EasyChair Preprint

№ 591

Simulation of the Super-scalar Processor Core Operation

Andrey Vishnekov and Elena Ivanova

October 26, 2018

Simulation of the Super-scalar Processor Core Operation

A V Vishnekov[1] **and E M Ivanova**[1]

[1]National Research University Higher School of Economics, 34, Tallinskaya St., Moscow, 123458, Russia

emivanova@hse.ru

Abstract. The article considers the features of super-scalar processors, their way of performing several operations on several pairs of operands simultaneously. The research focuses on the organization of processor pipeline execution operation of several machine instructions in one processor core. The simulating kit was developed for better understanding of a processor core microarchitecture. It includes two parts: program and methodical recommendations with multiple task options. The simulating kit demonstrates the pipeline architecture consisting of two clusters: front-end and back-end and the principle of translating complex multi-cycle CISC-like instructions into simpler RISC-like micro-operations. The main types of machine instructions are considered: data transfer between registers and memory cells (four variations), data processing of couple of operands from registers and memory cells (four variations), conditional jump to the specified address. The program-simulator makes it possible to conduct a more detailed simulation of one of the three mechanisms for calculations accelerating in the processor core: multi-functional (super-scalar) processing, out-of-order processing, speculative instructions execution after the branch prediction. The simulating kit is used in educational process when training masters of Higher School of Economics National Research University.

1. Introduction

Modern super-scalar processors [1-4] can perform several operations on several pairs of operands simultaneously. This is achieved by a variety of different technological and microarchitectural solutions which parallelize calculations. The main method is the pipeline processing of machine instructions [5-8]. Each processor may contain one or several cores. Each core contains one or more instruction pipelines. In fact, the pipeline is the main part of the processor. Understanding it's execution mechanism gives an understanding of computing principles in modern computers. A huge number of scientific and practical works are devoted to this issue, but the faster and more visual way to study the principle of pipeline processing is the launch and study of the processor core simulation model.

The processor is very complex device, so the model cannot display every aspect of its functioning. We focus only on some of them. We have built and investigated such a model that allows us to trace the process of machine instructions executing by the processor core pipeline. Intel processor with microarchitecture Nehalem [2] was chosen as a basis for the modeling, although the analyzed mechanisms are inherent to other modern processors. The most interesting mechanisms, which are the most complicated for understanding, are the following: multi-functional (super-scalar) processing [9], out-of-order (OOO) processing [1, 10, 11], branch prediction [12, 13], speculative instructions execution [14], pipeline reset after misprediction [2], the accelerated instructions fetch from L1i cache [15, 16], origins of pipeline hazards and bubbles [5, 10, 11], principles of pipeline division into two clusters:

front-end and back-end [3, 4, 17, 18], exception of data re-reading from L1d cache [1, 11], etc. Nehalem cores include all the mentioned technical solutions. Many scientific works and articles describe this microarchitecture. So students will easy find any additional information on all theoretical issues.

2. The front-end and back-end clusters of pipeline

Two parts of the pipeline, front-end and back-end, allow to integrate advantages of two processors architecture: convenience of the CISC program model and RISC high speed execution [18].

