# AN ARCHITECTURE AND EMPIRICAL RESEARCH OF DEDICATED KNOWLEDGE PROCESSING INTERPRETER

**Ivan V. Savchenko**

***Abstract***: *Developing of architectures of computer systems, which effectively support knowledge processing, remains a relevant problem. Architecture and empirical research of performance of dedicated knowledge processing interpreter are suggested in the paper. The studies were made by using the methods of mathematical statistics. The results proved reasonability of using dedicated knowledge processing interpreters. The knowledge processing interpreter can be used to develop high-performance knowledge processing systems.*

***Keywords***: *knowledge processing interpreter, performance, architecture, processor, instruction set, prototyping board.*

***ACM Classification Keywords***: *C.1.3 Other Architecture Styles*

## Introduction

Knowledge processing systems play an important role in human life and their using is constantly growing. The systems are used in such application areas [Rossitza Setchi at al., 2010]: management and control of production processes; diagnostics, trouble-shooting; robotics; image processing; computer vision; medical systems; monitoring and forecasting of the financial and stock markets, etc.

The volume and complexity of the tasks that solve knowledge processing systems are constantly growing. This causes hardware developers to find ways to create faster, more reliable and at the same time energy-conserving architectures of computer systems. The major tasks in this area are [Stallings, 2010]: increasing processor performance, increasing memory speed and increasing input-output speed of ports. In this paper a focus is concentrated on increasing processor performance task.

Modern knowledge processing systems use such hardware architectures of their processors [Stallings, 2010]: multi-core, multithreading, superscalar and very long instruction word. Today increasing architecture performance is achieved by [Stallings, 2010]: reduction in the size and density of gates; increasing size and speed of cache memory; implementation of changes to architecture and organization of the processor (using various forms of parallelism, instruction pipelining and an integration of additional sets of instructions). However, semantic gap, lack of architectural flexibility, inefficient use of memory and high hardware cost make it necessary to develop dedicated hardware architectures to efficiently support problem of knowledge processing [Kurgaev, 2008].

The first attempts to solve the problem of the semantic gap were made in the 60's and 70's. Their essence was to develop non-von Neumann architecture of computer systems, suitable for direct interpretation of the programs written in high or intermediate level. Well-known works of such famous scientists in this area are: Glushkov V.M., Skurihin V.I., Palagin A.V., Morozov A.O., Malinowski B.M., Boyun V.P., Klimenko V.P., Yakovlev Y.S., Amamiya M., Tanaka Y. et al. However, a huge implementation complexity of these architectures forced to abandon the approach [Amamiya at al., 1993].

Rapid development of modern system-on-a-chip technologies, suggest the developing of a new dedicated architectures for solving problems in subject area. Dedicated system structure (fig. 1) contains the following main modules: general-purpose processor, random-access memory (RAM), problem-oriented coprocessor (dedicated coprocessor), processor and peripheral bus. Depending on the location of the general-purpose processor to dedicated coprocessor, there are two types of coupling [Hauck, 2008]: 1) tightly coupling – general-purpose processor and dedicated logic are placed in the same module (general-purpose processor) or general-purpose processor and dedicated coprocessor are placed on the same processor bus; 2) loosely coupling – general-purpose processor and dedicated coprocessor are placed in various busses (processor and peripheral buses). Selecting of the type of connection depends on the intensity of the data exchange between general-purpose processor and dedicated coprocessor.

The main principle of operation of dedicated architecture is reducing sequential logic with combinational logic (fig. 2) in a system. The system has a number of independent dedicated arithmetic logic units (ALU) which operates simultaneously. Thanks to this adaptation the systems can provide a much more performance. However, this increases the amount of equipment that is involved in the system, and as a result increases energy consumptions [Hauck, 2008].

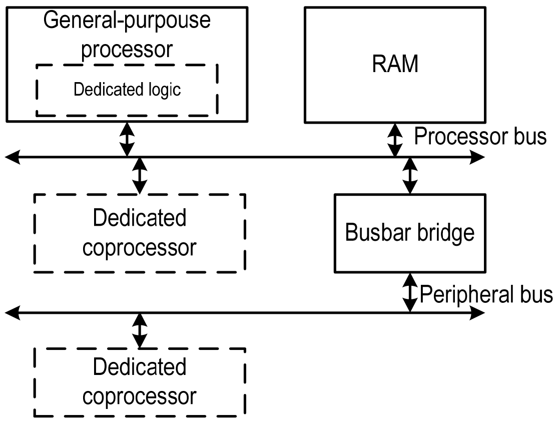

**Figure 1.** Structure of dedicated system

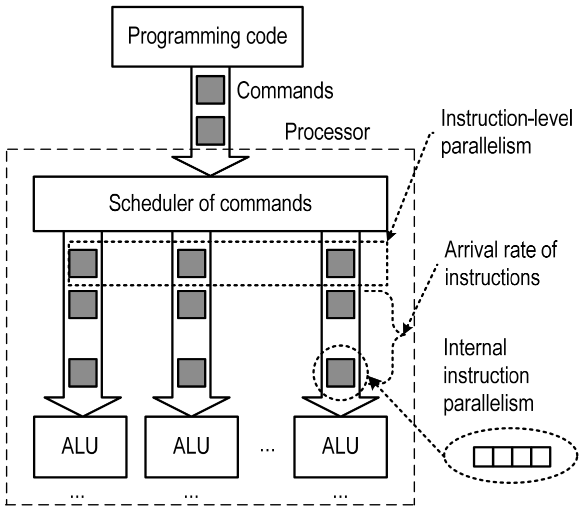
Programming code
Commands
Processor
Instruction-level parallelism
Scheduler of commands
Arrival rate of instructions
Internal instruction parallelism
ALU
ALU
...
ALU

**Figure 2.** The main principle of operation of dedicated architecture

Systems based on system-on-a-chip technologies are widely used for creation of dedicated hardware which increasing performance of knowledge processing systems (systems based on neural networks, genetic algorithms, database search, word-processing, logical conclusion).

