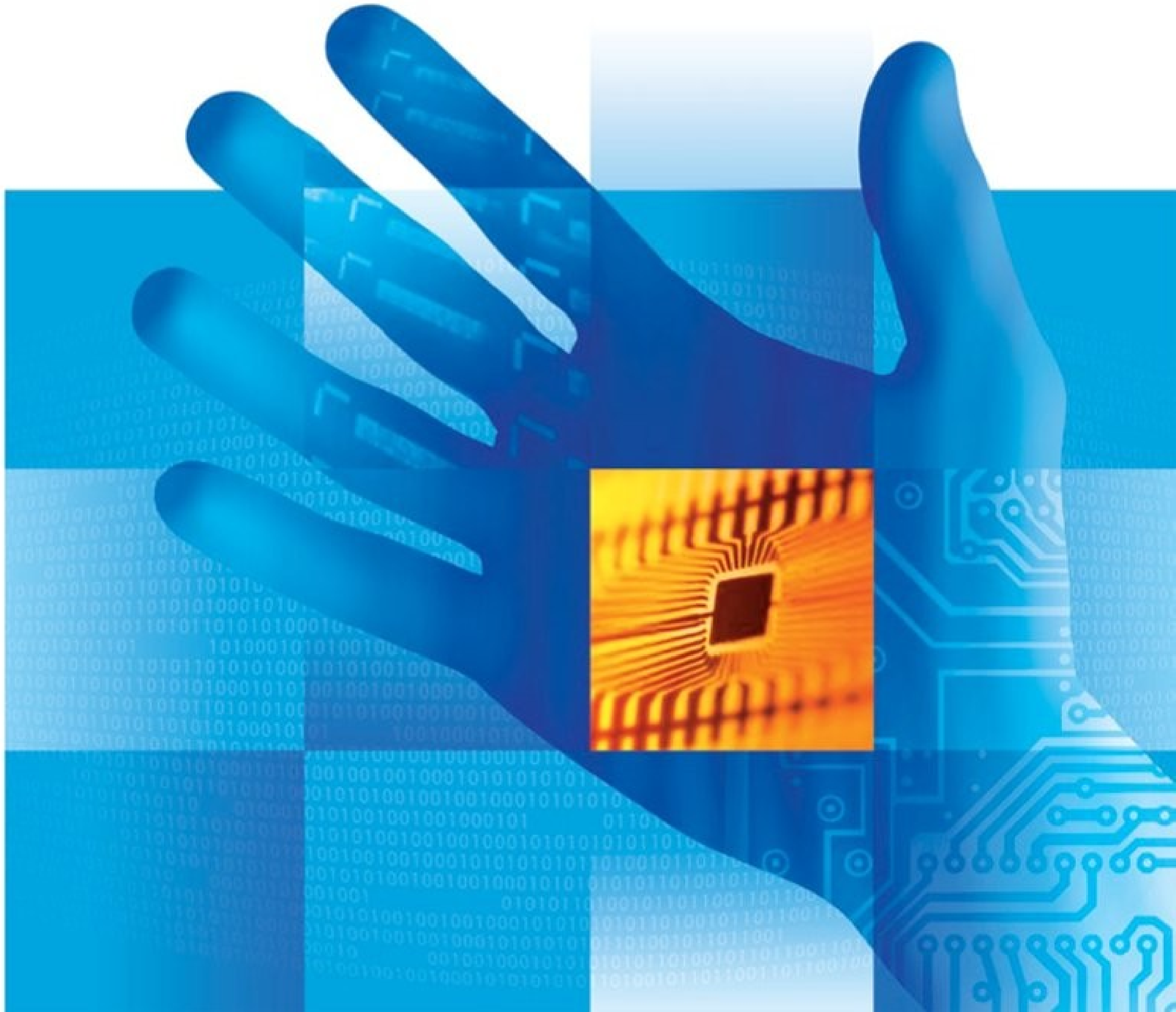
Architecture of Computers and Parallel Systems
Part 3: RISC Processors

