**2. What was the elementary base of a computer/processor (relays, vacuum tubes (aka thermoionic valves), transistors, magnetic cores, integrated circuits (ICs) – were they hybrid, low scale integration, Large Scale Integration (LSI), Very Large Scale Integration (VLSI), modern microprocessors)? What where the physical characteristics of the equipment (weight, size, power consumption)?**

**Kokia buvo elementinė kompiuterio/procesoriaus bazė (relės, vakuuminės lempos, tranzistoriai, magnetinės šerdys, integriniai grandynai (IC) – ar jie hibridiniai, mažo integracijos masto, Didelio integracijos masto (LSI), labai didelio integracijos masto (VLSI), ar tai buvo monokristaliniai šiuolaikiniai mikroprocesoriai)? Kokios buvo fizinės įrangos savybės (svoris, dydis, energijos suvartojimas)?**

**3. What type of architecture did both computers have? Were they accumulator based, register based, stack based, memory-to-memory or some other architectures?**

**Kokio tipo architektūrą turėjo abu kompiuteriai? Ar jie buvo akumuliatoriniai, registriniai, stekiniai (dėklo architektūros), atmintis-į-atmintį architektūros arba kokios nors kitos architektūros?**

**4. Were they zero-address, one-address, two-address, three-address or four address machines?**

**Ar tai buvo beadresinės (stekinės), vieno adreso, dviejų adresų,trijų adresų ar keturios adresų mašinos?**

**5. What were the registers in both architectures? Did they have registers at all? Were they general purpose registers or specialized ones? How many registers did each architecture have? What were the widths of these registers?**

**Kokie buvo registrai abiejose architektūrose? Ar šios architektūros iš viso turėjo registrus? Ar tai buvo bendrosios paskirties registrai, ar specializuoti registrai? Kiek registrų turėjo kiekviena architektūra? Kokie buvo šių registrų duomenų pločiai? Kokia buvo specifinė registrų paskirtis?**

**6. Were the flags used in the architectures? What flags were there?**

**Ar požymių bitai buvo naudojami šiose architektūrose? Kokie požymiai buvo naudojami?**

Naudojami. The Program Status includes six processor flags as follows: Carry (C), Zero (Z), Sign (S), Parity/Overflow (P/V), Decimal Adjust (D), and Half Carry (H).

**7. What was the data width (machine word) of each architecture?**

**Koks buvo kiekvienos architektūros duomenų plotis (mašininis žodis)?**

Data formats: The CPU manipulates bits, bytes (a bits), words (16 bits), longwords (32 bits), and quadwords (64 bits) of data. Within a byte, word, longword, or quadword, the bits are numbered from right to left, from least to most significant (Figure 2-1). This is consistent with the convention that bit n corresponds to position 2n in the represen tation of binary numbers.

All of the functional units and data paths listed abova are 32 bits wide.

**8. What was the memory layout of each system? Was address space continuous, or was it segmented, paged, subdivided into banks? What was the (effective) width of the address? What was the maximum possible amount of memory in each system? What was the typical amount of memory with which the system was used?**

**Koks buvo kiekvienos sistemos atminties išdėstymas? Ar adresų buvo erdvė ištisinė, ar ji buvo suskirstytas į segmentus, puslapius, atminties į bankus? Koks buvo (efektyvus) adreso plotis? Koks buvo maksimalus įmanomas atminties kiekis kiekvienoje sistemoje? Koks buvo tipiškas atminties kiekis, su kuria sistema buvo naudojama?**

**Adress spaces:**

The CPU uses 32-bit logical addresses, permitting direct access to 4G bytes of memory. The logical addresses are translated by the memory management mechanism to the physical addresses used to access memory and peripherals. The CPU supports three modes of address represen tation--compact, segmented, and linear--selected by two control bits in the Flag and Control Word register. Applications with an address space smaller than 64K bytes can take advantage of the dense code and efficient use of base registers with the 16-bit compact addresses. Although pro grams executing in compact mode can only manipu late 16-bit addresses, the logical address is extended to 32 bits by concatenating the 16 most significant bits of the Program Counter register. Compact mode is equivalent to the laOOO non-seg mented mode. Segmented mode supports two segment sizes--64K bytes and 16M bytes. Up to 32,76a of the small segments and 12a of the large segments are avail able. In segmented mode, address calculations do not affect the segment number, only the offset wi thin the segment. Allocating individual objects such as program modules, stacks, or large data structures to separate segments allows appli cations to benefit from the logical structure of a segmented memory space. The 32-bit addresses in linear mode provide uni form and unstructured access to 4G bytes of mem ory. Some applications benefit from the flexibil ity of linear addressing by allocating objects to arbitrary positions in the address space.

