# LISA – Machine Description Language and Generic Machine Model for HW/SW Co-Design

**Vojin Živojnović Stefan Pees Heinrich Meyr**

Integrated Systems for Signal Processing
Aachen University of Technology
Aachen, Germany

**Abstract - In the paper a new machine description language is presented. The new language LISA, and its generic machine model are able to produce bit- and cycle/phase-accurate processor models covering the specific needs of HW/SW co-design, and co-simulation environments. The development of a new language was necessary in order to cover the gap between coarse ISA models used in compilers, and instruction-set simulators on the one hand, and detailed models used for hardware design on the other. The main part of the paper is devoted to behavioral pipeline modeling. The pipeline controller of the generic machine model is represented as an ASAP (As Soon As Possible) sequencer parameterized by precedence and resource constraints of operations of each instruction. The standard pipeline description based on reservation tables and Gantt charts was extended by additional operation descriptors which enable the detection of data and control hazards, and permit modeling of pipeline flushes. Using the newly introduced L-charts we reduced the parameterization of the pipeline controller to a minimum and at the same time covered typical pipeline controls found in state-of-the-art signal processors. As an example, the application of the LISA model on the TI-TMS320C54x signal processor is presented.**

## 1 Introduction

Simultaneous design of hardware and software can take place at different abstraction levels. At the *HLL-level* compiler and processor are designed jointly in order to obtain optimum performance on selected high-level language constructs. At the *application-level* the on/off-chip hardware has the role of a processing accelerator, or external interface, and is optimized to deliver optimum results for a specific application or a class of them. The goal of *instruction-level* HW/SW co-design is to speedup frequently used instructions by appropriate design of the instruction set architecture (ISA) of the processor. All three levels correspond to *software-based* HW/SW co-design, where the realization in software is the starting point and hardware alternatives are introduced in order to speedup execution. Independent of the level, abstract processor models (machine models in compiler terminology) are an unavoidable part of each HW/SW co-design environment.

Currently available processor models cover a whole spectrum of applications (compilation, software/hardware design, architecture exploration) and design steps (simulation, synthesis, verification). Surprisingly, machine models tailored to specific HW/SW co-design needs are rare and mostly suffer from the rudimentary division between hardware and software.

Machine models of modern DSP and embedded processors follow the classical distinction between instruction set architecture (programmers view of the processor) and hardware implementation. In general purpose processing, hiding of implementational details of the processor is argumented by the programming comfort, and by the need for different hardware implementations of a single instruction set architecture [1]. In HW/SW co-design of DSP and embedded systems, both arguments, and especially the latter one, do not hold. Standard ISA models and description languages do not deliver the detailed pipeline and pin-related information necessary for HW/SW co-design. On the other hand, models and description languages suited for hardware implementation contain a lot of details which are superfluous for software design, and often even for the design of attached hardware. The consequence is a remarkable increase in necessary design effort and simulation time.

In this paper a new machine description language — LISA — is introduced. The development of a new language and its generic machine model was necessary in order to cover the gap between standard ISA models and description languages used in compilers and instruction-level simulators on the one hand, and detailed behavioral/structural models and description languages used in hardware design on the other. The main characteristics of LISA is the operation-level description of the pipeline which is able to model even complex interlocking and bypassing techniques. Instructions consist of multiple operations which are defined as register transfers during a single control step. Depending on the requested accuracy, a control step can be an instruction-, clock-, or phase-cycle. Operation scheduling in LISA is based on modified Gantt charts (L-charts) specifying time and resource allocation of operations and an operation sequencer with an ASAP (As Soon As Possible) operation sequencing strategy. The resulting timed ISA model delivers instruction-, clock-, or phase-accurate timing, depending on the selected control step. Combined with the bit-accurate description of all operations, the model contains the necessary information to produce a clock-/phase-accurate model of processor pins, or the core interface.

LISA is a description language developed to parameterize a generic machine model. The goal is to have a single generic machine model and a single description language covering a whole class of processor architectures (Fig. 1.). The resulting machine model of the target processor can be used for simulation, compilation, or other purposes.