Ing. Petr Olivka

petr.olivka@vsb.cz

Department of Computer Science

FEI VSB-TUO



Architecture of Computers and Parallel Systems

Part 3: RISC Processors

Ing. Petr Olivka

petr.olivka@vsb.cz

Department of Computer Science

FEI VSB-TUO

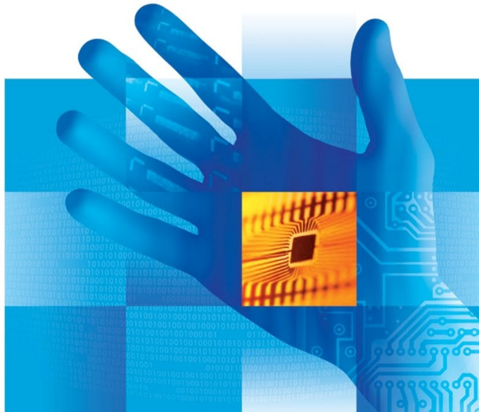
european social fund in the czech republic

EUROPEAN UNION

MINISTRY OF EDUCATION, YOUTH AND SPORTS

OP Education for Competitiveness

INVESTMENTS IN EDUCATION DEVELOPMENT



Architecture of Computers and Parallel Systems
Part 3: RISC Processors

Ing. Petr Olivka

petr.olivka@vsb.cz

Department of Computer Science

FEI VSB-TUO

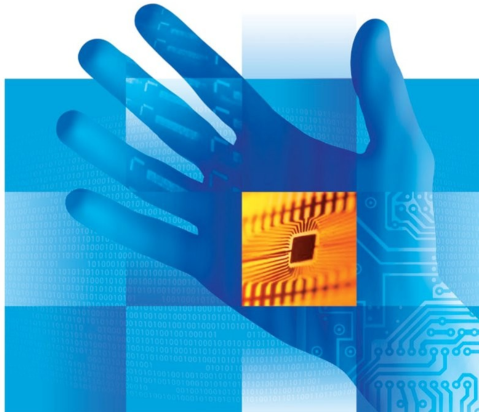
evropský sociální fond v ČR

EVROPSKÁ UNIE

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

OP Vzdělávání pro konkurenceschopnost

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ



CISC and RISC Processors

In the 1970's the processor was a very complicated circuit controlled by a microprogram controller that used to have hundreds of machine instructions. The processor development took long time, required complicated testing and large team of development experts. All new processors were very expensive.

Many experts pointed to uselessness of this construction and started to think about new technology.

Research teams from universities and computer manufacturers labs started to study available programs from various branches and used for different purposes. Researchers studied machine language of programs created directly in assembly code and programs compiled by programming language compilers.

To compare the results more objectively, they selected only programs for one specific computer – the IBM 360 System.

… CISC and RISC Processors

The result of research was a statistic of machine instructions usage. and it was introduced to the professional public.

The next table shows the static frequency of used instructions:

Instruction		Freq [%]
LOAD	Read from memory	26.6
STORE	Write to memory	15.6
Jcond	Conditional jump	10.0
LOADA	Load address	7.0
SUB	Subtract	5.8
CALL	Call subroutine	5.3
SLL	Bit shift left	3.6
IC	Insert character	3.2

… CISC and RISC Processors

When the program is running, some parts of it are used with a different frequency then other parts or a code is repeated in a loop.

The next table shows the dynamic frequency of used instructions:

Instruction		Freq [%]
LOAD	Read from memory	27.3
Jcond	Conditional jump	13.7
STORE	Write to memory	9.8
CMP	Compare	6.2
LOADA	Load address	6.1
SUB	Subtract	4.5
IC	Insert character	4.1
ADD	Addition	3.7

… CISC and RISC Processors

From the previous tables we can see that in 50% of cases there are only three instructions used! And eight instructions are used in 75%.

Other instructions are used with frequency less then one percent, many instruction are used in 1/1000 percent, and a part of instruction set is never used!

It is clear from this research that development of a processor with a large instruction set is not effective. At the end of 70's and early 80's new era of processors started. We started to divide processors to two groups:

- **CISC** – Complex Instruction Set Computer.
- **RISC** – Reduced Instruction Set Computer.

RISC Processors

The instruction set reduction is a well known feature of RISC processors. But developers had more goals in mind to improve the overall construction quality of the processor. Briefly they can be summarized into several points:

- Only basic instructions are implemented. Complex instructions are substituted by sequence of instructions.
- All instructions have the same length – reading instruction from the memory is faster.
- All instructions use the same format – instruction decoding is easier and decoding unit can be simple.
- Microprogramming controller is replaced by faster hardwired controller.
- Only two instructions can read/write data from/to memory – LOAD and STORE.

