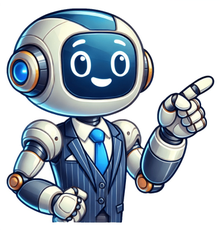
Continue



Double digit addition worksheets

Arithmetic operation For other uses, see Addition (disambiguation). "Add" redirects here. For other uses, see ADD (disambiguation). 3 + 2 = 5 with apples, a popular choice in textbooks[1] Addition (usually signified by the plus symbol, +) is one of the four basic operations of arithmetic, the other three being subtraction, multiplication, and division. The addition of two whole numbers results in the total or sum of those values combined. For example, the adjacent image shows two columns of apples, one with three apples and the other with two apples, totaling to five apples. This observation is expressed as "3 + 2 = 5", which is read as "three plus two equals five". Besides counting items, addition can also be defined and executed without referring to concrete objects, using abstractions called numbers instead, such as integers, real numbers, and complex numbers. Addition belongs to arithmetic, a branch of mathematics. In algebra, another area of mathematics, addition can also be performed on abstract objects such as vectors, matrices, subspaces, and subgroups. Addition has several important properties. It is commutative, meaning that the order of the numbers being added does not matter, so 3 + 2 = 2 + 3, and it is associative, meaning that when one adds more than two numbers, the order in which addition is performed does not matter. Repeated addition of 1 is the same as counting (see Successor function). Addition of 0 does not change a number. Addition also obeys rules concerning related operations such as subtraction and multiplication. Performing addition is one of the simplest numerical tasks to perform. Addition of very small numbers is accessible to toddlers; the most basic task, 1 + 1, can be performed by infants as young as five months, and even some members of other animal species. In primary education, students are taught to add numbers in the decimal system, beginning with single digits and progressively tackling more difficult problems. Mechanical aids range from the ancient abacus to the modern computer, where research on the most efficient implementations of addition continues to this day. Arithmetic operationsvte Addition (+) term + term summand + summand addend + addend augend + addend } = {\displaystyle \scriptstyle \left.{\begin{matrix}\scriptstyle {\text{term}}\,+\,{\text{term}}\\\scriptstyle {\text{summand}}\,+\,{\text{summand}}\\\scriptstyle {\text{addend}}\,+\,{\text{addend}}\\\scriptstyle {\text{augend}}\,+\,{\text{addend}}\end{matrix}}\right\}\,=\,} sum {\displaystyle \scriptstyle {\text{sum}}} Subtraction (−) term − term minuend − subtrahend } = {\displaystyle \scriptstyle \left.{\begin{matrix}\scriptstyle {\text{term}}\,-\,{\text{term}}\\\scriptstyle {\text{minuend}}\,-\,{\text{subtrahend}}\end{matrix}}\right\}\,=\,} difference {\displaystyle \scriptstyle {\text{difference}}} Multiplication (×) factor × factor multiplier × multiplicand } = {\displaystyle \scriptstyle \left.{\begin{matrix}\scriptstyle {\text{factor}}\,\times \,{\text{factor}}\\\scriptstyle {\text{multiplier}}\,\times \,{\text{multiplicand}}\end{matrix}}\right\}\,=\,} product {\displaystyle \scriptstyle {\text{product}}} Division (÷) dividend divisor numerator denominator } = {\displaystyle \scriptstyle \left.{\begin{matrix}\scriptstyle {\frac {\scriptstyle {\text{dividend}}}{\scriptstyle {\text{divisor}}}}\\[1ex]\scriptstyle {\frac {\scriptstyle {\text{numerator}}}{\scriptstyle {\text{denominator}}}}\end{matrix}}\right\}\,=\,} { fraction quotient ratio {\displaystyle \scriptstyle \left\{{\begin{matrix}\scriptstyle {\text{fraction}}\\\scriptstyle {\text{quotient}}\\\scriptstyle {\text{ratio}}\end{matrix}}\right.} Exponentiation (^) base exponent base power } = {\displaystyle \scriptstyle \left.{\begin{matrix}\scriptstyle {\text{base}}\^{\text{exponent}}\\\scriptstyle {\text{base}}\^{\text{power}}\end{matrix}}\right\}\,=\,} power {\displaystyle \scriptstyle {\text{power}}} nth root (√) radicand degree = {\displaystyle \scriptstyle {\sqrt[{\text{degree}}]{\scriptstyle {\text{radicand}}}}\,=\,} root {\displaystyle \scriptstyle {\text{root}}} Logarithm (log) log base (anti-logarithm) = {\displaystyle \scriptstyle \log _{\text{base}}({\text{anti-logarithm}})\,=\,} logarithm {\displaystyle \scriptstyle {\text{logarithm}}} vte The plus sign Addition is written using the plus sign "+" between the terms, and the result is expressed with an equals sign. For example, 1 + 2 = 3 {\displaystyle 1+2=3} reads "one plus two equals three".[2] Nonetheless, some situations where addition is "understood", even though no symbol appears: a whole number followed immediately by a fraction indicates the sum of the two, called a mixed number, with an example,[3] 3 1 2 = 3 + 1 2 = 3.5. {\displaystyle 3{\frac {1}{2}}=3+{\frac {1}{2}}=3.5.} This notation can cause confusion, since in most other contexts, juxtaposition denotes multiplication instead.[4] The terms of addends in the operation of an addition The numbers or the objects to be added in general addition are collectively referred to as the terms,[5] the addends or the summands.[2] This terminology carries over to the summation of multiple terms. This is to be distinguished from factors, which are multiplied. Some authors call the first addend the augend.