



# **cuda-binary-utilities**

*Release 12.3*

**NVIDIA**

**Oct 10, 2023**



# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>What is a CUDA Binary?</b>                    | <b>3</b>  |
| <b>2</b> | <b>Differences between cuobjdump and nvdiasm</b> | <b>5</b>  |
| <b>3</b> | <b>Command Option Types and Notation</b>         | <b>7</b>  |
| <b>4</b> | <b>cuobjdump</b>                                 | <b>9</b>  |
| 4.1      | Usage . . . . .                                  | 9         |
| 4.2      | Command-line Options . . . . .                   | 12        |
| <b>5</b> | <b>nvdiasm</b>                                   | <b>15</b> |
| 5.1      | Usage . . . . .                                  | 15        |
| 5.2      | Command-line Options . . . . .                   | 22        |
| <b>6</b> | <b>Instruction Set Reference</b>                 | <b>25</b> |
| 6.1      | Maxwell and Pascal Instruction Set . . . . .     | 25        |
| 6.2      | Volta Instruction Set . . . . .                  | 29        |
| 6.3      | Turing Instruction Set . . . . .                 | 34        |
| 6.4      | Ampere and Ada Instruction Set . . . . .         | 40        |
| 6.5      | Hopper Instruction Set . . . . .                 | 46        |
| <b>7</b> | <b>cu++filt</b>                                  | <b>55</b> |
| 7.1      | Usage . . . . .                                  | 55        |
| 7.2      | Command-line Options . . . . .                   | 56        |
| 7.3      | Library Availability . . . . .                   | 56        |
| <b>8</b> | <b>nvprune</b>                                   | <b>59</b> |
| 8.1      | Usage . . . . .                                  | 59        |
| 8.2      | Command-line Options . . . . .                   | 59        |
| <b>9</b> | <b>Notices</b>                                   | <b>61</b> |
| 9.1      | Notice . . . . .                                 | 61        |
| 9.2      | OpenCL . . . . .                                 | 62        |
| 9.3      | Trademarks . . . . .                             | 62        |



## CUDA Binary Utilities

The application notes for cuobjdump, nvdiasm, cu++filt, and nvprune.

This document introduces cuobjdump, nvdiasm, cu++filt and nvprune, four CUDA binary tools for Linux (x86, ARM and P9), Windows, Mac OS and Android.



---

# Chapter 1. What is a CUDA Binary?

A CUDA binary (also referred to as cubin) file is an ELF-formatted file which consists of CUDA executable code sections as well as other sections containing symbols, relocators, debug info, etc. By default, the CUDA compiler driver `nvcc` embeds cubin files into the host executable file. But they can also be generated separately by using the “`-cubin`” option of `nvcc`. cubin files are loaded at run time by the CUDA driver API.

---

**Note:** For more details on cubin files or the CUDA compilation trajectory, refer to [NVIDIA CUDA Compiler Driver NVCC](#).

---





---

## Chapter 2. Differences between cuobjdump and nvdiasm

CUDA provides two binary utilities for examining and disassembling cubin files and host executables: cuobjdump and nvdiasm. Basically, cuobjdump accepts both cubin files and host binaries while nvdiasm only accepts cubin files; but nvdiasm provides richer output options.

Here's a quick comparison of the two tools:

Table 1: Table 1. Comparison of cuobjdump and nvdiasm

|   | <b>cuobjdump</b> | <b>nvdiasm</b> |
|---|------------------|----------------|
| Disassemble cubin   | Yes              | Yes            |
| Extract ptx and extract and disassemble cubin from the following input files: <ul style="list-style-type: none"><li>▶ Host binaries<ul style="list-style-type: none"><li>▶ Executables</li><li>▶ Object files</li><li>▶ Static libraries</li></ul></li><li>▶ External fatbinary files</li></ul> | Yes              | No             |
| Control flow analysis and output  | No               | Yes            |
| Advanced display options  | No               | Yes            |



---

## Chapter 3. Command Option Types and Notation

This section of the document provides common details about the command line options for the following tools:

- ▶ `cuobjdump`
- ▶ `nvdiasm`
- ▶ `nvprune`

Each command-line option has a long name and a short name, which are interchangeable with each other. These two variants are distinguished by the number of hyphens that must precede the option name, i.e. long names must be preceded by two hyphens and short names must be preceded by a single hyphen. For example, `-I` is the short name of `--include-path`. Long options are intended for use in build scripts, where size of the option is less important than descriptive value and short options are intended for interactive use.

The tools mentioned above recognize three types of command options: boolean options, single value options and list options.

Boolean options do not have an argument, they are either specified on a command line or not. Single value options must be specified at most once and list options may be repeated. Examples of each of these option types are, respectively:

```
Boolean option : nvdiasm --print-raw <file>
Single value   : nvdiasm --binary SM70 <file>
List options   : cuobjdump --function "foo,bar,foobar" <file>
```

Single value options and list options must have arguments, which must follow the name of the option by either one or more spaces or an equals character. When a one-character short name such as `-I`, `-l`, and `-L` is used, the value of the option may also immediately follow the option itself without being separated by spaces or an equal character. The individual values of list options may be separated by commas in a single instance of the option or the option may be repeated, or any combination of these two cases.

Hence, for the two sample options mentioned above that may take values, the following notations are legal:

```
-o file
-o=file
-Idir1,dir2 -I=dir3 -I dir4,dir5
```

For options taking a single value, if specified multiple times, the rightmost value in the command line will be considered for that option. In the below example, `test.bin` binary will be disassembled

assuming SM75 as the architecture.

```
nvdiasm.exe -b SM70 -b SM75 test.bin  
nvdiasm warning : incompatible redefinition for option 'binary', the last value of  
↔this option was used
```

For options taking a list of values, if specified multiple times, the values get appended to the list. If there are duplicate values specified, they are ignored. In the below example, functions `foo` and `bar` are considered as valid values for option `--function` and the duplicate value `foo` is ignored.

```
cuobjdump --function "foo" --function "bar" --function "foo" -sass test.cubin
```

---

# Chapter 4. cuobjdump

cuobjdump extracts information from CUDA binary files (both standalone and those embedded in host binaries) and presents them in human readable format. The output of cuobjdump includes CUDA assembly code for each kernel, CUDA ELF section headers, string tables, relocators and other CUDA specific sections. It also extracts embedded ptx text from host binaries.

For a list of CUDA assembly instruction set of each GPU architecture, see [Instruction Set Reference](#).