**Memory management:**

The processor includes a [memory management unit](https://en.wikipedia.org/wiki/Memory_management_unit) that provides [protected memory](https://en.wikipedia.org/wiki/Protected_memory), important for [multitasking](https://en.wikipedia.org/wiki/Computer_multitasking), and [virtual memory](https://en.wikipedia.org/wiki/Virtual_memory) addressing for temporary storage of RAM on a [hard disk](https://en.wikipedia.org/wiki/Hard_disk). The processor has three methods of accessing memory:

* compact mode – meant for small programs, could only access 64 KB (16-bit addresses, equivalent to the Z8000's non-segmented mode). Address bits 31-16 of all virtual addresses comes from address bits 31-16 of the program counter.
* segmented mode – 32,768 segments of 64 KB (16-bit address; comprising memory from 0-2GB) *and* 128 segments of 16 MB (24-bit address; comprising memory from 2GB-4GB), making a total of 4 GB (32-bit address) of accessible memory.
* linear mode – direct 4 GB (32-bit address) accessible memory

Memory management provides two valuable func tions--address translation and access protection. Access protection ensures that proprietary por tions of memory, or those portions concerned with operating system functions, are protected from tampering. Address translation, the process of mapping a program's logical addresses to the phys ical addresses used to access memory, streamlines system performance, since the operating system can relocate programs in memory, free from rigid con straints. By integrating memory management with the processor in a single chip, the l80, 000 CPU reduces parts-count and improves memory access time. Another memory management function, demand-paged v irtual memory, a !lows programs to execute even when only a portion of their memory requirements is available in primary storage. The rest of the program can be stored in secondary storage, typi cally on disk. Thus, virtual memory improves a system's cost/performance by permitting programs to execute with varying amounts of memory. The CPU implements a paged translation mechanism similar to that of most mainframe and super-mini computers. The operating system creates transla tion tables in memory, then loads pointers to the tables in control registers. The CPU automatically refers to the tables to perform address transla tion and access protection. To manage the large logical address space, the translation scheme divides it into fixed-size, 1K byte pages. Similarly, the physical address space is divided into fixed-size frames, also 1K-bytes each. The memory management mechanism maps a log ical page to an arbitrary physical frame (Figure 1-1). Since both the pages and frames are of fixed and equal size, the operating system's mem ory allocation problem is simplified. The CPU implements a Translation Lookaside Buffer (TLB) to store the information needed to translate the sixteen most recent! y used pages. When the information needed to translate a page is missing from the TLB, the CPU automatically translates the address using the tables in memory, and then loads the information into the TLB. The memory management mechanism can be used to map logical memory addresses to physical I/O addresses. The use of memory-mapped I/O permits protected access by application programs to selected peripheral devices.

**9. Was virtual memory supported, and how? Was it paged or segmented?**

**Ar buvo palaikoma virtualioji atmintis ir kaip? Ar virtuali atmintis buvo realizuoti, naudojant puslapiavimą, segmentavimą, abu šiuos mechanizmus?**

(11psl.)Another memory management function, demand-paged v irtual memory, a !lows programs to execute even when only a portion of their memory requirements is available in primary storage. The rest of the program can be stored in secondary storage, typi cally on disk. Thus, virtual memory improves a system's cost/performance by permitting programs to execute with varying amounts of memory.

The CPU implements a paged translation mechanism similar to that of most mainframe and super-mini computers. The operating system creates transla tion tables in memory, then loads pointers to the tables in control registers. The CPU automatically refers to the tables to perform address transla tion and access protection.