… RISC Processors

- Addressing modes are reduced to minimum number.
- More registers are implemented directly in processor.
- Pipelined execution of instruction is used.
- In every machine cycle one instruction is completed.
- Complex technical processor equipment is transferred to the programming language compiler. Programming in assembly language is not recommended.

All these features make up sophisticated and coherent circuit.

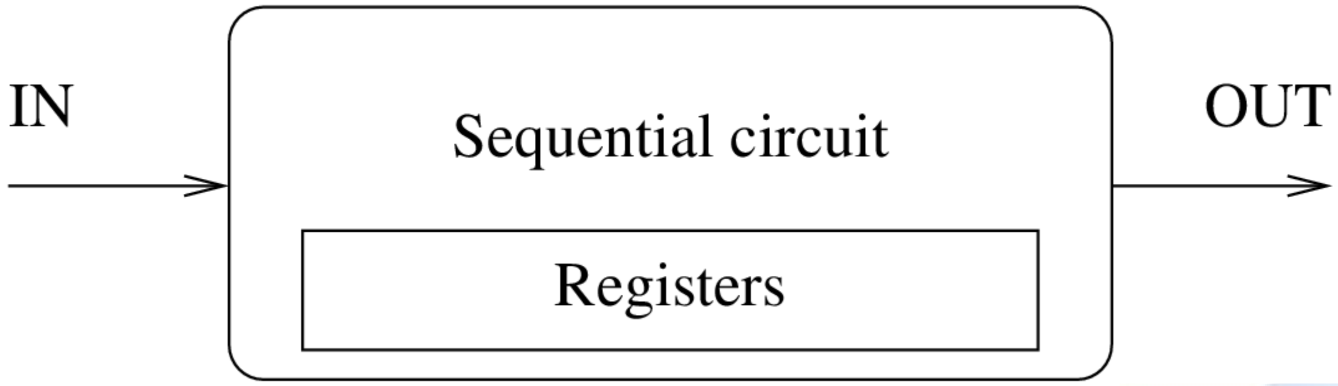
When all instructions have the same length and format, then Fetch and Decode Unit can have simple design and can work faster.

More registers have to be implemented, because all instructions are not able to access data directly in the memory and more temporary data is necessary to be stored in the processor.

Pipelining

The Pipelining is the second well known property of the RISC processors, and it is very often mentioned by manufacturers. Now we will explain this principle.

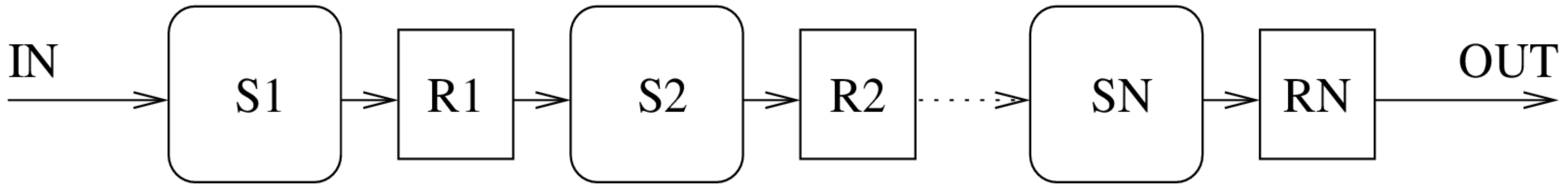
The processor is a sequential circuit. It takes input command for processing and until it is done, it does not accept any new command. The simple scheme of the sequential circuit is on the next figure:



The Input IN reads commands, the output OUT saves results and registers store the state of the circuit. The speed of execution is in the most cases given by external clock source.

... Pipelining

Because instructions are executed in more steps and these steps are processed in different part of circuit, experts tried to figure out, how to use all parts of processors permanently. They changed sequential circuit to chain of independent circuits. The result is visible on the next scheme:



Circuits S1 ÷ SN are separated stages of instruction execution. Blocks R1 ÷ RN are registers for temporary results passed between single steps. Input reads the instruction and output saves the result after processing.

To consider the circuit as pipelined, all stages have to work for the same time, otherwise the slowest one hinders the process.

… Pipelining

Now we can view instruction execution in five steps:

1. **FE** – Fetch instruction.
2. **DE** – Decoding instruction.
3. **LD** – Load data and operands.
4. **EX** – Instruction execution.
5. **ST** – Store result.

The instruction processing of CISC/RISC can be shown in the following diagrams – the dependencies of machine cycle T_N and individual stages of processing:

… Pipelining

Machine instruction processing in the CISC processor:

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
FE	I_1					I_2				
DE		I_1					I_2			
LD			I_1					I_2		
EX				I_1					I_2	
ST					I_1					I_2

On the diagram, you can see that the CISC processor is not able to start a new instruction execution, until previous instruction is finalized.

… Pipelining

Machine instruction processing in the RISC processor using the pipelining:

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
FE	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
DE		I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9
LD			I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
EX				I_1	I_2	I_3	I_4	I_5	I_6	I_7
ST					I_1	I_2	I_3	I_4	I_5	I_6

… Pipelining

From the previous two diagrams, it is visible that CISC processor finished instructions after 10 machine cycles 2, while pipelined RISC has done 6 instruction and the 10th instruction processing started.

The question is: how many times does the pipelining increase the performance?

Theoretically, in infinite time, the acceleration of execution is directly proportional to the length of the pipeline. In our case, the execution can be 5x faster. The common mistake is to think that longer pipeline can accelerate execution. But that is not quite true. The jump instruction changes the address of the next executed instruction and therefore instructions in progress are lost. The Fetch unit has to start loading instructions from new address.

From the beginning of this chapter we know, that the code contains up to 14% of conditional jumps. A very long pipeline can cause great ineffectiveness, given by unnecessary losses of instructions in progress.

Pipeline queue filling

The pipeline queue has to be filled by a continuous stream of instructions, as it was described earlier. Any delay will decrease the processor speed. The main problem is in the jumps instructions.

The jump instruction to fixed address can be detected early in the first pipeline stages and this information may be quickly passed on to the Fetch Unit to start loading instructions from the new address.

But the problem is with a conditional jump. The result, whether or not a jump will be done will be known just after the execution of the jump instruction. And this is the big weakness of pipelining. CPU manufacturers use different methods to decrease adverse effects of conditional jumps.

Yet, another problem with queue fulfillment arises when the program modifies itself. There are unprocessed instructions loaded in the processor. However, when it writes the instruction back to memory where it was already loaded, the old instruction is executed, not the new one. This problem is solved in modern processors in easy way: the program is not allowed to modify own code while running.

Pipeline Queue Filling
Delayed Jump

One simple method to manage conditional jumps, is a delayed jump. The processor starts loading instructions from the new address after all unfinished instructions are done. To explain how it works, imagine the next example:

The processor use three level pipelining:

1. **FD** – Fetch and decode instruction.
2. **EX** – Execute instruction.
3. **ST** – Store results.

And a simple code example in C language:

```
....
if ( i++ == LIMIT ) Res = 0;
Res += x;
....
```

... Delayed Jump

From the C language code the compiler will create the sequence of machine instructions:

```
....
I1:  CMP i, LIMIT   ; compare
I2:  INC I          ; increment
I3:  JMPne label    ; jump if not equal
I4:  MOV Res, 0     ; move value to variable
   label:
I5:  ADD Res, x     ; addition
....
```

Even though the code is in pseudo-assembly code, it is simple to understand. And now we can recreate the diagram of pipeline stages depending on the machine cycle:

… Delayed Jump

	T_1	T_2	T_3	T_4	T_5	T_6
FD	**CMP**	**INC**	**JMPne**	**MOV**	**NEW**	
EX		**CMP**	**INC**	**JMPne**	**MOV**	**NEW**
ST			**CMP**	**INC**	**-**	**MOV**

From the diagram it is visible that after the execution of the JMP instruction (it does not need to store result) the MOV instruction is in progress and NEW instruction is fetched depending on the JMP result. But MOV processing will continue regardless of the JMP result. And that is bad!

The compiler must reorder the instruction sequence and put one useful instruction after JMP (if not possible, use NOP – no operation).

… Delayed Jump

In our example the compiler can swap instruction I_2 and I_3 and execution with delayed jump will be correct now:

	T_1	T_2	T_3	T_4	T_5	T_6
FD	**CMP**	**JMPne**	**INC**	**NEW**		
EX		**CMP**	**JMPne**	**INC**	**NEW**	
ST			**CMP**	**-**	**INC**	**NEW**

The INC instruction will be executed always as needed. Instruction NEW is loaded correctly depending on JMP result. It can be seen that JMP result is evaluated one machine instruction later, than in normal sequence. Thus we say: delayed jump.

Pipeline Queue Filling
Bit Branch Prediction

The second well-know method used for better queue fulfillment is bit(s) for jump prediction. The prediction can be divided into 2 groups:

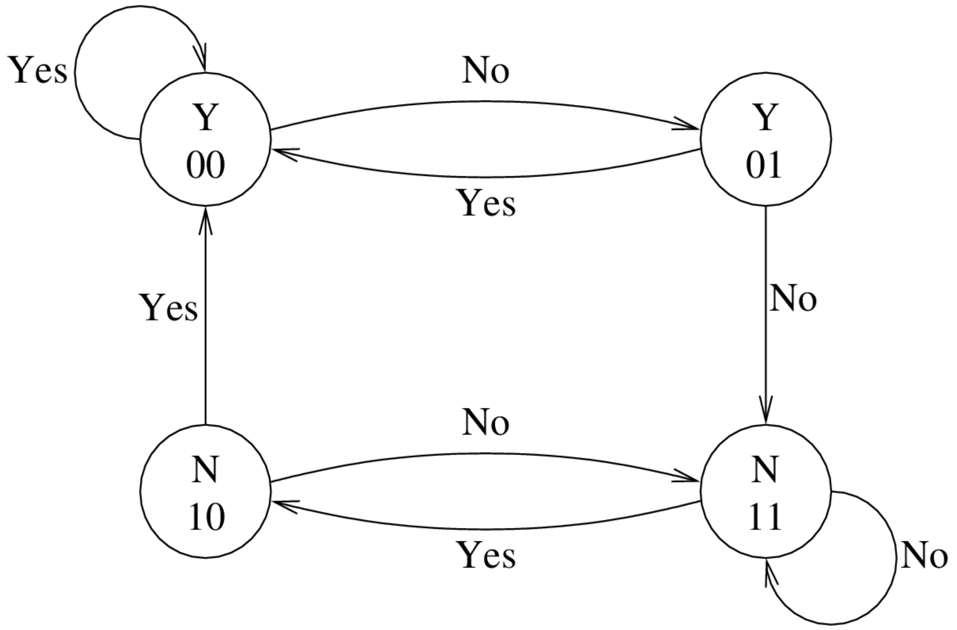
- Static prediction – bits are part of machine instruction and are set by a compiler or a programmer. They are set once and for all.
- Dynamic prediction – bits are in the processor and they are controlled dynamically during the code execution.

One bit static prediction is used in less powerful systems. It generates two failures in every loop – at the beginning and at the end. It does not matter for processors with short pipeline.

In modern high performance systems two bits dynamic prediction is used. The processor monitors the behavior of the conditional jump instruction and changes prediction only after two failures in the sequence. Bits are implemented directly in processor in Branch Prediction Table part to maximize the performance.

… Bit Branch Prediction

The two bits prediction can be depicted by the state diagram:



State N predicts that jump will continue in normal sequence and state Y predicts jump to target address. From the diagram it is clear, that only two jumps in sequence – the Yes edges - can change the prediction from N to Y. And vice-versa.

Pipeline Queue Filling
Super-Scalar Architecture

Manufacturers use super-scalar architecture in the highest performance computers. We can say in a simplified way that the processor has implemented two parallel pipelines.

Usually only one pipeline is working. When conditional jump is detected in early stages of instruction processing, first pipeline continues in processing in the normal way – following instruction sequence. The second pipeline obtains signal from the first one to start execution at conditional jump target address. But saving results is not allowed.

When the result of conditional jump is known, super-scalar control unit in processor decides, which pipeline will continue. The first one can continue in the normal way without losses. When the second one is selected, instructions from target address are in progress and processing can continue without delay.

The disadvantage of super-scalar architecture is the high price.

Hazards – Data and Structural

The pipeline filling is not the only problem of the RISC processors.

In many cases a problem may occur, when some pipeline stage needs data, which are not yet available. For example some instruction needs address of operand, but address is not yet stored by the previous instruction. If it happens, we call it a data hazard.

The problem can be solved directly in the pipeline, or by a compiler that prepares the correct instruction sequence.

Another type of the hazard occurs during handling the resources. When more pipeline stages need to load data from memory: e.g. the 1st fetches instruction, the 2nd loads data and the 3rd stores the result. All stage circuits need access to the bus. But computer contains only one bus and it is impossible to use the bus parallelly. Only one unit can use the bus at the same time.

These types of hazards are called structural.

Well-known RISC - ARM

One of the best known RISC processors is ARM – Acorn RISC Machine. This project started in 1983 and the first working chip was made in 1985. The first production chip ARM2 was available in 1986.

The ARM is 32-bit processor with 32-bit data bus and 26-bit address bus. It contains 27 registers and the core consists of 30 000 transistors. Performance was better than Intel 80286 CPU and Motorola 68000.

The processor is based on simplicity. It uses 3-stages pipeline (until 1996). The first stage loads instruction, the second decodes it and the third loads data and executes decoded instruction. The result is directly written to registers. This simple solution eliminates pipeline hazards.

In early 90's Acorn started cooperate with Apple and the result was ARM6 ant ARM7. These processors implement 32-bit address bus, cache on-chip and MMU.

… Well-known RISC - ARM

In the 4th ARM generation 5-stages pipeline was introduced in ARM9TDMI processor. Because the execution stage was the longest, it was split into 3 stages – arithmetic computation, memory access and write results back to registers.

The 5th generation of processor – ARM10 – introduced 64-bit data bus to allow fetching two instructions in one cycle. It reduces the problem with slow memory. Pipeline has 6-stages.

Latest ARM generation uses 11-stages pipeline.

ARM processors are very popular in many applications. Mobile phones, PDA, graphics accelerators, routers, play-consoles, notebooks, tablets, robots, etc.

In 2011 ARM have shipped more than 15 billion processors!

Well-known RISC - MIPS

The first MIPS (originally an acronym for Microprocessor without Interlocked Pipeline Stages) was introduced in 1985. Its development started in 1981.

The first processor known as R2000 was a full 32-bit version. It had 32 registers, MMU with flat 4GB addressing and virtual memory. The pipeline was 5-stages: fetch, decode, execute, memory-access, write-result. It was possible to work in little- and big-endian memory model.

Although design eliminated a number of useful instructions, such as multiply and divide, it was evident that the overall performance of the computer was dramatically improved, because the chip could run at much higher clock rates.

First 64-bit version was introduced as R4000 in 1991 and implements virtual memory and advanced TLB.

… Well-known RISC - MIPS

The Rx000 processor family was very important for SUN and SGI workstations.

One of the more interesting applications of the MIPS architecture is its usage in multiprocessor supercomputers. The Silicon Graphics (SGI) refocused its business from desktop graphics workstations to the high-performance computing market in the early 90's.

In the early 1990s MIPS started licensing their design to third-party vendors. The MIPS cores have been commercially successful, now being used in many consumer and industrial applications. MIPS cores can be found in newer Cisco, Mikrotik's routerboard routers, Wifi AP, cable modems and ADSL modems, smartcards, laser printer engines, set-top boxes, robots, handheld computers, Sony PlayStation 2 and Sony PlayStation Portable.

RISC Microcomputers – Microchip, Atmel

The RISC architecture is not only used for phones, desktops, workstations and supercomputers. The RISC is used today even for the smallest 8-bit microcomputers.

One of the best known, massively used microcomputers are PICs produced by Microchip. It implements one working register, 2-stages pipeline, after the jump instruction there is one machine cycle delay. It is based on Harvard architecture and implements only about 40 instructions. PICs are designed in wide range, from battery supply version, up to the fast version with Ethernet and USB.

Another well known RISC microcomputer is AVR from Atmel. The processor is designed as Harvard architecture. It implements 2-stages pipeline, 32 registers and instruction set is designed directly for C-language compiler. It is more user friendly than PIC.

The AVR is fast. The performance in MIPS is equal to clock frequency. PIC uses only quarter of clock signal.