In [Poznanovic, 2006] suggested FPGA based non-von Neuman processor which achieving performance through harnessing parallelism of data-flow processing, utilizing multi-core fixed logic technology, increasing chip capacity and higher clock rates and delivering power efficient performance. In [Salapura at al., 1994] suggested an economic architecture (cost of hardware resources) and fast topology map, which allows for large neural network. The proposed topology allows for a variety of models of neural networks. In [Netin at al., 2003] suggested FPGA implementation of an expert system, according to which the knowledge base is transformed into an equivalent hardware network whose nodes are the facts, and the connections between nodes – relationships. Results of experiments show a significant advantage of the expert system. In [Kokosinski at al., 2002] proposed architecture and multi-operand associative processor, which is capable of simultaneously and efficiently perform 16 operations logical search operation, including the operations (= $\neq$, <,>, ≤, ≥). In monograph [Teodorescu at al., 2001] presented broad overview of the architectures of hardware implementations of intelligent systems that using: reconfigurable logic, genetic algorithms, fuzzy logic, neural networks and parallel algorithms. In [Galuzzi at al., 2008] suggested increasing performance of architecture of computer system by expanding instruction set of processor with new instructions.

Among the main limitation of the existing architectures for knowledge processing systems are [Kurgaev, 2008]: 1) the need to implement a large number of operations for symbolic computations; 2) lack of effective knowledge representation in memory and lack of effective information processing of complex structures such as: nested, iterative and recursive.

**Research task** of the paper are suggesting an effective architecture of dedicated knowledge processing interpreter and empirical study of its performance.

## 2 Knowledge processing interpreter functioning principles

The functional structure (fig. 3) of the knowledge processing interpreter includes [Kurgaev, 2008]: compiler, interpreter of knowledge base, interpreter of terminal programs, glossary of terms, input and output arrays.

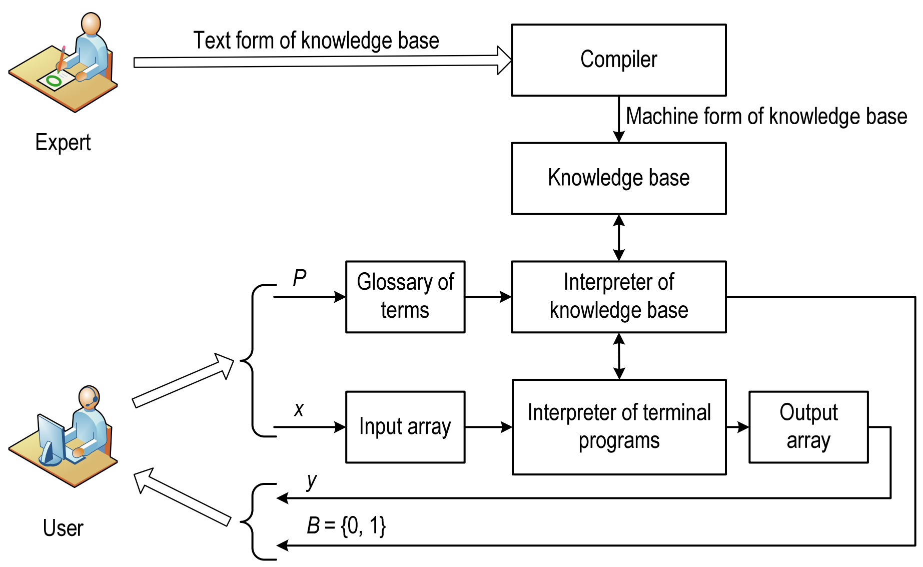
Expert
Text form of knowledge base
Compiler
Machine form of knowledge base
Knowledge base
P
Glossary of terms
Interpreter of knowledge base
x
Input array
Interpreter of terminal programs
Output array
y
B = {0, 1}
User

**Figure 3.** Functional structure of knowledge processing interpreter

The knowledge base of any structurally complex problem can be represented formally [Kurgaev, 2008]:

$$KB = \langle A_{TR}, A_{CON}, C, S \rangle, \quad (1)$$

where *KB* – knowledge base, $A_{TR}$ – a set of terminal programs defined outside the knowledge base; $A_{CON}$ – set of nonterminal concepts; *C* – set of concepts of high complexity; *S* – data structures of concepts in set C which describing the relevant concepts as an alternative or a sequence of some of the concepts, each of which may be information structure iteration, terminal program or constant.

Knowledge base contains related concepts of some subject area. Each concept is defined as an alternative or a sequence of some concepts (concept also may include itself). Each concept can be an iteration of structures of definitions or set definitions. Any simple concept (with lowest level of complexity) is defined in the form of a constant or a procedure (terminal program) and is executed by main processor or dedicated hardware unit [Kurgaev, 2008].

Compiler converts the text representation of the knowledge base in a machine form.

The input and output arrays in the form of ASCII code contain such data accordingly: subject *x* of categorical statement *P*(*x*) which is necessary to prove; subject *y* obtained as a result of this proving.

Interpreter of terminal programs contains a set of terminal programs that execute on the request of the interpreter of knowledge base. Its input data is received from the input array. Output results are generated in the form of the truth value and in the form of data written to the output array.

Glossary of terms contains a set of records, each of which consists of two fields: «Name of concept» and «Address of concept». «Name of concept» in symbolic form contains name of a concept *P*. And «Address of concept» contains an address *A* of a concept *P* in knowledge base. Converting of name of a concept *P* in the address *A* is based on search function $F_{CON}$:

$$F_{CON}(P) \rightarrow A \qquad (2)$$

Interpreter of knowledge base derive categorical statement *P*(*x*) according to the algorithm of interpretation $F_{INT}$:

$$F_{IHT}(x, P) \rightarrow y, B, \qquad (3)$$

where $B = \{0, 1\}$ – the logical result of proving of categorical statement *P*(*x*), *y* – the subject (set of characters) obtained as result of proving *P*(*x*).

The process of the knowledge processing consists of four stages: 1) an expert creates knowledge base in text form; 2) knowledge base is compiled into machine form; 3) user formulates the problem in subject-predicate form *P*(*x*); 4) searching for the problem solution.

## 3 Knowledge processing interpreter architecture

The structure of knowledge processing interpreter, which is based on prototyping board M1AGL-DEV-KIT-SCS of Actel corp., is presented in fig. 4.

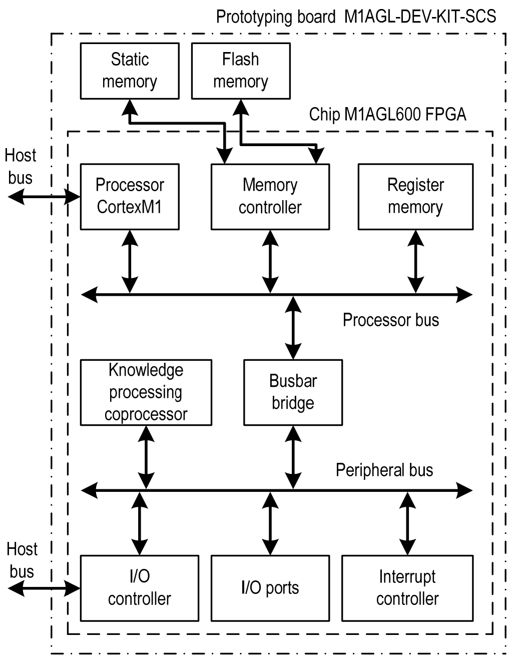

**Figure4.** Structure of the knowledge processing interpreter

The core of the interpreter is a chip M1AGL600 FPGA, which is located on prototyping board (fig. 4). The main modules of the interpreter are: static memory, flash memory, main processor (CortexM1), memory controller, register memory, processor bus (AHB), peripheral bus (APB), knowledge processing coprocessor, busbar bridge, I/O controller, I/O ports and interrupt controller. The interpreter also includes a host computer for problem statement and for reading the solved problem from interpreter. Host computer interacts with the interpreter through host busses. In order to extend an instruction set of the main processor, knowledge processing coprocessor was integrated into the interpreter.

Main processor performs following functions: initialization of knowledge processing interpreter; obtaining problem from host computer and transferring it to knowledge processing coprocessor; transferring result of solved problem to host computer; execution of a terminal programs by request of knowledge processing coprocessor.

Flash memory is intended for permanent storage of: program memory, knowledge base, glossary of terms, input and output arrays. Since flash memory is quite slow, but volatile, a data stored in it only when power is turned off. During the system initialization the data from flash memory is copying to faster memory – static memory.

Static memory is continuously using while the knowledge processing interpreter solving stated problem. And it also stores: program memory, stack of main processor (Cortex-M1), glossary of terms.

Register memory is used by main processor for storing local data (variables, arrays, constants). This provides maximum access to the local data.

Knowledge processing coprocessor performs interpretation of a knowledge base in accordance with the assigned problem stated by the host computer. It also maintains such memories: trace memory, stack memory, input and output array.

Interrupt controller is used for synchronizing the processes of exchanging of commands and data between the main processor and all functional modules of the interpreter. Interrupt controller also performs a major role in an exchange of data between the host computer and the interpreter.

I/O controller performs a function of a high-performance bridge between the interpreter and the host computer.

Timer and I/O ports are used to visually inform user about a current status of the interpreter.

The address space of the interpreter is divided into separated segments, so main processor has access to individual modules of the interpreter. The structure of knowledge processing interpreter memory, based on prototyping board, is shown in fig. 5.

Static memory includes program memory and glossary of memory of terms (fig. 5). Program memory contains program (algorithm) which controls interpreter functioning. The program provides interpreter functioning according to an algorithm (fig. 6). Consider the algorithm in more details.

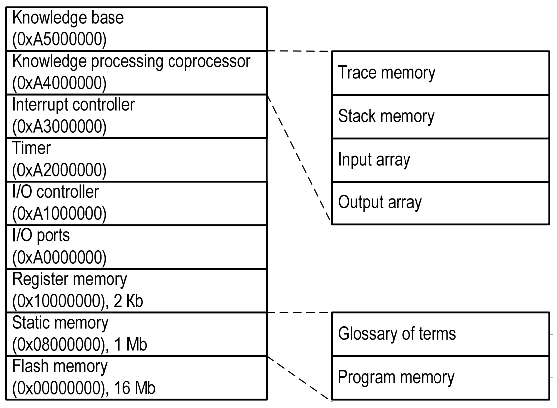
Knowledge base (0xA5000000)
Knowledge processing coprocessor (0xA4000000)
Interrupt controller (0xA3000000)
Timer (0xA2000000)
I/O controller (0xA1000000)
I/O ports (0xA0000000)
Register memory (0x10000000), 2 Kb
Static memory (0x08000000), 1 Mb
Flash memory (0x00000000), 16 Mb
Trace memory
Stack memory
Input array
Output array
Glossary of terms
Program memory