## 4.1. Usage

cuobjdump accepts a single input file each time it's run. The basic usage is as following:

```
cuobjdump [options] <file>
```

To disassemble a standalone cubin or cubins embedded in a host executable and show CUDA assembly of the kernels, use the following command:

```
cuobjdump -sass <input file>
```

To dump cuda elf sections in human readable format from a cubin file, use the following command:

```
cuobjdump -elf <cubin file>
```

To extract ptx text from a host binary, use the following command:

```
cuobjdump -ptx <host binary>
```

Here's a sample output of cuobjdump:

```
$ cuobjdump a.out -sass -ptx
Fatbin elf code:
=====
arch = sm_70
code version = [1,7]
producer = cuda
host = linux
compile_size = 64bit
identifier = add.cu

code for sm_70
      Function : _Z3addPiS_S_
.headerflags  @"EF_CUDA_SM70 EF_CUDA_PTX_SM(EF_CUDA_SM70)"
```

(continues on next page)

(continued from previous page)

```

/*0000*/      IMAD.MOV.U32 R1, RZ, RZ, c[0x0][0x28] ; /* 0x00000a00ff017624 */
/*0010*/      @!PT SHFL.IDX PT, RZ, RZ, RZ, RZ ; /* 0x000fd000078e00ff */
/*0020*/      IMAD.MOV.U32 R2, RZ, RZ, c[0x0][0x160] ; /* 0x00005800ff027624 */
/*0030*/      MOV R3, c[0x0][0x164] ; /* 0x000fe200078e00ff */
/*0040*/      IMAD.MOV.U32 R4, RZ, RZ, c[0x0][0x168] ; /* 0x00005a00ff047624 */
/*0050*/      MOV R5, c[0x0][0x16c] ; /* 0x00005b0000057a02 */
/*0060*/      LDG.E.SYS R2, [R2] ; /* 0x000fcc000000f00 */
/*0070*/      LDG.E.SYS R5, [R4] ; /* 0x0000000002027381 */
/*0080*/      IMAD.MOV.U32 R6, RZ, RZ, c[0x0][0x170] ; /* 0x00005c00ff067624 */
/*0090*/      MOV R7, c[0x0][0x174] ; /* 0x00005d0000077a02 */
/*00a0*/      IADD3 R9, R2, R5, RZ ; /* 0x0000000502097210 */
/*00b0*/      STG.E.SYS [R6], R9 ; /* 0x0000000906007386 */
/*00c0*/      EXIT ; /* 0x000000000000794d */
/*00d0*/      BRA 0xd0; /* 0x000fea000380000 */
/*00e0*/      NOP; /* 0xfffffff000007947 */
/*00f0*/      NOP; /* 0x000fc0000383ffff */
/*0100*/      NOP; /* 0x0000000000007918 */
/*0110*/      NOP; /* 0x000fc00000000000 */
/*0120*/      NOP; /* 0x0000000000007918 */
/*0130*/      NOP; /* 0x000fc00000000000 */
.....

```

Fatbin ptx code:

```

=====
arch = sm_70
code version = [7,0]
producer = cuda
host = linux
compile_size = 64bit
compressed
identifier = add.cu

.version 7.0
.target sm_70
.address_size 64

.visible .entry _Z3addPiS_S_(
.param .u64 _Z3addPiS_S__param_0,
.param .u64 _Z3addPiS_S__param_1,
.param .u64 _Z3addPiS_S__param_2
)
{
.reg .s32 %r<4>;
.reg .s64 %rd<7>;

```

(continues on next page)

(continued from previous page)

```
ld.param.u64 %rd1, [_Z3addPiS_S__param_0];
ld.param.u64 %rd2, [_Z3addPiS_S__param_1];
ld.param.u64 %rd3, [_Z3addPiS_S__param_2];
cvta.to.global.u64 %rd4, %rd3;
cvta.to.global.u64 %rd5, %rd2;
cvta.to.global.u64 %rd6, %rd1;
ld.global.u32 %r1, [%rd6];
ld.global.u32 %r2, [%rd5];
add.s32 %r3, %r2, %r1;
st.global.u32 [%rd4], %r3;
ret;
}
```

As shown in the output, the a.out host binary contains cubin and ptx code for sm\_70.

To list cubin files in the host binary use `-lelf` option:

```
$ cuobjdump a.out -lelf
ELF file 1: add_new.sm_70.cubin
ELF file 2: add_new.sm_75.cubin
ELF file 3: add_old.sm_70.cubin
ELF file 4: add_old.sm_75.cubin
```

To extract all the cubins as files from the host binary use `-xelf all` option:

```
$ cuobjdump a.out -xelf all
Extracting ELF file 1: add_new.sm_70.cubin
Extracting ELF file 2: add_new.sm_75.cubin
Extracting ELF file 3: add_old.sm_70.cubin
Extracting ELF file 4: add_old.sm_75.cubin
```

To extract the cubin named `add_new.sm_70.cubin`:

```
$ cuobjdump a.out -xelf add_new.sm_70.cubin
Extracting ELF file 1: add_new.sm_70.cubin
```

To extract only the cubins containing `_old` in their names:

```
$ cuobjdump a.out -xelf _old
Extracting ELF file 1: add_old.sm_70.cubin
Extracting ELF file 2: add_old.sm_75.cubin
```

You can pass any substring to `-xelf` and `-xptx` options. Only the files having the substring in the name will be extracted from the input binary.

To dump common and per function resource usage information:

```
$ cuobjdump test.cubin -res-usage

Resource usage:
Common:
  GLOBAL:56 CONSTANT[3]:28
Function calculate:
  REG:24 STACK:8 SHARED:0 LOCAL:0 CONSTANT[0]:472 CONSTANT[2]:24 TEXTURE:0 SURFACE:0
  ↪SAMPLER:0
Function mysurf_func:
```

(continues on next page)

(continued from previous page)

```
REG:38 STACK:8 SHARED:4 LOCAL:0 CONSTANT[0]:532 TEXTURE:8 SURFACE:7 SAMPLER:0  
Function mytexsampler_func:  
REG:42 STACK:0 SHARED:0 LOCAL:0 CONSTANT[0]:472 TEXTURE:4 SURFACE:0 SAMPLER:1
```

Note that value for REG, TEXTURE, SURFACE and SAMPLER denotes the count and for other resources it denotes no. of byte(s) used.

## 4.2. Command-line Options

Table 2 contains supported command-line options of cuobjdump, along with a description of what each option does. Each option has a long name and a short name, which can be used interchangeably.



Table 1: Table 2. cuobjdump Command-line Options

| Option (long)                              | Option (short) | Description  |
|--|----------------|--|
| --all-fatbin                               | -all           | Dump all fatbin sections. By default will only dump contents of executable fatbin (if exists), else relocatable fatbin if no executable fatbin.  |
| --dump-elf                                 | -elf           | Dump ELF Object sections.  |
| --dump-elf-symbols                         | -sym           | Dump ELF symbol names.   |
| --dump-ptx                                 | -ptx           | Dump PTX for all listed device functions.  |
| --dump-sass                                | -sass          | Dump CUDA assembly for a single cubin file or all cubin files embedded in the binary.  |
| --dump-resource-usage                      | -usage         | Print resource usage for each ELF. Useful in getting all the resource usage information at one place.  |
| --extract-elf <partial file name>, ..      | -xelf          | Extract ELF file(s) name containing <partial file name> and save as file(s). Use all to extract all files. To get the list of ELF files use -lelf option. Works with host executable/object/library and external fatbin. All dump and list options are ignored with this option. |
| --extract-ptx <partial file name>, ..      | -xptx          | Extract PTX file(s) name containing <partial file name> and save as file(s). Use all to extract all files. To get the list of PTX files use -lptx option. Works with host executable/object/library and external fatbin. All dump and list options are ignored with this option. |
| --extract-text <partial file name>, ..     | -xtex          | Extract text binary encoding file(s) name containing <partial file name> and save as file(s). Use 'all' to extract all files. To get the list of text binary encoding use -ltex option. All 'dump' and 'list' options are ignored with this option.                              |
| --function <function name>, ...            | -fun           | Specify names of device functions whose fat binary structures must be dumped.  |
| --function-index <function index>, ...     | -fidx          | Specify symbol table index of the function whose fat binary structures must be dumped.   |
| --gpu-architecture <gpu architecture name> | -arch          | Specify GPU Architecture for which information should be dumped. Allowed values for this option: sm_50, sm_52, sm_53, sm_60, sm_61, sm_62, sm_70, sm_72, sm_75, sm_80, sm_86, sm_87, sm_89, sm_90, sm_90a.   |
| --help                                     | -h             | Print this help information on this tool.  |
| --list-elf                                 | -lelf          | List all the ELF files available in the fatbin. Works with host executable/object/library and external fatbin. All other options are ignored with this flag. This can be used to select particular ELF with -xelf option later.  |
| --list-ptx                                 | -lptx          | List all the PTX files available in the fatbin. Works with host executable/object/library and external fatbin. All other options are ignored with this flag. This can be used to select particular PTX with -xptx option later.  |
| --list-text                                | -ltex          | List all the text binary function names available in the fatbin. All other options are ignored with the flag. This can be used to select particular function with -xtex option later.  |
| --options-file <file>, ...                 | -optf          | Include command line options from specified file.  |
| --sort-functions                           | -sort          | Sort functions when dumping sass.  |
| --version                                  | -V             | Print version information on this tool.  |



---

# Chapter 5. nvdiasm

nvdiasm extracts information from standalone cubin files and presents them in human readable format. The output of nvdiasm includes CUDA assembly code for each kernel, listing of ELF data sections and other CUDA specific sections. Output style and options are controlled through nvdiasm command-line options. nvdiasm also does control flow analysis to annotate jump/branch targets and makes the output easier to read.

---

**Note:** nvdiasm requires complete relocation information to do control flow analysis. If this information is missing from the CUDA binary, either use the nvdiasm option `-ndf` to turn off control flow analysis, or use the `ptxas` and `nvlink` option `-preserve-relocs` to re-generate the cubin file.

---

For a list of CUDA assembly instruction set of each GPU architecture, see [Instruction Set Reference](#).

## 5.1. Usage

nvdiasm accepts a single input file each time it's run. The basic usage is as following:

```
nvdiasm [options] <input cubin file>
```

Here's a sample output of nvdiasm:

```
.headerflags    @"EF_CUDA_TEXMODE_UNIFIED EF_CUDA_64BIT_ADDRESS EF_CUDA_SM70
                EF_CUDA_VIRTUAL_SM(EF_CUDA_SM70)"
.elftype        @"ET_EXEC"

//----- .nv.info -----
.section        .nv.info,"",@"SHT_CUDA_INFO"
.align 4
.....

//----- .text._Z9acos_main10acosParams -----
.section        .text._Z9acos_main10acosParams,"ax",@progbits
.sectioninfo    @"SHI_REGISTERS=14"
.align 128
.global        _Z9acos_main10acosParams
.type          _Z9acos_main10acosParams,@function
.size          _Z9acos_main10acosParams, (.L_21 - _Z9acos_main10acosParams)
.other         _Z9acos_main10acosParams,@"ST0_CUDA_ENTRY STV_DEFAULT"
_Z9acos_main10acosParams:
```

(continues on next page)

(continued from previous page)

```
.text._Z9acos_main10acosParams:
/*0000*/      MOV R1, c[0x0][0x28] ;
/*0010*/      NOP;
/*0020*/      S2R R0, SR_CTAID.X ;
/*0030*/      S2R R3, SR_TID.X ;
/*0040*/      IMAD R0, R0, c[0x0][0x0], R3 ;
/*0050*/      ISETP.GE.AND P0, PT, R0, c[0x0][0x170], PT ;
/*0060*/      @P0 EXIT ;

.L_1:
/*0070*/      MOV R11, 0x4 ;
/*0080*/      IMAD.WIDE R2, R0, R11, c[0x0][0x160] ;
/*0090*/      LDG.E.SYS R2, [R2] ;
/*00a0*/      MOV R7, 0x3d53f941 ;
/*00b0*/      FADD.FTZ R4, |R2|.reuse, -RZ ;
/*00c0*/      FSETP.GT.FTZ.AND P0, PT, |R2|.reuse, 0.5699, PT ;
/*00d0*/      FSETP.GEU.FTZ.AND P1, PT, R2, RZ, PT ;
/*00e0*/      FADD.FTZ R5, -R4, 1 ;
/*00f0*/      IMAD.WIDE R2, R0, R11, c[0x0][0x168] ;
/*0100*/      FMUL.FTZ R5, R5, 0.5 ;
/*0110*/      @P0 MUFU.SQRT R4, R5 ;
/*0120*/      MOV R5, c[0x0][0x0] ;
/*0130*/      IMAD R0, R5, c[0x0][0xc], R0 ;
/*0140*/      FMUL.FTZ R6, R4, R4 ;
/*0150*/      FFMA.FTZ R7, R6, R7, 0.018166976049542427063 ;
/*0160*/      FFMA.FTZ R7, R6, R7, 0.046756859868764877319 ;
/*0170*/      FFMA.FTZ R7, R6, R7, 0.074846573173999786377 ;
/*0180*/      FFMA.FTZ R7, R6, R7, 0.16667014360427856445 ;
/*0190*/      FMUL.FTZ R7, R6, R7 ;
/*01a0*/      FFMA.FTZ R7, R4, R7, R4 ;
/*01b0*/      FADD.FTZ R9, R7, R7 ;
/*01c0*/      @!P0 FADD.FTZ R9, -R7, 1.5707963705062866211 ;
/*01d0*/      ISETP.GE.AND P0, PT, R0, c[0x0][0x170], PT ;
/*01e0*/      @!P1 FADD.FTZ R9, -R9, 3.1415927410125732422 ;
/*01f0*/      STG.E.SYS [R2], R9 ;
/*0200*/      @!P0 BRA `(.L_1) ;
/*0210*/      EXIT ;

.L_2:
/*0220*/      BRA `(.L_2);

.L_21:
```

To get the control flow graph of a kernel, use the following:

```
nvdiasm -cfg <input cubin file>
```

nvdiasm is capable of generating control flow of CUDA assembly in the format of DOT graph description language. The output of the control flow from nvdiasm can be directly imported to a DOT graph visualization tool such as [Graphviz](#).

Here's how you can generate a PNG image (cfg.png) of the control flow of the above cubin (a.cubin) with nvdiasm and Graphviz:

```
nvdiasm -cfg a.cubin | dot -ocfg.png -Tpng
```

Here's the generated graph:

To generate a PNG image (bbcfg.png) of the basic block control flow of the above cubin (a.cubin) with nvdiasm and Graphviz:

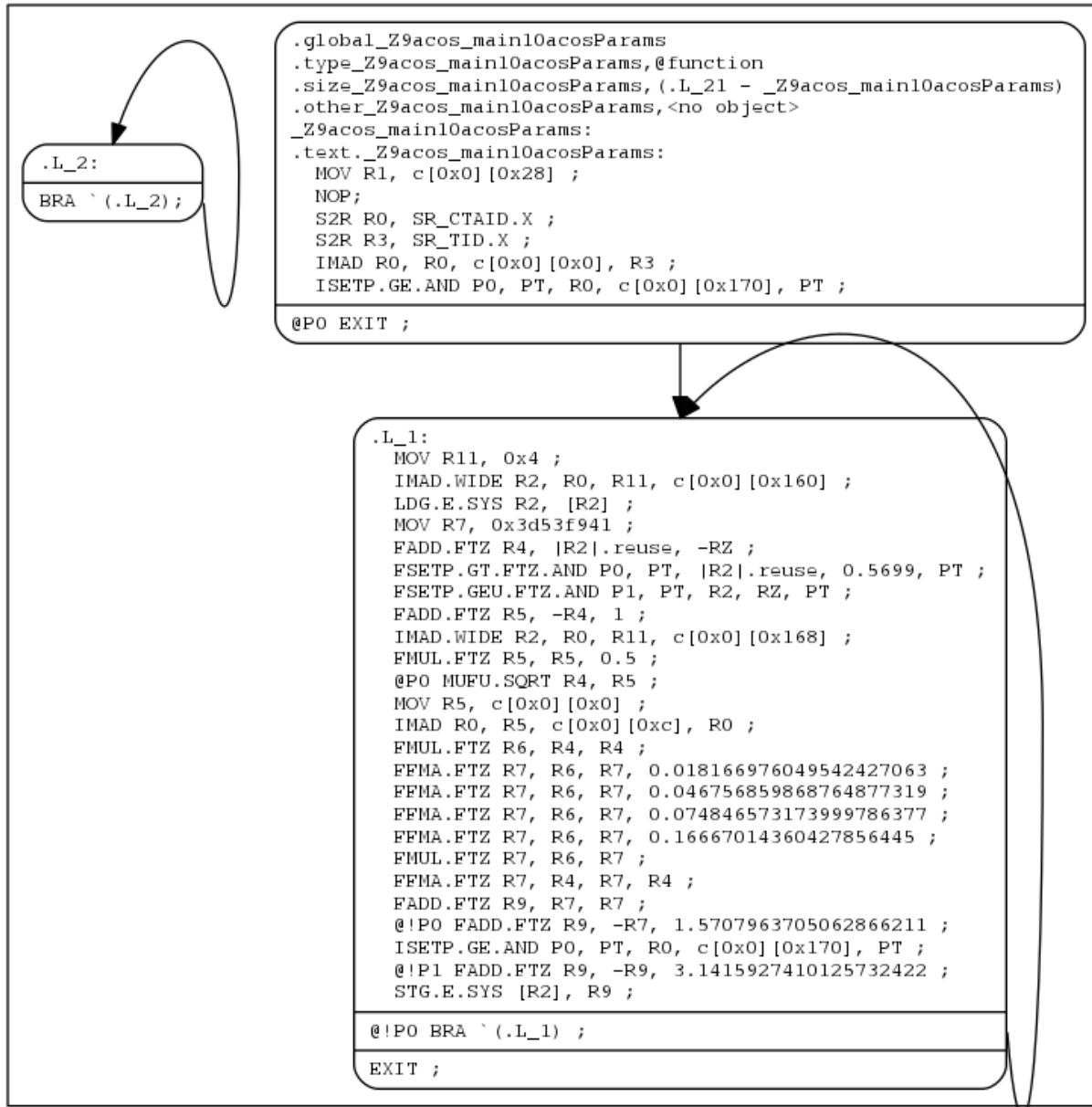


Fig. 1: Control Flow Graph

```
nvdiasm -bbcfcg a.cubin | dot -obbfcg.png -Tpng
```

Here's the generated graph:

nvdiasm is capable of showing the register (general and predicate) liveness range information. For each line of CUDA assembly, nvdiasm displays whether a given device register was assigned, accessed, live or re-assigned. It also shows the total number of registers used. This is useful if the user is interested in the life range of any particular register, or register usage in general.

Here's a sample output (output is pruned for brevity):

```

// +-----+-----+
// |          GPR          | PRED |
// |          |          |      |
// |          00000000011  |      |
// | # 012345678901 | # 01 |
// +-----+-----+
.global acos
.type acos,@function
.size acos,(.L_21 - acos)
.other acos,@"ST0_CUDA_ENTRY STV_DEFAULT"
acos:
.text.acos:
MOV R1, c[0x0][0x28] ;
NOP;
S2R R0, SR_CTAID.X ;
S2R R3, SR_TID.X ;
IMAD R0, R0, c[0x0][0x0], R3 ;
ISETP.GE.AND P0, PT, R0, c[0x0][0x170], PT ;
@P0 EXIT ;
.L_1:
MOV R11, 0x4 ;
IMAD.WIDE R2, R0, R11, c[0x0][0x160] ;
LDG.E.SYS R2, [R2] ;
MOV R7, 0x3d53f941 ;
FADD.FTZ R4, |R2|.reuse, -RZ ;
FSETP.GT.FTZ.AND P0, PT, |R2|.reuse, 0.5699, PT;
FSETP.GEU.FTZ.AND P1, PT, R2, RZ, PT ;
FADD.FTZ R5, -R4, 1 ;
IMAD.WIDE R2, R0, R11, c[0x0][0x168] ;
FMUL.FTZ R5, R5, 0.5 ;
@P0 MUFU.SQRT R4, R5 ;
MOV R5, c[0x0][0x0] ;
IMAD R0, R5, c[0x0][0xc], R0 ;
FMUL.FTZ R6, R4, R4 ;
FFMA.FTZ R7, R6, R7, 0.018166976049542427063 ;
FFMA.FTZ R7, R6, R7, 0.046756859868764877319 ;
FFMA.FTZ R7, R6, R7, 0.074846573173999786377 ;
FFMA.FTZ R7, R6, R7, 0.16667014360427856445 ;
FMUL.FTZ R7, R6, R7 ;
FFMA.FTZ R7, R4, R7, R4 ;
FADD.FTZ R9, R7, R7 ;
@!P0 FADD.FTZ R9, -R7, 1.5707963705062866211 ;
ISETP.GE.AND P0, PT, R0, c[0x0][0x170], PT ;
@!P1 FADD.FTZ R9, -R9, 3.1415927410125732422 ;
STG.E.SYS [R2], R9 ;
@!P0 BRA `(.L_1) ;

```

(continues on next page)