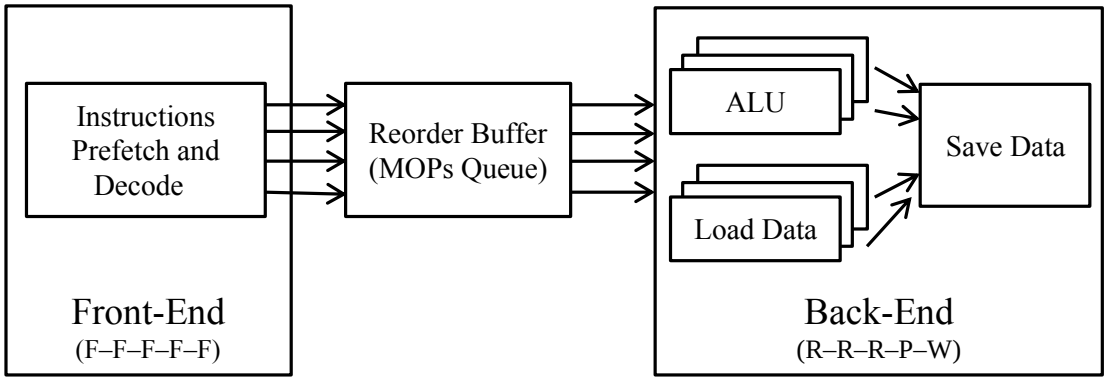

Figure 1. The simulated pipeline

The convenience is provided by the difficult instructions capable to process data from registers and memory. This is implemented via the Front-end CPU cluster. High speed is provided by simple and short operations. CISC-instructions are translated in simple RISC-like microoperations (MOPs) [15] which are processed in the Back-end CPU cluster by a set of executive devices (Figure 1): ALU (Processing), Load (Read data), Save (Write data). There is an instruction phase/ MOPs designation: F–F–F–F–F (for five Front-end phases)–R–R–R–P–W (for Back-end phases) under the scheme.

3. The simulating kit description

Simulating kit includes the program and the methodical recommendations [19] – a manual with multiple task options. Simulating kit demonstrates the pipeline architecture, which consists of two clusters: Front-end and Back-end, and the principle of translating complex multi-cycle CISC-like instructions into simpler RISC-like micro operations (MOPs). In addition, it is possible to conduct a more detailed simulation of one of the three accelerating mechanisms in the processor core:

- L.R. №1 – multi-functional processing,
- L.R. №2 – out-of-order processing,
- L.R. №3 – speculative instructions execution after the branch prediction.

The simulating program includes four windows of the model, which reflect the input parameters and simulation results. The computer simulator considers a code fragment written in a simplified Assembler as an initial data. The simulation program automatically generates a new version of the code fragment at startup or at the user's request. The "Task" window looks as shown in Figure 2. To study the pipeline executing principles, the following parameters of the pipeline and of the program code are additionally supposed to specify: the number of store/load devices, the number of ALU (Executive devices), and the percentage of operations with memory cells (Figure 2).

Program provides the following options as a result of the simulation:

- to explore the principle of translation of instructions in micro-operation ("MOPs" window), shown in Figure 3,

- to study the employment cycles of main back-end pipeline cluster when executing the given instructions ("Pipeline" window), shown in Figure 4,
- to examine the time diagram of instructions execution sequence from a given code fragment ("Diagram" window), shown in Figure 5.

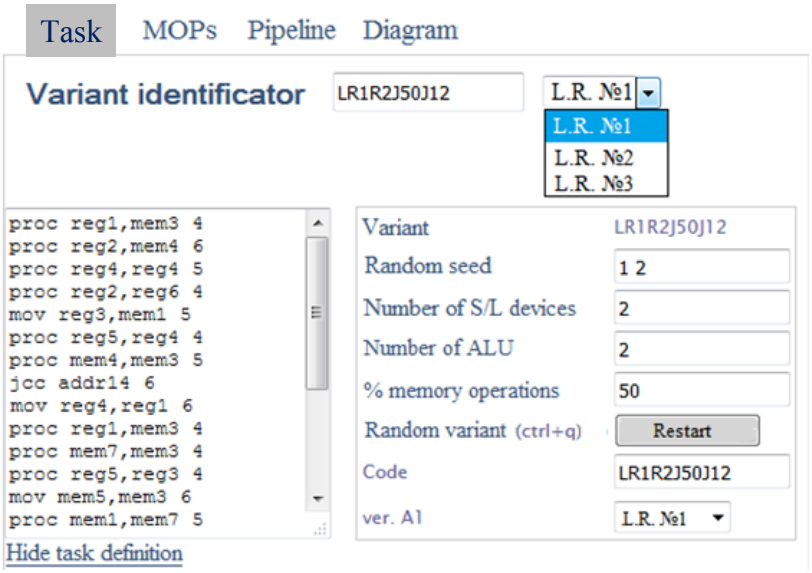

Figure 2. Simulator interface, "Task" window

The code fragment consists of about twenty instructions. Three main types of machine instructions [20] are considered: data transfer between registers and memory cells (four variations), data processing from registers and memory cells (four variations) and conditional jump to the specified address. These instructions differ from each other in a set of the MOPs which are carried out after decoding (Figure 3).

