SPIR-V Machine-readable Grammar

Lei Zhang, Google

January 2, 2018

Contents

1 Introduction

This document defines the schema used by the machine-readable JSON grammar file for SPIR-V. This document is **not** the SPIR-V specification fully defining SPIR-V itself.

1.1 Goals and Conventions

- The machine-readable JSON grammar primarily aims to describe the syntax of various SPIR-V instructions. Semantics and validation rules remain in the human-readable SPIR-V specification.
- If a grammar struct has no particular attribute, e.g., `OpNop` has no operands and requires no additional capabilities, the corresponding key for that attribute in the JSON object is excluded for terseness.
- Keys in various JSON objects are in the `snake_case` format, while values are in the `CamelCase` format.
- Operand kinds of a specific category tend to have the category name as the prefix.

2 Top Level Structure

The whole JSON grammar file consists of a single JSON object containing the following key-value pairs:

```
{
  "copyright" : [ ... ],
  "magic_number" : "...",
  "major_version" : ...,
  "minor_version" : ...,
  "revision" : ...,
  "instructions" : [ ... ],
  "operand_kinds" : [ ... ]
}
```

`magic_number`
: A string. The magic number for a SPIR-V module in the hexadecimal format.

`major_version`
: An integer. The major version of the SPIR-V represented in this grammar file.

`minor_version`
: An integer. The minor version of the SPIR-V represented in this grammar file.

`revision`
: An integer. The revision of the SPIR-V represented in this grammar file.

`instructions`
: A list of objects. Each object contains information about a specific instruction in the SPIR-V specification, including its name, opcode, required capabilities, layout of operands. The order of these objects dictates nothing. See the JSON Objects for Instructions Section for more details.

`operand_kinds`
: A list of objects. Each object contains information about a specific operand kind, including its name, category, possible values, required capabilities, used in the operand layouts of instructions. See the JSON Objects for Operand Kinds Section for more details.

3 Grammar Objects

3.1 JSON objects for instructions

This kind of JSON object contains the information about a specific SPIR-V instruction. It has the following key-value pairs:

```
{
  "opname" : "...",
  "opcode" : ...,
  "operands" : [
    { "kind" : "...", quantifier : "...", name : "..." },
    { "kind" : "...", quantifier : "...", name : "..." },
    ...
  ],
  "capabilities" : [ ... ]
}
```

`opname`
A string. The name of this instruction, starting with `Op`.

`opcode`
An integer. The opcode enumerant for this instruction.

`operands`
A list of objects. *Optional*. Each object contains the information about a logical operand for this instruction:

`kind`
A string. The kind of this operand.

`quantifier`
A string. *Optional*. If this key is missing, it means this operand should appear exactly once. Otherwise, the value can only be `"?"` or `"*"`. If the value is `"?"`, it means this operand appears zero or one time. If the value is `"*"`, it means this operand appears zero or more times.

`name`
A string. *Optional*. A short descriptive name for this operand.

`capabilities`
A list of strings. *Optional*. If existing, each string is the name of a required capability for this instruction.

3.2 Operand categories

Operand kinds are grouped into categories according to their possible values. Right now, there are only five categories:

BitEnum
For an operand kind belonging to this category, its value is a mask, which is formed by combining the bits specified as enumerants in an enum. See the BitEnum Operand Kind Section for more details.

ValueEnum
For an operand kind belonging to this category, its value is an enumerant from an enum. See the ValueEnum Operand Kind Section for more details.

Id
For an operand kind belonging to this category, its value is an <id> definition or reference.

Literal
For an operand kind belonging to this category, its value is an literal number or string.

Composite
For an operand kind belonging to this category, its value is composed from operand values from the above categories. See the Composite Operand Kind Section for more details.

3.3 JSON objects for operand kinds

This kind of JSON object contains the information about a specific operand kind. It has the following key-value pairs:

```
{
  "category" : "...",
  "kind" : "...",
  "doc" : "...",
  ...
}
```

`category`
A string. The category of this operand kind.

`kind`
A string. The name of this operand kind.

`doc`
A string. *Optional*. The human-readable definition of this operand kind.

Depending on the category of this operand kind, there may be more key-value pairs as explained in the follwing subsections.

3.3.1 The BitEnum operand kind

Apart from the general key-value pairs, This kind of JSON object additionaly has:

```
{
  ...,
  "enumerants": [
    {
      "enumerant" : "...",
      "value" : "...",
      "capabilities" : [ ... ],
      "parameters" : [
        { "kind" : "..." },
        ...
      ]
    }
}
```

`enumerants`
A list of objects. Each object describes a possible bitflag for this operand kind:

`enumerant`
A string. The name of this bitflag.

`value`
A string. The hexidecimal bit value of this bitflag.

`capabilities`
A list of strings. *Optional*. If existing, each string is the name of a required capability for this bitflag.

`parameters`
A list of objects. *Optional*. Each object describes a logical parameter for this bitflag:

`kind`
A string. The kind of this operand.

3.3.2 The ValueEnum operand kind

Apart from the general key-value pairs, This kind of JSON object additionaly has:

```
{
  ...,
  "enumerants": [
    {
      "enumerant" : "...",
      "value" : ...,
      "capabilities" : [ ... ],
      "parameters" : [
        { "kind" : "..." },
        ...
      ]
    }
}
```

`enumerants`
A list of objects. Each object describes an possible enumerant for this operand kind:

`enumerant`
A string. The name of this enumerant.

`value`
An integer. The value of this enumerant.

`capabilities`
A list of strings. *Optional*. If existing, each string is the name of a required capability for this enumerant.

`extensions`
A list of strings. *Optional*. The names of extensions that enable this feature. If absent, the feature is always enabled.

`parameters`
A list of objects. *Optional*. Each object describes a logical parameter for this enumerant:

`kind`
A string. The kind of this operand.

The Capability operand kind

For example, capabilities are represented as a **ValueEnum** operand kind:

```
{
  "category" : "ValueEnum",
  "kind" : "Capability",
  "enumerants" : [
    {
      "enumerant" : "Matrix",
      "value" : 0
    },
    {
      "enumerant" : "Shader",
      "value" : 1,
      "capabilities" : [ "Matrix" ]
    },
    {
      "enumerant" : "Geometry",
      "value" : 2,
      "capabilities" : [ "Shader" ]
```

```
    },
    {
      "enumerant" : "Tessellation",
      "value" : 3,
      "capabilities" : [ "Shader" ]
    },
    {
      "enumerant" : "Addresses",
      "value" : 4
    },
    ...
}
```

3.3.3 The Id operand kind

This kind of JSON object has no additional key-value pairs. All the kinds in this category are prefixed with `Id`.

3.3.4 The Literal operand kind

This kind of JSON object has no additional key-value pairs. All the kinds in this category are prefixed with `Literal`.

3.3.5 The Composite operand kind

This kind of JSON object additionaly contains:

```
{
  ...,
  bases: [ ... ]
}
```

`bases`
: A list of strings. Each string names an operand kind forming this **Composite** kind.

When an operand is of the **Composite** kind, all the operands listed in its `bases` must appear in the concrete case, in the given order.

Right now there are only three kinds defined in this category:

- `PairIdRefIdRef`: used by `OpPhi`.
- `PairLiteralIntegerIdRef`: used by `OpSwitch`.
- `PairIdRefLiteralInteger`: used by `OpGroupMemberDecorate`.

4 Example Usage

Please see the SPIRV-Tools project for an example of how this grammar can be used.