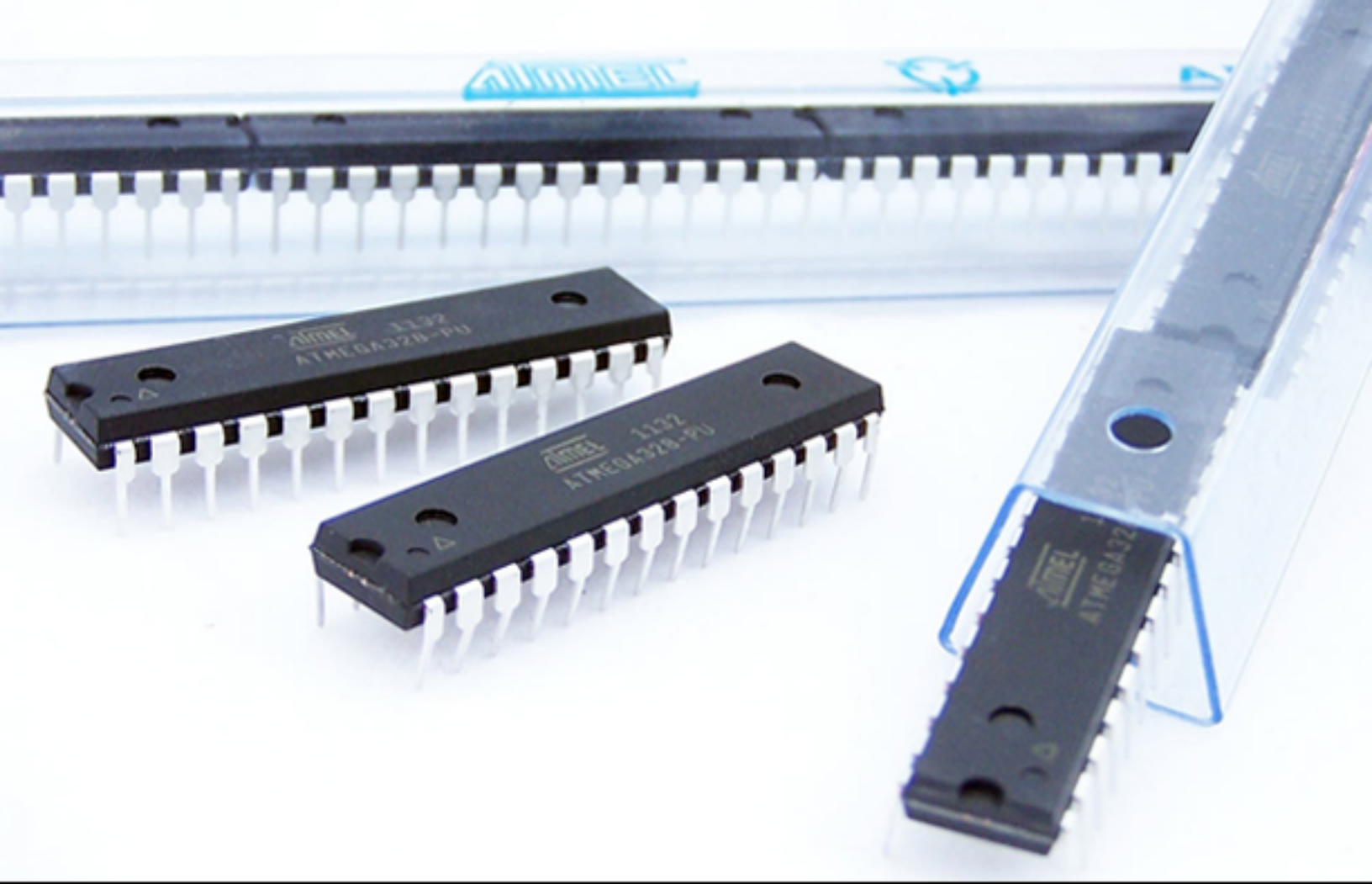


Microcontrollers & Microprocessors

Cooper Perryman

Justina Strunk

First Edition, 2012

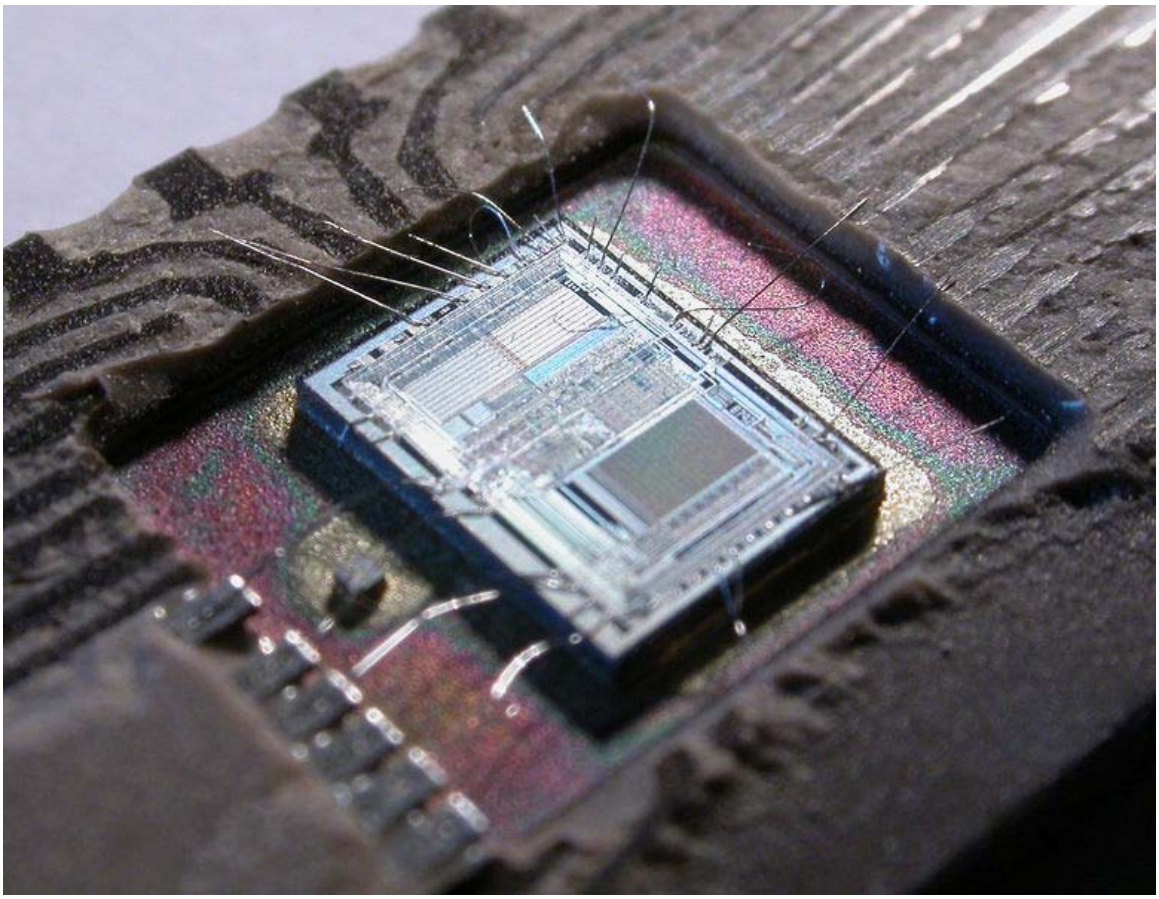
ISBN 978-81-323-0953-6

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

Table of Contents

Chapter-1

Microcontroller



The die from an Intel 8742, an 8-bit microcontroller that includes a CPU running at 12 MHz, 128 bytes of RAM, 2048 bytes of EPROM, and I/O in the same chip.

A **microcontroller** (sometimes abbreviated **µC**, **uC** or **MCU**) is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. 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 embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications.

Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, 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.

Some microcontrollers may use four-bit words and operate at clock rate frequencies as low as 4 kHz, for 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.

Embedded design

A microcontroller can be considered a self-contained system with a processor, memory and peripherals and can be used as an embedded system. The majority of microcontrollers 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.

Programs

Microcontroller programs must fit in the available on-chip program memory, since it would be costly to provide a system with external, expandable, memory. Compilers and assemblers are used to convert high-level language and assembler language codes into a compact machine code for storage in the microcontroller's memory. Depending on the device, the program memory may be permanent, read-only memory that can only be programmed at the factory, or program memory may be field-alterable flash or erasable read-only memory.

Other microcontroller features

Microcontrollers usually contain from several to dozens of general purpose input/output pins (GPIO). GPIO pins are software configurable to either an input or an output state. When GPIO pins are configured to an input state, they are often used to read sensors or external signals. Configured to the output state, GPIO pins can drive external devices such as LEDs or motors.

Many embedded systems need to read sensors that produce analog signals. This is the purpose of the analog-to-digital converter (ADC). Since processors are built to interpret and process digital data, i.e. 1s and 0s, they are not able to do anything with the analog signals that may be sent to it by a device. So the analog to digital converter is used to convert the incoming data into a form that the processor can recognize. A less common feature on some microcontrollers is a digital-to-analog converter (DAC) that allows the processor to output analog signals or voltage levels.

In addition to the converters, many embedded microprocessors include a variety of timers as well. One of the most common types of timers is the Programmable Interval Timer (PIT). A PIT may either count down from some value to zero, or up to the capacity of the count register, overflowing to zero. Once it reaches zero, it sends an interrupt to the processor indicating that it has finished counting. This is useful for devices such as thermostats, which periodically test the temperature around them to see if they need to turn the air conditioner on, the heater on, etc.

Time Processing Unit (TPU) is a sophisticated timer. In addition to counting down, the TPU can detect input events, generate output events, and perform other useful operations.

A dedicated Pulse Width Modulation (PWM) block makes it possible for the CPU to control power converters, resistive loads, motors, etc., without using lots of CPU resources in tight timer loops.

Universal Asynchronous Receiver/Transmitter (UART) block makes it possible to receive and transmit data over a serial line with very little load on the CPU. Dedicated

on-chip hardware also often includes capabilities to communicate with other devices (chips) in digital formats such as I2C and Serial Peripheral Interface (SPI).

Higher integration

In contrast to general-purpose CPUs, micro-controllers may not implement an external address or data bus as they integrate RAM and non-volatile memory on the same chip as the CPU. Using fewer pins, the chip can be placed in a much smaller, cheaper package.

Integrating the memory and other peripherals on a single chip and testing them as a unit increases the cost of that chip, but often results in decreased net cost of the embedded system as a whole. Even if the cost of a CPU that has integrated peripherals is slightly more than the cost of a CPU and external peripherals, having fewer chips typically allows a smaller and cheaper circuit board, and reduces the labor required to assemble and test the circuit board.

A micro-controller is a single integrated circuit, commonly with the following features:

- central processing unit - ranging from small and simple 4-bit processors to complex 32- or 64-bit processors
- volatile memory (RAM) for data storage
- ROM, EPROM, EEPROM or Flash memory for program and operating parameter storage
- discrete input and output bits, allowing control or detection of the logic state of an individual package pin
- serial input/output such as serial ports (UARTs)
- other serial communications interfaces like I²C, Serial Peripheral Interface and Controller Area Network for system interconnect
- peripherals such as timers, event counters, PWM generators, and watchdog
- clock generator - often an oscillator for a quartz timing crystal, resonator or RC circuit
- many include analog-to-digital converters, some include digital-to-analog converters
- in-circuit programming and debugging support

This integration drastically reduces the number of chips and the amount of wiring and circuit board space that would be needed to produce equivalent systems using separate chips. Furthermore, on low pin count devices in particular, each pin may interface to several internal peripherals, with the pin function selected by software. This allows a part to be used in a wider variety of applications than if pins had dedicated functions. Micro-controllers have proved to be highly popular in embedded systems since their introduction in the 1970s.

Some microcontrollers use a Harvard architecture: separate memory buses for instructions and data, allowing accesses to take place concurrently. Where a Harvard architecture is used, instruction words for the processor may be a different bit size than

the length of internal memory and registers; for example: 12-bit instructions used with 8-bit data registers.

The decision of which peripheral to integrate is often difficult. The microcontroller vendors often trade operating frequencies and system design flexibility against time-to-market requirements from their customers and overall lower system cost. Manufacturers have to balance the need to minimize the chip size against additional functionality.

Microcontroller architectures vary widely. Some designs include general-purpose microprocessor cores, with one or more ROM, RAM, or I/O functions integrated onto the package. Other designs are purpose built for control applications. A micro-controller instruction set usually has many instructions intended for bit-wise operations to make control programs more compact. For example, a general purpose processor might require several instructions to test a bit in a register and branch if the bit is set, where a micro-controller could have a single instruction to provide that commonly-required function.

Microcontrollers typically do not have a math coprocessor, so floating point arithmetic is performed by software.

Volumes

About 55% of all CPUs sold in the world are 8-bit microcontrollers and microprocessors. According to Semico, over four billion 8-bit microcontrollers were sold in 2006.

A typical home in a developed country is likely to have only four general-purpose microprocessors but around three dozen microcontrollers. A typical mid-range automobile has as many as 30 or more microcontrollers. They can also be found in many electrical devices such as washing machines, microwave ovens, and telephones.

A PIC 18F8720 **microcontroller** in an 80-pin TQFP package.

Manufacturers have often produced special versions of their microcontrollers in order to help the hardware and software development of the target system. Originally these included EPROM versions that have a "window" on the top of the device through which program memory can be erased by ultraviolet light, ready for reprogramming after a programming ("burn") and test cycle. Since 1998, EPROM versions are rare and have been replaced by EEPROM and flash, which are easier to use (can be erased electronically) and cheaper to manufacture.

Other versions may be available where the ROM is accessed as an external device rather than as internal memory, however these are becoming increasingly rare due to the widespread availability of cheap microcontroller programmers.

The use of field-programmable devices on a microcontroller may allow field update of the firmware or permit late factory revisions to products that have been assembled but not yet shipped. Programmable memory also reduces the lead time required for deployment of a new product.

Where hundreds of thousands of identical devices are required, using parts programmed at the time of manufacture can be an economical option. These "mask programmed" parts have the program laid down in the same way as the logic of the chip, at the same time.

Programming environments

Microcontrollers were originally programmed only in assembly language, but various high-level programming languages are now also in common use to target microcontrollers. These languages are either designed specially for the purpose, or versions of general purpose languages such as the C programming language. Compilers for general purpose languages will typically have some restrictions as well as enhancements to better support the unique characteristics of microcontrollers. Some microcontrollers have environments to aid developing certain types of applications. Microcontroller vendors often make tools freely available to make it easier to adopt their hardware.

Many microcontrollers are so quirky that they effectively require their own non-standard dialects of C, such as SDCC for the 8051, which prevent using standard tools (such as code libraries or static analysis tools) even for code unrelated to hardware features. Interpreters are often used to hide such low level quirks.

Interpreter firmware is also available for some microcontrollers. For example, BASIC on the early microcontrollers Intel 8052; BASIC and FORTH on the Zilog Z8 as well as some modern devices. Typically these interpreters support interactive programming.

Simulators are available for some microcontrollers, such as in Microchip's MPLAB environment and the Revolution Education PICAXE range. These allow a developer to analyze what the behavior of the microcontroller and their program should be if they were using the actual part. A simulator will show the internal processor state and also that of the outputs, as well as allowing input signals to be generated. While on the one hand most simulators will be limited from being unable to simulate much other hardware in a system, they can exercise conditions that may otherwise be hard to reproduce at will in the physical implementation, and can be the quickest way to debug and analyze problems.

Recent microcontrollers are often integrated with on-chip debug circuitry that when accessed by an in-circuit emulator via JTAG, allow debugging of the firmware with a debugger.

Types of microcontrollers

As of 2008 there are several dozen microcontroller architectures and vendors including:

- Parallax Propeller
- Freescale 68HC11 (8-bit)
- Intel 8051
- Silicon Laboratories Pipelined 8051 Microcontrollers
- ARM processors (from many vendors) using ARM7 or Cortex-M3 cores are generally microcontrollers
- STMicroelectronics STM8 (8-bit), ST10 (16-bit) and STM32 (32-bit)

- Atmel AVR (8-bit), AVR32 (32-bit), and AT91SAM (32-bit)
- Freescale ColdFire (32-bit) and S08 (8-bit)
- Hitachi H8, Hitachi SuperH (32-bit)
- Hyperstone E1/E2 (32-bit, First full integration of RISC and DSP on one processor core [1996])
- Infineon Microcontroller: 8, 16, 32 Bit microcontrollers for automotive and industrial applications
- MIPS (32-bit PIC32)
- NEC V850 (32-bit)
- PIC (8-bit PIC16, PIC18, 16-bit dsPIC33 / PIC24)
- PowerPC ISE
- PSoC (Programmable System-on-Chip)
- Rabbit 2000 (8-bit)
- Texas Instruments Microcontrollers MSP430 (16-bit), C2000 (32-bit), and Stellaris (32-bit)
- Toshiba TLCS-870 (8-bit/16-bit)
- XMOS XCore XS1 (32-bit)
- Zilog eZ8 (16-bit), eZ80 (8-bit)

and many others, some of which are used in very narrow range of applications or are more like applications processors than microcontrollers. The microcontroller market is extremely fragmented, with numerous vendors, technologies, and markets. Note that many vendors sell (or have sold) multiple architectures.

Interrupt latency

In contrast to general-purpose computers, microcontrollers used in embedded systems often seek to optimize interrupt latency over instruction throughput. Issues include both reducing the latency, and making it be more predictable (to support real-time control).

When an electronic device causes an interrupt, the intermediate results (registers) have to be saved before the software responsible for handling the interrupt can run. They must also be restored after that software is finished. If there are more registers, this saving and restoring process takes more time, increasing the latency. Ways to reduce such context/restore latency include having relatively few registers in their central processing units (undesirable because it slows down most non-interrupt processing substantially), or at least having the hardware not save them all (this fails if the software then needs to compensate by saving the rest "manually"). Another technique involves spending silicon gates on "shadow registers": one or more duplicate registers used only by the interrupt software, perhaps supporting a dedicated stack.

Other factors affecting interrupt latency include:

- Cycles needed to complete current CPU activities. To minimize those costs, microcontrollers tend to have short pipelines (often three instructions or less), small write buffers, and ensure that longer instructions are continuable or

restartable. RISC design principles ensure that most instructions take the same number of cycles, helping avoid the need for most such continuation/restart logic.

- The length of any critical section that needs to be interrupted. Entry to a critical section restricts concurrent data structure access. When a data structure must be accessed by an interrupt handler, the critical section must block that interrupt. Accordingly, interrupt latency is increased by however long that interrupt is blocked. When there are hard external constraints on system latency, developers often need tools to measure interrupt latencies and track down which critical sections cause slowdowns.
    - One common technique just blocks all interrupts for the duration of the critical section. This is easy to implement, but sometimes critical sections get uncomfortably long.
    - A more complex technique just blocks the interrupts that may trigger access to that data structure. This often based on interrupt priorities, which tend to not correspond well to the relevant system data structures. Accordingly, this technique is used mostly in very constrained environments.
    - Processors may have hardware support for some critical sections. Examples include supporting atomic access to bits or bytes within a word, or other atomic access primitives like the LDREX/STREX exclusive access primitives introduced in the ARMv6 architecture.
- Interrupt nesting. Some microcontrollers allow higher priority interrupts to interrupt lower priority ones. This allows software to manage latency by giving time-critical interrupts higher priority (and thus lower and more predictable latency) than less-critical ones.
- Trigger rate. When interrupts occur back-to-back, microcontrollers may avoid an extra context save/restore cycle by a form of tail call optimization.

Lower end microcontrollers tend to support fewer interrupt latency controls than higher end ones.

History

The first single-chip microprocessor was the 4-bit Intel 4004 released in 1971. With the Intel 8008 and more capable microprocessors available over the next several years.

These however all required external chip(s) to implement a working system, raising total system cost, and making it impossible to economically computerize appliances.

The first computer system on a chip optimized for control applications was the Intel 8048, released in 1975, with both RAM and ROM on the same chip. This chip would find its way into over one billion PC keyboards, and other numerous applications. At this time Intels President, Luke J. Valenter, stated that the (Microcontroller) was one of the most successful in the companies history, and expanded the division's budget over 25%.

Most microcontrollers at this time had two variants. One had an erasable EPROM program memory, which was significantly more expensive than the PROM variant which was only programmable once.

In 1993, the introduction of EEPROM memory allowed microcontrollers (beginning with the Microchip PIC16x84)) to be electrically erased quickly without an expensive package as required for EPROM, allowing both rapid prototyping, and In System Programming.

The same year, Atmel introduced the first microcontroller using Flash memory.

Other companies rapidly followed suit, with both memory types.

Cost has plummeted over time, with the cheapest 8-bit microcontrollers being available for under $0.25 in quantity (thousands) in 2009, and some 32-bit microcontrollers around $1 for similar quantities.

Nowadays microcontrollers are low cost and readily available for hobbyists, with large online communities around certain processors.

In the future, MRAM could potentially be used in microcontrollers as it has infinite endurance and its incremental semiconductor wafer process cost is relatively low.

Microcontroller embedded memory technology

Since the emergence of microcontrollers, many different memory technologies have been used. Almost all microcontrollers have at least two different kinds of memory, a non-volatile memory for storing firmware and a read-write memory for temporary data.

Data

From the earliest microcontrollers to today, six-transistor SRAM is almost always used as the read/write working memory, with a few more transistors per bit used in the register file. MRAM could potentially replace it as it is 4-10 times denser which would make it more cost effective.

In addition to the SRAM, some microcontrollers also have internal EEPROM for data storage; and even ones that do not have any (or not enough) are often connected to external serial EEPROM chip (such as the BASIC Stamp) or external serial flash memory chip.

A few recent microcontrollers beginning in 2003 have "self-programmable" flash memory.

Firmware

The earliest microcontrollers used hard-wired or mask ROM to store firmware. Later microcontrollers (such as the early versions of the Freescale 68HC11 and early PIC microcontrollers) had quartz windows that allowed ultraviolet light in to erase the EPROM.

The Microchip PIC16C84, introduced in 1993, was the first microcontroller to use EEPROM to store firmware

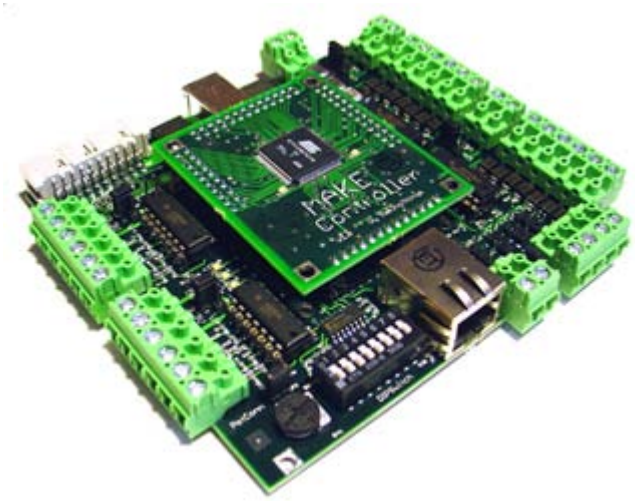
Also in 1993, Atmel introduced the first microcontroller using NOR Flash memory to store firmware.

PSoC microcontrollers, introduced in 2002, store firmware in SONOS flash memory.

MRAM could potentially be used to store firmware.

Chapter-2

Single-board Microcontroller



The Make Controller Kit

A **single-board microcontroller** is a microcontroller pre-built onto a single printed circuit board. This board provides all of the circuitry necessary for a useful control task: microprocessor, I/O circuits, clock generator, RAM, stored program memory and any support ICs necessary. The intention is that the board is immediately useful to an application developer, without needing to spend time and effort in developing the controller board.

As they are usually low-cost hardware, and have an especially low capital cost when starting out with their development, single-board microcontrollers have long been popular in education. They are also a popular means for experienced microcontroller developers to gain hands-on experience with a new processor family.

Origins of the single-board microcontroller

Single-board microcontrollers appeared in the late 1970s when the first generations of microprocessors, such as the 6502 and the Z80, made it practical to build an entire

controller on a single board, and affordable to dedicate a processor chip to such a relatively minor task.

Processors of this era required a number of support chips in addition. RAM and EPROM were separate, often requiring memory management or refresh circuitry for dynamic memory as well. I/O processing might be carried out by a single chip such as the 8255, but frequently required several more chips.

The difference between a single-board microcontroller and a single-board computer is that the microcontroller has no aspirations beyond being a controller. The computer will have some facility for a user interface and bulk storage peripherals that the controller does not. In the 1970s these extra features were sparse and there was often little to distinguish the two roles. Some microcontrollers such as the KIM-1 / AIM-65 began as microcontrollers and were expanded over time and with additional boards to a system more like a microcomputer.

Compared to a microprocessor development board, the purpose of the single board microcontroller is still to be a controller first and foremost. The development board exists to showcase or to train on some particular processor family and this internal implementation is more important than the external function. Obviously there is a large overlap between all three systems.

Internal bus

The bus was universally a Von Neumann architecture, with program and data memory accessed by the same shared bus, even though they were frequently stored in fundamentally different types of memory: ROM for programs and RAM for data. This bus architecture was chosen to economise on the number of pins needed from the limited 40 available for the processor's ubiquitous dual-in-line IC package. When single-chip microcontrollers became available later on, the bus no longer needed to be exposed outside the package and so the Harvard architecture of separate program and data buses (both internal to the chip) became popular.

It was common to offer the internal bus through an expansion connector, or at least the space for such a connector to be soldered on. This was a low-cost option and offered the potential for expansion, even if it was rarely made use of. Typical expansions would be I/O devices, or memory expansion. It was unusual to add peripheral devices such as tape or disk storage, or even a CRT display.

I/O

I/O features on microcontrollers would comprise a few bits of bitwise digital I/O. The signal levels were low-powered and un-isolated, so additional signal conditioning or power outputs would be needed if real-world hardware was to be controlled. Rarely an analogue port (usually input only) might also be found. To control component costs, many boards were designed with extra hardware interface circuits but the components for

these circuits weren't installed and the board was left bare. The circuit was only added as an option on delivery, or could be populated later.

It was common practice for boards to include "prototyping areas", areas of the board already laid out as a solderable breadboard area with the bus and power rails available, but without a defined circuit. Several controllers, particularly those intended for training, also included a pluggable re-usable breadboard for easy prototyping of extra I/O circuits that could be changed or removed for later projects.

Communications and user interfaces

Communications interfaces were fairly uncommon, although several boards supported a serial port. This could either be used to integrate with a host computer when running, or just to download programs when programming.

Microcomputers of this period usually supported the Kansas City or CUTS tape interface and a few even had the ability to use floppy disk storage. There could also be user interface connections for keyboards, CRT screen displays or terminals. These were not found on microcontrollers.

Programming

Many of the earliest systems had no internal facility for programming at all, and relied on a separate "host" system. This programming was typically in assembly language, sometimes C or even PL/M, and then cross-assembled or cross-compiled on the host.

EPROM burning

The completed object code from the host system (usually in Intel HEX format) would then be "burned" onto an EPROM with an EPROM programmer, this EPROM then being physically plugged into the board. As the EPROM would be removed and replaced many times during program development, it was usual to provide a ZIF socket to avoid wear or damage. As "washing" an EPROM with a UV eraser takes a considerable time, it was also usual for a developer to have several EPROMs in circulation at any one time.

Keypad monitors

KIM-1 6502-based single-board computer (1975)

Where the single-board controller formed the entire development environment (typically in education) the board might also be provided with a simple hexadecimal keypad, calculator-style LED display and a "monitor" program set permanently in ROM. This monitor allowed machine code programs to be entered directly through the keyboard and held in RAM. These programs were in machine code, not even in assembly language, and were assembled by hand on paper first. It's arguable as to which process was more time-consuming and error prone: assembling by hand, or keying byte-by-byte.

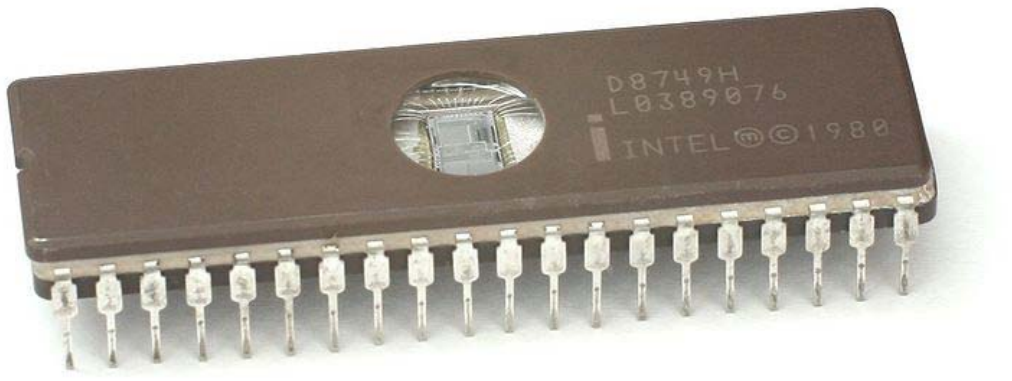
Single-board "keypad and calculator display" microcontrollers of this type were very similar to some low-end microcomputers of the time, such as the KIM-1 or the Microprofessor I. Some of these microprocessor "trainer" systems are still in production today, as a very low-cost introduction to microprocessors at the hardware programming level.

Hosted development

When the cheap desktop PC appeared in the mid-1980s, there was a shift to hosted development but without the need for EPROM burning. Hardware was now cheaper and RAM capacity had expanded such that it was possible to download the program through the serial port and hold it in RAM. This massive reduction in the cycle time to test a new version of a program gave an equally large boost in development speed.

This program memory was still volatile and would be lost if power was turned off. Flash memory was not yet available at a viable price. As a completed controller project usually required to be non-volatile, the final step in a project was often to burn an EPROM again.

Single-chip microcontrollers



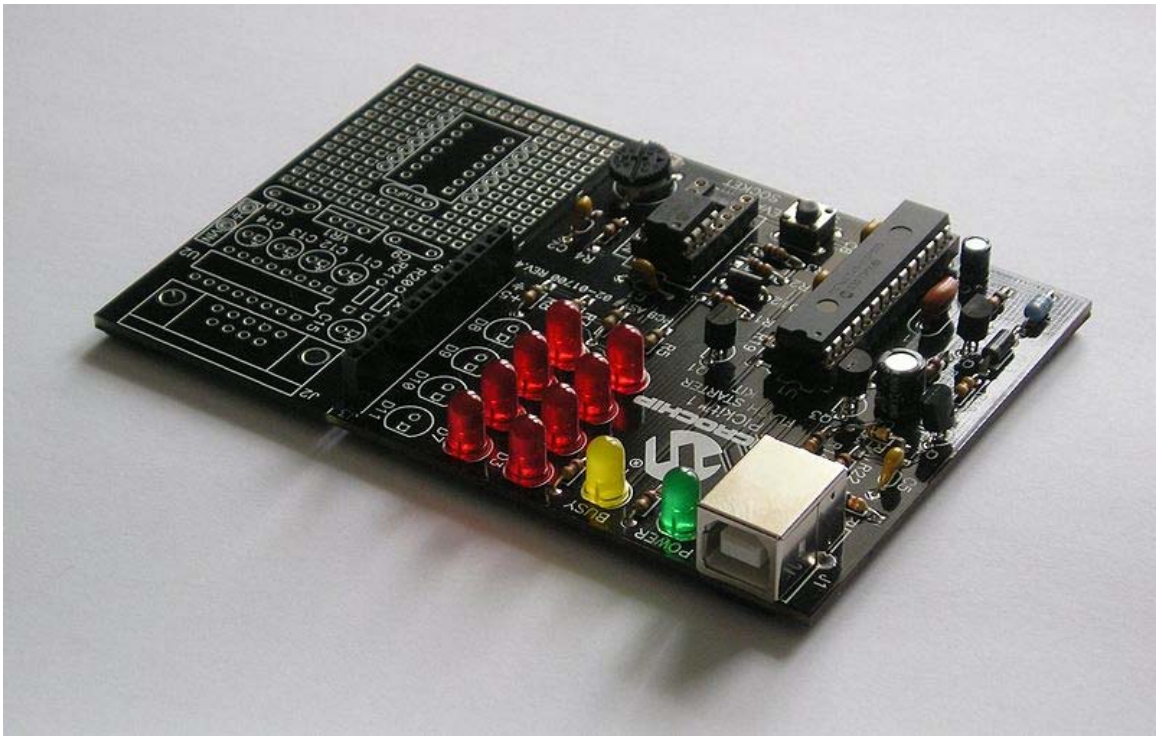
A 8048-family microcontroller with on-board UV EPROM, the 8749

With the development of single-chip microcontrollers such as the 8748, it became possible to combine most of the features of the previous board into a single IC package. Often only the clock generator remained separate, as it remained easier to use a quartz crystal oscillator than to integrate a stable clock circuit into a low-cost IC.

Single-chip microcontrollers integrate their necessary memory (both RAM and ROM) on-package and so do not need to expose their bus through the IC package's pins. These pins are thus freed-up for I/O lines. Both of these changes make the design of a single-board microcontroller simpler, but also less necessary. Single-board development and training systems remained popular, but there was now less need to provide single-board systems as embeddable components for production systems.

Program memory

For production use as embedded systems, the on-board ROM would be either mask programmed at the chip factory or one-time programmed (OTP) by the developer as a PROM. PROMs often used the same UV EPROM technology for the chip, but in a cheaper package without the transparent erasure window. During program development it was still necessary to burn EPROMs, this time the entire controller IC, and so ZIF sockets would be provided.



A development board for a PIC family device

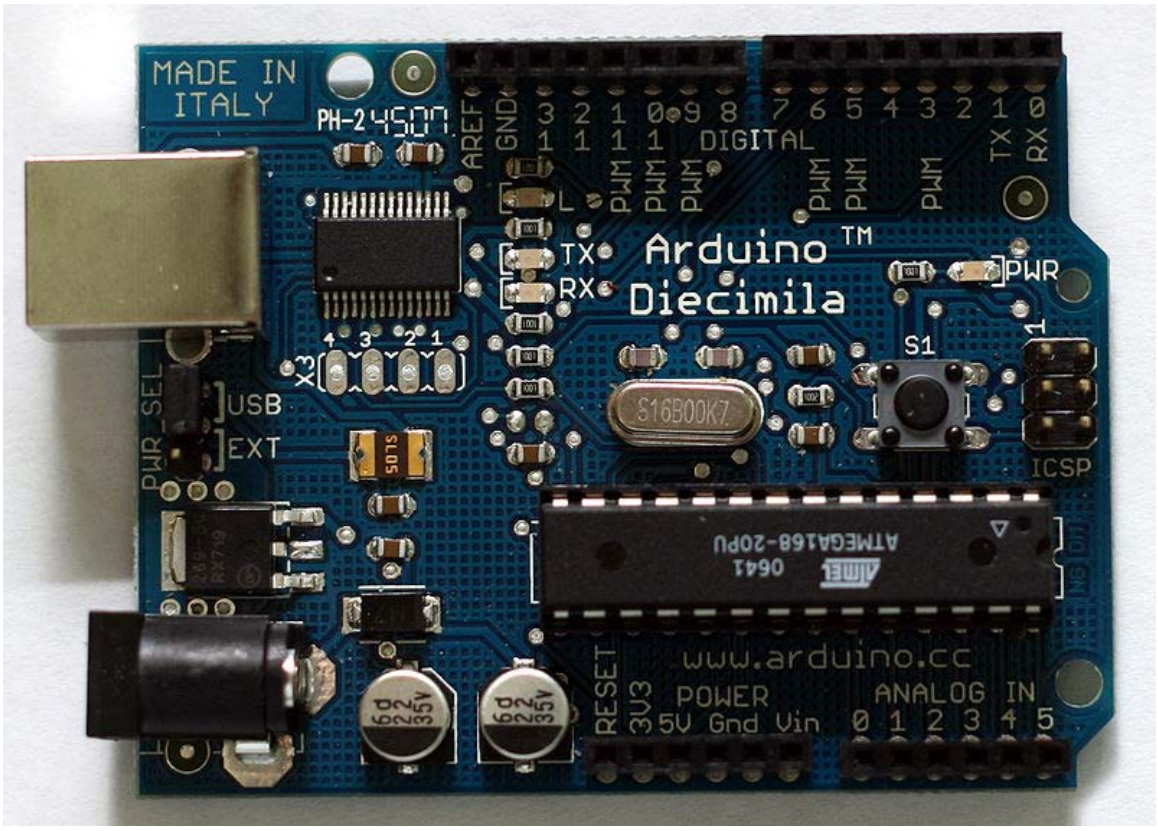
With the development of affordable EEPROM, EAROM and eventually flash memory, it became practical to attach the controller permanently to the board and to download program code to it through a serial connection to a host computer. This was termed "in-circuit programming". Erasure of old programs was carried out by either over-writing them with a new download, or bulk erasing them electrically (for EEPROM) which was slower, but could be carried out in-situ.

The main function of the controller board was now to carry the support circuits for this serial interface, or USB on later boards. As a further convenience feature during development, many boards also carried low-cost features like LED monitors of the I/O lines or reset switches mounted on-board.

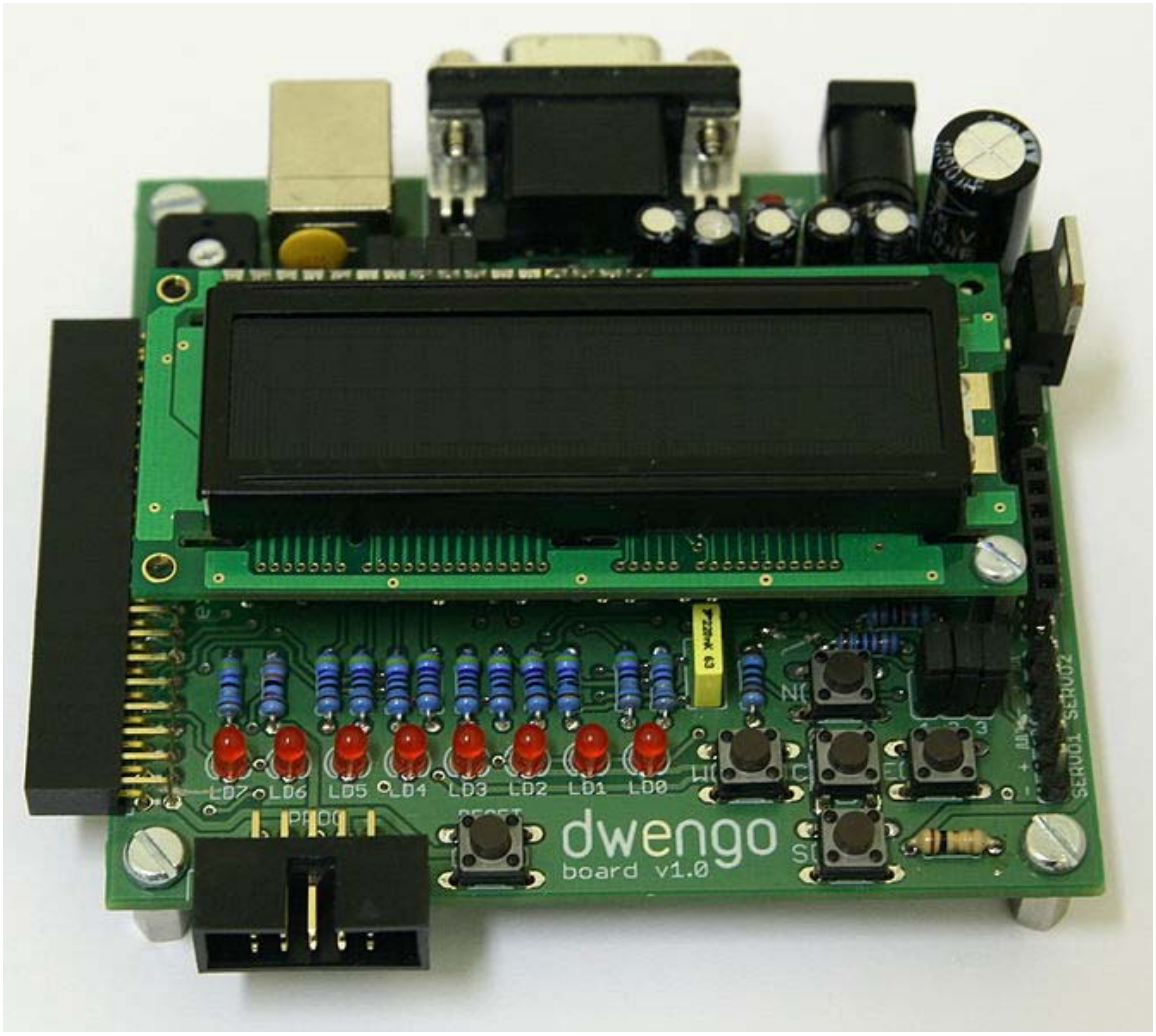
- 8748

- PIC
- Atmel AVR

Single-board microcontrollers today



Arduino Diecimila



Dwengo board

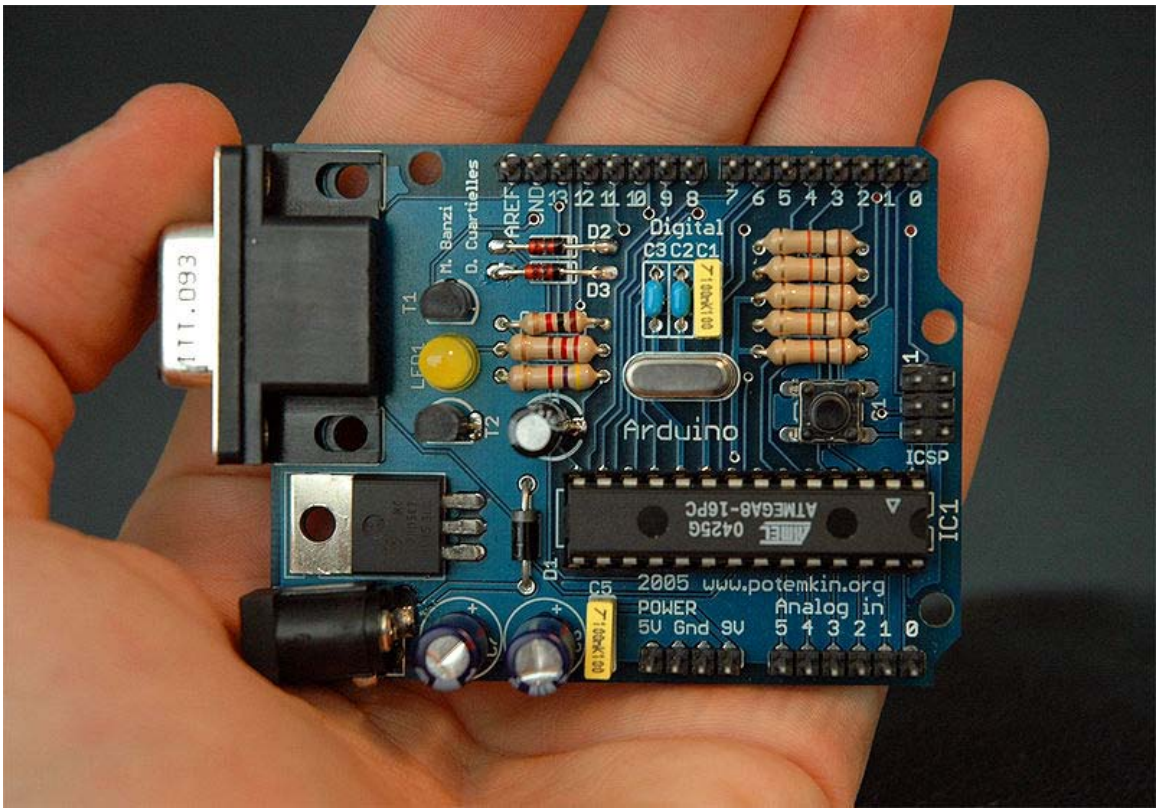
Microcontrollers are now cheap and simple to design circuit boards for. Development host systems are also cheap, especially when using open source software. Higher level programming languages abstract details of the hardware, making differences between specific processors less obvious to the application programmer. Rewritable flash memory has replaced slow programming cycles, at least during program development. Accordingly almost all development now is based on cross-compilation from personal computers and download to the controller board through a serial-like interface, usually appearing to the host as a USB device.

The original market demand of a simplified board implementation is no longer so relevant to microcontrollers. Single-board microcontrollers are still important, but have shifted their focus to:

- Easily accessible platforms aimed at traditionally "non-programmer" groups, such as artists. These controllers may be embedded to form part of a physical computing project. Popular choices for this work are the Arduino, Dwengo or the Wiring project.
- Technology demonstrator boards for innovative processors or peripheral features:
    - AVR Butterfly
    - Parallax Propeller

Chapter-3

Arduino



Arduino compared to a human hand

Arduino is an open-source single-board microcontroller, designed to make the process of using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open hardware design for the Arduino board with an Atmel AVR processor and on-board I/O support. The software consists of a standard programming language and the boot loader that runs on the board.

Arduino hardware is programmed using a Wiring-based language (syntax + libraries), similar to C++ with some simplifications and modifications, and a Processing-based IDE.

