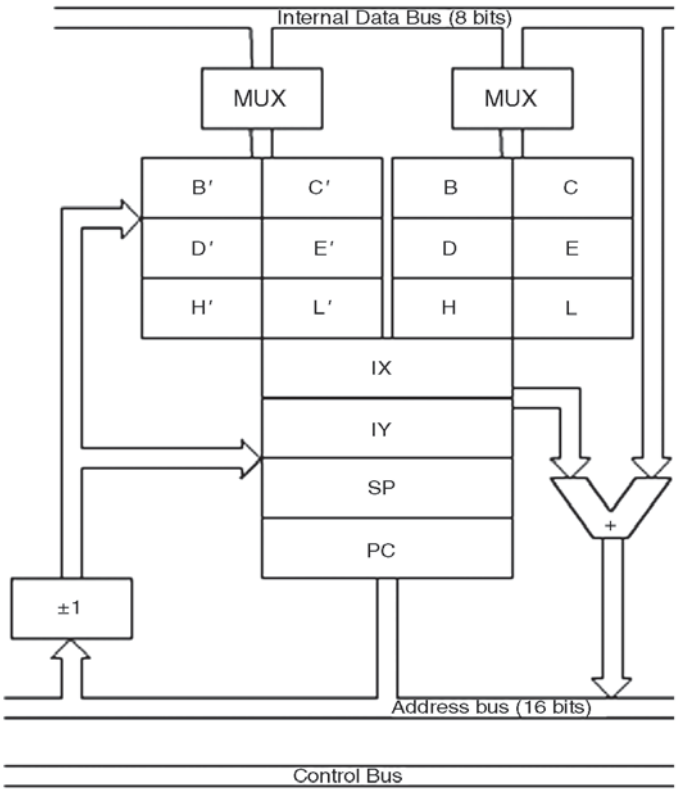


Fig. Center: Internal Z80 Organization (Register Block)

A large block of registers is shown at the center of the illustration. On top of the block of registers, two identical groups can be recognized. Each one includes six registers labeled B, C, D, E, H, L. These are the *general-purpose eight-bit registers* of the Z80. There are two peculiarities of the Z80 with respect to standard microprocessors which has been

described at the beginning of this chapter.

First, the Z80 is equipped with *two* banks of register, *i.e.*, two identical groups of 6 registers. Only six may be used at any one time. However, special instructions are provided to switch between the two banks of registers. One bank, therefore, behaves as an internal memory, while the other one behaves as a working set of internal registers. The possible uses of the special facility will be described in the next chapter. Conceptually, it will be assumed, for the time being, that there are only six working registers, B, C, D, E, H, and L, and the second register bank will temporarily be ignored, in order to avoid confusion.

The MUX symbol which appears above the memory bank is an abbreviation for *multiplexer*. The data coming from the internal data bus will be gated through the multiplexer to the selected register. However, only one of these registers can be connected to the internal data bus at any one time.

A second characteristic of these six registers, in addition to being general-purpose eight-bit registers, is that they are equipped with a connection to the *address bus*. This is why they have been grouped in *pairs*. For example, the contents of B and C can be gated simultaneously onto the 16-bits address bus which appears at the bottom of the illustration. As a result, this group of 6 registers may be used to store either eight-bit data or else 16-bit *pointers* for memory addressing.

The third group of registers, which appears below the two previous ones in the middle of figure above, contains four "pure" address registers. As in any microprocessor, we find the program counter (PC) and the stack pointer (SP). Recall

that the program counter contains the address of the next instruction to be executed. The stack pointer points to the top of the stack in the memory. In the case of the Z80, the stack pointer points to the *last actual entry* in the stack. (In other microprocessors, the stack pointer points just above the last entry.) Also, the stack grows *"downwards" i.e.* towards the lower addresses.

This means that the stack pointer must be *decremented* any time a new word is *pushed* on the stack. Conversely, whenever a word is *removed* (popped) from the stack, the stack pointer must be *incremented* by one. In the case of the Z80, the "push" and "pop" always involve two words at the same time, so that the contents of the stack pointer will be decremented or incremented by two.

Looking at the remaining two registers of this group of four registers, we find a new type of register which has not been described yet: two *index registers*, labeled IX (Index Register X) and IY (Index Register Y). These two registers are equipped with a special adder shown as a miniature V-shaped ALU on the right of these registers in Figure. A byte brought along the internal data bus may be added to the contents of IX or IY. This byte is called the *displacement*, when using an indexed instruction. Special instructions are provided which will automatically add this displacement to the contents of IX or IY and generate an address. This is called *indexing*. It allows convenient access to any sequential block of data.

Finally, a special box labeled "+/-1" appears below and to the left of the block of registers. This is an increment/ decrement. The contents of any of the register pairs SP, PC, BC, DE, HL (the "pure address" registers) may be

automatically incremented or decremented every time they deposit an address on the internal address bus. This is an essential facility for implementing automated *program loops* which will be described in the next section. Using this feature it will be possible to access successive memory locations conveniently.

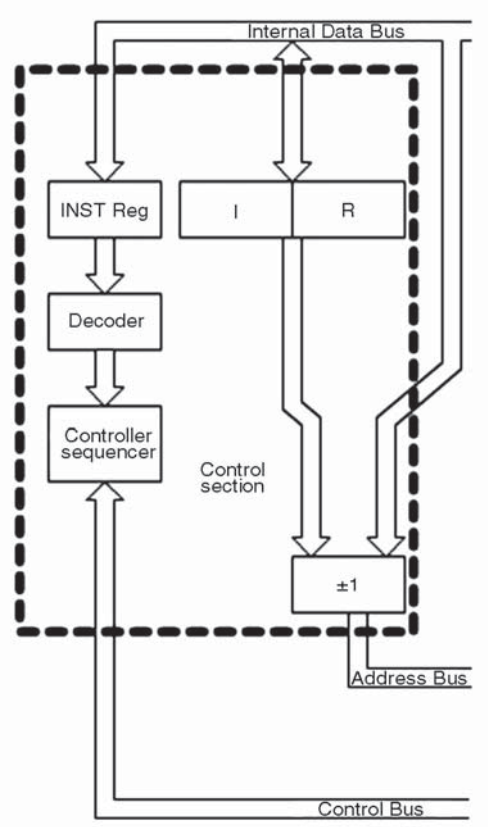


Fig. Left side: Internal Z80 Organization (Control Section)

Let us move to the left side of the illustration. One register pair is shown, isolated on the left: I and R. The I register is called the *interrupt page address register*. Its rol will be described in the section on interrupts of Chapter (Input/ Output Techniques). It is used only in a special mode where an indirect call to a memory location is generated in response to an interrupt. The I register is used to store the high-order

part of the indirect address. The lower part of the address is supplied by the device which generated the interrupt. The R register is the *memory-refresh register*. It is provided to refresh dynamic memories automatically. Such a register has traditionally been located outside the microprocessor, since it is associated with the dynamic memory. It is a convenient feature which minimizes the amount of external hardware for some types of dynamic memories. It will not be used here for programming purposes, as it is essentially a hardware feature. However, it is possible to use it as a software clock, for example.

Let us move now to the far left of the illustration. There the control section of the microprocessor is located. From top to bottom, we find first the *instruction register* IR, which will contain the instruction to be executed. The IR register is totally distinct from the "I, R" register pair described above. The instruction is received from the memory via the data bus, is transmitted along the internal data bus and is finally deposited into the instruction register. Below the instruction register appears the *decoder* which will send signals to the controller-sequencer and cause the execution of the instruction within the microprocessor and outside it. The *control ection* generates and manages the control bus which appears at the bottom part of the illustration.

The three buses managed or generated by the system, *i.e.*, the data bus, the address bus, and the control bus, propagate outside the microprocessor through its pins. The external connections are shown on the right-most part of the illustration. The buses are isolated from the outside through buffers shown in Figure above.

All the logical elements in the Z80 have now been described. It is not essential to understand the detailed operation of the Z80 in order to start writing programs. However, for the programmer who wishes to write efficient codes, the speed of a program and its size will depend upon the correct choice of registers as well as the correct choice of techniques. To make a correct choice, it is necessary to understand how instructions are executed within the microprocessor. We will therefore examine here the execution of typical instructions inside the Z80 to demonstrate the role and use of the internal registers and buses.

INSTRUCTION FORMATS

Z80 instructions may be formated in one, two, three or four bytes. An instruction specifies the operation to be performed by the microprocessor. From a simplified standpoint, every instruction may be represented as an opcode followed by an optional literal or address field, comprising one or two words. The opcode field specifies the operation to be carried out. In strict computer terminology, the opcode represents only those bits which specify the operation to be performed, exclusive of the register pointers which it might incorporate. In the microprocessor world, it is convenient to call opcode the operation code itself, as well as any register pointers which it might incorporate. This "generalized opcode" must reside in an eight-bit word for efficiency (this is the limiting factor on the number of instructions available in a microprocessor).

The 8080 uses instructions which may be one, two, or three bytes long. However, the Z80 is equipped with additional

indexed instructions, which require one more byte. In the case of the Z80, opcodes are, in general, one byte long, except for special instructions which require a two-byte opcode. Some instructions require that one byte of data follow the opcode. In such a case, the instruction will be a two-byte instruction, the second byte of which is data (except for indexing, which adds an extra byte). In other cases, the instruction might require the specification of an address. An address requires 16 bits and, therefore, two bytes. In that case, the instruction will be a three-byte or a four-byte instruction.

For each byte of the instruction, the control unit will have to perform a memory fetch, which will require four clock cycles. The shorter the instruction, the faster the execution.