[6] In fact, during the Renaissance, many authors did not consider the first addend an "addend" at all. Today, due to the commutative property of addition, "augend" is rarely used, and both terms are generally called addends.[7] All of the above terminology derives from Latin. "Addition" and "add" are English words derived from the Latin verb addere, which is in turn a compound of ad "to" and dare "to give", from the Proto-Indo-European root *deh₃- "to give"; thus to add is to give to.[7] Using the gerundive suffix -nd results in "addend", "thing to be added".[a] Likewise from augere "to increase", one gets "augend", "thing to be increased". Redrawn illustration from The Art of Nombryng, one of the first English arithmetic texts, in the 15th century.[8] "Sum" and "summand" derive from the Latin noun summa "the highest, the top" and associated verb summare. This is appropriate not only because the sum of two positive numbers is greater than either, but because it was common for the ancient Greeks and Romans to add upward, contrary to the modern practice of adding downward, so that a sum was literally at the top of the addends.[9] Addere and summare date back at least to Boethius, if not to earlier Roman writers such as Vitruvius and Frontinus; Boethius also used several other terms for the addition operation. The later Middle English terms "adden" and "adding" were popularized by Chaucer.[10] Addition is one of the four basic operations of arithmetic, with the other three being subtraction, multiplication, and division. This operation works by adding two or more terms.[11] An arbitrary of many operation of additions is called the summation.[12] An infinite summation is a delicate procedure known as a series,[13] and it can be expressed through capital sigma notation ∑ {\textstyle \sum } , which compactly denotes iteration of the operation of addition based on the given indexes.[14] For example, ∑ k = 1 5 k 2 = 1 2 + 2 2 + 3 2 + 4 2 + 5 2 = 55. {\displaystyle \sum _{k=1}^{5}k^{2}=1^{2}+2^{2}+3^{2}+4^{2}+5^{2}=55.} Addition is used to model many physical processes. Even for the simple case of adding natural numbers, there are many possible interpretations and even more visual representations. One set has three shapes while the other set has two. The total of shapes is five, which is a consequence of the addition of the objects from the two sets: 3 + 2 = 5 {\displaystyle 3+2=5} . Possibly the most basic interpretation of addition lies in combining sets, that is:[2] When two or more disjoint collections are combined into a single collection, the number of objects in the single collection is the sum of the numbers of objects in the original collections. This interpretation is easy to visualize, with little danger of ambiguity. It is also useful in higher mathematics (for the rigorous definition it inspires, see § Natural numbers below). However, it is not obvious how one should extend this version of an addition's operation to include fractional or negative numbers.[15] One possible fix is to consider collections of objects that can be easily divided, such as pies or, still better, segmented rods. Rather than solely combining collections of segments, rods can be joined end-to-end, which illustrates another conception of addition: adding not the rods but the lengths of the rods.[16] A number-line visualization of the algebraic addition 2 + 4 = 6 {\displaystyle 2+4=6} . A "jump" that has a distance of 2 {\displaystyle 2} followed by another that is as long as 4 {\displaystyle 4} , is the same as a translation by 6 {\displaystyle 6} .A number-line visualization of the unary addition 2 + 4 = 6 {\displaystyle 2+4=6} . A translation by 4 {\displaystyle 4} is equivalent to four translations by 1 {\displaystyle 1} . A second interpretation of addition comes from extending an initial length by a given length:[17] When an original length is extended by a given amount, the final length is the sum of the original length and the length of the extension. The sum a + b {\displaystyle a+b} can be interpreted as a binary operation that combines a {\displaystyle a} and b {\displaystyle b} algebraically, or it can be interpreted as the addition of b {\displaystyle b} more units to a {\displaystyle a} . Under the latter interpretation, the parts of a sum a + b {\displaystyle a+b} play asymmetric roles, and the operation a + b {\displaystyle a+b} is viewed as applying the unary operation + b {\displaystyle +b} to a {\displaystyle a} .[18] Instead of calling both a {\displaystyle a} and b {\displaystyle b} addends, it is more appropriate to call a {\displaystyle a} the "augend" in this case, since a {\displaystyle a} plays a passive role. The unary view is also useful when discussing subtraction, because each unary addition operation has an inverse unary subtraction operation, and vice versa. 4 + 2 = 2 + 4 with blocks Addition is commutative, meaning that one can change the order of the terms in a sum, but still get the same result. Symbolically, if a {\displaystyle a} and b {\displaystyle b} are any two numbers, then:[19] a + b = b + a . {\displaystyle a+b=b+a.} The fact that addition is commutative is known as the "commutative law of addition"[20] or "commutative property of addition".[21] Some other binary operations are commutative too as in multiplication,[22] but others are not as in subtraction and division.[23] 2 + (1 + 3) = (2 + 1) + 3 with segmented rods Addition is associative, which means that when three or more numbers are added together, the order of operations does not change the result. For any three numbers a {\displaystyle a} , b {\displaystyle b} , and c {\displaystyle c} , it is true that:[24] (a + b) + c = a + (b + c) . {\displaystyle (a+b)+c=a+(b+c).} For example, (1 + 2) + 3 = 1 + (2 + 3) {\displaystyle (1+2)+3=1+(2+3)} . When addition is used together with other operations, the order of operations becomes important. In the standard order of operations, addition is a lower priority than exponentiation, nth roots, multiplication and division, but is given equal priority to subtraction.[25] 5 + 0 = 5 with bags of dots Adding zero to any number does not change the number. In other words, zero is the identity element for addition, and is also known as the additive identity. In symbols, for every a {\displaystyle a} , one has:[24] a + 0 = 0 + a = a . {\displaystyle a+0=0+a=a.} This law was first identified in Brahmagupta's Brahmasphutasiddhanta in 628 AD, although he wrote it as three separate laws, depending on whether a {\displaystyle a} is negative, positive, or zero itself, and he used words rather than algebraic symbols. Later Indian mathematicians refined the concept; around the year 830, Mahavira wrote, "zero becomes the same as what is added to it", corresponding to the unary statement 0 + a = a {\displaystyle 0+a=a} . In the 12th century, Bhaskara wrote, "In the addition of cipher, or subtraction of it, the quantity, positive or negative, remains the same", corresponding to the unary statement a + 0 = a {\displaystyle a+0=a} .[26] Main article: Successor function Within the context of integers, addition of one also plays a special role: for any integer a {\displaystyle a} , the integer a + 1 {\displaystyle a+1} is the least integer greater than a {\displaystyle a} , also known as the successor of a {\displaystyle a} . For instance, 3 is the successor of 2, and 7 is the successor of 6. Because of this succession, the value of a + b {\displaystyle a+b} can also be seen as the b {\displaystyle b} -th successor of a {\displaystyle a} , making addition an iterated succession. For example, 6 + 2 is 8, because 8 is the successor of 7, which is the successor of 6, making 8 the second successor of 6.[27] To numerically add physical quantities with units, they must be expressed with common units.[28] For example, adding 50 milliliters to 150 milliliters gives 200 milliliters. However, if a measure of 5 feet is extended by 2 inches, the sum is 62 inches, since 60 inches is synonymous with 5 feet. On the other hand, it is usually meaningless to try to add 3 meters and 4 square meters, since those units are incomparable; this sort of consideration is fundamental in dimensional analysis.[29] Studies on mathematical development starting around the 1980s have exploited the phenomenon of habituation: infants look longer at situations that are unexpected.[30] A seminal experiment by Karen Wynn in 1992 involving Mickey Mouse dolls manipulated behind a screen demonstrated that five-month-old infants expect 1 + 1 to be 2, and they are comparatively surprised when a physical situation seems to imply that 1 + 1 is either 1 or 3. This finding has since been affirmed by a variety of laboratories using different methodologies.[31] Another 1992 experiment with older toddlers, between 18 and 35 months, exploited their development of motor control by allowing them to retrieve ping-pong balls from a box; the youngest responded well for small numbers, while older subjects were able to compute sums up to 5.[32] Even some nonhuman animals show a limited ability to add, particularly primates. In a 1995 experiment imitating Wynn's 1992 result (but using eggplants instead of dolls), rhesus macaque and cottontop tamarin monkeys performed similarly to human infants. More dramatically, after being taught the meanings of the Arabic numerals 0 through 4, one chimpanzee was able to compute the sum of two numerals without further training.[33] More recently, Asian elephants have demonstrated an ability to perform basic arithmetic.[34] Typically, children first master counting. When given a problem that requires that two items and three items be combined, young children model the situation with physical objects, often fingers or a drawing, and then count the total. As they gain experience, they learn or discover the strategy of "counting-on": asked to find two plus three, children count three past two, saying "three, four, five" (usually ticking off fingers), and arriving at five. This strategy seems almost universal; children can easily pick it up from peers or teachers.[35] Most discover it independently. With additional experience, children learn to add more quickly by exploiting the commutativity of addition by counting up from the larger number, in this case, starting with three and counting "four, five." Eventually children begin to recall certain addition facts ("number bonds"), either through experience or rote memorization. Once some facts are committed to memory, children begin to derive unknown facts from known ones. For example, a child asked to add six and seven may know that 6 + 6 = 12 and then reason that 6 + 7 is one more, or 13.[36] Such derived facts can be found very quickly and most elementary school students eventually rely on a mixture of memorized and derived facts to add fluently.[37] Different nations introduce whole numbers and arithmetic at different ages, with many countries teaching addition in pre-school.[38] However, throughout the world, addition is taught by the end of the first year of elementary school.[39] The prerequisite to addition in the decimal system is the fluent recall or derivation of the 100 single-digit "addition facts". One could memorize all the facts by rote, but pattern-based strategies are more enlightening and, for most people, more efficient:[40] Commutative property: Mentioned above, using the pattern a + b = b + a {\displaystyle a+b=b+a} reduces the number of "addition facts" from 100 to 55. One or two more: Adding 1 or 2 is a basic task, and it can be accomplished through counting on or, ultimately, intuition.[40] Zero: Since zero is the additive identity, adding zero is trivial. Nonetheless, in the teaching of arithmetic, some students are introduced to addition as a process that always increases the addends; word problems may help rationalize the "exception" of zero.[40] Doubles: Adding a number to itself is related to counting by two and to multiplication. Doubles facts form a backbone for many related facts, and students find them relatively easy to grasp.[40] Near-doubles: Sums such as 6 + 7 = 13 can be quickly derived from the doubles fact 6 + 6 = 12 by adding one more, or from 7 + 7 = 14 but subtracting one.[40] Five and ten: Sums of the form 5 + x and 10 + x are usually memorized early and can be used for deriving other facts. For example, 6 + 7 = 13 can be derived from 5 + 7 = 12 by adding one more.[40] Making ten: An advanced strategy uses 10 as an intermediate for sums involving 8 or 9; for example, 8 + 6 = 8 + 2 + 4 = 10 + 4 = 14.[40] As students grow older, they commit more facts to memory and learn to derive other facts rapidly and fluently. Many students never commit all the facts to memory, but can still find any basic fact quickly.[37] Main article: Carry (arithmetic) An addition with carry The standard algorithm for adding multidigit numbers is to align the addends vertically and add the columns, starting from the ones column on the right. If a column exceeds nine, the extra digit is "carried" into the next column. For example, in the following image, the ones in the addition of 59 + 27 is 9 + 7 = 16, and the digit 1 is the carry.[b] An alternate strategy starts adding from the most significant digit on the left; this route makes carrying a little clumsier, but it is faster at getting a rough estimate of the sum. There are many alternative methods. Decimal fractions can be added by a simple modification of the above process.[41] One aligns two decimal fractions above each other, with the decimal point in the same location. If necessary, one can add trailing zeros to a shorter decimal to make it the same length as the longer decimal. Finally, one performs the same addition process as above, except the decimal point is placed in the answer, exactly where it was placed in the summands. As an example, 45.1 + 4.34 can be solved as follows: 4 5 . 1 0 + 0 4 . 3 4 ——————————— 4 9 . 4 4 Main article: Scientific notation § Basic operations In scientific notation, numbers are written in the form x = a × 10 b {\displaystyle x=a\times 10^{b}} , where a {\displaystyle a} is the significand and 10 b {\displaystyle 10^{b}} is the exponential part. Addition requires two numbers in scientific notation to be represented using the same exponential part, so that the two significands can simply be added. For example: 2.34 × 10 − 5 + 5.67 × 10 − 6 = 2.34 × 10 − 5 + 0.567 × 10 − 5 = 2.907 × 10 − 5 . {\displaystyle {\begin{aligned}&2.34\times 10^{-5}+5.67\times 10^{-6}\\&\quad =2.34\times 10^{-5}+0.567\times 10^{-5}\\&\quad =2.907\times 10^{-5}.\end{aligned}}} Main article: Binary addition Addition in other bases is very similar to decimal addition. As an example, one can consider addition in binary.[42] Adding two single-digit binary numbers is relatively simple, using a form of carrying: 0 + 0 → 0 0 + 1 → 1 1 + 0 → 1 1 + 1 → 0, carry 1 (since 1 + 1 = 2 = 0 + (1 × 21)) Adding two "1" digits produces a digit "0", while 1 must be added to the next column. This is similar to what happens in decimal when certain single-digit numbers are added together; if the result equals or exceeds the value of the radix (10), the digit to the left is incremented: 5 + 5 → 0, carry 1 (since 5 + 5 = 10 = 0 + (1 × 101)) 7 + 9 → 6, carry 1 (since 7 + 9 = 16 = 6 + (1 × 101)) This is known as carrying.[43] When the result of an addition exceeds the value of a digit, the procedure is to "carry" the excess amount divided by the radix (that is, 10/10) to the left, adding it to the next positional value. This is correct since the next position has a weight that is higher by a factor equal to the radix. Carrying works the same way in binary: 1 1 1 1 1 (carried digits) 0 1 1 0 1 + 1 0 1 1 1 ——————————— 1 0 0 1 0 0 = 36 In this example, two numerals are being added together: 011012 (1310) and 101112 (2310). The top row shows the carry bits used. Starting in the rightmost column, 1 + 1 = 102. The 1 is carried to the left, and the 0 is written at the bottom of the rightmost column. The second column from the right is added: 1 + 0 + 1 = 102 again; the 1 is carried, and 0 is written at the bottom. The third column: 1 + 1 + 1 = 112. This time, a 1 is carried, and a 1 is written in the bottom row. Proceeding like this gives the final answer 1001002 (3610). Addition with an op-amp. See Summing amplifier for details. Analog computers work directly with physical quantities, so their addition mechanisms depend on the form of the addends. A mechanical adder might represent two addends as the positions of sliding blocks, in which case they can be added with an averaging lever. If the addends are the rotation speeds of two shafts, they can be added with a differential. A hydraulic adder can add the pressures in two chambers by exploiting Newton's second law to balance forces on an assembly of pistons. The most common situation for a general-purpose analog computer is to add two voltages (referenced to ground); this can be accomplished roughly with a resistor network, but a better design exploits an operational amplifier.