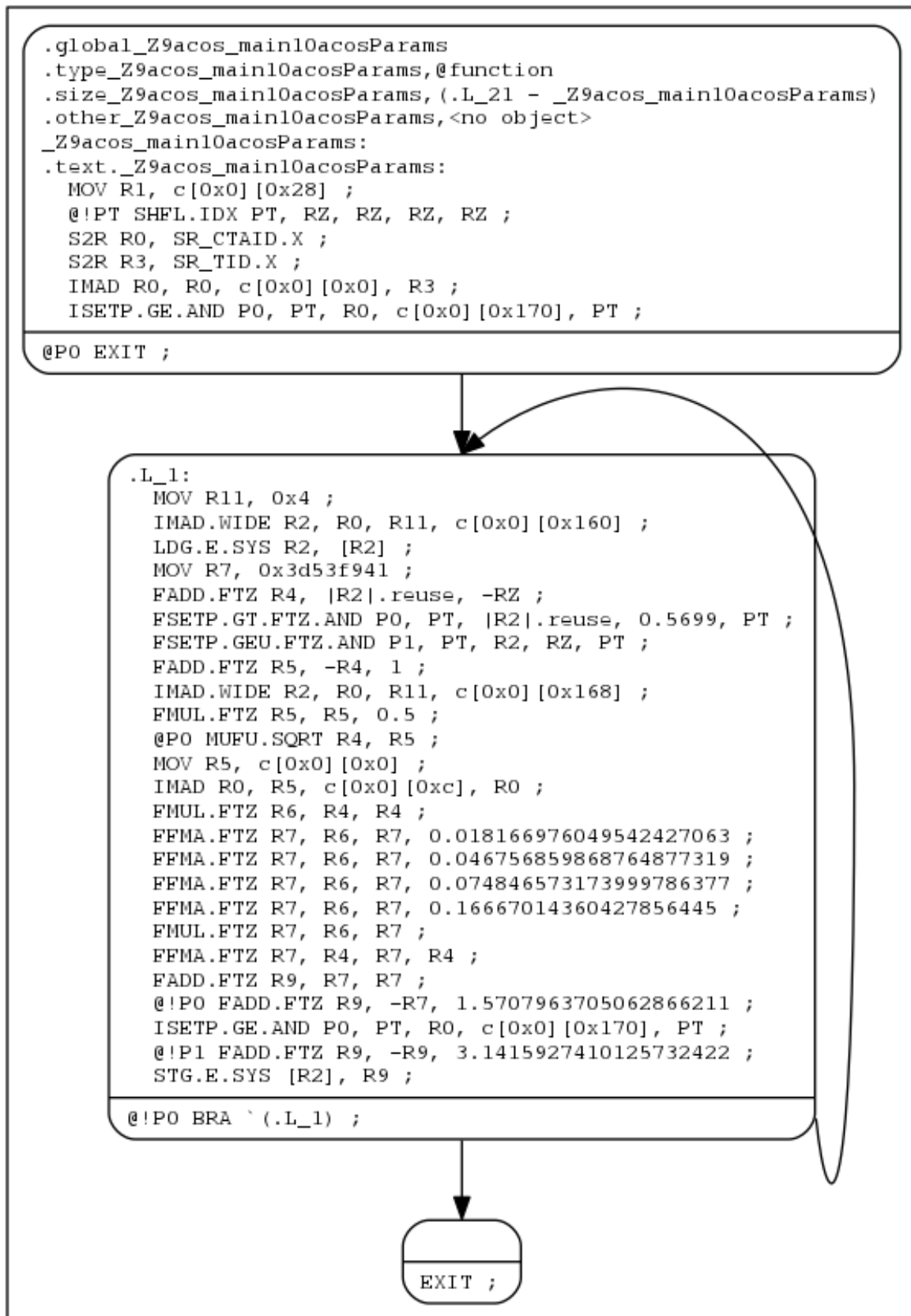


Fig. 2: Basic Block Control Flow Graph

(continued from previous page)

```

EXIT ; // | 1 : | |
.L_2: // +.....+.....+
BRA `(.L_2); // | | |
.L_21: // +-----+-----+
// Legend:
// ^ : Register
↪assignment // v : Register usage
// x : Register usage
↪and reassignment // : : Register in use
// <space> : Register not
↪in use // # : Number of
↪occupied registers

```

nvdiasm is capable of showing line number information of the CUDA source file which can be useful for debugging.

To get the line-info of a kernel, use the following:

```
nvdiasm -g <input cubin file>
```

Here's a sample output of a kernel using nvdiasm -g command:

```

//----- .text._Z6kernali -----
.section .text._Z6kernali,"ax",@progbits
.sectioninfo @"SHI_REGISTERS=24"
.align 128
.global _Z6kernali
.type _Z6kernali,@function
.size _Z6kernali,(.L_4 - _Z6kernali)
.other _Z6kernali,@"STO_CUDA_ENTRY STV_DEFAULT"
_Z6kernali:
.text._Z6kernali:
/*000*/ MOV R1, c[0x0][0x28] ;
/*001*/ NOP;
///## File "/home/user/cuda/sample/sample.cu", line 25
/*002*/ MOV R0, 0x160 ;
/*003*/ LDC R0, c[0x0][R0] ;
/*004*/ MOV R0, R0 ;
/*005*/ MOV R2, R0 ;
///## File "/home/user/cuda/sample/sample.cu", line 26
/*006*/ MOV R4, R2 ;
/*007*/ MOV R20, 32@lo((_Z6kernali + .L_1@srel)) ;
/*008*/ MOV R21, 32@hi((_Z6kernali + .L_1@srel)) ;
/*009*/ CALL.ABS.NOINC `(_Z3fooi) ;
.L_1:
/*00a*/ MOV R0, R4 ;
/*00b*/ MOV R4, R2 ;
/*00c*/ MOV R2, R0 ;
/*00d*/ MOV R20, 32@lo((_Z6kernali + .L_2@srel)) ;
/*00e*/ MOV R21, 32@hi((_Z6kernali + .L_2@srel)) ;
/*00f*/ CALL.ABS.NOINC `(_Z3bari) ;
.L_2:
/*010*/ MOV R4, R4 ;
/*011*/ IADD3 R4, R2, R4, RZ ;

```

(continues on next page)



(continued from previous page)

```

/*0120*/          MOV R2, 32@lo(arr) ;
/*0130*/          MOV R3, 32@hi(arr) ;
/*0140*/          MOV R2, R2 ;
/*0150*/          MOV R3, R3 ;
/*0160*/          ST.E.SYS [R2], R4 ;
///## File "/home/user/cuda/sample/sample.cu", line 27
/*0170*/          ERRBAR ;
/*0180*/          EXIT ;
.L_3:
/*0190*/          BRA `(.L_3);
.L_4:

```

nvdiasm is capable of showing line number information with additional function inlining info (if any). In absence of any function inlining the output is same as the one with `nvdiasm -g` command.

Here's a sample output of a kernel using `nvdiasm -gi` command:

```

//----- .text._Z6kernali -----
.section .text._Z6kernali,"ax",@progbits
.sectioninfo @"SHI_REGISTERS=16"
.align 128
.global _Z6kernali
.type _Z6kernali,@function
.size _Z6kernali,(.L_18 - _Z6kernali)
.other _Z6kernali,@"STO_CUDA_ENTRY STV_DEFAULT"
_Z6kernali:
.text._Z6kernali:
/*0000*/          IMAD.MOV.U32 R1, RZ, RZ, c[0x0][0x28] ;
///## File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
///## File "/home/user/cuda/inline.cu", line 23
/*0010*/          UMOV UR4, 32@lo(arr) ;
/*0020*/          UMOV UR5, 32@hi(arr) ;
/*0030*/          IMAD.U32 R2, RZ, RZ, UR4 ;
/*0040*/          MOV R3, UR5 ;
/*0050*/          ULDC.64 UR4, c[0x0][0x118] ;
///## File "/home/user/cuda/inline.cu", line 10 inlined at "/home/user/cuda/inline.
↪cu", line 17
///## File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
///## File "/home/user/cuda/inline.cu", line 23
/*0060*/          LDG.E R4, [R2.64] ;
/*0070*/          LDG.E R5, [R2.64+0x4] ;
///## File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
///## File "/home/user/cuda/inline.cu", line 23
/*0080*/          LDG.E R0, [R2.64+0x8] ;
///## File "/home/user/cuda/inline.cu", line 23
/*0090*/          UMOV UR6, 32@lo(ans) ;
/*00a0*/          UMOV UR7, 32@hi(ans) ;
///## File "/home/user/cuda/inline.cu", line 10 inlined at "/home/user/cuda/inline.
↪cu", line 17
///## File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
///## File "/home/user/cuda/inline.cu", line 23
/*00b0*/          IADD3 R7, R4, c[0x0][0x160], RZ ;
///## File "/home/user/cuda/inline.cu", line 23

```

(continues on next page)

(continued from previous page)

```

/*00c0*/          IMAD.U32 R4, RZ, RZ, UR6 ;
  /// File "/home/user/cuda/inline.cu", line 10 inlined at "/home/user/cuda/inline.
↪cu", line 17
  /// File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
  /// File "/home/user/cuda/inline.cu", line 23
  /*00d0*/          IADD3 R9, R5, c[0x0][0x160], RZ ;
  /// File "/home/user/cuda/inline.cu", line 23
  /*00e0*/          MOV R5, UR7 ;
  /// File "/home/user/cuda/inline.cu", line 10 inlined at "/home/user/cuda/inline.
↪cu", line 17
  /// File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
  /// File "/home/user/cuda/inline.cu", line 23
  /*00f0*/          IADD3 R11, R0.reuse, c[0x0][0x160], RZ ;
  /// File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
  /// File "/home/user/cuda/inline.cu", line 23
  /*0100*/          IMAD.IADD R13, R0, 0x1, R7 ;
  /// File "/home/user/cuda/inline.cu", line 10 inlined at "/home/user/cuda/inline.
↪cu", line 17
  /// File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
  /// File "/home/user/cuda/inline.cu", line 23
  /*0110*/          STG.E [R2.64+0x4], R9 ;
  /*0120*/          STG.E [R2.64], R7 ;
  /*0130*/          STG.E [R2.64+0x8], R11 ;
  /// File "/home/user/cuda/inline.cu", line 23
  /*0140*/          STG.E [R4.64], R13 ;
  /// File "/home/user/cuda/inline.cu", line 24
  /*0150*/          EXIT ;
.L_3:
  /*0160*/          BRA (.L_3);
.L_18:

```

## 5.2. Command-line Options

Table 3 contains the supported command-line options of `nvdiasm`, along with a description of what each option does. Each option has a long name and a short name, which can be used interchangeably.