Currently, the primary application domain of LISA is timed ISA simulation used in HLL-, application-, and instruction-level HW/SW co-designs. However, the proposed machine model can be used equally well in other ap-

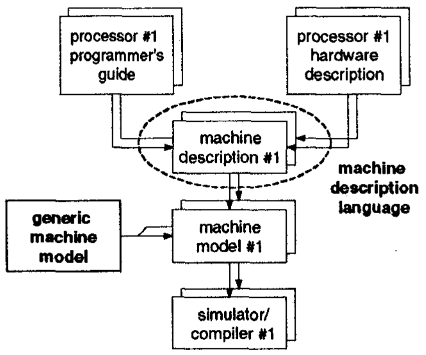

processor #1 programmer's guide
processor #1 hardware description
machine description #1
machine description language
generic machine model
machine model #1
simulator/ compiler #1

Figure 1: Generic Machine Model and Machine Description Language.

plications, as in compilation. The main difference between simulation and compilation lies in the relation between instruction selection/scheduling and behavior modeling. The code generator of a compiler starts with the behavior (mostly in form of a three-address intermediate representation), and has to select and schedule target instructions in order to match the behaviour. In simulations the process is reversed. The instructions are already selected and scheduled, and the task of the simulator is to reproduce the behavior as specified by the simulated program. The essential difference between models used for simulation and compilation is the organization and semantics of instructions which have to be understood by the compiler, but are irrelevant for simulation. The newly introduced LISA pipeline model based on L-charts is equally useful for both simulation and compilation. In this paper our primary focus is on simulation aspects of machine modeling (LISA/S). We hope to present the work on the LISA compilation model (LISA/C) soon.

## 2 Previous Work

The work on LISA was motivated by the wish to enable easy retargeting of our fast compiled simulators [2,3] to various existing, or exploratory architectures. With this goal in mind we analyzed previous work on machine models and machine description languages.

At the first glance, hardware description languages (HDL) seem to be the most natural selection for describing processors. Most of the existing processor hardware models are described using HDLs, like VHDL or Verilog. Unfortunately, using HDL models for real-time software design, or HW/SW co-design, has a number of disadvantages. HDL models contain a lot of superfluous details which are unnecessary for purposes other than design and verification of processor hardware. Although the relevant information, e.g. the ISA, or the control of the pipeline sequencer can be always extracted, it is mostly a highly complex task even for the person who designed the processor. Automatic architecture extraction can be done only for processors with a simple program sequencing logic and data path. Also, HDL descriptions of a single machine can differ enormously in representation level and style complicating the task further.

In the HW/SW co-design field the issue of machine modeling and machine descriptions has not received any specific attention [4,5]. The work mostly concentrated on system models of multiple processors, ASICs, and their interfaces [6]. In the PTOLEMY environment developed at the Berkeley University, a fixed system-level behavioral simulator of the Motorola DSP56000 signal processor was integrated into the Thor hardware design environment [7]. No attempt to cover other processors was reported. The PIXIE simulator developed at the Stanford University provides detailed modeling of the pipeline and memory system. Here the system-level simulator SABLE models the system aspects (e.g. external exceptions) and the I/O system [8]. Again, no details about the machine specification formalism are provided.

Machine description is an unavoidable part of every compiler. Two approaches to compilation-related pipeline modeling can be distinguished. The first one is based on reservation tables. The machine description language MARIL models the pipeline of RISC processors using reservation tables [9]. A similar reservation table based approach is reported for VLIW compilation [10]. In both cases the standard reservation table approach was used, which is unable to capture all the details of the pipeline, such as data/control hazards, interlocks, or pipeline flushes.

The alternative approach is based on latency specification. The machine description formalism nML [11] provides common architectural information for the compiler and the instruction set simulator and uses storage latencies in order to describe pipelining effects [12]. A similar pipeline model is part of the GNU-gcc compiler [13]. Storage latencies provide a highly simplified model of the pipeline, which cannot be used for simulation, and is questionable even as a compilation model.

Our wish to capture more detailed timing information than available at the ISA level, motivated the introduction of operation-level behaviour and scheduling description. In terms of pipeline sequencing representation, LISA follows the same main idea of reservation tables and Gantt charts, as in [9] and [10]. However, in order to enable additional modeling of data/control hazards and pipeline flushes, we extended the modeling ability of Gantt charts by introducing L-charts and operation descriptors.

We posed the following requirements on the machine model:

- **application domain** - real-time software design (machine-dependent finite-word-length analysis, speed/memory optimization), DSP/embedded system design, HW/SW co-verification (on-/off-chip accelerator and interface design), architecture exploration;

- **processor class** - digital signal processors and microcontrollers of low or medium complexity with pipelined, VLIW, and RISC architectures;

- **model accuracy** - selectable instruction, clock or phase accurate timing; bit-accurate register transfers; exact pipeline, interrupt and wait state modeling; spatial accuracy on the software-level (registers, memory), system-level (interrupts, peripherals), and hardware-level (pins);

- **state visibility** - complete state visibility at selected control steps.