**10. What was the ISA of each architecture? What instructions formats did each architecture supportz?**

**Kokia buvo kiekvienos architektūros komandų sistema (ISA)?** Kiek mašinos komandų turėjo kiekviena architektūra?Kokios buvo instrukcijų (komandų) klasės? **Kokius instrukcijų formatus palaikė kiekviena architektūra?** Pateikite 8–16 instrukcijų pavyzdžius**.**

(apie 50-60 psl. dokumentacijoj)

Turėjo apie 800 skirtingų komandos kodų.

Komandų klasės:

• Load and Exchange (pvz.: EX, EXB, EXL dst,src – Exchange) (čia įeina ir load, pop, push ir pan.)

• Arithmetic (INDEX, INDEXL dst,sub,src Index), aišku čia ir visos + - \* / compare

• Logical ( XOR XORB XOR dst,src Exclusive Or) (and, or, test)

• Program Control (Trap cc, src Contditional trap) call, jp, jr, enter, exit

• Bit Manipulation (RES RESB RES dst,src reset bit), set Bit, bit test

• Bit Field (INSRT dst, src, pos, siz Insert field)

• Rotate and Shift (RlC RlCB RlCl dst,src Rotate left through Carry)

• Block Transfer and String Manipulation (CPD CPDB CPDL dst,src,r,cc Compare and Decrement)

• Input/Output (INIR INIRB INIRL dst,src,r Input, Increment and Repeat)

• CPU Control (SETFLG flag Set Flag)

• Extended Instructions (The CPU supports four types of extended instruc tions: EPU internal operations that do not require any data transfer; transfer of one to six teen words of data between the EPU and consecutive word or longword general-purpose registers; transfer of one byte of data between the EPU and the flag byte of the FCW; and the transfer of one to sixteen bytes or words of data between the EPU and memory. The flags are affected only when the flag byte is loaded.)

**11. What were the addressing modes supported by each architecture? Which modes were similar, and which were different?**

**Kokius adresavimo būdus palaikė kiekviena architektūra? Kurie režimai buvo panašūs, o kurie skyrėsi?**

The CPU locetes operands (the data manipulated by inatructions) in regiaters, memory, peripheral ports, or in the inatruction. The location of an operand is specified by one of nine general addresaing modes: Register, Immediate, Indirect Register, Direct Address, Index, Baae Addresa, Base Index, Relative Address, and Relative Index. Instruction formats provide compact encodinga for the most frequently used addressing modes.

**12. What were the I/O capabilities of each architecture? Kokios buvo kiekvienos architektūros I/O galimybės?**

Komandos: Input; Input and Decrement; Input, Decrement and Repeat; Input and Increment; Input, Increment and Repeat; Output, Decrement and Repeat; Output, Increment and Repeat; Output; Output and Decrement; Output and Increment

**13. Were interrupts supported for each architecture? How was the interrupt support similar, and how was it different in both architectures? Ar buvo palaikomi pertraukimai? Kuo pertraukimų mechanizmai buvo panašūs, kuo jie skyrėsi abiejose architektūrose?**

267 ir 293 psl dokumentacijoj

Ths CPU recognizes three kinds of interrupt sig nslled on separsts pinal non-masksble, vectored, snd non-vectored. Non-mssksble interrupts sre slwsys enabled. Vsctored and non-vectored inter rupta can be selectively enabled by bita VIE snd NVIE in the FCW.

An interrupt occurs when an enabled interrupt request is signalled on a CPU pin. The CPU gener ates an interrupt acknowledge trsnsaction on tha external intarface to fetch the idantifier word, which is then savad on the system stack. For vec tored interrupts, the low-order byte of the iden tifier word is used to select a pointer to a par ticular interrupt handler routine. Refer to Sec tion 8.7.5 for more details about interrupt request and acknowledge.,

**14. What data types did each architecture support on instruction level? Was fixed point or floating point supported by hardware and how? Where integers sign-magnitude, one's complement, two's complement? What other "exotic" data types did architecture support (decimal numbers, complex numbers, etc.)?**