Table 1: Table 3. nvdiasm Command-line Options

| Option (long)  | Option (short)                          | Description  |
|--|---|--|
| <code>--base-address &lt;value&gt;</code>                  | <code>-base</code>                      | Specify the logical base address of the image to disassemble. This option is only valid when disassembling a raw instruction binary (see option <code>--binary</code> ), and is ignored when disassembling an Elf file. Default value: 0.  |
| <code>--binary &lt;SMxy&gt;</code>                         | <code>-b</code>                         | When this option is specified, the input file is assumed to contain a raw instruction binary, that is, a sequence of binary instruction encodings as they occur in instruction memory. The value of this option must be the asserted architecture of the raw binary. Allowed values for this option: SM50, SM52, SM53, SM60, SM61, SM62, SM70, SM72, SM75, SM80, SM86, SM87, SM89, SM90, SM90a.  |
| <code>--cuda-function &lt;symbol index&gt;, ...</code>     | <code>-restrict</code>                  | Restrict the output to the CUDA functions represented by symbols with the given indices. The CUDA function for a given symbol is the enclosing section. This only restricts executable sections; all other sections will still be printed.   |
| <code>--help</code>  | <code>-h</code>                         | Print this help information on this tool.  |
| <code>--life-range &lt;mode&gt;</code>                     | <code>-lmode</code>                     | This option implies option <code>--print-life-ranges</code> , and determines how register live range info should be printed. <code>count</code> : Not at all, leaving only the # column (number of live registers); <code>wide</code> : Columns spaced out for readability (default); <code>narrow</code> : A one-character column for each register, economizing on table width. Allowed values for this option: <code>count</code> , <code>narrow</code> , <code>wide</code> . |
| <code>--no-dataflow</code>                                 | <code>-lowdf</code>                     | Disable dataflow analyzer after disassembly. Dataflow analysis is normally enabled to perform branch stack analysis and annotate all instructions that jump via the GPU branch stack with inferred branch target labels. However, it may occasionally fail when certain restrictions on the input nvelv/cubin are not met.   |
| <code>--no-vliw</code>                                     | <code>-novliw</code>                    | Conventional mode; disassemble paired instructions in normal syntax, instead of VLIW syntax.   |
| <code>--options-file &lt;file&gt;, ...</code>              | <code>-ifile</code>                     | Include command line options from specified file.  |
| <code>--output-control-flow-graph</code>                   | <code>-cfgraph</code>                   | When specified, output the control flow graph, where each node is a hyperblock, in a format consumable by graphviz tools (such as dot).  |
| <code>--output-control-flow-graph-with-basic-blocks</code> | <code>-cfgraph-with-basic-blocks</code> | When specified, output the control flow graph, where each node is a basicblock, in a format consumable by graphviz tools (such as dot).  |
| <code>--print-codec</code>                                 |   | Only print code sections.  |
| <code>--print-instruction-offsets</code>                   | <code>-ifsoff</code>                    | When specified, print instruction offsets in the control flow graph. This should be used along with the option <code>-output-control-flow-graph</code> or <code>-output-control-flow-graph-with-basic-blocks</code> .  |
| <code>--print-instruction-encoding</code>                  | <code>-ie</code>                        | When specified, print the encoding bytes after each disassembled operation.  |
| <code>--print-life-ranges</code>                           | <code>-lrange</code>                    | Print register life range information in a trailing column in the produced disassembly.  |
| <code>--print-lineinfo</code>                              | <code>-lineinfo</code>                  | Annotate disassembly with source line information obtained from <code>.debug_line</code> section, if present.  |
| <code>--print-lineinfo</code>                              | <code>-lineinfo</code>                  | Annotate disassembly with source line information obtained from <code>.debug_line</code> section along with function inlining info, if present.  |

5.2. Command-line Options

`--print-lineinfo` Annotate disassembly with source line information obtained from `.nv_debug_line_sass` section, if present.

`--print-raw-raw` Print the disassembly without any attempt to beautify it.



---

# Chapter 6. Instruction Set Reference

This is an instruction set reference for NVIDIA® GPU architectures Kepler, Maxwell, Pascal, Volta, Turing and Ampere.

## 6.1. Maxwell and Pascal Instruction Set

The Maxwell (Compute Capability 5.x) and the Pascal (Compute Capability 6.x) architectures have the following instruction set format:

```
(instruction) (destination) (source1), (source2) ...
```

Valid destination and source locations include:

- ▶ RX for registers
- ▶ SRX for special system-controlled registers
- ▶ PX for condition registers
- ▶ c[X][Y] for constant memory

Table 4 lists valid instructions for the Maxwell and Pascal GPUs.

Table 1: Table 4. Maxwell and Pascal Instruction Set

| Opcode                             | Description                            |
|------------------------------------|--|
| <b>Floating Point Instructions</b> |  |
| FADD                               | FP32 Add                               |
| FCHK                               | Single Precision FP Divide Range Check |
| FCMP                               | FP32 Compare to Zero and Select Source |
| FFMA                               | FP32 Fused Multiply and Add            |
| FMNMX                              | FP32 Minimum/Maximum                   |
| FMUL                               | FP32 Multiply                          |
| FSET                               | FP32 Compare And Set                   |
| FSETP                              | FP32 Compare And Set Predicate         |

continues on next page

Table 1 – continued from previous page

| <b>Opcode</b>               | <b>Description</b>                        |
|-----------------------------|---|
| FSWZADD                     | FP32 Add used for FSWZ emulation          |
| MUFU                        | Multi Function Operation                  |
| RRO                         | Range Reduction Operator FP               |
| DADD                        | FP64 Add                                  |
| DFMA                        | FP64 Fused Mutiply Add                    |
| DMNMX                       | FP64 Minimum/Maximum                      |
| DMUL                        | FP64 Multiply                             |
| DSET                        | FP64 Compare And Set                      |
| DSETP                       | FP64 Compare And Set Predicate            |
| HADD2                       | FP16 Add                                  |
| HFMA2                       | FP16 Fused Mutiply Add                    |
| HMUL2                       | FP16 Multiply                             |
| HSET2                       | FP16 Compare And Set                      |
| HSETP2                      | FP16 Compare And Set Predicate            |
| <b>Integer Instructions</b> |   |
| BFE                         | Bit Field Extract                         |
| BFI                         | Bit Field Insert                          |
| FLO                         | Find Leading One                          |
| IADD                        | Integer Addition                          |
| IADD3                       | 3-input Integer Addition                  |
| ICMP                        | Integer Compare to Zero and Select Source |
| IMAD                        | Integer Multiply And Add                  |
| IMADSP                      | Extracted Integer Multiply And Add.       |
| IMNMX                       | Integer Minimum/Maximum                   |
| IMUL                        | Integer Multiply                          |
| ISCADD                      | Scaled Integer Addition                   |
| ISSET                       | Integer Compare And Set                   |
| ISSETP                      | Integer Compare And Set Predicate         |
| LEA                         | Compute Effective Address                 |
| LOP                         | Logic Operation                           |
| LOP3                        | 3-input Logic Operation                   |
| POPC                        | Population count                          |