The requirements on the machine model have a direct impact on the generic machine model and the machine description language. The main requirement on the generic machine model is to capture the maximum of common features of the target processor class, and in this way minimize description size and description generation time.

The machine specification language should capture the differences between the processors of the given class. The main requirement is low redundancy of the description. Also, the language should have a syntax which can be easily understood by the user and language parser. Below we shall concentrate on the specification of operation constraints and the sequencer of the generic machine model.

## 3 Operation Sequencer of the Generic Machine Model

In order to enable cycle/phase-true modeling, instructions have to be partitioned into operations as basic schedulable units. At each control step $t$ the admissible operations form the transition function $F_t$ which changes the machine state. Figure 2 shows three instructions and their operations. The transition function $F_t = \{O3, P2, Q1\}$ is applied to the machine state at control step $t$. At the next control step $t+1$, the new set of admissible operations

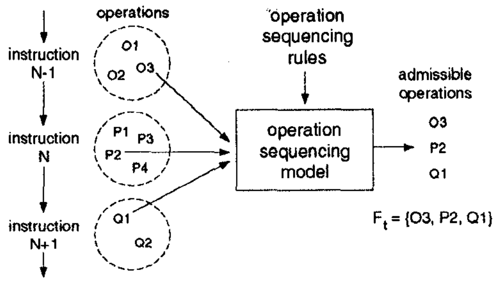


Figure 2: Operation Sequencing.

is determined.

Admissible operations are determined from precedence and resource constraints specified for operations of each instruction. Direct specification using a more general rule-based approach was rejected because of the large number of rules which must accompany each instruction. Additionally, the constraint-based approach has proven to be advantageous if instructions are added or removed from the instruction set. Despite of reduced modeling power compared to the rule-based approach, the constraint-based model captured all the pipeline sequencing strategies of the aforementioned processor class.

A well known formalism to describe precedence and resource constraints of operations are the reservation tables [14]. Reservation tables are two-dimensional representations of resource allocation in the resource-time space. A mark at some place in the table indicates that the corresponding processor resource, such as a bus or a functional unit is in use during the indicated time interval. In Gantt charts [15] instead of reservation marks, operations are uniquely specified. For pipeline [14], and instruction scheduling [9] reservation tables deliver all the necessary details. Clock-accurate modeling, however, requires additional information about operation precedence and Gantt charts are more appropriate.

If we convert the time axis of a Gantt chart into a precedence axis, the Gantt chart describes precedence constraints. Figure 3 shows how the information from the Gantt charts is used by the operation sequencer in order to determine admissible operations at each control step. The sequencer deter-

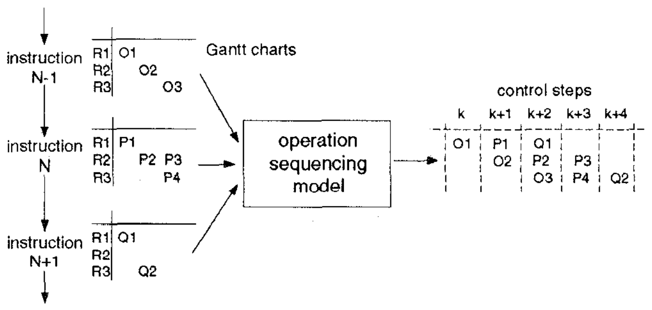


Figure 3: Gantt Chart Sequencing.

mines admissible operations according to the As Soon As Possible (ASAP) principle, thereby accounting for precedence and resource constraints imposed by the Gantt charts. If multiple operations coming from different instructions compete for the same resource, the precedence is determined by the (logical) precedence of their instructions as specified by the program thread.

The main assumption thereby is that the pipeline sequencer inserts bubbles in the pipeline only if resource conflicts have to be resolved. So, processors with out-of-order execution have to be excluded. A great deal of processors we are dealing with, use ASAP sequencing because of the reduced overhead in the realization of the pipeline controller.

In LISA a more compact specification style than Gantt charts is used. Under the assumption that each operation takes a single control step, a Gantt chart can be specified as

```
O1(R1) | O2(R2) | O3(R2) | O4(R3),O5(R4) | O6(R4)
```

where vertical lines, commas and parenthesized objects specify precedence, parallelism and resources, respectively. For example, operation O1 uses resource R1 and has to be scheduled before operation O2 which uses resource R2. Note that no precedence between operations O4 and O5 is specified.