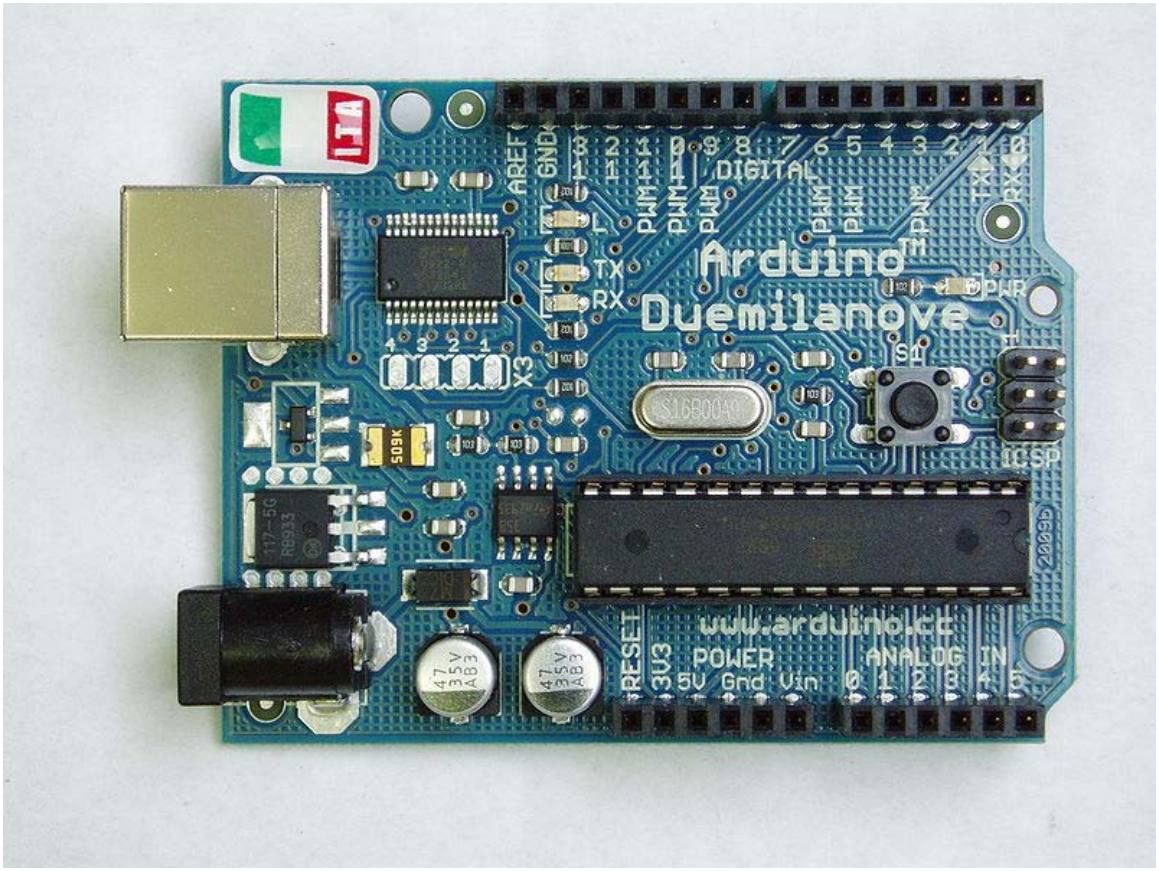
Currently shipping versions can be purchased pre-assembled; hardware design information is available for those who would like to assemble an Arduino by hand. Additionally, variations of the Italian-made Arduino—with varying levels of compatibility—have been released by third parties.

The Arduino project received an honorary mention in the Digital Communities category at the 2006 Prix Ars Electronica.

The project began in Ivrea, Italy in 2005 to make a device for controlling student-built interaction design projects less expensively than other prototyping systems available at the time. As of February 2010 more than 120,000 Arduino boards had been shipped. Founders Massimo Banzi and David Cuartielles named the project after a local bar named Arduino. The name is an Italian masculine first name, meaning "strong friend". The English pronunciation is "Hardwin", a namesake of Arduino of Ivrea.

Platform

Hardware



An official Arduino Duemilanove (rev 2009b).

An Arduino board consists of an 8-bit Atmel AVR microcontroller with complementary components to facilitate programming and incorporation into other circuits. An important

aspect of the Arduino is the standard way that connectors are exposed, allowing the CPU board to be connected to a variety of interchangeable add-on modules (known as shields). Official Arduinos have used the megaAVR series of chips, specifically the ATmega8, ATmega168, ATmega328, and ATmega1280. A handful of other processors have been used by Arduino compatibles. Most boards include a 5 volt linear regulator and a 16 MHz crystal oscillator (or ceramic resonator in some variants), although some designs such as the LilyPad run at 8 MHz and dispense with the onboard voltage regulator due to specific form-factor restrictions. An Arduino's microcontroller is also pre-programmed with a bootloader that simplifies uploading of programs to the on-chip flash memory, compared with other devices that typically need an external chip programmer.

At a conceptual level, when using the Arduino software stack, all boards are programmed over an RS-232 serial connection, but the way this is implemented varies by hardware version. Serial Arduino boards contain a simple inverter circuit to convert between RS-232-level and TTL-level signals. Current Arduino boards are programmed via USB, implemented using USB-to-serial adapter chips such as the FTDI FT232. Some variants, such as the Arduino Mini and the unofficial Boarduino, use a detachable USB-to-serial adapter board or cable, Bluetooth or other methods. (When used with traditional microcontroller tools instead of the Arduino IDE, standard AVR ISP programming is used.)

The Arduino board exposes most of the microcontroller's I/O pins for use by other circuits. The Diecimila, now superseded by the Duemilanove, for example, provides 14 digital I/O pins, six of which can produce PWM signals, and six analog inputs. These pins are on the top of the board, via female 0.1 inch headers. Several plug-in application "shields" are also commercially available.

The Arduino Nano, and Arduino-compatible Bare Bones Board and Boarduino boards provide male header pins on the underside of the board to be plugged into solderless breadboards.

Sortable table

Arduino	Processor	Flash KiB	EEPROM KiB	SRAM KiB	Digital I/O pins	...with PWM	Analog input pins	Dimensions (inches)	Dimensions (mm)
Diecimila	ATmega168	16	0.5	1	14	6	6	2.7"x2.1"	68.6mmx53.3 mm
Duemilanove	ATmega168/ 328	16	0.5	1	14	6	6	2.7"x2.1"	68.6mmx53.3 mm
Uno	ATmega328P	32	1	2	14	6	6	2.7"x2.1"	68.6mmx53.3 mm
Mega	ATmega1280	128	4	8	54	14	16	4"x2.1"	101.6mmx53. 3mm
Fio	ATmega328P	32	1	2	14	6	8	1.1"x1.6"	27.9mmx40.6 mm
Mega2560	ATmega2560	256	4	8	54	14	16	4"x2.1"	101.6mmx53. 3mm

Software

The Arduino IDE is a cross-platform application written in Java, and is derived from the IDE for the Processing programming language and the *Wiring* project. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. There is typically no need to edit makefiles or run programs on the command line.

The Arduino IDE comes with a C/C++ library called "Wiring" (from the project of the same name), which makes many common input/output operations much easier. Arduino programs are written in C/C++, although users only need define two functions to make a runnable program:

- setup() – a function run once at the start of a program that can initialize settings
- loop() – a function called repeatedly until the board powers off

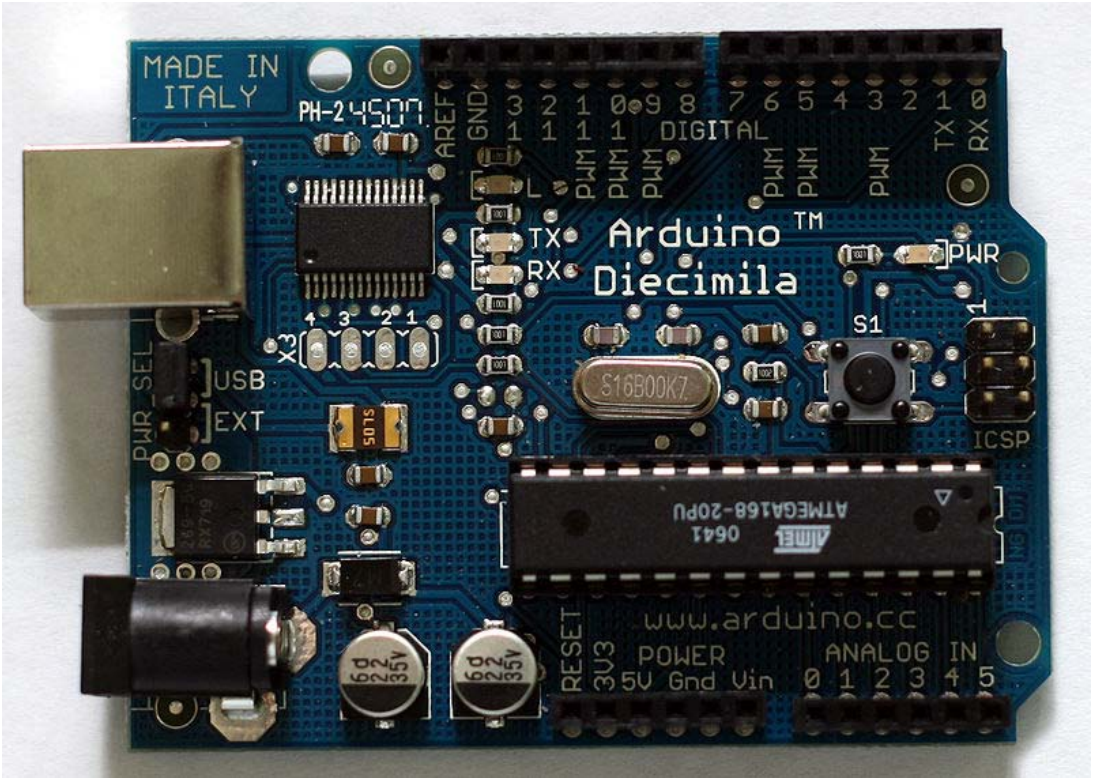
A typical first program for a microcontroller simply blinks a LED (light-emitting diode) on and off. In the Arduino environment, the user might write a program like this:

```
#define LED_PIN 13

void setup () {
    pinMode (LED_PIN, OUTPUT);     // enable pin 13 for digital output
}

void loop () {
    digitalWrite (LED_PIN, HIGH);  // turn on the LED
    delay (1000);                  // wait one second (1000
milliseconds)
    digitalWrite (LED_PIN, LOW);   // turn off the LED
    delay (1000);                  // wait one second
}
```

The above code would not be seen by a standard C++ compiler as a valid program, so when the user clicks the "Upload to I/O board" button in the IDE, a copy of the code is written to a temporary file with an extra include header at the top and a very simple main() function at the bottom, to make it a valid C++ program.

The Arduino IDE uses the GNU toolchain and AVR Libc to compile programs, and uses avrdude to upload programs to the board.

Official hardware



The Arduino Diecimila

The original Arduino hardware is manufactured by the Italian company Smart Projects. Some Arduino-branded boards have been designed by the American company SparkFun Electronics.

Thirteen versions of the Arduino hardware have been commercially produced to date:

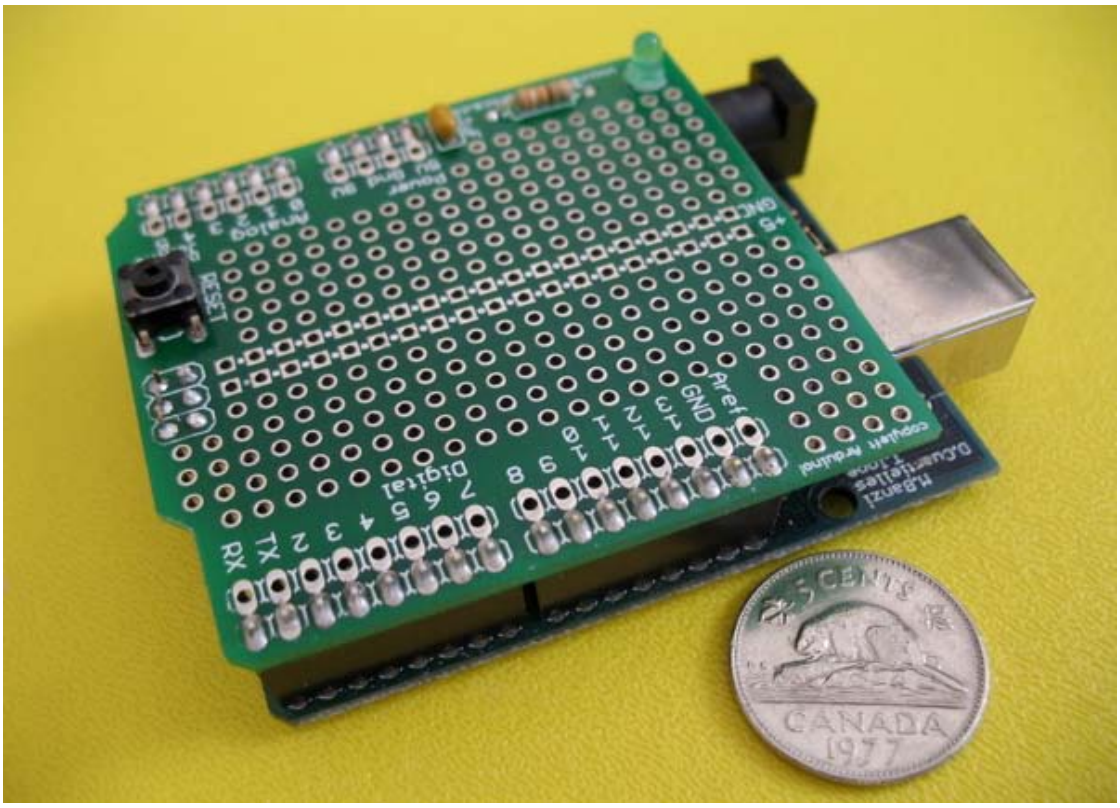
1. The Serial Arduino, programmed with a DE-9 serial connection and using an ATmega8
2. The Arduino Extreme, with a USB interface for programming and using an ATmega8
3. The Arduino Mini, a miniature version of the Arduino using a surface-mounted ATmega168
4. The Arduino Nano, an even smaller, USB powered version of the Arduino using a surface-mounted ATmega168 (ATmega328 for newer version)
5. The LilyPad Arduino, a minimalist design for wearable application using a surface-mounted ATmega168
6. The Arduino NG, with a USB interface for programming and using an ATmega8
7. The Arduino NG plus, with a USB interface for programming and using an ATmega168
8. The Arduino Bluetooth, with a Bluetooth interface for programming using an ATmega168

9. The Arduino Diecimila, with a USB interface and utilizes an ATmega168 in a DIL28 package (pictured)
10. The Arduino Duemilanove ("2009"), using the ATmega168 (ATmega328 for newer version) and powered via USB/DC power, switching automatically
11. The Arduino Mega1280, using a surface-mounted ATmega1280 for additional I/O and memory.
12. The Arduino Uno, uses the same ATmega328 as late-model Duemilanove, but whereas the Duemilanove used an FTDI chipset for USB, the Uno uses an ATmega8U2 programmed as a serial converter.
13. The Arduino Mega2560, uses a surface-mounted ATmega2560, bringing the total memory to 256 kB. It also incorporates the new ATmega8U2 USB chipset.

Open hardware and open source

The Arduino hardware reference designs are distributed under a Creative Commons Attribution Share-Alike 2.5 license and are available on the Arduino Web site. Layout and production files for some versions of the Arduino hardware are also available. The source code for the IDE and the on-board library are available and released under the GPLv2 license.

Accessory hardware



A prototyping shield, mounted on an Arduino

Arduino and Arduino-compatible boards make use of *shields*, which are printed circuit boards that sit atop an Arduino, and plug into the normally supplied pin-headers. These are expansions to the base Arduino. There are many functions of shields, from motor controls, to breadboarding (prototyping).

For example:

- Arduino Ethernet Shield
- XBee Shield
- TouchShield from Liquidware
- Datalog Shield: RTC, SD card storage, temperature sensing, etc. From NuElectronics
- USB Host Shield from Circuits@Home
- Cosmo WiFi Connect from JT5

Arduino-compatible boards

Although the hardware and software designs are freely available under copyleft licenses, the developers have requested that the name "Arduino" be exclusive to the official product and not be used for derivative works without permission. The official policy document on the use of the Arduino name emphasizes that the project is open to incorporating work by others into the official product.

As a result of the protected naming conventions of the Arduino, a group of Arduino users forked the Arduino Diecimila, releasing an equivalent board called Freeduino. The name "Freeduino" is not trademarked and is free to use for any purpose.

Several Arduino-compatible products commercially released have avoided the "Arduino" name by using "-duino" name variants.

Arduino footprint-compatible boards

Example of a Arduino-compatible board: the Freetronics TwentyTen

The following boards are fully or almost fully compatible with both the Arduino hardware and software, including being able to accept "shield" daughterboards.

- The 'Freeduino SB', manufactured and sold as a mini-kit by Solarbotics Ltd..
- The 'Cosmo Black Star', manufactured and sold by JT5.
- The 'Freeduino MaxSerial', a board with a standard DE-9 serial port. It was manufactured and sold assembled or as a kit by Fundamental Logic until May 2010.
- The 'Freeduino Through-Hole', a board that avoids surface-mount soldering, manufactured and sold as a kit by NKC Electronics.

- The 'Illuminato Genesis', a board that uses an ATmega644 instead of an ATmega168. This provides 64 kB of flash, 4 kB of RAM and 42 general I/O pins. Hardware and firmware are open source.

- The 'metaboard', a board that is designed to have a very low complexity and thus a very low price. Hardware and firmware are open source. It was developed by Metalab, a hackerspace in Vienna.

- The 'Seeeduino', derived from the Diecimila.

- The 'eJackino', kit by CQ publisher in Japan. Similar to Seeeduino, eJackino can use Universal boards as Shields. On back side, there is a "Akihabara station" silk, just like Italia on Arduino.

- The 'Japanino' is a kit by Otonano Kagaku publisher in Japan. The board an a POV kit was included in Vol. 27 of the eponymous series. An ATmega168 powers it. It is unique in having a regular size USB A connector.

- The 'Wiseduino' is an Arduino-compatible microcontroller board, which includes a DS1307 real-time clock (RTC) with backup battery, a 24LC256 EEPROM chip and a connector for XBee adapter for wireless communication.

- The 'TwentyTen' is an Arduino-compatible microcontroller board, based on the Duemilanove, with some improvements, including a prototyping area, rearranged LEDs, mini-USB connector, and altered pin 13 circuitry so that the LED and resistor do not interfere with pin function when acting as an input.

- The 'Volksduino' is a low cost, high power, shield compatible, complete Arduino-compatible board kit. Based on the Duemilanove, it comes with a 5 V / 1 A voltage regulator and has an option for a 3.3 V regulator. The Volksduino was designed by Applied Platonics to have a low component count and to be "easy for anyone of any age to put together".

- The 'ZArdino' is a South African Arduino-compatible board derived from the Duemilanove, features mostly through hole construction with the exception of the SMD FT232RL IC, power selection switches, option for a phoenix power connector instead of DC jack, extra I/O pads for using Veroboard as shields, designed to be easy to construct in countries where exotic components are hard to find.

- The 'Zigduino', is an Arduino using the ATmega128RFA1 to integrate Zigbee (IEEE 802.15.4). It can be used with other 802.15.4 network standards as well as ZigBee. The board is the same shape as the Duemilanove and includes an external RPSMA jack on the side of the board opposite the power jack. It is compatible with shields that work with other 3.3 V boards. Due for release Q1 2011 and now taking reservations for the first production run.

- The 'InduinoX' is an Arduino using the ATmega168 and designed for training. It includes on board peripherals such as an RGB LED, switches, and IR Tx/Rx. It was developed and marketed by Simplelabs.

Special purpose Arduino-compatible boards

Special purpose Arduino-compatible boards add additional hardware optimised for a specific application. It is kind of like having an Arduino and a shield on a single board. Some are shield compatible, others are not.

- The "DFRobotShop Rover" is a versatile, minimalist tracked platform based on the Arduino Duemilanove. The PCB incorporates an ATmega328 chip with Arduino bootloader, as well as a dual H-bridge and additional prototyping space and headers. The PCB is compatible with many shields, though four digital pins are used when operating the motor controller. In addition to this, there is an onboard voltage regulator, additional LEDs, a temperature sensor, and a light sensor. The DFRobotShop Rover kit includes a twin motor gearbox, tracks, and minimalist frame to create a tracked mobile robot.

- The "Lightuino", a Arduino-compatible shield that drives LEDs (70 constant-current channels) and LED matrices (1100 LEDs). It also features an adjustable voltage regulator to power the LEDs, an ambient light sensor to decide when to turn "on", and an IR receiver to control your project.

- The "ArduPilot", an Arduino-compatible board designed for auto-piloting and autonomous navigation of aircraft, cars, and boats. It uses GPS for navigation and thermopile sensors or an IMU for stabilization.

Arduino-compatible boards with software-compatibility only

These boards are compatible with the Arduino software but do not accept standard shields. They have different connectors for power and I/O, such as a series of pins on the underside of the board for use with breadboards for easy prototyping, or more specific connectors. One of the important choices made by Arduino-compatible board designers is whether or not to include USB circuitry in the Arduino-compatible board. It is easy to put that circuitry in the cable between development PC and board, thus making each instance of the board less expensive. For many Arduino tasks, the USB circuitry is redundant once the device has been programmed.

- The "Ardweeny" - an inexpensive, even more compact breadboardable device from Solarbotics.
- The "Bare Bones Board" (BBB) and "Really Bare Bones Board" (RBBB) by Modern Device - compact inexpensive Arduino-compatible boards suitable for breadboarding.
- The "Boarduino" - an inexpensive Arduino-Diecimila-compatible board made for breadboarding, produced by Adafruit.

- The "Breaduino" - a complete, very low cost Arduino-compatible kit made to be assembled entirely on a breadboard, made by Applied Platonics.
- The "Diavolino" - another Arduino compatible board from Evil Mad Scientist Laboratories.
- The "DragonFly", a compact board with Molex connectors, aimed at environments where vibration could be an issue. DragonFly features the ATmega1280 and have all 86 IO lines pinned out to connectors.
- The "Femtoduino" - an ultra-small (20.7x15.2 mm) Arduino compatible board designed by Fabio Varesano. Femtoduino is currently the smallest Arduino compatible board available.
- The "iDuino", a USB board for breadboarding, manufactured and sold as a kit by Fundamental Logic.
- The "JeeNode" is a low-cost, low-size, radio-enabled Arduino-compatible board. Based on RBBB with a HopeRF RFM12B wireless module but with a modular approach to I/O interfaces.
- The "LEDuino", a board with enhanced I²C, DCC decoder and CAN-bus interfaces. Manufactured using surface mount and sold assembled by Siliconrailway.
- The "NB1A", is an Arduino-compatible board that includes a battery backed up real-time clock and a four channel DAC. Most Arduino-compatible boards require an additional shield for these resources.
- The "NB2A", is an Sanguino-compatible board that includes a battery backed up real-time clock and a two channel DAC. Sanguino's feature the ATmega644P, which has additional memory, I/O lines and a second UART.
- The "Nymph", a compact board with Molex connectors, aimed at environments where vibration could be an issue. Nymph features the ATmega328P.
- The "Oak Micros om328p" - an Arduino Duemilanove compacted down to a breadboardable device (36 mm x 18 mm) that can be inserted into a standard 600 mil 28-pin socket, with USB capability, ATmega328P, and 6 onboard LEDs.
- The "Rainbowduino", is an Arduino-compatible board designed specifically for driving LEDs. It is generally used to drive an 8x8 RGB LED matrix using row scanning, but can be used for other things.
- The "Roboduino", designed for robotics. All of its connections have neighboring power buses that accommodate servos and sensors. Additional headers for power and serial communication are also provided. It was developed by Curious Inventor, LLC.
- The "Sanguino" - An open source enhanced Arduino-compatible board that uses an ATMega644P instead of an ATMega168. This provides 64 kB of flash, 4 kB of RAM and 32 general I/O pins in a 40 pin DIP device. It was developed with the RepRap Project in mind.
- The "Seeeduino Mega", is an Arduino-Mega-compatible board with 16 extra I/O Pins.
- The "Stickduino", similar to a USB key.
- The "Wireless Widget", is a compact (35 mm x 70 mm), low voltage, battery powered Arduino-compatible board with onboard wireless capable of ranges up to

120 m. The Wireless Widget was designed for both portable and low cost Wireless sensor network applications.
- The "Teensy and Teensy++" - a pair of boards from PJRC.com that run most Arduino sketches using the Teensyduino software add-on to the Arduino IDE.
- The "ZB1", is an Arduino-compatible board that includes a Zigbee radio (XBee). The ZB1 can be powered by USB, a wall adapter or an external battery source. It is designed for low-cost Wireless sensor network applications.

Non-ATmega boards

The following boards accept Arduino "shield" daughter boards but do not use ATmega micro-controllers, and are therefore incompatible with the Arduino IDE.

- The "Amicus18", The Amicus18 is an embedded system platform based on PIC architecture (18F25K20). Can be programmed with any programming language, though the Amicus IDE is completely free and extremely powerful.
- The "PROplus", ARM 100 MHz Cortex M3 and ARM7TDMI-based shield-compatible boards from Coridium, programmable in BASIC or C.
- The "Cortino", a development system for the 32-bit ARM Cortex M3 Microprocessor.
- The "Pinguino", is a board based on a PIC microcontroller, with native USB support and compatibility with the Arduino programing language plus an IDE built with Python and sdcc as compiler.
- The "Unduino", is a board based on the dsPIC33FJ128MC202 microcontroller, with integrated motor control peripherals.
- The "Leaflabs Maple", a 72 MHz 32-bit ARM Cortex M3 micro-controller with USB support, compatibility with Arduino shields, and 39 GP I/O pins. Programmable with the open source Maple IDE (which is a branch of the Arduino library) or low-level native code (with support from the libmaple C library).
- The "Netduino", a 48 MHz 32-bit ARM7 micro-controller board with support for the .NET Micro Framework. Pin compatible with Arduino shields although drivers are required for some shields.
- The "Vinculo", a USB development board for the FTDI Vinculum II microcontroller.
- The "FEZ Domino" and "FEZ Panda", 72 MHz 32-bit ARM (GHI Electronics USBizi chips) micro-controller boards with support for the .NET Micro Framework. Pin compatible with Arduino shields, although drivers are required for some shields.

Development team

The core Arduino developer team is composed of Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, David Mellis and Nicholas Zambetti. Massimo Banzi was interviewed on the March 21st, 2009 episode (Episode 61) of FLOSS Weekly on the TWiT.tv network, in which he discussed the history and goals of the Arduino project.

Chapter-4

BASIC Stamp and Freescale 68HC11

BASIC Stamp

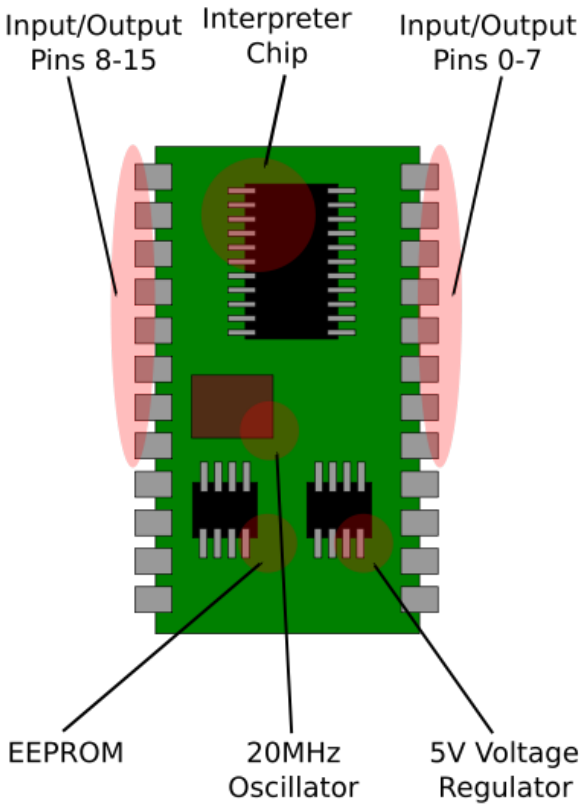


Diagram of BASIC Stamp

The **BASIC Stamp** is a microcontroller with a small, specialized BASIC interpreter (PBASIC) built into ROM. It is made by Parallax, Inc. and has been popular with

electronics hobbyists since the early 1990s due to its low threshold of learning and ease of use (due to its simple BASIC language).

Technical specifications

Although the BASIC Stamp has the form of a DIP chip, it is in fact a small Printed Circuit Board that contains the essential elements of a microprocessor system:

- A Microcontroller containing the CPU, a built in ROM containing the BASIC interpreter, and various peripherals
- Memory (an i²C EEPROM)
- A clock, usually in the form of a ceramic resonator
- A power supply
- External input and output

The end result is that a hobbyist can connect a 9 V battery to a BASIC Stamp and have a complete system. A connection to a personal computer allows the programmer to download software to the BASIC Stamp, which is stored in the onboard non-volatile memory device: it remains programmed until it is erased or reprogrammed, even when the power is removed.

Programming

The BASIC Stamp is programmed in a variant of the BASIC language, called PBASIC. PBASIC incorporates common microcontroller functions, including PWM, serial communications, I²C and 1-Wire communications, communications with common LCD driver circuits, hobby servo pulse trains, pseudo-sine wave frequencies, and the ability to time an RC circuit which may be used to detect an analog value.

Once the program has been written, it is tokenized and sent to the chip through a serial cable.

Versions

The BASIC Stamp 2

There are currently four variants of the interpreter:

1. BASIC Stamp 1 (BS1),
2. BASIC Stamp 2 (BS2), with six sub-variants:
    1. BS2e
    2. BS2sx
    3. BS2p24
    4. BS2p40
    5. BS2pe
    6. BS2px
3. Javelin Stamp
4. Spin Stamp.

The BS2 sub-variants feature more memory, faster execution speed, additional specialized PBASIC commands, extra I/O pins, etc, in comparison to the original BS2 model. While the BS1 and BS2 use a PIC, the remaining BASIC Stamp 2 variants use a Parallax SX processor.

The third variant is the Javelin Stamp. This module uses a subset of Sun Microsystems' Java programming language instead of Parallax's PBASIC. It does not include any networking facilities.

The fourth variant is the Spin Stamp. The module is based on the Parallax Propeller and therefore uses the SPIN programming language instead of PBASIC.

A number of companies now make "clones" of the BASIC Stamp with additional features, such as faster execution, analog-to-digital converters and hardware-based PWM which can run in the background. However, while many use the same pinout as the BASIC Stamp in order to be hardware-compatible in larger-scale projects, they are not necessarily software-compatible.

The Parallax Propeller is gradually accumulating software libraries which give it similar functionality to the BASIC Stamp, however there is no uniform list of which PBASIC facilities now have Spin equivalents.

Freescale 68HC11

Motorola MC68HC11, plastic DIP.

The MC68HC11A8 is available in a 48-pin dual in-line package (DIP), as well as the 52-pin plastic leaded chip carrier (PLCC) as shown above.

The **68HC11** (**6811** or **HC11** for short) is an 8-bit microcontroller (μC) family introduced by Motorola in 1985. Now produced by Freescale Semiconductor, it descended from the Motorola 6800 microprocessor. It is a CISC microcontroller. The 68HC11 devices are more powerful and more expensive than the 68HC08 microcontrollers, and are used in barcode readers, hotel card key writers, amateur robotics, and various other embedded systems. The MC68HC11A8 was the first MCU to include CMOS EEPROM.

Internally, the HC11 instruction set is upward compatible with the 6800, with the addition of a Y index register. (Instructions using the Y register have opcodes prefixed with the byte 0x18). It has two eight-bit accumulators, A and B, two sixteen-bit index registers, X and Y, a condition code register, a 16-bit stack pointer, and a program counter. In addition, some instructions treat the A and B registers as a combined 16-bit D register.

The standard bootloader for the HC11 family is called BUFFALO, "Bit User Fast Friendly Aid to Logical Operation" (a BUFFALO prompt seen on the serial port at bootup is a sign that a board's flash memory has been erased). Not all HC11 models come with the BUFFALO bootloader. The 68HC11A0 and A1 do not but the A8 does.

Different versions of the HC11 have different numbers of external ports, labeled alphabetically. The most common version has five ports, A, B, C, D, and E, but some have as few as 3 ports (version D3). Each port is eight-bits wide except for D, which is six bits (in some variations of the chip, D also has eight bits). It can be operated with an internal program and RAM (1 to 768 bytes) or an external memory of up to 64 kilobytes. With external memory, B and C are used as address and data bus. In this mode, port C is multiplexed to carry both the lower byte of the address and data.

A MC68HC24 port replacement unit is available for the HC11. When placed on the external address bus, it replicates the original functions of B and C. Port A has input capture, output compare, pulse accumulator, and other timer functions; port D has serial I/O, and port E has an analog to digital converter (ADC).

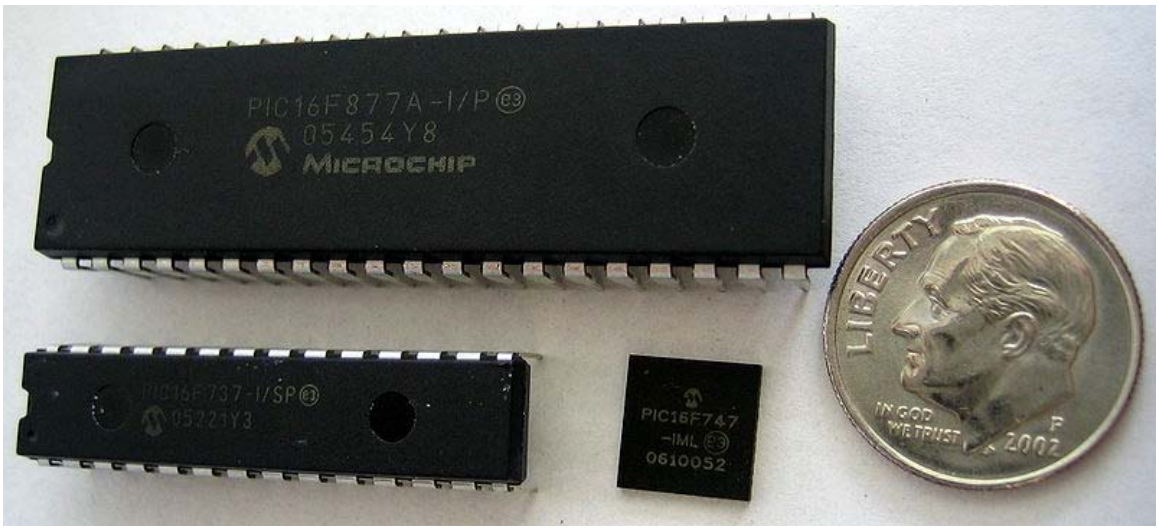
In the early 1990s Motorola produced an evaluation board kit for the 68HC11 with several UARTs, RAM, and an EPROM. The cost of the evaluation kit was $68.11.

The Freescale 68HC12 is an enhanced 16-bit version of the 68HC11.

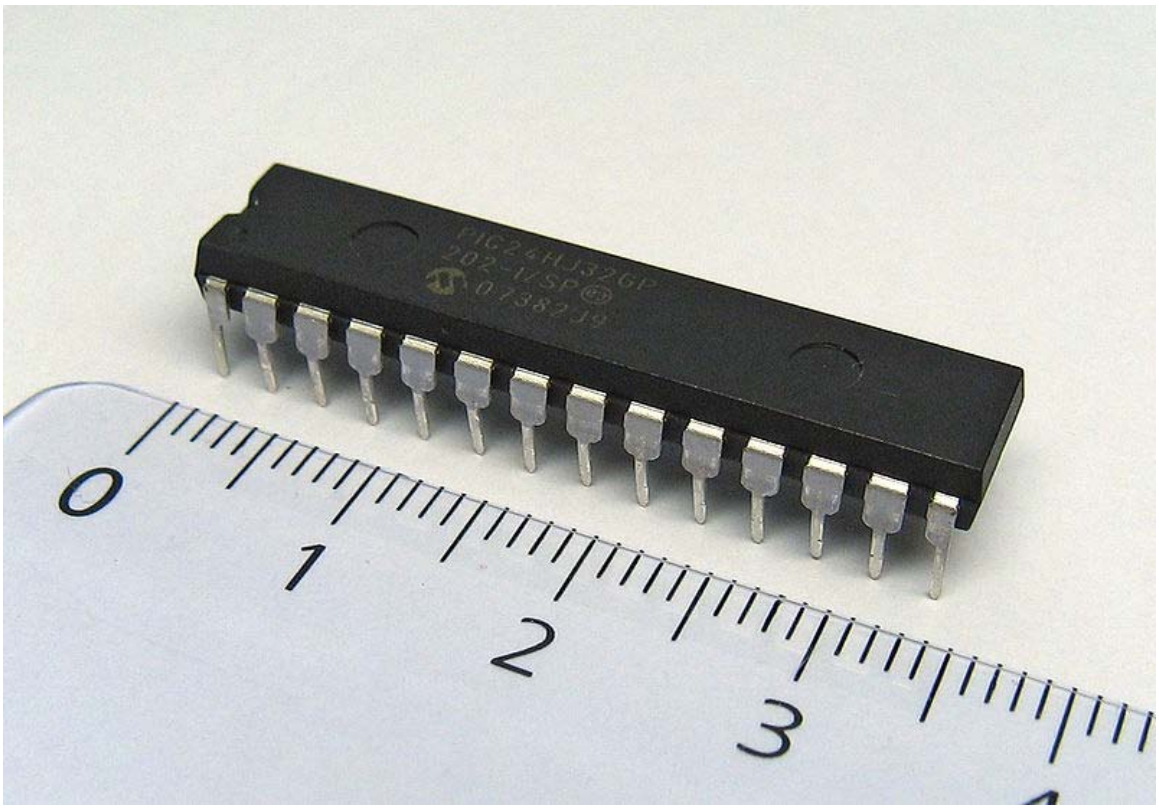
The Freescale 68HC16 microcontroller is intended as a 16-bit mostly software compatible upgrade of the 68HC11.

Chapter-5

PIC Microcontroller



PIC microcontrollers in DIP and QFN packages



16-bit 28-pin PDIP PIC24 microcontroller next to a metric ruler

PIC is a family of Harvard architecture microcontrollers made by Microchip Technology, derived from the PIC1640 originally developed by General Instrument's Microelectronics Division. The name PIC initially referred to "**Peripheral Interface Controller**".

PICs are popular with both industrial developers and hobbyists alike due to their low cost, wide availability, large user base, extensive collection of application notes, availability of low cost or free development tools, and serial programming (and re-programming with flash memory) capability.

Microchip announced on February 2008 the shipment of its six billionth PIC processor.

Core architecture

The PIC architecture is characterized by its multiple attributes:

- Separate code and data spaces (Harvard architecture) for devices other than PIC32, which has a Von Neumann architecture.
- A small number of fixed length instructions
- Most instructions are single cycle execution (2 clock cycles), with one delay cycle on branches and skips

- One accumulator (W0), the use of which (as source operand) is implied (i.e. is not encoded in the opcode)
- All RAM locations function as registers as both source and/or destination of math and other functions.
- A hardware stack for storing return addresses
- A fairly small amount of addressable data space (typically 256 bytes), extended through banking
- Data space mapped CPU, port, and peripheral registers
- The program counter is also mapped into the data space and writable (this is used to implement indirect jumps).

There is no distinction between memory space and register space because the RAM serves the job of both memory and registers, and the RAM is usually just referred to as the register file or simply as the registers.

Data space (RAM)

PICs have a set of registers that function as general purpose RAM. Special purpose control registers for on-chip hardware resources are also mapped into the data space. The addressability of memory varies depending on device series, and all PIC devices have some banking mechanism to extend addressing to additional memory. Later series of devices feature move instructions which can cover the whole addressable space, independent of the selected bank. In earlier devices, any register move had to be achieved via the accumulator.

To implement indirect addressing, a "file select register" (FSR) and "indirect register" (INDF) are used. A register number is written to the FSR, after which reads from or writes to INDF will actually be to or from the register pointed to by FSR. Later devices extended this concept with post- and pre- increment/decrement for greater efficiency in accessing sequentially stored data. This also allows FSR to be treated almost like a stack pointer (SP).

External data memory is not directly addressable except in some high pin count PIC18 devices.

Code space

The code space is generally implemented as ROM, EPROM or flash ROM. In general, external code memory is not directly addressable due to the lack of an external memory interface. The exceptions are PIC17 and select high pin count PIC18 devices.

Word size

All PICs handle (and address) data in 8-bit chunks. However, the unit of addressability of the code space is not generally the same as the data space. For example, PICs in the baseline and mid-range families have program memory addressable in the same wordsize

as the instruction width, i.e. 12 or 14 bits respectively. In contrast, in the PIC18 series, the program memory is addressed in 8-bit increments (bytes), which differs from the instruction width of 16 bits.

In order to be clear, the program memory capacity is usually stated in number of (single word) instructions, rather than in bytes.

Stacks

PICs have a hardware call stack, which is used to save return addresses. The hardware stack is not software accessible on earlier devices, but this changed with the 18 series devices.

Hardware support for a general purpose parameter stack was lacking in early series, but this greatly improved in the 18 series, making the 18 series architecture more friendly to high level language compilers.

Instruction set

A PIC's instructions vary from about 35 instructions for the low-end PICs to over 80 instructions for the high-end PICs. The instruction set includes instructions to perform a variety of operations on registers directly, the accumulator and a literal constant or the accumulator and a register, as well as for conditional execution, and program branching.

Some operations, such as bit setting and testing, can be performed on any numbered register, but bi-operand arithmetic operations always involve W (the accumulator), writing the result back to either W or the other operand register. To load a constant, it is necessary to load it into W before it can be moved into another register. On the older cores, all register moves needed to pass through W, but this changed on the "high end" cores.

PIC cores have skip instructions which are used for conditional execution and branching. The skip instructions are 'skip if bit set' and 'skip if bit not set'. Because cores before PIC18 had only unconditional branch instructions, conditional jumps are implemented by a conditional skip (with the opposite condition) followed by an unconditional branch. Skips are also of utility for conditional execution of any immediate single following instruction.

The 18 series implemented shadow registers which save several important registers during an interrupt, providing hardware support for automatically saving processor state when servicing interrupts.

In general, PIC instructions fall into 5 classes:

1. Operation on working register (WREG) with 8-bit immediate ("literal") operand. E.g. `movlw` (move literal to WREG), `andlw` (AND literal with WREG). One

instruction peculiar to the PIC is `retlw`, load immediate into WREG and return, which is used with computed branches to produce lookup tables.
2. Operation with WREG and indexed register. The result can be written to either the Working register (e.g. `addwf reg,w`). or the selected register (e.g. `addwf reg,f`).
3. Bit operations. These take a register number and a bit number, and perform one of 4 actions: set or clear a bit, and test and skip on set/clear. The latter are used to perform conditional branches. The usual ALU status flags are available in a numbered register so operations such as "branch on carry clear" are possible.
4. Control transfers. Other than the skip instructions previously mentioned, there are only two: `goto` and `call`.
5. A few miscellaneous zero-operand instructions, such as return from subroutine, and `sleep` to enter low-power mode.

Performance

The architectural decisions are directed at the maximization of speed-to-cost ratio. The PIC architecture was among the first scalar CPU designs, and is still among the simplest and cheapest. The Harvard architecture—in which instructions and data come from separate sources—simplifies timing and microcircuit design greatly, and this benefits clock speed, price, and power consumption.

The PIC instruction set is suited to implementation of fast lookup tables in the program space. Such lookups take one instruction and two instruction cycles. Many functions can be modeled in this way. Optimization is facilitated by the relatively large program space of the PIC (e.g. 4096 x 14-bit words on the 16F690) and by the design of the instruction set, which allows for embedded constants. For example, a branch instruction's target may be indexed by W, and execute a "RETLW" which does as it is named - return with literal in W.

Execution time can be accurately estimated by multiplying the number of instructions by two cycles; this simplifies design of real-time code. Similarly, interrupt latency is constant at three instruction cycles. External interrupts have to be synchronized with the four clock instruction cycle, otherwise there can be a one instruction cycle jitter. Internal interrupts are already synchronized. The constant interrupt latency allows PICs to achieve interrupt driven low jitter timing sequences. An example of this is a video sync pulse generator. This is no longer true in the newest PIC models, because they have a synchronous interrupt latency of three or four cycles.

Advantages

The PIC architectures have these advantages:

- Small instruction set to learn
- RISC architecture
- Built in oscillator with selectable speeds

- Inexpensive microcontrollers
- Wide range of interfaces including I2C, SPI, USB, USART, A/D, programmable Comparators, PWM, LIN, CAN, PSP, and Ethernet

Limitations

The PIC architectures have these limitations:

- One accumulator
- Register-bank switching is required to access the entire RAM of many devices
- Operations and registers are not orthogonal; some instructions can address RAM and/or immediate constants, while others can only use the accumulator

The following limitations have been addressed in the **PIC18** series, but still apply to earlier cores:

- Stack:

1. The hardware call stack is not addressable, so preemptive task switching cannot be implemented
2. Software-implemented stacks are not efficient, so it is difficult to generate reentrant code and support local variables

With paged program memory, there are two page sizes to worry about: one for CALL and GOTO and another for computed GOTO (typically used for table lookups). For example, on PIC16, CALL and GOTO have 11 bits of addressing, so the page size is 2048 instruction words. For computed GOTOs, where you add to PCL, the page size is 256 instruction words. In both cases, the upper address bits are provided by the PCLATH register. This register must be changed every time control transfers between pages. PCLATH must also be preserved by any interrupt handler.

Compiler development

While several commercial compilers are available, in 2008, Microchip released their own C compilers, C18 and C30, for the line of 18F 24F and 30/33F processors. By contrast, Atmel's AVR microcontrollers—which are competitive with PIC in terms of hardware capabilities and price, but feature a more traditional instruction set—have long been supported by the GNU C Compiler.

The easy to learn RISC instruction set of the PIC assembly language code can make the overall flow difficult to comprehend. Judicious use of simple macros can increase the readability of PIC assembly language. For example, the original Parallax PIC assembler ("SPASM") has macros which hide W and make the PIC look like a two-address machine. It has macro instructions like "`mov b, a`" (move the data from address *a* to address *b*) and "`add b, a`" (add data from address *a* to data in address *b*). It also hides

the skip instructions by providing three operand branch macro instructions such as "`cjne a, b, dest`" (compare *a* with *b* and jump to *dest* if they are not equal).

Family core architectural differences

Baseline core devices

These devices feature a 12-bit wide code memory, a 32-byte register file, and a tiny two level deep call stack. They are represented by the PIC10 series, as well as by some PIC12 and PIC16 devices. Baseline devices are available in 6-pin to 40-pin packages.

Generally the first 7 to 9 bytes of the register file are special-purpose registers, and the remaining bytes are general purpose RAM. If banked RAM is implemented, the bank number is selected by the high 3 bits of the FSR. This affects register numbers 16–31; registers 0–15 are global and not affected by the bank select bits.

The ROM address space is 512 words (12 bits each), which may be extended to 2048 words by banking. `CALL` and `GOTO` instructions specify the low 9 bits of the new code location; additional high-order bits are taken from the status register. Note that a CALL instruction only includes 8 bits of address, and may only specify addresses in the first half of each 512-word page.

The instruction set is as follows. Register numbers are referred to as "f", while constants are referred to as "k". Bit numbers (0–7) are selected by "b". The "d" bit selects the destination: 0 indicates W, while 1 indicates that the result is written back to source register f.

12-bit PIC instruction set

Opcode (binary)	Mnemonic	Description
`0000 0000 0000`	NOP	No operation
`0000 0000 0010`	OPTION	Load OPTION register with contents of W
`0000 0000 0011`	SLEEP	Go into standby mode
`0000 0000 0100`	CLRWDT	Reset watchdog timer
`0000 0000 01ff`	TRIS f	Move W to port control register (f=1..3)
`0000 001 fffff`	MOVWF f	Move W to f
`0000 010 xxxxx`	CLRW	Clear W to 0 (a.k.a CLR x, W)
`0000 011 fffff`	CLRF f	Clear f to 0 (a.k.a. CLR f, F)
`0000 10d fffff`	SUBWF f, d	Subtract W from f (d = f − W)
`0000 11d fffff`	DECF f, d	Decrement f (d = f − 1)
`0001 00d fffff`	IORWF f, d	Inclusive OR W with F (d = f OR W)
`0001 01d fffff`	ANDWF f, d	AND W with F (d = f AND W)
`0001 10d fffff`	XORWF f, d	Exclusive OR W with F (d = f XOR W)

```
0001 11d fffff ADDWF f, d Add W with F (d = f + W)
0010 00d fffff MOVF f, d   Move F (d = f)
0010 01d fffff COMF f, d   Complement f (d = NOT f)
0010 10d fffff INCF f, d   Increment f (d = f + 1)
0010 11d fffff DECFSZ f, d Decrement f (d = f − 1) and skip if zero
0011 00d fffff RRF f, d    Rotate right F (rotate right through carry)
0011 01d fffff RLF f, d    Rotate left F (rotate left through carry)
0011 10d fffff SWAPF f, d  Swap 4-bit halves of f (d = f<<4 | f>>4)
0011 11d fffff INCFSZ f, d Increment f (d = f + 1) and skip if zero

0100 bbb fffff BCF f, b    Bit clear f (Clear bit b of f)
0101 bbb fffff BSF f, b    Bit set f (Set bit b of f)
0110 bbb fffff BTFSC f, b  Bit test f, skip if clear (Test bit b of f)
0111 bbb fffff BTFSS f, b  Bit test f, skip if set (Test bit b of f)

1000 kkkkkkkk  RETLW k     Set W to k and return
1001 kkkkkkkk  CALL k      Save return address, load PC with k
101k kkkkkkkk  GOTO k      Jump to address k (9 bits)
1100 kkkkkkkk  MOVLW k     Move literal to W (W = k)
1101 kkkkkkkk  IORLW k     Inclusive or literal with W (W = k OR W)
1110 kkkkkkkk  ANDLW k     AND literal with W (W = k AND W)
1111 kkkkkkkk  XORLW k     Exclusive or literal with W (W = k XOR W)
```

Mid-range core devices

These devices feature a 14-bit wide code memory, and an improved 8 level deep call stack. The instruction set differs very little from the baseline devices, but the increased opcode width allows 128 registers and 2048 words of code to be directly addressed. The mid-range core is available in the majority of devices labeled PIC12 and PIC16.

The first 32 bytes of the register space are allocated to special-purpose registers; the remaining 96 bytes are used for general-purpose RAM. If banked RAM is used, the high 16 registers (0x70–0x7F) are global, as are a few of the most important special-purpose registers, including the STATUS register which holds the RAM bank select bits. (The other global registers are FSR and INDF, the low 8 bits of the program counter PCL, the PC high preload register PCLATH, and the master interrupt control register INTCON.)

The PCLATH register supplies high-order instruction address bits when the 8 bits supplied by a write to the PCL register, or the 11 bits supplied by a `GOTO` or `CALL` instruction, is not sufficient to address the available ROM space.

14-bit PIC instruction set

Opcode (binary)	Mnemonic	Description
00 0000 0000 0000	NOP	No operation
00 0000 0000 1000	RETURN	Return from subroutine, W unchanged
00 0000 0000 1001	RETFIE	Return from interrupt
00 0000 0110 0010	OPTION	Write W to OPTION register
00 0000 0110 0011	SLEEP	Go into standby mode
00 0000 0110 0100	CLRWDT	Reset watchdog timer
00 0000 0110 01ff	TRIS f	Write W to tristate register f
00 0000 1 fffffff	MOVWF f	Move W to f
00 0001 0 xxxxxxx	CLRW	Clear W to 0 (W = 0)
00 0001 1 fffffff	CLRF f	Clear f to 0 (f = 0)
00 0010 d fffffff	SUBWF f, d	Subtract W from f (d = f − W)
00 0011 d fffffff	DECF f, d	Decrement f (d = f − 1)
00 0100 d fffffff	IORWF f, d	Inclusive OR W with F (d = f OR W)
00 0101 d fffffff	ANDWF f, d	AND W with F (d = f AND W)
00 0110 d fffffff	XORWF f, d	Exclusive OR W with F (d = f XOR W)
00 0111 d fffffff	ADDWF f, d	Add W with F (d = f + W)
00 1000 d fffffff	MOVF f, d	Move F (d = f)
00 1001 d fffffff	COMF f, d	Complement f (d = NOT f)
00 1010 d fffffff	INCF f, d	Increment f (d = f + 1)
00 1011 d fffffff	DECFSZ f, d	Decrement f (d = f − 1) and skip if zero
00 1100 d fffffff	RRF f, d	Rotate right F (rotate right through carry)
00 1101 d fffffff	RLF f, d	Rotate left F (rotate left through carry)
00 1110 d fffffff	SWAPF f, d	Swap 4-bit halves of f (d = f<<4 | f>>4)
00 1111 d fffffff	INCFSZ f, d	Increment f (d = f + 1) and skip if zero
01 00 bbb fffffff	BCF f, b	Bit clear f (Clear bit b of f)
01 01 bbb fffffff	BSF f, b	Bit set f (Set bit b of f)
01 10 bbb fffffff	BTFSC f, b	Bit test f, skip if clear (Test bit b of f)
01 11 bbb fffffff	BTFSS f, b	Bit test f, skip if set (Test bit b of f)
10 0 kkkkkkkkkk	CALL k	Save return address, load PC with k
10 1 kkkkkkkkkk	GOTO k	Jump to address k (11 bits)
11 00xx kkkkkkkk	MOVLW k	Move literal to W (W = k)
11 01xx kkkkkkkk	RETLW k	Set W to k and return

11 1000 kkkkkkkk	IORLW k	Inclusive or literal with W (W = k OR W)
11 1001 kkkkkkkk	ANDLW k	AND literal with W (W = k AND W)
11 1010 kkkkkkkk	XORLW k	Exclusive or literal with W (W = k XOR W)
11 110x kkkkkkkk	SUBLW k	Subtract W from literal (W = k − W)
11 111x kkkkkkkk	ADDLW k	Add literal to W (W = k + W)

Enhanced Mid-range core devices

Enhanced Mid-range core devices introduce a deeper hardware stack, additional reset methods, 14 additional instructions and 'C' programming language optimizations.

PIC17 high end core devices

The 17 series never became popular and has been superseded by the PIC18 architecture. It is not recommended for new designs, and availability may be limited.

Improvements over earlier cores are 16-bit wide opcodes (allowing many new instructions), and a 16 level deep call stack. PIC17 devices were produced in packages from 40 to 68 pins.

The 17 series introduced a number of important new features:

- a memory mapped accumulator
- read access to code memory (table reads)
- direct register to register moves (prior cores needed to move registers through the accumulator)
- an external program memory interface to expand the code space
- an 8-bit x 8-bit hardware multiplier
- a second indirect register pair
- auto-increment/decrement addressing controlled by control bits in a status register (ALUSTA)

PIC18 high end core devices

Microchip introduced the PIC18 architecture in 2000. Unlike the 17 series, it has proven to be very popular, with a large number of device variants presently in manufacture. In contrast to earlier devices, which were more often than not programmed in assembly, C has become the predominant development language .

The 18 series inherits most of the features and instructions of the 17 series, while adding a number of important new features:

- much deeper call stack (31 levels deep)
- the call stack may be read and written
- conditional branch instructions

- indexed addressing mode (PLUSW)
- extending the FSR registers to 12 bits, allowing them to linearly address the entire data address space
- the addition of another FSR register (bringing the number up to 3)

The auto increment/decrement feature was improved by removing the control bits and adding four new indirect registers per FSR. Depending on which indirect file register is being accessed it is possible to postdecrement, postincrement, or preincrement FSR; or form the effective address by adding W to FSR.

In more advanced PIC18 devices, an "extended mode" is available which makes the addressing even more favorable to compiled code:

- a new offset addressing mode; some addresses which were relative to the access bank are now interpreted relative to the FSR2 register
- the addition of several new instructions, notable for manipulating the FSR registers.

These changes were primarily aimed at improving the efficiency of a data stack implementation. If FSR2 is used either as the stack pointer or frame pointer, stack items may be easily indexed—allowing more efficient re-entrant code. Microchip's MPLAB C18 C compiler chooses to use FSR2 as a frame pointer.

PIC24 and dsPIC 16-bit microcontrollers

In 2001, Microchip introduced the dsPIC series of chips, which entered mass production in late 2004. They are Microchip's first inherently 16-bit microcontrollers. PIC24 devices are designed as general purpose microcontrollers. dsPIC devices include digital signal processing capabilities in addition.

Architecturally, although they share the PIC moniker, they are very different from the 8-bit PICs. The most notable differences are:

- they feature a set of 16 working registers (W0-W15)
- they fully support a stack in RAM, and do not have a hardware stack
- bank switching is not required to access RAM or special function registers
- data stored in program memory can be accessed directly using a feature called Program Space Visibility
- interrupt sources may be assigned to distinct handlers using an interrupt vector table

Some features are:

- hardware MAC (multiply-accumulate)
- barrel shifting
- bit reversal

- (16×16)-bit single-cycle multiplication and other DSP operations
- hardware divide assist (19 cycles for 16/32-bit divide)
- hardware support for loop indexing
- Direct memory access

dsPICs can be programmed in C using a variant *[What Variant ?]* of gcc.

PIC32 32-bit microcontrollers

In November 2007 Microchip introduced the new PIC32MX family of 32-bit microcontrollers. The initial device line-up is based on the industry standard MIPS32 M4K Core. The device can be programmed using the Microchip MPLAB C Compiler for PIC32 MCUs, a variant of the GCC compiler. The first 18 models currently in production (PIC32MX3xx and PIC32MX4xx) are pin to pin compatible and share the same peripherals set with the PIC24FxxGA0xx family of (16-bit) devices allowing the use of common libraries, software and hardware tools.

The PIC32 architecture brings a number of new features to Microchip portfolio, including:

- The highest execution speed 80 MIPS (120+ Dhrystone MIPS @ 80 MHz)
- The largest flash memory: 512 kByte
- One instruction per clock cycle execution
- The first cached processor
- Allows execution from RAM
- Full Speed Host/Dual Role and OTG USB capabilities
- Full JTAG and 2 wire programming and debugging
- Real-time trace

Device variants and hardware features

PIC devices generally feature:

- Sleep mode (power savings).
- Watchdog timer.
- Various crystal or RC oscillator configurations, or an external clock.

Variants

Within a series, there are still many device variants depending on what hardware resources the chip features.

- General purpose I/O pins.
- Internal clock oscillators.
- 8/16/32 Bit Timers.
- Internal EEPROM Memory.

- Synchronous/Asynchronous Serial Interface USART.
- MSSP Peripheral for I²C and SPI Communications.
- Capture/Compare and PWM modules.
- Analog-to-digital converters (up to ~1.0 MHz).
- USB, Ethernet, CAN interfacing support.
- External memory interface.
- Integrated analog RF front ends (PIC16F639, and rfPIC).
- KEELOQ Rolling code encryption peripheral (encode/decode)
- And many more.

Trends

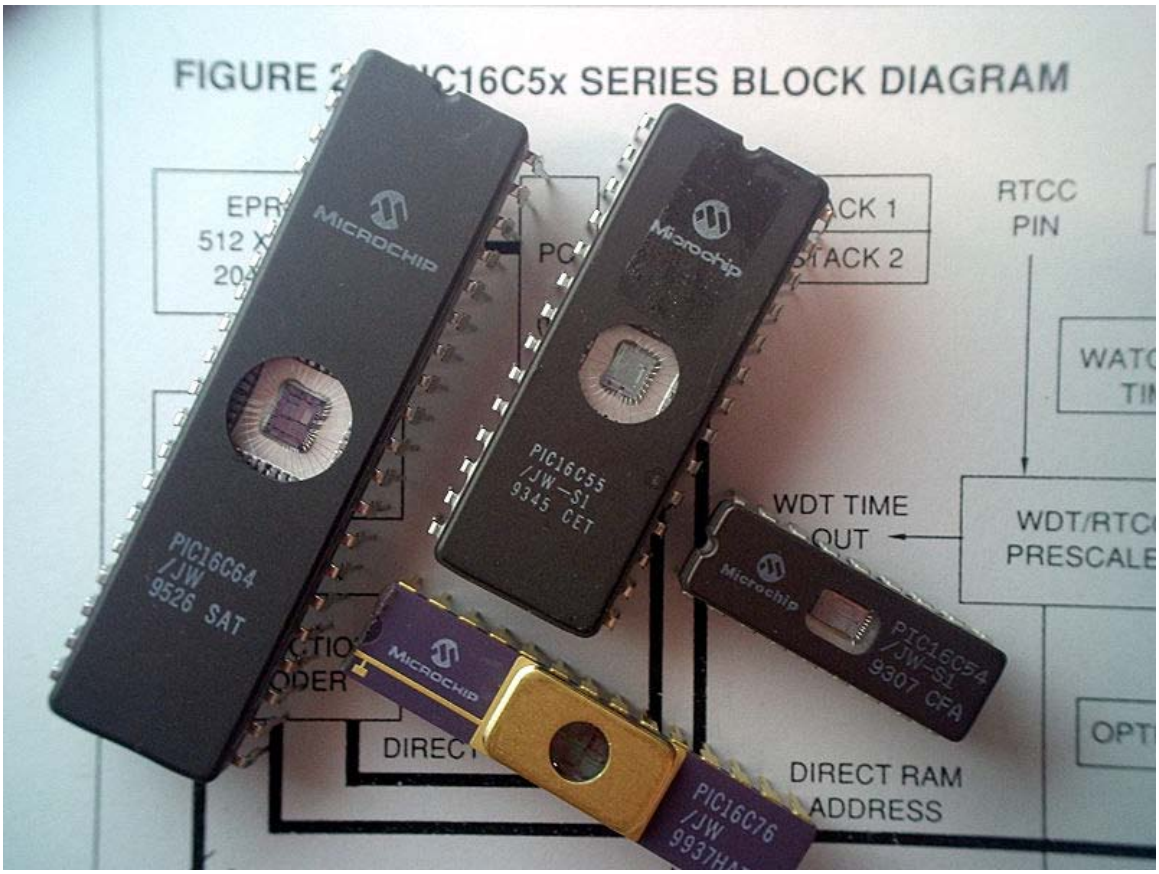
The first generation of PICs with EPROM storage are almost completely replaced by chips with Flash memory. Likewise, the original 12-bit instruction set of the PIC1650 and its direct descendants has been superseded by 14-bit and 16-bit instruction sets. Microchip still sells OTP (one-time-programmable) and windowed (UV-erasable) versions of some of its EPROM based PICs for legacy support or volume orders. The Microchip website lists PICs that are not electrically erasable as OTP despite the fact that UV erasable windowed versions of these chips can be ordered.

History

The original PIC was built to be used with General Instruments' new 16-bit CPU, the CP1600. While generally a good CPU, the CP1600 had poor I/O performance, and the 8-bit PIC was developed in 1975 to improve performance of the overall system by offloading I/O tasks from the CPU. The PIC used simple microcode stored in ROM to perform its tasks, and although the term was not used at the time, it shares some common features with RISC designs.

In 1985, General Instruments spun off their microelectronics division and the new ownership cancelled almost everything — which by this time was mostly out-of-date. The PIC, however, was upgraded with internal EPROM to produce a programmable channel controller and today a huge variety of PICs are available with various on-board peripherals (serial communication modules, UARTs, motor control kernels, etc.) and program memory from 256 words to 64k words and more (a "word" is one assembly language instruction, varying from 12, 14 or 16 bits depending on the specific PIC micro family).

PIC and PICmicro are registered trademarks of Microchip Technology. It is generally thought that PIC stands for **Peripheral Interface Controller**, although General Instruments' original acronym for the initial PIC1640 and PIC1650 devices was "**Programmable Interface Controller**". The acronym was quickly replaced with "**Programmable Intelligent Computer**".



Various older (EPROM) PIC microcontrollers

The Microchip 16C84 (PIC16x84), introduced in 1993 , was the first Microchip CPU with on-chip EEPROM memory. This electrically-erasable memory made it cost less than CPUs that required a quartz "erase window" for erasing EPROM.

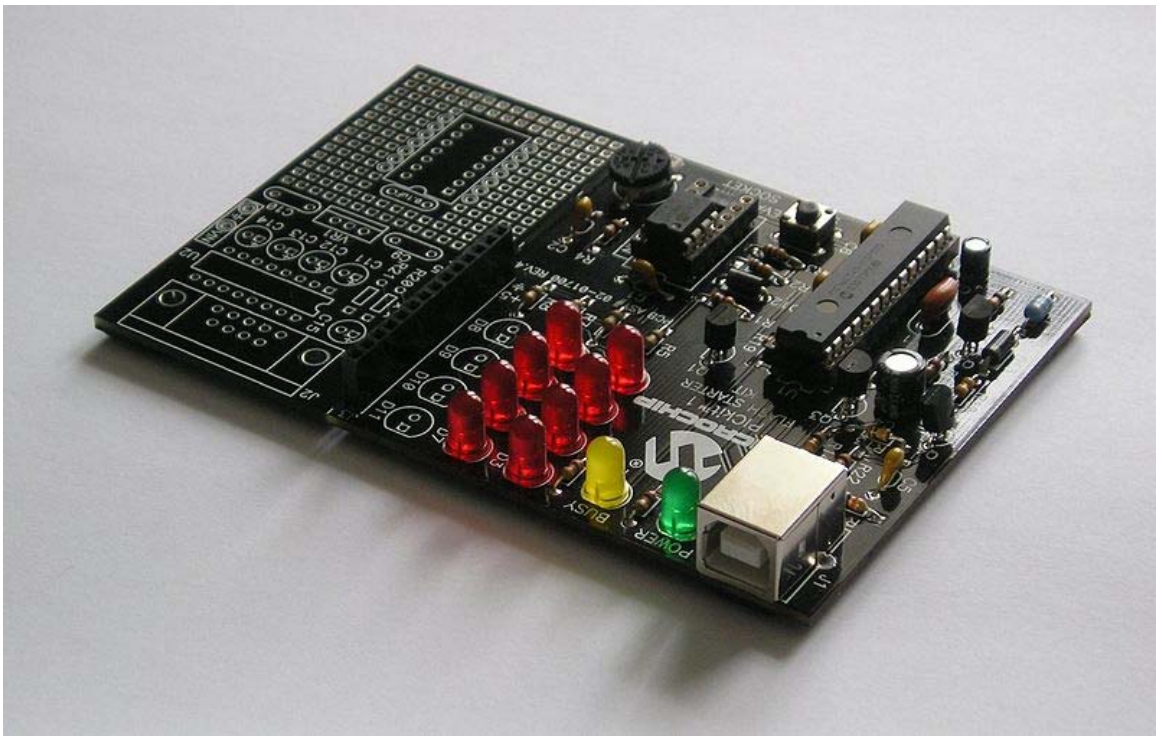
Development tools

Microchip provides a freeware IDE package called MPLAB, which includes an assembler, linker, software simulator, and debugger. They also sell C compilers for the PIC18 and dsPIC which integrate cleanly with MPLAB. Free student versions of the C compilers are also available with all features. But for the free versions, optimizations will be disabled after 60 days.

Several third parties make C language compilers for PICs, many of which integrate to MPLAB and/or feature their own IDE. A fully featured compiler for the PICBASIC language to program PIC microcontrollers is available from meLabs, Inc.

Development tools are available for the PIC family under the GPL or other free software or open sources licenses.

Device programmers



A development board for low pin-count MCU, from Microchip

Devices called "programmers" are traditionally used to get program code into the target PIC. Most PICs that Microchip currently sell feature ICSP (In Circuit Serial Programming) and/or LVP (Low Voltage Programming) capabilities, allowing the PIC to be programmed while it is sitting in the target circuit. ICSP programming is performed using two pins, clock and data, while a high voltage (12V) is present on the Vpp/MCLR pin. Low voltage programming dispenses with the high voltage, but reserves exclusive use of an I/O pin and can therefore be disabled to recover the pin for other uses (once disabled it can only be re-enabled using high voltage programming).

There are many programmers for PIC microcontrollers, ranging from the extremely simple designs which rely on ICSP to allow direct download of code from a host computer, to intelligent programmers that can verify the device at several supply voltages. Many of these complex programmers use a pre-programmed PIC themselves to send the programming commands to the PIC that is to be programmed. The intelligent type of programmer is needed to program earlier PIC models (mostly EPROM type) which do not support in-circuit programming.

Many of the higher end flash based PICs can also self-program (write to their own program memory). Demo boards are available with a small bootloader factory programmed that can be used to load user programs over an interface such as RS-232 or USB, thus obviating the need for a programmer device. Alternatively there is bootloader

firmware available that the user can load onto the PIC using ICSP. The advantages of a bootloader over ICSP is the far superior programming speeds, immediate program execution following programming, and the ability to both debug and program using the same cable.

Microchip PICSTART Plus programmer

Programmers/debuggers are available directly from Microchip. Third party programmers range from plans to build your own, to self-assembly kits and fully tested ready-to-go units. Some are simple designs which require a PC to do the low-level programming signalling (these typically connect to the serial or parallel port and consist of a few simple components), while others have the programming logic built into them (these typically use a serial or USB connection, are usually faster, and are often built using PICs themselves for control). The major problem of home-made or very simple programmers is that these programmers do not comply with programming specifications and this can cause premature loss of data in the flash or EEPROM.

Debugging

Software emulation

Commercial and free emulators exist for the PIC family processors.

In-circuit debugging

Later model PICs feature an ICD (in-circuit debugging) interface, built into the CPU core. ICD debuggers (MPLAB ICD2 and other third party) can communicate with this interface using three lines. This cheap and simple debugging system comes at a price however, namely limited breakpoint count (1 on older pics 3 on newer PICs), loss of some IO (with the exception of some surface mount 44-pin PICs which have dedicated lines for debugging) and loss of some features of the chip. For small PICs, where the loss of IO caused by this method would be unacceptable, special headers are made which are fitted with PICs that have extra pins specifically for debugging.

In-circuit emulators

Microchip offers three full in circuit emulators: the MPLAB ICE2000 (parallel interface, a USB converter is available); the newer MPLAB ICE4000 (USB 2.0 connection); and most recently, the REAL ICE. All of these ICE tools can be used with the MPLAB IDE for full source-level debugging of code running on the target.

The ICE2000 requires emulator modules, and the test hardware must provide a socket which can take either an emulator module, or a production device.

The REAL ICE connects directly to production devices which support in-circuit emulation through the PGC/PGD programming interface, or through a high speed connection which uses two more pins. According to Microchip, it supports "most" flash-based PIC, PIC24, and dsPIC processors.

The ICE4000 is no longer directly advertised on Microchip's website, and the purchasing page states that it is not recommended for new designs.

PIC clones

Third party manufacturers make compatible products.

PICKit 2 open source structure and clones

PICKit 2 has been an interesting PIC programmer from Microchip. It can program all PICs and debug most of the PICs (as of May-2009, only the PIC32 family is not supported for MPLAB debugging). Ever since its first releases, all software source code (firmware, PC application) and hardware schematic are open to the public. This makes it relatively easy for an end user to modify the programmer for use with a non-Windows operating system such as Linux or Mac OS. In the mean time, it also creates lots of DIY interest and clones. This open source structure brings many features to the PICKit 2 community such as Programmer-to-Go, the UART Tool and the Logic Tool, which have been contributed by PICKit 2 users. Users have also added such features to the PICKit 2 as 4MB Programmer-to-go capability, USB buck/boost circuits, RJ12 type connectors and others.

Part number suffixes

The F in a name generally indicates the PICmicro uses flash memory and can be erased electronically. A C generally means it can only be erased by exposing the die to ultraviolet light (which is only possible if a windowed package style is used). An exception to this rule is the PIC16C84 which uses EEPROM and is therefore electrically erasable.

Chapter-6

Parallax Propeller

Parallax Propeller chip

The **Parallax P8X32 Propeller**, introduced in 2006, is a multi-core architecture parallel microcontroller with eight 32-bit RISC CPU cores.

The Parallax Propeller microcontroller, Propeller Assembly language, and Spin interpreter were designed by one person, Parallax's co-founder and president Chip Gracey. The Spin Programming language and "Propeller Tool" integrated development environment were designed by Chip Gracey and Parallax's software engineer Jeff Martin. The Propeller is known for being easy to program.

Multi-core architecture

Each of the eight 32-bit cores (called a **cog**) has an elementary CPU (the ALU of which does not directly support division) and access to 512 32-bit long words (2 KB) of instructions and data. Self-modifying code is possible and is used internally, for example by an instruction that is used to create a subroutine call/return mechanism without the need for a stack. Access to shared memory (32 KB RAM; 32 KB ROM) is controlled in round-robin fashion by an internal bus controller called the **hub**. Each cog also has access to two dedicated hardware counters and two special "video registers" for use in generating PAL, NTSC, VGA, servo-control, or other timing signals.

Speed and power management

The Propeller can be clocked using either an internal, on-chip oscillator (providing a lower total parts count, but sacrificing some accuracy and thermal stability) or an external crystal or resonator (providing higher maximum speed with greater accuracy at an increased total cost). Only the external oscillator may be run through an on-chip PLL clock multiplier, which may be set at 1x, 2x, 4x, 8x, or 16x.

Both the on-board oscillator frequency (if used) and the PLL multiplier value may be changed at run-time. If used correctly, this can improve power efficiency; for example, the PLL multiplier can be decreased before a long "no operation" wait required for timing purposes, then increased afterwards, causing the processor to use less power. However, the utility of this technique is limited to situations where no other cog is executing timing-dependent code (or is carefully designed to cope with the change), since the effective clock rate is common to all cogs.

The effective clock rate ranges from 32 kHz up to 80 MHz (with the exact values available for dynamic control dependent on the configuration used, as described above). When running at 80 MHz, the proprietary interpreted **Spin programming language** executes approximately 80,000 instruction-tokens per second on each core, giving 8 times 80,000 for 640,000 high level instructions per second. Most machine-language instructions take 4 clock-cycles to execute, resulting in 20 MIPS per cog, or 160 MIPS in total for an 8-cog Propeller.

In addition to lowering the clock rate to that actually required, power consumption can be reduced by turning off cogs (which then use very little power), and by reconfiguring I/O pins which are not needed, or can be safely placed in a high-impedance state ("tristated"), as inputs. Pins can be reconfigured dynamically, but again, the change applies to all cogs, so synchronization is important for certain designs. (Some protection is available for situations where one core attempts to use a pin as an output while another attempts to use it as an input; this is explained in Parallax's technical reference manual.)

On-board peripherals

Each cog has access to some dedicated counter/timer hardware, and a special timing signal generator intended to simplify the design of video output stages, such as composite PAL or NTSC displays (including modulation for broadcast) and VGA monitors. Parallax thus makes sample code available which can generate video signals (text and somewhat low-resolution graphics) using a minimum parts count consisting of the Propeller, a crystal oscillator, and a few resistors to form a crude DAC. The frequency of the oscillator is important, as the correction ability of the video timing hardware is limited to the clock rate. It is possible to use multiple cogs in parallel to generate a single video signal. More generally, the timing hardware can be used to implement various pulse-width modulated (PWM) timing signals.

ROM extensions

In addition to the Spin interpreter and a bootloader, the built-in ROM provides some data which may be useful for certain sound, video, or mathematical applications:

- a bitmap font is provided, suitable for typical character generation applications (but not customizable);
- a logarithm table (base 2, 2048 entries);
- an antilog table (base 2, 2048 entries); and
- a sine table (16-bit, 2049 entries).

The math extensions are intended to help compensate for the lack of a floating-point unit as well as more primitive missing operations, such as multiplication and division (this is masked in Spin but is a limitation for assembly language routines). The Propeller is a 32-bit processor, however, and these tables may not have sufficient accuracy for higher-precision applications.

Built in SPIN byte code interpreter

Spin is a multitasking high level computer programming language created by Parallax's Chip Gracey, who also designed the Propeller microcontroller on which it runs, for their line of Propeller microcontrollers.

Spin code is written on the Propeller Tool, a GUI oriented software development platform written for Windows XP. This compiler converts the Spin code into bytecodes that can be loaded (with the same tool) into the main 32 KB RAM, and optionally into the serial boot FLASH EEPROM, of the Propeller chip. After booting the propeller a bytecode interpreter is copied from the built in ROM into the 2 KB RAM of the primary COG. This COG will then start interpreting the bytecodes in the main 32 KB RAM. More than one copy of the bytecode interpreter can run in other COGs, so several Spin code threads can run simultaneously. Within a Spin code program, assembler code program(s) can be "inline" inserted. These assembler program(s) will then run on their own COG's.

Like Python, Spin uses indentation/whitespace, rather than curly braces or keywords, to delimit blocks.

The Propeller's interpreter for its proprietary multi-threaded SPIN computer language is a byte code interpreter. This interpreter decodes strings of instructions, one instruction per byte, from user code which has been edited, compiled, and loaded onto the Propeller from within a purpose-specific IDE. This IDE, which Parallax simply calls "The Propeller tool", is intended for use under the Windows operating system.

The SPIN language is a high level language. Because it is interpreted in software, it runs slower than pure Propeller assembler but can be more space-efficient (Propeller assembly opcodes are 32 bits long; SPIN directives are 8 bits long, which may be followed by a number of 8-bit bytes to specify how that directive operates). SPIN also allows users to avoid significant memory segmentation issues that must be considered for assembly code.

Mixing SPIN and assembly code is straightforward; SPIN is more appropriate for high-level logic, while assembly routines may be required for I/O routines that require exact timing.

At startup, a copy of the byte code interpreter (less than 2 KB in size), will be copied into the dedicated RAM of a cog and will then start interpreting byte code in the main 32 KB RAM. Additional cogs can be started from that point, loading a separate copy of the interpreter into the new cog's dedicated RAM (a total of eight interpreter threads can, therefore, run simultaneously). Notably, this means that at least a minimal amount of startup code *must* be SPIN code, for all Propeller applications.

Syntax

The syntax of Spin can be broken down into blocks. The blocks are as following.

VAR Holds global variables

CON Holds program constants

PUB Holds code for a public subroutine

PRI Holds code for a private subroutine

OBJ Holds code for objects

Example keywords

reboot: causes the microcontroller to reboot

waitcnt: wait for the system counter to equal or exceed a specified value

waitvid: Waits for a video timing event before outputting video data to I/O pins.

coginit: starts a processor on a new task

Example program

An example program, (as it would appear in the "Propeller Tool" editor) which outputs the current system counter every 3,000,000 cycles, then is shut down by another cog after 40,000,000 cycles:

```
VAR
  long Stk[3] ''declare an array of longs

PUB Main
  cognew(DispCnt, @Stk) ''Acvtivate cog 1 and run the DispCnt routine in it
  ''also pass the adress of the array we created, for use as a call stack
  Waitandstop ''Run Wait routine in this cog (cog 0)

PUB DispCnt
  dira~~ ''set all bits in port a direction register to 1 (output)
  repeat
    waitcnt(3_000_000 + cnt)
    outa := cnt ''move value current system counter to port a

PUB Waitandstop
  waitcnt(40_000_000 + cnt) ''wait until counter = current value + 40,000,000 (wait 40mil clocks)
  cogstop(1) ''stop cog 1
  cogstop(0) ''stop cog 0
```

The Parallax Propeller is gradually accumulating software libraries which give it similar functionality to Parallax's older BASIC Stamp product; however there is no uniform list of which PBASIC facilities now have Spin equivalents.

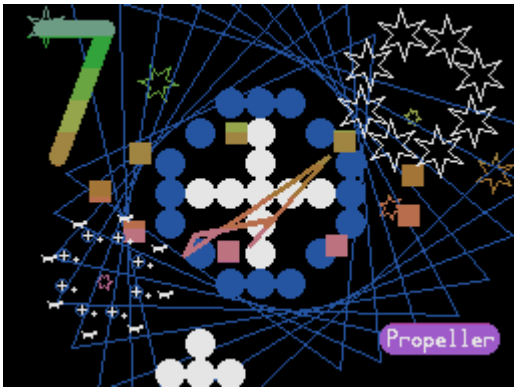
Package and I/O

The initial version of the chip (called the P8X32) provides one 32 bit port in a 40-pin 0.6" DIP, 44-pin LQFP, or QFN package. Of the 40 available pins, 32 are used for I/O, four for power and ground pins, two for an external crystal (if used), one to enable brownout-detection, and one for reset.

All 8 cores can access the 32-bit port (designated "A"; there is currently no "B") simultaneously. A special control mechanism is used to avoid I/O conflicts if one core attempts to use an I/O pin as an output while another tries to use it as input. Any of these pins can be used for the timing or pulse-width modulation output techniques described above.

Parallax has stated that it expects later versions of the Propeller to offer more I/O pins and/or more memory.

Virtual I/O devices



Screen capture of the graphics demo that Parallax created to demonstrate the NTSC library

The Propeller's designers designed it around the concept of "virtual I/O devices". For example, the "HYDRA Game Development Kit", (a computer system designed for hobbyists, to learn to develop "retro-style" video games) uses the built in character generator and video support logic to generate a virtual Video display generator that outputs VGA colour pictures, PAL/NTSC compatible colour pictures or broadcast RF video+audio in software.

The screen capture displayed here was made using a software "virtual display driver" that sends the pixel data over a serial link to a PC.

Software libraries are available to implement several I/O devices ranging from simple UARTs and Serial I/O interfaces such as SPI, I2C and PS/2 compatible serial mouse and keyboard interfaces, motor drivers for robotic systems, MIDI interfaces and LCD controllers.

Dedicated cores instead of interrupts

The design philosophy of the Propeller is that a hard real-time, multi-core architecture negates the need for dedicated interrupt hardware and support in assembly. In traditional CPU architecture external interrupt lines are fed to an on-chip interrupt controller and serviced by one or more interrupt service routines. When an interrupt occurs the interrupt controller suspends normal CPU processing and saves internal state (typically on the stack) then vectors to the designated interrupt service routine. After handling the interrupt the service routine executes a "return from interrupt" instruction which restores the internal state and resumes CPU processing.

To handle an external signal promptly on the Propeller, any one of the 32 I/O lines is configured as an input. A cog is then configured to wait for a transition (either positive or negative edge) on that input using one of the two counter circuits available to each cog. While waiting for the signal, the cog operates in low-power mode, essentially sleeping.

Extending this technique, a Propeller can be set up to respond to eight independent "interrupt" lines with essentially zero handling delay. Alternately, a single line can be used to signal the "interrupt" and then additional input lines can be read to determine the nature of the event. The code running in the other cores is not affected by the interrupt handling cog. Unlike a traditional multitasking single-processor interrupt architecture, the signal response timing remains predictable , and indeed using the term "interrupt" in this context can cause confusion, since this functionality can be more properly thought of as polling with a zero loop time.

Boot mechanism

On power up, brownout detection, software reset, or external hardware reset, the Propeller will load a machine-code boot routine from the internal ROM into the RAM of its first (primary) cog and execute it. This code emulates an I^2C interface in software, temporarily using two I/O pins for the needed serial clock and data signals to load user code from an external I^2C EEPROM.

Simultaneously, it emulates a serial port, using two other I/O pins that can be used to upload software directly to RAM (and optionally to the external EEPROM). If the Propeller does not see any commands from the serial port, it will load the user program (the entry code of which must be written in SPIN, as described above) from the serial EEPROM into the main 32K RAM. After that it will load the SPIN interpreter from its built-in ROM into the dedicated RAM of its first cog, thereby overwriting most of the bootloader.

Regardless of how the user program is loaded, execution starts by interpreting initial user bytecode with the SPIN interpreter running in the primary cog. After this initial SPIN code runs, the application can turn on any unused cog to start a new thread, and/or start assembler code routines.

External persistent memory

The Propeller boots from an external serial EEPROM; once the boot sequence completes, this device may be accessed as an external peripheral.

C compiler

There is also a C compiler available from ImageCraft, the ICCV7 for Propeller. It supports the 32K Large Memory Model, to bypass the 2K limitation per cog, and is typically 5 to 10 times faster than standard SPIN code. A free ANSI C compiler called Catalina is also available. It is based on LCC.

Propeller and Java

There is also a movement in place to put the JVM on Propeller. A compiler, debugger, and emulator are being developed.

Future versions

Parallax is currently building a new Propellerwith cogs that each will run at about 160 MIPS, whereas the current Propeller's cogs each run at around 20 MIPS. The improved performance would result from a maximum clock speed increase to 160 MHz (from 80 MHz) and an architecture that pipelines instructions, achieving an average execution of nearly one instruction per clock cycle (approx. 4x increase).

Chapter-7

Intel MCS-51 and Intel MCS-48

Intel MCS-51

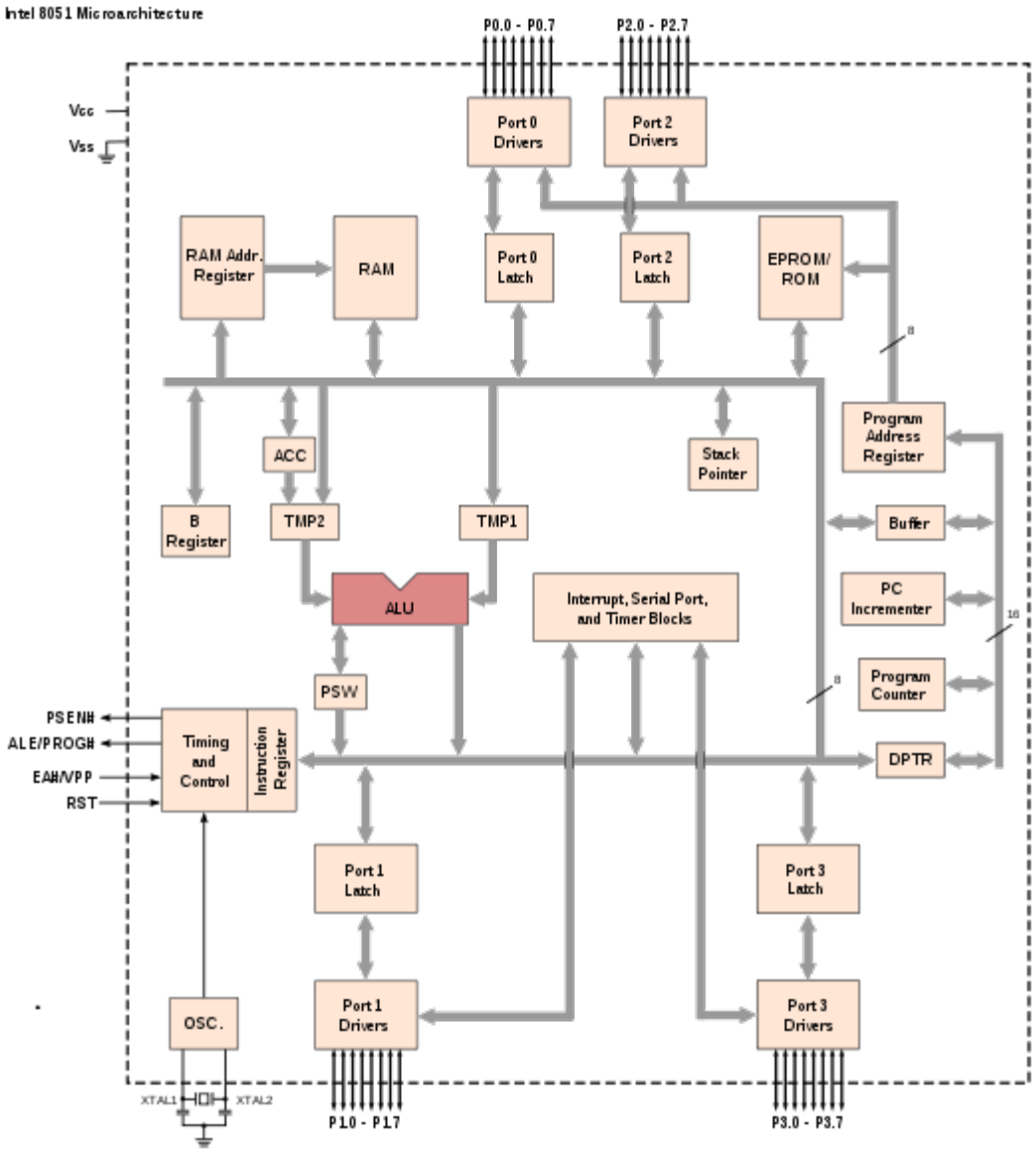
Intel P8051 microcontroller

SAB-C515-LN by Infineon is based on the 8051

The **Intel MCS-51** is a Harvard architecture, single chip microcontroller (µC) series which was developed by Intel in 1980 for use in embedded systems. Intel's original versions were popular in the 1980s and early 1990s, but has today largely been superseded by a vast range of faster and/or functionally enhanced 8051-compatible devices manufactured by more than 20 independent manufacturers including Atmel, Infineon Technologies (formerly Siemens AG), Maxim Integrated Products (via its Dallas Semiconductor subsidiary), NXP (formerly Philips Semiconductor), Nuvoton (formerly Winbond), ST Microelectronics, Silicon Laboratories (formerly Cygnal), Texas Instruments, Ramtron International, Silicon Storage Technology, and Cypress Semiconductor.

Intel's original MCS-51 family was developed using NMOS technology, but later versions, identified by a letter C in their name (e.g., 80C51) used CMOS technology and were less power-hungry than their NMOS predecessors. This made them more suitable for battery-powered devices.

Important features and applications



i8051 microarchitecture.

The 8051 architecture provides many functions (CPU, RAM, ROM, I/O, interrupt logic, timer, etc.) in a single package

- 8-bit ALU, Accumulator and 8-bit Registers; hence it is an 8-bit microcontroller
- 8-bit data bus – It can access 8 bits of data in one operation
- 16-bit address bus – It can access 2^{16} memory locations – 64 KB (65536 locations) each of RAM and ROM
- On-chip RAM – 128 bytes (data memory)
- On-chip ROM – 4 kByte (program memory)
- Four byte bi-directional input/output port

- UART (serial port)
- Two 16-bit Counter/timers
- Two-level interrupt priority
- Power saving mode (on some derivatives)

A particularly useful feature of the 8051 core is the inclusion of a boolean processing engine which allows bit-level boolean logic operations to be carried out directly and efficiently on internal registers and RAM. This feature helped cement the 8051's popularity in industrial control applications. Another valued feature is that it has four separate register sets, which can be used to greatly reduce interrupt latency compared to the more common method of storing interrupt context on a stack.

The MCS-51 UARTs make it simple to use the chip as a serial communications interface. External pins can be configured to connect to internal shift registers in a variety of ways, and the internal timers can also be used, allowing serial communications in a number of modes, both synchronous and asynchronous. Some modes allow communications with no external components. A mode compatible with an RS-485 multi-point communications environment is achievable, but the 8051's real strength is fitting in with existing ad-hoc protocols (e.g., when controlling serial-controlled devices).

Once a UART, and a timer if necessary, have been configured, the programmer needs only to write a simple interrupt routine to refill the *send* shift register whenever the last bit is shifted out by the UART and/or empty the full *receive* shift register (copy the data somewhere else). The main program then performs serial reads and writes simply by reading and writing 8-bit data to stacks.

MCS-51 based microcontrollers typically include one or two UARTs, two or three timers, 128 or 256 bytes of internal data RAM (16 bytes of which are bit-addressable), up to 128 bytes of I/O, 512 bytes to 64 kB of internal program memory, and sometimes a quantity of extended data RAM (ERAM) located in the external data space. The original 8051 core ran at 12 clock cycles per machine cycle, with most instructions executing in one or two machine cycles. With a 12 MHz clock frequency, the 8051 could thus execute 1 million one-cycle instructions per second or 500,000 two-cycle instructions per second. Enhanced 8051 cores are now commonly used which run at six, four, two, or even one clock per machine cycle, and have clock frequencies of up to 100 MHz, and are thus capable of an even greater number of instructions per second. All SILabs, some Dallas and a few Atmel devices have single cycle cores.

Common features included in modern 8051 based microcontrollers include built-in reset timers with brown-out detection, on-chip oscillators, self-programmable Flash ROM program memory, bootloader code in ROM, EEPROM non-volatile data storage, I²C, SPI, and USB host interfaces, CAN or LIN bus, PWM generators, analog comparators, A/D and D/A converters, RTCs, extra counters and timers, in-circuit debugging facilities, more interrupt sources, and extra power saving modes.

Memory Architecture

The MCS-51 has four distinct types of memory – internal RAM, special function registers, program memory, and external data memory.

Internal RAM (IRAM) is located from address 0 to address 0xFF. IRAM from 0x00 to 0x7F can be accessed directly, and the bytes from 0x20 to 0x2F are also bit-addressable. IRAM from 0x80 to 0xFF must be accessed indirectly, using the @R0 or @R1 syntax, with the address to access loaded in R0 or R1.

Special function registers (SFR) are located from address 0x80 to 0xFF, and are accessed directly using the same instructions as for the lower half of IRAM. Some of the SFR's are also bit-addressable.

Program memory (PMEM, though less common in usage than IRAM and XRAM) is located starting at address 0. It may be on- or off-chip, depending on the particular model of chip being used. Program memory is read-only, though some variants of the 8051 use on-chip flash memory and provide a method of re-programming the memory in-system or in-application. Aside from storing code, program memory can also store tables of constants that can be accessed by MOVC A, @DPTR, using the 16-bit special function register DPTR.

External data memory (XRAM) also starts at address 0. It can also be on- or off-chip; what makes it "external" is that it must be accessed using the MOVX (Move eXternal) instruction. Many variants of the 8051 include the standard 256 bytes of IRAM plus a few KB of XRAM on the chip. If more XRAM is required by an application, the internal XRAM can be disabled, and all MOVX instructions will fetch from the external bus.

Programming

There are various high-level programming language compilers for the 8051. Several C compilers are available for the 8051, most of which feature extensions that allow the programmer to specify where each variable should be stored in its six types of memory, and provide access to 8051 specific hardware features such as the multiple register banks and bit manipulation instructions. There are many commercial C compilers. SDCC is a popular open source C compiler. Other high level languages such as Forth, BASIC, Pascal/Object Pascal, PL/M and Modula-2 are available for the 8051, but they are less widely used than C and assembly.

Because IRAM, XRAM, and PMEM all have an address 0, C compilers for the 8051 architecture provide compiler-specific pragmas or other extensions to indicate where a particular piece of data should be stored (i.e. constants in PMEM or variables needing fast access in IRAM). Since data could be in one of three memory spaces, a mechanism is usually provided to allow determining to which memory a pointer refers, either by constraining the pointer type to include the memory space, or by storing metadata with the pointer.

Instruction set

The MCS-51 instruction set offers several addressing modes, including

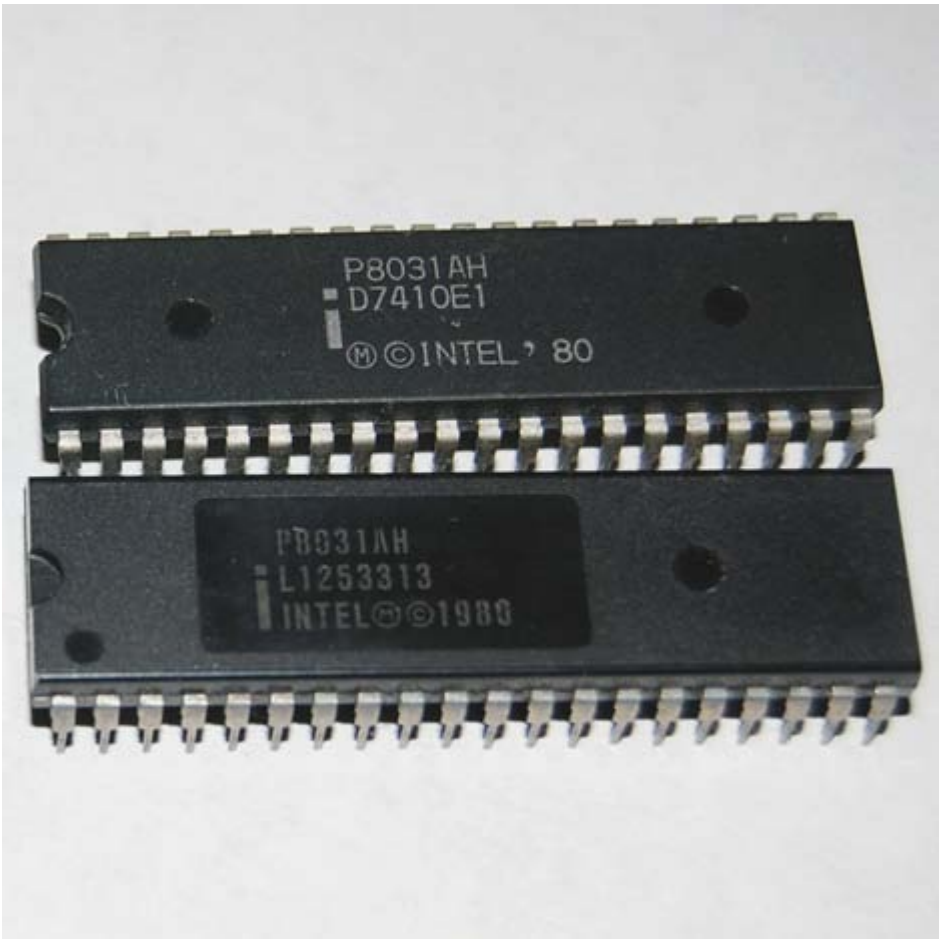
- direct register, using ACC (the accumulator) and R0-R7
- direct memory, which access the internal RAM or the SFR's, depending on the address
- indirect memory, using R0, R1, or DPTR to hold the memory address. The instruction used may vary to access internal RAM, external RAM, or program memory.
- individual bits of a range of IRAM and some of the SFR's

Many of the operations allow any addressing mode for the source or the destination, for example, MOV 020h, 03fh will copy the value in memory location 0x3f in the internal RAM to the memory location 0x20, also in internal RAM.

Because the 8051 is an accumulator-based architecture, all arithmetic operations must use the accumulator, e.g. ADD A, 020h will add the value in memory location 0x20 in the internal RAM to the accumulator.

One does not need to master these instructions to program the 8051. With the availability of good quality C compilers, including open source SDCC, virtually all programs can be written with high-level language.

Related processors



Intel 8031 processors

The 8051's predecessor, the 8048, was used in the keyboard of the first IBM PC, where it converted keypresses into the serial data stream which is sent to the main unit of the computer. The 8048 and derivatives are still used today for basic model keyboards.

The *8031* was a cut down version of the original Intel 8051 that did not contain any internal program memory (ROM). To use this chip, external ROM had to be added containing the program that the 8031 would fetch and execute. An 8051 chip could be sold as a ROM-less 8031, as the 8051's internal ROM is disabled by the normal state of the EA pin in an 8031-based design. A vendor might sell an 8051 as an 8031 for any number of reasons, such as faulty code in the 8051's ROM, or simply an oversupply of 8051's and undersupply of 8031's.

The *8052* was an enhanced version of the original 8051 that featured 256 bytes of internal RAM instead of 128 bytes, 8 KB of ROM instead of 4 KB, and a third 16-bit timer. The *8032* had these same features except for the internal ROM program memory. The 8052 and 8032 are largely considered to be obsolete because these features and more are included in nearly all modern 8051 based microcontrollers.

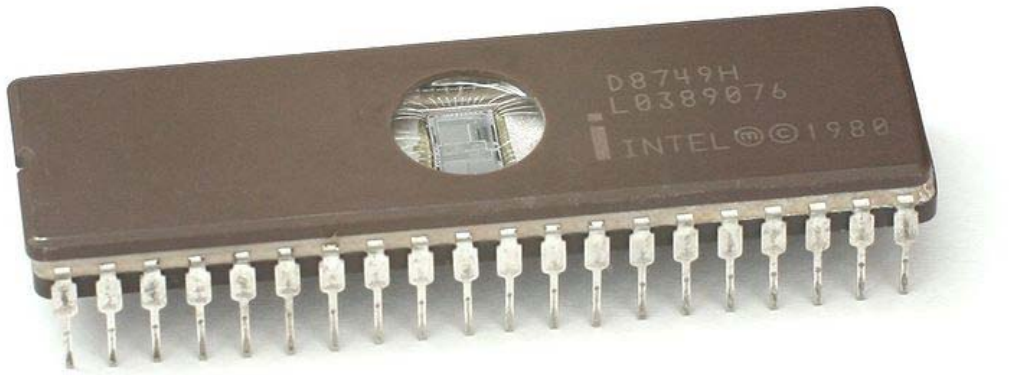
Intel discontinued its MCS-51 product line in March 2007, however there are plenty of enhanced 8051 products or silicon IP added regularly from other vendors.

The 80C537 and 80C517 are CMOS versions, designed for the automotive industry. Enhancements mostly new peripheral features and expanded arithmetic instructions. The 80C517 has fail save mechanisms, analog signal processing facilities and timer capabilities and 8 KB on-chip program memory. Other features include:

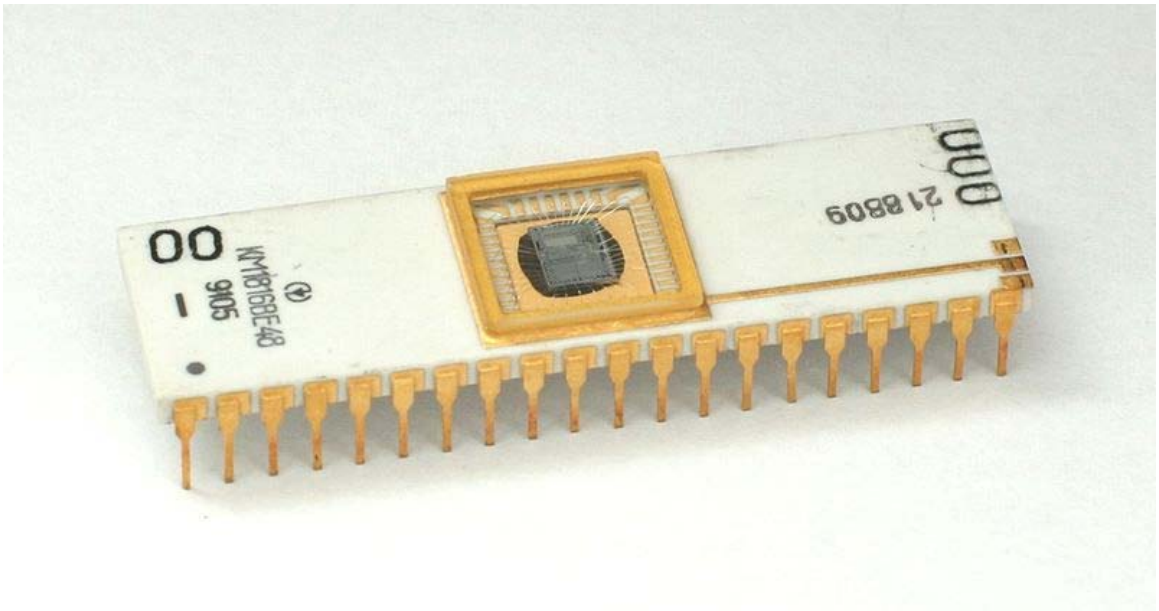
- 256 byte on-chip RAM
- 256 directly addressable bits
- External program and data memory expandable up to 64 KB
- 8-bit A/D converter with 12 multiplexed inputs
- Arithmetic unit can make division, multiplication, shift and normalize operations
- Eight data pointers instead of one for indirect addressing of program and external data memory
- Extended watchdog facilities
- Nine ports
- Two full-duplex serial interfaces with own baud rate generators
- Four priority level interrupt systems, 14 interrupt vectors
- Three power saving modes

Intel MCS-48

Intel 8048 microcontroller



The 8749 with UV EPROM



USSR KM1816BE48, an Intel 8748 clone.

The **MCS-48** microcontroller (μC) series, Intel's first microcontroller, was originally released in 1976. Its first members were 8048, 8035 and 8748.

The MCS-48 series has a Modified Harvard architecture, with internal or external program ROM and 64–256 bytes of internal (on-chip) RAM. The I/O is mapped into its own address space, separate from programs and data. The 8048 is probably the most

prominent member of Intel's MCS-48 family of microcontrollers. It was inspired by, and is somewhat similar to, the Fairchild F8 microprocessor.

Though the MCS-48 series was eventually replaced by the very popular Intel MCS-51, even at the turn of the millennium it remains quite popular, due to its low cost, wide availability, memory efficient one-byte instruction set, and mature development tools. Because of this it is much used in high-volume consumer electronics devices such as TV sets, TV remotes, toys, and other gadgets where cost-cutting is essential.

Variants

The *8049* has 2 KB of masked ROM (the *8748* and *8749* had EPROM) that can be replaced with a 4 KB external ROM, as well as 128 bytes of RAM and 27 I/O ports. The µC's oscillator block divides the incoming clock into 15 internal phases, thus with its 11 MHz max. crystal one gets 0.73 MIPS (of one-clock instructions). Some instructions are single byte/cycle ones, but a large amount of opcodes need two cycles and/or two bytes, so the raw performance would be closer to 0.5 MIPS.

Another variant, the ROM-less *8035*, was used in Nintendo's arcade game Donkey Kong. Although not being a typical application for a microcontroller, its purpose was to generate the background music of the game.

The Intel *8748* has on-chip clock oscillator, 2 8-bit timers, 27 I/O ports, 64 bytes of RAM and 1 KB of EPROM. A version with 2 KB EPROM and 128 bytes RAM was also available under the *8749* number.

Device	Internal	Memory	Remarks
8020	1K x 8 ROM	64 x 8 RAM	Subset of 8048, 20 pins, Only 13 I/O Lines
8021	1K x 8 ROM	64 x 8 RAM	Subset of 8048, 28 pins ,21 I/O Lines
8022	2K x 8 ROM	64 x 8 RAM	Subset of 8048, A/D-converter
8035	none	64 x 8 RAM	
8039	none	128 x 8 RAM	
8040	none	256 x 8 RAM	
8048	1K x 8 ROM	64 x 8 RAM	
8049	2K x 8 ROM	128 x 8 RAM	
8050	4K x 8 ROM	256 x 8 RAM	
8748	1K x 8 EPROM	64 x 8 RAM	
8749	2K x 8 EPROM	128 x 8 RAM	
8648	1K x 8 OTP EPROM	64 x 8 RAM	Factory OTP EPROM
Device	Internal	Memory	Remarks
8041	1K x 8 ROM	64 x 8 RAM	Universal Peripheral Interface (UPI)
8041AH	1K x 8 ROM	128 x 8 RAM	UPI
8741A	1K x 8 EPROM	64 x 8 RAM	UPI, EPROM version of 8041
8741AH	1K x 8 OTP EPROM	128 x 8 RAM	UPI, OTP EPROM version of 8041AH
8042AH	2K x 8 ROM	256 x 8 RAM	UPI
8742	2K x 8 EPROM	128 x 8 RAM	UPI, EPROM version

```
8742AH  2K x 8 OTP EPROM  256 x 8 RAM   UPI, OTP EPROM version of
8042AH
```

Uses

The 8048 was used in the Magnavox Odyssey² video game console, the Korg Trident series, Roland Jupiter-4 and Roland ProMars analog synthesizers.

The original IBM PC keyboard used an 8048 as its internal microcontroller. The PC AT replaced the PC's Intel 8255 peripheral interface chip at I/O port addresses `0x60-63` with an 8042 accessible through port addresses `0x60` and `0x64`. As well as managing the keyboard interface the 8042 controlled the A20 line of the AT's Intel 80286 CPU, and could be commanded by software to reset the 80286 (unlike the 80386 and later processors, the 80286 had no way of switching from protected mode back to real mode except by being reset). Later PC compatibles integrate the 8042's functions into their super I/O devices.

Chapter-8

PICAXE

PICAXE is the name of a UK-sourced microcontroller system based on a range of Microchip PICs. There are 13 PICAXE variants of differing pin counts from 8 to 40 pins. Initially marketed for use in education and by electronics hobbyists, they are also used in commercial and technical fields, including rapid prototype development. All use pre-loaded factory bootstrap interpretation code to allow user generated programs to be downloaded using a simple USB or RS-232 serial connection.

Hardware

General

Based upon a variety of Microchip PICs, the chips come in a number of DIP footprints, from an 8-pin, 128-byte program capacity device through to a 40-pin, 4096-byte program capacity device. With their DIP footprint they are suited for use with solderless breadboards and more traditional PCBs designs, although surface mount versions are available.

The 8-pin PICAXE-08M is priced below GB£2 and is often chosen as an entry-level option as it lends itself to easy prototyping.

The current recommended parts, as of November 2009, are 08M, 14M, 18M, 20M, 18X, 28X1, 40X1, 20X2, 28X2, 40X2.

As of 12 August 2010, Rev Ed has released the 18M2. This chip is promoted as a replacement for the 18X, 18M.

The 28, 28A, 28X and 40X have been discontinued as they have been superseded by the X1 parts. The 18 and 18A have been discontinued as they have been superseded by the 18M. The 08, 18, 18A are still available but are not recommended for new designs.

All current devices use an internal 4/8 MHz oscillator and hence require few components to create a basic hardware platform. The X1 and X2 parts also have the ability to operate

from 31 kHz to 8 MHz in 7 steps using internal low frequency oscillators in addition to the internal resonator.

The X2 parts can also operate at higher speeds from the internal oscillators. The X1 and X2 parts can also use an external resonator for up to 20 MHz (X1 parts) and up to 64 MHz for the X2 parts.

The M2 Series PICAXE Chips

Rev Ed is releasing a new M2 series of chips. The 18M2 is the first to be released which was available from 12 August 2010. The 14M2 and 20M2 will be available at a future time.

Features of the 18M2 include:

- Almost every pin is individually configurable, so, for instance, the 18M2 can now have 13 outputs instead of 8. Therefore M2 chips can be used in either the 'traditional' fixed pin format or the new flexible 'user configured' format.
- Many extra ADC channels are now also available on other pins, with new support for touch sensor inputs.
- The reset and serial input pins can now be used as 2 extra input pins, giving 2 more general purpose input pins. New software 'reset' command if required.
- It can now run four separate tasks in parallel, but limited to a single core. This allows BASIC programs to execute diferent tasks at the same time, as well as having four separate starts in the Logicator flowcharting program.
- The 18M2 device is meant to replace the older 18, 18A, 18M and 18X parts.
- It gives eight times the equivalent memory capacity (2048 bytes, up to 1800 lines of program) at the same price as the older 256 byte 18A/18M, this is due to the effect of the Moore's law, which states that the amount of transistor that can be placed inside an integrated circuit is doubled every two years.
- The 18M2 incorporates I²C commands (but not 1-wire) for use with an external RTC, memory and other chips.
- Fully backwards compatible with all existing 18 pin PICAXE project boards and programs written for any older 18 pin PICAXE part.
- It can operate at 1.8V, saving considerable space and cost in portable designs. However, being this technology relatively new to the hobbyist market, most of the parts use 3.3V or 5V for operation.
- 28 general purpose byte variables, as opposed to the 14 previously available on older parts, with up to a total of 256 bytes of RAM.
- New 'time' variable counts elapsed seconds.
- 256 bytes of non-volatile data EEPROM memory.
- Faster internal resonator (up to 32 MHz) means up to 8x faster program processing.
- Full support for common features such as ring tone tunes, servos, digital temperature sensors and infra-red input and output on any pin, most of these using inbuilt BASIC program commands.

- Full support for advanced features like 5 bit DAC, SR latch, hardware serial commands (for much faster baud rates) and PWM control of motors.

Minimal configurations

All current devices require nothing more than the connection of power and configuration of the Serial In pin used for downloading.

The 18, 28 and 40-pin devices (but not the 18M2) requires a pull-up resistor from the Reset pin. The early PICAXE 28 and 40X chips require the connection of a resonator or crystal.

Power supply requirements

Power supply voltages are very flexible allowing operation from batteries or regulated power supplies. Most of the PICAXE range is nominally 4.5 Vdc to 5 Vdc but can operate down to approximately 3 Vdc. The X2 range also includes special low power (1.8 Vdc to 3.3 Vdc) variants.

The new M2 series PICAXE chips can operate from 1.8 Vdc to 5 Vdc

Low power modes

There are a number of commands (SLEEP, NAP, HIBERNATE, DOZE) to put the device into low power operating modes in order to conserve power and extend operational lifetime when powered by batteries.

Many of the variants have controllable clocks, allowing for operation below their nominal operating frequencies, to achieve minimal lower power consumption. Execution speed can be controlled by the user program.

Non-volatile data storage

All devices contain non-volatile data memory which allows data to be stored and recovered when power is removed. Access to the non-volatile data memory is through the use of the READ and WRITE commands and, on the 28A, the READMEM and WRITEMEM commands can also be used.

The amount of non-volatile data memory available depends upon the device:

- The 08 and 18 have 128 bytes of EEPROM which is used to store both downloaded program and for non-volatile data memory use.
- The 08M, 14M, 18M, 20M have 256 bytes of EEPROM which is used to store both downloaded program and for non-volatile data memory use.
- The 18A, 18X snd 18M2 have 256 bytes of EEPROM exclusively available for non-volatile data memory use.

- The 28 has 64 bytes of EEPROM exclusively available for non-volatile data memory use.
- The 28A has 64 bytes of EEPROM exclusively available for non-volatile data memory use and also has 256 bytes of Flash memory which may be used for non-volatile data storage using the READMEM and WRITEMEM commands.
- The 28X and 40X have 128 bytes of EEPROM exclusively available for non-volatile data memory use.
- The 28X1 and 40X1 have 4096 bytes of program space, double the variable memory and athe addition of 128 bytes of scratchpad memory. These also many new features for IO etc.
- The 20X2, 28X2 and 40X2 a slightly different arrangement with up to 4 internal program memory slots, 32 external memory slots, 1024 bytes of scratchpad memory and more versatile bidirectional IO.

PICAXE chip factory marking/identification

- PICAXE-08 PIC12F629
- PICAXE-08M PIC12F683
- PICAXE-14M PIC16F684
- PICAXE-18M PIC16F819
- PICAXE-18X PIC16F88
- PICAXE-18M2 PICAXE 18M2
- PICAXE-20M PIC16F677
- PICAXE-20X2 PIC18F14K22
- PICAXE-28X1 PIC16F886
- PICAXE-28X2 PIC18F2520 (3V version PIC18F25K20)
- PICAXE-40X1 PIC16F887
- PICAXE-40X2 PIC18F4520 (3V Version PIC18F45K20)
- PICAXE-18 PIC16F627(A) Superseded by 18M
- PICAXE-18A PIC16F819 Superseded by 18M
- PICAXE-28A PIC16F872 Superseded by 28X1
- PICAXE-28X PIC16F873A Superseded by 28X1
- PICAXE-40X PIC16F874A Superseded by 40X1

Note that the PICAXE-18M2 is a new custom part factory manufactured by Microchip Inc. for Revolution Education and so is factory engraved with the full PICAXE-18M2 name. This was done to avoid confusion in educational environments.

Software

All programming development is done using the Programming Editor or the AXEpad editor. These are both free software downloads supporting the entire development process from source code editing to program downloading. Microsoft (Windows 98 and later), Linux and Macintosh operating systems are supported. Historically a serial port was required to download programs, however a USB download cable is now available from

the developers. The cable incorporates a USB-serial adapter moulded into its plug, eliminating the need for a separate adapter for users of modern USB-only PCs.

The Programming Editor and AXEpad also include a number of Wizards which aid in program and project development.

Editing can be done in plain text or RTF with color syntax highlighting. Source code can be created outside the Programming Editor and imported for downloading. Programs can be created in a Basic-like language or by using a visual flowcharting tool. The Programming Editor also supports PICmicro Assembly Language programming.

Simulator

The Programming Editor software includes a comprehensive line by line on-screen simulation of the BASIC program. This enables users to step through the program on-screen, to watch the program in operation, and help identify any programming errors.

Tune wizard

This Wizard is used to create TUNE commands which can be used with the 08M, 14M, 18M, 20M, 28X1 and 40X1. The Wizard includes the ability to import suitable mobile telephone ring tones and convert them to appropriate TUNE commands.

Serial LCD CGRAM wizard

This Wizard provides a visual means to create custom defined characters to be used with serial LCD displays. The desired character is specified by selecting pixels which will be displayed on a 7x5 grid and the necessary command to program that character within the LCD is generated.

Datalogger wizard

This Wizard is used to generate datalogging programs.

There is a Datalogger sub-wizard to take the current PC time and create a small program to set the time in a DS1307 (or compatible) RTC. The wizard will then download to the PICAXE chip and run the program. While ostensibly for the AXE110 datalogger this will also work with other i2c enabled PICAXE chips.

PICAXE connect wizard

This Wizard allows the configuration of MaxStream XBee (ZigBee) modules to be configured for use.

PICAXE net server wizard

This Wizard allows configuration of the PICAXE.net Web Server product. The Wizard provides the ability to create and upload page images and to upgrade the web server firmware.

Programming language

The programming language is BASIC-like and very similar to that used by the Basic Stamp 1 (BS1). Most programs written for the BS1 should be easily convertible for use. The most notable difference is that the BS1's POT command has been replaced by the READADC command which allows an analogue voltage to be read directly. READADC10 allows analogue inputs to be read to 10-bit resolution.

The programming language includes high-level support for underlying processor capabilities and additional functionality for various devices. All variants support a common core programming command set but not all support all commands.

Variables

The programming language provides 14 byte variables on smaller PICAXE chips, 28 byte variables for X1 and M2 parts and 56 byte variables for the latest X2 parts through the Programming Editor. This memory can be manipulated in measures of bits (as bit0-15 for smaller parts at bit0-31 for larger parts), bytes (b0-b13, b27 or b55), and 16-bit words (w0-6, w0-13 and w0-27). The bit, byte, and word variables all overlap in the same memory area. Every two byte variables overlap with a word variable, so b0 & b1 make up w0 (where b1 is the most significant byte), likewise b2 & b3 compose w1 (where b3 is the most significant byte), and so on. The bit values bit0 to bit7 overlap b0, and bit8 to bit15 overlap b1, etc (which also covers w0). This can be used to carve apart variables, or simply make the most efficient use of the limited memory given.

Variables can be given meaningful names (aliases) for use within a program through use of the SYMBOL directive. The SYMBOL directive can also be used to create named constants.

In addition to the pre-defined variables, all but the 08 have access to the some of the internal SFR (Special Function Registers) and General Purpose Registers of the PICmicro they are based upon, allowing many of the unused Registers to be used as Random Access Memory (RAM) during execution. This can be used for temporary storage of variable values (using PEEK and POKE) and allows re-use of variables within subroutines and other code sections. The smaller PICAXE parts have 48 bytes free for use, while the other parts have access to greater number of registers enhanced parts as 95/96 bytes for the 18X, 28X1 and 40X1, 112 bytes for the 28X and 40X, 72 bytes for the 20X2 and 200 bytes for the 28X2/40X2.

The PEEK and POKE commands can also be used to implement byte arrays and software stacks.

The 20X2, 28X1, 40X1 parts also have 128 bytes of 'scratchpad' memory available while the 28X2 and 40X2 parts have 1024 bytes of ‘scratchpad’ memory. The scratchpad memory can be accessed directly (PUT and GET commands) or indirectly via the scratchpad pointer '@ptr'.

The READ and WRITE commands, which allow data to be stored in and retrieved from non-volatile EEPROM memory, can be used to retain data and settings while powered down.

Arithmetical manipulation

All arithmetical operations are performed using 16-bit, unsigned, positive only operations. Variables are expanded as required to 16-bits on use by leading zero padding and results of processing are stored by truncating the resultant value to an appropriate number of bits before storage. A byte variable will have the eight least significant bits of the result stored, a bit variable will be set to the least significant bit value of the result.

Note that when using byte size variables, the internal maths are done using 16-bits so intermediate results on a line can exceed the maximum byte value of 255 but must not exceed 65535 otherwise erroneous results will occur.

Care must therefore be taken when manipulating variables and values to consider the effect of wrap-round, underflow and overflow. In particular it must be noted that a value can never be less than zero, and a byte value can never exceed 255. These issues must be considered in the implementation of FOR-NEXT loops and other looping constructs, which may, under some circumstances, never meet their terminating conditions. There is no facility to automatically report or indicate wrap-round, underflow or overflow.

All arithmetical expressions in assignments (LET) are evaluated in a strictly left-to-right manner. There is no operator precedence. Note that the keyword "LET" is optional.

In the future, the use of parenthesis for maths precedence to control calculations will be possible on the X1 and X2 parts. This is currently awaiting an update of the Programming Editor by the PICAXE makers, Rev Ed. The intention to include parenthesis was made over a year ago in the yahoo picaxe forum but to date there has been no mention of any intention.

The 28X1 and 40X1 parts have additional unary maths functions which include: Sin, Cos, Sqr, Inv, NCD, DCD, BintoBCD and BCDtoBin functions. When unary maths functions are used, they MUST be the first command on a program line. The unary functions can be followed by additional "normal" maths functions on the same line. For example:

```
LET b1 = COS 30 + 55 / 10
```

is valid

Users should check picaxe documents before using COS, SIN etc to be sure that the table lookpup method that it uses is accurate for their purposes.

Program flow control

A number of control constructs are provided to handle program flow -

- GOTO - Redirects program execution to another location in the program.
- IF-THEN - Not the usual "IF *condition* THEN *statement*". The only *statement* IF/THEN takes is a Label: , which it does a "GOTO" when the *condition* is true. IF/THEN falls to the next line when the *condition* is false.
- BRANCH - Conditionally redirects program execution to one of a number of locations in the program (same as ON-GOTO).
- GOSUB - Redirects program execution to another location in the program, then continues after the GOSUB command following the execution of a RETURN in the subroutine.
- RETURN - Returns program execution to after the most recent GOSUB.
- SETINT - Enables polled interrupts and automatically redirects program execution to an interrupt handler when an interrupt is detected.
- SETINTFLAGS - Enables polled interrupts on certain flag-byte conditions.

```
For both SETINT and SETINTFLAGS, program execution continues from where
the program was diverted from upon execution
of a RETURN within the interrupt handler.
```

- FOR-NEXT - Repeats a section of code while a variable value is within a specified range.

A number of block structured constructs and constructs found in other BASIC language dialects exist -

- IF-THEN-ELSE-ENDIF - Selects one of two paths of execution dependent upon conditions. Also supports an ELSEIF clause.
- SELECT-CASE-ENDSELECT - Selects one of a number of paths of execution depending upon value of a variable matched against a list of conditions.
- DO-LOOP - Conditionally repeats a section of code. Allows construction of WHILE-DO and REPEAT-UNTIL loops.
- ON-GOTO - Conditionally redirects program execution to one of a number of locations in the program.
- ON-GOSUB - Conditionally calls one of a number of subroutines in the program.

Program size

The size of program permitted is dictated by the on-chip EEPROM or Flash capacity.

- The 08 and 18 have 128 bytes of program memory allowing programs of approximately 40 lines of source code.
- The 08M, 18A, 18M, 28 and 28A have 256 bytes of program memory allowing programs of approximately 80 lines of source code.
- the 18M2 has 2048 bytes allowing 800 to 1800 lines of of source code
- The X-range have 2048 bytes of program memory allowing approximately 600 lines of source code.
- The X1 parts have 4096 bytes of program memory allowing approximately 800 to 1800 lines of source code.
- The X2 parts are based upon "slots" for program space where each "slot" has 4096 bytes of program memory available.

The 20X2 has one internal slot (#0) and can use one external slot (#4) in an i2c EEPROM (24LC128 or larger).

The 28X2 and 40X2 parts have 4 internal slots (#0 to #3) and can access three external slots (#4 to #7) in an i2c EEPROM (24LC128 or larger) allowing 2000 to 3200 lines of source code over the total of 4 internal program "slots".

Because the program downloaded is stored as variable length tokens, it is not possible to easily predict the size of program which will be generated from any given source code. In particular, the program size will vary depending upon what value constants are used and which input and output pins are referenced in various commands. The Programming Editor does however provide a Syntax Check function which allows the size of program generated to be determined without having to download the program. This also allows programs to be developed and checked even without access to target hardware.

The variable length size of tokens, coupled with the inability to predict their alignment within the program memory also means that it is not possible to accurately predict the execution speed of any particular command, although typically this will be a minimum of 250 microseconds at normal 4 MHz operating speed. To date there has been no public acknowledgment by Revolution Education as to the "specific" speed of execution of their commands, and how their system may compare to compiling basic into hex directly. This makes it difficult for users to determine how well a PICAXE system compares to other forms of system design.

Subroutine nesting

For non “X” part PICAXE chips, a maximum of 16 subroutine call statements (GOSUB) is supported within a program (15 if interrupts are supported). The X, X1 and X2 ranges support up to 256 GOSUB statements.

Subroutines can be nested to a depth of 4 levels for most PICAXE chips and to a depth of 8 levels for the X1 and X2 parts. One level of depth must be reserved for interrupt handler use when interrupts are enabled.

Illusion of limited capabilities

Although these are constrained devices, with a limited number of variables, limitations in the programming command structures and, on lower-end devices, limited program memory, these limitations have not prevented many successful projects and applications from emerging. The PICAXE is increasingly being used in (and to support) many commercial products.

Cost Effectiveness

Provided as a range of devices, each offering differing capabilities and functionality, projects and designs can be tailored to meet requirements such as cost.

Users should be aware that the cost of a PICAXE, whilst being claimed to be cheap, may not in the long run be cheaper than simply programming blank PIC processors via hardware such as PicStart plus or via a home made programmer. A quick examination of the costs of PICAXE chips compared to the price of blank PICs from Microchip demonstrates this. Using multiple devices also has many advantages in system design and modularisation.

For users unfamiliar with microcontroller developments and related tool chains, the rapid development nature of the PICAXE, simplicity of use and minimal learning curve may mitigate extra cost, and this can be particularly true for users who are undertaking single development projects.

Points of Interest

PEEK and READ

Whereas most internal processing is done using 16-bits, a value read using the PEEK or READ commands is returned as an 8-bit value, and when stored in a word variable, only the least significant bits of that word will be altered. The addresses reachable by peek or limited by picaxe to only the lowest data bank and so users cannot read the entire data area.

RANDOM

The RANDOM command provides pseudo-random number generator functionality which updates a variable to a new value when it is used. The new value is determined by the existing value of the variable when the RANDOM command is used.

Because all variables are initialised to zero when power is applied or a reset occurs, the variables to be used with RANDOM should be seeded first to prevent the same sequence of random numbers being generated whenever the program is started. Seeding can be done by storing and updating a seeding value within non-volatile EEPROM memory.

A mechanism to avoid seeding is to repeatedly execute the RANDOM command (such as when waiting for an input condition) so the value it has generated will be unpredictable when it is used later. For the X1 and X2 parts, an alternative to repeated execution of the RANDOM command is to set the timer running and then use the timer variable to ‘seed’ the random command. This will give much better results.

Although RANDOM can be used with a byte variable, because such a byte value is expanded to 16-bits before the randomising function is applied, the sequence of 'random results' will be very short and produce only a limited set of values.

MIN and MAX

The MIN and MAX operators using in assignments are 'limiting operations', ensuring that the result of the expression evaluated to that point never falls below or exceeds a specified value respectively.

Although somewhat counter-intuitive, MIN can also be thought of as returning the highest of two values, and MAX can be thought of as returning the lowest of the two.

Programming interface

Programs are downloaded from the Programming Editor using a serial link, either a physical serial port or a USB to serial adapter. USB serial ports must be able to support RS232 break signalling for successful use.

The basic programming interface consists of just two resistors and does not need RS232 level converters.

Debugging facilities

- If spare output lines are available, these can be used to control LEDs or Piezo sounders to visually or audibly indicate the program's execution reaching the points where such commands are placed. The Serial Output command (SEROUT) can be used to send execution progress indicators and variable content information to a PC or other terminal device.
- Many of the variants support the SERTXD command which is equivalent to using the SEROUT command to return information about program execution but it uses the Serial Out line of the download interface alleviating the need to use a Digital Output line for this purpose.
- The DEBUG command can be used with the Debug Monitoring facilities of the Programming Editor, the contents of variables when the DEBUG command is executed can be observed.

Interfacing

Being based upon the PICmicro, the PICAXE has great versatility in interfacing. Most variants support the on-chip hardware of the underlying PICmicro.

Digital outputs

Digital Outputs can each sink and source around 20mA and are capable of driving LEDs and other small loads directly. Be aware that while each output can handle 20mA there are limits for each port and for the entire chip that typically prevents 20mA being used on every output. For some PICAXE chips, the port and total limit is around 95mA while for others the limit can be around 200mA. A review of the datasheet for the core PICmicro chip is recommended.

Digital inputs

Most Digital Inputs are protected by diode clamps to the power rails, which as well as offering good ESD protection, allows interfacing to high voltage signals often with little more than a current limiting resistor being required.

This is particularly useful for interfacing to PCs, PDAs and terminals where a complete serial interface can be implemented using just one resistor. This allows for low-cost designs which do not necessitate the use of RS232 level conversion circuits, although such converters can be used to provide a more traditional interface if desired.

The voltage required to make a 1 or 0 on each type of pin is listed below.

TTL (Vsupply > 4.5V)

Vih : >= 2.0V Vil : <= 0.8V

TTL (Vsupply <= 4.5V)

Vih : >= 0.25 * Vsupply + 0.8V Vil : <= 0.15 * Vsupply

Schmitt Trigger (ST)

Vih : >= 0.8 * Vsupply (>= 4V @ 5V) Vil : <= 0.2 * Vsupply (<= 1V @ 5V)

Bi-directional inputs and outputs

The 18, 18A, 18M, 18X, 20M, 28 and 28A have fixed direction Input and Output pins. That is to say that every pin is pre-defined as either an input or an output and this use cannot be explicitly changed under program control.

The 08 and 08M have three bi-directional pins which can be either input or output pins and can be explicitly set as such and changed under program control. The 14M has nine bi-directional pins when used in its advanced configuration.

The 28X/X1 and 40X/X1 also have eight 'PORTC' lines which can be made inputs or outputs under program control. The 28X/X1 and 40X/X1 have another eight "PortB' lines which are always defined as output. Some of the additional pins/legs on the 40X/X1 chips provide a further eight 'PortD' lines which are always set as inputs.

At the start of program execution (after power is applied or reset occurs) all of the bi-directional pins will automatically be set as inputs and only become outputs when instructed by the executing program.

Because it is not always clear what program has previously been downloaded into a PICAXE which supports bi-directional pins, care must be taken when inserting those into alternative hardware. It is recommended that any existing program be erased before it is inserted into hardware different to that which it was previously used with. Inadvertently, or deliberately, making an input line an output line may have damaging effects upon the chip itself and any hardware connected to those pins.

Interrupts

The 08M, 18A, 18M and all X parts support hardware interrupting on a pattern match with the input pins. But this isn't actually the case in practice. According to picaxe documents, the picaxe "polls" the inputs and responds to the change in pin status via firmware. Users should be aware of the difference between programming a PIC microcontroller to respond to changes in the PORTB input register hardware and any claims that Picaxe supports "Hardware Interrupts" using firmware to simulate the same.

Analogue inputs

Analogue input is supported across the entire range. The 08 and 18 support only coarse, low-resolution analogue inputs which is delivered as one of 16 8-bit levels. The others support full 256-level, 8-bit analogue input, and some offer 1024-level, 10-bit inputs.

The number of analogue inputs depends on the actual device, and in some cases the analogue inputs and digital inputs share the same pins, allowing them to be used as either analogue or digital inputs, and with some design, both.

Analogue outputs

Unlike PICs that are capable of producing an analogue output voltage via its internal ladder (requiring external hardware to amplify the signal), there is no ability to generate an analogue output voltage directly, however analogue voltages can be produced by using PWM output capabilities plus the addition of suitable circuitry.

The new 18M2 includes one DAC with a 5-bit resolution giving 32 discrete voltage steps at the DAC output pin.

Dallas/Maxim one-wire device/network connectivity

The firmware supports interfacing with the Dallas/Maxim type 1-Wire devices, including the iButton range of sensors and logger devices together with various 1-Wire chips such as the DS18B20 temperature Sensor, DS243x series EEPROMs, DS2406, DS2408 and DS2413 programmable IO chips with 2, 8 and 2 channels respectively and many more 1-Wire devices.

The firmware for virtually all PICAXE chips provides the ability, through the READOWSN command, to read the unique serial number of any single 1-Wire device connected to a PICAXE input pin, whether or not the particular PICAXE chip firmware includes the additional commands to use that 1-Wire device's capabilities.

The READTEMP and READTEMP12 commands available on virtually all PICAXE chips provides the ability to easily read the temperature from a single DS18B20 temperature sensor connected to a PICAXE input pin.

The OWIN and OWOUT commands which are available only in the firmware of the more recently produced PICAXE chips, such as the X1 and X2 parts, enables communications and use of the capabilities of virtually all available 1-Wire devices including 1-Wire networks comprising many devices connected to a single PICAXE input pin.

I²C

All of the X Parts (18X, 20X2, 28X, 28X1, 28X2, 40X, 40X1 and 40X2) have the capability to use the in-built I²C commands for reading and writing to most I²C devices.

The new 18M2 chip also has i2c communications capability.

Prior to firmware revision 8.6 on the 18X, and 7.7 on the 28X and 40X, the device being accessed must require at least two data bytes (3 bytes total including the address byte) to be able to be used with the built-in commands. Starting with the above mentioned firmware revisions, and the latest editor software, devices that only require two bytes (an address byte and a data byte) can be used.

Some limitations with the firmware controlled I²C functions are that there is no control beyond read and write commands. Protocol options like repeated start, early stop, attention are not supported. By bit-banging the signals, however, the entire gamut of I²C options can be used, but this requires an in depth familiarity with the I²C protocol itself.

The X1 and X2 parts can also be operated as I²C slave devices.

Infrared

For the non X1/X2 parts, there are three commands to support infrared interfacing -

- INFRAIN - Allows reception of infrared signals which use Sony's SIRC protocol. This command is tailored to supporting key press codes which would be sent from a TV remote control and supports only a subset of possible remote control commands.
- INFRAIN2 - Allows reception of infrared signals which use Sony's SIRC protocol. This command allows the reception of any remote control command.
- INFRAOUT - Allows the generation of an infrared signal using Sony's SIRC protocol.

The INFRAIN and INFRAIN2 commands require that an external infrared sensor (TSOP18) is used to demodulate the incoming infrared signal for processing. INFRAOUT can be used to drive an infrared LED (and optional visible light LED) directly.

Even without the INFRAIN and INFRAIN2 commands, infrared signals which use Sony's SIRC protocol can be read by means of bit-banging; sampling the signal provided by the infrared sensor and determining the remote control command sent. This technique is also applicable for decoding infrared signals sent using other protocols, but is not suitable for all such protocols.

For the 28X1 and 40X1 parts, there are two different commands (keywords) with slightly different functionality:

- IRIN - This command is similar to the ‘infrain2’ command found on other PICAXE devices, but can be used on any input pin. There is also an optional timeout feature
- IROUT - This command is similar to the ‘infraout’ command found on other PICAXE devices, but can be used on any output pin.

LCD

All variants can interface to serially controlled LCDs using the SEROUT command providing they use a baud rate which is supported.

All but the 08 and 08M can directly control Hitachi HD44780 and similar LCDs operating in parallel, 4-bit mode. The 08 and 08M have too few output lines to support direct interfacing but can connect to such displays through additional interface circuitry, for example by using shift registers. Such interfacing schemes can always be used instead of direct parallel connections.

PC keyboard

The 18A, 18M, 28A and the X-range all provide the ability to interface to a PC keyboard using the KEYIN and KEYLED commands.

These two commands respectively determine which key was pressed on a PC keyboard through its scan code and control the setting of the keyboard LEDs. The KEYLED command can be used to flash the keyboard LEDs to indicate when a key press has been received.

PWM output

The 08 and 08M support the PWM command and the 08M, 18M and the X-range support the PWMOUT command.

The PWM command provides for a burst of a pulse-width modulated signal to be generated which may be used to charge a simple Resistor-Capacitor circuit to generate an analogue voltage output. PWM can be used with any Digital Output pin.

The PWMOUT command allows a pulse-width modulated signal to be continually generated while execution continues, which may be used to generate analogue voltages, as with PWM, but can also be used for speed control of motors and brightness control of visual indicators. PWMOUT can only be used with certain Digital Output pins. The 08M and 18X provide only a single PWMOUT output, while the 28X and 40X provide two.

For the 14M, the 28X1 and 40X1 parts, there are different PWM commands.

- HPWM - (which stands for hardware PWM) allows selection of the mode, polarity, setting period and duty. Hardware PWM is an advanced method of motor control using PWM methods. HPWM can provide up to 4 separate pins with a PWM output as defined by the PIC microcontroller's internal pwm hardware.

hpwm can be used instead of, not at the same time as, the pwmout command on 2. However pwmout on 1 can be used simultaneously if desired.

HPWMDUTY – On the X1 and X2 parts only, the hpwmduty command can be used to alter the hpwm duty cycle without resetting the internal timer (as occurs with a hpwm command). A hpwm command must be issued before this command will function.

Serial

User programmed serial input and output is implemented in the firmware, allowing the serial communications to be received on any Digital Input pin and transmitted on any Digital Output Pin. The serial interface supports a variety of baud rates between 300 and 4800 baud when operating at 4 MHz, up to 9600 baud at 8 MHz (M, A, and X parts), and up to 19,200 baud at 16 MHz (28X and 40X only).

The X1 and X2 parts also support background serial in, and much faster serial out bauds on the hardware serial pins.

On the 18X, 28X and 40X access to the on-chip AUSART through the use of PEEK and POKE allows bytes to be sent and received at higher baud rates. Those which allow access to the AUSART are capable of transmitting MIDI compatible data and potentially DMX-512 data streams.

Servo control

All except the 08 and 18 can simultaneously support the control of multiple servos connected to any of its Digital Output pins.

Up to eight servos can be controlled directly using the SERVO command. Once the servo positions have been specified, the required signal stream for each servo is sent while program execution continues.

Using SERVO has a severe affect on other time based commands, eg pause/sound. Users should be aware that the PICAXE does not take into consideration the servo pulse length delay when processing delay loops so any time critical functions are bound to be longer than specified in code. A quick test to enable several servos, and then try to compute a valid delay should confirm this.

Sound and tune control

The SOUND command allows tone generation on any of the Digital Outputs. Piezo sounders can be directly connected to the Digital Output pins and loudspeakers can be driven through a simple amplifier circuit.

The 08M, 14M, 18M, 20M, 20X2, 28X1, 28X2, 40X1 and 40X2 have the PLAY command which allows the playing of between one and four pre-programmed tunes depending upon which PICAXE chip is used. For the 08M only, there is capability to optionally flash LEDs in step with the tune's beat.

The next command available for the 08M, 14M, 18M, 20M, 20X2, 28X1, 28X2, 40X1 and 40X2 is the TUNE command. Whereas the SOUND command provides for the frequency of a tone (or white noise) and its duration to be specified, the TUNE command allows the tone to be user specified in terms of a musical note and the tempo of the tune to be specified.