**Kokius duomenų tipus palaikė kiekviena architektūra aparatūros lygyje? Ar buvo palaikoma fiksuoto kablelio, slankiojo kablelio aritmetika? Ar sveikieji skaičiai buvo koduojami kaip ženklas-dydis, kaip vieneto papildinys (atvirkštinis kodas), dvejeto papildyti (papildomas kodas)? Kokius kitus „egzotiškus“ duomenų tipus palaikė architektūra (pvz. dešimtainius skaičius, kompleksinius skaičius ir kt.)?**

he Z80,OOO CPU supports operations on nine data types: bit, bit field, aigned integer, unaigned integer, logical value, address, packed BCD integer, stack, and string. Integer and logical values can be byte, word, or longword in size. In addition, floating-point operations are imple mented through the Extended Processing Architec ture (EPA) facility by a coproceasor (ZB07D Arith metic Processing Unit) or by software emulation.

Iš komandų dalies: The arithmetic group consists of instructions for performing integer arithmetic. The basic instruc tions operate on unsigned binary integers or signed twos complement binary integers. Support is provided for Binary Coded Decimal (BCD) arith metic and multiple precision arithmetic. Add and Subtract operate on bytes, words, or longwords. The Multiply instructions operate on words or long words and compute a double-precision product. The Divide instructions operate on words or longwords, using a double-precision dividend.

**15. What was the speed of each system? How did clock frequencies, clock cycles per instruction, instruction rates compare? Which system was faster?**

**Kokia buvo kiekvienos sistemos greitaveika? Kokie buvo taktinių generatorių dažniai, vidutinis/mažiausias/didžiausias ciklų skaičius, reikalingas kiekvienai komandai įvykdyti, vidutinė sistemos greitaveika? Kuri sistema buvo našesnė? Koks buvo kainos ir našumo santykis?**

1.4 pradžioj kažkur

The system designer can fine-tune performance by selecting not only the CPU clock rate and bus speed (1/2 or 114 the CPU clock), but also the access time and date path width for the memory. for two independent regions of memory the CPU can be programmed for both the number of wait states automatically inserted, and whether the data path is 16 or 32 bits wide. With these options, a system can essily accommodate slow, 16-bit-wide bootstrap read-only memory (ROM) in one region and fast, 32-bit-wide random access memory (RAM) in the other. furthermore, the CPU supports an optional burst transfer of several memory words from consecutive locations. Burst transfers can increase memory bandwidth for interleaved and "nibble-mode" memory systems.

**16. Did the architectures use cache memory? If yes, how much? Ar architektūros naudojo spartinančią atmintį? Jei taip, kokio dydžio?**

276psl.

The CPU implements a cache mechanism that keeps a copy of recently used memory locations on-chip. These locations can contain both instructions and data. On memory fetches, the CPU examines the cache to determine if the addressed information is stored there. If the information is in the cache (a hit), then the CPU fetches the copy from the cache, and no transaction is necessary on the external interface. If the information is not in the cache (a miss), then the CPU performs a memory read transaction to fetch the missing information and stores a copy of the information into the cache, replacing the least recently used data in the cache. Thus, the cache serves to reduce the number of memory read transactions, providing a substantial boost to performance. Software can control the cache mechanism in several ways. The System Configuration Control Longword register contains separate control bits (CI and CD) that enable the cache for instruction and data references and another bit (CR) that enables the cache replacement algorithm. In page table entries, the NC bit can be set to disable the use of the cache for selected pages. The Purge Cache instruction can be executed to invali date the contents of the cache when a memory loca tion that may have been copied into the cache has been modified by another processor. For example, if a slave processor reads from a peripheral port to a memory location that may be copied in the cache, the cache must be purged. Similarly, if two or more tightly-coupled CPUs can alternately execute one process, the cache must be purged when the operating system changes from executing one user-process to another. Appendix C describes the cache mechanism in more detail, control and interface.

The 180,000 CPU contains an on-chip cache buffer to store copies of memory locations that were recently referred to. Most memory references are either to a location that was referred to recently (temporal locality) or to a nearby location (spatial locality). Therefore, on most memory fetches the CPU is able to find the required data in the cache (a hit), thus avoiding a slower access to external memory. Whsn the required data is missing from the csche (a miss), the CPU fetches the dsta from sxternal memory snd losds a copy into the cache. The fetched data replaces the least recently used dats in the cache. The cache provides significant cost/performance advantages by allowing the CPU to execute instructions at a faster rste than permitted by external memory alone. The cache can be separately enabled to store both instructions and data. The effec tiveness of the cache is enhanced by storing data along with instructions, but an spplication can cache instructions only. Cache replacement on a misa can also be inhibited. This option can be used to lock desired locations into the cache for fast, on-chip access.