In pipelined architectures three classes of pipeline hazards occur — structural, data and control hazards. These hazards have to be detected and resolved. Gantt chart based models support detection of structural hazards. However, data hazards and control hazards, which can be seen as data hazards on the program counter cannot be detected properly. For example, it is not possible to distinguish between a read-after-write (RAW) hazard and a read-after-read (RAR) access.

In order to detect data and control hazards it is necessary to extend the Gantt chart concept. First, for those operations which access storage resources, like memory or registers, it has to be specified whether the access is a read or a write access. Second, the access has to be announced in advance of the operation which manipulates the storage.

The following example shows two instructions producing a hazard and the introduced notation

```
IF | ID(!w:R0) | IA        | IE(w:R0) |        % instruction #1
     IF        | ID(r:R0)| IA         | IE    % instruction #2
```

Instruction #1 reserves register R0 for writing already during the ID operation by announcing the write access to register R0 using the resource descriptor !w: and it performs the write during the IE operation (specified by the w: descriptor). Instruction #2 (shown shifted) attempts to read register R0 during the decode operation (the r: descriptor is used). Using the supplied information the data hazard on register R0 can be easily detected and resolved using interlocking, as shown by the same example

```
IF | ID(!w:R0) |  IA    | IE(w:R0) |
     IF        |  nop   | nop      | ID(r:R0)| IA | IE
```

The same mechanism is used to describe control hazards and effects of short circuiting.

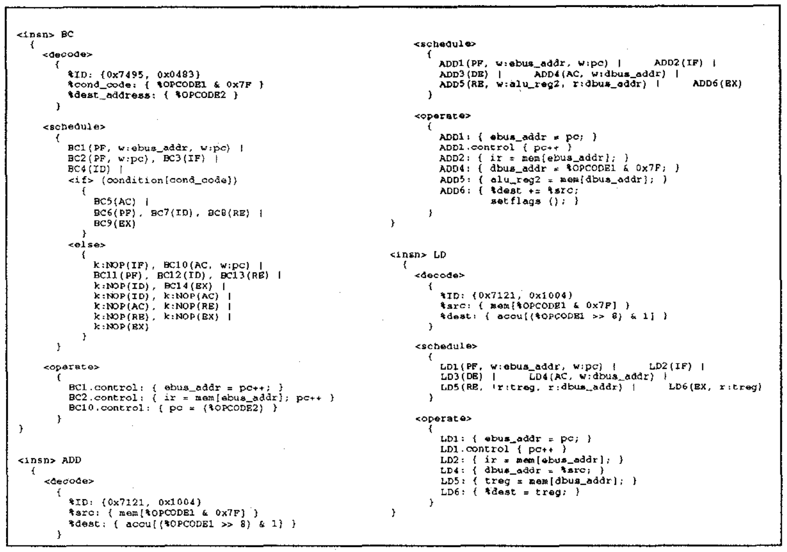
A special class of stalls are pipeline stalls introduced by memory wait states or by the cache. The pipeline behaviour in these cases can be described as resource reservation by external events. In order to describe pipeline flushing, it is necessary to permit some of the control instructions to explicitly change the sequencing mechanism of the generic machine model. According to this mechanism all operations of the instruction which entered the pipeline will be executed. Control instructions mostly do not follow this rule. In order to model pipeline flushing we introduced the k: descriptor for operations (e.g. k:O3). The kill descriptor is described in an example in the following section.

The proposed generic machine model is well suited to model statically scheduled pipelines. We assume that the pipeline fetches an instruction and issues it, unless there is a conflict with previous instructions. Dynamic scheduling of the pipeline, where the hardware rearranges the instruction execution to reduce the stalls, needs a more sophisticated modeling.

## 4 Example

We shall illustrate machine modeling using LISA on the example of the TMS320C54x signal processor. The pipeline of the TMS320C54x has six stages and no interlocking or short circuiting. Our example is based on the description of the pipeline specification of the conditional branch instruction provided in the user's guide of the processor [16].

