doi: https://doi.org/10.15407/emodel.41.06.077
UDC 004.165+004.2+004.27

V.K. Dobrovolskyi, Ph.D., independent CPU architect
(Kyiv, Ukraine, тел. (+38) 0982798517; e-mail: vol.dobrovol@gmail.com)

Microprocessor Based on the Minimal Hardware Principle

A project of the new RISC microprocessor architecture is proposed on the basis of the minimal hardware principle (the MHP RISC processor) targeted on effective parallelization. The notion of instruction group is postulated which is formed by the smart compiler. The header instruction of the group points out how many instructions should be issued in parallel. The concept of the flux as a composite of instruction stream and data flow, supported by certain flux hardware, and used for parallelization on higher levels is developed. Formats of typical instructions and their usage are explained on examples. A new method for the loop control which is applicable to loops with the increasing/decreasing numeric loop variable, and also, a new method for the branch parallelization are proposed. The proposed architecture does not contain simultaneous multithreading, register renaming, instruction reordering, out-of-order execution, speculative execution, superscalar execution, delayed branch, branch prediction which all require much hardware. These all are substituted by the notion of instruction group, concept of flux, special instructions, and strong compiler support.

K e y w o r d s: microprocessor, parallelism, comparisons, loop control, branch parallelization.

Introduction. Parallelization and low power consumption are the main trends in the contemporary microprocessor architecture, and various approaches are implemented to realize it. There are a lot of publications, monographs, and manuals on these problems, e. g. [1—5]. Within a single processor (uniprocessor) a substantial raise in throughput is achieved due to simultaneous multithreading (SMT), register renaming, instruction reordering, out-of-order execution, speculative execution, superscalar execution, delayed branch, branch prediction which all require much hardware. Some of these approaches indirectly improve parallelization. A particular case of a uniprocessor architecture is the very long instruction word (VLIW) architecture which has been successful for the specialized processors for image and graphical data processing etc, but failed as the general purpose processor architecture. All above-listed fa-

cilities and units are very complex hardware that means larger transistor count, and energy consumption.

A project of the new microprocessor architecture is proposed based of the minimal hardware principle (the MHP RISC processor) which overcomes the mentioned shortcomings. The over-listed hardware is removed and substituted by other more simple facilities, and specially designed instructions. The main means are the notion of the instruction group, and the concept of the flux. The instruction group is formed by the smart compiler. The header instruction of the group points out with its 4-bit field how many instructions up to 16 should be issued in parallel. For internal instructions of the group the mentioned field is zero. The flux is a composite consisting of instruction stream and data flow, supported by certain flux hardware, that includes the register file, program counter, stack with stack register, program state record (word) etc. The strong compiler support is necessary, resembling that for the VLIW architecture, though essentially simpler. The uniprocessor must have several fluxes and multiple functional units. Fluxes borrow functional units from their processor pool. Pipelining is preserved. Time sharing (multitasking) remains. Cache memory, interrupt mechanisms, I/O and the like are without perceptible changes.

The proposed microprocessor architecture may be provided with accelerators that use their native instruction sets, and process data in parallel on multiple registers organized as vectors. Also the GPUs may be attached. There are no problems with allocation of multiple uniprocessor cores on a single die.

The proposed processor has the register file containing hardware 32 64-bit registers. The registers are enumerated r0, r1, r2, …, r31, thus pointing out the absolute register positions (addresses) in the register file, i. e. the sequential numbers of registers. The designations R1, R2, R3, and R4 reference the register fields in the instruction formats. There are the following register data types: 64-bit integer, 64-bit floating point, 128-bit floating point (optionally). The bit/byte/dibyte data types may occupy one register, two or four concatenated registers, i. e. occupying 64, 128, or 256 bits of space, the last two data types being optional.

Instruction group. One of the basic notions of the proposed microprocessor architecture is the notion of the instruction group. The first instruction of the group is the header grouping instruction, other instructions are internal ones. The header grouping instruction in each group points out with its special 4-bit “parallelizing” operand field F0 (Fig. 1) how many instructions up to 16 should be issued in parallel including the header instruction. For other instructions of the group the mentioned field is zero. A group may contain one instruction only (one-instruction group) in which the F0 field is zero. The two-

instruction instruction group has the header instruction with F0 equal to one. The instruction group containing 16 instructions has the F0 field equal to 15. The most of the instructions have field F0, in particular the arithmetic, logical, and move instructions. Of course, the header grouping instruction performs its basic role to execute arithmetic or logical operation. The rest of instructions are, generally, instructions for transfer of control. Thus, availability of a header grouping instruction is guaranteed.