continues on next page

Table 1 – continued from previous page

| <b>Opcode</b>                          | <b>Description</b>                                      |
|--|---|
| SHF                                    | Funnel Shift  |
| SHL                                    | Shift Left  |
| SHR                                    | Shift Right   |
| XMAD                                   | Integer Short Multiply Add                              |
| <b>Conversion Instructions</b>         |   |
| F2F                                    | Floating Point To Floating Point Conversion             |
| F2I                                    | Floating Point To Integer Conversion                    |
| I2F                                    | Integer To Floating Point Conversion                    |
| I2I                                    | Integer To Integer Conversion                           |
| <b>Movement Instructions</b>           |   |
| MOV                                    | Move  |
| PRMT                                   | Permute Register Pair                                   |
| SEL                                    | Select Source with Predicate                            |
| SHFL                                   | Warp Wide Register Shuffle                              |
| <b>Predicate/CC Instructions</b>       |   |
| CSET                                   | Test Condition Code And Set                             |
| CSETP                                  | Test Condition Code and Set Predicate                   |
| PSET                                   | Combine Predicates and Set                              |
| PSETP                                  | Combine Predicates and Set Predicate                    |
| P2R                                    | Move Predicate Register To Register                     |
| R2P                                    | Move Register To Predicate/CC Register                  |
| <b>Texture Instructions</b>            |   |
| TEX                                    | Texture Fetch   |
| TLD                                    | Texture Load  |
| TLD4                                   | Texture Load 4  |
| TXQ                                    | Texture Query   |
| TEXS                                   | Texture Fetch with scalar/non-vec4 source/destinations  |
| TLD4S                                  | Texture Load 4 with scalar/non-vec4 source/destinations |
| TLDS                                   | Texture Load with scalar/non-vec4 source/destinations   |
| <b>Compute Load/Store Instructions</b> |   |
| LD                                     | Load from generic Memory                                |
| LDC                                    | Load Constant   |

continues on next page

Table 1 – continued from previous page

| <b>Opcode</b>                      | <b>Description</b>                        |
|------------------------------------|---|
| LDG                                | Load from Global Memory                   |
| LDL                                | Load within Local Memory Window           |
| LDS                                | Local within Shared Memory Window         |
| ST                                 | Store to generic Memory                   |
| STG                                | Store to global Memory                    |
| STL                                | Store to Local Memory                     |
| STS                                | Store to Shared Memory                    |
| ATOM                               | Atomic Operation on generic Memory        |
| ATOMS                              | Atomic Operation on Shared Memory         |
| RED                                | Reduction Operation on generic Memory     |
| CCTL                               | Cache Control                             |
| CCTLL                              | Cache Control                             |
| MEMBAR                             | Memory Barrier                            |
| CCLT                               | Texture Cache Control                     |
| <b>Surface Memory Instructions</b> |   |
| SUATOM                             | Atomic Op on Surface Memory               |
| SULD                               | Surface Load                              |
| SURED                              | Reduction Op on Surface Memory            |
| SUST                               | Surface Store                             |
| <b>Control Instructions</b>        |   |
| BRA                                | Relative Branch                           |
| BRX                                | Relative Branch Indirect                  |
| JMP                                | Absolute Jump                             |
| JMX                                | Absolute Jump Indirect                    |
| SSY                                | Set Synchronization Point                 |
| SYNC                               | Converge threads after conditional branch |
| CAL                                | Relative Call                             |
| JCAL                               | Absolute Call                             |
| PRET                               | Pre-Return From Subroutine                |
| RET                                | Return From Subroutine                    |
| BRK                                | Break                                     |
| PBK                                | Pre-Break                                 |

continues on next page



Table 1 – continued from previous page

| Opcode                            | Description                       |
|-----------------------------------|-----------------------------------|
| CONT                              | Continue                          |
| PCNT                              | Pre-continue                      |
| EXIT                              | Exit Program                      |
| PEXIT                             | Pre-Exit                          |
| BPT                               | BreakPoint/Trap                   |
| <b>Miscellaneous Instructions</b> |                                   |
| NOP                               | No Operation                      |
| CS2R                              | Move Special Register to Register |
| S2R                               | Move Special Register to Register |
| B2R                               | Move Barrier To Register          |
| BAR                               | Barrier Synchronization           |
| R2B                               | Move Register to Barrier          |
| VOTE                              | Vote Across SIMD Thread Group     |

## 6.2. Volta Instruction Set

The Volta architecture (Compute Capability 7.x) has the following instruction set format:

```
(instruction) (destination) (source1), (source2) ...
```

Valid destination and source locations include:

- ▶ RX for registers
- ▶ SRX for special system-controlled registers
- ▶ PX for predicate registers
- ▶ c[X][Y] for constant memory

Table 5 lists valid instructions for the Volta GPUs.

Table 2: Table 5. Volta Instruction Set

| Opcode                             | Description                |
|------------------------------------|----------------------------|
| <b>Floating Point Instructions</b> |                            |
| FADD                               | FP32 Add                   |
| FADD32I                            | FP32 Add                   |
| FCHK                               | Floating-point Range Check |

continues on next page

Table 2 – continued from previous page

| <b>Opcode</b>               | <b>Description</b>                 |
|-----------------------------|------------------------------------|
| FFMA32I                     | FP32 Fused Multiply and Add        |
| FFMA                        | FP32 Fused Multiply and Add        |
| FMNMX                       | FP32 Minimum/Maximum               |
| FMUL                        | FP32 Multiply                      |
| FMUL32I                     | FP32 Multiply                      |
| FSEL                        | Floating Point Select              |
| FSET                        | FP32 Compare And Set               |
| FSETP                       | FP32 Compare And Set Predicate     |
| FSWZADD                     | FP32 Swizzle Add                   |
| MUFU                        | FP32 Multi Function Operation      |
| HADD2                       | FP16 Add                           |
| HADD2_32I                   | FP16 Add                           |
| HFMA2                       | FP16 Fused Mutiply Add             |
| HFMA2_32I                   | FP16 Fused Mutiply Add             |
| HMMA                        | Matrix Multiply and Accumulate     |
| HMUL2                       | FP16 Multiply                      |
| HMUL2_32I                   | FP16 Multiply                      |
| HSET2                       | FP16 Compare And Set               |
| HSETP2                      | FP16 Compare And Set Predicate     |
| DADD                        | FP64 Add                           |
| DFMA                        | FP64 Fused Mutiply Add             |
| DMUL                        | FP64 Multiply                      |
| DSETP                       | FP64 Compare And Set Predicate     |
| <b>Integer Instructions</b> |                                    |
| BMSK                        | Bitfield Mask                      |
| BREV                        | Bit Reverse                        |
| FLO                         | Find Leading One                   |
| IABS                        | Integer Absolute Value             |
| IADD                        | Integer Addition                   |
| IADD3                       | 3-input Integer Addition           |
| IADD32I                     | Integer Addition                   |
| IDP                         | Integer Dot Product and Accumulate |