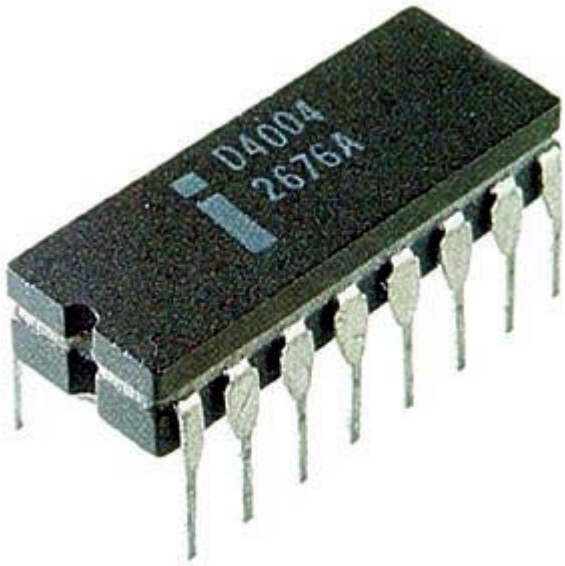
The Programming Editor includes a Wizard to convert suitable mobile phone ringtones to an equivalent TUNE command with the necessary data.

SPI

Native support for SPI and similar interfacing is provided on the X1 and X2 and can be implemented by direct control of the digital output lines, using a technique known as bit-banging.

Chapter-9

Microprocessor



Intel 4004, the first general-purpose, commercial microprocessor

A **microprocessor** incorporates most or all of the functions of a computer's central processing unit (CPU) on a single integrated circuit (IC, or microchip).

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-bit and 8-bit microprocessors, such as terminals, printers, various kinds of automation etc., followed soon after. Affordable 8-bit microprocessors with 16-bit addressing also led to the first general-purpose microcomputers from the mid-1970s on.

During the 1960s, computer processors were often constructed out of small and medium-scale ICs containing from tens to a few hundred transistors. The integration of a whole CPU onto a single chip greatly reduced the cost of processing power. From these humble beginnings, continued increases in microprocessor capacity have rendered other forms of

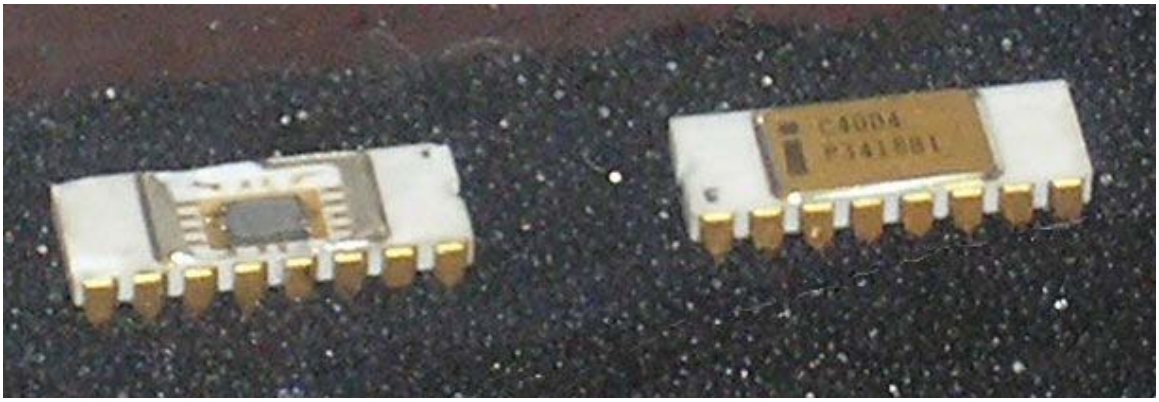
computers almost completely obsolete, with one or more microprocessors used in everything from the smallest embedded systems and handheld devices to the largest mainframes and supercomputers.

Since the early 1970s, the increase in capacity of microprocessors has followed Moore's law, which suggests that the number of transistors that can be fitted onto a chip doubles every two years. Although originally calculated as a doubling every year, Moore later refined the period to two years. It is often incorrectly quoted as a doubling of transistors every 18 months.

Firsts

Three projects delivered a microprocessor at about the same time: Intel's 4004, Texas Instruments (TI) TMS 1000, and Garrett AiResearch's Central Air Data Computer (CADC).

Intel 4004



The 4004 with cover removed (left) and as actually used (right).

The Intel 4004 is generally regarded as the first microprocessor, and cost thousands of dollars. The first known advertisement for the 4004 is dated November 1971 and 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 programmable chip set consisting of seven different chips. Three of the chips were to make a special-purpose CPU with its program stored in ROM and its data stored in shift register read-write memory. Ted Hoff, the Intel engineer assigned to evaluate the project, believed the Busicom design could be simplified by using dynamic RAM storage for data, rather than shift register memory, and a more traditional general-purpose CPU architecture. Hoff came up with a four–chip architectural proposal: a ROM chip for storing the programs, a dynamic RAM chip for storing data, a simple I/O device and a 4-bit central processing unit (CPU). Although not a chip designer, he felt the CPU could be integrated into a single chip. This chip would later be called the 4004 microprocessor.

The architecture and specifications of the 4004 came from the interaction of Hoff with Stanley Mazor, a software engineer reporting to him, and with Busicom engineer Masatoshi Shima, during 1969. In April 1970, Intel hired Federico Faggin to lead the design of the four-chip set. Faggin, who originally developed the silicon gate technology (SGT) in 1968 at Fairchild Semiconductor and designed the world's first commercial integrated circuit using SGT, the Fairchild 3708, had the correct background to lead the project since it was SGT that made it possible to implement a single-chip CPU with the proper speed, power dissipation and cost. Faggin also developed the new methodology for random logic design, based on silicon gate, that made the 4004 possible. Production units of the 4004 were first delivered to Busicom in March 1971 and shipped to other customers in late 1971.

TMS 1000

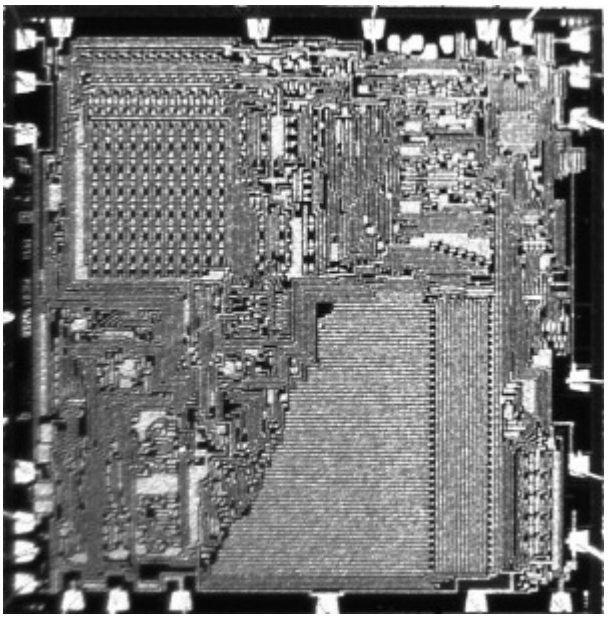
The Smithsonian Institution says TI engineers Gary Boone and Michael Cochran succeeded in creating the first microcontroller (also called a microcomputer) in 1971. The result of their work was the TMS 1000, which went commercial in 1974.

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.

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.

A computer-on-a-chip is a variation of a microprocessor that combines the microprocessor core (CPU), some program memory and read/write memory, and I/O (input/output) lines onto one chip. 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 more akin to a microcontroller.

Pico/General Instrument



The PICO1/GI250 chip introduced in 1971. This was designed by Pico Electronics (Glenrothes, Scotland) and manufactured by General Instrument of Hicksville NY

In 1971 Pico Electronics and General Instrument (GI) introduced their first collaboration in ICs, a complete single chip calculator IC for the Monroe/Litton Royal Digital III calculator. This chip could also arguably lay claim to be one of the first microprocessors or microcontrollers having ROM, RAM and a RISC instruction set on-chip. The layout for the four layers of the PMOS process was hand drawn at x500 scale on mylar film, a significant task at the time given the complexity of the 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. The key team members had originally been tasked by Elliott Automation to create an 8 bit computer in MOS and had helped establish a MOS Research Laboratory in Glenrothes, Scotland in 1967.

Calculators were becoming the largest single market for semiconductors and Pico and GI went on to have significant success in this burgeoning market. GI continued to innovate in microprocessors and microcontrollers with products including the PIC1600, PIC1640 and PIC1650. In 1987 the GI Microelectronics business was spun out into the very successful PIC microcontroller business.

CADC

In 1968, Garrett AiResearch (which employed designers Ray Holt and Steve Geller) was 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 20-bit, pipelined, parallel multi-microprocessor". 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. Ray Holt graduated California Polytechnic University in 1968, and began his computer design career with the CADC. From its inception, it was shrouded in secrecy until 1998 when at Holt's request, the US Navy allowed the documents into the public domain. Since then several have debated if this was the first microprocessor. Holt has stated that no one has compared this microprocessor with those that came later. According to Parab et al. (2007), *"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.

Gilbert Hyatt

Gilbert Hyatt was awarded a patent claiming an invention pre-dating both TI and Intel, describing a "microcontroller". The patent was later invalidated, but not before substantial royalties were paid out.

Four-Phase Systems AL1

The Four-Phase Systems AL1 was an 8-bit bit slice chip containing eight registers and an ALU. It was designed by Lee Boysel in 1969. At the time, it formed part of a nine-chip, 24-bit CPU with three AL1s, but it was later called a microprocessor when, in response to 1990s litigation by Texas Instruments, a demonstration system was constructed where a single AL1 formed part of a courtroom demonstration computer system, together with RAM, ROM, and an input-output device.

8-bit designs

The Intel 4004 was followed in 1972 by the Intel 8008, the world's first 8-bit microprocessor. The 8008 was not, however, an extension of the 4004 design, but instead the culmination of a separate design project at Intel, arising from a contract with Computer Terminals Corporation, of San Antonio TX, for a chip for a terminal they were designing, the Datapoint 2200 — fundamental aspects of the design came not from Intel but from CTC. In 1968, CTC's Austin O. "Gus" Roche developed the original design for the instruction set and operation of the processor. In 1969, CTC contracted two companies, Intel and Texas Instruments, to make a single-chip implementation, known as the CTC 1201. In late 1970 or early 1971, TI dropped out being unable to make a reliable part. In 1970, with Intel yet to deliver the part, CTC opted to use their own

implementation in the Datapoint 3300, using traditional TTL logic instead (thus the first machine to run “8008 code” was not in fact a microprocessor at all!). Intel's version of the 1201 microprocessor arrived in late 1971, but was too late, slow, and required a number of additional support chips. CTC had no interest in using it. CTC had originally contracted Intel for the chip, and would have owed them $50,000 for their design work. To avoid paying for a chip they did not want (and could not use), CTC released Intel from their contract and allowed them free use of the design. Intel marketed it as the 8008 in April, 1972, as 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 was the precursor to the very successful Intel 8080 (1974), which offered much improved performance over the 8008 and required fewer support chips, 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 the integration of extra circuitry (e.g. the Z80's built-in memory refresh circuitry) allowed the home computer "revolution" to accelerate sharply in the early 1980s. This delivered such inexpensive machines as the Sinclair ZX-81, which sold for US$99.

The Western Design Center, Inc. (WDC) introduced the CMOS 65C02 in 1982 and licensed the design to several firms. It was used as the CPU in the Apple IIe and IIc 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 1990s.

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 CISC 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 onboard the Galileo probe to Jupiter (launched 1989, arrived 1995). RCA COSMAC was the first to implement CMOS technology. The CDP1802 was used because it could be run at very low power, and because a variant was available fabricated using a special production process (Silicon on Sapphire), providing much better protection against cosmic radiation and electrostatic discharges than that of any other processor of the era. Thus, the SOS version of the 1802 was 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 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.

12-bit designs

The Intersil 6100 family consisted of a 12-bit microprocessor (the 6100) and a range of peripheral support and memory ICs. The microprocessor recognised the DEC PDP-8 minicomputer instruction set. As such it was sometimes referred to as the **CMOS-PDP8**. Since it was also produced by Harris Corporation, it was also known as the **Harris HM-6100**. By virtue of its CMOS technology and associated benefits, the 6100 was being incorporated into some military designs until the early 1980s.

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.

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.

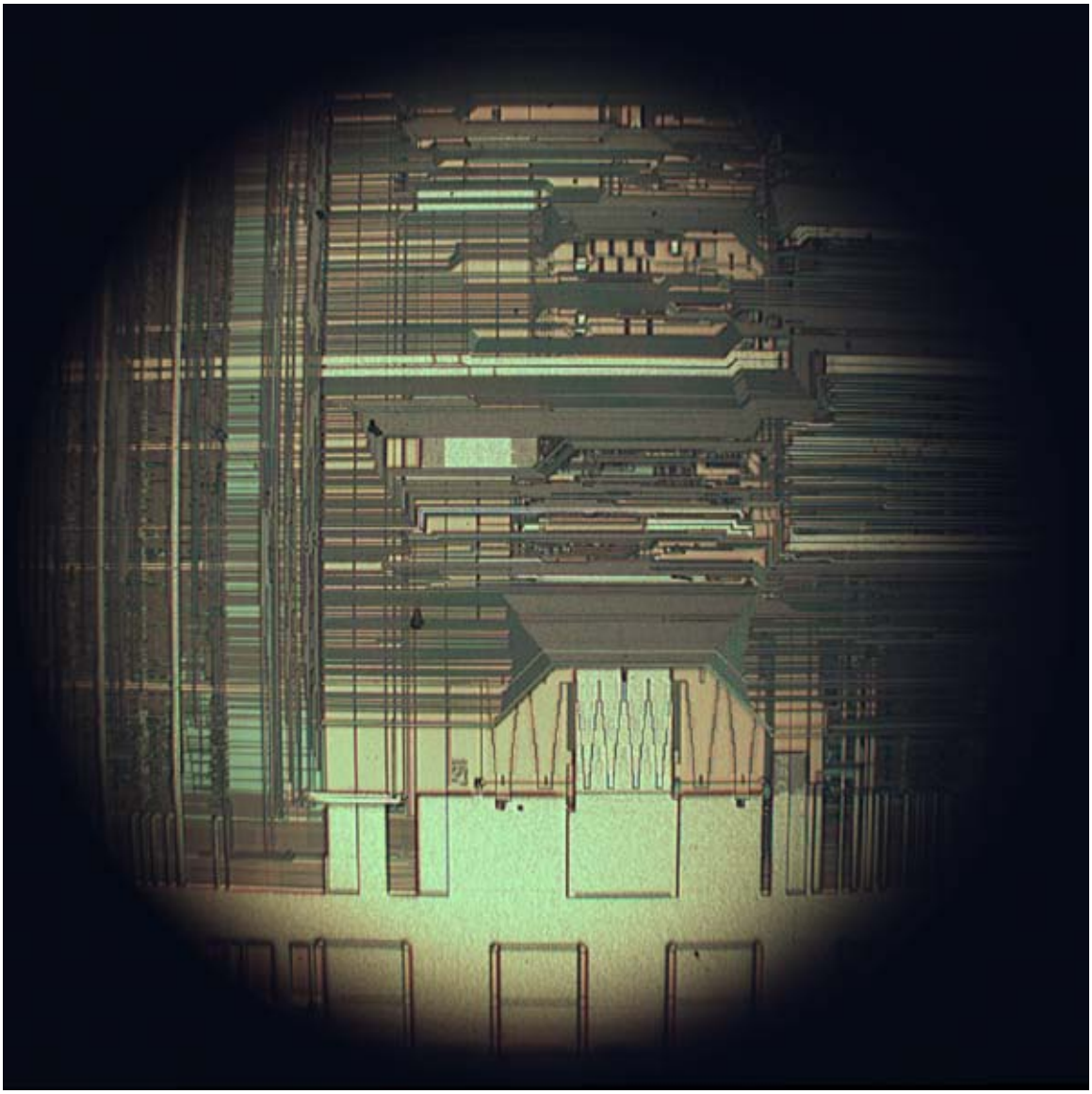
In 1975, National introduced the first 16-bit single-chip microprocessor, the National Semiconductor PACE, which was later followed by an NMOS version, the INS8900.

Another early 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 Center, 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 8086 and 80186 had a crude method of segmentation, while the 80286 introduced a full-featured segmented memory management unit (MMU), and the 80386 introduced a flat 32-bit memory model with paged memory management.

32-bit designs



Upper interconnect layers on an Intel 80486DX2 die.

16-bit designs had only been on the market briefly when 32-bit implementations started to appear.

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.

The ARM first appeared in 1985. This is a RISC processor design, which has since come to dominate the 32-bit embedded systems processor space due in large part to its power efficiency, its licensing model, and its wide selection of system development tools. Semiconductor manufacturers generally license cores such as the ARM11 and integrate them into their own system on a chip products; only a few such vendors are licensed to modify the ARM cores. Most cell phones include an ARM processor, as do a wide variety of other products. There are microcontroller-oriented ARM cores without virtual memory support, as well as SMP applications processors with virtual memory.

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. Later the NS 32132 was introduced which allowed two CPUs to reside on the same memory bus, with built in arbitration. The NS32016/32 outperformed the MC68000/10 but the NS32332 which arrived at approximately the same time the MC68020 did not have enough performance. The third generation chip, the NS32532 was different. It had about double the performance of the MC68030 which was released around the same time. The appearance of RISC processors like the AM29000 and MC88000 (now both dead) influenced the architecture of the final core, the NS32764. Technically advanced, using a superscalar RISC core, internally overclocked, with a 64 bit bus, it was still capable of executing Series 32000 instructions through real time translation.

When National Semiconductor decided to leave the Unix market, the chip was redesigned into the Swordfish Embedded processor with a set of on chip peripherals. The chip turned out to be too expensive for the laser printer market and was killed. The design team went to Intel and there designed the Pentium processor which is very similar to the NS32764 core internally The big success of the Series 32000 was in the laser printer market, where the NS32CG16 with microcoded BitBlt instructions had very good price/performance and was adopted by large companies like Canon. 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 Z80000, 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.

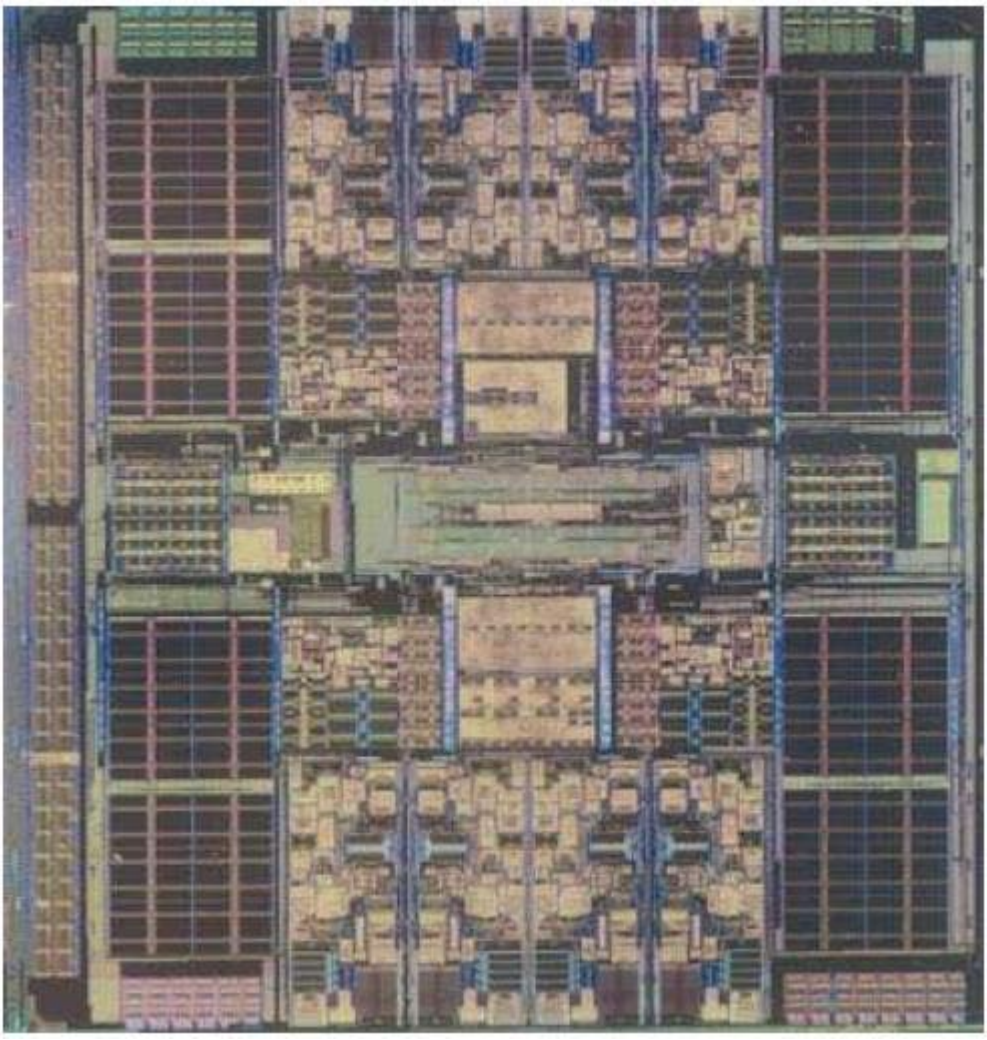
64-bit designs in personal computers

While 64-bit microprocessor designs have been in use in several markets since the early 1990s, the early 2000s 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 (also 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, Windows 7 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.

Multicore designs



Niagara 8 Core Processor

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 and LGA1156 socket and the Intel i7 and i5 chips, quad core is now considered mainstream, but with the release of the i7-980x, six core processors are now well within reach.

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 RISC 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.

Special-purpose designs

A microprocessor is a general purpose system. Several specialized processing devices have followed from the technology. Microcontrollers integrate a microprocessor with periphal devices for control of embedded system. A digital signal processor (DSP) is specialized for signal processing. Graphics processing units may have no, limited, or general programming facilities. For example, GPUs through the 1990s were mostly non-programmable and have only recently gained limited facilities like programmable vertex shaders.

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 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.

Chapter-10

Multi-core Processor

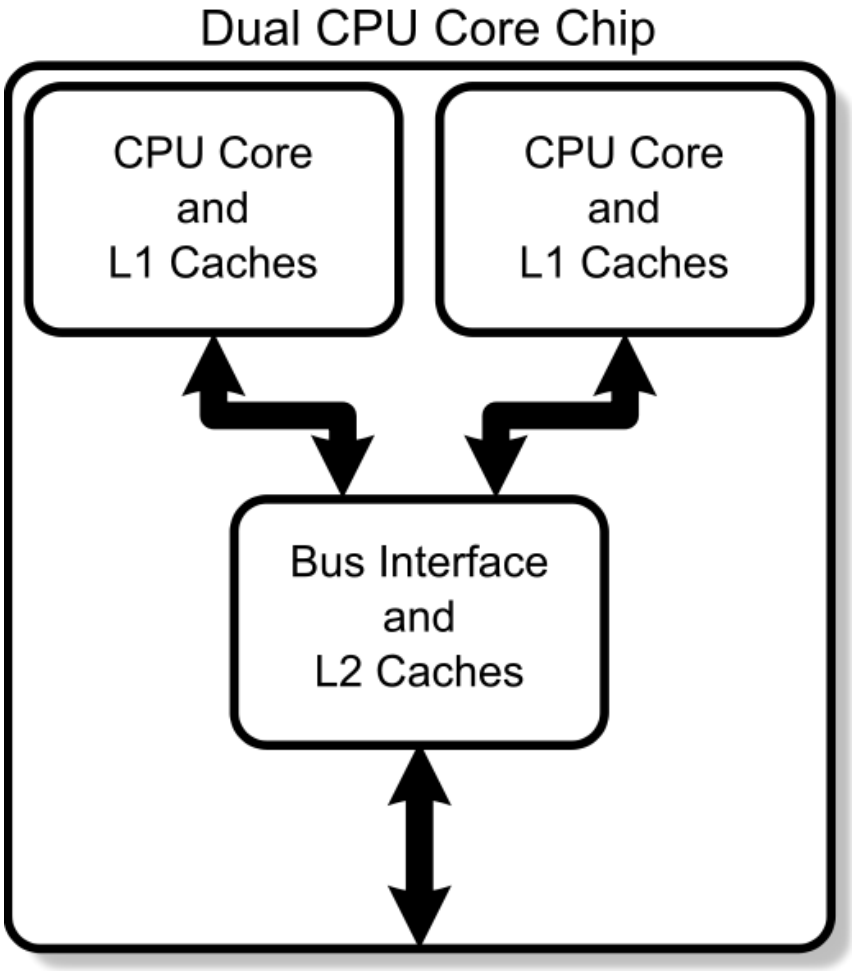
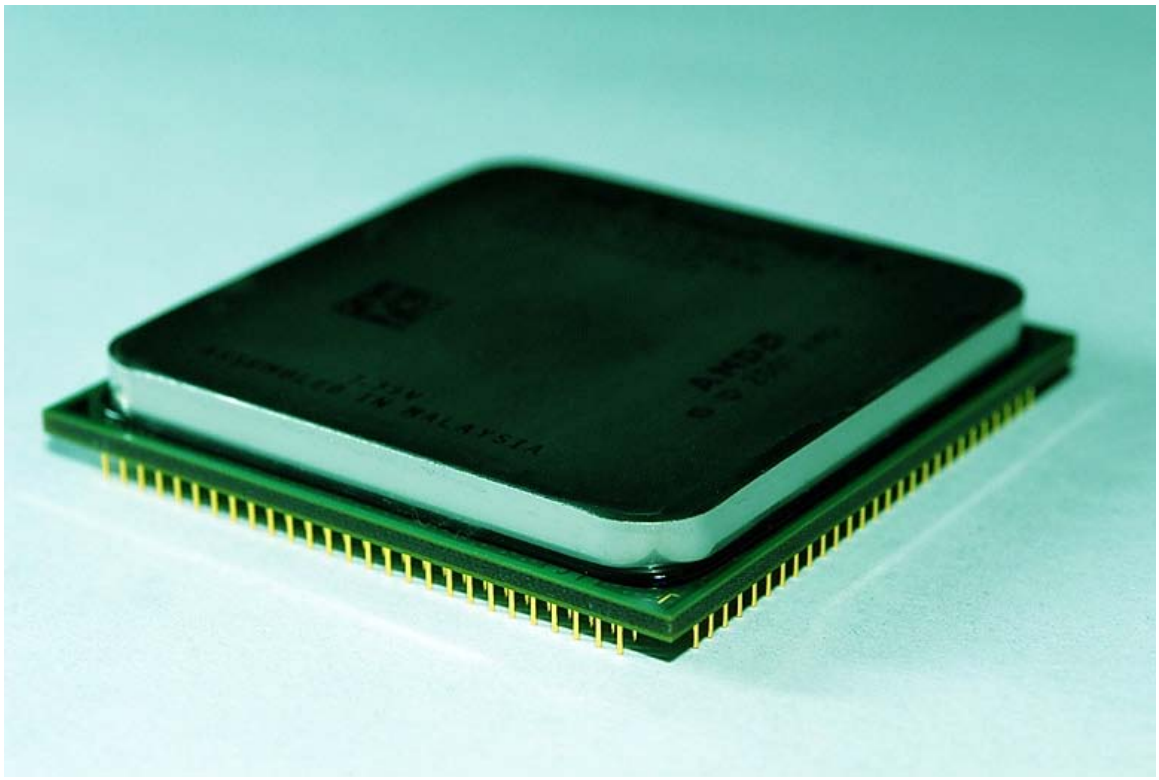


Diagram of a generic dual-core processor, with CPU-local level 1 caches, and a shared, on-die level 2 cache.

An Intel Core 2 Duo E6750 dual-core processor.



An AMD Athlon X2 6400+ dual-core processor.

In computing, a **processor** is the unit that reads and executes program instructions, which are fixed-length (typically 32 or 64 bit) or variable-length chunks of data. The data in the instruction tells the processor what to do. The instructions are very basic things like reading data from memory or sending data to the user display, but they are processed so rapidly that we experience the results as the smooth operation of a program.

Processors were originally developed with only one **core**. The core is the part of the processor that actually performs the reading and executing of instructions. A **multi-core processor** is a single component with two or more independent actual processors (called "cores"). Manufacturers typically integrate the cores onto a single integrated circuit die (known as a chip multiprocessor or CMP), or onto multiple dies in a single chip package. A **many-core** processor is one in which the number of cores is large enough that traditional multi-processor techniques are no longer efficient — largely due to issues with congestion in supplying instructions and data to the many processors. The many-core threshold is roughly in the range of several tens of cores; above this threshold network on chip technology is advantageous.

A **dual-core processor** has two cores (e.g. AMD Phenom II X2, Intel Core Duo), a **quad-core processor** contains four cores (e.g. AMD Phenom II X4, the Intel 2010 core line that includes 3 levels of quad core processors), and a **hexa-core processor** contains six cores (e.g. AMD Phenom II X6, Intel Core i7 Extreme Edition 980X). A multi-core processor implements multiprocessing in a single physical package. Designers may couple cores in a multi-core device tightly or loosely. For example, cores may or may not share caches, and they may implement message passing or shared memory inter-core communication methods. Common network topologies to interconnect cores include bus, ring, 2-dimensional mesh, and crossbar. *Homogeneous* multi-core systems include only identical cores, heterogeneous multi-core systems have cores which are not identical. Just as with single-processor systems, cores in multi-core systems may implement architectures such as superscalar, VLIW, vector processing, SIMD, or multithreading.

Multi-core processors are widely used across many application domains including general-purpose, embedded, network, digital signal processing (DSP), and graphics.

The improvement in performance gained by the use of a multi-core processor depends very much on the software algorithms used and their implementation. In particular, possible gains are limited by the fraction of the software that can be parallelized to run on multiple cores simultaneously; this effect is described by Amdahl's law. In the best case, so-called embarrassingly parallel problems may realize speedup factors near the number of cores, or even more if the problem is split up enough to fit within each core's cache(s), avoiding use of much slower main system memory. Most applications, however, are not accelerated so much. The parallelization of software is a significant ongoing topic of research.

Terminology

The terms *multi-core* and *dual-core* most commonly refer to some sort of central processing unit (CPU), but are sometimes also applied to digital signal processors (DSP) and system-on-a-chip (SoC). Additionally, some use these terms to refer only to multi-core microprocessors that are manufactured on the *same* integrated circuit die. These people generally refer to separate microprocessor dies in the same package by another name, such as *multi-chip module*. Here we, uses the terms "multi-core" and "dual-core" for CPUs manufactured on the *same* integrated circuit, unless otherwise noted.

In contrast to multi-core systems, the term *multi-CPU* refers to multiple physically separate processing-units (which often contain special circuitry to facilitate communication between each other).

The terms *many-core* and *massively multi-core* are sometimes used to describe multi-core architectures with an especially high number of cores (tens or hundreds).

Some systems use many soft microprocessor cores placed on a single FPGA. Each "core" can be considered a "semiconductor intellectual property core" as well as a CPU core.

Development

While manufacturing technology improves, reducing the size of individual gates, physical limits of semiconductor-based microelectronics have become a major design concern. These physical limitations can cause significant heat dissipation and data synchronization problems. Various methods are used to improve CPU performance. Some *instruction-level parallelism* (ILP) methods such as superscalar pipelining are suitable for many applications, but are inefficient for others that contain difficult-to-predict code. Many applications are better suited to *thread level parallelism* (TLP) methods, and multiple independent CPUs are commonly used to increase a system's overall TLP. A combination of increased available space (due to refined manufacturing processes) and the demand for increased TLP led to the development of multi-core CPUs.

Commercial incentives

Several business motives drive the development of dual-core architectures. For decades, it was possible to improve performance of a CPU by shrinking the area of the integrated circuit, which drove down the cost per device on the IC. Alternatively, for the same circuit area, more transistors could be utilized in the design, which increased functionality, especially for CISC architectures.

Eventually these techniques reached their limit and were unable further to improve CPU performance. Multiple processors had to be employed to gain speed in computation. Multiple cores were used on the same chip to improve performance, which could then lead to better sales of CPU chips which had two or more cores. Intel has produced a 48-core processor for research in cloud computing.

Technical factors

Since computer manufacturers have long implemented symmetric multiprocessing (SMP) designs using discrete CPUs, the issues regarding implementing multi-core processor architecture and supporting it in software are well known.

Additionally:

- Utilizing a proven processing-core design without architectural changes reduces design risk significantly.
- For general-purpose processors, much of the motivation for multi-core processors comes from greatly diminished gains in processor performance from increasing the operating frequency. This is due to three primary factors:
    1. The *memory wall*; the increasing gap between processor and memory speeds. This effect pushes cache sizes larger in order to mask the latency of memory. This helps only to the extent that memory bandwidth is not the bottleneck in performance.
    2. The *ILP wall*; the increasing difficulty of finding enough parallelism in a single instructions stream to keep a high-performance single-core processor busy.
    3. The *power wall*; the trend of consuming exponentially increasing power with each factorial increase of operating frequency. This increase can be mitigated by "shrinking" the processor by using smaller traces for the same logic. The *power wall* poses manufacturing, system design and deployment problems that have not been justified in the face of the diminished gains in performance due to the *memory wall* and *ILP wall*.

In order to continue delivering regular performance improvements for general-purpose processors, manufacturers such as Intel and AMD have turned to multi-core designs, sacrificing lower manufacturing-costs for higher performance in some applications and systems. Multi-core architectures are being developed, but so are the alternatives. An especially strong contender for established markets is the further integration of peripheral functions into the chip.

Advantages

The proximity of multiple CPU cores on the same die allows the cache coherency circuitry to operate at a much higher clock-rate than is possible if the signals have to travel off-chip. Combining equivalent CPUs on a single die significantly improves the performance of cache snoop (alternative: Bus snooping) operations. Put simply, this means that signals between different CPUs travel shorter distances, and therefore those signals degrade less. These higher-quality signals allow more data to be sent in a given time period, since individual signals can be shorter and do not need to be repeated as often.

The largest boost in performance will likely be noticed in improved response-time while running CPU-intensive processes, like antivirus scans, ripping/burning media (requiring file conversion), or file searching. For example, if the automatic virus-scan runs while a movie is being watched, the application running the movie is far less likely to be starved of processor power, as the antivirus program will be assigned to a different processor core than the one running the movie playback.

Assuming that the die can fit into the package, physically, the multi-core CPU designs require much less printed circuit board (PCB) space than do multi-chip SMP designs. Also, a dual-core processor uses slightly less power than two coupled single-core processors, principally because of the decreased power required to drive signals external to the chip. Furthermore, the cores share some circuitry, like the L2 cache and the interface to the front side bus (FSB). In terms of competing technologies for the available silicon die area, multi-core design can make use of proven CPU core library designs and produce a product with lower risk of design error than devising a new wider core-design. Also, adding more cache suffers from diminishing returns.

Disadvantages

Maximizing the utilization of the computing resources provided by multi-core processors requires adjustments both to the operating system (OS) support and to existing application software. Also, the ability of multi-core processors to increase application performance depends on the use of multiple threads within applications. The situation is improving: for example the Valve Corporation's Source engine offers multi-core support, and Crytek has developed similar technologies for CryEngine 2, which powers their game, *Crysis*. Emergent Game Technologies' Gamebryo engine includes their Floodgate technology which simplifies multicore development across game platforms. In addition, Apple Inc.'s latest OS, Mac OS X Snow Leopard has a built-in multi-core facility called Grand Central Dispatch for Intel CPUs.

Integration of a multi-core chip drives chip production yields down and they are more difficult to manage thermally than lower-density single-chip designs. Intel has partially countered this first problem by creating its quad-core designs by combining two dual-core on a single die with a unified cache, hence any two working dual-core dies can be used, as opposed to producing four cores on a single die and requiring all four to work to produce a quad-core. From an architectural point of view, ultimately, single CPU designs may make better use of the silicon surface area than multiprocessing cores, so a development commitment to this architecture may carry the risk of obsolescence. Finally, raw processing power is not the only constraint on system performance. Two processing cores sharing the same system bus and memory bandwidth limits the real-world performance advantage. If a single core is close to being memory-bandwidth limited, going to dual-core might only give 30% to 70% improvement. If memory bandwidth is not a problem, a 90% improvement can be expected. It would be possible for an application that used two CPUs to end up running faster on one dual-core if communication between the CPUs was the limiting factor, which would count as more than 100% improvement.

Hardware

Trends

The general trend in processor development has moved from dual-, tri-, quad-, hexa-, octo-core chips to ones with tens or even hundreds of cores. In addition, multi-core chips mixed with simultaneous multithreading, memory-on-chip, and special-purpose "heterogeneous" cores promise further performance and efficiency gains, especially in processing multimedia, recognition and networking applications. There is also a trend of improving energy-efficiency by focusing on performance-per-watt with advanced fine-grain or ultra fine-grain power management and dynamic voltage and frequency scaling (i.e. laptop computers and portable media players).

Architecture

The composition and balance of the cores in multi-core architecture show great variety. Some architectures use one core design repeated consistently ("homogeneous"), while others use a mixture of different cores, each optimized for a different, "heterogeneous", role .

The article *CPU designers debate multi-core future* by Rick Merritt, EE Times 2008, includes comments:
"Chuck Moore [...] suggested computers should be more like cellphones, using a variety of specialty cores to run modular software scheduled by a high-level applications programming interface.
[...] Atsushi Hasegawa, a senior chief engineer at Renesas, generally agreed. He suggested the cellphone's use of many specialty cores working in concert is a good model for future multi-core designs.
[...] Anant Agarwal, founder and chief executive of startup Tilera, took the opposing view. He said multi-core chips need to be homogeneous collections of general-purpose cores to keep the software model simple."

Software impact

An outdated version of an anti-virus application may create a new thread for a scan process, while its GUI thread waits for commands from the user (e.g. cancel the scan). In such cases, a multicore architecture is of little benefit for the application itself due to the single thread doing all heavy lifting and the inability to balance the work evenly across multiple cores. Programming truly multithreaded code often requires complex co-ordination of threads and can easily introduce subtle and difficult-to-find bugs due to the interleaving of processing on data shared between threads (thread-safety). Consequently, such code is much more difficult to debug than single-threaded code when it breaks. There has been a perceived lack of motivation for writing consumer-level threaded applications because of the relative rarity of consumer-level multiprocessor hardware. Although threaded applications incur little additional performance penalty on single-processor machines, the extra overhead of development has been difficult to justify due to

the preponderance of single-processor machines. Also, serial tasks like decoding the entropy encoding algorithms used in video codecs are impossible to parallelize because each result generated is used to help create the next result of the entropy decoding algorithm.

Given the increasing emphasis on multicore chip design, stemming from the grave thermal and power consumption problems posed by any further significant increase in processor clock speeds, the extent to which software can be multithreaded to take advantage of these new chips is likely to be the single greatest constraint on computer performance in the future. If developers are unable to design software to fully exploit the resources provided by multiple cores, then they will ultimately reach an insurmountable performance ceiling.

The telecommunications market had been one of the first that needed a new design of parallel datapath packet processing because there was a very quick adoption of these multiple-core processors for the datapath and the control plane. These MPUs are going to replace the traditional Network Processors that were based on proprietary micro- or pico-code.

Parallel programming techniques can benefit from multiple cores directly. Some existing parallel programming models such as Cilk++, OpenMP, FastFlow, Skandium, and MPI can be used on multi-core platforms. Intel introduced a new abstraction for C++ parallelism called TBB. Other research efforts include the Codeplay Sieve System, Cray's Chapel, Sun's Fortress, and IBM's X10.

Multi-core processing has also affected the ability of modern computational software development. Developers programming in newer languages might find that their modern languages do not support multi-core functionality. This then requires the use of numerical libraries to access code written in languages like C and Fortran, which perform math computations faster than newer languages like C#. Intel's MKL and AMD's ACML are written in these native languages and take advantage of multi-core processing.

Managing concurrency acquires a central role in developing parallel applications. The basic steps in designing parallel applications are:

Partitioning
: The partitioning stage of a design is intended to expose opportunities for parallel execution. Hence, the focus is on defining a large number of small tasks in order to yield what is termed a fine-grained decomposition of a problem.

Communication
: The tasks generated by a partition are intended to execute concurrently but cannot, in general, execute independently. The computation to be performed in one task will typically require data associated with another task. Data must then be transferred between tasks so as to allow computation to proceed. This information flow is specified in the communication phase of a design.

Agglomeration

In the third stage, development moves from the abstract toward the concrete. Developers revisit decisions made in the partitioning and communication phases with a view to obtaining an algorithm that will execute efficiently on some class of parallel computer. In particular, developers consider whether it is useful to combine, or agglomerate, tasks identified by the partitioning phase, so as to provide a smaller number of tasks, each of greater size. They also determine whether it is worthwhile to replicate data and/or computation.

Mapping

In the fourth and final stage of the design of parallel algorithms, the developers specify where each task is to execute. This mapping problem does not arise on uniprocessors or on shared-memory computers that provide automatic task scheduling.

On the other hand, on the server side, multicore processors are ideal because they allow many users to connect to a site simultaneously and have independent threads of execution. This allows for Web servers and application servers that have much better throughput.

Licensing

Typically, proprietary enterprise-server software is licensed "per processor". In the past a CPU was a processor and most computers had only one CPU, so there was no ambiguity.

Now there is the possibility of counting cores as processors and charging a customer for multiple licenses for a multi-core CPU. However, the trend seems to be counting dual-core chips as a single processor: Microsoft, Intel, and AMD support this view. Microsoft have said they would treat a socket as a single processor.

Oracle counts an AMD X2 or Intel dual-core CPU as a single processor but has other numbers for other types, especially for processors with more than two cores. IBM and HP count a multi-chip module as multiple processors. If multi-chip modules count as one processor, CPU makers have an incentive to make large expensive multi-chip modules so their customers save on software licensing. It seems that the industry is slowly heading towards counting each die as a processor, no matter how many cores each die has.

Embedded applications

Embedded computing operates in an area of processor technology distinct from that of "mainstream" PCs. The same technological drivers towards multicore apply here too. Indeed, in many cases the application is a "natural" fit for multicore technologies, if the task can easily be partitioned between the different processors.

In addition, embedded software is typically developed for a specific hardware release, making issues of software portability, legacy code or supporting independent developers less critical than is the case for PC or enterprise computing. As a result, it is easier for

developers to adopt new technologies and as a result there is a greater variety of multicore processing architectures and suppliers.

As of 2010, multi-core network processing devices have become mainstream, with companies such as Freescale Semiconductor, Cavium Networks, Wintegra and Broadcom all manufacturing products with eight processors.

In digital signal processing the same trend applies: Texas Instruments has the three-core TMS320C6488 and four-core TMS320C5441, Freescale the four-core MSC8144 and six-core MSC8156 (and both have stated they are working on eight-core successors). Newer entries include the Storm-1 family from Stream Processors, Inc with 40 and 80 general purpose ALUs per chip, all programmable in C as a SIMD engine and Picochip with three-hundred processors on a single die, focused on communication applications.

Chapter-11

Microprocessor Development Board and Memory Dependence Prediction

Microprocessor development board

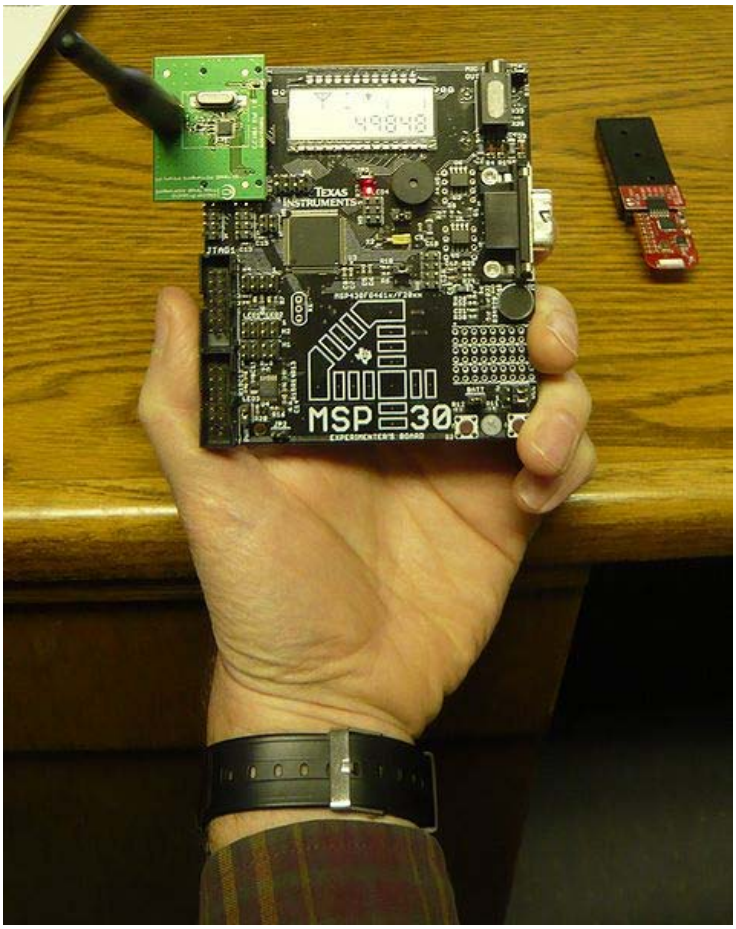


Photo of two experimenter boards for the MSP430 chipset by Texas Instruments. On the left the larger chip version, on the right a small version in USB format.

A **Microprocessor Development Board** is a printed circuit board containing a microprocessor and the minimal support logic needed for an engineer to become

acquainted with the microprocessor on the board, and to learn to do some elementary assembler programming on it. It also served for the producer of the microprocessor as a platform for testing their new chip.

It differs from a home computer by not having any logic above what is absolutely necessary to create a working system with an ability to enter and execute a machine language program, and evaluate the result. So normally all the things you would expect to have in a computer system designed for entertainment, such as a Video Display Controller, a sound-chip, and a keyboard usable for Basic, would not be available as a standard feature.

History

The reason for the existence of a development board was only to provide a system for learning to use a new microprocessor, not for entertainment. So everything superfluous was left out to keep costs down. Even an enclosure was not supplied, nor a power supply. This is because the board would only be used in a "laboratory" environment so it did not need an enclosure, and the board could be powered by a typical bench power supply already available to an electronic engineer.