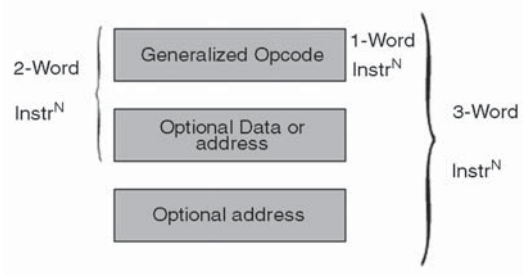


Fig. Typical Instruction Formats.

ONE-WORD INSTRUCTION

One-word instructions are, in principle, fastest and are favored by the programmer. A typical such instruction for the Z80 is:

LD R,R'

This instruction means: "Transfer the contents of register r' into r." This is a typical "register-to-register" operation. Every microprocessor must be equipped with such

instructions, which allow the programmer to transfer information from any of the machine's registers into another one. Instructions referencing special registers of the machine, such as the accumulator or other special-purpose registers, may have a special opcode.

After execution of the above instruction, the contents of r will be equal to the contents of r'. The contents of r' will *not* have been modified by the read operation. Every instruction must be represented internally in a binary format. The above representation "LD r,r' " is symbolic or *mnemonic*. It is called the *assembly-language* representation of an instruction. It is simply meant as a convenient symbolic representation of the actual binary encoding for that instruction. The binary code which will represent this instruction inside the memory is: 0 1 D D D S S S (bits 0 to 7).

This representation is still partially symbolic. Each of the letters S and D stands for a binary bit. The three D's, "D D D", represent the three bits pointing to the *destination* register. Three bits allow selection of one out of eight possible registers. The codes for these registers appear in figure. For example, the code for register B is "0 0 0", the code for register C is "0 0 1", and so on.

Similarly, "S S S" represents the three bits pointing to the *source* register. The convention here is that register r' is the source, and that register r is the destination. The placement of bits in the binary representation of an instruction is not meant for the convenience of the programmer, but for the convenience of the control section of the microprocessor, which must decode and execute the instruction. The *assembly-language* representation, however, is meant for the

convenience of the programmer. It could be argued that LD r,r' should really mean: "Transfer contents of r into r'." However, the convention has been chosen in order to maintain compatibility with the binary representation in this case. It is naturally arbitrary.

Exercise: Write below the binary code which will transfer the contents of register C into register B. Consult Figure for the codes corresponding to C and B.

Code	Register	Code	Register
0 0 0	B	1 0 0	H
0 0 1	C	1 0 1	L
0 1 0	D	1 1 0	- (Memory)
0 1 1	E	1 1 1	A

TWO-WORD INSTRUCTION

ADD A, N

This simple two-word instruction will add the contents of the second byte of the instruction to the accumulator. The contents of the second word of the instruction are said to be a "literal." They are data and are treated as eight bits without any particular significance. They could happen to be a character or numerical data. This is irrelevant to the operation. The code for this instruction is:

1 1 0 0 0 1 1 0 FOLLOWED BY THE 8-BIT BYTE "N"

This is an *immediate* operation. "Immediate," in most programming languages, means that the next word, or words, within the instruction contains a piece of data which should not be *interpreted* (the way an opcode is). It means that the next one or two words are to be treated as a *literal*.

The control unit is programmed to "know" how many words each instruction has. It will, therefore, always fetch and execute the right number of words for each instruction. However, the longer the possible number of words for the instruction, the more complex it is for the control unit to decode.

THREE-WORD INSTRUCTION

LD A, (NN)

The instruction requires three words. It means: "Load the accumulator from the memory address specified in the next two bytes of the instruction." Since addresses are 16-bits long, they require two words. In binary, this instruction is represented by:

0 0 1 1 1 0 1 0: 8 bits for the opcode
Low address: 8 bits for the lower part of the address
High address: 8 bits for the upper part of the address.

EXECUTION OF INSTRUCTIONS IN Z80

We have seen that all instructions are executed in three phases: FETCH, DECODE, EXECUTE. We now need to introduce some definitions. Each of these phases will require several clock cycles. The Z80 executes each phase in one or more logical cycles, called a "machine cycle." The shortest machine cycle lasts three clock cycles. Accessing the memory requires three cycles for any operands, four clock cycles for the initial fetch. Since each instruction must be fetched first from memory, the fastest instruction will require four clock cycles. Most instruction will require more. Each machine cycle is labeled M1, M2, *etc.*, and will require three or more clock cycles, or "states," labeled T1, T2, *etc.*

FETCH PHASE

The FETCH phase of an instruction is implemented during the first three states of machine cycle M1; they are called T1, T2, and T3. These three states are common to all instructions of the microprocessor, as all instructions must be fetched prior to execution. The FETCH mechanism is the following:

T1: PC OUT

The first step is to present the address of the next instruction to the memory. This address is contained in the program counter (PC). As the first step of any instruction fetch, the contents of PC are placed on the address bus. At this point, an address is presented to the memory, and the memory address decoders will decode this address in order to select the appropriate location within the memory. Several hundred ns (a nanosecond is 10^{-9} second) will elapse before the contents of the selected memory location become available on the output pins of the memory, which are connected to the data bus. It is standard computer design to use the memory read time to perform an operation within the microprocessor. The operation is the incrementation of the program counter:

T2: PC = PC + 1

While the memory is reading, the contents of the PC are incremented by 1. At the end of state T2, the contents of the memory are available and can be transferred within the microprocessor:

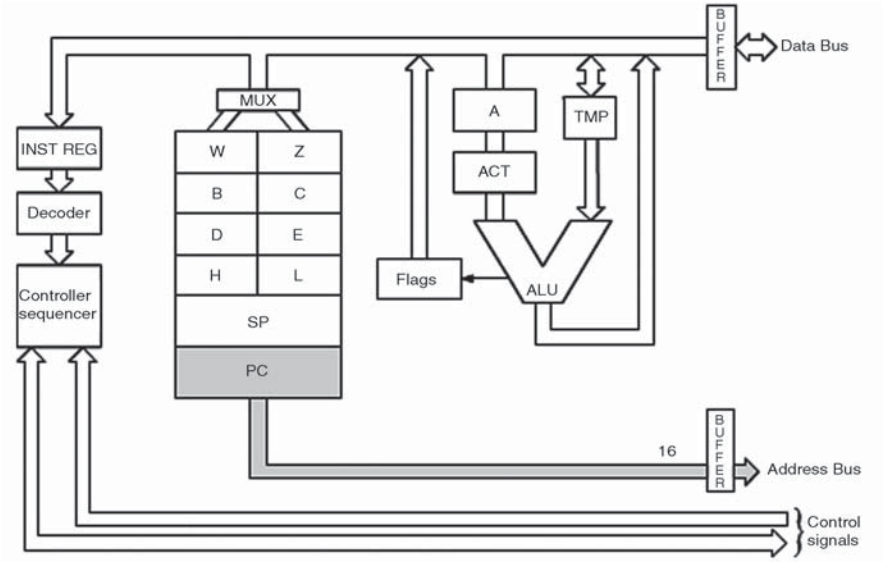


Fig. Instruction Fetch - (PC) is Sent to the Memory.

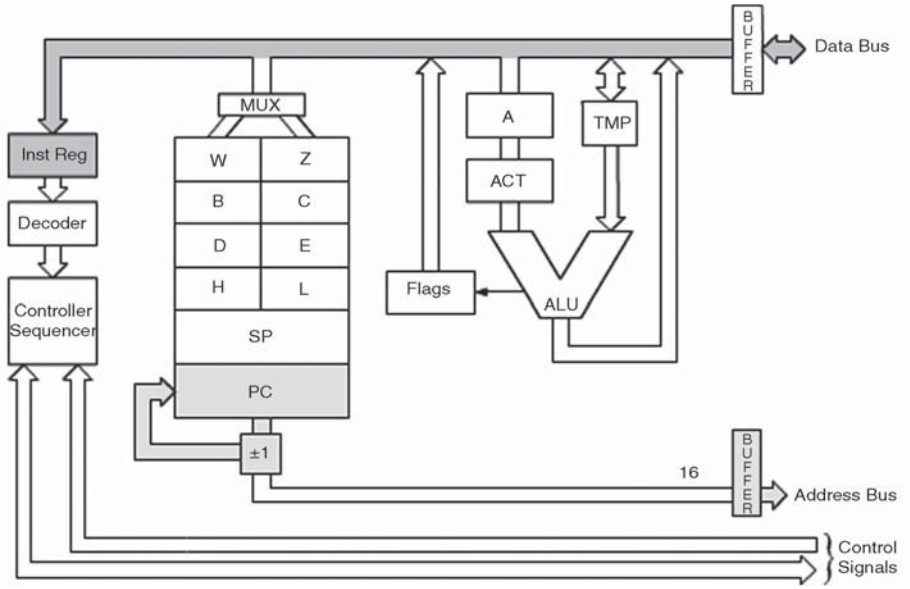


Fig. PC is Incremented.

DECODE AND EXECUTE PHASES

During state T3, the instruction which has been read out of the memory is deposited on the data bus and transferred into the instruction register of the Z80, from which point it is decoded.