The zao,ooo CPU also includes a cache memory, which complements the pipelined design to achieve high performance with moderate memory speeds.

**17. What were the typical application areas of each architecture? How were they used? Describe briefly (on paragraph) one particular installation of each architecture.**

**Kokios buvo tipinės kiekvienos architektūros taikymo sritys? Kaip šios architektūros buvo naudojamos? Trumpai apibūdinkite (vienoje pastraipoje) vieną konkretų kiekvienos architektūros panaudojimo pavyzdį.**

**18. How much software was written for each discussed architecture, is it (still) available, what application domains did it target? What compilers and programming tools (debuggers, profilers, assemblers) did architectures have? What software libraries were available?**

**Kiek programinės įrangos buvo parašyta kiekvienai aptariamai architektūrai, ar ji (vis dar) prieinama, kur ji buvo naudojama? Kokie buvo prieinami kompiliatoriai ir programavimo įrankiai (derintojai, profiliuotojai, surinkėjai)? Kokios programinės įrangos bibliotekos buvo prieinamos?**

**19. (optional) are there emulators available for both architectures? If you find some, provide their URLs and/or publication metadata.**

**(neprivaloma) ar yra emuliatorių abiem architektūroms? Jei tokių rasite, pateikite jų URL ir (arba) leidinio metaduomenis.**

**Šaltiniai:**

[**https://chatgpt.com/c/67573623-a2dc-800e-8234-e8efd221e61a**](https://chatgpt.com/c/67573623-a2dc-800e-8234-e8efd221e61a)

Panaudojimo data ir laikas: 12-09 20:27 ChatGPT 4o

Klausimas: What was the elementary base of a computer/processor z80000 (relays, vacuum tubes (aka thermoionic valves), transistors, magnetic cores, integrated circuits (ICs) – were they hybrid, low scale integration, Large Scale Integration (LSI), Very Large Scale Integration (VLSI), modern microprocessors)? What where the physical characteristics of the equipment (weight, size, power consumption)?

Atsakymas: The **Zilog Z8000** processor, introduced in 1979, was a microprocessor built using **integrated circuits (ICs)**, specifically **Large Scale Integration (LSI)** technology. It was a 16-bit processor and represented an advancement in microprocessor design, focusing on medium to high-performance computing for its time.

**Elementary Base of the Z8000**

* **Technology:** Integrated Circuits (ICs)
* **Integration Scale:** **Large Scale Integration (LSI)**. This refers to the integration of thousands of transistors on a single chip.
* **Transistors:** It used **metal-oxide-semiconductor (MOS)** technology, which was common in the late 1970s and early 1980s.

**Physical Characteristics of Z8000-based Equipment**

1. **Weight and Size:**
   * The Z8000 was used in systems such as minicomputers, industrial control systems, and embedded systems, which varied in size.
   * Typical systems housing the Z8000 could range from small desktop units (weighing around 20–50 pounds for compact systems) to larger rack-mounted units.
2. **Power Consumption:**
   * The processor itself consumed **low power relative to earlier computing technologies** like vacuum tubes or early transistors.
   * The typical power consumption of the Z8000 processor chip was in the range of **several watts**, though the entire system’s power consumption depended on peripheral components.
3. **Environmental Characteristics:**
   * Systems using the Z8000 were smaller and less power-hungry compared to older vacuum tube or magnetic core systems.
   * These systems typically required modest cooling solutions compared to earlier computing technologies.

The Z8000 was not designed for ultra-high-performance computing but targeted versatile applications, such as running operating systems, embedded systems control, and general-purpose computing in industries.