The 16-byte block of program code (which includes several instructions with a total length up to 16 bytes) is read from the L1i cache to Prefetch Buffer during each CPU clock cycle. Each window displays the executable code fragment. "MOPs" window shows relative instruction number (first column), instruction mnemonics (second column), instruction length (third column), prefetching register names (fourth column), and the columns 5-9 show us the executable MOPs in CPU Back-end cluster devices (Address Generation – GA, Searching data in Cache L1d – SC, Fetching data from cache into Buffer – FB, data Processing – P, Saving data in Register – SR or Saving data in Memory – SM). Sequentially fetching blocks of program code are depicted by alternating light and dark lines. Here one can explore the exception of data re-reading from L1d cache. For example, instruction 6 must decode into eight MOPs (Table 1). But it's both operands (mem3, mem4) were already read by previous instruction0 (mem3) and instruction1 (mem4). Six MOPs are cancelled. CPU performs fewer operations. Program code executes faster.

Table 1. Theoretically possible and actually executed MOPs for instruction 6 (*proc mem4,mem3*).

Theoretically possible MOPs (*with re-reading*)	Removed MOPs	Actually executed MOPs (*without re-reading*)
1) mem4 address generation (6GA), 2) searching mem4 in cache (6SC), 3) load mem4 from cache into buffer (6FB), 4) mem3 address generation (6GA), 5) searching mem3 in cache (6SC), 6) load mem3 from cache into buffer (6FB), 7) data processing (6P), 8) saving result in memory (6SM)	1) 6GA, 2) 6SC, 3) 6FB, 4) 6GA, 5) 6SC, 6) 6FB	7) data processing (6P), 8) saving result in memory (6SM)

Task MOPs Pipeline Diagram LR1R2J50J12

№	Instruction	Len	Prefetch					
0	proc reg1,mem3	4	reg1	Address Generation	Search in Cache	Fetch into Buffer	Processing	Saving in Register
1	proc reg2,mem4	6	reg2	Address Generation	Search in Cache	Fetch into Buffer	Processing	Saving in Register
2	proc reg4,reg4	5	reg4	Processing	Saving in Register			
3	proc reg2,reg6	4	reg6	Processing	Saving in Register			
4	proc reg3,mem3	4	reg3	Processing	Saving in Register			
5	proc reg5,reg4	4	reg5	Processing	Saving in Register			
6	proc mem4,mem3	5		Processing	Saving in Memory			
7	jcc addr14	6		Jump				
8	proc mem7,reg3	5		Address Generation	Search in Cache	Fetch into Buffer	Processing	Saving in Memory
9	proc reg1,mem3	4		Processing	Saving in Register			
10	proc mem7,mem3	4		Processing	Saving in Memory			
11	proc reg5,reg3	4		Processing	Saving in Register			
12	mov mem5,mem3	6		Saving in Memory				
13	proc mem1,mem7	5		Address Generation	Search in Cache	Fetch into Buffer	Processing	Saving in Memory
14	proc mem3,mem2	4		Address Generation	Search in Cache	Fetch into Buffer	Processing	Saving in Memory
15	proc mem3,reg7	6	reg7	Processing	Saving in Memory			
16	proc reg3,mem3	5		Processing	Saving in Register			
17	mov reg7,reg7	7		Saving in Register				
18	jcc addr21	2		Jump				
19	proc mem7,mem1	5		Processing	Saving in Memory			
20	proc reg1,reg6	6		Processing	Saving in Register			

Figure 3. Simulator interface, "MOPs" window

Figure 4 displays the "Pipeline" window: relative number of CPU clock cycles (first column), block of instructions (second column) and the columns 3-9 show us the the employment cycles of named CPU Back-end cluster devices (Read0…2, ALU0...1, SM, SR). Empty cells mean the halt of specified device in specified cycle. Each MOP has a number of the appropriate instruction.

Task MOPs Pipeline Diagram LR1R2J50J12

	Front-end	Read0	Read1	Read2	ALU0	ALU1	SM	SR
0	Instr 0, Instr 1							
1	Instr 2, Instr 3, Instr 4	0GA	1GA					
2	Instr 5, Instr 6, Instr 7	0SC	1SC	4GA	3P			2SR
3	Instr 8, Instr 9	0FB	1FB	4SC	7I			3SR
4	Instr 10, Instr 11	4FB	6GA		0P	9P		1SR
5	Instr 12, Instr 13	6SC			4P		10SM	0SR, 9SR
6	Instr 14, Instr 15, Instr 16	6FB			5P	11P	12SM	4SR, 8SR
7	Instr 17, Instr 18	16GA			13P		6SM	5SR, 11SR
8	Instr 19, Instr 20	16SC			15P		13SM	
9		16FB			19P	20I	14SM	15SR
10					16P	17P	19SM	
11								16SR, 17SR
12		18GA						

Figure 4. Simulator interface, "Pipeline" window

One can study the parallel cycle-by-cycle MOPs execution. Out-of-order MOPs processing can be tracked according to the number of the executed MOPs on each clock cycle. For example, five MOPs are executed simultaneously on a cycle number 4: loading data from cache L1d in buffer for instruction 4 (4FB), address generation for instruction 6 (6GA), data processing for instruction 0 (0P), data

processing for instruction 9 (9P), saving data in register for instruction 1 (1SR). MOP 9P is processed out of order.