Microprocessor training development kits were not always produced by microprocessor manufacturers. Many systems that can be classified as microprocessor development kits were produced by third parties, one example is the Sinclair MK14, which was inspired by the official SC/MP development board from National Semiconductor, the "NS introkit".

Although these development boards were not designed for hobbyists, they were often bought by them because they were the earliest cheap microcomputer devices you could buy. They often added all kinds of expansions, such as more memory, a video interface etc. It was very popular to use (or write) an implementation of Tiny Basic. The most popular microprocessor board, the KIM-1, received the most attention from the hobby community, because it was much cheaper than most other development boards, and you could get more software for it (Tiny Basic, games, assemblers), and cheap expansion cards to add more memory or other functionality. Also much more articles were published in magazines like "Kilobaud Microcomputing" that described home-brew software and hardware for the KIM-1 you could copy than for other development boards.

Today some chip producers still release "test boards" to demonstrate their chips, and to use them as a "reference design". Their significance these days is much smaller than it was in the days that such boards, (the KIM-1 being the canonical example) were the only low cost way to get "hands-on" acquainted with microprocessors.

Features

The most important feature of the microprocessor development board was the ROM based built-in machine language monitor, or "debugger" as it was also sometimes called. Often the name of the board was related to the name of this monitor program, for

example the name of the monitor program of the KIM-1 was "Keyboard Input Monitor", because the ROM based software allowed entry of programs without the rows of cumbersome toggle switches that older systems used. The popular 6800 based systems often used a monitor with a name with the word "bug" for "debugger" in it, for example the popular "MIKBUG".

Input was normally done with a hexadecimal keyboard, using a machine language monitor program, and the display only consisted of a 7-segment display. Backup storage of written assembler programs was primitive: only a cassette type interface was typically provided, or the serial telex interface was used to read (or punch) a papertape.

Often the board has some kind to expansion connector that brought out all the necessary CPU signals, so that an engineer could build and test an experimental interface or other electronic device.

External interfaces on the bare board were often limited to an RS232 serial port, so a terminal, printer, or teletypewriter could be connected.

List of historical development boards

- 8085AAT a 8085 microprocessor training unit from Paccom
- CDP18S020 evaluation board for the RCA CDP1802 microprocessor
- EVK 300 6800 single board from American Microsystems (AMI)
- Explorer/85 expandable learning system based on the 8085, by Netronics's research and development ltd.
- ITT experimenter used switches and LEDs, and an intel 8080
- KIM-1 the development board for the MOS Technology/Rockwell/Synertek 6502 microprocessor. The name KIM is short for "keyboard input monitor"
    - SYM-1 a slightly improved KIM-1 with better software, more memory, and I/O. Also known as the VIM
    - AIM-65 an improved KIM-1 with an alphanumerical LED display, and a built-in printer.
    - The KIM-1 also lead to some unofficial copies, such as the super-KIM and the Junior from the magazine Elektor, and the MCS Alpha 1
- LC 80 by Robotron
- MAXBOARD development board for the Motorola 6802.
- MEK6800D2 the official development board for the Motorola 6800 microprocessor. The name of the monitor software was MIKBUG
- MicroChroma 68 color graphics kit. Developed by Motorola to demonstrate their new 6847 video display processor. The monitor software was called TVBUG
- Motorola EXORcisor development system (rack based) for the Motorola 6809
- Microprofessor I (MPF-1) Z80 development and training system by Acer
- MST-80B 8080 training system by the Lawrence Livermore National Laboratory
- NS introkit by National Semiconductor featuring the SC/MP, the predecessor to the Sinclair MK14

- NRI microcomputer, a system developed to teach computer courses by McGraw-Hill and the National Radio InstituteI (NRI)
- MK14 Trainings system for the SC/MP microprocessor from Sinclair Research Ltd.
- SDK-80 Intels development board for their 8080 microprocessor
- SDK-51 Intels development board for their Intel MCS-51
- SDK-85 Intels development board for their 8085 microprocessor
- SDK-86 Intels development board for their 8086 microprocessor
- Siemens Microset-8080 boxed system based on a 8080.
- Signetics Instructor 50 based on the Signetics 2650.
- RCA Cosmac Super Elf by RCA . a 1802 learning system with an RCA 1861 Video Display Controller.
- TK-80 the development board for NEC's clone of Intel's i8080, the μPD 8080A
- TM 990/100M evaluation board for the Texas Instruments TMS9900
- TM 990/180M evaluation board for the Texas Instruments TMS9800
- XPO-1 Texas Instruments development system for the PPS-4/1 line of microcontrollers

Memory dependence prediction

Memory dependence prediction is a technique, employed by high-performance out-of-order execution microprocessors that execute memory access operations (loads and stores) out of program order, to predict true dependences between loads and stores at instruction execution time. With the predicted dependence information, the processor can then decide to speculatively execute certain loads and stores out of order, while preventing other loads and stores from executing out-of-order (keeping them in-order). Later in the pipeline, memory disambiguation techniques are used to determine if the loads and stores were correctly executed and, if not, to recover.

By using the memory dependence predictor to keep most dependent loads and stores in order, the processor gains the benefits of aggressive out-of-order load/store execution but avoids many of the memory dependence violations that occur when loads and stores were incorrectly executed. This increases performance because it reduces the number of pipeline flushes that are required to recover from these memory dependence violations.

In general memory dependence prediction predicts whether two memory operations are dependent, that is, if they interact by accessing the same memory location. Besides using store to load (RAW or true) memory dependence prediction for the out-of-order scheduling of loads and stores, other applications of memory dependence prediction have been proposed.

Memory dependence prediction is an optimization on top of **memory dependence speculation**. Sequential execution semantics imply that stores and loads appear to execute in the order specified by the program. However, as with out-of-order execution

of other instructions, it may be possible to execute two memory operations in a different than the program implied order. This is possible when the two operations are independent. In memory dependence speculation a load may be allowed to execute before a store that precedes it. Speculation succeeds when the load is independent of the store, that is, when the two instructions access different memory locations. Speculation fails when the load is dependent upon the store, that is, when the two accesses overlap in memory. In the first, modern out-of-order designs, memory speculation was not used as its benefits were limited. As the scope of the our-of-order execution increased over few tens of instructions, naive memory depedence speculation was used. In **naive memory dependence speculation**, a load is allowed to bypass any preceding store. As with any form of speculation, it is important to weight the benefits of correct speculation against the penalty paid on incorrect speculation. As the scope of out-of-order execution increases further into several tens of instructions, the performance benefits of naive speculation decrease. To retain the benefits of aggressive memory dependence speculation while avoiding the costs of mispeculation several predictors have been proposed.

Selective memory dependence prediction stalls specific loads until it is certain that no violation may occur. It does not explicitly predict dependences. This predictor may delay loads longer than necessary and hence result in suboptimal performance. In fact, in some cases it performs worse than naively speculating all loads as early as possible. This is because often its faster to mispeculate and recover than to wait for all preceding stores to execute. Exact memory dependence prediction was developed at the University of Wisconsin–Madison. Specifically, **Dynamic Speculation and Synchronization** delays loads only as long as it is necessary by predicting the exact store a load should wait for. This predictor predicts exact dependences (store and load pair). The **synonym predictor** groups together all dependences that share a common load or store instruction. The **store sets** predictor represents multiple potential dependences efficiently by grouping together all possible stores a load may dependent upon. The **store barrier** predictor treats certain store instructions as barriers. That is, all subsequent load or store operations are not allowed to bypass the specific store. The store barrier predictor does not explicitly predict dependences. This predictor may unnecessarily delay subsequent, yet independent loads. Memory dependence prediction has other applications beyond the scheduling of loads and stores. For example, **speculative memory cloaking** and **speculative memory bypassing** use memory dependence prediction to streamline the communication of values through memory.

Analogy to branch prediction

Memory dependence prediction for loads and stores is analogous to branch prediction for conditional branch instructions. In branch prediction, the branch predictor predicts which way the branch will resolve before it is known. The processor can then speculatively fetch and execute instructions down one of the paths of the branch. Later, when the branch instruction executes, it can be determined if the branch instruction was correctly predicted. If not, this is a branch misprediction, and a pipeline flush is necessary to throw away instructions that were speculatively fetched and executed.

Branch prediction can be thought of as a two step process. First, the predictor determines the direction of the branch (taken or not). This is a binary decision. Then, the predictor determines the actual target address. Similarly, memory dependence prediction can be thought of as a two step process. First, the predictor determines whether there is a dependence. Then it determines which this dependence is.

Concept and Mechanisms

Memory Dependence Prediction using Store Sets, Chrysos and Emer, in the Proceedings of the 25th Annual ACM/IEEE Conference on Computer Architecture, June 1998.

Aparatus to dynamically control the out-of-order execution of load-store instructions in a processor capable of dispatching, issuing and executing multiple instructions in a single processor cycle, Hesson, LeBlanc and Ciavaglia, IBM, US Patent 5,615,350, March 1997.

Dynamic Speculation and Synchronization of Memory Dependences, Moshovos, Breach, Vijaykumar and Sohi, in the Proceedings of the 24th Annual ACM/IEEE Conference on Computer Architecture, June 1997. Also as technical report, Computer Sciences Department, University of Wisconsin–Madison, March 1996.

Streamlining Inter-Operation Memory Communication via Memory Dependence Prediction, Moshovos and Sohi, in the Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture, Dec. 1997.

Memory Dependence Prediction, Moshovos, Ph.D. Thesis, Computer Sciences Department, University of Wisconsin–Madison, Dec. 1998.

Chapter-12

Reduced Instruction Set Computing

Reduced instruction set computing, or **RISC** is a CPU design strategy based on the insight that simplified (as opposed to complex) instructions can provide higher performance if this simplicity enables much faster execution of each instruction. A computer based on this strategy is a **reduced instruction set computer** (also **RISC**). There are many proposals for precise definitions, but the term is slowly being replaced by the more descriptive **load-store architecture**. Well known RISC families include DEC Alpha, AMD 29k, ARC, ARM, Atmel AVR, MIPS, PA-RISC, Power (including PowerPC), SuperH, and SPARC.

Some aspects attributed to the first RISC-*labeled* designs around 1975 include the observations that the memory-restricted compilers of the time were often unable to take advantage of features intended to facilitate *manual* assembly coding, and that complex addressing modes take many cycles to perform due to the required additional memory accesses. It was argued that such functions would be better performed by sequences of simpler instructions if this could yield implementations small enough to leave room for many registers, reducing the number of slow memory accesses. In these simple designs, most instructions are of uniform length and similar structure, arithmetic operations are restricted to CPU registers and only separate *load* and *store* instructions access memory. These properties enable a better balancing of pipeline stages than before, making RISC pipelines significantly more efficient and allowing higher clock frequencies.

Non-RISC design philosophy

In the early days of the computer industry, programming was done in assembly language or machine code, which encouraged powerful and easy-to-use instructions. CPU designers therefore tried to make instructions that would do as much work as feasible. With the advent of higher level languages, computer architects also started to create dedicated instructions to directly implement certain central mechanisms of such languages. Another general goal was to provide every possible addressing mode for every instruction, known as orthogonality, to ease compiler implementation. Arithmetic operations could therefore often have results as well as operands directly in memory (in addition to register or immediate).

The attitude at the time was that hardware design was more mature than compiler design so this was in itself also a reason to implement parts of the functionality in hardware or microcode rather than in a memory constrained compiler (or its generated code) alone. This design philosophy became retroactively termed complex instruction set computing (CISC) after the RISC philosophy came onto the scene.

CPUs also had relatively few registers, for several reasons:

- More registers also implies more time-consuming saving and restoring of register contents on the machine stack.
- A large number of registers requires a large number of instruction bits as register specifiers, meaning less dense code.
- CPU registers are more expensive than external memory locations; large register sets were cumbersome with limited circuit boards or chip integration.

An important force encouraging complexity was very limited main memories (on the order of kilobytes). It was therefore advantageous for the density of information held in computer programs to be high, leading to features such as highly encoded, variable length instructions, doing data loading as well as calculation (as mentioned above). These issues were of higher priority than the ease of decoding such instructions.

An equally important reason was that main memories were quite slow (a common type was ferrite core memory); by using dense information packing, one could reduce the frequency with which the CPU had to access this slow resource. Modern computers face similar limiting factors: main memories are slow compared to the CPU and the fast cache memories employed to overcome this are limited in size. This may partly explain why highly encoded instruction sets have proven to be as useful as RISC designs in modern computers.

RISC design philosophy

In the mid 1970s researchers (particularly John Cocke) at IBM (and similar projects elsewhere) demonstrated that the majority of combinations of these orthogonal addressing modes and instructions were not used by most programs generated by compilers available at the time. It proved difficult in many cases to write a compiler with more than limited ability to take advantage of the features provided by conventional CPUs.

It was also discovered that, on microcoded implementations of certain architectures, complex operations tended to be *slower* than a sequence of simpler operations doing the same thing. This was in part an effect of the fact that many designs were rushed, with little time to optimize or tune every instruction, but only those used most often. One infamous example was the VAX's `INDEX` instruction.

As mentioned elsewhere, core memory had long since been slower than many CPU designs. The advent of semiconductor memory reduced this difference, but it was still

apparent that more registers (and later caches) would allow higher CPU operating frequencies. Additional registers would require sizeable chip or board areas which, at the time (1975), could be made available if the complexity of the CPU logic was reduced.

Yet another impetus of both RISC and other designs came from practical measurements on real-world programs. Andrew Tanenbaum summed up many of these, demonstrating that processors often had oversized immediates. For instance, he showed that 98% of all the constants in a program would fit in 13 bits, yet many CPU designs dedicated 16 or 32 bits to store them. This suggests that, to reduce the number of memory accesses, a fixed length machine could store constants in unused bits of the instruction word itself, so that they would be immediately ready when the CPU needs them (much like immediate addressing in a conventional design). This required small opcodes in order to leave room for a reasonably sized constant in a 32-bit instruction word.

Since many real-world programs spend most of their time executing simple operations, some researchers decided to focus on making those operations as fast as possible. The clock rate of a CPU is limited by the time it takes to execute the slowest *sub-operation* of any instruction; decreasing that cycle-time often accelerates the execution of other instructions. The focus on "reduced instructions" led to the resulting machine being called a "reduced instruction set computer" (RISC). The goal was to make instructions so simple that they could *easily* be pipelined, in order to achieve a *single clock* throughput at *high frequencies*.

Later it was noted that one of the most significant characteristics of RISC processors was that external memory was only accessible by a *load* or *store* instruction. All other instructions were limited to internal registers. This simplified many aspects of processor design: allowing instructions to be fixed-length, simplifying pipelines, and isolating the logic for dealing with the delay in completing a memory access (cache miss, etc.) to only two instructions. This led to RISC designs being referred to as *load/store* architectures.

Instruction set size and alternative terminology

A common misunderstanding of the phrase "reduced instruction set computer" is the mistaken idea that instructions are simply eliminated, resulting in a smaller set of instructions. In fact, over the years, RISC instruction sets have grown in size, and today many of them have a larger set of instructions than many CISC CPUs. Some RISC processors such as the INMOS Transputer have instruction sets as large as, say, the CISC IBM System/370; and conversely, the DEC PDP-8 – clearly a CISC CPU because many of its instructions involve multiple memory accesses – has only 8 basic instructions, plus a few extended instructions.

The term "reduced" in that phrase was intended to describe the fact that the amount of work any single instruction accomplishes is reduced – at most a single data memory cycle – compared to the "complex instructions" of CISC CPUs that may require dozens of data memory cycles in order to execute a single instruction. In particular, RISC processors typically have separate instructions for I/O and data processing; as a consequence,

industry observers have started using the terms "register-register" or "load-store" to describe RISC processors.

Some CPUs have been retroactively dubbed RISC — a Byte magazine article once referred to the 6502 as "the original RISC processor" due to its simplistic and nearly orthogonal instruction set (most instructions work with most addressing modes) as well as its 256 zero-page "registers". The 6502 is no load/store design however: arithmetic operations may read memory, and instructions like INC and ROL even modify memory. Furthermore, orthogonality is equally often associated with "CISC". However, the 6502 may be regarded as similar to RISC (and early machines) in the fact that it uses no microcode sequencing. As for the well known fact that it employed longer but fewer clock cycles compared to many contemporary microprocessors was due to a more asynchronous design with less subdivision of internal machine cycles. This is similar to early machines, but not to RISC.

Some CPUs have been specifically designed to have a very small set of instructions – but these designs are very different from classic RISC designs, so they have been given other names such as minimal instruction set computer (MISC), Zero Instruction Set Computer (ZISC), one instruction set computer (OISC), transport triggered architecture (TTA), etc.

Alternatives

RISC was developed as an alternative to what is now known as CISC. Over the years, other strategies have been implemented as alternatives to RISC and CISC. Some examples are VLIW, MISC, OISC, massive parallel processing, systolic array, reconfigurable computing, and dataflow architecture.

Typical characteristics of RISC

For any given level of general performance, a RISC chip will typically have far fewer transistors dedicated to the core logic which originally allowed designers to increase the size of the register set and increase internal parallelism.

Other features, which are typically found in RISC architectures are:

- Uniform instruction format, using a single word with the opcode in the same bit positions in every instruction, demanding less decoding;
- Identical general purpose registers, allowing any register to be used in any context, simplifying compiler design (although normally there are separate floating point registers);
- Simple addressing modes. Complex addressing performed via sequences of arithmetic and/or load-store operations;
- Few data types in hardware, some CISCs have byte string instructions, or support complex numbers; this is so far unlikely to be found on a RISC.

Exceptions abound, of course, within both CISC and RISC.

RISC designs are also more likely to feature a Harvard memory model, where the instruction stream and the data stream are conceptually separated; this means that modifying the memory where code is held might not have any effect on the instructions executed by the processor (because the CPU has a separate instruction and data cache), at least until a special synchronization instruction is issued. On the upside, this allows both caches to be accessed simultaneously, which can often improve performance.

Many early RISC designs also shared the characteristic of having a branch delay slot. A branch delay slot is an instruction space immediately following a jump or branch. The instruction in this space is executed, whether or not the branch is taken (in other words the effect of the branch is delayed). This instruction keeps the ALU of the CPU busy for the extra time normally needed to perform a branch. Nowadays the branch delay slot is considered an unfortunate side effect of a particular strategy for implementing some RISC designs, and modern RISC designs generally do away with it (such as PowerPC, more recent versions of SPARC, and MIPS).

Early RISC

The first system that would today be known as RISC was the CDC 6600 supercomputer, designed in 1964, a decade before the term was invented. The CDC 6600 had a load-store architecture with only two addressing modes (register+register, and register+immediate constant) and 74 opcodes (whereas an Intel 8086 has 400). The 6600 had eleven pipelined functional units for arithmetic and logic, plus five load units and two store units; the memory had multiple banks so all load-store units could operate at the same time. The basic clock cycle/instruction issue rate was 10 times faster than the memory access time. Jim Thornton and Seymour Cray designed it as a number-crunching CPU supported by 10 simple computers called "peripheral processors" to handle I/O and other operating system functions. Thus the joking comment later that the acronym RISC actually stood for "**R**eally **I**nvented by **S**eymour **C**ray".

Another early load-store machine was the Data General Nova minicomputer, designed in 1968 by Edson de Castro. It had an almost pure RISC instruction set, remarkably similar to that of today's ARM processors; however it has not been cited as having influenced the ARM designers, although Novas were in use at the University of Cambridge Computer Laboratory in the early 1980s.

The earliest attempt to make a chip-based RISC CPU was a project at IBM which started in 1975. Named after the building where the project ran, the work led to the IBM 801 CPU family which was used widely inside IBM hardware. The 801 was eventually produced in a single-chip form as the ROMP in 1981, which stood for 'Research OPD [Office Products Division] Micro Processor'. As the name implies, this CPU was designed for "mini" tasks, and when IBM released the IBM RT-PC based on the design in 1986, the performance was not acceptable. Nevertheless the 801 inspired several research projects, including new ones at IBM that would eventually lead to their POWER system.

The most public RISC designs, however, were the results of university research programs run with funding from the DARPA VLSI Program. The VLSI Program, practically unknown today, led to a huge number of advances in chip design, fabrication, and even computer graphics.

UC Berkeley's RISC project started in 1980 under the direction of David Patterson and Carlo H. Sequin, based on gaining performance through the use of pipelining and an aggressive use of a technique known as register windowing. In a normal CPU one has a small number of registers, and a program can use any register at any time. In a CPU with register windows, there are a huge number of registers, e.g. 128, but programs can only use a small number of them, e.g. 8, at any one time. A program that limits itself to 8 registers per procedure can make very fast procedure calls: The call simply moves the window "down" by 8, to the set of 8 registers used by that procedure, and the return moves the window back. (On a normal CPU, most calls must save at least a few registers' values to the stack in order to use those registers as working space, and restore their values on return.)

The RISC project delivered the RISC-I processor in 1982. Consisting of only 44,420 transistors (compared with averages of about 100,000 in newer CISC designs of the era) RISC-I had only 32 instructions, and yet completely outperformed any other single-chip design. They followed this up with the 40,760 transistor, 39 instruction RISC-II in 1983, which ran over three times as fast as RISC-I.

At about the same time, John L. Hennessy started a similar project called MIPS at Stanford University in 1981. MIPS focused almost entirely on the pipeline, making sure it could be run as "full" as possible. Although pipelining was already in use in other designs, several features of the MIPS chip made its pipeline far faster. The most important, and perhaps annoying, of these features was the demand that all instructions be able to complete in one cycle. This demand allowed the pipeline to be run at much higher data rates (there was no need for induced delays) and is responsible for much of the processor's performance. However, it also had the negative side effect of eliminating many potentially useful instructions, like a multiply or a divide.

In the early years, the RISC efforts were well known, but largely confined to the university labs that had created them. The Berkeley effort became so well known that it eventually became the name for the entire concept. Many in the computer industry criticized that the performance benefits were unlikely to translate into real-world settings due to the decreased memory efficiency of multiple instructions, and that that was the reason no one was using them. But starting in 1986, all of the RISC research projects started delivering products.

Later RISC

Berkeley's research was not directly commercialized, but the RISC-II design was used by Sun Microsystems to develop the SPARC, by Pyramid Technology to develop their line of mid-range multi-processor machines, and by almost every other company a few years

later. It was Sun's use of a RISC chip in their new machines that demonstrated that RISC's benefits were real, and their machines quickly outpaced the competition and essentially took over the entire workstation market.

John Hennessy left Stanford (temporarily) to commercialize the MIPS design, starting the company known as MIPS Computer Systems. Their first design was a second-generation MIPS chip known as the **R2000**. MIPS designs went on to become one of the most used RISC chips when they were included in the PlayStation and Nintendo 64 game consoles. Today they are one of the most common embedded processors in use for high-end applications.

IBM learned from the RT-PC failure and went on to design the RS/6000 based on their new POWER architecture. They then moved their existing AS/400 systems to POWER chips, and found much to their surprise that even the very complex instruction set ran considerably faster. POWER would also find itself moving "down" in scale to produce the PowerPC design, which eliminated many of the "IBM only" instructions and created a single-chip implementation. Today the PowerPC is one of the most commonly used CPUs for automotive applications (some cars have more than 10 of them inside). It was also the CPU used in most Apple Macintosh machines from 1994 to 2006. (Starting in February 2006, Apple switched their main production line to Intel x86 processors.)

Almost all other vendors quickly joined. From the UK similar research efforts resulted in the INMOS transputer, the Acorn Archimedes and the Advanced RISC Machine line, which is a huge success today. Companies with existing CISC designs also quickly joined the revolution. Intel released the i860 and i960 by the late 1980s, although they were not very successful. Motorola built a new design called the 88000 in homage to their famed CISC 68000, but it saw almost no use and they eventually abandoned it and joined IBM to produce the PowerPC. AMD released their 29000 which would go on to become the most popular RISC design of the early 1990s.

Today the vast majority of all 32-bit CPUs in use are RISC CPUs, and microcontrollers. RISC design techniques offers power in even small sizes, and thus has become dominant for low-power 32-bit CPUs. Embedded systems are by far the largest market for processors: while a family may own one or two PCs, their car(s), cell phones, and other devices may contain a total of dozens of embedded processors. RISC had also completely taken over the market for larger workstations for much of the 90s (until taken back by inexpensive PC-based solutions). After the release of the Sun SPARCstation the other vendors rushed to compete with RISC based solutions of their own. The high-end server market today is almost completely RISC based. The #1 spot among supercomputers as of 2008 was held by IBM's Roadrunner system, which uses Power Architecture-based Cell processors to provide most of its computing power; however, the #1 spot as of November 2010 was held by the Tianhe-IA, which uses a combination of Intel Xeon processors, Nvidia Tesla GPGPUs, and custom processors, and most of the other machines in the top 10 spots use x86 CISC processors instead.

RISC and x86

However, despite many successes, RISC has made few inroads into the desktop PC and commodity server markets, where Intel's x86 platform remains the dominant processor architecture. There are three main reasons for this:

1. The very large base of proprietary PC applications are written for x86, whereas no RISC platform has a similar installed base, and this meant PC users were locked into the x86.
2. Although RISC was indeed able to scale up in performance quite quickly and cheaply, Intel took advantage of its large market by spending vast amounts of money on processor development. Intel could spend many times as much as any RISC manufacturer on improving low level design and manufacturing. The same could not be said about smaller firms like Cyrix and NexGen, but they realized that they could apply (tightly) pipelined design practices also to the x86-architecture, just like in the 486 and Pentium. The 6x86 and MII series did exactly this, but was more advanced, it implemented superscalar speculative execution via register renaming, directly at the x86-semantic level. Others, like the Nx586 and AMD K5 did the same, but *indirectly*, via dynamic microcode buffering and semi-independent superscalar scheduling and instruction dispatch at the micro-operation level (older or simpler 'CISC' designs typically executes rigid micro-operation sequences directly). The first *available* chip deploying such dynamic buffering and scheduling techniques was the NexGen Nx586, released in 1994; the AMD K5 was severely delayed and released in 1995.
3. Later, more powerful processors such as Intel P6, AMD K6, AMD K7, Pentium 4, etc. employed similar dynamic buffering and scheduling principles and implemented loosely coupled superscalar (and speculative) execution of micro-operation sequences generated from several parallel x86 decoding stages. Today, these ideas have been further refined (some x86-pairs are instead merged, into a more complex micro-operation, for example) and are still used by modern x86 processors such as Intel Core 2 and AMD K8.

While early RISC designs were significantly different than contemporary CISC designs, by 2000 the highest performing CPUs in the RISC line were almost indistinguishable from the highest performing CPUs in the CISC line.

A number of vendors, including Qualcomm, are attempting to enter the PC market with ARM-based devices dubbed smartbooks, riding off the netbook trend and rising acceptance of Linux distributions, a number of which already have ARM builds. Other companies are choosing to use Windows CE.

Diminishing benefits for desktops and servers

Over time, improvements in chip fabrication techniques have improved performance exponentially, according to Moore's law, whereas architectural improvements have been comparatively small. Modern CISC implementations have implemented many of the

performance improvements introduced by RISC, such as single-clock throughput of simple instructions. Compilers have also become more sophisticated, and are better able to exploit complex as well as simple instructions on CISC architectures, often carefully optimizing both instruction selection and instruction and data ordering in pipelines and caches. The RISC-CISC distinction has blurred significantly in practice.

Expanding benefits for mobile and embedded devices

The hardware translation from x86 instructions into RISC operations, which cost relatively little in microprocessors for desktops and servers as Moore's Law provided more transistors, become significant in area and energy for mobile and embedded devices. Hence, ARM processors dominate cell phones and tablets today just like x86 processors dominate PCs.

RISC success stories

RISC designs have led to a number of successful platforms and architectures, some of the larger ones being:

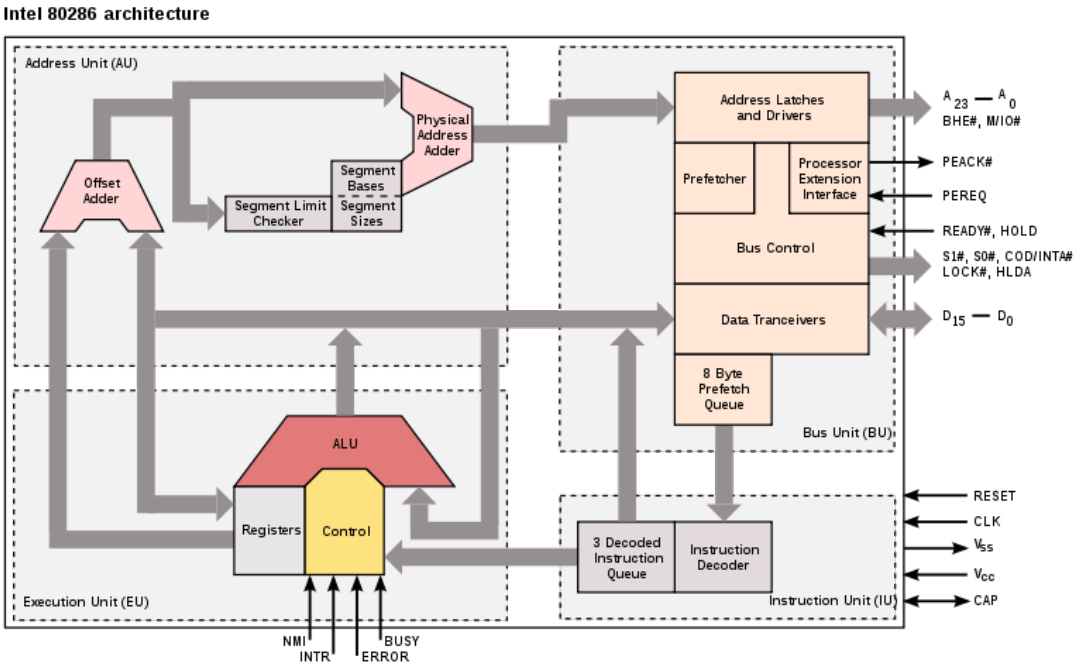
- ARM — The ARM architecture dominates the market for low power and low cost embedded systems (typically 100–500 MHz in 2008). ARM Ltd., which licenses intellectual property rather than manufacturing chips, reported that 10 billion licensed chips had been shipped as of early 2008. The various generations, variants and implementations of the ARM core are deployed in over 90% of mobile electronics devices, including almost all modern mobile phones, mp3 players and portable video players. Some high profile examples are
    - Apple iPods (custom ARM7TDMI SoC)
    - Apple iPhone and iPod Touch (Samsung ARM1176JZF, ARM Cortex-A8, Apple A4)
    - Apple iPad (Apple A4 ARM-based SoC)
    - Palm and PocketPC PDAs and smartphones (Marvell XScale family, Samsung SC32442 - ARM9)
    - RIM BlackBerry smartphone/email devices.
    - Microsoft Windows Mobile
    - Nintendo Game Boy Advance (ARM7TDMI)
    - Nintendo DS (ARM7TDMI, ARM946E-S)
    - Sony Network Walkman (Sony in-house ARM based chip)
    - T-Mobile G1 (HTC Dream Android, Qualcomm MSM7201A ARM11 @ 528 MHz)
- MIPS's MIPS line, found in most SGI computers and the PlayStation, PlayStation 2, Nintendo 64 (discontinued), PlayStation Portable game consoles, and residential gateways like Linksys WRT54G series.
- IBM's and Freescale's (formerly Motorola SPS) Power Architecture, used in all of IBM's supercomputers, midrange servers and workstations, in Apple's PowerPC-based Macintosh computers (discontinued), in Nintendo's Gamecube and Wii, Microsoft's Xbox 360 and Sony's PlayStation 3 game consoles, EMC's DMX

range of the Symmetrix SAN, and in many embedded applications like printers and cars.

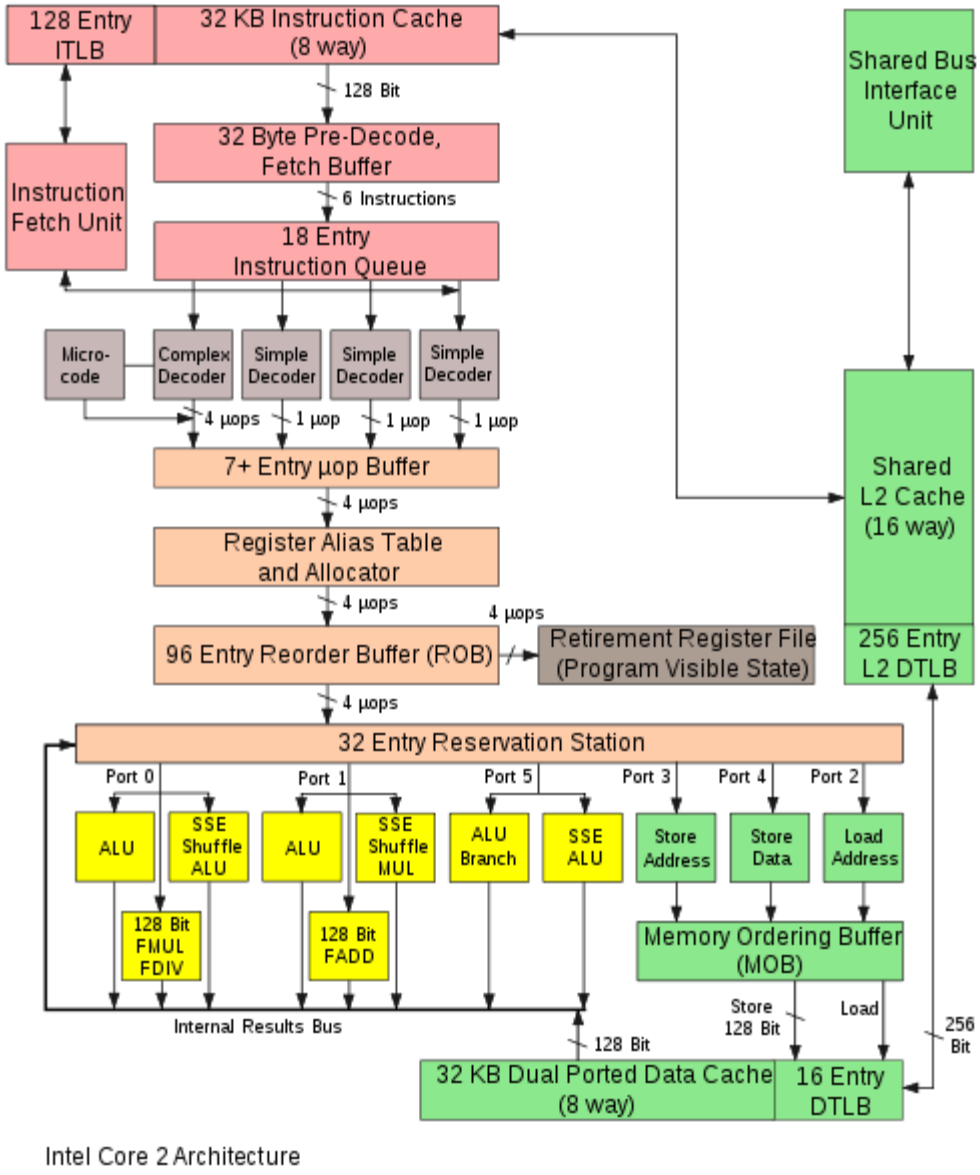
- SPARC, by Oracle (previously Sun Microsystems), and Fujitsu
- Hewlett-Packard's PA-RISC, also known as HP-PA, discontinued December 31, 2008.
- Alpha, used in single-board computers, workstations, servers and supercomputers from Digital Equipment Corporation, Compaq and HP, discontinued as of 2007.
- XAP processor used in many low-power wireless (Bluetooth, wifi) chips from CSR.
- Hitachi's SuperH, originally in wide use in the Sega Super 32X, Saturn and Dreamcast, now at the heart of many consumer electronics devices. The SuperH is the base platform for the Mitsubishi - Hitachi joint semiconductor group. The two groups merged in 2002, dropping Mitsubishi's own RISC architecture, the M32R.
- Atmel AVR used in a variety of products including ranging from Xbox handheld controllers to BMW cars.

Chapter-13

Microarchitecture

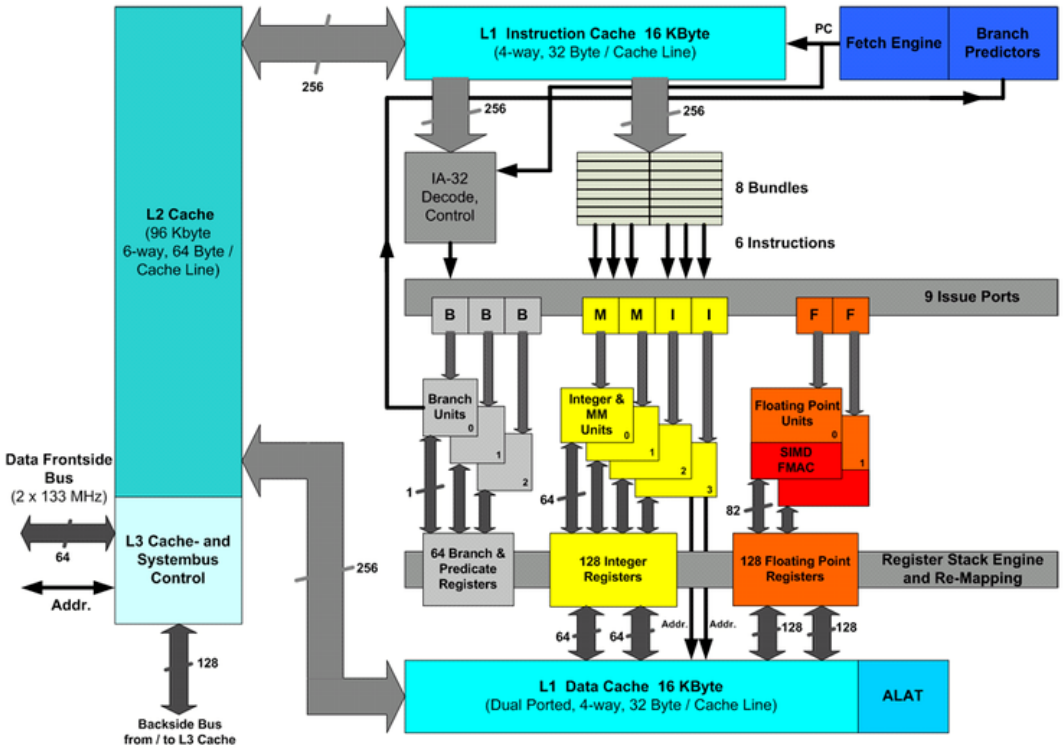


Intel 80286 microarchitecture.

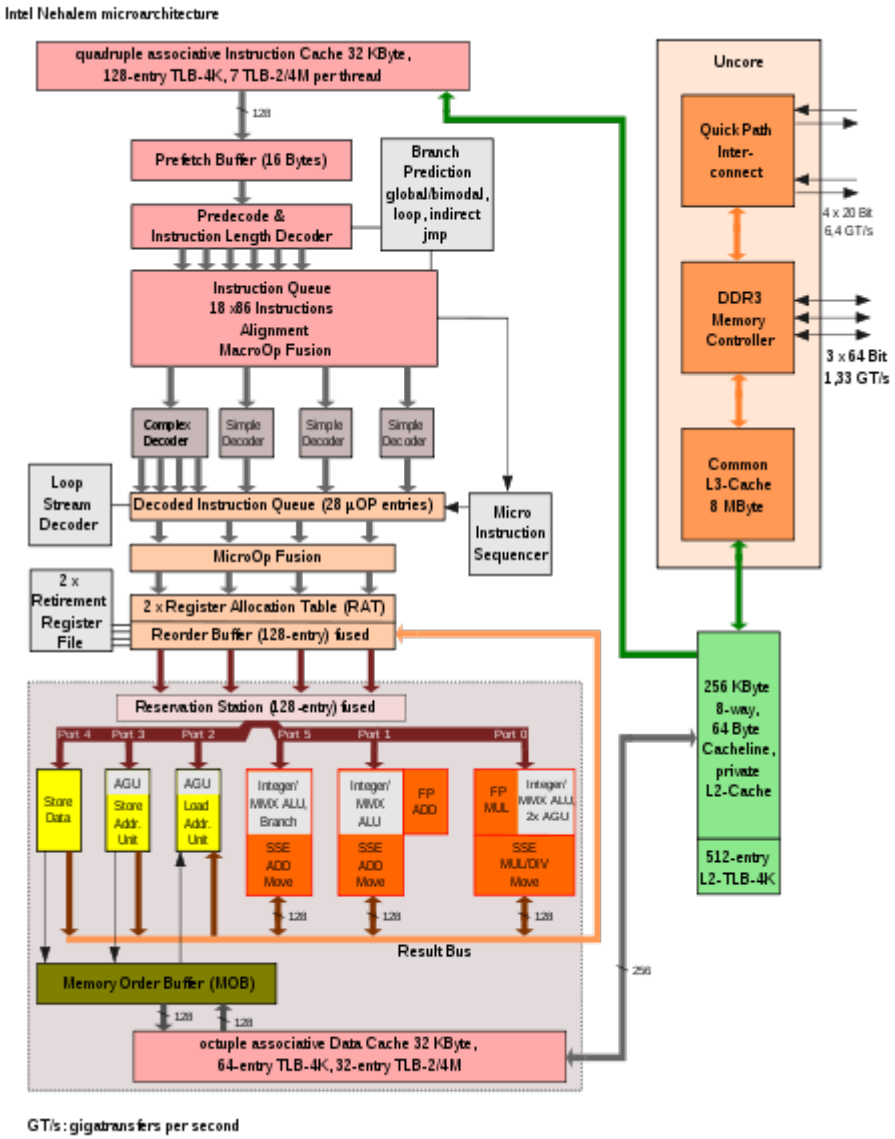


Intel Core microarchitecture.

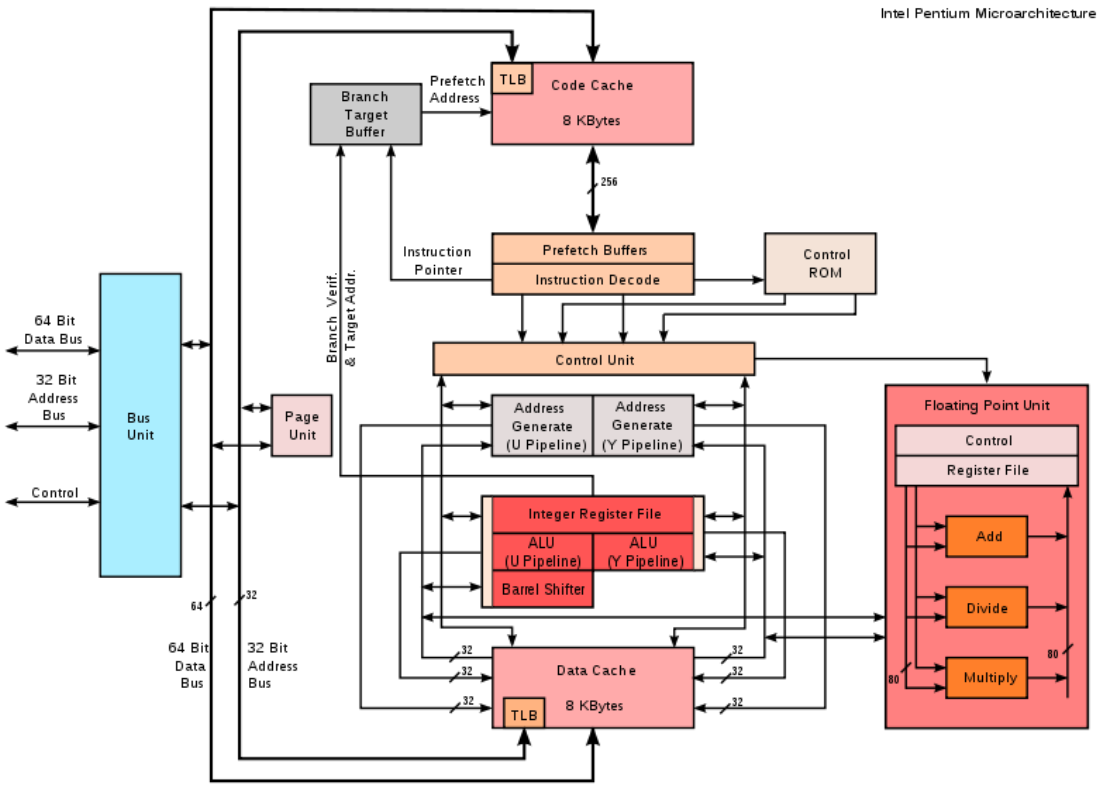
In computer engineering, **microarchitecture** (sometimes abbreviated to µarch or uarch), also called **computer organization**, is the way a given instruction set architecture (ISA) is implemented on a processor. A given ISA may be implemented with different microarchitectures. Implementations might vary due to different goals of a given design or due to shifts in technology. Computer architecture is the combination of microarchitecture and instruction set design.



Intel itanium



Intel Nehalem



Intel Pentium

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 microarchitectural elements of the machine, which may be everything from single gates and registers, to complete arithmetic logic units (ALU)s 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).

Each microarchitectural element is in turn represented by a schematic describing the interconnections of logic gates used to implement it. Each logic gate is in turn represented by a circuit diagram describing the connections of the transistors used to implement it in some particular logic family. 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.

In principle, a single microarchitecture could execute several different ISAs with only minor changes to the microcode.

