μSystems Research Group
School of Electrical and Electronic Engineering
Merz Court
Newcastle University
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

Modelling concurrency in processor instruction sets

Alessandro de Gennaro

a.de-gennaro@ncl.ac.uk

NCL-EEE-MICRO-MEMO-2015-010

August 2015

Abstract

Complexity of processors available in the market continuously increases, forcing designers to have more systematic approaches to plan new processor architectures. The task of finding new models to represent event-based systems (likewise processor's instructions) becomes increasingly important, pushing research to be focused on this topic. Models should be able to elegantly capture either sequential and concurrent behaviours; the latter ones are particularly important as they are the key to a high throughput. Another increasingly required feature of processor's controller is the scalability. Instruction Set Architectures (ISAs) are often updated for supporting more instructions, this requires the possibility of adding any instructions into a system to be a non-invasive process.

The main topic of this work is the development of automated software tools for the formal specification, verification and synthesis of processor instruction sets, with particular focus on overcoming the challenges associated with concurrency. Due to the page limit, we only give a brief overview of the problem domain and highlight recent results in automated encoding of instructions presented at the International Conference on Application of Concurrency to System Design, ACSD'2015 [1].

Modelling processor instructions

A number of conventional models can be employed for the specification of processor instructions. Finite State Machines (FSMs) is a well-known and tool supported formalism that has already been used in this application domain [2], however, FSM specifications do not scale well in the presence of concurrency. Petri Net based models, such as Signal Transition Graphs [3], is another popular choice, however, when one needs to reason not only about individual instructions, but about instruction sets, Petri Nets tend to quickly grow in complexity and become difficult to comprehend, as has been shown in [4]. We therefore employ Conditional Partial Order Graphs [4] as our specification language.

This is a graph-based representation composed by nodes and arcs. The former ones stand for the actions that might happen inside a generic system, whereas the arcs model the various dependencies between them. CPOG turns out to represent well the instructions that a generic processor can execute.

Consider the "store" instruction as an example. The intended behaviour of the instruction can be defined by the following microprogram:

Store instruction specification.

```
storeInstruction opcode = do
  pc <− incrementPC
  address <− fetch pc
  data <− readRegister(decodeRegisterID opcode)
  store address data
  pcNext <− incrementPC
  opcodeNext <− fetch pcNext
  return opcodeNext
```

Starting from this specification, the flow that the instruction should follow can be extracted and depicted through a partial order, see Fig. 1(b). This means representing each single action that the instruction needs to execute as a node, and each dependency via an arc between nodes. The graph can be automatically derived from the specification, avoiding mistakes and possible inconsistencies. The automated process also allows to recognise concurrent actions, that are actions which can be potentially executed at the same time: $\{readRegister, store\}$ and $\{incrementPC, fetch\}$.

The partial order shown in Fig. 1(a) represents the arithmetic instructions with register addressing mode, while the one in Fig. 1(c) models the unconditional branch with immediate addressing mode. The detailed description of these graphs is omitted for sake of brevity.

From instructions to instruction sets

Once partial orders of all instructions have been extracted from the corresponding formal specifications, we derive a Conditional Partial Order Graph model that captures the full instruction set in a compact form. This step is also automated, in particular, it is possible to generate optimal instruction opcodes for the given instructions using a recently presented algorithm [1]. The CPOG model containing the three instructions defined above is shown in Fig. 2. Boolean conditions assigned to vertices and arcs of the CPOG are used to activate/deactivate