Figure 4 presents the LISA machine description of three TMS320C54x instructions. Using the keywords **decode, schedule** and **operate** the respective information is specified. The L-chart is specified by the keyword `schedule`. The L-chart of the `BC` (branch conditional) is more complex than that of the single-word `ADD` and `LD` instructions. The `BC` instruction needs 3 cycles if the branch is not taken, and 5 if the branch is taken. The kill descriptor was used to describe flushing of operations which already advanced in the pipeline.



```
<insn> BC
  {
    <decode>
      {
        %ID: {0x7495, 0x0483}
        %cond_code: { %OPCODE1 & 0x7F }
        %dest_address: { %OPCODE2 }
      }

    <schedule>
      {
        BC1(PF, w:ebus_addr, w:pc) |
        BC2(PF, w:pc), BC3(IF) |
        BC4(ID) |
        <if> (condition[cond_code])
          {
            BC5(AC) |
            BC6(PF), BC7(ID), BC8(RE) |
            BC9(EX)
          }
        <else>
          {
            k:NOP(IF), BC10(AC, w:pc) |
            BC11(PF), BC12(ID), BC13(RE) |
            k:NOP(ID), BC14(EX) |
            k:NOP(ID), k:NOP(AC) |
            k:NOP(AC), k:NOP(RE) |
            k:NOP(RE), k:NOP(EX) |
            k:NOP(EX)
          }
      }

    <operate>
      {
        BC1.control: { ebus_addr = pc++; }
        BC2.control: { ir = mem[ebus_addr]; pc++ }
        BC10.control: { pc = (%OPCODE2) }
      }
}


<insn> ADD
  {
    <decode>
      {
        %ID: {0x7121, 0x1004}
        %src: { mem[%OPCODE1 & 0x7F] }
        %dest: { accu[(%OPCODE1 >> 8) & 1] }
      }

    <schedule>
      {
        ADD1(PF, w:ebus_addr, w:pc) |     ADD2(IF) |
        ADD3(DE) |     ADD4(AC, w:dbus_addr) |
        ADD5(RE, w:alu_reg2, r:dbus_addr) |     ADD6(EX)
      }

    <operate>
      {
        ADD1: { ebus_addr = pc; }
        ADD1.control { pc++ }
        ADD2: { ir = mem[ebus_addr]; }
        ADD4: { dbus_addr = %OPCODE1 & 0x7F; }
        ADD5: { alu_reg2 = mem[dbus_addr]; }
        ADD6: { %dest += %src;
                setflags (); }
      }
}


<insn> LD
  {
    <decode>
      {
        %ID: {0x7121, 0x1004}
        %src: { mem[%OPCODE1 & 0x7F] }
        %dest: { accu[(%OPCODE1 >> 8) & 1] }
      }

    <schedule>
      {
        LD1(PF, w:ebus_addr, w:pc) |     LD2(IF) |
        LD3(DE) |     LD4(AC, w:dbus_addr) |
        LD5(RE, r:treg, r:dbus_addr) |     LD6(EX, r:treg)
      }

    <operate>
      {
        LD1: { ebus_addr = pc; }
        LD1.control { pc++ }
        LD2: { ir = mem[ebus_addr]; }
        LD4: { dbus_addr = %src; }
        LD5: { treg = mem[dbus_addr]; }
        LD6: { %dest = treg; }
      }
}
```

Figure 4: LISA description of TMS320C54x instructions.

The program segment of the example consists of three sequential instructions {`BC`,`ADD`,`LD`} and the `ADD` instruction on the target address of the branch instruction. The scheduling of operations delivered by the ASAP scheduler of the LISA generic machine model is presented on the following two tables. Table 1 provides the operation scheduling if the branch is not taken. Note that the fetch of the `ADD` instruction happens in the 4th cycle and decode in the 6th. The sequencer of the generic machine model was able to successfully model the pipeline using only the information provided in L-charts.

Table 2 shows the case when the branch is taken. In this case the `BC` instruction specifies kill resources using the kill descriptor. Without the kill descriptor the `ADD3` operation would execute in the 6th cycle. The kill de-

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| PF | BC1 | BC2 | ADD1 | LD1 | BC6 | | | | | |
| IF | | BC3 | | ADD2 | LD2 | | | | | |
| ID | | | BC4 | | BC7 | ADD3 | LD3 | | | |
| AC | | | | BC5 | | | ADD4 | LD4 | | |
| RE | | | | | BC8 | | | ADD5 | LD5 | |
| EX | | | | | | BC9 | | | ADD6 | LD6 |

Table 1: Scheduling of the BC Instruction (branch not taken).

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| PF | BC1 | BC2 | ADD1 | LD1 | BC11 | ADD1 | | | | |
| IF | | BC3 | | ADD2 | LD2 | | ADD2 | | | |
| ID | | | BC4 | | BC12 | NOP | NOP | ADD3 | | |
| AC | | | | BC10 | | | NOP | NOP | ADD4 | |
| RE | | | | | BC13 | | | NOP | NOP | ... |
| EX | | | | | | BC14 | | | NOP | ... |

Table 2: Scheduling of the BC Instruction (branch taken).

scriptor simply overloads the **ADD3** operation with its own operation, in this case **NOP**. In this way the operation cancelation takes place to stop further propagation (issuing) of instructions **ADD** and **LD**.

## 5 Conclusions

Recent examples of DSP processor/compiler, and processor/accelerator co-designs have emphasized the importance of accurate machine models. In the same time the model-before-silicon approach is established nowadays as the main strategy to shorten time-to-market. Tools, and applications are developed using the model before the silicon of the processor is available.

In the paper the new machine description language LISA and its generic machine model have been introduced. LISA enables fast and comfortable specification of DSP and embedded processor architectures used in HW/SW co-design. Although currently targeted primarily to simulators, the generic machine model of LISA can be used equally well for other purposes, e.g. compilation.

Main contribution of LISA is the introduction of extended Gantt charts or L-charts as we named them. Compared to classical reservation tables, and Gantt charts, the L-charts additionally permit modeling of data/control hazards, as well as pipeline flushing. In this way pipeline interlocking and bypassing can be modeled. Operation sequencing in LISA is modeled using a simple ASAP sequencing strategy which obeys the time and resource constraints specified in the L-chart of each instruction. Limitation of the approach is that it cannot model out-of-order execution found in superscalar processors.

Our future work will concentrate on two issues. First, we have to explore further the set of architectures which can be modeled with LISA and the L-charts. Second, we shall extend LISA to efficiently support compilation (LISA/C) and in this way produce a unique machine description language for processor/compiler co-design.

## References

[1] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., 1990.

[2] V. Živojnović, S. Tjiang, and H. Meyr, "Compiled simulation of programmable DSP architectures," in *Proc. of IEEE Workshop on VLSI in Signal Processing, Osaka, Japan*, pp. 187–196, Oct. 1995.

[3] V. Živojnović and H. Meyr, "Compiled HW/SW co-simulation," in *Proc. Design Automation Conference*, (Las Vegas, NV), June 1996.

[4] in *Int. Workshop on Hardware-Software Co-Design, Cambridge, MA*, 1993.

[5] G. De Micheli and M. Sami, *Hardware/Software Co-Design.* Kluwer Academic Publishers, 1995.

[6] D. Gajski and F. Vahid, "Specification and design of embedded hardware-software systems," *IEEE Design & Test of Computers*, spring 1995.

[7] A. Kalavade and E. Lee, "A hardware-software codesign methodology for DSP applications," *IEEE Design & Test of Computers*, pp. 16–28, Sept. 1993.

[8] J. Hennessy and M. Heinrich, "Hardware/software codesign of processors: concepts and examples," in *Hardware/Software Co-Design* (G. De Micheli and M. Sami, eds.), Kluwer Academic Publishers, 1995.

[9] D. Bradlee, R. Henry, and S. Eggers, "The Marion system for retargetable instruction scheduling," in *Proc. ACM SIGPLAN'91 Conference on Programming Language Design and Implementation, Toronto, Canada*, pp. 229–240, 1991.

[10] B. Rau, "VLIW compilation driven by a machine description database," in *Proc. 2nd Code Generation Workshop, Leuven, Belgium*, 1996.

[11] M. Freericks, "The nML machine description formalism," Tech. Rep. 1991/15, Technische Universität Berlin, Fachbereich Informatik, Berlin, 1991.

[12] A. Fauth, J. Van Praet, and M. Freericks, "Describing instruction set processors using nML," in *Proc. European Design and Test Conf., Paris*, Mar. 1995.

[13] R. Stallman, *Using and Porting GNU CC.* Free Software Foundation, Cambridge,MA, 2.4 ed., 1993.

[14] P. Kogge, *The Architecture of Pipelined Computers.* McGraw Hill, New York, NY, 1981.

[15] K. Baker, *Introduction to Sequencing and Scheduling.* John Wiley & Sons, 1974.

[16] Texas Instruments Inc., *TMS320C54x User's Guide*, 1995.