Aspects of microarchitecture

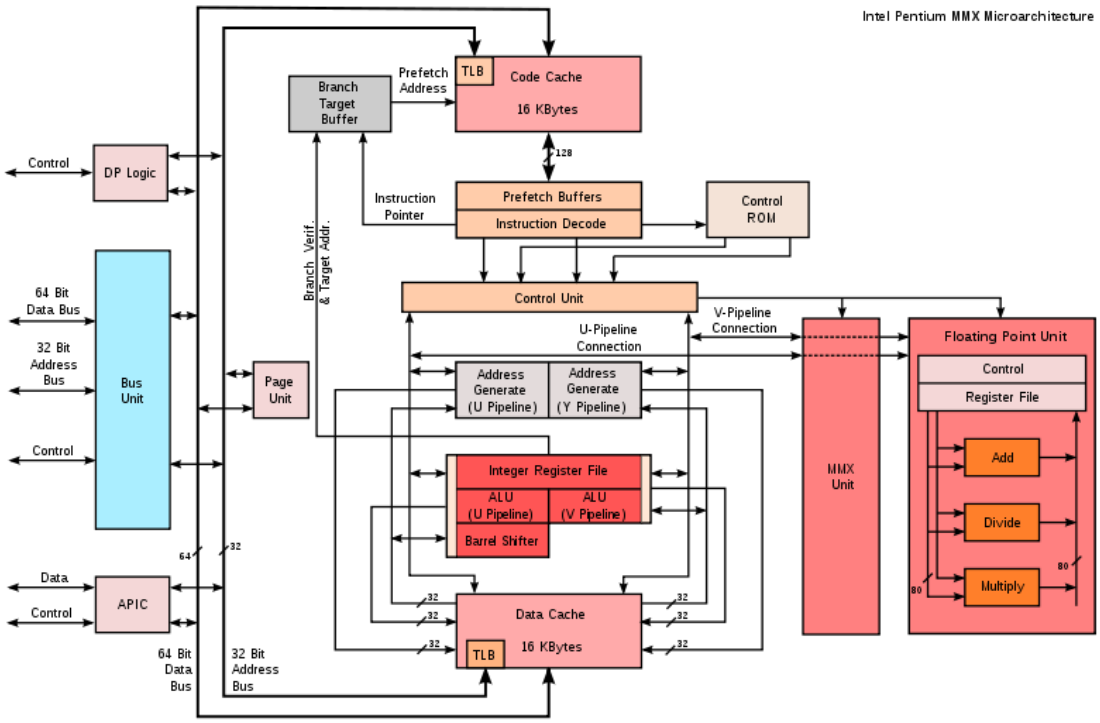
The pipelined datapath is the most commonly used datapath design in microarchitecture today. This technique is used in most modern microprocessors, microcontrollers, and DSPs. The pipelined architecture allows multiple instructions to overlap in execution, much like an assembly line. The pipeline includes 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.

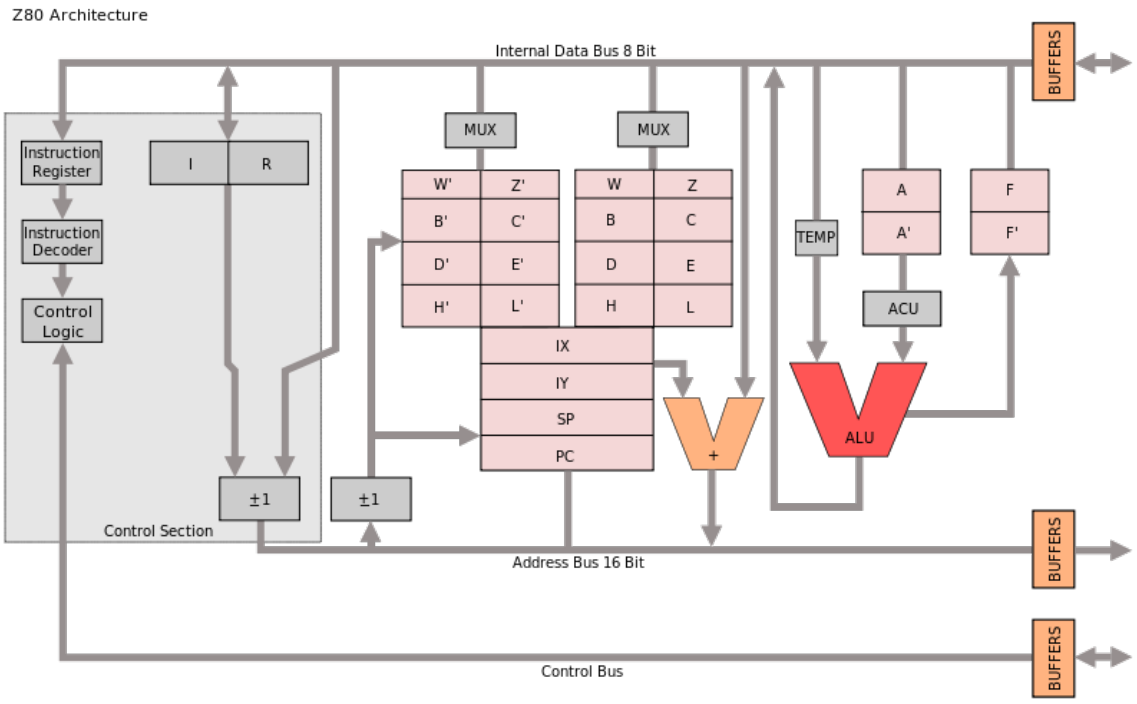
System-level design decisions such as whether or not to include peripherals, such as memory controllers, can be considered part of the microarchitectural design process. This includes decisions on the performance-level and connectivity of these peripherals.

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



Intel Pentium MMX



Z80

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 particularly 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.

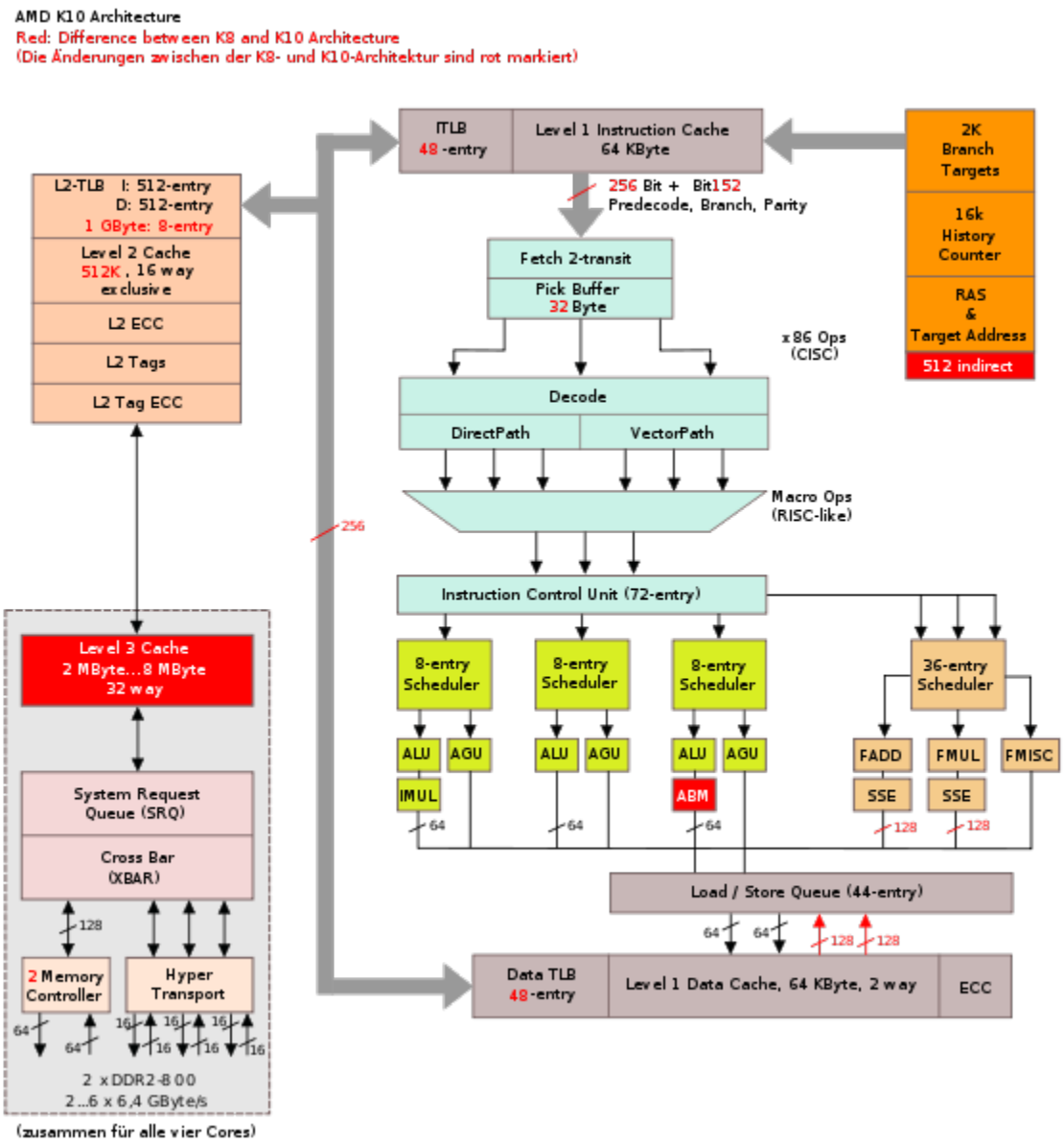
Instruction pipelining

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.

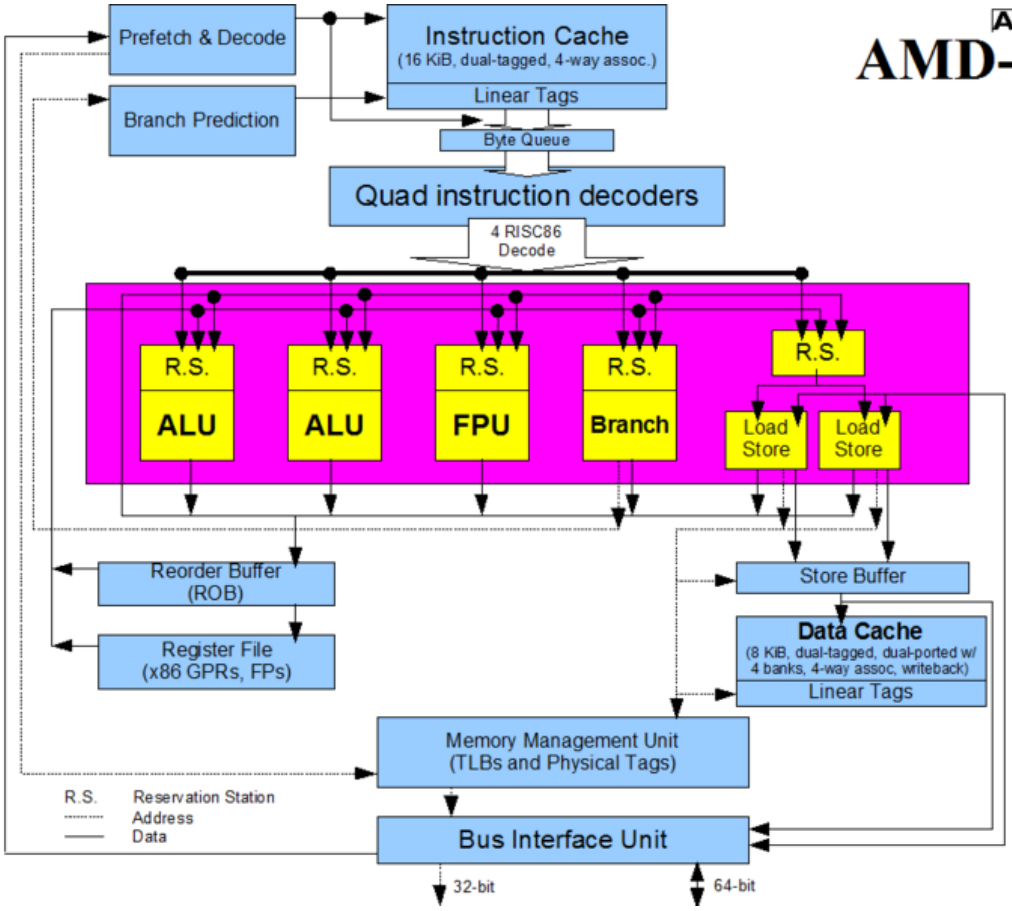
Pipelines improve performance by allowing a number of instructions to work their way through the processor at the same time. In the same basic example, the processor would start to decode (step 1) a new instruction while the last one was waiting for results. This would allow up to four instructions to 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.

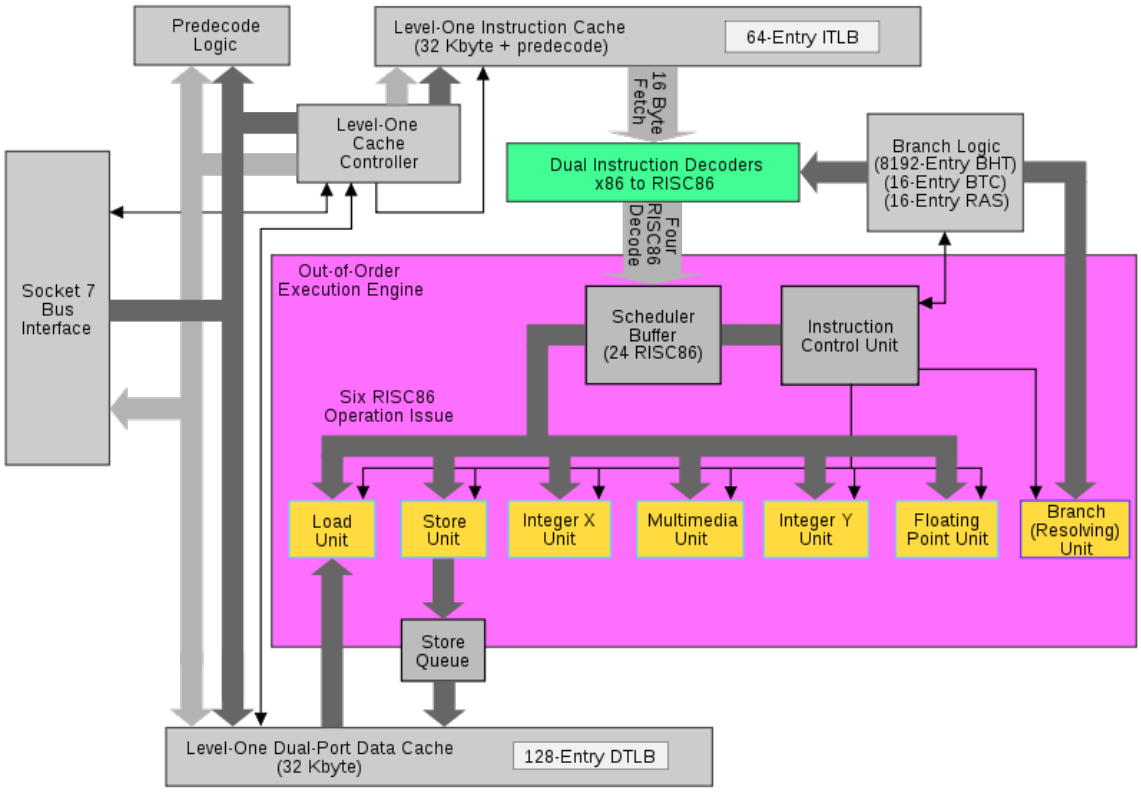
Pipelines are by no means limited to RISC designs. By 1986 the top-of-the-line VAX implementation (VAX 8800) was a heavily pipelined design, slightly predating the first commercial MIPS and SPARC designs. Most modern CPUs (even embedded CPUs) are now pipelined, and microcoded CPUs with no pipelining 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.


AMD K10 Architecture
Red: Difference between K8 and K10 Architecture
(Die Änderungen zwischen der K8- und K10-Architektur sind rot markiert)
ITLB
48 -entry
Level 1 Instruction Cache
64 KByte
2K
Branch
Targets
16k
History
Counter
RAS
&
Target Address
512 indirect
L2-TLB I: 512-entry
D: 512-entry
1 GByte: 8-entry
Level 2 Cache
512K , 16 way
exclusive
L2 ECC
L2 Tags
L2 Tag ECC
256 Bit + Bit152
Predecode, Branch, Parity
Fetch 2-transit
Pick Buffer
32 Byte
x86 Ops
(CISC)
Decode
DirectPath
VectorPath
Macro Ops
(RISC-like)
256
Instruction Control Unit (72-entry)
8-entry
Scheduler
8-entry
Scheduler
8-entry
Scheduler
36-entry
Scheduler
ALU
AGU
ALU
AGU
ALU
AGU
FADD
FMUL
FMISC
IMUL
ABM
SSE
SSE
64
64
64
128
128
Load / Store Queue (44-entry)
64
64
128
128
Level 3 Cache
2 MByte...8 MByte
32 way
System Request
Queue (SRQ)
Cross Bar
(XBAR)
128
2 Memory
Controller
Hyper
Transport
64
64
16
16
16
16
16
16
2 xDDR2-800
2...6 x 6,4 GByte/s
(zusammen für alle vier Cores)
Data TLB
48-entry
Level 1 Data Cache, 64 KByte, 2 way
ECC

AMD K10



AMDK5Diagram



Amdk6

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, due 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 prefetch instructions without waiting for the register read. Speculative execution is a further enhancement in which the code along the predicted path is not just prefetched 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

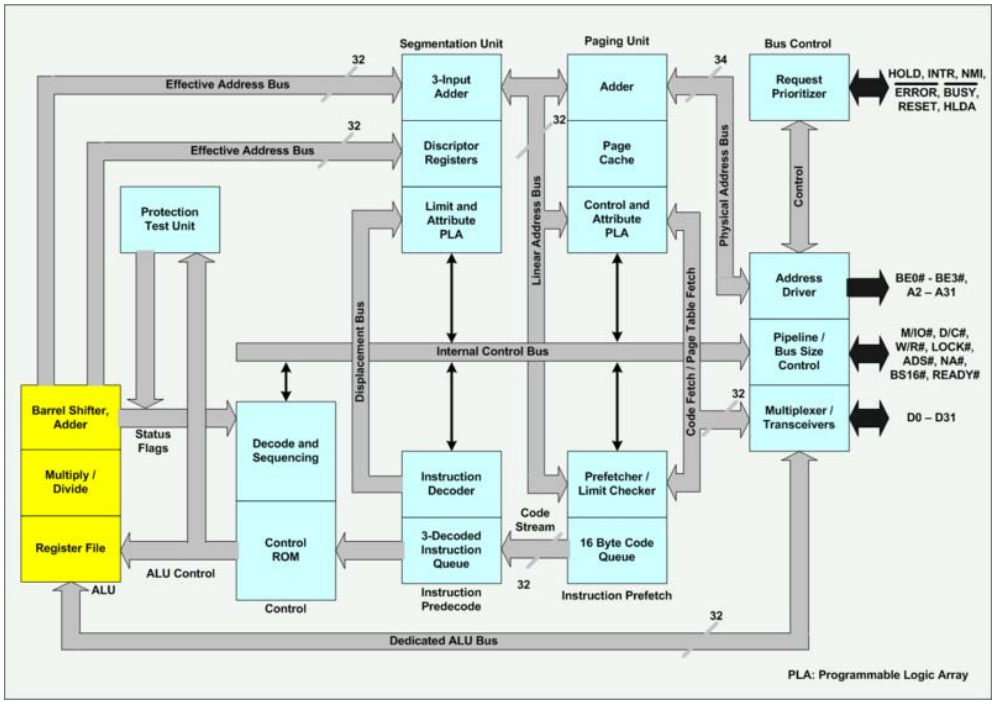
execution units that are idle at that point. The results are then collected and re-ordered at the end.

Out-of-order execution

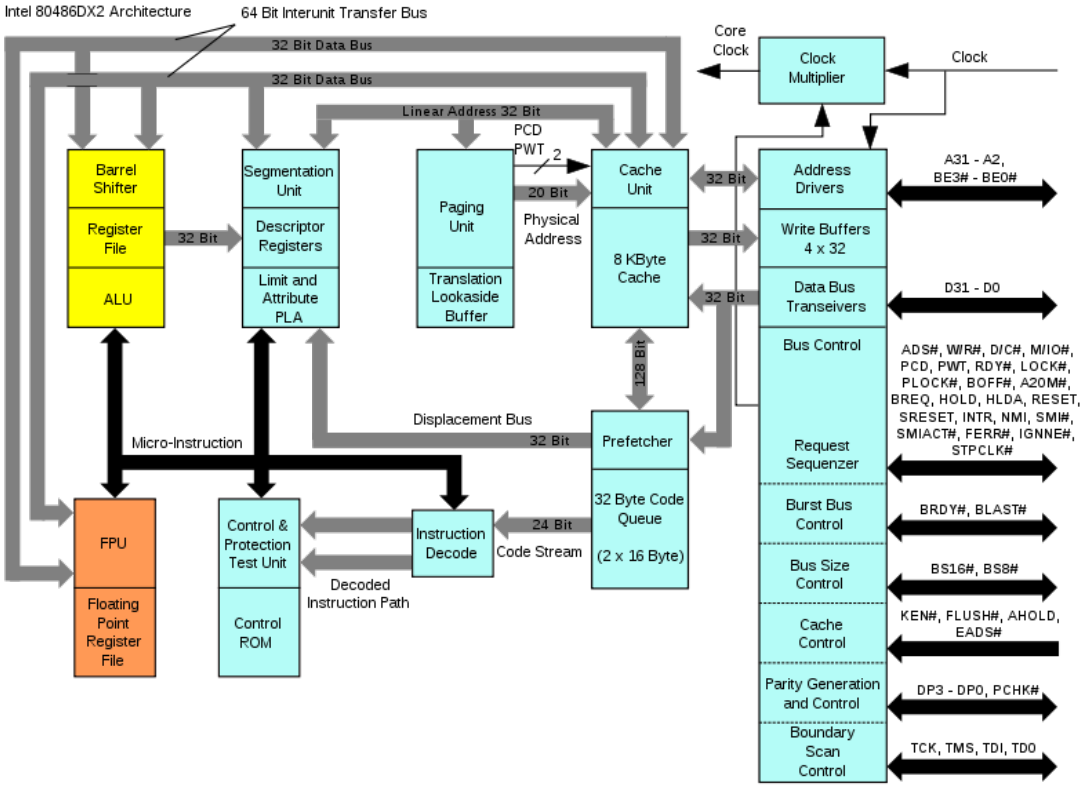
The addition of caches reduces the frequency or duration of stalls due to waiting for data to be fetched from the memory hierarchy, but does not get rid of these stalls entirely. In early designs a *cache miss* would force the cache controller to stall the processor and wait. Of course there may be some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

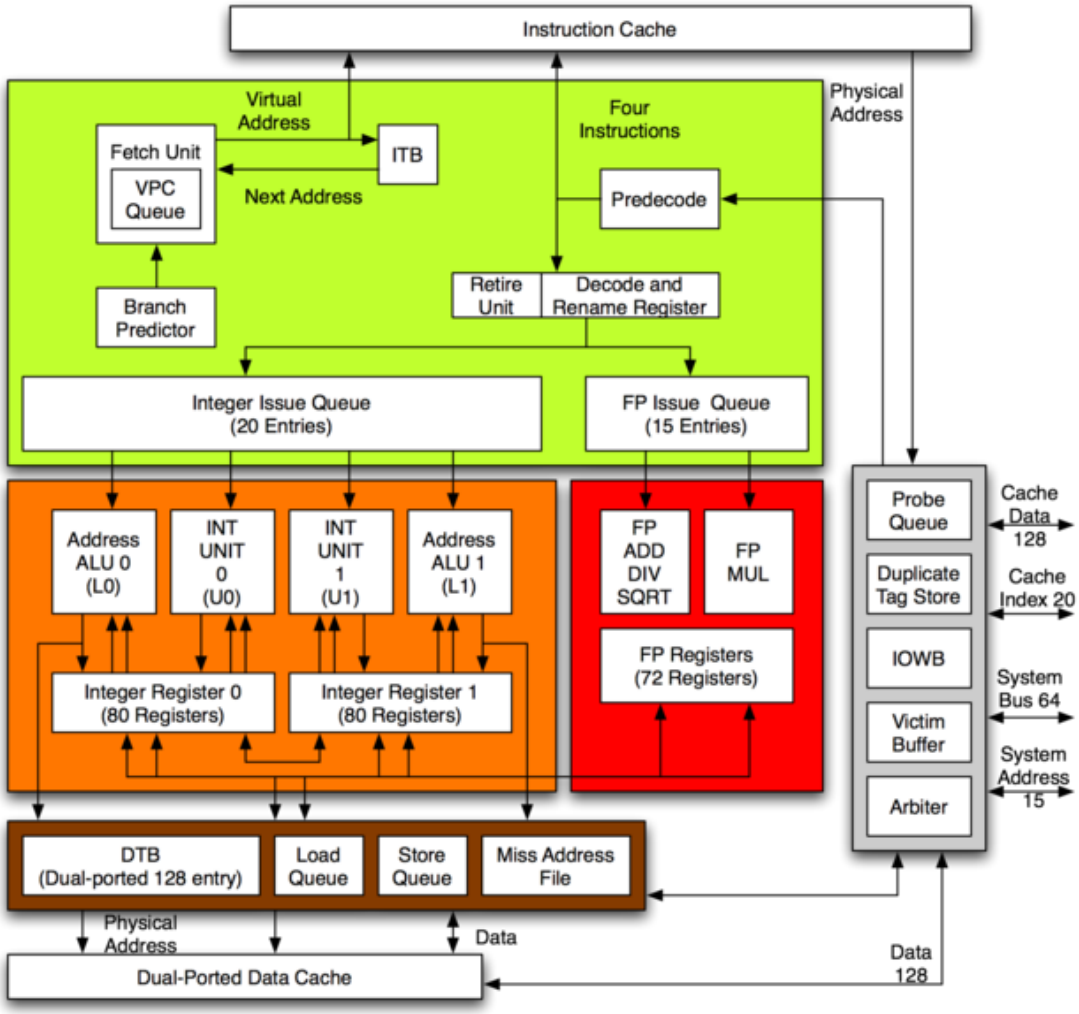
Register renaming refers to a technique used to avoid unnecessary serialized execution of program instructions because of the reuse of the same registers by those instructions. Suppose we have two groups of instruction that will use the same register. One set of instructions is executed first to leave 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.



80386DX


Intel 80486DX2 Architecture
64 Bit Interunit Transfer Bus
32 Bit Data Bus
32 Bit Data Bus
Core Clock
Clock Multiplier
Clock
Linear Address 32 Bit
PCD
PWT
2
Barrel Shifter
Register File
ALU
32 Bit
Segmentation Unit
Descriptor Registers
Limit and Attribute PLA
Paging Unit
Translation Lookaside Buffer
20 Bit
Physical Address
Cache Unit
8 KByte Cache
32 Bit
32 Bit
32 Bit
128 Bit
Address Drivers
Write Buffers 4 x 32
Data Bus Transeivers
Bus Control
Request Sequenzer
Burst Bus Control
Bus Size Control
Cache Control
Parity Generation and Control
Boundary Scan Control
A31 - A2, BE3# - BE0#
D31 - D0
ADS#, W/R#, D/C#, M/IO#, PCD, PWT, RDY#, LOCK#, PLOCK#, BOFF#, A20M#, BREQ, HOLD, HLDA, RESET, SRESET, INTR, NMI, SMI#, SMIACT#, FERR#, IGNNE#, STPCLK#
BRDY#, BLAST#
BS16#, BS8#
KEN#, FLUSH#, AHOLD, EADS#
DP3 - DP0, PCHK#
TCK, TMS, TDI, TDO
Displacement Bus
32 Bit
Prefetcher
32 Byte Code Queue
(2 x 16 Byte)
Micro-Instruction
FPU
Floating Point Register File
Control & Protection Test Unit
Control ROM
Instruction Decode
24 Bit
Code Stream
Decoded Instruction Path

80486DX2



Alpha 21264

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 web-site 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.

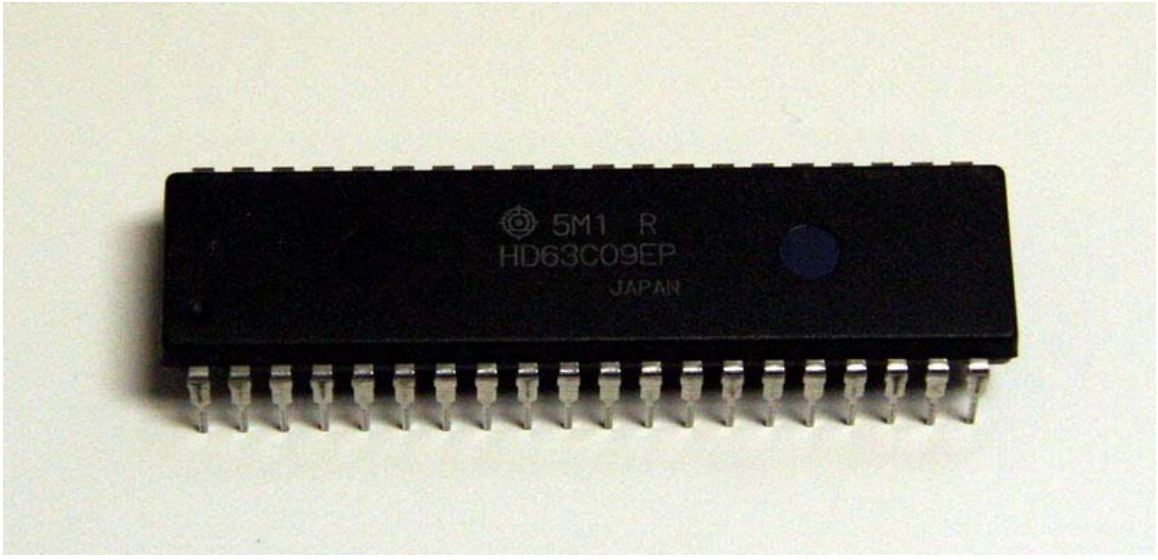
Conceptually, multithreading is equivalent to a context switch at the operating system level. The difference is that a multithreaded CPU can do a thread switch in one CPU cycle instead of the hundreds or thousands of CPU cycles a context switch normally requires. This is achieved by replicating the state 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.

Chapter-14

Hitachi 6309 and RCA 1802

Hitachi 6309

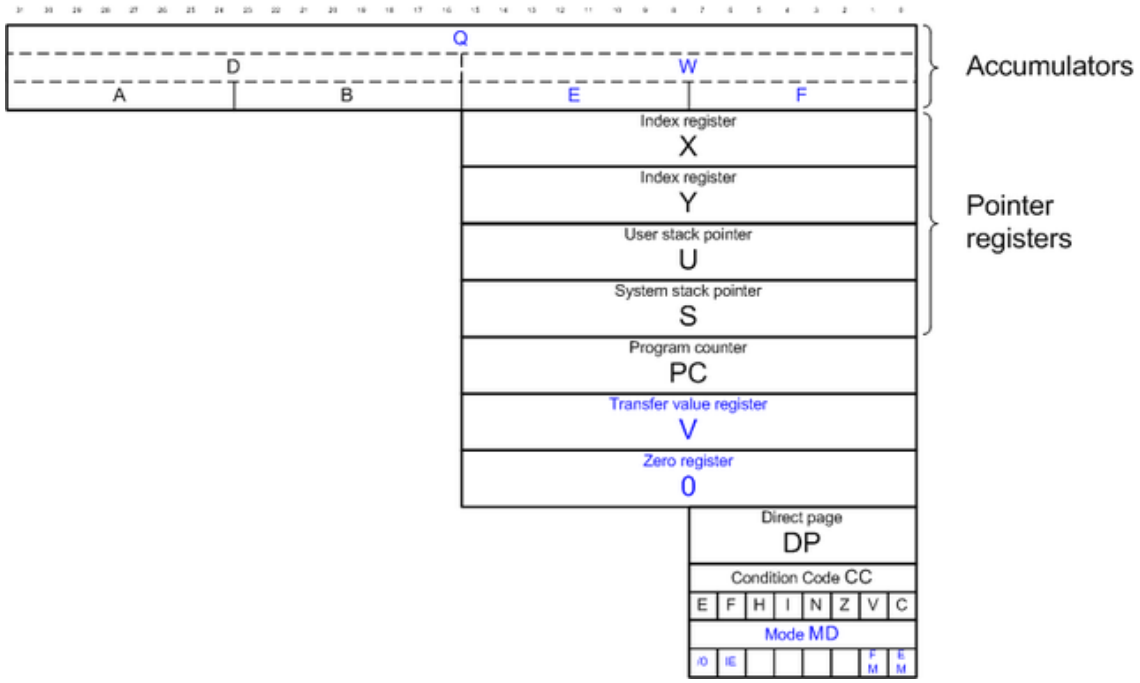


Hitachi 63C09E, a 3MHz external clock version of the 6309

The **6309** is Hitachi's CMOS version of the Motorola 6809 microprocessor. While in "Emulation Mode" it is fully compatible with the 6809. To the 6809 specifications it adds higher clock rates, enhanced features, new instructions, and additional registers. Most new instructions were added to support the additional registers, as well as up to 32-bit math, hardware division, bit manipulations, and block transfers. The 6309 is generally 30% faster in native mode than the 6809.

Surprisingly, this information was never published by Hitachi. The April 1988 issue of *Oh! FM*, a Japanese magazine for Fujitsu personal computer users, contained the first description of the 6309's additional capabilities. Later, Hirotsugu Kakugawa posted details of the 6309's new features and instructions to comp.sys.m6809. This led to the development of NitrOS9 for the Tandy Color Computer 3.

Programming Model



6309 Programming Model, showing register layout. Additions to the 6809 are shown with blue type.

Differences from the Motorola 6809

The 6309 differs from the 6809 in several key areas.

Process Technology

The 6309 is fabricated in CMOS technology, while the 6809 is an NMOS device. As a result, the 6309 requires less power to operate than the 6809. It is also a fully static device, which will not lose internal state information. This means it can be used with external DMA without needing refresh every 14 cycles as the 6809 does.

Clock Speed

The 6309 has B (2 MHz) versions as the 6809 does. However, a "C" speed rating was produced with either a 3.0 or 3.5 MHz maximum clock rate, depending on which datasheet is referenced. (Several Japanese computers had 63C09 CPUs clocked at 3.58 MHz, the NTSC colorburst frequency, so the 3.5 rating seems most likely). Anecdotal and individual reports indicate that the 63C09 variant can be clocked at 5 MHz with no ill effects. Like the 6809, the Hitachi CPU comes in both internal and external clock versions (HD63B/C09 and HD63B/C09E respectively)

Computational Efficiency

When switched into 6309 Native Mode (as opposed to the default 6809-compatible mode) many key instructions will complete in fewer clock cycles. This often improves execution speeds by up to 30%.

Additional Registers

- There are two additional 8-bit accumulators, E and F. These can be concatenated to form a 16-bit accumulator called W. The existing 6809 16-bit accumulator, D, can also be concatenated with W to form a 32-bit accumulator Q. (Presumably standing for "Quad").
- A "Transfer register", V, which is only accessible via inter-register instructions. Its value is not cleared during a hardware reset, so it can maintain a constant 'Value', hence "V".
- An 8/16-bit Zero register, called 0, is provided for speeding up operations where a zero constant is used. This register always returns a zero value, and writing to it has no effect.
- A new mode register, MD, which controls the 6309's operating mode and operates as a secondary condition code. Only 4 bits of this register are defined.

Additional Instructions

Most of the new instructions are modifications of existing instructions to handle the existence of the additional registers, such as load, store, add, and the like. Genuine 6309 additions include inter-register arithmetic, block transfers, hardware division, and bit-level manipulations.

Despite the user-friendliness of the additional instructions, analysis by 6809 programming gurus indicates that many of the new instructions are actually slower than the equivalent 6809 code, especially in tight loops. Careful analysis should be done to ensure that the programmer uses the most efficient code for the particular application.

Additional Hardware Features

It is possible to change the mode of operation for the FIRQ interrupt. Instead of stacking the PC and CC registers (normal 6809 behavior) the FIRQ interrupt can be set to stack the entire register set, as the IRQ interrupt does. In addition, the 6309 has two possible trap modes, one for an illegal instruction fetch and one for division by zero. The illegal instruction fetch is not maskable, and many TRS-80 Color Computer users reported that their 6309's were "buggy" when in reality it was an indicator of enhanced and unknown features.

RCA 1802

RCA CDP 1802.

The **RCA CDP1802**, also known as the **COSMAC** (***Co**mplementary **S**ymmetry **M**onolithic **A**rray **C**omputer*), is an 8-bit CMOS microprocessor (μP) introduced by RCA in early 1976. It is currently being manufactured by Intersil Corporation as a high-reliability microprocessor. The 1802 has an architecture different from most other 8-bit microprocessors.

In 1970 and 1971, Joseph Weisbecker developed a new 8-bit architecture computer system,. RCA released Weisbecker's work as the COSMAC 1801R and 1801U in early 1975, using its CMOS process (called *COSMOS*, an acronym for ***Co**mplementary **S**ilicon/**M**et**a**l-**o**xide **S**emiconductor*). In 1976, a team led by Jerry Herzog integrated the two chips into one, the 1802.

Introduction

The RCA 1802 has a static CMOS design with no minimum clock frequency, so that it can be run at very low speeds and low power. It has an 8-bit parallel bus with a bidirectional data bus and a multiplexed address bus (i.e., the high order byte of the 16-bit address and the low order byte of the address take turns in using the 8-bit physical address bus lines, by accessing the bus lines in different clock cycles).

The RCA 1802 has a single bit, programmable output port, and four input pins which are directly tested by branch instructions.

Its I/O mode is flexible and programmable, and it has a single-phase clock with an on-chip oscillator. Its register set consists of sixteen 16-bit registers. The program counter

(PC) can reside in any of these, providing a simple way to implement multiple PCs, pointers, or registers.

Applications

In addition to standard CMOS technology, the 1802 was also available fabricated in Silicon on Sapphire semiconductor process technology, which gives it a degree of resistance to radiation and electrostatic discharge (ESD). Along with its extreme low-power abilities, this makes the chip well-suited in space applications (also, at the time the 1802 was introduced, very few, if any, other radiation-hardened microprocessors were available in the market).

The Galileo spacecraft used multiple 1802 microprocessors. The 1802 has often been incorrectly claimed to have been used in the earlier Viking and Voyager spacecraft, but it was not available at the time those spacecraft were being designed, and primary sources describe the Viking and Voyager computers as having architectures very dissimilar to the 1802, and not being microprocessor-based. The 1802 has been widely used in Earth-orbiting satellites mainly for their primary computer but since the 1990s its use as a low complexity flight control and telecom systems computer has dominated.

The 1802 was also used in ACAL, a microprocessor based system for the detection of gasses in chemical warfare. ACAL was designed by Oldelft / Delft Instruments, a Dutch company in the military & defence business. ACAL only got as far as prototype stage and never made production.

Commercial applications included the MIL Key building access control system, made in Australia, and marketed by Philips in Europe in the 1980s.

A number of early microcomputers were based on the 1802, including the Comx-35, COSMAC ELF (1976), COSMAC VIP, Netronics ELF II, Quest SuperELF, Finnish Telmac 1800 and Oscom Nano, and Yugoslav Pecom 32 and 64, as well as the RCA Studio II video game console (one of the first consoles to use bitmapped graphics).

The first high-level language available for the 1802 was Forth, provided by Forth, Inc. in 1976.

Technical description

Registers and I/O

An important feature of the 1802 is the register file of sixteen registers of 16 bits each. Using the SEP instruction, you can select any of the 16 registers to be the program counter; using the SEX instruction, you can select any of the 16-bit registers to be the index register. Register R0 has the special use of holding the memory address for the built-in DMA controller.

The processor has 5 special I/O lines. There's a single Q output that can be set with the SEQ instruction and reset with the REQ instruction. There are four external flag inputs: EF1, EF2, EF3, EF4 and there are 8 dedicated branch instructions to conditionally branch based on the state of those input lines.

The EF and Q lines were typically overused on RCA 1802 based hobbyist computers because of the lines' favorable handling. It was typical for the Q line to drive a status LED, a cassette interface, an RS-232 interface, and the speaker. This meant that the user could actually hear RS-232 and cassette data being transmitted.

Subroutine calls

The processor does not have standard subroutine CALL immediate and RET instructions, though they can be emulated. The register file makes possible some interesting subroutine call and return mechanisms, though they are better suited to small programs than general purpose coding. A few commonly used subroutines can be called quickly by keeping their address in one of the 16 registers; the SEP instruction is used to call a subroutine pointed to by one of the 16 bit registers and another SEP to return to the caller (SEP stands for *Set Program Counter*, and selects which one of the 16 registers is to be used as the program counter from that point onwards). Before a subroutine returns, it jumps to the location immediately preceding its entry point so that after the SEP instruction returns control to the caller, the register will be pointing to the right value for next time. An interesting variation of this scheme is to have two or more subroutines in a ring so that they are called in round robin order. On early hobbyist computers, tricks like this were commonly used in the horizontal refresh interrupt to reprogram the scan line address to repeat each scan line 4 times for the video controller. Computed subroutine calls were no problem because all CALL instructions were indexed (some processors only had CALL immediate).

Addressing modes

Because of the 16-bit address bus, and the 8-bit data bus, the sixteen general purpose registers are 16 bits wide, but the accumulator (the so-called data register, or D-register) is only 8 bits wide. The accumulator, therefore, tends to be a bottleneck. Transferring the contents of one register to another involves four instructions (one Get and one Put on the HI byte of the register, and a similar pair for the LO byte: GHI R1; PHI R2; GLO R1; PLO R2). Similarly, loading a new constant into a register (such as a new address for a subroutine jump, or the address of a data variable) also involves four instructions (two load immediate, LDI, instructions, one for each half of the constant, each one followed by a Put instruction to the register, PHI and PLO).

The two addressing modes *Indirect register*, and *Indirect register with auto-increment* are then fairly efficient, to perform 8-bit operations on the data in the accumulator. There are no other addressing modes, though. Thus, the *direct addressing* mode needs to be emulated using the four instructions mentioned earlier to load the address into a spare

register; followed by an instruction to select that register as the index register; followed, finally, by the intended operation on the data variable that is pointed to by that address.

DMA and Load Mode

The CDP1802 has a built-in DMA controller, having two DMA request lines for DMA input and output operations. R0 is used as the DMA address pointer.

The DMA controller also provides a special "load mode", which allows loading of memory while the CLEAR and WAIT inputs of the processor are active. This allows a program to be loaded without the need for a ROM-based bootstrap loader. This was used by the COSMAC Elf microcomputer and its successors to load a program from toggle switches or a hexadecimal keypad.

Instruction timing

Clock cycle efficiency is poor in comparison to most 8-bit microprocessors. Eight clock cycles makes up one machine cycle. Most instructions take two machine cycles (16 clock cycles) to execute; the remaining instructions take three machine cycles (24 clock cycles). By comparison, the MOS Technology 6502 takes two to seven clock cycles to execute an instruction, and the Intel 8080 takes 4 to 18 clock cycles.

Support chips

Video

In early microcomputers the companion graphics Video Display Controller chip, CDP1861 for the NTSC video format, (CDP1864 variant for PAL), used the built-in DMA controller to display bitmapped graphics.

The 1861 chip could display 64 pixels horizontally and 128 pixels vertically, though by reloading the R0 register, the resolution could be reduced to 64×64 or 64×32 to use less memory or to make square pixels. Since the frame buffer was similar in size to the memory size, it was not unusual to display your program/data on the screen allowing you to watch the computer "think" (i.e. process its data).

Programs which ran amok and accidentally overwrote themselves could be spectacular. Although the faster version of 1802 could operate at 5 MHz (at 5 V; it was faster at 10 V), it was usually operated at 3.58 MHz/2 to suit the requirements of the 1861 chip which gave a speed of a little over 100,000 instructions per second.

Code samples

This code snippet tests ALU OPS, it is a diagnostic routine.

```
..  TEST ALU OPS
```

```
0000 90         GHI 0     .. SET UP R6
0001 B6         PHI 6
0002 F829       LDI DOIT  .. FOR INPUT OF OPCODE
0004 A6         PLO 6
0005 E0         SEX 0     .. (X=0 ALREADY)
0006 6400       OUT 4,00  .. ANNOUNCE US READY
0008 E6         SEX 6     .. NOW X=6
0009 3F09       BN4 *     .. WAIT FOR IT
000B 6C         INP 4     .. OK, GET IT
000C 64         OUT 4     .. AND ECHO TO DISPLAY
000D 370D       B4 *      .. WAIT FOR RELEASE
000F F860       LDI #60   .. NOW GET READY FOR
0011 A6         PLO 6     .. FIRST OPERAND
0012 E0         SEX 0     .. SAY SO
0013 6401       OUT 4,01
0015 3F15       BN4 *
0017 E6         SEX 6     .. TAKE IT IN AND ECHO
0018 6C         INP 4     .. (TO 0060)
0019 64         OUT 4     .. (ALSO INCREMENT R6)
001A 371A       B4 *
001C E0         SEX 0     .. DITTO SECOND OPERAND
001D 6402       OUT 4,02
001F E6         SEX 6
0020 3F20 LOOP: BN4 *     .. WAIT FOR IT
0022 6C         INP 4     .. GET IT (NOTE: X=6)
0023 64         OUT 4     .. ECHO IT
0024 3724       B4 *      .. WAIT FOR RELEASE
0026 26         DEC 6     .. BACK UP R6 TO 0060
0027 26         DEC 6
0028 46         LDA 6     .. GET 1ST OPERAND TO D
0029 C4   DOIT: NOP       .. DO OPERATION
002A C4         NOP       .. (SPARE)
002B 26         DEC 6     .. BACK TO 0060
002C 56         STR 6     .. OUTPUT RESULT
002D 64         OUT 4     .. (X=6 STILL)
002E 7A         REQ       .. TURN OFF Q
002F CA0020     LBNZ LOOP .. THEN IF ZERO,
0032 7B         SEQ       .. TURN IT ON AGAIN
0033 3020       BR LOOP   .. REPEAT IN ANY CASE
```

Chapter-15

Memory Disambiguation

Memory disambiguation is a set of techniques employed by high-performance out-of-order execution microprocessors that execute memory access instructions (loads and stores) out of program order. The mechanisms for performing memory disambiguation, implemented using digital logic inside the microprocessor core, detect true dependencies between memory operations at execution time and allow the processor to recover when a dependence has been violated. They also eliminate spurious memory dependencies and allow for greater instruction-level parallelism by allowing safe out-of-order execution of loads and stores.

Background

Dependencies

When attempting to execute instructions out of order, a microprocessor must respect true dependencies between instructions. For example, consider a simple true dependence:

```
1: add $1, $2, $3      # R1 <= R2 + R3
2: add $5, $1, $4      # R5 <= R1 + R4 (dependent on 1)
```

In this example, the `add` instruction on line 2 is dependent on the `add` instruction on line 1 because the register R1 is a source operand of the addition operation on line 2. The `add` on line 2 cannot execute until the `add` on line 1 completes. In this case, the dependence is **static** and easily determined by a microprocessor, because the sources and destinations are registers. The destination register of the `add` instruction on line 1 (`R1`) is part of the instruction encoding, and so can be determined by the microprocessor early on, during the decode stage of the pipeline. Similarly, the source registers of the `add` instruction on line 2 (`R1` and `R4`) are also encoded into the instruction itself and are determined in decode. To respect this true dependence, the microprocessor's scheduler logic will issue these instructions in the correct order (instruction 1 first, followed by instruction 2) so that the results of 1 are available when instruction 2 needs them.

Complications arise when the dependence is not statically determinable. Such non-static dependencies arise with memory instructions (loads and stores) because the location of

the operand may be indirectly specified as a register operand rather than directly specified in the instruction encoding itself.

```
1: store ($2), $1       # Mem[R2] <= R1
2: load  $3, ($4)       # R3 <= Mem[R4] (possibly dependent on 1)
```

Here, the store instruction writes a value to the memory location specified by the value in R2, and the load instruction reads the value at the memory location specified by the value in R4. The microprocessor cannot statically determine, prior to execution, if the memory locations specified in these two instructions are different, or are the same location, because the locations depend on the values in R2 and R4. If the locations are different, the instructions are independent and can be successfully executed out of order. However, if the locations are the same, then the load instruction is dependent on the store to produce its value. This is known as an **ambiguous dependence**.

Out-of-order execution and memory access operations

Executing loads and stores out of order can produce incorrect results if a dependent load/store pair was executed out of order. Consider the following code snippet, given in MIPS assembly:

```
1: div $27, $20
2: sw  ($30), $27
3: lw  $08, ($31)
4: sw  ($30), $26
5: lw  $09, ($31)
```

Assume that the scheduling logic will issue an instruction to the execution unit when all of its register operands are ready. Further assume that registers `$30` and `$31` are ready: the values in `$30` and `$31` were computed a long time ago and have not changed. However, assume `$27` is not ready: its value is still in the process of being computed by the `div` (integer divide) instruction. Finally, assume that registers `$30` and `$31` hold the same value, and thus all the loads and stores in the snippet access the same memory word.

In this situation, the `sw ($30), $27` instruction on line 2 is not ready to execute, but the `lw $08, ($31)` instruction on line 3 is ready. If the processor allows the `lw` instruction to execute before the `sw`, the load will read an old value from the memory system; however, it should have read the value that was just written there by the `sw`. The load and store executed out of program order, but there was a memory dependence between them that was violated.

Similarly, assume that register `$26` *is* ready. The `sw ($30), $26` instruction on line 4 is also ready to execute, and it may execute before the preceding `lw $08, ($31)` on line 3. If this occurs, the `lw $08, ($31)` instruction will read the *wrong* value from the memory system, since a later store instruction wrote its value there before the load executed.

Characterization of memory dependencies

Memory dependencies come in three flavors:

- **Read-After-Write** (RAW) dependencies: Also known as true dependencies, RAW dependencies arise when a load operation reads a value from memory that was produced by the most recent preceding store operation to that same address.
- **Write-After-Read** (WAR) dependencies: Also known as anti dependencies, WAR dependencies arise when a store operation writes a value to memory that a preceding load reads.
- **Write-After-Write** (WAW) dependencies: Also known as output dependencies, WAW dependencies arise when two store operations write values to the same memory address.

The three dependencies are shown in the preceding code segment (reproduced for clarity):

```
1: div $27, $20
2: sw  ($30), $27
3: lw  $08, ($31)
4: sw  ($30), $26
5: lw  $09, ($31)
```

- The `lw $08, ($31)` instruction on line 3 has a RAW dependence on the `sw ($30), $27` instruction on line 2, and the `lw $09, ($31)` instruction on line 5 has a RAW dependence on the `sw ($30), $26` instruction on line 4. Both load instructions read the memory address that the preceding stores wrote. The stores were the most recent producers to that memory address, and the loads are reading that memory address's value.
- The `sw ($30), $26` instruction on line 4 has a WAR dependence on the `lw $08, ($31)` instruction on line 3 since it writes the memory address that the preceding load reads from.
- The `sw ($30), $26` instruction on line 4 has a WAW dependence on the `sw ($30), $27` instruction on line 2 since both stores write to the same memory address.

Memory disambiguation mechanisms

Modern microprocessors use the following mechanisms, implemented in hardware, to resolve ambiguous dependences and recover when a dependence was violated.

Avoiding WAR and WAW dependencies

Values from store instructions are not committed to the memory system (in modern microprocessors, CPU cache) when they execute. Instead, the store instructions, including the memory address and store data, are buffered in a **store queue** until they reach the retirement point. When a store retires, it *then* writes its value to the memory

system. This avoids the WAR and WAW dependence problems shown in the code snippet above where an earlier load receives an incorrect value from the memory system because a later store was allowed to execute before the earlier load.

Additionally, buffering stores until retirement allows processors to speculatively execute store instructions that follow an instruction that may produce an exception (such as a load of a bad address, divide by zero, etc.) or a conditional branch instruction whose direction (taken or not taken) is not yet known. If the exception-producing instruction has not executed or the branch direction was predicted incorrectly, the processor will have fetched and executed instructions on a "wrong path." These instructions should not have been executed at all; the exception condition should have occurred before any of the speculative instructions executed, or the branch should have gone the other direction and caused different instructions to be fetched and executed. The processor must "throw away" any results from the bad-path, speculatively-executed instructions when it discovers the exception or branch misprediction. The complication for stores is that any stores on the bad or mispredicted path should not have committed their values to the memory system; if the stores had committed their values, it would be impossible to "throw away" the commit, and the memory state of the machine would be corrupted by data from a store instruction that should not have executed.

Thus, without store buffering, stores cannot execute until all previous possibly-exception-causing instructions have executed (and not caused an exception) and all previous branch directions are known. Forcing stores to wait until branch directions and exceptions are known significantly reduces the out-of-order aggressiveness and limits ILP (Instruction level parallelism) and performance. With store buffering, stores can execute ahead of exception-causing or unresolved branch instructions, buffering their data in the store queue but not committing their values until retirement. This prevents stores on mispredicted or bad paths from committing their values to the memory system while still offering the increased ILP and performance from full out-of-order execution of stores.

Store to load forwarding

Buffering stores until retirement avoids WAW and WAR dependencies but introduces a new issue. Consider the following scenario: a store executes and buffers its address and data in the store queue. A few instructions later, a load executes that reads from the same memory address to which the store just wrote. If the load reads its data from the memory system, it will read an old value that would have been overwritten by the preceding store. The data obtained by the load will be incorrect.

To solve this problem, processors employ a technique called **store-to-load forwarding** using the store queue. In addition to buffering stores until retirement, the store queue serves a second purpose: forwarding data from completed but not-yet-retired ("in-flight") stores to later loads. Rather than a simple FIFO queue, the store queue is really a Content-Addressable Memory (CAM) searched using the memory address. When a load executes, it searches the store queue for in-flight stores to the same address that are logically earlier in program order. If a matching store exists, the load obtains its data value from that store

instead of the memory system. If there is no matching store, the load accesses the memory system as usual; any preceding, matching stores must have already retired and committed their values. This technique allows loads to obtain correct data if their producer store has completed but not yet retired.

Multiple stores to the load's memory address may be present in the store queue. To handle this case, the store queue is priority encoded to select the *latest* store that is logically earlier than the load in program order. The determination of which store is "latest" can be achieved by attaching some sort of timestamp to the instructions as they are fetched and decoded, or alternatively by knowing the relative position (slot) of the load with respect to the oldest and newest stores within the store queue.

RAW dependence violations

Detecting RAW dependence violations

Modern out-of-order CPUs can use a number of techniques to detect a RAW dependence violation, but all techniques require tracking in-flight loads from execution until retirement. When a load executes, it accesses the memory system and/or store queue to obtain its data value, and then its address and data are buffered in a **load queue** until retirement. The load queue is similar in structure and function to the store queue, and in fact in some processors may be combined with the store queue in a single structure called a **load-store queue**, or **LSQ**. The following techniques are used or have been proposed to detect RAW dependence violations:

Load queue CAM search

With this technique, the load queue, like the store queue, is a CAM searched using the memory access address, and keeps track of all in-flight loads. When a store executes, it searches the load queue for completed loads from the same address that are logically later in program order. If such a matching load exists, it must have executed before the store and thus read an incorrect, old value from the memory system/store queue. Any instructions that used the load's value have also used bad data. To recover if such a violation is detected, the load is marked as "violated" in the retirement buffer. The store remains in the store queue and retirement buffer and retires normally, committing its value to the memory system when it retires. However, when the violated load reaches the retirement point, the processor flushes the pipeline and restarts execution from the load instruction. At this point, all previous stores have committed their values to the memory system. The load instruction will now read the correct value from the memory system, and any dependent instructions will re-execute using the correct value.

This technique requires an associative search of the load queue on every store execution, which consumes circuit power and can prove to be a difficult timing path for large load queues. However, it does not require any additional memory (cache) ports or create resource conflicts with other loads or stores that are executing.

Disambiguation At Retirement

With this technique, load instructions that have executed out-of-order are re-executed (they access the memory system and read the value from their address a second time) when they reach the retirement point. Since the load is now the retiring instruction, it has no dependencies on any instruction still in-flight; all stores ahead of it have committed their values to the memory system, and so any value read from the memory system is guaranteed to be correct. The value read from memory at re-execution time is compared to the value obtained when the load first executed. If the values are the same, the original value was correct and no violation has occurred. If the re-execution value differs from the original value, a RAW violation has occurred and the pipeline must be flushed because instructions dependent on the load have used an incorrect value.

This technique is conceptually simpler than the load queue search, and it eliminates a second CAM and its power-hungry search (the load queue can now be a simple FIFO queue). Since the load must re-access the memory system just before retirement, the access must be very fast, so this scheme relies on a fast cache. No matter how fast the cache is, however, the second memory system access for every out-of-order load instruction does increase instruction retirement latency and increases the total number of cache accesses that must be performed by the processor. The additional retire-time cache access can be satisfied by re-using an existing cache port; however, this creates port resource contention with other loads and stores in the processor trying to execute, and thus may cause a decrease in performance. Alternatively, an additional cache port can be added just for load disambiguation, but this increases the complexity, power, and area of the cache. Some recent work (Roth 2005) has shown ways to filter many loads from re-executing if it is known that no RAW dependence violation could have occurred; such a technique would help or eliminate such latency and resource contention.

A minor benefit of this scheme (compared to a load-queue search) is that it will not flag a RAW dependence violation and trigger a pipeline flush if a store that would have caused a RAW dependence violation (the store's address matches an in-flight load's address) has a data value that matches the data value already in the cache. In the load-queue search scheme, an additional data comparison would need to be added to the load-queue search hardware to prevent such a pipeline flush.

Avoiding RAW dependence violations

CPUs that fully support out-of-order execution of loads and stores must be able to detect RAW dependence violations when they occur. However, many CPUs avoid this problem by forcing all loads and stores to execute in-order, or by supporting only a limited form of out-of-order load/store execution. This approach offers lower performance compared to supporting full out-of-order load/store execution, but it can significantly reduce the complexity of the execution core and caches.

The first option, making loads and stores go in-order, avoids RAW dependences because there is no possibility of a load executing before its producer store and obtaining incorrect

data. Another possibility is to effectively break loads and stores into two operations: address generation and cache access. With these two separate but linked operations, the CPU can allow loads and stores to access the memory system only once all previous loads and stores have had their address generated and buffered in the LSQ. After address generation, there are no longer any ambiguous dependencies since all addresses are known, and so dependent loads will not be executed until their corresponding stores complete. This scheme still allows for some "out-of-orderness"—the address generation operations for any in-flight loads and stores can execute out-of-order, and once addresses have been generated, the cache accesses for each load or store can happen in any order that respects the (now known) true dependences.

Additional Issues

Memory dependence prediction

Processors that fully support out-of-order load/store execution can use an additional, related technique, called memory dependence prediction, to attempt to predict true dependences between loads and stores *before* their addresses are known. Using this technique, the processor can prevent loads that are predicted to be dependent on an in-flight store from executing before that store completes, avoiding a RAW dependence violation and thus avoiding the pipeline flush and the performance penalty that is incurred.

Chapter-16

LEON and EnCore Processor

LEON

LEON is a 32-bit CPU microprocessor core, based on the SPARC-V8 RISC architecture and instruction set. It was originally designed by the European Space Research and Technology Centre, part of the European Space Agency, and after that by Gaisler Research. It is described in synthesizable VHDL. LEON has a dual license model: A LGPL/GPL FLOSS license that can be used without licensing fee, or a proprietary license that can be purchased for integration in a proprietary product. . The core is configurable through VHDL generics, and is used in system-on-a-chip (SOC) designs both in research and commercial settings.

History

The LEON project was started by the European Space Agency (ESA) in late 1997 to study and develop a high-performance processor to be used in European space projects. The objectives for the project were to provide an open, portable and non-proprietary processor design, capable to meet future requirements for performance, software compatibility and low system cost. Another objective was to be able to manufacture in a Single Event Upset (SEU) sensitive semiconductor process. To maintain correct operation in the presence of SEUs, extensive error detection and error handling functions were needed. The goals have been to detect and tolerate one error in any register without software intervention, and to suppress effects from Single Event Transient (SET) errors in combinational logic.

The LEON family includes the first LEON1 VHSIC Hardware Description Language (VHDL) design that was used in the LEONExpress test chip developed in 0.25 μm technology to prove the fault-tolerance concept. The second LEON2 VHDL design was used in the processor device AT697 from Atmel (F) and various system-on-chip devices. These two LEON implementations were developed by ESA. Gaisler Research, now Aeroflex Gaisler, developed the third LEON3 design and has announced the availability of the fourth generation LEON, the LEON4 processor.

LEON processor models and distributions

A LEON processor can be implemented in programmable logic such as an FPGA or manufactured into an ASIC. This section and the subsequent subsections focus on the LEON processors as soft IP cores and summarise the main features of each processor version and the infrastructure with which the processor is packaged, referred to as a LEON *distribution*.

All processors in the LEON series are based on the SPARC-V8 RISC architecture. LEON2(-FT) has a five-stage pipeline while later versions have a seven-stage pipeline. LEON2 and LEON2-FT are distributed as a system-on-chip design that can be modified using a graphical configuration tool. While the LEON2(-FT) design can be extended and re-used in other designs, its structure does not emphasise re-using parts of the design as building blocks or enable designers to easily incorporate new IP cores in the design.

The standard LEON2(-FT) distribution includes the following support cores.:

- Interrupt controller
- Debug support unit with trace buffer
- Two 24-bit timers
- Two UARTs
- 16-bit I/O port
- Memory controller.

The LEON3, LEON3FT and LEON4 cores are typically used together with the GRLIB IP Library. While the LEON2 distributions contain one design that can be used on several target technologies, GRLIB contains several template designs, both for FPGA development boards and for ASIC targets that can be modified using a graphical configuration tool similar to the one in the LEON2 distribution. The LEON/GRLIB package contains a larger number of cores compared to the LEON2 distributions and also include a plug and play extension to the on-chip AMBA bus. IP cores available in GRLIB include:

- 32-bit SDRAM controller
- 32-bit PCI bridge with DMA
- 10/100/1000 Mbit Ethernet MAC
- 8/16/32-bit PROM and SRAM controller
- 16/32/64-bit DDR/DDR2 controllers
- USB 2.0 host and device controllers
- CAN controller
- TAP controller
- SPI, I2C, ATA controllers
- UART with FIFO
- Modular timer unit
- Interrupt controller
- General purpose I/O port

Terminology

The term LEON2/LEON2-FT often refer to the LEON2 system-on-chip design, which is the LEON2 processor core together with the standard set of peripherals available in the LEON2(-FT) distribution. Later processors in the LEON series are used in a wide range of designs and are therefore not as tightly coupled with a standard set of peripherals. With LEON3 and LEON4 the name typically refers to only the processor core, while LEON/GRLIB is used to refer to the complete system-on-chip design.

LEON2 processor core

LEON2 has the following characteristics:

- The GNU LGPL allows a high degree of freedom of intervention on the freely-available source code.
- Configurability is a key feature of the project, and is achieved through the usage of VHDL generics.
- It offers all basic functions of a pipelined in-order processor.
- It is a fairly-sized VHDL project (about 90 files, for the complete LEON2 distribution, including peripheral IP cores)

LEON2-FT processor core

The LEON2-FT processor is the single event upset tolerant version of the LEON2 processor. Flip-flops are protected by triple modular redundancy and all internal and external memories are protected by EDAC or parity bits. Special licence restrictions apply to this IP (distributed by the European Space Agency).

LEON3 processor core

The LEON3 is a synthesisable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. The model is highly configurable, and particularly suitable for system-on-a-chip (SOC) designs. The full source code is available under the GNU GPL license, allowing use for any purpose without licensing fee. LEON3 is also available under a proprietary license, allowing it to be used in proprietary applications.

There are several differences between the two LEON2 processor models and the LEON3. LEON3 includes SMP support and a seven-stage pipeline, while LEON2 does not support SMP and has a five-stage pipeline.

LEON3-FT processor core

The LEON3FT is a fault-tolerant version of the standard LEON3 SPARC V8 Processor. It has been designed for operation in the harsh space environment, and includes functionality to detect and correct single event upset (SEU) errors in all on-chip RAM

memories. The LEON3FT processor support most of the functionality in the standard LEON3 processor, and adds the following features:

- Register file SEU error-correction of up to 4 errors per 32-bit word
- Cache memory error-correction of up to 4 errors per tag or 32-bit word
- Autonomous and software transparent error handling
- No timing impact due to error detection or correction

The following features of the standard LEON3 processor are not supported by LEON3FT

- Local scratchpad RAM (neither for instruction nor for data)
- Cache locking
- LRR (least recently replaced) cache replacement algorithm

The LEON3FT core is distributed together with a special FT version of the GRLIP IP library. Only netlist distribution is possible.

A FPGA implementation called LEON3FT-RTAX is proposed for critical space applications.

LEON4 processor core

In January 2010, the fourth version of the LEON processor was released. This release has the following new features:

- Static branch prediction added to pipeline
- Optional level-2 cache
- 64-bit or 128-bit path to AMBA AHB interface
- Higher performance possible (claimed by manufacturer: 1.7 DMIPS/MHz as opposed to 1.4 DMIPS/MHz of LEON3)

Real-time OS support

The SPARC-V8 architecture of the LEON core is not well-supported among Real-time operating systems as can be seen in the list of real-time operating systems. The Real-time operating systems that support the LEON core, are currently RTLinux, PikeOS, eCos, RTEMS, Nucleus, ThreadX, VxWorks (as per a port by Gaisler Research), LynxOS (also per a port by Gaisler Research) and POK (a free ARINC653 implementation released under the BSD licence).

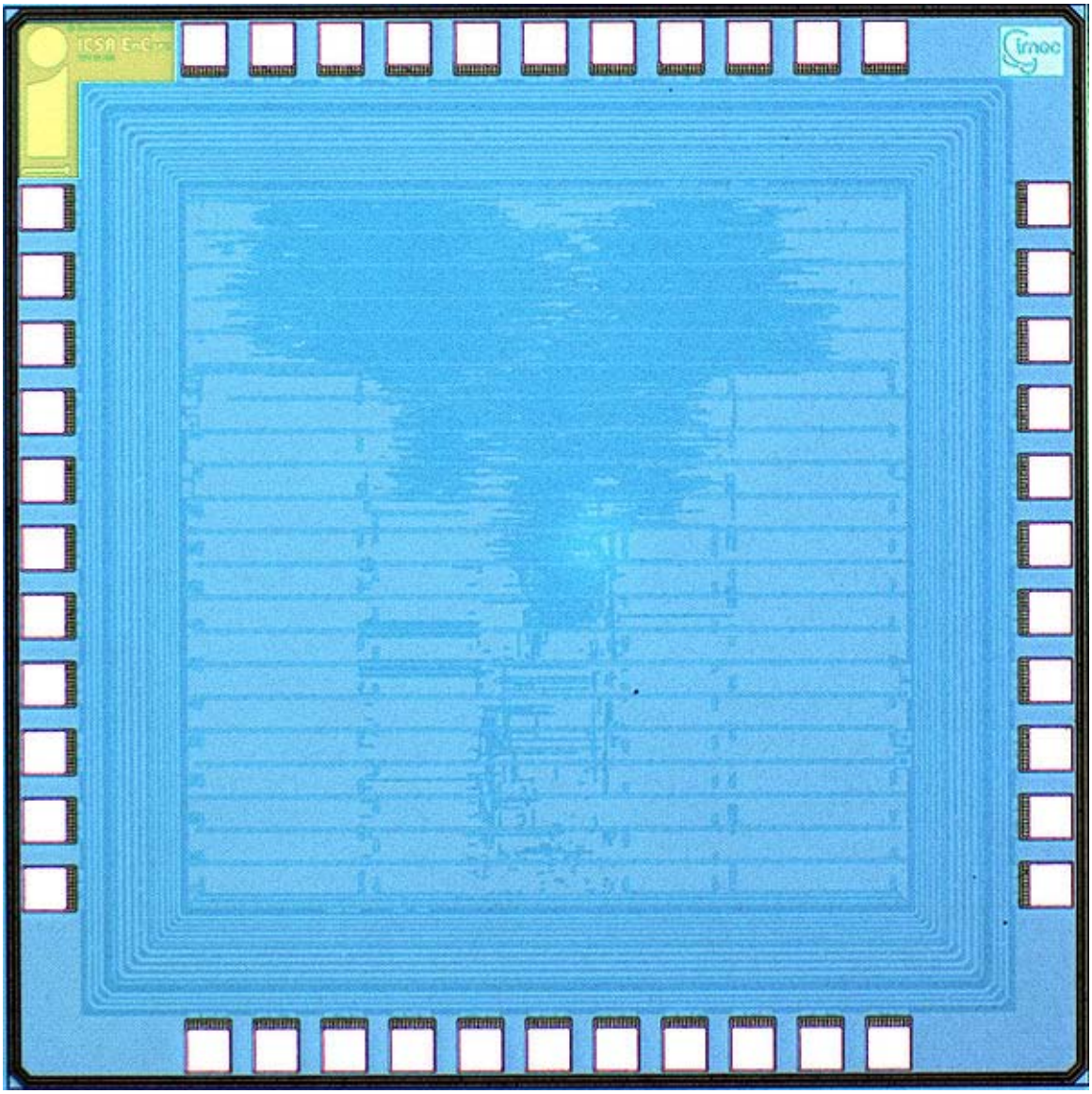
EnCore Processor

The **EnCore** microprocessor family is a *configurable* and *extendable* implementation of a compact 32-bit RISC instruction set architecture - developed by the PASTA Research Group at the University of Edinburgh School of Informatics. The following are key features of the EnCore microprocessor family:

- 5 stage pipeline
- highest operating frequency in its class
- lowest possible dynamic energy consumption - 99% of flip-flops automatically clock-gated using typical synthesis tools
- most non-memory operations achieving single-cycle latency, and no more than one load-delay slot
- easy configurability of cache architectures
- compact baseline instruction set architecture (ISA), including freely-mixed 16-bit and 32-bit encodings for maximum code density
- no overhead for switching between 16- and 32-bit instruction encodings

All of the EnCore test chips are named after hills in Edinburgh; Calton, being the smallest, is the first of these.

EnCore Calton

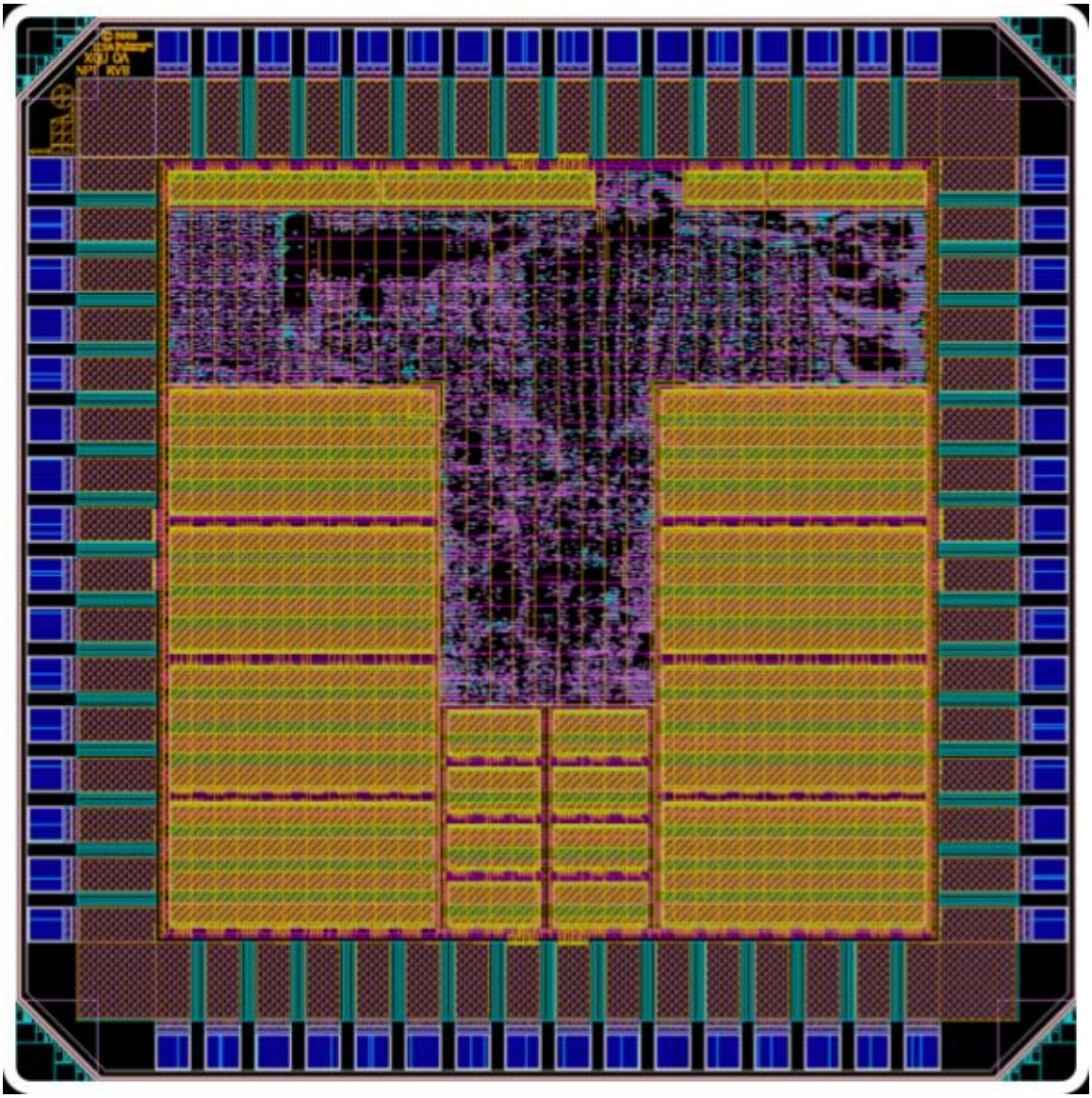


Photomicrograph picture of EnCore Calton

The first silicon implementation of the EnCore processor is a test-chip code-named Calton, fabricated in a generic 130nm CMOS process using a standard ASIC flow.

- **130nm** implementation of EnCore processor in baseline configuration extended with barrel shifter, multiplier, and a full set of 32 general purpose registers.
- Contains bus interface and system control functions, in addition to the processor.
- Implemented with 8KB direct-mapped instruction- and data-cache.
- Complete system-on-chip occupies 1 mm^2 of silicon at 75% utilization.
- Chip-level power consumption is 25 mW at **250** MHz.
- First silicon samples operate above a frequency of **375** MHz at typical voltage and temperature.

EnCore Castle



Chip Layout of EnCore Castle

The second silicon implementation of an extended EnCore processor is a test-chip codenamed Castle, fabricated in a generic 90nm CMOS process. All of the EnCore test chips are named after hills in Edinburgh; Castle is named after the rock on which Edinburgh Castle is built.

The Castle chip contains an extended version of the EnCore processor, together with a 32KB 4-way set-associative Instruction Cache, and a 32KB 4-way set-associative Data Cache. It is embedded within a system-on-chip (SoC) design that provides a generic 32-bit memory interface, as well as interrupt, clocks and reset signals.

- **90nm** implementation is based on a generic free foundry libraries, and a stack of 9 metal layers.
- Complete design occupies 2.25 sq.mm on a 1.875 x 1.875 mm die. This includes the baseline CPU, the reconfigurable Configurable Flow Accelerator (CFA) extension logic, two 32KB caches, and the off-chip interfaces.
- Designed to operate on a core voltage of 0.9V to 1.1V, with 2.5V LVCMOS I/O signals.
- Packaged in a 68-pin Ceramic LCC.
- First silicon samples operate at **600MHz**.
- Chip-level power consumption is 70mW at 600 MHz, under typical conditions.
- Complete design flow, from RTL to GDSII, was performed by the PASTA team. This was based on an in-house developed design flow using Synopsys Design Compiler for topological synthesis, and IC Compiler for automated place-and-route.
- Over 97% of all flip-flops in the design were automatically clock-gated during logic synthesis.
- LVS and DRC checks were performed using Calibre, from Mentor Graphics.

Chapter-17

Geode (Processor)

AMD Geode LX 800 (500MHz, 0.9W) processor.

Alix.1C Mini-ITX embedded board with AMD Geode LX 800 together with Compact Flash, miniPCI and PCI slots, 44-pin IDE interface and 256MB RAM.

Geode is a series of x86-compatible system-on-a-chip microprocessors and I/O companions produced by AMD, targeted at the embedded computing market.

The series was originally launched by National Semiconductor as the **Geode** family in 1999. The original Geode processor core itself is derived from the Cyrix MediaGX platform, which was acquired in National's merger with Cyrix in 1997. AMD bought the Geode business from National in August 2003 to augment its existing line of embedded x86 processor products. AMD expanded the Geode series to two classes of processor: the MediaGX-derived Geode GX and LX, and the modern Athlon-derived Geode NX.

Geode processors are optimized for low power consumption and low cost while still remaining compatible with software written for the x86 platform. The MediaGX-derived

processors lack modern features such as SSE and a large on-die L1 cache but these are offered on the more recent Athlon-derived Geode NX. Geode processors tightly integrate some of the functions normally provided by a separate chipset, such as the northbridge. Whilst the processor family is best suited for thin client, set top box and embedded computing applications, it can be found in unusual applications such as the Nao robot and the Win Enterprise IP-PBX

The One Laptop per Child project originally used the GX series Geode processor in the OLPC XO, but it has since moved to the Geode LX. The Linutop (rebranded FIC Mini PC Ion A603) is also based on the Geode LX. 3Com Audrey was powered by a 200 MHz Geode GX1.

The SCxxxx range of Geode devices are a single-chip version, comparable to the SiS 552, VIA CoreFusion or Intel's Tolapai, which integrate the CPU, memory controller, graphics and I/O devices into one package. Single processor boards based on these processors are manufactured by Artec Group, PC Engines (WRAP), Soekris, and Win Enterprises.

These processors are named after geodes.

National Semiconductor Geode

Geode GXm

Cyrix MediaGXm clone. Returns "CyrixInstead" on CPUID.

- MediaGX-derived core
- 0.35 µm four layer metal CMOS
- MMX instructions
- 3.3 V I/O, 2.9 V core
- 16 Kb write-back unified L1 cache
- PCI controller
- 64-bit SDRAM memory
- CS5530 companion chip (implements sound and video functions)
- VSA architecture
- 1280x1024x8 or 1024x768x16 display

Geode GXLV

Geode GXLV.

- MediaGX-derived core
- 0.25 µm four layer metal CMOS
- 3.3 V I/O
- 2.2 V, 2.5 V, 2.9 V core
- 16 kb write-back unified L1 cache
- Fully static design
- 1.0 W @2.2 V/166 MHz, 2.5 W @2.9 V/**266** MHz

Geode GX1

National Semiconductor Geode GX1, 233 MHz

- MediaGX-derived core
- 0.18 µm CMOS
- 200 - 333 MHz
- 1.6 - 2.2 V core
- 16 kB (16 KiB) L1 cache
- 0.8 W - 1.2 W typical
- SDRAM memory 111 MHz
- CS5530A companion chip
- 85 Hz VGA refresh rate

National Semiconductor/AMD SC1100 is based on the Cyrix GX1 core and the CS5530 support chip.

Geode GX2

Announced by National Semiconductor Corporation October, 2001 at Microprocessor Forum. First demonstration at COMPUTEX Taiwan, June, 2002.

- 0.15 µm process technology
- MMX and 3DNow! instructions
- 16 kB Instruction and 16 kB Data caches
- GeodeLink architecture, 6 GB/s on-chip bandwidth, up to 2 GB/s memory bandwidth
- Integrated 64-bit PC133 SDRAM and DDR266 controller
- Clockrate: 266, 333 and 400 MHz
- 3 PCI masters supported
- 1600x1200x24 bit display with video scaling
- CRT DACs and an UMA DSTN/TFT controller.
- Geode CS5535 companion chip

AMD Geode

In 2002, AMD introduced the **Geode GX** series, which was a re-branding of the National Semiconductor GX2. This was quickly followed by the **Geode LX**, running up to 667 MHz. LX brought many improvements, such as higher speed DDR, a re-designed instruction pipe, and a more powerful display controller. The upgrade from the CS5535 I/O Companion to the CS5536 brought higher speed USB.

Geode GX and LX processors are typically found in devices such as thin clients and industrial control systems. However, they have come under competitive pressure from VIA on the x86 side, and ARM and XScale taking much of the low-end business.

Because of the relative performance, albeit higher PPW, of the GX and LX core design, AMD introduced the **Geode NX**, which is an embedded version of the highly-successful Athlon processor, K7. Geode NX uses the **Thoroughbred** core and is quite similar to the Athlon XP-M that use this core. The Geode NX includes 256KB of level 2 cache, and runs fanless at up to 1 GHz in the NX1500@6W version. The NX2001 part runs at 1.8 GHz, the NX1750 part runs at 1.4 GHz, and the NX1250 runs at 667 MHz.

The Geode NX, with its strong FPU, is particularly suited for embedded devices with graphical performance requirements, such as information kiosks and casino gaming machines, such as video slots.

However, it was reported that the specific design team for Geode processors in Longmont, Colorado, has been closed, and 75 employees are being relocated to the new

development facility in Fort Collins, Colorado. It is expected that the Geode line of processors will be updated less frequently due to the closure of the Geode design center.

In 2009, comments by AMD indicated that there are no plans for any future micro architecture upgrades to the processor and that there will be no successor; however, the processors will still be available with the planned availability of the Geode LX extending through 2015.

Geode GX

1. Geode GX 466@0.9 W: clock speed: 333 MHz
2. Geode GX 500@1.0 W: clock speed: 366 MHz
3. Geode GX 533@1.1 W: clock speed: 400 MHz

Geode LX

AMD Geode LX 800 (500MHz) CPU.

1. LX 600@0.7 W: clock speed: 366 MHz, with power consumption: 1.2 watts. (TDP 2.8 W)
2. LX 700@0.8 W: clock speed: 433 MHz, with power consumption: 1.3 watts. (TDP 3.1 W)
3. LX 800@0.9 W: clock speed: 500 MHz, with power consumption: 1.8 watts. (TDP 3.6 W)
4. LX 900@1.5 W: clock speed: 600 MHz, with power consumption: 2.6 watts. (TDP 5.1 W)

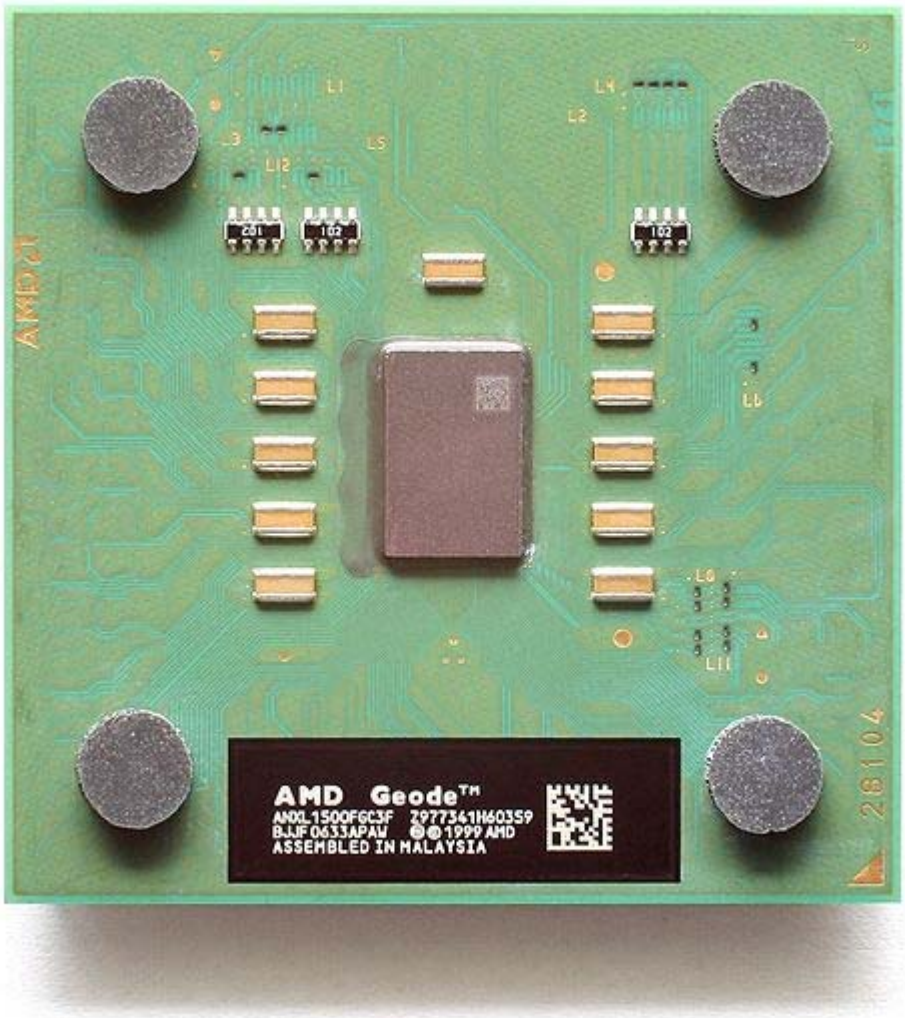
Features:

- Low power.
- Full x86 compatibility.
- Processor functional blocks:
    - CPU Core
    - GeodeLink Control Processor
    - GeodeLink Interface Units
    - GeodeLink Memory Controller
    - Graphics Processor
    - Display Controller
    - Video Processor
    - Video Input Port
    - GeodeLink PCI Bridge
    - Security Block
        - 128-Bit Advanced Encryption Standard (AES) - (CBC/ECB)
        - True Random Number Generator

Specification:

- Processor frequency up to 600 MHz (LX900), 500 MHz (LX800) and 433 MHz (LX700).
- Power management: ACPI, lower power, wakeup on SMI/INTR.
- 64K Instruction / 64K Data L1 cache and 128K L2 cache
- Split Instruction/Data cache/TLB.
- DDR Memory 400 MHz (LX 800), 333 MHz (LX 700)
- Integrated FPU with MMX and 3DNow!
- 9 GB/s internal GeodeLink Interface Unit (GLIU)
- Simultaneous, high-res CRT and TFT (High and standard definition). VESA 1.1 and 2.0 VIP/VDA support
- Manufactured at a 0.13 micrometre process
- 481-terminal PBGA (Plastic Ball grid array)
- GeodeLink active hardware power management
- Compatible with Socket 7 motherboards

Geode NX



AMD Geode NX 1500.

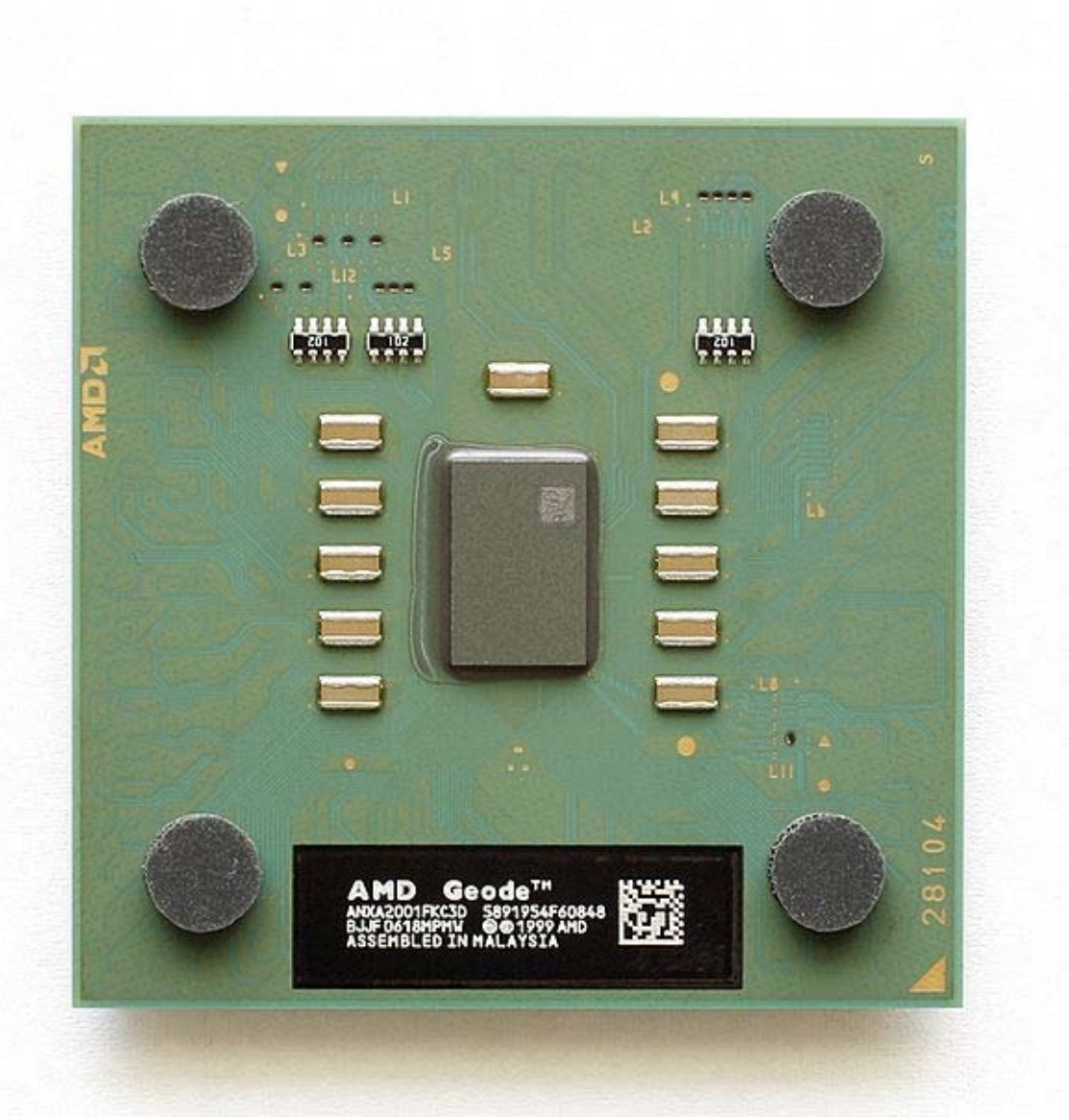
1. NX 1250@6W: Clock speed: 667 MHz, average power consumption 6 watts, TDP 9 watts (1.0 volts core operating voltage).
2. NX 1500@6W: Clock speed: 1 GHz, average power consumption 6 watts, TDP 9 watts (1.1 volts core operating voltage).
3. NX 1750@14W: Clock speed: 1.4 GHz, average power consumption 14 watts, TDP 25 watts (1.25 volts core operating voltage).

Features:

- 7th generation core (based on Mobile Athlon XP-M).

- Power management: AMD PowerNow!, ACPI 1.0b and ACPI 2.0.
- 128 KB L1 cache.
- 256 KB L2 cache with hardware data prefetch
- 133 MHz Front Side Bus (FSB)
- 3DNow!, MMX and SSE instruction sets
- 0.13 µm (130 nm) fabrication process
- Pin compatibility between all NX family processors.
- OS support: Linux, Windows CE, MS Windows XP.
- Compatible with Socket A motherboards

Geode NX 2001



Geode NX 2001.

In 2007, there was a *Geode NX 2001* model on sale, which in fact was a relabelled Athlon XP 2200+ Thoroughbred. The processors, with part numbers AANXA2001FKC3G or ANXA2001FKC3D, their specifications are 1.8 GHz clock speed, and 1.65 volt core operating voltage, the power consumption is not specified. There are no official references to this processor except officials explaining that the batch of CPUs were "being shipped to specific customers", though it is clear it has no relation with the other Geode NX CPUs other than sharing the same CPU socket (Socket A).

Chipsets for Geode

1. NSC Geode CS5530A Southbridge for Geode GX1.
2. NSC/AMD Geode CS5535 Southbridge for Geode GX(2) and Geode LX (USB 1.1). Integrates four USB ports, one ATA-66 UDMA controller, one Infrared communication port, one AC'97 controller, one SMBUS controller, one LPC port, as well as GPIO, Power Management, and legacy functional blocks.
3. AMD Geode CS5536 Southbridge for Geode GX and Geode LX (USB 2.0). Power consumption: 1.9 W (433 MHz) and 2.4 W (500 MHz). This chipset is also used on PowerPC board (Amy'05).
4. Geode NX processors are "100 percent socket and chipset compatible" with AMD's Socket A Athlon XP processors: SIS741CX Northbridge and SIS 964 Southbridge, VIA KM400 Northbridge and VIA VT8235 Southbridge, VIA KM400A Northbridge and VIA VT8237R Southbridge and other Socket A chipsets.