"Diagram" window allows one to examine the sequence and duration of instructions execution from given code fragment. Figure 5 (big picture) illustrates the diagram of the twenty instructions execution sequence: a relative instruction number (first column), instruction mnemonic (second column) and the CPU clock cycles from 0 to 16 (3-19 columns) show us the MOPs of instruction execution phase (F-fetch, R-read from memory, P-processing, W-result writing/saving). In the same Figure (small picture) one can see instructions execution sequence for a case of pipeline reset after misprediction for both front-end (F-F-F-F-F) and back-end (R-R-R-P-W) cluster. The situation of pipeline reset is represented by crossing out of the cancelled phases/MOPs (red colour letters).

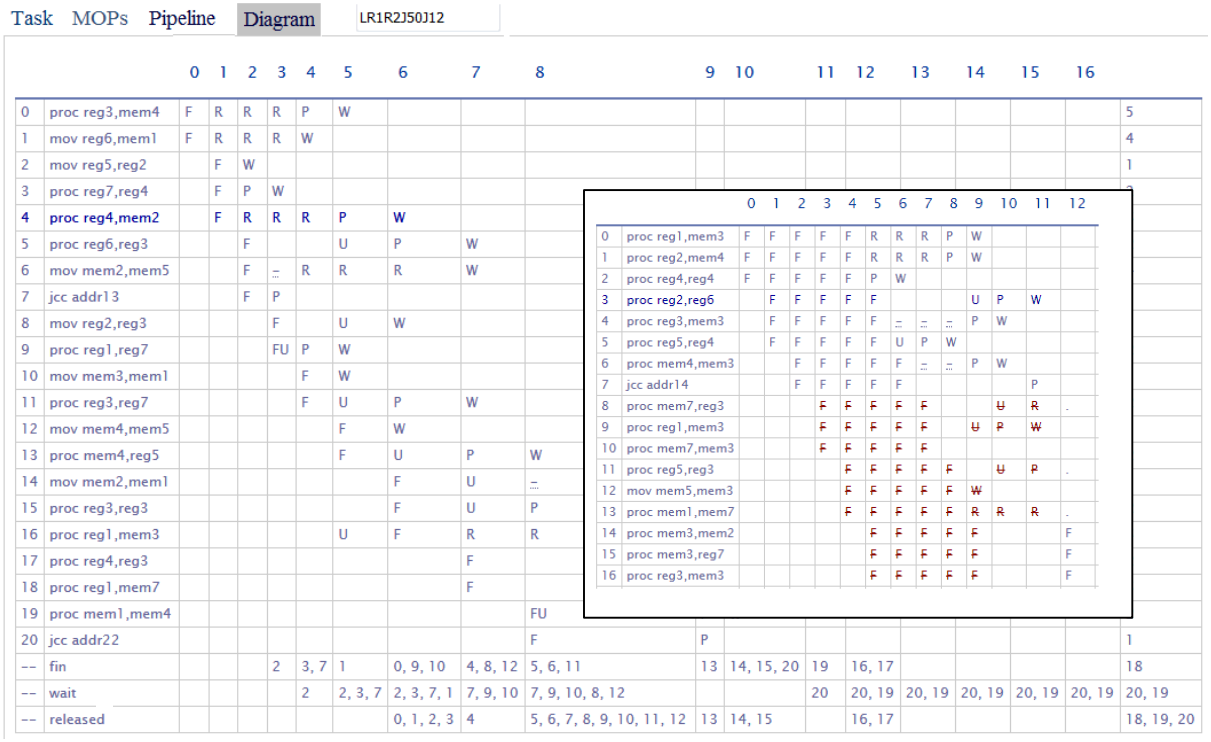

Figure 5. Simulator interface, two variants "Diagram" window: multi-functional processing (big picture) and pipeline reset after misprediction (small picture)

The sequence in-order recovering for instructions results saving after OOO execution is shown in two lines in the bottom of the diagram. Recovering is executed in two steps: fin (wait in the queue) and release. Diagram makes it easy to realize how common data and devices are used by instructions. One can easily trace the instruction halt.