[44] Addition is also fundamental to the operation of digital computers, where the efficiency of addition, in particular the carry mechanism, is an important limitation to overall performance.[45] Part of Charles Babbage's Difference Engine including the addition and carry mechanisms The abacus, also called a counting frame, is a calculating tool that was in use centuries before the adoption of the written modern numeral system and is still widely used by merchants, traders and clerks in Asia, Africa, and elsewhere; it dates back to at least 2700–2300 BC, when it was used in Sumer.[46] Blaise Pascal invented the mechanical calculator in 1642;[47] it was the first operational adding machine. It made use of a gravity-assisted carry mechanism. It was the only operational mechanical calculator in the 17th century[48] and the earliest automatic, digital computer. Pascal's calculator was limited by its carry mechanism, which forced its wheels to only turn one way so it could add. To subtract, the operator had to use the Pascal's calculator's complement, which required as many steps as an addition. Giovanni Poleni followed Pascal, building the second functional mechanical calculator in 1709, a calculating clock made of wood that, once setup, could multiply two numbers automatically. "Full adder" logic circuit that adds two binary digits, A and B, along with a carry input Cin, producing the sum bit, S, and a carry output, Cout. Adders execute integer addition in electronic digital computers, usually using binary arithmetic. The simplest architecture is the ripple carry adder, which follows the standard multi-digit algorithm. One slight improvement is the carry skip design, again following human intuition; one does not perform all the carries in computing 999 + 1, but one bypasses the group of 9s and skips to the answer.[49] In practice, computational addition may be achieved via XOR and AND bitwise logical operations in conjunction with bitshift operations as shown in the pseudocode below. Both XOR and AND gates are straightforward to realize in digital logic, allowing the realization of full adder circuits, which in turn may be combined into more complex logical operations. In modern digital computers, integer addition is typically the fastest arithmetic instruction, yet it has the largest impact on performance, since it underlies all floating-point operations as well as such basic tasks as address generation during memory access and fetching instructions during branching. To increase speed, modern designs calculate digits in parallel; these schemes go by such names as carry select, carry lookahead, and the Ling pseudocarry. Many implementations are, in fact, hybrids of these last three designs.[50] Unlike addition on paper, addition on a computer often changes the addends. Both addends are destroyed on the ancient abacus and adding board, leaving only the sum. The influence of the abacus on mathematical thinking was strong enough that early Latin texts often claimed that in the process of adding "a number to a number", both numbers vanish.[51] In modern times, the ADD instruction of a microprocessor often replaces the augend with the sum but preserves the addend.[52] In a high-level programming language, evaluating a + b {\displaystyle a+b} does not change either a {\displaystyle a} or b {\displaystyle b} ; if the goal is to replace a {\displaystyle a} with the sum this must be explicitly requested, typically with the statement a = a + b {\displaystyle a=a+b} . Some languages like C or C++ allow this to be abbreviated as a += b. // Iterative algorithm int add(int x, int y) { int carry = 0; while (y != 0) { carry = AND(x, y); // Logical AND x = XOR(x, y); // Logical XOR y = carry 32 bits quotient and 32 bits remainder) instructions, and bit field manipulations Addressing modes added scaled indexing and another level of indirection Low cost, EC = 24-bit address 68030: Split instruction and data cache of 256 bytes each On-chip memory management unit (MMU) (68851) Low cost EC = No MMU Burst Memory Interface 68040: Instruction and data caches of 4 KB each Six stage pipeline On-chip floating-point unit (FPU) FPU lacks IEEE transcendental function ability FPU emulation works with 2E71M and later chip revisions Low cost LC = No FPU Low cost EC = No FPU or MMU 68060: Instruction and data caches of 8 KB each 10 stage pipeline Two cycle integer multiplication unit Branch prediction Dual instruction pipeline Instructions in the address generation unit (AGU) and thereby supply the result two cycles before the ALU Low cost LC = No FPU Low cost EC = No FPU or MMU Year CPU Package Frequency (max) [in MHz] Address bus bits MMU FPU 1979 68000 64-pin dual in-line package (DIP), 64-pin SPDIP, 68-pin PLCC, 68-pin CLCC, 68-pin pin grid array (PGA), 64-pin QFP, 68-pin QFP[2] 8–50[3] 24 - - 1982 68008 48-pin dual in-line package (DIP), 52-pin PLCC[4] 8–16.67 24 - - 1982 68010 64-pin DIP, 68-pin PLCC, 68-pin PGA[5] 8–16.67 24 68451 - 1982 68012 84-pin PGA[6] 8–12.5 31 68451 - 1984 68020 114-pin PGA[7] 12.5–33.33 32 68851 68881 - 68EC020 100-pin Quad Flat Package (QFP)[8] 16.7–25 24 - - 1987 68030 132-pin QFP (max 33 MHz), 128-pin PGA[9] 16–50 32 MMU 68881 68EC030 132-pin QFP, 128-pin PGA 25-40[10][11] 32 - 68881 1991 68040 179-pin PGA,[12] 184-pin QFP[13] 20–40 32 MMU FPU 68LC040 PGA,[13] 184-pin QFP[13] 20–33 32 MMU - 68EC040 20–33[13] 32 - - 1994 68060 206-pin PGA[14][15] 50–133[16][17] 32 MMU FPU 68LC060 206-pin PGA,[14][15] 208-pin QFP[18] 50–133[16][17] 32 MMU - 68EC060 206-pin PGA[14][15] 50–133[16][17] 32 - - The Genesis has a 68000 clocked at 7.67 MHz as its main CPU. The 680x0 line of processors has been used in a variety of systems, from high-end Texas Instruments calculators (the TI-89, TI-92, and Voyage 200 lines) to all of the members of the Palm Pilot series that run Palm OS 1.x to 4.x (OS 5.x is ARM-based), and even radiation-hardened versions in the critical control systems of the Space Shuttle. The 680x0 CPU family became most well known for powering desktop computers and video game consoles such as the Macintosh 128K, Amiga, Sinclair QL, Atari ST, Genesis / Mega Drive, NG AES/Neo Geo CD, CDTV. They were the processors of choice in the 1980s for Unix workstations and servers such as AT&T's UNIX PC, Tandy's Model 16/16B/6000, Sun Microsystems' Sun-1, Sun-2, Sun-3, NeXT Computer, Silicon Graphics (SGI), and numerous others. The Saturn uses the 68000 for audio processing and other I/O tasks, while the Jaguar includes a 68000 intended for basic system control and input processing, but was frequently used for running game logic. Many arcade boards also use 68000 processors including those from Capcom, SNK, and Sega. The first several versions of Adobe's PostScript interpreters were 68000-based. The 68000 in the Apple LaserWriter and LaserWriter Plus was clocked faster than the version used then in Macintosh computers. A fast 68030 in later PostScript interpreters, including the standard resolution LaserWriter IIntx, IIf and IIg (also 300 dpi), the higher resolution LaserWriter Pro 600 series (usually 600 dpi, but limited to 300 dpi with minimum RAM installed) and the very high resolution Linotronic imagesetters, the 200PS (1500+ dpi) and 300PS (2500+ dpi). Thereafter, Adobe generally preferred a RISC for its processor, as its competitors, with their PostScript clones, had already gone with RISCs, often an AMD 29000-series. The early 68000-based Adobe PostScript interpreters and their hardware were named for Cold War-era U.S. rockets and missiles: Atlas, Redstone, etc. Microcontrollers derived from the 68000 family have been used in a huge variety of applications. CPU32 and ColdFire microcontrollers have been manufactured in the millions as automotive engine controllers. Many proprietary video editing systems used 68000 processors, such as the MacroSystem Casablanca, which was a black box with an easy to use graphic interface (1997). It was intended for the amateur and hobby videographer market. It is also worth noting its earlier, bigger and more professional counterpart, the "DraCo" (1995). The groundbreaking Quantel Paintbox series of early based 24-bit paint and effects system was originally released in 1981 and during its lifetime it used nearly the entire range of 68000 family processors, with the sole exception of the 68060, which was never implemented in its design. Another contender in the video arena, the Abekas 8150 DVE system, used the 680EC30, and the Play Trinity, later renamed Globecaster, uses several 68030s. The Bosch FGS-4000/4500 Video Graphics System manufactured by Robert Bosch Corporation, later BTS (1983), used a 68000 as its main processor; it drove several others to perform 3D animation in a computer that could easily apply Gouraud and Phong shading. It ran a modified Motorola VERSAdos operating system. Motorola 68000 series registers 31 ... 23 ... 15 ... 07 ... 00 (bit position) Data registers D0 Data 0 D1 Data 1 D2 Data 2 D3 Data 3 D4 Data 4 D5 Data 5 D6 Data 6 D7 Data 7 Address registers A0 Address 0 A1 Address 1 A2 Address 2 A3 Address 3 A4 Address 4 A5 Address 5 A6 Address 6 Stack pointers A7 / USP Stack Pointer (user) A7' / SSP Stack Pointer (supervisor) Program counter PC Program Counter Status Register 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 (bit position) T S M 0 I 0 0 0 X N Z V C SR People who are familiar with the PDP-11 or VAX usually feel comfortable with the 68000 series. With the exception of the split of general-purpose registers into specialized data and address registers, the 68000 architecture is in many ways a 32-bit PDP-11. It had a more orthogonal instruction set than those of many processors that came before (e.g., 8080) and after (e.g., x86). That is, it was typically possible to combine operations freely with operands, rather than being restricted to using certain addressing modes with certain instructions. This property made programming relatively easy for humans, and also made it easier to write code generators for compilers. The 68000 series has eight 32-bit general-purpose data registers (D0-D7), and eight address registers (A0-A7). The last address register is the stack pointer, and assemblers accept the label SP as equivalent to A7. In addition, it has a 16-bit status register. The upper 8 bits is the system byte, and modification of it is privileged. The lower 8 bits is the user byte, also known as the condition code register (CCR), and modification of it is not privileged. The 68000 comparison, arithmetic, and logic operations modify condition codes to record their results for use by later conditional jumps. The condition code bits are "zero" (Z), "carry" (C), "overflow" (V), "extend" (X), and "negative" (N). The "extend" (X) flag deserves special mention, because it is separate from the carry flag. This permits the extra bit from arithmetic, logic, and shift operations to be separated from the carry for flow-of-control and linkage. While the 68000 had a 'supervisor mode', it did not meet the Popek and Goldberg virtualization requirements due to the single instruction 'MOVE from SR', which copies the status register to another register, being unprivileged but sensitive. In the Motorola 68010 and later, this was made privileged, to better support virtualization software. The 68000 series instruction set can be divided into the following broad categories: Load and store (MOVE) Arithmetic (ADD, SUB, MULS, MULU, DIVS, DIVU) Bit shifting (ASL, ASR, LSL, LSR) Bit rotation (ROR, ROL, ROXL, ROXR) Logic operations (AND, OR, NOT, EOR) Type conversion (byte to word and vice versa) Conditional and unconditional branches (BRA, Bcc - BEQ, BNE, BHI, BLO, BMI, BPL, etc.) Subroutine invocation and return (BSR, RTS) Stack management (LINK, UNLK, PEA) Causing and responding to interrupts Exception handling There is no equivalent to the x86 CPUID instruction to determine what CPU or MMU or FPU is present. The Motorola 68020 added some new instructions that include some minor improvements and extensions to the supervisor state, several instructions for software management of a multiprocessing system (which were removed in the 68060), some support for high-level languages which did not get used much (and was removed from future 680x0 processors), bigger multiply (32×32→64 bits) and divide (64÷32→32 bits quotient and 32 bits remainder) instructions, and bit field manipulations. The standard addressing modes are: Register direct Data register, e.g. "D0" Address register, e.g. "A0" Register indirect Simple address, e.g. (A0) Address with post-increment, e.g. (A0)+ Address with pre-decrement, e.g. −(A0) Address with a 16-bit signed offset, e.g. 16(A0) Register indirect with index register and 8-bit signed offset e.g. 8(A0,D0) or 8(A0,A1) For (A0)+ and −(A0), the actual increment or decrement value is dependent on the operand size: a byte access adjusts the address register by 1, a word by 2, and a long by 4. PC (program counter) relative with displacement Relative 16-bit signed offset, e.g. 16(PC). This mode was very useful for position-independent code. Relative with 8-bit signed offset with index, e.g. 8(PC,D2) Absolute memory location Either a number, e.g. "$4000", or a symbolic name translated by the assembler Most assemblers used the "$" symbol for hexadecimal, instead of "0x" or a trailing H. There were 16 and 32-bit versions of this addressing mode Immediate mode Data stored in the instruction, e.g. "#400" Quick immediate mode 3-bit unsigned (or 8-bit signed with moveq) with value stored in opcode In addq and subq, 0 is the equivalent to 8 e.g. moveq #0,d0 was quicker than clr.l d0 (though both made D0 equal to 0) Plus: access to the status register, and, in later models, other special registers. The Motorola 68020 added a scaled indexing address mode, and added another level of indirection to many of the pre-existing modes. Most instructions have dot-letter suffixes, permitting operations to occur on 8-bit bytes (".b"), 16-bit words (".w"), and 32-bit longs (".l"). Most instructions are dyadic, that is, the operation has a source, and a destination, and the destination is changed. Notable instructions were: Arithmetic: ADD, SUB, MULU (unsigned multiply), MULS (signed multiply), DIVU, DIVS, NEG (additive negation), and CMP (a comparison done by subtracting the arguments without storing the result, setting the status bits) Binary-coded decimal arithmetic: ABCD, NBCD, and SBCD Logic: EOR (exclusive or), AND, NOT (logical not), OR (inclusive or) Shifting: (logical, i.e. right shifts put zero in the most-significant bit) LSL, LSR, (arithmetic shifts, i.e. sign-extend the most-significant bit) ASR, ASL, (rotates through eXtend and not) ROXL, ROXR, ROL, ROR Bit test and manipulation in memory or data register: BSET (set to 1), BCLR (clear to 0), BCHG (invert) and BTST (no change). All of these instructions first test the destination bit and set (clear) the CCR Z bit if the destination bit is 0 (1), respectively. Multiprocessing control: TAS, test-and-set, performed an indivisible bus operation, permitting semaphores to be used to synchronize several processors sharing a single memory Flow of control: JMP (jump), JSR (jump to subroutine), BSR (relative address jump to subroutine), RTS (return from subroutine), RTE (return from exception, i.e. an interrupt), TRAP (trigger a software exception similar to software interrupt), CHK (a conditional software exception) Branch: Bcc (where the "cc" specified one of 14 tests of the condition codes in the status register: equal, greater than, less-than, carry, and most combinations and logical inversions, available from the status register). Of the remaining two possible conditions, always true and always false, BRA (branch always) has a separate mnemonic, and BSR (branch to subroutine) takes the encoding that would otherwise have been 'branch never'. Decrement-and-branch: DBcc (where "cc" was as for the branch instructions), which, provided the condition was false, decremented the low word of a D-register and, if the result was not -1 ($FFFF), branched to a destination. This use of −1 instead of 0 as the terminating value allowed the easy coding of loops that had to do nothing if the count was 0 to start with, with no need for another check before entering the loop. This also facilitated nesting of DBcc. This section does not cite any sources. Please help improve this section by adding citations to reliable sources. Unsourced material may be challenged and removed. (October 2013) (Learn how and when to remove this message) Motorola mainly used even numbers for major revisions to the CPU core such as 68000, 68020, 68040 and 68060. The 68010 was a revised version of the 68000 with minor modifications to the core, and likewise the 68030 was a revised 68020 with some more powerful features, none of them significant enough to classify as a major upgrade to the core. The 68050 was reportedly "a minor upgrade of the 68040" that lost a battle for resources within Motorola, competing against projects that had been scheduled to succeed it: the 0.5µm, low-power, low-cost "LP040", and the superscalar, superpipelined "Q", borrowing from the 88110 and anticipated as the 68060.[19] Subsequent reports indicated that Motorola had considered the 68050 as not meriting the necessary investment in production of the part.[20] Odd-numbered releases had always been reactions to issues raised within the prior even numbered part; hence, it was generally expected that the 68050 would have reduced the 68040's power consumption (and thus heat dissipation), improved exception handling in the FPU, used a smaller feature size and optimized the microcode in line with program use of instructions. Many of these optimizations were included with the 68060 and were part of its design goals. For any number of reasons, likely that the 68060 was in development, that the Intel 80486 was not progressing as quickly as Motorola assumed it would, and that 68060 was a demanding project, the 68050 was cancelled early in development. There is also no revision of the 68060, as Motorola was in the process of shifting away from the 68000 and 88k processor lines into its new PowerPC business, so the 68070 was never developed. Had it been, it would have been a revised 68060, likely with a superior FPU (pipelining was widely speculated upon on Usenet). There was a CPU with the 68070 designation, which was a licensed and somewhat slower version of the 16/32-bit 68000 with a basic DMA controller, I²C host and an on-chip serial port. This 68070 was used as the main CPU in the Philips CD-i. This CPU was, however, produced by Philips and not officially part of Motorola's 680x0 lineup. Motorola had announced a product roadmap beyond the 68060 featuring the 68080 rated at 200-350 MIPS, due by 1995, and a product rated at 800 MIPS, possibly with the name 68100, by 2000.[20] The 4th-generation 68060 provided equivalent functionality (though not instruction-set-architecture compatibility) to most of the features of the Intel P5 microarchitecture. The Personal Computers XT/370 and AT/370 PC-based IBM-compatible mainframes each included two modified Motorola 68000 processors with custom microcode to emulate S/370 mainframe instructions.[21][22] An Arizona-based company, Edge Computer Corp, reportedly founded by former Honeywell designers, produced processors compatible with the 68000 series, these being claimed as having "a three to five times performance – and 18 to 24 months' time – advantage" over Motorola's own products.[23] In 1987, the company introduced the Edge 1000 range of "32-bit superminicomputers implementing the Motorola instruction set in the Edge mainframe architecture", employing two independent pipelines - an instruction fetch pipeline (IFP) and operand executive pipeline (OEP) - relying on a branch prediction unit featuring a 4096-entry branch cache, retrieving instructions and operands over multiple buses.[24] An agreement between Edge Computer and Olivetti subsequently led to the latter introducing products in its own "Linea Duo" range based on Edge Computer's machines.[25] The company was subsequently renamed to Edgcore Technology Inc.[26]:12 (also reported as Edgecore Technology Inc.[27]). Edgcore's deal with Olivetti in 1987 to supply the company's E1000 processor was followed in 1989 by another deal with Philips Telecommunications Data Systems to supply the E2000 processor, this supporting the 68030 instruction set and reportedly offering a performance rating of 16 VAX MIPS.[28] Similar deals with Nixdorf Computer and Hitachi were also signed in 1989.[29][30] Edge Computer reportedly had an agreement with Motorola.[27] Despite increasing competition from RISC products, Edgcore sought to distinguish its products in the market by emphasising its "alliance" with Motorola, employing a mar