It is proposed to form instruction groups by the smart compiler. The compiler explores the large enough fragments (windows) of the source program code, and extracts all possible parallelism. The compiler secures the absence of interdependencies between data inside each group and between groups. The smart compiler reforms the source program text by reordering the source instructions. The formed instruction groups may be enclosed in parentheses. It is the instruction level parallelism (ILP) maintained both by the compiler and hardware. The actual effect of parallelization depends on the availability of multiple functional units. Thus, in the case of shortage of functional units the issue of a group may happen in two or more machine cycles. Distantly the proposed approach resembles a peculiar VLIW processor with variable number, one to four, of operative fields [6].

Concept of flux. The flux is defined as a composite that includes the software and hardware components. From the program point of view the flux is a stream of instructions and the flow of processed data [7]. These streams and flows are maintained by the flux hardware that includes the register file, the register with the program status record (PSR), and the instruction buffer. The PSR contains: program counter register (1); stack pointer register with address of the stack in the main computer memory (2); various state and interrupt flags (3). For effective work the uniprocessor should have at least two fluxes, i. e. the processor should be the multi-flux one. The main computer memory and pipelined functional units are reckoned as common resource for all fluxes, and are used on request. Fluxes may have either individual L1 instruction caches, or a common L1 instruction cache for the whole uniprocessor. A flux looks like a partial processor. Also a flux may be defined as a channel, or window, in which a program or process executes, using a register file and PSR, and borrowing the necessary functional units from their pool.

The maximal number of fluxes in a uniprocessor may not be larger than eight – the width of datapaths is a limiting factor. The hardware discerns fluxes through their 2-, or 3-bit flux distinctive labels. The functional units remember from which flux data to process are received, and to which flux the results should be returned. A flux maintains either a single program process, or a number of processes in the time-sharing mode. Also the pieces of a pro-

Opcode	F0	R1s	R2s	R3s	R4d
8	4	5	5	5	5

a

Opcode	F0	F1s	R2s	R3s	R4s	R5d
8	4	4	4	4	4	4

b

Fig. 1. General instruction format: four 5-bit fields for registers (*a*), five 4-bit fields for registers (*b*)

perly designed program may execute in different fluxes in parallel, interac-ting between each other. This is maintained by the operating system. The further parallelization may be expanded by using multiple cores. The fluxes are at programmer's disposal, securing the full access to them by means of the OS, and the programmer may set access to the necessary flux in a specified core.

Each flux is provided with an instruction buffer as a small and very fast intermediate storage where the instruction groups are accumulated to issue them into functional units in parallel. The instruction buffer has at least two instruction slots, each slot accumulates an instruction group with up to 16 instructions. The flux control unit fills the first slot and simultaneously schedules the instruction group from the second slot onto the functional units. Then the slots change their purpose. Instructions came into buffer in a sequential stream, but output of instructions is performed in groups. The hardware controls the completeness of issued instruction groups. The concept of the flux only remotely resembles the notion of multithreading [8, 9].

Instruction formats. All instructions are fixed 4-byte length, i. e. are half-word, they may have different numbers of operand fields. The basic format has four register operands with 5-bit register operands, thus, the register file has 32 registers, whereas the second format has five register operands with five 4-bit registers. Further only the former format is considered. It should be mentioned that 4-bit field F0 for the grouping of instructions may not be considered as an “expense” of bits, for this field arises naturally as an “surplus” over bits for the operation code and registers. Fig. 1 shows the mentioned two instruction formats (numerals mean the widths of operand fields).

The Opcode occupies the 8-bit operation code field. The 4-bit operand F0 in the header instruction with value 1 to 15 points out the number of instructions in the instruction group in addition to the header instruction; for internal instructions of the instruction group field F0 is equal to zero. Registers R1s, R2s, and R3s are used mainly as source registers, register R4d is a destination

register. These register fields contain the ordinal numbers of registers r0 to r31 in the register file. Fields may merge to give place for immediate constants sometimes including the field F0.

The 8-bit operation code field means 256 instructions in the instruction set. In actual fact, there are more instructions, e. g. there are about 150 multimedia instructions. To overcome this contradiction it is rational to implement several instruction sets in one microprocessor, and an instructions that switch instruction sets. An additional instruction set may include a subset of the most used instructions, plus, e. g. multimedia extension instructions or vector instructions. For distinction instructions from different instruction sets must have a suffix in their internal representation. The 2-bit suffix manages with four instruction sets.

Examples of instructions. Here are some examples of characteristic instructions with particular formats. Arithmetic instruction for combined multiply and add is very useful in two cases:

1) scalar multiplication of the two arrays (vectors) for floating point data;

2) evaluation of the current integer index for two-dimensional array.

The evaluation formula has the appearance: R4d = R4d + R2s * R3s, where the summation is algebraic. Register R1s is not used, registers R2s and R3s are source registers, register R4d is the destination register which accumulates all multiplied pairs.

Frequently arithmetic operations go in chains, and some operation succeeds one on just computed data. It is reckoned that this occurs for about four-in-ten of arithmetic operations. Therefore, it is reasonable to implement instructions which performs two arithmetic operations in consecutive order. Such instructions use four-operand format. The instructions compute the floating point numbers. Barely it is necessary to deal with the integer numbers. These instructions economize two machine cycles for the write into and read from the registers. The intermediate result is held in the functional unit, and is not accessible. The operands R1s, R2s, and R3s are the source registers; the operand R4d is the destination register. Variants of the instructions performances are the following:

$$R4d = R1s * (R2s + R3s);\ R4d = R1s * (R2s - R3s);$$

$$R4d = R2s * R3s + R1s;\ R4d = R2s * R3s - R1s;$$

$$R4d = R1s + R2s + R3s;\ R4d = R1s + R2s - R3s;\ R4d = R1s - R2s - R3s.$$

As it is seen the arithmetic operations are three, or four-operand. The two-operand arithmetic instructions are not present in the proposed architecture,

Opcode	19-bit Immediate constant	R4d
8	19	5

Fig. 2. Arithmetic instructions with an immediate constant as an operand

i. e. a source register is not used as a destination one. For the integer division the operand R1s is used for the remainder.

Arithmetic instructions with an immediate constant as an operand use merged fields except the field for destination register (Fig. 2).

The Opcode sets one of two arithmetic operations: addition or multiplication. The source operands are the 19-bit immediate constants, integer, or floating point. Register R4d which at first is the source one, after the operation becomes the destination register. The floating point immediate constant has the standard 64-bit structure (one bit for sign, eight bits for exponent) but with mantissa cut to 10 bits. When the immediate constant is -1 the multiplication operation changes the sign of the data in the destination register to the opposite. When the immediate constant is zero the multiplication operation zeros the destination register.

There are eight logical instructions: logical addition (OR), logical multiplication (AND), logical exclusive OR (XOR), logical inversion (LINV), logical shift right (LSR), logical shift left (LSL), logical rotate right LRR), logical rotate left (LRL). The most of instructions are four-operand. The first four instructions use register R1s for the mask. The logical operation are performed on the 64-bit data in registers.

The Jump instruction has merged 24-bit constant displacement operand that is able to transfer unconditional control in the range -8388607 to 8388608 in the 4-byte instruction length measure. The zero value is not permitted. The displacement is added algebraically to the program counter (PC). When the jump instruction does not cover the larger address space the long jump instruction is used in which the R4 register contains a base value filled in by the compiler. The 19-bit constant displacement operand embraces the range -262143 to 262144. The zero value is meaningful. The R4d register and the displacement are added algebraically to the PC.

Comparisons and branches. The comparison instruction compares two magnitudes, m1 and m2, of the same data type. The result of comparison is yielded by the hardware. It is so called logical result of comparison, and it is contained in two 1-bit logical condition flags N and Z in the PSR. The branches are classified into general branches with condition flags Z and N, and loop branches purposed to control loop operations with condition flags NL and ZL. Designations R3m1 and R4m2 stand for registers R3 and R4 containing compared magnitudes.

If two compared magnitudes m1 and m2 are equal, then logical flag Z = '1'b, otherwise Z = '0'b. If the first magnitude m1 is less than the second magnitude m2, then flag N = '1'b, otherwise N = '0'b, i. e. magnitude m1 is greater than magnitude m2. Both flags Z and N are mutually dependable. The same is for logical flags ZL and NL. The logical results of comparison of two magnitudes are shown in the Table 1.

The comparison of the integer numbers, or the bit or byte strings does not evoke any problems. Comparison of floating point numbers may produce a "not number" resulting difference, located in the narrow inaccessible neighborhood of the zero. In such a case it is proposed to take the compared magnitudes as equal. That will help to realize many algorithms based on floating point numbers.

To use the logical result of a comparison for transfer of control the notion of comparison expression (CE) and its value is introduced. The value is calculated, and is either true or false, predetermining the further operations in a program. There are 6 options to calculate, and accordingly to fulfill the necessary transfer of control. The options are interpreted by the programmer, and are used to select single of two branch paths stipulated by the algorithm, thus, changing the instruction stream. The options are shown in Table 2. Some options are supported by the logical operations OR, or AND when it is necessary to use both logical condition flags. The given material concerning comparison is in compliance with the IEEE-754 standard.

Table 1. **Meanings of the Z and N logical flags after comparison**

Logical results of Z and N flags	Interpretations
Z = '1'b AND N = '0'b	m1 is equal (=) to m2
Z = '0'b AND N = '1'b	m1 is less than (<) m2
Z = '0'b AND N = '0'b	m1 is greater then (>) m2

Table 2. **Interpretation of the logical condition flags**

No	Logical expression with Z and N flags	Interpretations
1	Z = '1'b	m1 is equal (=) to m2
2	N = '1'b	m1 is less (<) than to m2
3	Z = '1'b or N = '1'b	m1 is less than or equal (≤) to m2
4	Z = '0'b and N = '0'b	m1 is greater (>) than m2
5	Z = '1'b or N = '0'b	m1 is greater than or equal (≥) to m2
6	Z = '0'b	m1 is not equal (≠) to m2

Opcode	F0	Not used	R3m1	R4m2
8	4	10	5	5

Fig. 3. Pure comparison instruction format

Opcode	CE	11-bit Displacement	R3m1	R4m2
8	3	11	5	5

Fig. 4. Comparison and branch instruction format

There are pure comparison instructions among others, i. e. instructions without the accompanied transfer of control. They compare three register types: 64-bit integer, 64-bit floating point, or 128-bit floating point. The compared data must be of the same data type, otherwise the nonsensical and not controlled situation arises. The instruction intended for ordinary comparison fills in the logical flags Z and N, the instruction intended for the loop control fills the logical flags ZL and NL in the PSR. Such instructions have the format shown on Fig. 3.

The comparison instructions format combined with the transfer of control is shown on Fig. 4. For each of register data types there are different instructions.

The instructions perform comparison of two magnitudes with subsequent transfer of control. The r3m1 and r4m2 are registers with the compared magnitudes of the same data type. The CE field contains the reference on the 3-bit comparison expression. The 11-bit Displacement operand contains an immediate constant in the range –1023 to 1024 in the instruction measure to add to the program counter. If the result of comparison stipulated by the condition operand CE is true, then the control is made by using 11-bit Displacement, otherwise the transfer to the next instruction takes place.

Instruction format for the conditional transfer of control directly stipulated by the comparison expression (CE) is shown on Fig. 4.

If the result of comparison stipulated by the comparison expression CE is true, then the transfer of control is made by using 21-bit Displacement in the range –1048575 to 1048576, the measure is given in the number of instructions; otherwise the transfer to the next instruction takes place. The Displacement is added to the program counter.

A method of loop control. The proposed method is applicable to loops with the loop structure like: for i = ms to me step st; <loop body>; endfor; i. e. where the increasing/decreasing numeric loop variable (parameter) is used. The loop control is realized with the help of the two instructions shown on Fig. 4, and on Fig. 5. The loop condition evaluation is made by the instruction on Fig. 6.

Opcode	CE	21-bit Displacement
8	3	21

Fig. 5. Instruction format for conditional transfer of control stipulated by the comparison expression

Opcode	F0	Not used	R2st	R3ms	R4me
8	4	5	5	5	5

Fig. 6. Structure of comparison instruction for loop control with usage of step parameter

The registers R3ms and R4me contain the comparable magnitudes ms and me with the start and end values. The register R2st contains the step parameter of the loop. Before the comparison register with R2st step is added algebraically to the register R3ms. The instruction may be placed in any location of the loop body, but so that several other instructions were before the end of the loop body. This provides for the logical result of comparison becomes known at the end of the loop. Logical result of comparison fills in the loop condition flags ZL and NL. The flags ZL and NL are assigned the initial values before the loop. The increase/decrease of the loop parameter takes place at the end of the loop as usual. The comparison instruction may execute concurrently with other instructions being a member of the instruction group, and the field F0 may be used for the header grouping instruction. For comparable magnitudes of different types (integer, 64-bit and 128-bit floating point) different instructions are provided for.

The instruction for the conditional transfer of control (see Fig. 5), directly stipulated by the comparison expression, is located at the end of the loop body. It secures the transfer of control to the beginning of the loop, or to go out of the loop. The described case of the loop is where the condition evaluation is made at the end of the loop body. Other kinds of loop made such a test at the beginning of the loop body. Then the instruction for the conditional transfer of control is the first instruction of the loop body, and the comparison instruction may be placed as the next. The smart compiler organizes described loop constructs.

Parallelization of branches. The parallelization of branches means avoidance of delays caused when the resolving branch condition occurs. It is proposed to merge the instructions of initial parts of two branch paths in the interleaving stratified manner to create a combined instruction stream to execute in parallel economizing machine cycles [7]. After the resolving of the branch condition the transfer of control to the rest of the true branch path takes place. The method is illustrated on Fig. 7.

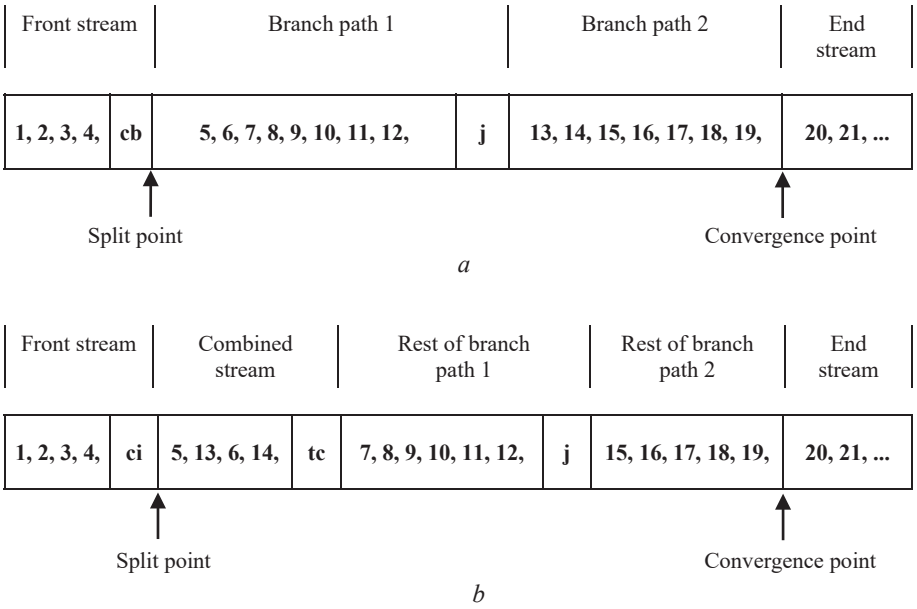

Fig. 7. Exemplary scheme of parallelization of branched instruction stream: *a* – traditional form of the split of instruction stream in two branch paths; *b* – transformed instruction stream having combined instruction stream. Designations: cb is compare and branch instruction; j is jump instruction transferring control after the last instruction of branch path 1 to convergence point; ci is condition instruction that fills the condition flags only; tc is transfer of control instruction using comparison flags. Numerals are the serial numbers of instructions. Front stream – the initial flow of branching instructions; End stream – finishing stream after branching; Branch path 1, 2 – two alternative branch paths; Split point – point of branching; Convergence point – point of confluence; Combined stream – stratified instruction stream; Rest of branch path 1, 2 – the rest of the instructions in the two alternative branches after removing some of the initial instructions.

Traditional structure of the branch construct is shown in Fig. 7, *a*. The front instruction stream ending with the compare and branch instruction is split into two paths which are located in the memory one after another. When the branch condition is resolved the transfer of control takes place either to the branch path 1, or to the branch path 2, the choice being defined by the algorithm. The jump instruction in the end of the branch path 1 transfers control to the convergence point, to the non-split end stream. The split point is a location in the non-split instruction stream after the instruction which causes the branch (e. g., comparison one). The convergence point is a location before the instruction to which both branches transfer the control after the branching is fulfilled. These definitions are abstractions for convenient description of the branch occurrence.

A drawback of the described branch construct is a delay after the comparison instruction. Contemporary microprocessors have a special complex hardware which inserts some other instructions just after the split point, thus avoiding processor stalls.

It is proposed new branch construct to parallelize branches. This is shown as an example on Fig. 7, *b*. The front instruction stream is completed with the pure comparison instruction without transfer of control which fills the condition flags in the PSR to use in the sequel. Further the combined instruction stream follows, which consists of some initial instructions from two branch paths merged in the interleaving manner. The combined stream is created by the smart compiler. The last instruction of the combined stream is the transfer of control instruction that uses the information from the condition flags left by the pure comparison instruction after the logical result of comparison is solved and known. After that the transfer of control is carried out either to the rest of the branch path 1, or to the rest of the branch path 2, and further to the convergence point, i. e. to the end stream. Instructions in the combined stream should be mutually independent, and not change the variables which secure correctness of the algorithm programmed. This is verified by the compiler.

The advantages of the proposed method are avoidance of pipeline stalls, utilization of the general register file only, and usage of the same program counter. The instructions of the combined instruction stream, comparison instruction, and instruction for transfer of control may be members of instruction groups, executing in parallel. The proposed method is the deterred branch approach, and is different from the well known delayed branch approach. No special hardware is needed. There are some peculiarities, e. g., the instructions like $s = s + a * b$ are not permitted in the combined instruction stream as they may violate the correctness of the algorithm.

Conclusion

The proposed project of the general purpose microprocessor architecture is a deep deviation from the habitual guidelines and principles of processor designing, overcoming much stereotypes. For the sake of it the notion of the instruction group is developed, the groups being formed by the smart compiler. The instruction group permits to extract all possible parallelism from the large source program text instruction window, and to execute it in parallel. The concept of the flux, which is developed as a composite that includes the software and hardware components within the scope of the uniprocessor, permits convenient execution several program streams concurrently. The fluxes are at programmer's disposal securing the full access to them by means of the OS.

The minimal hardware principle is postulated on which the proposed microprocessor architecture is founded. The architecture does not use SMT, register renaming, instruction reordering, out-of-order execution, speculative execution, superscalar execution, delayed branch, branch prediction which all require very complex hardware units. These all are substituted by the notion of instruction group, concept of flux, specially designed instructions, and all this gives a considerable economy of hardware.

The method of loop control, applicable to loops where the numeric loop variable is used, replaces the widely used branch prediction in contemporary processors, and needs no additional hardware. The method of the parallelization of ordinary branches ensures a sort of linearization of the instruction stream avoiding extra machine cycles.

The proposed microprocessor architecture ensures less hardware, higher performance, less power, less cost, no vulnerabilities, it secures effective parallelization both on the instruction set level, and on the higher levels.