**Figure 5**. Structure of memory of knowledge processing interpreter

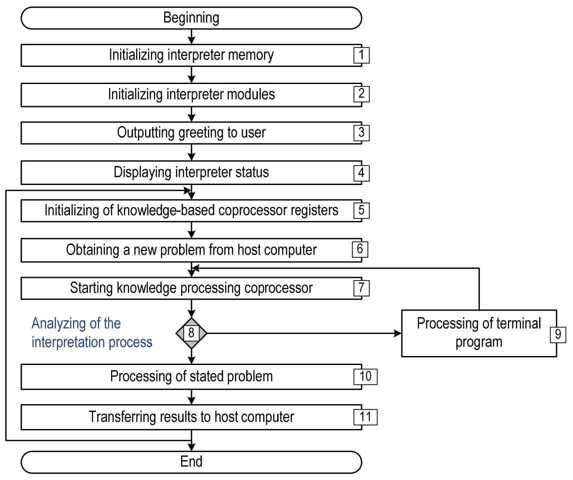
Beginning
Initializing interpreter memory 1
Initializing interpreter modules 2
Outputting greeting to user 3
Displaying interpreter status 4
Initializing of knowledge-based coprocessor registers 5
Obtaining a new problem from host computer 6
Starting knowledge processing coprocessor 7
Analyzing of the interpretation process 8
Processing of terminal program 9
Processing of stated problem 10
Transferring results to host computer 11
End

**Figure 6.** Algorithm of knowledge processing interpreter functioning

**1. Initializing interpreter memory**. The procedure begins by applying a power to the interpreter or after pressing the «RESET» button on the prototyping board. The information that is stored in flash memory is automatically rewriting to a static memory.

**2. Initializing interpreter modules**. Main processor writes specific data to the registers of all functional modules (input-output ports, I/O controller, interrupt controller, I/O ports, timer and knowledge processing coprocessor).

**3. Outputting greeting to user.** Main processor sends the host computer a message that indicates the interpreter is ready for problem execution.

**4. Displaying interpreter status.** Main processor reads data from registers (COUNTER, input and output array) of knowledge processing coprocessor and refers them to the host computer.

**5. Initializing of knowledge processing coprocessor registers.** Main processor initializing the registers ($R_{02}$–$R_{03}$, $R_{06}$–$R_{09}$, $R_{OI}$, $R_{LM}$) to zero in order to prepare of the interpreter to solve next problem.

**6. Obtaining a new problem from host computer**. Main processor receives from the host computer a new problem which comes as three parameters: a function argument, a function name and a mode of interpretation. The function argument (sequence of characters) stores in the input array. Then function name is searching in glossary of terms. If function name is found, then the address of function name in knowledge base is stored in knowledge processing coprocessor register $R_{01}$. In other case, you are prompted to re-enter the correct function name. Registers $R_{10}$ and $R_{11}$ are initializing depending on the mode of interpretation (table 1).

**Table 1**. Interpretation modes

| № | Regime | $R_{10}$ | $R_{11}$ |
|---|---|---|---|
| 0 | Analysis without tracing | 0 | 0 |
| 1 | Analysis with tracing | 1 | 0 |

**7. Starting knowledge processing coprocessor.** Main processor sending «START» message to knowledge processing coprocessor and it becomes solve the stated problem.

**8. Analyzing of the interpretation process.** If next to be done is terminal program, then go to Point 9, otherwise, go to Point 10.

**9. Processing of terminal program.** Knowledge processing coprocessor terminates its work and transfers control to the main processor. When processing of the terminal program is finished, control and results of processing are transferred to the knowledge processing coprocessor and go to Point 7.

**10. Processing of stated problem.** Knowledge processing coprocessor solves an interpretation problem according with its algorithm and current data in its registers and memories.

**11. Transferring results to host computer.** The main processor reads data from knowledge processing coprocessor registers, input, output arrays and forward them to the host computer. Next go to Point 5.

## 4 Knowledge processing coprocessor architecture

Since all functional modules of the knowledge processing interpreter are standard (we can find them in CAD system library), besides knowledge processing coprocessor, then let us discuss its architecture.

Knowledge processing coprocessor interprets knowledge base presented in the form of meta-language.

Knowledge processing coprocessor consists of (fig. 7): knowledge-based processor, knowledge base, trace memory, stack memory, input and output arrays [Kurgaev, 2008].

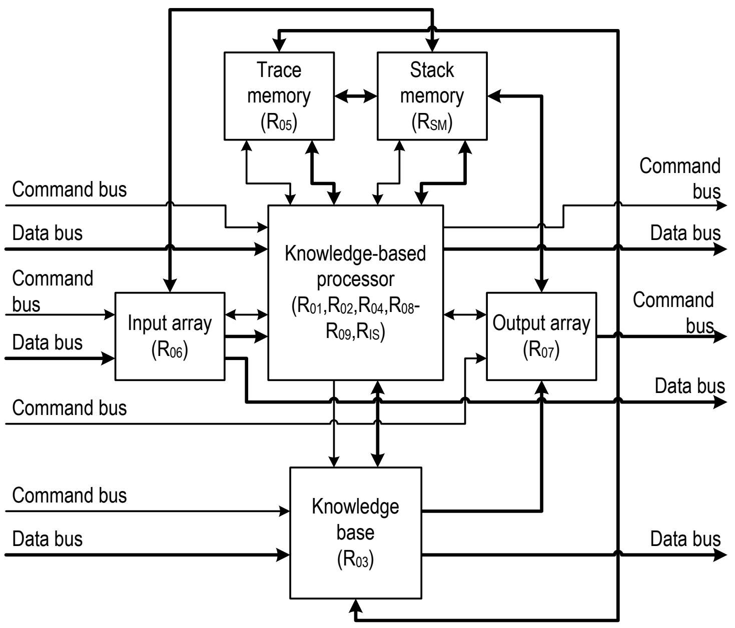

**Figure 7.** Structure of the knowledge processing coprocessor

Knowledge-based processor consists of control unit and processing unit. The control unit is designed as a Moore finite state machine. Processing unit contains such registers: $R_{01}$, $R_{02}$, $R_{04}$, $R_{08}$–$R_{09}$ and $R_{IS}$. Detailed list of knowledge processing coprocessor register list are represented in table 2.

**Table 2.** Set of registers of knowledge processing coprocessor

| Register | | Description | Width, bits |
|---|---|---|---|
| $R_{01}$ | | Name of structure register | 16 |
| $R_{02}$ | | Number words frame register | 16 |
| $R_{03}$ | | Address knowledge base register | 16 |
| $R_{04}$ | | Coordinates trace register | 16 |
| $R_{05}$ | | Top trace register | 16 |
| $R_{06}$ | | Address register in the input array | 16 |
| $R_{07}$ | | Address register in the output array | 16 |
| $R_{08}$ | | Iteration register | 16 |
| $R_{09}$ | | Truth register | 16 |
| $R_{IS}$ | $R_{10}$ | Trace signs register | 1 |
| | $R_{11}$ | Production signs register | 1 |
| | $R_{12}$ | Inversion 1 register | 1 |
| | $R_{13}$ | Inversion 2 register | 1 |
| | $R_{14}$ | Type frame register | 3 |
| | $R_{15}$ | Sign last element register | 1 |
| $R_{SM}$ | | Stack memory register | 16 |

Stack memory contains the current state of interpretation process. States are stored in fixed-length records. Each record contains data about: input and output array index position, trace iteration register, the number of successful iterations and mode of interpretation.

The trace memory contains an array of records with fixed-length, which reflects the progress the interpretation of concepts in the form of derivation tree. Each record can contain data of two types of data: physical address of successfully interpreted alternatives or a number of successful executed iterations.

Knowledge base contains information structure of a knowledge, which consists of a set of frames. Each frame contains head and frame elements that are related with each other by ratio: conjunction, disjunction or iteration.

Head of frame and any frame elements occupy several adjacent locations in memory. The frame elements are located in following memory locations. Head of a frame is the first in this sequence and frame size (number of elements in a frame) can be various. Iteration frame consists of two components: a head and an iterated element. Iterated element is represented by a reference frame, which is the main component of a description of this item. This ensures connectivity of different frames in a single multiply nested structure that allows recursive structures. Difficulty of the description can be arbitrary and limits by resources of a particular implementation.

Instruction set of knowledge-based processor is showed in table 3.

**Table 3**. Instruction set of knowledge-based processor

| № | Command name | Bin value | Description |
|---|---|---|---|
| 1 | NOP | 000000 $_b$ | No operation |
| 2 | INC_PC | 000001 $_b$ | Incrementing $R_{02}$ |
| 3 | TDEEP_PC | 000011 $_b$ | Loading $R_{02}$ to $R_{03}$ |
| 4 | PUSH_REG | 000100 $_b$ | Pushing ($R_{04}$, $R_{05}$, $R_{06}$, $R_{07}$, $R_{10}$, $R_{11}$ and $R_{14}$) into stack memory |
| 5 | POP_REG1 | 000101 $_b$ | Popping ($R_{04}$, $R_{05}$, $R_{06}$, $R_{07}$, $R_{10}$, $R_{11}$ and $R_{14}$) from stack memory |
| 6 | POP_REG2 | 000110 $_b$ | |
| 7 | GET_TRUTH | 000111 $_b$ | Getting $R_{09}$ data from terminal program |
| 8 | SET_TRUTH | 001000 $_b$ | Setting $R_{09}$ to «1» value |
| 9 | INV_TRUTH | 001001 $_b$ | Inverting $R_{09}$ value |

**Table 3 prolongation**. Instruction set of knowledge-based processor

| | | | |
|---|---|---|---|
| 10 | WR_TRACE_ALT | 001010 $_b$ | Recording the address of successfully interpreted alternative into trace memory |
| 11 | WR_TRACE_ITER | 001011 $_b$ | Recording the number of successfully interpreted iterations into trace memory |
| 12 | SAVE_TR_WR_ADDR | 001100 $_b$ | Saving ($R_{04}$, $R_{05}$) data into trace memory |
| 13 | REST_TR_WR_ADDR | 001101 $_b$ | Loading ($R_{04}$, $R_{05}$) data from trace memory |
| 14 | SET_TR_WR_ADDR | 001110 $_b$ | Loading ($R_{05}$) data from trace memory |
| 15 | SET_TRACE_REG | 001111 $_b$ | Setting $R_{10}$ to «1» value |
| 16 | CLR_TRACE_REG | 010000 $_b$ | Setting $R_{10}$ to «0» value |
| 17 | SET_PROD_REG | 010001 $_b$ | Setting $R_{11}$ to «1» value |

| | | | |
|---|---|---|---|
| 18 | CLR_PROD_REG | 010010 b | Setting $R_{11}$ to «0» value |
| 19 | INIT_NUM_ITER | 010011 b | Setting iteration counter register to «0» value |
| 20 | INC_NUM_ITER | 010100 b | Incrementing iteration counter register |
| 21 | DEC_NUM_ITER | 010101 b | Decrementing iteration counter register |
| 22 | LOAD_NUM_ITER | 010110 b | Loading iteration counter register from trace memory |
| 23 | SET_TR_RD_ADDR | 010111 b | Loading $R_{04}$ from trace memory |
| 24 | RD_TR | 011000 b | Reading data from trace memory |
| 25 | SET_KB_WR | 011001 b | Setting knowledge base in writing mode |
| 26 | SET_KB_RD | 011010 b | Setting knowledge base in reading mode |
| 27 | TERMINAL | 011011 b | Requesting terminal program |
| 28 | REQUEST | 011100 b | Requesting for main processor resources |
| 29 | CLR_ALL | 011101 b | Clear all the registers of knowledge processing coprocessor |
| 30 | IDLE | 011110 b | Knowledge processing coprocessor is ready for processing |
| 31 | RESTORE_REG | 011111 b | Restoring ($R_{04}$, $R_{05}$, $R_{06}$, $R_{07}$, $R_{10}$, $R_{11}$ and $R_{14}$) from stack memory |
| 32 | WR_OUT_MAS | 100000 b | Write data to output array |
| 33 | WR_TR_NUL | 100001 b | Write «0» data to trace memory |

A simplified example of an algorithm of knowledge-based processor functioning in a form of directed graph is showed in fig. 8. The algorithm solves only analysis task. Analysis with the tracing and analysis with generation tasks is omitted for simplicity. Consider the algorithm in more details.

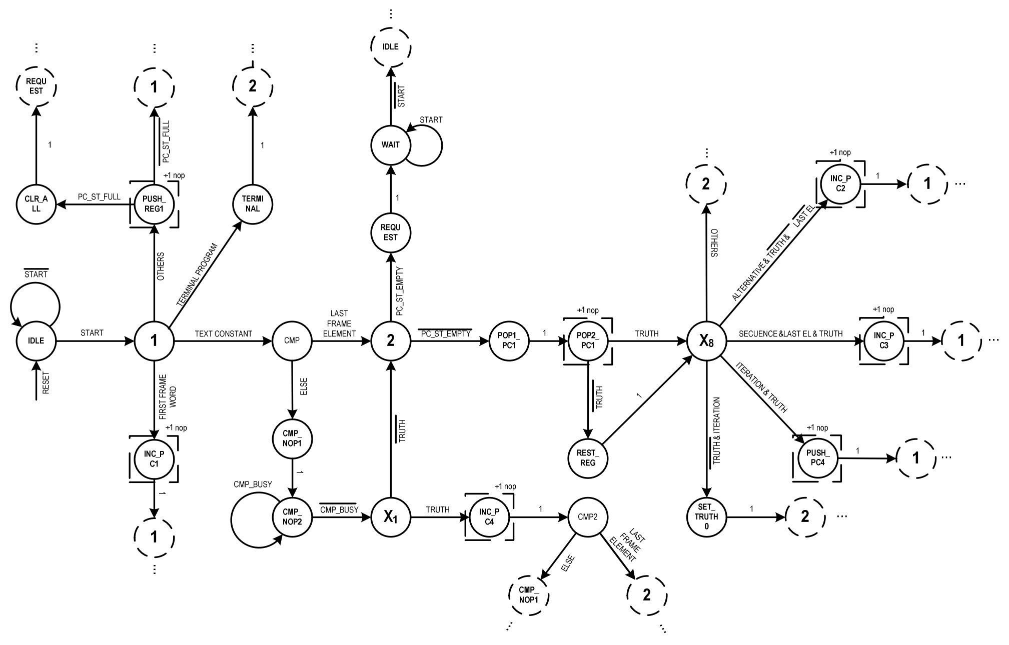

**Figure 8.** Algorithm of knowledge-based processor functioning

**IDLE.** Knowledge-based processor waits for «START» signal from main processor and sends instruction «IDLE». When «START» signal arrives then it goes to the state 1.

**STATE №1.** Knowledge-based processor sends instruction «NOP». If current word of frame is: 1) terminal – then it goes to the state «TERMINAL»; 2) text constant – then it goes to the state «CMP»; 3) first frame word – then it goes to the state «INC_PC1»; 4) else – then it goes to the state «PUSH_REG1».

**PUSH_REG1.** Knowledge-based processor sends instruction «PUSH_REG». Then it sends instruction «NOP». If memory stack is full then it goes to the state «CLR_ALL» else it goes to the state «STATE №1».

**CLR_ALL.** Knowledge-based processor sends instruction «CLR_ALL». Then it goes to the state «REQUEST».

**TERMINAL.** Knowledge-based processor sends instruction «TERMINAL». Then it goes to the state «STATE №2».

**CMP.** Knowledge-based processor sends instruction «TERMINAL». If current frame element is last then processor goes to the state «STATE №2» else it goes to the state «CMP_NOP1».

**CMP2.** Knowledge-based processor sends instruction «TERMINAL». If current frame element is last then processor goes to the state «STATE №2» else it goes to the state «CMP_NOP1».

**CMP_NOP1.** Knowledge-based processor sends instruction «NOP» while the terminal program is executing. When terminal program is executed then it goes to the state «X1».

**STATE №2.** Knowledge-based processor sends instruction «NOP». If memory stack is empty then processor goes to the state «REQUEST» else it goes to the state «POP_PC1».

**REQUEST.** Knowledge-based processor sends instruction «REQUEST». Then it waits until signal «START» is reset. Then it goes to the state «IDLE».

**POP_PC1.** Knowledge-based processor sends a sequence of instruction «POP_REG1», «POP_REG2» and «NOP». If signal «TRUTH» arrived then processor goes to the state «X8» else it goes to the state «REST_REG».

**REST_REG.** Knowledge-based processor sends instruction «RESTORE_REG». Then it goes to the state «X8».

**X1.** Knowledge-based processor sends instruction «NOP». If signal «TRUTH» arrived then processor goes to the state «INC_PC4» else it goes to the state «STATE №2».

**X8.** Knowledge-based processor sends instruction «NOP». If current frame element is «ALTERNATIVE» and signals «TRUTH» and «LAST FRAME ELEMENT» is turned off then processor goes to the state «INC_PC2». If current frame element is «SEQUENCE» and signals «TRUTH» and «LAST FRAME ELEMENT» is turned on then processor goes to the state «INC_PC3». If current frame element is «ITERATION» and signals «TRUTH» is turned off then processor goes to the state «SET_TRUTH0».

**INC_PC1.** Knowledge-based processor sends instruction «INC_PC» and after that sends instruction «NOP». Then it goes to the state «STATE №1».

**INC_PC2.** Knowledge-based processor sends a sequence of instruction «INC_PC» and «NOP». Then it goes to the state «STATE №1».

**INC_PC3.** Knowledge-based processor sends a sequence of instruction «INC_PC» and «NOP». Then it goes to the state «STATE №1».

**INC_PC4.** Knowledge-based processor sends a sequence of instruction «INC_PC» and «NOP». Then it goes to the state «STATE №1».

**SET_TRUTH0.** Knowledge-based processor sends instruction «SET_TRUTH». Then it goes to the state «STATE №2».

## 5 Empirical research of knowledge processing interpreter performance

Empirical research was investigated by comparing an efficiency of software-based knowledge processing interpreter (SBKPI) and hardware-based knowledge processing interpreter (HBKI). HBKI was implemented on the prototyping board M1AGL-DEV-KIT-SCS. SBKI was compiled in an environment Microsoft Visual Studio 2010 using the library Boost/Spirit [Boost Spirit About, 2011] and implemented in two configurations – SBKI2 and SBKI4. SBKI2 operates on a PC with following features: operating system – MS Windows XP Professional 32-bit SP3; processor – Intel Mobile Core 2 Duo T8100 2.10 GHz; processor cores – 2; RAM size – 2GB. SBKI4 operates on a PC with the following features: operating system – Windows Server 2008 R2 Standard SP 1; processor – Intel Xeon CPU E5504 2.00 GHz; processor cores – 4, RAM size – 12 GB.

In order to eliminate the influence of the technological features of the HBKI, SBKI2 and SBKI4 implementations, it is useful to compare not absolute time in seconds but in the number of processor clock cycles.

During of the empirical research was used a knowledge base which contains two concepts: Identifier (4) and Sentence (5). The knowledge base has been implemented in accordance with the syntax of interpreter and loaded into SBKI2, SBKI4 and HBKI. During the empirical research, the experimental data was generated automatically or obtained from a literature. All the data were analyzed in turn on SBKI2, SBKI4 and HBKI and written in a spreadsheet for analysis. The analysis of the experimental data proved that the algorithmic complexity of interpretation algorithm is linear. The main empirical results are shown in fig. 9-10.

*Identifier* $=_{df}$ *Letter (Letter/Numeral);* (4)

*Letter* $=_{df}$ *A/B/C/D/E/F/G/H/I/J/K/L/M/N/O/P/Q/R/S/T/U/V/W/X/Y/Z/a/b/c/d/e/f/g/h/i/j/k/l/m/n/o/p/q/r/s/t/u/v/w/x/y/z;*

*Sentence* $=_{df}$ *NP* $Aux_1$ *VP;* (5)

*NP* $=_{df}$ $Art_1$ $AP_1$ *N* $PP_1$;

*VP* $=_{df}$ *Verb* $NP_1$ $PP_1$ $Adv_1$;

*PP* $=_{df}$ *Prep NP;*

*AP* $=_{df}$ *Adj* $PP_1$;

$Aux_1$ $=_{df}$ *Aux/1;*

$Art_1$ $=_{df}$ *Art/1;*

$AP_1$ $=_{df}$ *AP/1;*

$PP_1$ $=_{df}$ *PP/1;*

$NP_1$ $=_{df}$ *NP/1;*

$Adv_1$ $=_{df}$ *Adv/1;*

*Adj* $=_{df}$ *old/red/slimy/white/new/hungry;*

*Adv* $=_{df}$ *slowly/now/quite_quick;*

*Art* $=_{df}$ *the/an/a;*

*Aux* $=_{df}$ *will/can/might;*

*N* $=_{df}$ *tree/wind/children/toys/toy/box/boy/ball/house/shorts/letter/jon/man/fish/toby/book/dogs/swimmer;*

*Prep* $=_{df}$ *at/in/to/with/out_of;*

*Verb* $=_{df}$ *swayed/put/found/kicked/was reading/gave/saw/read/fed/was/pulled/have_been_reading;*

In fig. 9 are showed the results of empirical research of SBKI and HBKI during interpretation of the concept «Identifier».

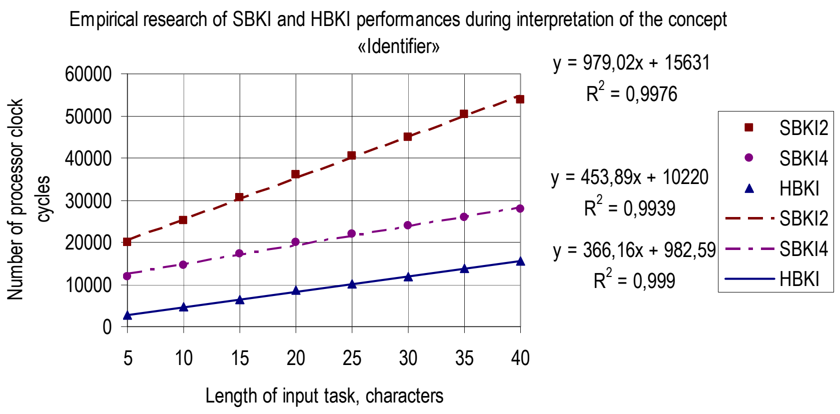

**Figure 9.** Empirical research of SBKI and HBKI during interpretation of the concept «Identifier»

These results represent dependence of the run duration of interpreting task on length of the task. The functions of approximations are got for SBKI2, SBKI4 and HBKI, which are accordingly equal: $y_{\text{SBKI2}}(x) = 979{,}02x + 15631$, $y_{\text{SBKI4}}(x) = 453{,}89x + 10220$ and $y_{\text{HBKI}}(x) = 366{,}16x + 982{,}59$. It is possible to make conclusion, that algorithmic complexity of problem of interpreting of concept «Identifier» is linear. Otherwise: $y_{\text{HBKI2}}(n) = O(n)$, $y_{\text{HBKI4}}(n) = O(n)$ and $y_{\text{SBKI}}(n) = O(n)$. In comparison with functions $y_{\text{HBKI2}}(x)$ and $y_{\text{SBKI4}}(x)$, rate of function $y_{\text{HBKI}}(x)$ is least (366,16 < 453,89 < 979,02). Middle run time of interpretation of concept «Identifier» on SBKI2, SBKI4 and HBKI accordingly equals: 37658,775; 20483,15 and 9221,15 processor clock cycles. The middle advantage of HBKI in comparison with SBKI2 and SBKI4 accordingly equal 4,08 and 2,22 times faster.

In fig. 10 are showed the results of empirical research of SBKI and HBKI during interpretation of the concept «Sentence». Middle run time of interpretation of concept «Sentence» on SBKI2, SBKI4 and HBKI accordingly equals: 27510,556; 15284,556 and 2272,2 processor clock cycles. The middle advantage of HBKI in comparison with SBKI2 and SBKI4 accordingly equal 12,11 and 6,73 times faster.

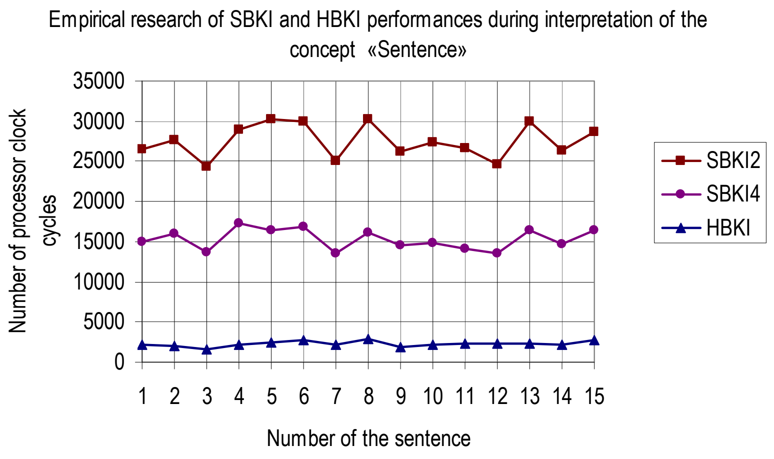

**Figure 10.** Empirical research of SBKI and HBKI during interpretation of the concept «Sentence»

## Conclusion

The empirical researches of dedicated knowledge processing interpreter proved that architecture of modern computer is not effective for knowledge processing. The main reason is semantic gap between the concepts of relations and their objects in the high-level languages of representation of knowledge and concepts of operations and data which are determined by architecture of modern computer.

It is necessary to develop dedicated architectures of computer systems, which will be more efficient for knowledge interpretation. Advisability of this tendency is stipulated by modern system on chip technologies, which allow fast and cheap realization of new dedicated architectures.

It is shown that dedicated knowledge processing interpreter provides advantage in performance in comparison with the programmable interpreters, which function on the base of universal architecture. The middle advantage in performance varies from 2 to 12 times faster.

Advanced research of the paper can be developing of a knowledge processing interpreter with multiple independent processing cores based the suggested architecture.

## Bibliography

[Amamiya at al., 1993] Amamiya M., Tanaka U. Computer architecture and artificial intelligence: Moscow: Mir, 1993, 400 p. (in Russian)

[Andreas Konig at al., 2011] Andreas Konig, Andreas Dengel, Knut Hinkelman, Koicji Kise at al. Knowledge-based intelligent information and engineering systems: 15th International conference, KES 2011 Kaiserslautern, Germany, September 2011, Part I. Copyright © Springer-Verlag Berlin Heidelberg, 2011, pp. 609.

[Boost Spirit About, 2011] Boost Spirit About. http://boost-spirit.com/home/about-2, visited in 2011.09.12.

[Galuzzi at al., 2008] Galuzzi C., Bertels K. The Instruction-set extension problem: A Survey // In Proceedings of The 4th International Workshop, ARC 2008 London, UK, March 2008. pp. 209-220.

[Hauck, 2008] Hauck S. Reconfigurable computing: the theory and practice of FPGA-based computation. Copyright © 2008 by Elsevier Inc., 2008, 908 p.

[Kokosinski at al., 2002] Kokosinski Z., Sikora W. An FPGA implementation of a multi-compared multi-search associative processor // Proceedings of the 12th International workshop, FPL 2002, Montpellier, France, pp. 826–835.

[Kurgaev, 2008] Kurgaev A.F. Problem oriented architecture of computer systems. Kyiv: Stal, 2008, 540 p. (in Russian)

[Netin at al., 2003] Netin A., Roman D., Cret O., Pusztai K., at al. FPGA-Based hardware/software codesign of an expert system shell // Proceedings of the 13th International workshop, FPL 2003, Lisbon, Portugal, pp. 1067–1070.

[Poznanovic, 2006] Poznanovic D.S. The emergence of Non-von Neuman processors // In Proc. of second international workshop, ARC 2006 Delft, The Netherlands, March 2006, pp. 243–254.

[Salapura. at al., 1994] Salapura V., Gschwind M., Maischberger O. A Fast implementation of a general purpose neuron. Proceedings of the 4th International workshop, FPL 94, Prague, Czech Republic. 1994, pp. 175–182.

[Stallings, 2010] Stallings W. Computer organization and architecture. Designing for performance. Copyright © by Pearson Education, Inc., 2010, p. 763.

[Teodorescu at al., 2001] Teodorescu H.N., Jain L.C., Kandel A. Hardware implementation of intelligent systems. Copyright © Physica-Verlag Heidelberg, 2001, 282 p.

## Authors' Information

***Ivan V. Savchenko** – research assistant; National Academy of Sciences of Ukraine Institute of Cybernetics, 40 Glushkov Avenu, 03680, Kyiv, Ukraine;*
*e-mail: savchenko_ivan@ukr.net*
*Major Fields of Scientific Research: Computer systems and components*