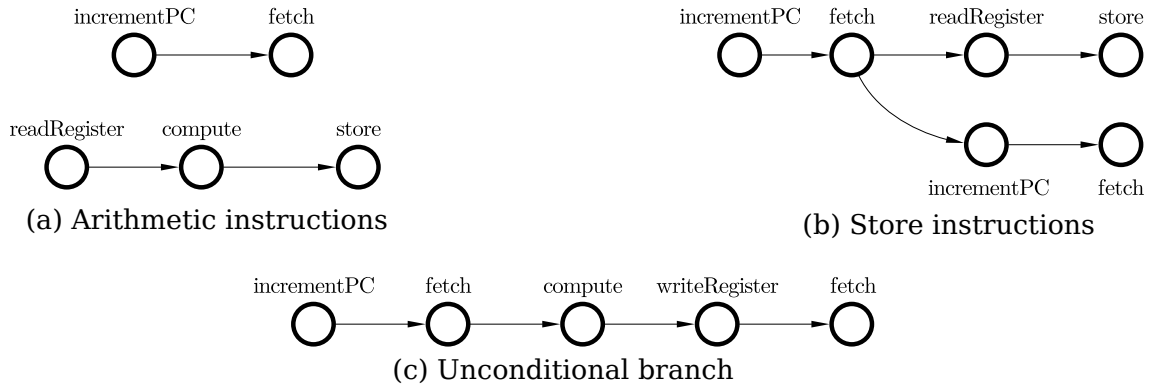

Figure 1: Three instruction classes of a generic processor.

actions and/or dependencies between them according to the current opcode, specified by variables $\{x_0, x_1\}$. The instructions are given the following opcodes: $(a) = 11$, $(b) = 00$, $(c) = 10$.

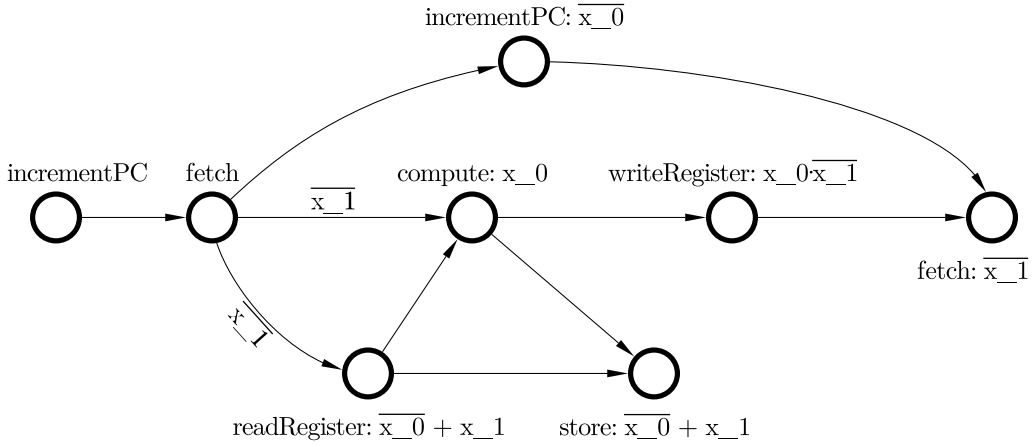

Figure 2: Conditional Partial Order Graph derived by the three partial orders in Figure 1.

Hence, the first step to derive the CPOG is to find an injective function between the elements contained inside the set of the partial orders and the set of the opcodes. Finding a good assignment is a demanding task, as the number of possible permutations (possibly dispositions) exponentially explodes. This is the key point for deriving an optimised and compact CPOG.

A lot of research might be carried on around this topic. Different approaches are present in the literature to optimally group a bunch of partial order graphs into a CPOG, all of them have been implemented as an external plugin in Workcraft (`http://www.workcraft.org/`). In [1] we present a heuristic cost function which targets area minimisation, however, other functions might be employed and tailored to other purposes: a low power system for low profile applications, or a reduced latency decoder for high performance tasks.

Acknowledgements: The author would like to thank Dr. Andrey Mokhov for helping with this work, as well as Newcastle University for the opportunity to attend LASER summer school.

References

[1] A. de Gennaro, P. Stankaitis, A. Mokhov. *"A Heuristic Algorithm for Deriving Compact Models of Processor Instruction Sets"*. 15th International Conference on Application of Concurrency to System Design, June 24-26, 2015.

[2] G. De Micheli *"Synthesis and Optimization of Digital Circuits"*. 1994, ISBN: 0070163332. McGraw-Hill Higher Education.

[3] J. Cortadella, M.Kishinevsky, A.Kondratyev, L. Lavagno and A.Yakovlev. *"Logic synthesis of asynchronous controllers and interfaces"*. Advanced Microelectronics, 2002, Springer-Verlag.

[4] A. Mokhov, A. Yakovlev. *"Conditional partial order graphs: Model, synthesis, and application"*. IEEE Transactions on Computers, Volume 59, Pages 1480-1493, November 2010.