4. Conclusion

The offered simulation of the super-scalar processor core allows to study the CPU core execution in dynamics cycle-by-cycle. This Simulating kit is useful for studying computer architecture. Understanding the features of the processor is useful for both system developers and programmers. Improving the style of writing programs will let the computer speed up their execution. The Simulating kit is used when teaching the discipline "Computing systems" in Higher School of Economics [20].

5. Acknowledgments

Authors thank the HSE student Lazutov A.N. for help in creation of the kit program part.

6. References

[1] Intel® 64 and IA-32 Architectures Optimization Reference Manual https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

[2] Korogod Sergey New in the Silvermont architecture of Intel Atom Z3xxx Processors. 2013. http://www.ixbt.com/portopc/atom-clover5.shtml

[3] Swinburne Richard. Feature - Intel Core i7 - Nehalem Architecture Dive. bit-tech. 2008. http://www.bit-tech.net/reviews/tech/cpus/intel-core-i7-nehalem-architecture-dive/5/

[4] Ozerov Sergei. CPU architecture. Computerra. 2005, 37(609). http://www.kinnet.ru/cterra/609/233266.html

[5] Kanter David. Intel's Sandy Bridge Microarchitecture. Website Real World Tech. 2010. https://www.realworldtech.com/sandy-bridge/10/

[6] Patterson David A., Hennessy John L. Computer Architecture: A Quantitative Approach, 5th Edition. — Morgan Kaufmann, 2011.

[7] Tanenbaum A.S., Austin T. Structured Computer Organization. Prentice Hall, 2012.

[8] Tullsen D.M., Eggers S.J., Levy H.M. "Simultaneous multithreading: Maximizing on-chip parallelism". 22nd Annual International Symposium on Computer Architecture. IEEE. 1995: 392–403.

[9] Abeydeera Maleen, Subramanian Suvinay, Jeffrey Mark C., Emer Joel, Sanchez Daniel, "SAM: Optimizing Multithreaded Cores for Speculative Parallelism", Parallel Architectures and Compilation Techniques (PACT) 2017 26th International Conference. 2017: 64-78.

[10] Intel Sandy Bridge microarchitecture overview. Out-of-Order Cluster. https://hw-lab.com/intel-sandy-bridge-cpu-microarchitecture.html/4

[11] Rappoport Lihu. Intel Core Microarchitecture. https://moodle.technion.ac.il/enrol/index.php?id=5916

[12] Coursera. Branch Prediction Introduction https://www.coursera.org/lecture/comparch/branch-prediction-introduction-BF58C

[13] Branch Prediction Review https://courses.cs.washington.edu/courses/csep548/06au/lectures/branchPred.pdf

[14] Intel Analysis of Speculative Execution Side Channels. White Paper. 2018. https://www.intel.com/content/www/us/en/architecture-and-technology/intel-analysis-of-speculative-execution-side-channels-paper.html?wapkw=speculative+execution

[15] Polyvyalov Alexander. In anticipation of the Willamette — the history of the IA-32 architecture and the way the P6 family processors work. 2000. https://www.ixbt.com/cpu/pentium4-1.html

[16] Garmatyuk Stanislav. The "new old" Core i7 architecture: what's more-similarities or differences? Website ixbt.com/PC Platform. 2008. https://www.ixbt.com/cpu/intel-ci7-theory.shtml

[17] Nechai Oleg. Selection of author's articles http://www.libma.ru/kompyutery_i_internet/cifrovoi_zhurnal_kompyuterra_74/p2.php

[18] Hinton Glenn, Sager Dave, Upton Mike, etc. The Microarchitecture of the Pentium4 Processor. Intel Technology Journal Q1, 2001 http://www.ecs.umass.edu/ece/koren/ece568/papers/Pentium4.pdf

[19] Ivanova E. M. Core superscalar CPU simulation. Methodical recommendations on "Computer systems" for the direction 09.04.01. "Computer science and engineering" master's degree. M.: HSE, 2018. https://publications.hse.ru/en/books/214154058

[20] Intel 64 and IA-32 Architectures Software Developer's Manual. http://www.intel.ie/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf