Institute for System Programming of the Russian Academy of Sciences

nML Reference Manual (UNDER DEVELOPMENT)

Moscow 2016

Contents

Chapter 1

Introduction

1.1 Disclaimer

This reference manual describes syntax and semantics of the nML architecture description language. It covers only language facilities supported by the ISP RAS version of nML. It may differ from documentation provided by other vendors as there are differences in language implementations.

1.2 Overview

nML is an architecture description language (ADL) used to describe the instruction set architecture (ISA) of an arbitrary microprocessor. It is a flexible and easy-to-use language based on attribute grammar. nML was designed to provide a retargetable way to specify microprocessor architecture for various microprocessor-related software tools including instruction-set simulators, assemblers, disassemblers, compiler back-ends etc. It works at the instruction-set level concentrating on behavioral properties and hiding implementation details. An nML specification represents a programmer's model of the microprocessor which covers the following aspects:

- supported data types;
- registers and memory;
- addressing modes;
- syntax and semantics of instructions.

nML uses a hierarchical tree-like structure to describe an instruction set. Such a structure facilitates grouping related instructions and sharing their common parts. An instruction is described as a path in the tree from the root node to a leaf node. The set of all possible paths represents an instruction set. A node describes a primitive operation responsible for some task within an instruction. Each node has certain attributes that can be shared with its descendants. Actions performed by instructions are described as operations with registers and memory represented by bit vectors of arbitrary size.

A specification in nML starts with definitions of types and global constants. For example, a type definition for a 32-bit unsigned integer looks as follows:

```
let WORD_SIZE = 32
type WORD = card(WORD_SIZE)
```

Type definitions and constants can be used to describe registers and memory. In addition to registers and memory, it is also possible to define temporary variables that are internal abstractions provided by nML to store intermediate results of operations. They do not correspond to any data storage in real hardware and do not save their data across instruction calls. Also, there is often a need to specify some properties of the described model. For this purpose, special constants are used. For example, the code below defines general-purpose registers, memory and a temporary variable. Also, it includes a special constant to establish a correspondence between the general purpose register number 15 and the program counter (PC). Here is the code:

1.3 Constants

A declaration like

```
let A=100
```

declares a global constant A to have the value 100. Such a constant might be used in every context its value could stand. Any constant may be defined only once. Constants may be used to extend nML: Any information about a machine that can be given with a single number or string can easily be defined as a constant (with a default value, so that standard nML descriptions still work).

In core nML, there is just one such constant (or global parameter). This is the pipeline factor. On machines with an instruction pipeline visible to the programmer, there are delay slots whenever a jump occurs. Usually, there is one such slot, but two are not unheard of. A declaration

```
let pipeline_factor=1
```

introduces one delay slot after each instruction that changes the program counter. The default value is 0.

1.4 Data Types

A data type specifies the format of values stored in registers or memory. nML supports the following data types:

- int(N): N-bit signed integer data type. Negative numbers are stored in two's complement form. The range of possible values is [-2n-1 ... 2n-1 - 1].
- card(N): N-bit unsigned integer data type. The range of possible values is [0 ... 2n - 1].
- float(N, M): IEEE 754 floating point number, where fraction size is N and exponent size is M. The resulting type size is N + M +1 bits, where 1 is an implicitly added bit for store the sign. Supported floating-point formats include:
  - 32-bit single-precision. Defined as float(23, 8).
  - 64-bit double-precision. Defined as float(52, 11).
  - 80-bit double-extended-precision. Defined as float(64, 15).
  - 128-bit quadruple-precision. Defined as float(112, 15).

nML allows declaring aliases for data types. Here is a simple type declaration:

```
type DWORD = card(32)
```

In this example, type is a reserved word, DWORD is the declared alias type name, the card(32) is the actual data type.

1.4.1 Converting Data Types

In nML, type conversion is performed explicitly using the following functions:

- sign_extend(type, value). Sign-extends the value to the specified type. Applied to any types. Requires the new type to be larger or equal to the original type.
- zero_extend(type, value). Zero-extends the value to the specified type. Applied to any types. Requires the new type to be larger or equal to the original type.
- coerce(type, value). Converts an integer value to another integer type. For smaller types the value is truncated, for larger types it is extended. If the original type is a signed integer, sign extension is performed. Otherwise, zero-extension is performed.
- cast(type, value). Reinterprets the value as the specified type. Applied to any types. The new type must be of the same size as the original type.
- int_to_float(type, value). Converts 32 and 64-bit integers into 32, 64, 80 and 128-bit floating-point values (IEEE 754).
- float_to_int(type, value). Converts 32, 64, 80 and 128-bit floating-point values (IEEE 754) into 32 and 64-bit integers.
- float_to_float(type, value). Converts 32, 64, 80 and 128-bit floating-point values (IEEE 754) into each other.

```
type HWORD = card(16)
type DWORD = card(64)

type INT   = int(32)
type LONG  = int(64)

reg GPR [32, DWORD]

mode R (i : card(5)) = GPR[i]
  syntax = format("r%d", i)

op lui (rt: R, immediate: HWORD)
  syntax = format("lui␣%s,␣0x%x", rt.syntax, immediate)
  action = {
    rt = coerce(LONG, (coerse(INT, immedidate) << 16));
  }
```

Chapter 2

Appendixes

2.1 Grammar of nML

2.2 References

Bibliography

[1] M. Freericks. *The nML Machine Description Formalism*. Technical Report TR SM-IMP/DIST/08, TU Berlin CS Department, 1993.