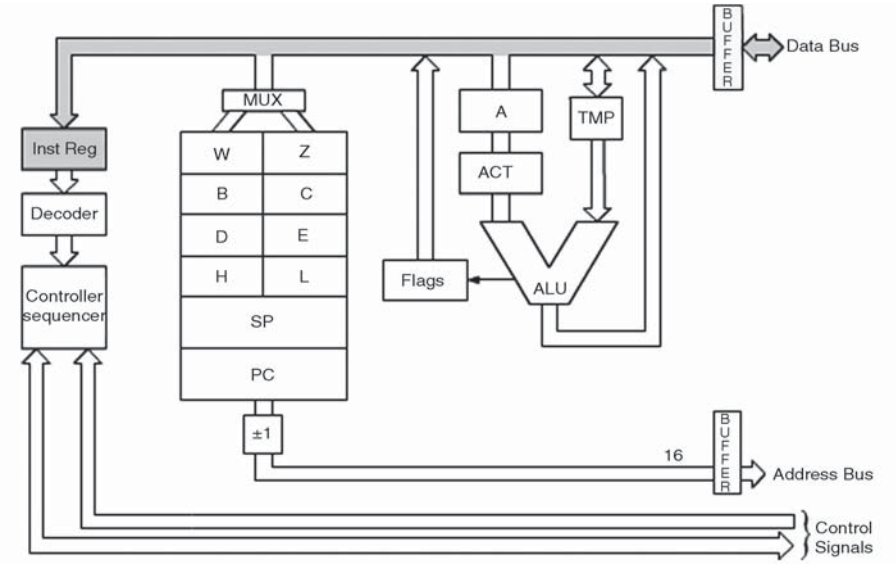


Fig. The Instruction Arrives from the Memory into IR.

It should be noted that states T4 of M1 will always be required. Once the instruction has been deposited into IR during T3, it is necessary to *decode* and *execute* it. This will require at least one machine state, T4.

A few instructions require an extra state of M1 (state T5). It will be skipped by the processor for most instructions. Whenever the execution of an instruction requires more than M1, *i.e.,* M1, M2 or more cycles, the transition will be directly from state T4 of M1 into T1 of M2. Let us examine an example. The detailed internal sequencing for each example is shown in the tables of figure. As these tables have not been released for the Z80, the 8080 tables are used instead. They provide an indepth understanding of the instruction execution.

LD D, C

This corresponds to MOV r1,r2 for the 8080. Refer to item 1 of Figure. By coincidence, the destination register in this example happens to be named "D". The transfer is illustrated

in Figure below. This instruction has been described in the previous section. It transfers the contents of register C, denoted by "C", into register D.

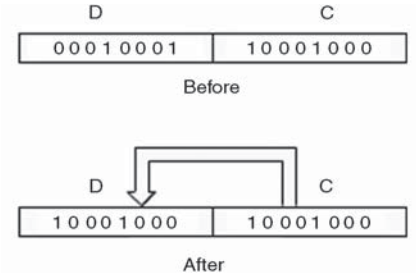


Fig. Transferring C into D.

The first three states of cycle M1 are used to fetch the instruction from the memory. At the end of T3, the instruction is in IR, the Instruction Register, from which point it can be decoded.

During T4: (S S S) →TMP

The contents of C are deposited into TMP.

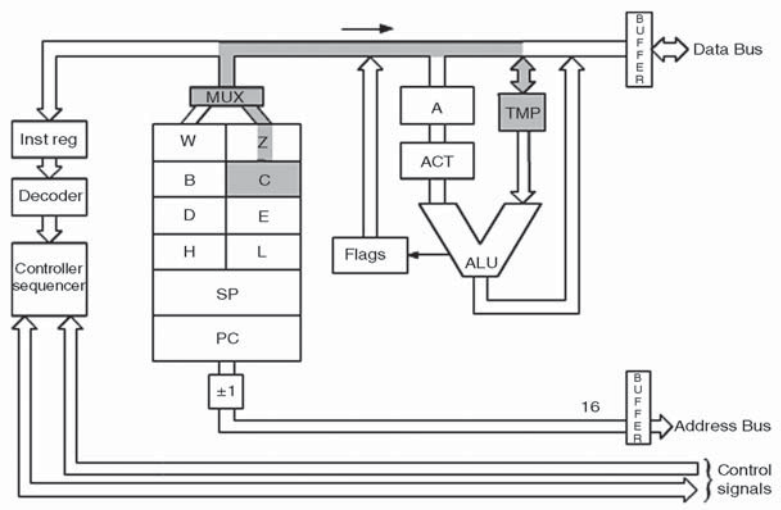


Fig. The Contents of C Are Deposited into TMP

During T5: (TMP) → DDD

The contents of TMP are deposited into D. This is shown in Figure below.

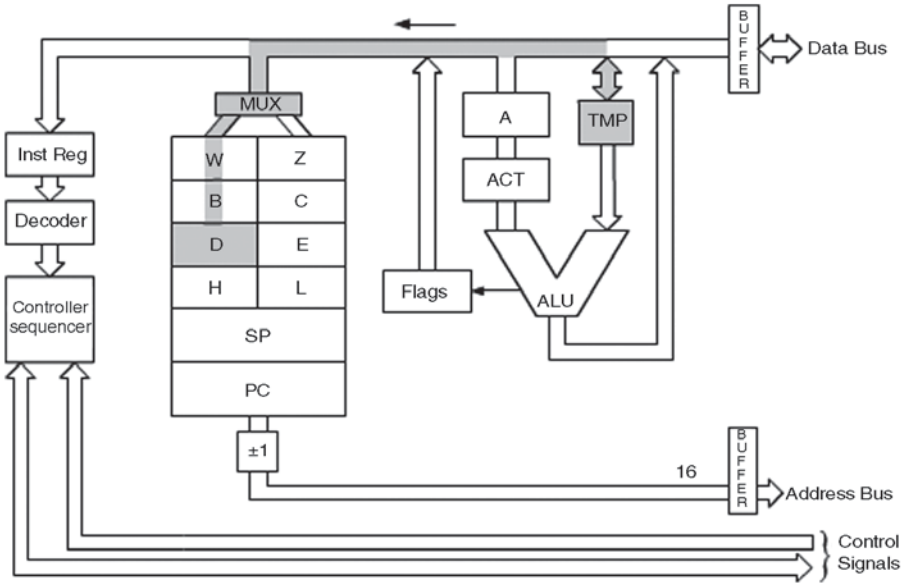


Fig. The Contents of TMP Are Deposited into D.

Execution of the instruction is now complete. The contents of register C have been transferred into the specified destination register D. This terminates execution of the instruction. The other machine cycles M2, M3, M4, and M5 will not be necessary and execution stops with M1. It is possible to compute the duration of this instruction easily. The duration of every state for standard Z80 is the duration of the clock: 500 ns. The duration of this instruction is the duration of five states, or 5 × 500 = 2500 ns = 2.5 us. With a 400 ns clock, 5 × 400 = 2000 ns = 2.0 us.

EXAMPLE

At this point, it is highly recommended that the user review himself the sequencing of this simple instruction before we proceed to more complex ones. For this purpose, go back to Figure above. Assemble a few small-sized "symbols" such as matches, paperclips, *etc.* Then move the symbols on Figure

above to simulate the flow of data from the registers into the buses. For example, deposit a symbol into PC. T1 will move the symbol contained in PC out on the address bus towards the memory. Continue simulated execution in this fashion until you feel comfortable with the transfer along the buses and between the registers. At this point, you should be ready to proceed. Progressively more complex instructions will now be studied:

ADD A, R

This instruction means: "Add the contents of register r (specified by a binary code S S S) to the accumulator (A), and deposit the result in the accumulator." This is an *implicit* instruction. It is called implicit as it does not explicitly reference a second register.

The instruction explicitly refers only to register r. It implies that the other register involved in the operation is the accumulator. The accumulator, when used in such an implicit instruction, is referenced both as source and destination. The advantage of such an implicit instruction is that its complete opcode is only eight bits in length. It requires only a three-bit register field for the specification of r. This is a fast way to perform an addition operation.

Other implicit instructions exist in the system which will reference other specialized registers. More complex examples of such implicit instructions are, for example, the PUSH and POP operations, which will transfer information between the top of the stack and the accumulator, and will at the same time update the stack pointer (SP), decrementing it or incrementing it. They implicitly manipulate the SP register. The execution of the ADD A, r instruction will now be

examined in detail. This instruction will require two machine cycles, M1 and M2. As usual, during the first three states of M1, the instruction is fetched from the memory and deposited in the IR register. At the beginning of T4, it is decoded and can be executed. It will be assumed here that register B is added to the accumulator. The code for the instruction will then be 1 0 0 0 0 0 0 0 (the code for register B is 0 0 0). The 8080 equivalent is ADD r.

T4: (S S S) → TMP, (A) → ACT

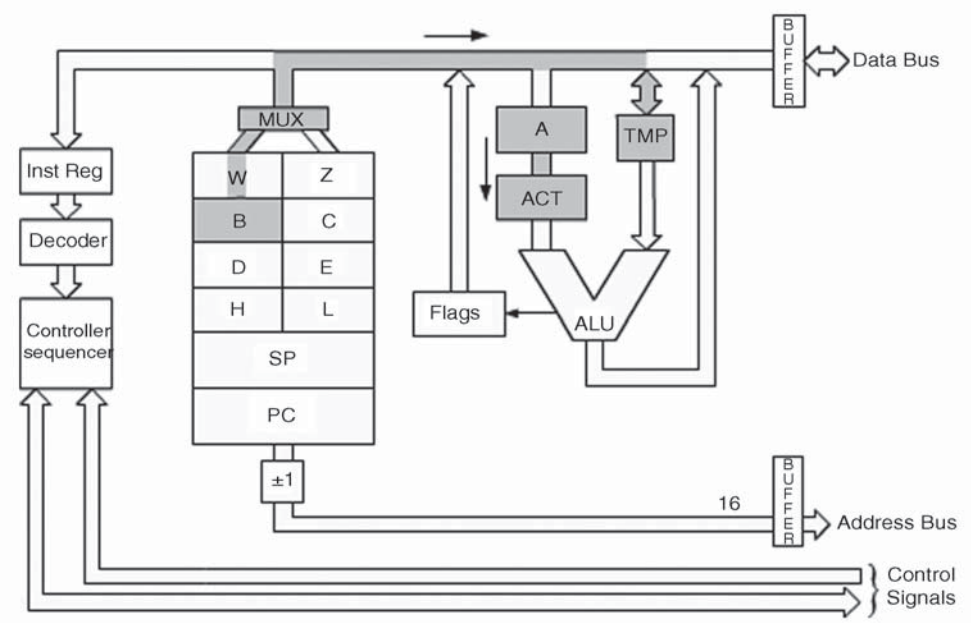


Fig. Two Transfers Occur Simultaneously.

Two transfers will be executed simultaneously. First, the contents of the specified register (here B) are transferred into TMP, *i.e.,* to the right input of the ALU. At the same time, the contents of the accumulator are transferred to the temporary accumulator (ACT).

By inspecting Figure, you will ascertain that those can occur in parallel. They use different paths within the system.

The transfer from B to TMP uses the internal data bus. The transfer from A to ACT uses a short internal path independent of this data bus. In order to gain time, both transfers are done simultaneously. At this point, both the left and the right input of the ALU are correctly conditioned.

The left input of the ALU is now conditioned by the accumulator contents, and the right input of the ALU is conditioned by the contents of register B. We are ready to perform the addition. We would normally expect to see the addition take place during state T5 of M1. However, this state is simply not used. The addition is not performed! We enter machine cycle M2. During state T1, nothing happens! It is only in state T2 of M2 that the addition takes place (refer to ADD r in Figure below):

T2 of M2: (ACT) + (TMP) → A

The contents of ACT are added to the contents of TMP, and the result is finally deposited in the accumulator. See Figure below. The operation is now complete.

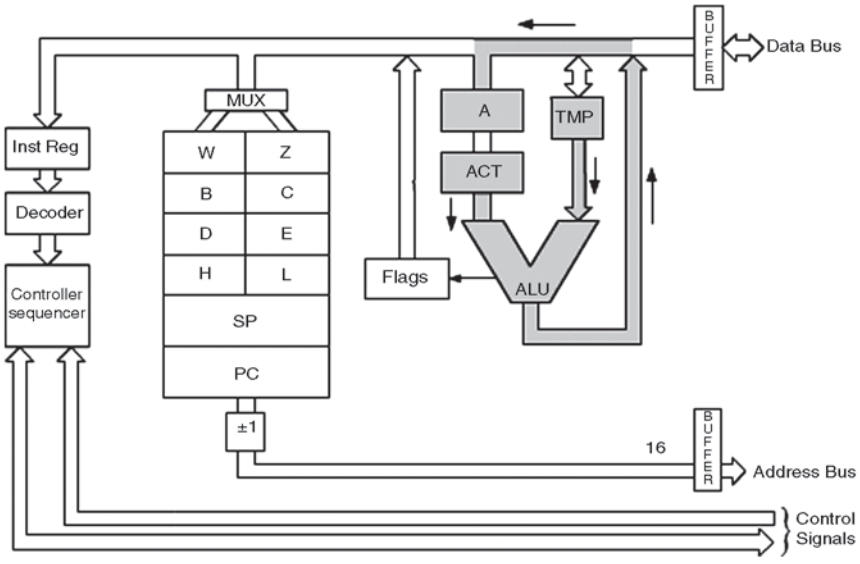


Fig. End of ADD r

Question: Why was the completion of the addition deferred until state T2 of machine cycle M2, rather than taking place during state T5 of M1? (This is a difficult question, which requires an understanding of CPU design. However, the technique involved is fundamental to clock-synchronous CPU design. Try to see what happens.)

Answer: This is a standard design "trick" used in most CPU's. It is called "fetch/execute overlap." The basic idea is the following: looking back at Figure it can be seen that the actual execution of the addition will only require the use of the ALU and of the data bus. In particular, it will not access the register RAM (register block). We (or the control unit) know that the next three states which will be executed after the completion of any instruction will be T1, T2, T3 of machine cycle M1 of the next instruction. Looking back at the execution of these three states, it can be seen that their execution will only require access to the program counter (PC) and the use of the address bus. Access to the program counter will require access to the register RAM. (This explains why the same trick would not be used in the instruction LD r,r'.) It is therefore possible to use simultaneously the shaded area in Figure.

The data bus is used during state T1 of M1 to carry status information out. It cannot be used for the addition that we wish to perform. For that reason, it becomes necessary to wait until state T2 before the addition can be effectively carried out. This is what occurred in the chart: the addition is completed during state T2 of M2. The mechanism has now been explained. The advantage of this approach should now be clear. Let us assume that we had implemented a

straightforward scheme, and performed the addition during state T5 of machine cycle M1.

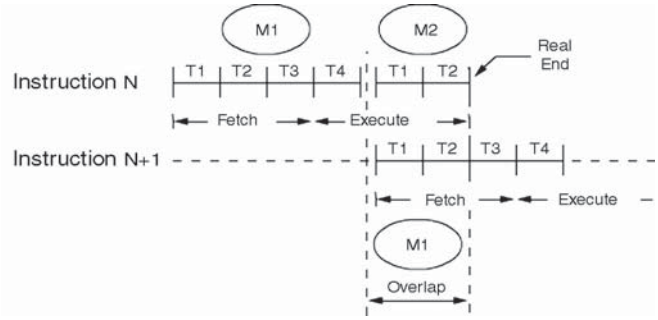


Fig. FETCH-EXECUTE Overlap during T1-T2.

The duration of the ADD instruction would have been 5 × 500 ns = 2500 ns. With the overlap approach which has been implemented, once state T4 has been executed, the next instruction is initiated. In a manner that is invisible to this next instruction, the "clever" control unit will use state T2 to carry out the end of the addition. On the chart T2 is shown as part of M2. Conceptually, M2 will be the second machine cycle of the addition. In fact, this M2 will be overlapped, *i.e.,* be identical to machine cycle M1 of the next instruction. For the programmer, the delay introduced by ADD will be only four states, *i.e.* 4 × 500 = 2000 ns, instead of 2500 ns using the "straightforward" approach. The speed improvement is 500 ns, or 20%! The overlap technique is illustrated in Figure above. It is used whenever possible to increase the appearent execution speed of the microprocessor. Naturally, it is not possible to overlap in all cases. Required buses or facilities must be available without conflict. The control unit "knows" whether an overlap is possible.

Notes:

1. The first memory cycle (M1) is always an instruction fetch; the first (or only) byte, containing the op code, is fetched during this cycle.

2. If the READY input from memory is not high during T2 of each memory cycle, the processor will enter a wait state (TW) until READY is sampled as high.
3. States T4 and T5 are present, as required, for operations which are completely internal to the CPU. The contents of the internal bus during T4 and T5 are available as the data bus; this is designed for testing purposes only. An "X" denotes that the state is present, but only used for such internal operations as instruction decoding.
4. Only register pairs rp = B (registers B and C) or rp = D (registers D and E) may be specified.
5. These states are skipped.
6. Memory read sub-cycles; an instruction or data word will be read.
7. Memory write sub-cycle.
8. The READY signal is not required during the second and third sub-cycles (M2 and M3). The HOLD signal is accepted during M2 and M3. The SYNC signal is not generated during M2 and M3. During the execution of DAD, M2 and M3 are required for an internal register-pair add; memory is not referenced.
9. The results of these arithmetic, logical or rotate instructions are not moved into the accumulator (A) until state T2 of the next instruction cycle. That is, A is loaded while the next instruction is being fetched; this overlapping of operations allows for faster processing.
10. If the value of the least significant 4-bits of the accumulator is greater than 9, or if the auxiliary carry

bit is set, 6 is added to the accumulator. If the value of the most significant 4-bits of the accumulator is now greater than 9, or if the carry bit is set, 6 is added to the most significant 4-bits of the accumulator.

11. This represents the first sub-cycle (the instruction fetch) of the next instruction cycle.
12. If the condition was met, the contents of register pair WZ are output on the address lines (A0-15) instead of the contents of the program counter (PC).
13. If the condition was not met, sub-cycles M4 and M5 are skipped; the processor instead proceeds immediately to the instruction fetch (M1) of the next instruction cycle.
14. If the condition was not met, sub-cycles M2 and M3 are skipped; the processor instead proceeds immediately to the instruction fetch (M1) of the next instruction cycle.
15. Stack read sub-cycle.
16. Stack write sub-cycle.
17. CONDITION CCC

    NZ - not zero (Z = 0)000
    Z - zero (Z = 1) 001
    NC - no carry (CY = 0)010
    C - carry (CY = 1) 011
    PO - parity odd (P = 0)100
    PE - paritty even (P = 1) 101
    P - plus (S = 0) 110
    M - minus (S = 1) 111
18. I/O sub-cycle: The I/O port's 8-bit select code is duplicated on address lines 0-7 (A0-7) and 8-15 (A8-15).

19. Output sub-cycle.
20. The processor will remain idle in the halt state until an interrupt, a reset or a hold is accepted. When a hold request is accepted, the CPU enters the hold mode; after the hold mode is terminated, the processor returns to the halt state. After a reset is accepted, the processor begins executing at memory location zero. After an interrupt is accepted, the processor executes the instruction forced onto the data bus (usually a restart instruction).

SSS or DDD	Value	rp	Value
A	111	B	00
B	000	D	01
C	001	H	10
D	010	SP	11
E	011		
H	100		
L	101		

The following abbreviations are used for operations:

+ addition
- subtraction
^ logical AND
v logical OR
x logical XOR
_ logical NOT (underlined)

Question: Would it be possible to go further using this scheme, and to also use state T3 of M3 if we have to execute a longer instruction?

Answer: No. During T3 of M1 the first byte of an instruction is sent over the internal data bus to the instruction register IR, while most instructions that use T3 in M2, M3, or later, also use the data bus in some way. *E.g.*, LD r,(HL) puts the data received from the memory location "(HL)" into the

general-purpose register "r" using the data bus. In order to clarify the internal sequencing mechanism, it is suggested that you examine Figure above, which shows the detailed instruction execution for the 8080. The Z80 includes all 8080 instructions, and more. The information represented in Figure above is not available for the Z80. It is shown here for its educational value in understanding the internal operation of this microprocessor. The equivalence between Z80 and 8080 instructions is shown in Appendices F and G.