DOP – A CPU CORE FOR TEACHING BASICS OF COMPUTER ARCHITECTURE

Milos Becvar, Alois Pluhacek and Jiri Danecek

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague,

Abstract: A simple 16-bit processor core called DOP and its teaching environment is presented. The DOP processor illustrates the basic principles of computer organization and is therefore used in the introductory hardware course. Its major features are simplicity, availability of an FPGA implementation and a C compiler. This paper presents the description of the core, HW and SW tools and teaching methodology.

1. INTRODUCTION

An introductory computer hardware course should teach students to the fundamental principles of computer internal functionality. Students, who are familiar with programming in high-level languages, are required to understand the interaction between a processor, a memory and I/O devices, an internal organization of processor, computer arithmetic and basics of digital design. Our experience has shown that it is not an easy task for most of them. The functionality of the processor executing instructions seems to be something almost mythical and totally unrelated to intuitive execution of a program in high-level language.

It is obvious that we have to provide some practical experience helping to understand these abstract principles. One common way is to let students to create a program in high-level language, which simulates a simple processor. However, this approach is not good for illustrating the relation between high-level languages, compilers, program in assembly language and actual "binary" program executed by processor.
Another approach uses visualization simulators and HW emulators (Bruschi, 1999; Yurcik *et al*, 2001; Brorson, 2002; Ellard *et. al,* 2002).

We present a simple 16-bit processor core called DOP that is currently used in our introductory computer organization course. The processor is simple, yet fully operational, and could be used in the embedded applications, which do not require an excessive computing performance. The DOP processor core was developed at our department together with various SW and HW visualization tools (Danecek *et al.*, 1994a).

The goal of this paper is to describe this processor core and its learning environment for teaching basics of computer organization. The paper is organized as follows - section 2 outlines the introductory course and characterizes the students, section 3 describes the DOP processor core, section 4 describes the SW and HW tools supporting this processor and finally section 5 outlines the use of the DOP in our introductory course.

2. COURSE REQUIREMENTS

The introductory computer organization is a one-semester course, which is mandatory for all undergraduate students of 3rd year of computer science and engineering. Almost 200 students take this course every year. The course covers the basic principles of computer functionality, the data representation, computer arithmetic and the controller design. Furthermore it introduces basics of practical digital design.

Students have relatively strong background in high-level languages and assembly language of x86 and are mostly SW-oriented. The majority of students thoughts that the only computer is an Intel x86 PC. The minority of students has experience in implementing simple digital circuits and wants to choose HW specialization in graduate study.

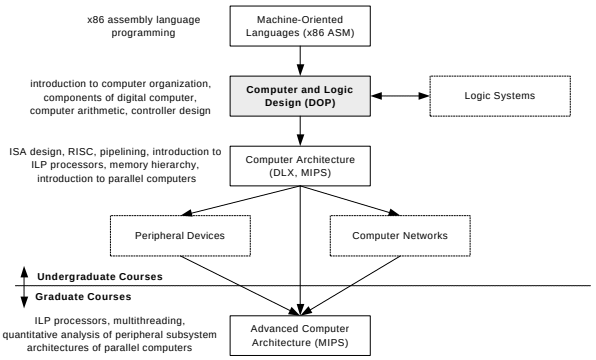

Fig.1. Computer architecture course flow at CTU

Figure 1 shows the place of "Computer and Logic Design" course in the overall Computer Architecture flow at Czech Technical University. It also outlines the main topics covered at each level and a reference platform.

The main goal of "Computer and Logic Design" course is to explain the components of digital computer and functionality of sequential processor built from a simple datapath and controller. The reference processor should illustrate these principles. The processor internal organization should be reasonably simple to be understood without deep knowledge in the digital design (note that "Logic Systems" course is unfortunately scheduled in parallel).

The next undergraduate course traditionally called "Computer Architecture" revisits the processor architecture and introduces the concept of pipelining and basics of techniques used in modern ILP processors as well as other concepts found in modern computers. Some topics are covered in separated courses "Peripheral Devices" and "Computer Networks". Undergraduate courses together cover all topics in the popular reference book "Computer Organization and Design – HW/SW Interface" (Patterson and Hennessy, 1998). Some topics from the area of peripheral subsystem and computer networks are covered in more detail than in this book. The main graduate course "Advanced Computer Architectures" is obligatory only for students of HW specialization. This course is based on the book (Patterson and Hennessy, 2002).

With this course flow in mind, we can describe the organization of the DOP, which is used to illustrate the basic principles of computer organization and design. The relation between the DOP and more advanced courses is discussed in the section 5.4.

3. DOP ARCHITECTURE OVERVIEW

The abbreviation "DOP" means "Danecek's Original Processor" according to one of its proponents. The DOP processor has not been primarily designed to be an educational platform. Its ISA was designed as a result of experience with writing HLL compilers for 8-bit and 16-bit microcontrollers like 8051, 68HC11, PIC16C5x, SAB80C166. These simple processors were designed for assembly language programming and writing efficient compilers for them is difficult and sometimes even impossible (Danecek *et al*., 1994b).
The DOP was intended to be a simple 16-bit processor core suitable for embedded systems and implementation in FPGA (Danecek *et al*, 1994a). The main feature of the processor is the simple compilation of high-level languages. From the nature of applications comes the requirement to optimize the program size over the speed of execution.

It is not surprising that the result of this development is an accumulator-oriented processor with variable length instruction encoding. Its main characteristics are outlined in Table 1. The processor contains only few programmer-visible registers. Local variables, temporaries and parameters are allocated on the stack in the main memory. This arrangement is valuable for illustrating relation between HLL programs and actual "binary" program executed by the processor.

The DOP processor is connected to the byte-organized main memory and peripheral subsystem by 16-bit Address Bus and 8-bit Data Bus. (Multibyte data are stored using Little Endian format) The main memory is common for the data and instructions.

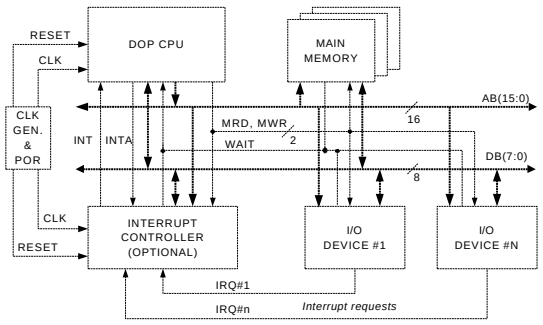

Fig.2. DOP system level overview

Peripheral devices for DOP could be memory mapped and processor supports single external maskable interrupt signal. The interrupt subsystem could be further expanded by an external interrupt controller. Interrupt subsystem was added to illustrate the interrupt service cycle at the HW level. Figure 2 shows the

interconnection between DOP processor, memory and peripherals. Some parts of this system exists as an FPGA implementation, others are only modeled in SW or exist only in the specification (peripheral devices). For further discussion of HW and SW tools please refer to section 4.

Table 1 DOP Characteristics

DOP ALU width	16 bit
Internal bus width	16 bit
Address bus width	16 bit
Data bus width	8 bit
Encoding of signed numbers	2's complement
Data types supported by ISA	Word (16 bit), unsigned byte (8 bit), signed short (8 bit)
Multibyte data storage format	Little Endian
I/O subsystem	Memory Mapped
Programmer visible 16-bit registers	PC, SP, W, S, D
Programmer visible 8-bit registers	L (loop counter), F (Flags)
External interrupts	1 (maskable), 16 interrupts with external interrupt controller

3.1 DOP Instruction Set Architecture

The DOP ISA is an example of an accumulator-oriented instruction set with several enhancements.
First operand of ALU instruction is always an accumulator – register W. Second operand can be register or memory location (typically on the stack where local variables and temporaries are located).
The instruction set also includes a dedicated instruction LLA, which computes the address of local variable on the stack. Figure 4 shows the example of computation with local variables.

DOP instruction set supports three data types – 16-bit word, 8-bit unsigned byte and 8-bit signed short integer. Bytes and short integers are internally extended to word length and all operations are performed with these 16-bit operands. This is another solution than in x86 ISA, which provides separated instructions for 8-bit operands. Moreover, all data manipulation instructions support all three data types. This regularity simplifies the task of code generation and is also a good educational example.

DOP ISA also includes several provisions for efficient support of operands longer than 16-bit word and addition and subtraction of operands of different sizes (see Fig.5). Firstly, lower words of the two operands are added or subtracted (if the shorter operand is an 8-bit short integer, it is sign-extended). In the same time, the sign of the second ALU operand is stored in the special Auxiliary Flag (AF) (see also AUXF signal on the fig. 6). The addition or subtraction is finished by applying instruction AAF to the remaining words of the longer operand (instruction adds extended sign XAF of shorter operand stored in the AF and carry flag CF from the previous operation.)

There are also two special instruction prefixes modifying the behavior of the following ALU instruction. First prefix is an UCF (use carry flag) this prefix enforces the use of the CF in the following ALU instruction. For example the prefix UCF followed by an ADD instruction is equivalent to the ADDC instruction (add with carry) whereas the UCF followed by the SUB behaves like the SUBB (subtract with borrow). Second prefix is the SWW (suppress write to W) that allows synthesizing the comparison and test instruction. The SWW followed by the SUB behaves like the CMP (only flags are set, W is not modified) and the SWW followed by an AND is similar to the TEST instruction.

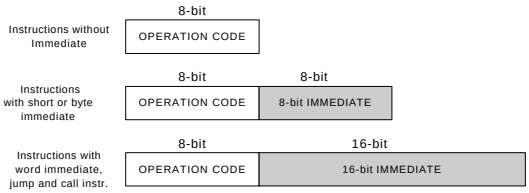

Fig.3. DOP instruction formats

The DOP is oriented to the high instruction encoding density and uses three formats of instruction. Most of instructions occupy only a single byte (1st format); other formats are used for instructions with 8-bit or 16-bit immediate. The DOP encoding density is superior over comparable processors. It has been reported that programs compiled for DOP occupy less than 60 % of memory space than for 8051. This feature can be very valuable for embedded systems (Danecek *et. al*, 1994c)

```
LLA S, 0x04      ;  S <= SP – 0x04 (address of a)
LLA W, 0x07      ;  W<=SP – 0x07 (address of b)
LLA D, 0x03      ;  W<=SP – 0x03 (address of c)
LD W, [W]        ;  W<=Mem[W]
ADD [S+]         ; W<= W + Mem[S], S++
ST [D]           ; Mem[D]<=W
```

Fig. 4 Example of c:=a+b in DOP assembly language (*a,b and c* are local variables allocated on stack)

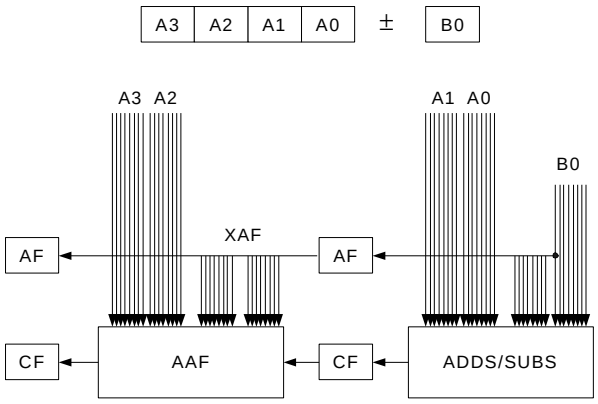

Fig.5 Example of addition/subtraction of short integer (8-bit signed) to doubleword (32-bit signed)

3.2 DOP Arithmetic and Logic Unit

The DOP Arithmetic and Logic Unit is based on the 16-bit W register which serves as an accumulator. This organization is very simple yet efficient for this class of processor. The second operand for ALU could be an internal register or an operand read from memory (to temporary register). The second operand is connected by the internal 16-bit bus to the second ALU input.

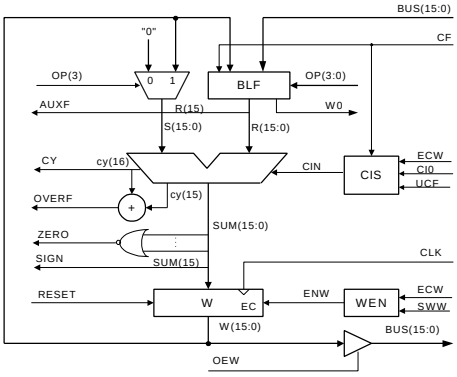

Fig.6. DOP ALU organization

The ALU internally contains the Block of Logic Functions (BLF) and 16-bit binary adder. This organization implements all basic binary arithmetic and logical operations (addition, subtraction, logical and, or, xor, negation and shifts). The logical operations are implemented in BLF, while the second input of adder is connected to zero. More complex operations such as multiplication or division could be synthesized by SW routine (library of these routines is available).
The blocks labeled CIS (carry in selection) and WEN (write enable) implement the prefixing instructions UCF and SWW.

The ALU also contains Flags – Carry Flag, 2's complement Overflow Flag, Zero Flag, Sign Flag and Auxiliary Flag used for addition or subtraction of operands longer than 16-bit word. All Flags are set after each ALU operation.

3.3 DOP Registers

Beside mandatory 16-bit PC (program counter) register the DOP contains only a few programmer-visible registers - SP (stack pointer), W (accumulator), which is a part of the ALU, S (source register) and D (destination register). Both S and D register could be used as general-purpose data storage during expression evaluation or address computation. All registers could be connected via 16-bit internal bus to ALU or to 16-bit external address bus and serve as an address for the main memory. The SP, S and D register support autoincrement and autodecrement addressing modes for accessing arrays and longer operands. Therefore these registers are implemented as bi-directional counters.

The DOP also includes the 8-bit L register, which is used as loop counter for an easy compilation of short *for* loops. The L register could be accessed separately or together with the FLAGS register as a 16-bit PSW (program status word).

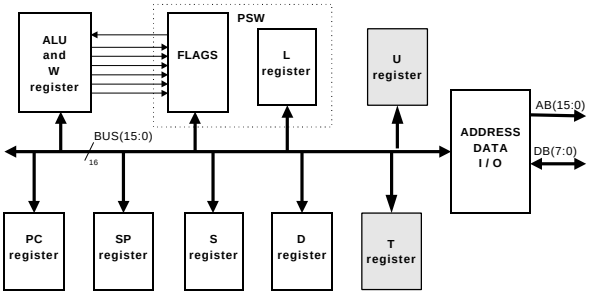

Fig. 7. DOP datapath

The whole datapath is outlined in figure 7. Note that U and T registers are not visible to the programmer and could be used as a temporary storage for the complex instructions. These registers were added only for educational purposes (original DOP ISA could be fully implemented without these registers).

3.4 DOP Controller

The DOP processor was originally implemented with optimized hardwired controller. However, this controller is not very suitable for practical experiments of our students. Consequently, a very simple horizontal microprogrammed controller was designed. The DOP controller currently needs the 58-bit wide microinstruction for all control signals (actual width is 64-bit including several spare control signals). Each

microinstruction corresponds to a single clock cycle. An Address of the next microinstruction is specified as a field in the current microinstruction. Three the lowest significant bits of this address could be modified by the condition multiplexer. This organization allows branching to the 8 destination addresses in a single microinstruction depending on the status of various internal and external signals. The DOP controller uses the condition multiplexer for gradual instruction decoding (decoding takes 2-3 clock cycles per instruction). The condition multiplexer is also employed for the testing of flags and external signal WAIT (from main memory or I/O) and INT (from interrupt controller).

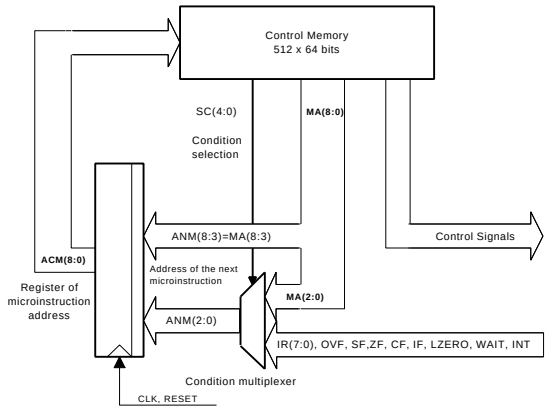

Fig.8. DOP microprogrammed controller

The complete implementation of the DOP instruction set needs less than 270 microinstructions in the control memory. This controller is definitely not the fastest possible microprogrammed one for this processor. However, its major advantage is the simplicity and regularity.

4. DOP HW AND SW TOOLS

4.1 DOP simulators and models

Four SW models of the DOP currently available are characterized in Table 2. Each of them allows the analyzing of the processor behavior on the different level of abstraction for various purposes. The first two simulators are executable on almost any personal computer, whereas the VHDL models require complex and relatively expensive VHDL simulator. Therefore only a limited number of students could use the VHDL models simultaneously. Recently, the major FPGA vendors offers relatively cheap versions of VHDL simulators but it is still unlikely that every student can buy one and run it at home on his computer.

Table 2 DOP SW simulators and models

Type of simulator	Modeling Language	Purpose
Instruction Cycle Accurate	C++	Assembly program debugging, compiler debugging
Functional Clock Cycle Accurate	Pascal	Microprogram development, observation of the int. function of the processor
VHDL RTL model	VHDL	Design verification, detailed view of signal flow
VHDL post P&R model	VHDL + SDF	Timing verification, detailed view of real delays on FPGA

The instruction-cycle accurate simulator can be used for debugging of DOP assembly program and the compiler development. The functional clock-cycle accurate simulator is used for development of DOP microprogram (firmware) for each instruction. The third type of model is synthesizable RTL VHDL model including memory and an interrupt controller. This model could be used to confirm results of functional clock-cycle accurate simulation. It allows more detailed view of the CPU behavior in time (namely signal sequences). The most accurate model is the post-place and route generated VHDL Vital model with SDF file. It shows the real delays of signals on FPGA.

4.2 HW emulator board

A FPGA based emulator board was designed for the DOP processor. The board contains the control memory 8 k x 64 bits made from 8 SRAM chips. The size of control memory is significantly higher than necessary for the DOP. It allows reusing the emulator board for different processors. The rest of the DOP is implemented in the Main FPGA (XC4013E-PQ160). The board also includes the additional 8k x 8bit SRAM circuit as the main memory for the program and data and some interface circuitry with the host system (implemented in separate Interface and Control FPGA). SW on the host computer controls all functionality of the emulator board.

Having configured the Main FPGA, the control memory could be downloaded with the microprogram. The SW on the host computer generates clock signal for CPU. The status of each register could be read out from the DOP processor after each clock cycle. Besides this debugging mode, the DOP processor is able to execute the sequence of instructions independently and later generates an interrupt to the host system.

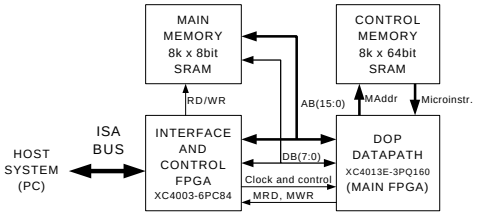

Fig. 9. DOP HW emulator board block diagram

Moreover, the HW emulator board offers the possibility of experiments that are not possible with the SW simulators. For example, it is possible to extend the basic DOP processor by additional functional units such as multiplier or divider and control them by currently unused signals in the control memory.

5. USE OF DOP IN COMPUTER AND LOGIC DESIGN COURSE

5.1 Overview of seminars

The "Computer and Logic Design" course is in the typical format of CTU. It takes one semester (14 weeks). Every week is a single 90-minutes lecture and 90-minutes seminar. Seminars are held in classrooms or in laboratories. The table 4 describes the current schedule of these course seminars. Most of the seminars are held in classroom and there are only two laboratory seminars. This is not optimal, but it is the result of limited availability of laboratories, which are used by parallel Logic Design course.
It can be also seen that DOP currently occupies approximately half of the semester.

Table 4 Example of Computer and Logic Design seminar schedule

Seminar	Scope
1	Introduction to DOP processor, Instruction Set Architecture and Data Types
2	Principles of synchronous design, Datapath of DOP - design of registers, ALU and interface circuitry, arithmetic
3	DOP controller implementation, principles of horizontal microprogram., discussion of possible enhancements
4	DOP basic cycle, DOP firmware, homework assignment
5 (laboratory)	Exercise with DOP cycle accurate simulator
6 (optional)	Evaluation of homework on VHDL simulator and HW emulator

5.2 DOP classroom seminars

During the first four seminars the internal organization of the DOP processor is explained to students. This is done with aim to maximally involve students in explaining the DOP schematic diagrams. These seminars have a strong link to lectures and goal is to illustrate the topics of lectures on practical examples.
For example: during the DOP ALU description, arithmetic is exercised and other possible organizations are discussed. Similar approach is used for explaining the DOP controller and basic cycle.

5.3 DOP laboratory seminar

After explaining the DOP schematics, the SIMDOP – functional clock cycle accurate simulator is introduced during laboratory seminar. Students are let to write a short program in DOP assembly language, translate instructions into hexadecimal form and execute them step-by-step on the simulator. Students are also shown that the same program is executed in the VHDL simulator and most importantly on the HW emulator board. The majority of students does not understand the VHDL simulator and the HW emulator board but they appreciate that the processor "really exists".

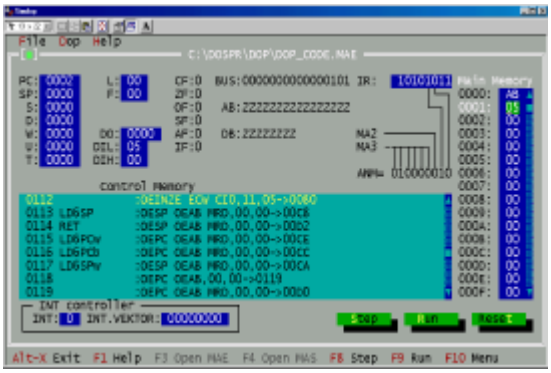

Fig. 10. SIMDOP – functional cycle accurate simulator

5.4 DOP homework assignment

The main educational method based on the DOP is the homework, which is solved by every student independently. Each student has to write a microprogram implementing some complex instruction, which can possibly extend the DOP instruction set. Currently, we have collection of around 50 complex instructions usable as homework. Most instructions are extending DOP arithmetic capabilities (e.g. multiplication and division, operations on long operands). Typical complexity of homework is between 10 and 20 microinstructions. One of the requirements is

preserving functionality of the original instructions. Two registers U and T were added to the DOP for making the implementation of these complex instructions easier.

The microprograms are developed using functional cycle accurate simulator (SIMDOP), which is available to all students (see figure 10). For easier writing of horizontal microinstructions symbolic language - *microassembler* and its compiler were developed. Microassebler represents each microinstruction as a set of active signals and uses labels to represent addresses in Control Memory (see Control Memory section on figure 10). Compiler is responsible for allocation of microinstruction in the Control Memory and translates the microprogram to the binary format. It also does some correctness checks on the microprogram – for example it checks the control of tri-state buffers to avoid the most frequent mistakes coming from the collision on the internal bus.

This compiler simplifies the student task but can also lead to misunderstanding of the real content of Control Memory (binary format of microinstructions is created as a side effect of compilation, this format can be used with VHDL models).

Besides the implementation of the instruction, student has to write a report, which can become a part of DOP's documentation for a programmer. Students have to describe the number of clock cycles and typical use of the added instruction.
Homework are reviewed at the end of semester by the teaching assistants and become part of students evaluation.

5.4 Optional seminar

At the end of semester one optional laboratory seminar is offered. There is usually no time to present this for every student. Volunteers are typically students more interested in HW. During this seminar, homework are evaluated on the VHDL models and downloaded to the HW emulator.

5.5 *Experience with the DOP*

Before the DOP processor was chosen a simple CPU based on AMD 2900 CPU slices had been used for the same purpose. It means that this approach of teaching this course has a relatively long tradition.
Student reports shows that t he main pedagogical tool is the homework. It has to be stressed that nowadays the main goal of the homework is not to teach microprogramming but make the students understand how the processor works. Microprogrammed controller offers the possibility to easy modify the processor functionality, which is advantage over the hardwired controller.

Student reports shows very frequently that they were afraid from the complexity of the homework but finally found it simple and interesting. They claim that homework helps them to finally understand how the processor works. It also seems that classroom seminars are not very efficient way to explain the DOP and students forgot most of it before they start to solve the homework assignment.

It is also interesting to note the most frequent mistakes students make in homework. Typically they use some dedicated register as PC or SP for intermediate storage in instruction and do not understand that it is not possible. Second most frequent mistake is made in reports where students are not able to write a reasonable description of instruction for a programmer. Description usually contains a lot of implementation details but important requirements for the programmer are omitted (state of input registers, modified registers, and flags).

5.6 Relation of the DOP to more advanced courses

It has been shown that the DOP reasonably describes the basic principles of computer organization, which are explained in the "Computer and Logic Design" course. It introduces some old concepts (accumulator-oriented ISA, microprogramming), which are not currently used in the mainstream general-purpose computers On the other hand similar processors are still used in the area of embedded systems. It seems more appropriate to show these principles on this type of architecture than building some artificial combination of RISC ISA and non-pipelined datapath with microprogrammed controller.

After introducing the quantitative approach and ILP in the following "Computer Architecture" course, students can see that the main idea behind the DOP (orientation to simple HLL compilation and high instruction encoding density) has lead to simple processor but with poor performance. More advanced concepts such as pipelining and superscalar execution are explained on the RISC processors. This is perhaps not a direct way to the contemporary computer architecture but it explains the evolution of processor architectures and ISA and relation between them.

6. CONCLUSIONS AND FUTURE WORK

The DOP core presents an example of a simple processor that could be used to illustrate the basics of computer organization and digital design. In comparison with commercial products, it has simple and orthogonal architecture. The DOP is currently used in introductory computer organization course seminars. Students participate on the design of the DOP datapath and controller and finally write a microprogram for DOP, implementing some complex instruction. Students use the cycle accurate simulator for homework assignment and could later evaluate the results on more accurate VHDL model or HW emulator board. The feedback from our students is mostly positive. They claim that homework assignment finally helped them to understand the processor functionality. More experienced students appreciated introduction to FPGA and VHDL style of circuit description.

In the future we want to increase the number of laboratory seminars which are more efficient than explaining the DOP processor in the classroom. We prepare new laboratory seminar for experimenting with C compiler and DOP instruction-cycle accurate simulator. The HW emulator board can be also used more efficiently. New experiments with this emulator would allow extending the DOP by additional units such as multiplier and controlling it by spare bits in the microinstruction. However, this requires significant rearrangement of the seminars and other courses.

At the same time the DOP is used also in more advanced digital design courses for experiments with FPGAs and as a simple target for developing compiler from subset of C in the introductory to compilers course. Currently, we plan to make the DOP documentation and tools available via Internet and JAVA version of SIMDOP is prepared.

REFERENCES

Brorson, M. (2002). MipsIT – a Simulator and Development Environment using Animation for Computer Architecture Education, In: *Proc. of Workshop of Computer Architecture Education*, Anchorage, USA

Bruschi, S. M. (1999). Simulation as a Tool for Computer Architecture Teaching, In: *Proc. of SCS Summer Simulation Conferenc*e, pp. 81-86., Orlando, USA

Danecek, J., Drápal, F., Pluhácek, A., Salcic, Z., Servít, M. (1994a): DOP – A Simple Processor for Custom Computing Machines. In: *Journal of Microcomputer Applications* , vol. 17, pp. 239-253, Academic Press Limited

Danecek, J., Drápal, F., Pluhácek, A., Servít, M. (1994b) The Architecture of General-Purpose Processor Cell. In: *Proc. of 4th International Workshop on Field-Programmable Logic and Applications, FPL94*, pp. 321-325, Prague

Danecek, J., Drápal, F., Pluhácek, A., Salcic Z.,Servít, M. (1994c) Methodologies for Computer Aided Hardware/Software Co-Design Using Field Programmable Gate Arrays. In: *Research Report.* Department of Computers, CTU Prague

Drápal, F., Danecek, J., Pluhácek, A., Servít, M. (1995), Implementation of a General-Purpose Processor Macro, In: *Proc. Design Methodologies for Microelectronics*, pp. 89 –97, Smolenice, Slovakia

Ellard, D., Holland, D., Murphy, N., Seltzer M. (2002) On the Design of a New CPU Architecture for Pedagogical Purposes In: *Proc. of the Workshop of Computer Architecture Education*, Anchorage, USA

Patterson, D., Hennessy, J. (1998), Computer Organization and Design: The Hardware/Software Interface, 2nd edition, Morgan Kaufmann Publishers , San Francisco, USA

Patterson, D., Hennessy, J. (2002), Computer Architecture A Quantitative Approach, , 3rd edition, Morgan Kaufmann Publishers, San Francisco, USA

Yurcik, W., Wolffe, G., Holliday, M. (2001). A Survey of Simulators Used in Computer Organization/Architecture Courses, In: *Proc. of the 2001 Summer Computer Simulation Conference (SCS 2001)*, Orlando, USA