Computer Science Journal of Moldova, vol.**1**, no.1(1), 1993

The main concepts of a new HLL computer "CAMCOH"

A.Terekhov

Abstract

The concepts of a new computer and the peculiarities of its architecture are discussed.

A new industrial technology of creating complicated embedded systems was elaborated in Sankt-Petersburg university in collaboration with some industrial institutes; it was successfully used in several large projects (electronic telephone stations, network control, etc.). The main idea of this technology is using fixed high level algorithmic language (HLL) through all life cycle stages beginning with requirements and specification to translation to the target computer codes and maintenance. It was shown that any HLL is not fit enough for successful use as a basic language. For technology purpose the best are languages of static type (i.e. with full mode checking in compile time), e.g., Algol 68, Ada, Modula 2.

Habitually in embedded systems specialized computers are applied which are possible to use in extreme situations. But it is only very rare that such a computer has an involved operating system, disks and displays or HLL compilers. That is why large software projects are to be worked out using powerful tool (host) computers and cross-compilers. In Computer Science Laboratory of Sankt-Petersburg University more than 15 compilers and cross-compilers were implemented, best of which provide the coefficient of object code enlarging in compare with hand written programs less than 1.4. The implementation of a compiler, however, is very sophisticated problem and even a small loss of 1.4 times is often unfit for limited resources of specialized computers.

The decision was carried out to build a new computer which was later on called "CAMCOH" (pronounce as "SAMSON") satisfying the demands of embedded applications, but comfortable enough to use it as a tool. HLL computers have already been worked out both in our country and abroad but the results were usually complex and expensive. In our opinion the complexity wasn't due to the HLL correspondence but to universality of applications. We made an effort to restrict the class of problems to solve to the of so-called "communication type problems". In fact programming these problems demand integer arithmetics and logics. However, connection net control needs a dialog with operator, file system, string handling, information equipment needs effective database handling, reconfiguration of connection nets — complicated probability calculations with floating point numbers and so on. That's why even this class of problems is too wide.

We convinced now that to carry out a computer according to a small class of problems doesn't make any practical sense, but for to produce simple cheap computer we were to search for some restrictions.

The decision has come unexpectedly. We decided to limit the class of used languages. The practical experience of using static languages has convinced us that a compiler from such a language can accomplished by hardware. In fact if the scheme "for problem solving" is like this:

$$problem \rightarrow compiler \rightarrow computer \rightarrow result$$

and no detour can be found. One can see that if a computer and a compiler are planned and implemented simultaneously their duties can freely distributed between them. Evidently the price of installation is disappearing in compare with price of computations and advantages are to be taken of complicating the compiler and simplifying the hardware. On the other side, an orthogonal design, absence of traditional arbitrary restrictions simplifies the code generation and at least compiler written for such a computer is a great deal more simple than for a usual computer.

Let us list out some peculiarities of "CAMCOH".

1. "CAMCOH" hardware is not oriented to one HLL, but its instruc-

tion set supports main operators of modern HLL (procedure call, loop, slice of an array and so on). It can be safely affirmed now that a "gold fund" of HLL constructions has already been fixed, that varies in syntax but coincides in semantics.

2. The instructions set is addressless, like Polish reverse notation. For instance, an expression

$$(a + b) * (c - d)$$

can be written

$$ab + cd - *$$

This form of notation was invented more than half a century ago, it is compact and easily interpreted. But an error in this notation is easily made and difficulty found, that's way only very few software systems trust the user to make it (e.g. FORTH). The compiler will turn the program into this notation without any errors.

3. The main part of instructions are coded by one byte, the more rare ones being coded by 12 or 16 bits. Four of the most frequent instructions are coded by 4 bits, for instance, loading to the stack of the first 16 variables of current procedure is coded by one byte instruction.

4. There are 3 register stacks in "CAMCOH":

   - integer stack of 16 16-bit registers;
   - real stack of 8 32-bit registers;
   - address stack of 16 40-bit registers.

   The compiler for static HLL always defines the stack it is needed, there is no hardware of checking out overflow or underflow (stack is empty) of stacks — that is all for compiler duty. That means "CAMCOH" has got enough internal register store (40 registers) and direct access for every register, not only for the top of a stack,

which permits to keep many objects (for instance, loop variables) in register, and not in main store.