REFERENCES

1. Dumas II, J.D. (2017), Computer Architecture. Fundamentals and Principles of Computer Design, Taylor & Francis Group.
2. Stallings, W. (2013), Computer Organization and Architecture. Designing for Performance, Ninth edition, Pearson Education.
3. Patterson, D.A. and Hennessy, J.L. (2009), Computer Organization and Design. The Hardware/ Software Interface. Fourth edition, Morgan Kaufmann Publishers.
4. Melnyk, A.O. (2008), Architecture of Computer. Manual, Lutsk regional printing, Ukraine.
5. Sima, D., Fountain, T.J. and Kacsuk, P. (1997), Advanced Computer Architectures: A Design Space Approach, Addison-Wesley.
6. Tremblay, M., Chan, J., Conigliaro, S.W. and Tse, S.S. (2000), "The MAJC Architecture: A Synthesis of Parallelism and Scalability", IEEE MACRO, November-December, pp. 12-25.
7. Dobrovolskyi, V.K. (2018), "Microprocessor with Explicit Parallelism'', the Proceedings of SIMULATION-2018, September 12-14, 2018, Kyiv, Ukraine, pp. 135-138. ISBN 978-966-02-8587-3
8. Eggers, S.J., Emer, J.S., Levy, H.M., Lo, J.L., Stamm, R.L. and Tullsen, D.M. (1997), "Simultaneous Multithreading: A Platform for Next-Generation Processors" IEEE Micro, pp. 12-19.
9. Lo, J., Eggers, S., Emer, J., Levy, H., Stamm, R. and Tullsen, D. (1997), "Converting Thread-Level Parallelism Into Instruction-Level Parallelism via Simultaneous Multithreading", ACM Transactions on Computer Systems, pp. 322-354.

Received 15.07.19

В.К. Добровольский

МИКРОПРОЦЕССОР ОСНОВАННЫЙ
НА ПРИНЦИПЕ МИНИМИЗАЦИИ АППАРАТУРЫ

Предложен проект микропроцессорной архитектуры, основанной на принципе минимизации аппаратуры и ориентированной на эффективную параллелизацию. Постулируется понятие группы машинных инструкций, формируемой интеллектуальным компилятором. Головная инструкция группы указывает, сколько инструкций группа содержит для параллельного исполнения. Разработана концепция флакса как агрегата, состоящего из потоков инструкций и данных, поддерживаемых аппаратно. Флакс предназначен для параллелизации на более высоких уровнях. Форматы характерных инструкций рассмотрены на примерах. Разработан новый метод управления циклами, имеющими числовой параметр, и новый метод параллелизации ветвлений. Предлагаемая архитектура не содержит одновременной многопоточности, переименования регистров, переупорядочивания инструкций, внеочередного исполнения, упреждающего исполнения, суперскалярного исполнения, отложенного ветвления, прогнозирования ветвлений, всего того, для чего требуется сложная аппаратура. Все это замещается понятием группы инструкций, концепцией флакса, специальными инструкциями и существенной компилятивной поддержкой.

К л ю ч е в ы е с л о в а: микропроцессор, параллелизм, инструкции сравнения, управление циклами, параллелизация разветвлений.

В.К. Добровольський

МІКРОПРОЦЕСОР ЗАСНОВАНИЙ
НА ПРИНЦИПІ МІНІМІЗАЦІЇ АПАРАТУРИ

Запропоновано проєкт мікропроцесорної архітектури, заснованої на принципі мінімізації апаратури і орієнтованої на ефективну паралелізацію. Постульовано поняття групи машинних інструкцій, яка формується інтелектуальним компілятором. Головна інструкція групи вказує, скільки інструкцій група містить для паралельного виконання. Розроблено концепцію флакса як агрегата, що складається з потоків інструкцій та даних, підтримуваних апаратно. Флакс призначено для паралелізації на більш високих рівнях. Формати характерних інструкцій розглянуто на прикладах. Розроблено новий метод управління циклами, які мають числовий параметр, і новий метод паралелізації розгалужень. Запропонована архітектура не містить одночасної багатопотоковості, перейменування регістрів, переупорядкування інструкцій, позачергового виконання, випереджувального виконання, суперскалярного виконання, відкладеного розгалуження, прогнозування розгалужень, всього того, що потребує складної апаратури. Все це заміщується поняттям групи інструкцій, концепцією флакса, спеціальними інструкціями і істотною компіляторною підтримкою.

К л ю ч о в і с л о в а: мікропроцесор, паралелізм, інструкції порівняння, управління циклами, паралелізація розгалужень.

DOBROVOLSKYI Volodymyr (Dobrovolsky in some publications) Ph.D., graduated in 1961 from Lviv Polytechnic Institute, Ukraine, in technology of machine building, received a Ph.D. in mathematical economics from Institute of Economics of National Academy of Sciences of Ukraine. The research interests are the CPU and microprocessor architecture, the mathematical modeling of economy, and energy economics.