<https://www.cpushack.com/CPU/cpu3.html#Sec3Part3>

The Z-8000 was introduced not long after the [8086](https://www.cpushack.com/CPU/cpu3.html#8086), but had superior features. It was basically a 16 bit processor, but could address up to 23 bits in some versions by using [segment](https://www.cpushack.com/CPU/cpuAppendC.html#segment) registers (to supply the upper 7 bits). There was also an unsegmented version, but both could be extended further with an additional MMU that used 64 segment registers. The Z-8070 was a memory mapped FPU.

Internally, the Z-8000 had sixteen 16 bit registers, but register size and use were exceedingly flexible - the first eight Z-8000 registers could be used as sixteen 8 bit subregisters (identified RH0, RL0, RH1 ...), or all sixteen could be grouped into eight 32 bit registers (RR0, RR2, RR4 ...), or four 64 bit registers. They were all general purpose registers - the stack pointer was typically register 15, with register 14 holding the stack segment (both accessed as one 32 bit register (RR14) for painless address calculations). The instruction set included 32-bit multiply (into 64 bits) and divide.

The Z-8000 was one of the first to feature two modes, one for the operating system and one for user programs. The user mode prevented the user from messing about with interrupt handling and other potentially dangerous stuff (each mode had its own stack register).

Finally, like the [Z-80](https://www.cpushack.com/CPU/cpu1.html#Z80), the Z-8000 featured automatic RAM refresh circuitry. Unfortunately the processor was somewhat slow, but the features generally made up for that.

A later version, the Z-80000, was introduced about at the beginning of 1986, at about the same time as the 32 bit [MC68020](https://www.cpushack.com/CPU/cpu3.html#68020) and [Intel 80386](https://www.cpushack.com/CPU/cpu3.html#80386) CPUs, though the Z-80000 was quite a bit more advanced. It was fully expanded to 32 bits internally, giving it sixteen 32 bit physical registers (the 16 bit registers became subregisters), doubling the number of 32 bit and 64 bit registers (sixteen 8-bit and 16-bit subregisters, 32-bit physical registers, eight 64-bit double registers). The system stack remained in RR14.

In addition to the addressing modes of the Z-8000, larger 24 bit (16Mb) segment addressing was added, as well as an integrated MMU (absent in the [68020](https://www.cpushack.com/CPU/cpu3.html#68020) but added later in the [68030](https://www.cpushack.com/CPU/cpu3.html#68030)) which included an on chip 16 line 256-byte fully associated write-through cache (which could be set to cache only data, instructions, or both, and could also be frozen by software once 'primed' - also found on later versions of the [AMD 29K](https://www.cpushack.com/CPU/cpu4.html#29K)). It also featured multiprocessor support by defining some memory pages to be exclusive and others to be shared (and non-cacheable), with separate memory signals for each (including GREQ (Global memory REQuest) and GACK lines). There was also support for coprocessors, which would monitor the data bus and identify instructions meant for them (the CPU had two coprocessor control lines (one in, one out), and would produce any needed bus transactions).

Finally, the Z-80000 was fully pipelined (six stages), while the fully pipelined [80486](https://www.cpushack.com/CPU/cpu3.html#80486) and [68040](https://www.cpushack.com/CPU/cpu3.html#68040) weren't introduced until 1991.

But despite being technically advanced, the Z-8000 and Z-80000 series never met mainstream acceptance, due to initial bugs in the Z-8000 (the complex design did not use [microcode](https://www.cpushack.com/CPU/cpuAppendC.html#microcode) - it used only 17,500 transistors) and to delays in the Z-80000. There was a radiation resistant military version, and a CMOS version of the Z-80000 (the Z-320). Zilog eventually gave up and became a second source for the AT&T WE32000 32-bit (1986) CPU instead (a [VAX-like](https://www.cpushack.com/CPU/cpuAppendA.html#VAX) microprocessor derived from the Bellmac 32A minicomputer, which also became obsolete).

The Z-8001 was used for Commodore's CBM 900 prototype, but the Unix based machine was never released - instead, Commodore bought Amiga, and released the [68000](https://www.cpushack.com/CPU/cpu3.html#68000) based machine it was designing. A few companies did produce Z-8000 based computers, with Olivetti being the most famous, and the Plexus P40 being the last - the [68000](https://www.cpushack.com/CPU/cpu3.html#68000) quickly became the processor of choice, although the Z8000 continued to be used in embedded systems.