The most efficient register allocation is a somewhat complicated problem, but modern compilers solve it successfully. One argument more against using registers in computer design is that procedure call works slowly because registers have to be saved as the compiler does not know the register allocation at the moment of call inside of a procedure. "CAMCOH" provides each procedure with independent register enumeration (from top to bottom of the stack) which permits not to save all the registers but only a few number of them just to provide an "appetite" of procedure calculated at compile time. There is no need to restore all the registers immediately after return to the procedure, may be they'll have to be saved once more (this situation can be called "crush"). Our strategy may be described as a following: save the registers when the stack is nearly overflown, restore them when the stack is nearly empty. Such strategy helps to reduce the expenses as possible.

5. A special scheme "pipeline" is foreseen in "CAMCOH" architecture for not to waste the time for fetching the instructions. The "pipeline" buffer has 8 bytes with 4 bytes access to store. The speed of "pipeline" is such that the fetching time can be neglected in all cases except branches.

6. "CAMCOH" is a microprogrammed computer based on section microprocessing sets (like AM2900). A special microprogramming automation system based on Algol 68 was worked out for to implement rather complex "CAMCOH" architecture. This system permitted to prepare microprograms without any microassembler. Algol 68 microprogramming appeared to be so convenient that it was decided to supply every "CAMCOH" with this system and microprogram RAM (4K words of 64 bits, 2K of them containing microprograms of standard instruction set). We have very good experience of improving several applications

more than 10 times after microprogramming some selected program parts.

7. It is generally known that virtual memory makes much easier work of computer scientists but it used very seldom because of implementation difficulties. This problem is solved in "CAMCOH" with the help.of the compiler too. The main store is divided into segments of varied length, the length of each is defined by the compiler. Only the mathematical addresses (segment number and offset in it) can be stored. Mathematical address can be loaded to address stack, at that moment it transforms to physical address with the help of segment table. This is a very typical decision and loading costs rather much (about 10 clocks), but loaded address can be used several times without any additional expenditures. The computer, its operating system and compiler provide immobility of the segment while there is any reference to it from address stack. Compilers optimize usage loading address stack instructions, in particular carry them out of loops. Consider the example:

for i **to** n **do** $a[i] := b[i]$ **od;**

The corresponding object code is:

	load1	% $1 \rightarrow$ int stack	1 byte
	load n	% $n \rightarrow$ int stack	1 or 2 bytes
	loada a	% $a \rightarrow$ addr stack	2 bytes
	loada b	% $b \rightarrow$ addr stack	2 bytes
	begloop end	% if $n < 1$ then goto end	2 bytes
lab	slice $1, 1$	% a in areg 1, i in ireg 1	2 bytes
		% result to addr stack	
	sliceandload $1, 1$	% now b in areg 1	2 bytes
		% value of $b[i] \rightarrow$ int stack	
	:=	% (addr stack):= int stack	1 byte
	endloop lab	% ireg 1+ := 1;	2 bytes

```
            % if ireg 1 <= ireg 0
            % then goto lab
end
```

Note that a repeating part of the loop consists of only 7 bytes!

Address stack is 40 bits wide: 24 bits is a physic address (that's why directly addressed main store may be up to 16 MB) and 16 bits - segment number, it used when physical address is to be transformed back to mathematical one.

8. An opinion has be formed that a HLL program doesn't depend on the computer. For embedded systems it isn't so. Obviously, an expression $a := b + c$ can be equally translated to codes of any specialized computer, but as to real-time systems it are not arithmetical expressions but OS features, technological demands and concrete time characteristics that matters. "CAMCOH" is only a small part of a large industrial technology program and "CAMCOH" architecture was designed with strong connections with other technological parts. Microprogram implementation of main OS functions, large database handling, simultaneous existence of thousands real time processes with fast switching are good examples of such approach.

A.N.Terekhov, Ph.D.
NIIM, St.-Petersburg University
2, Bibliotechnaja sq.
Petrodvorets, St.-Petersburg
198904, Russia

Received May 27, 1992