continues on next page

Table 2 – continued from previous page

| <b>Opcode</b>                  | <b>Description</b>                          |
|--------------------------------|---|
| IDP4A                          | Integer Dot Product and Accumulate          |
| IMAD                           | Integer Multiply And Add                    |
| IMMA                           | Integer Matrix Multiply and Accumulate      |
| IMNMX                          | Integer Minimum/Maximum                     |
| IMUL                           | Integer Multiply                            |
| IMUL32I                        | Integer Multiply                            |
| ISCADD                         | Scaled Integer Addition                     |
| ISCADD32I                      | Scaled Integer Addition                     |
| ISETP                          | Integer Compare And Set Predicate           |
| LEA                            | LOAD Effective Address                      |
| LOP                            | Logic Operation                             |
| LOP3                           | Logic Operation                             |
| LOP32I                         | Logic Operation                             |
| POPC                           | Population count                            |
| SHF                            | Funnel Shift                                |
| SHL                            | Shift Left                                  |
| SHR                            | Shift Right                                 |
| VABSDIFF                       | Absolute Difference                         |
| VABSDIFF4                      | Absolute Difference                         |
| <b>Conversion Instructions</b> |   |
| F2F                            | Floating Point To Floating Point Conversion |
| F2I                            | Floating Point To Integer Conversion        |
| I2F                            | Integer To Floating Point Conversion        |
| I2I                            | Integer To Integer Conversion               |
| I2IP                           | Integer To Integer Conversion and Packing   |
| FRND                           | Round To Integer                            |
| <b>Movement Instructions</b>   |   |
| MOV                            | Move  |
| MOV32I                         | Move  |
| PRMT                           | Permute Register Pair                       |
| SEL                            | Select Source with Predicate                |
| SGXT                           | Sign Extend                                 |

continues on next page

Table 2 – continued from previous page

| <b>Opcode</b>                  | <b>Description</b>                        |
|--------------------------------|---|
| SHFL                           | Warp Wide Register Shuffle                |
| <b>Predicate Instructions</b>  |   |
| PLOP3                          | Predicate Logic Operation                 |
| PSETP                          | Combine Predicates and Set Predicate      |
| P2R                            | Move Predicate Register To Register       |
| R2P                            | Move Register To Predicate Register       |
| <b>Load/Store Instructions</b> |   |
| LD                             | Load from generic Memory                  |
| LDC                            | Load Constant                             |
| LDG                            | Load from Global Memory                   |
| LDL                            | Load within Local Memory Window           |
| LDS                            | Load within Shared Memory Window          |
| ST                             | Store to Generic Memory                   |
| STG                            | Store to Global Memory                    |
| STL                            | Store to Local Memory                     |
| STS                            | Store to Shared Memory                    |
| MATCH                          | Match Register Values Across Thread Group |
| QSPC                           | Query Space                               |
| ATOM                           | Atomic Operation on Generic Memory        |
| ATOMS                          | Atomic Operation on Shared Memory         |
| ATOMG                          | Atomic Operation on Global Memory         |
| RED                            | Reduction Operation on Generic Memory     |
| CCTL                           | Cache Control                             |
| CCTLL                          | Cache Control                             |
| ERRBAR                         | Error Barrier                             |
| MEMBAR                         | Memory Barrier                            |
| CCTLT                          | Texture Cache Control                     |
| <b>Texture Instructions</b>    |   |
| TEX                            | Texture Fetch                             |
| TLD                            | Texture Load                              |
| TLD4                           | Texture Load 4                            |
| TMML                           | Texture MipMap Level                      |

continues on next page

Table 2 – continued from previous page

| <b>Opcode</b>                     | <b>Description</b>                             |
|-----------------------------------|--|
| TXD                               | Texture Fetch With Derivatives                 |
| TXQ                               | Texture Query                                  |
| <b>Surface Instructions</b>       |  |
| SUATOM                            | Atomic Op on Surface Memory                    |
| SULD                              | Surface Load                                   |
| SURED                             | Reduction Op on Surface Memory                 |
| SUST                              | Surface Store                                  |
| <b>Control Instructions</b>       |  |
| BMOV                              | Move Convergence Barrier State                 |
| BPT                               | BreakPoint/Trap                                |
| BRA                               | Relative Branch                                |
| BREAK                             | Break out of the Specified Convergence Barrier |
| BRX                               | Relative Branch Indirect                       |
| BSSY                              | Barrier Set Convergence Synchronization Point  |
| BSYNC                             | Synchronize Threads on a Convergence Barrier   |
| CALL                              | Call Function                                  |
| EXIT                              | Exit Program                                   |
| JMP                               | Absolute Jump                                  |
| JMX                               | Absolute Jump Indirect                         |
| KILL                              | Kill Thread                                    |
| NANOSLEEP                         | Suspend Execution                              |
| RET                               | Return From Subroutine                         |
| RPCMOV                            | PC Register Move                               |
| RTT                               | Return From Trap                               |
| WARPSYNC                          | Synchronize Threads in Warp                    |
| YIELD                             | Yield Control                                  |
| <b>Miscellaneous Instructions</b> |  |
| B2R                               | Move Barrier To Register                       |
| BAR                               | Barrier Synchronization                        |
| CS2R                              | Move Special Register to Register              |
| DEPBAR                            | Dependency Barrier                             |
| GETLMEMBASE                       | Get Local Memory Base Address                  |

continues on next page

Table 2 – continued from previous page

| Opcode      | Description                       |
|-------------|-----------------------------------|
| LEPC        | Load Effective PC                 |
| NOP         | No Operation                      |
| PMTRIG      | Performance Monitor Trigger       |
| R2B         | Move Register to Barrier          |
| S2R         | Move Special Register to Register |
| SETCTAID    | Set CTA ID                        |
| SETLMEMBASE | Set Local Memory Base Address     |
| VOTE        | Vote Across SIMD Thread Group     |

## 6.3. Turing Instruction Set

The Turing architecture (Compute Capability 7.3 and 7.5) have the following instruction set format:

```
(instruction) (destination) (source1), (source2) ...
```

Valid destination and source locations include:

- ▶ RX for registers
- ▶ URX for uniform registers
- ▶ SRX for special system-controlled registers
- ▶ PX for predicate registers
- ▶ c[X][Y] for constant memory

Table 6 lists valid instructions for the Turing GPUs.

Table 3: Table 6. Turing Instruction Set

| Opcode                             | Description                 |
|------------------------------------|-----------------------------|
| <b>Floating Point Instructions</b> |                             |
| FADD                               | FP32 Add                    |
| FADD32I                            | FP32 Add                    |
| FCHK                               | Floating-point Range Check  |
| FFMA32I                            | FP32 Fused Multiply and Add |
| FFMA                               | FP32 Fused Multiply and Add |
| FMNMX                              | FP32 Minimum/Maximum        |
| FMUL                               | FP32 Multiply               |

continues on next page

Table 3 – continued from previous page

| <b>Opcode</b>               | <b>Description</b>                     |
|-----------------------------|--|
| FMUL32I                     | FP32 Multiply                          |
| FSEL                        | Floating Point Select                  |
| FSET                        | FP32 Compare And Set                   |
| FSETP                       | FP32 Compare And Set Predicate         |
| FSWZADD                     | FP32 Swizzle Add                       |
| MUFU                        | FP32 Multi Function Operation          |
| HADD2                       | FP16 Add                               |
| HADD2_32I                   | FP16 Add                               |
| HFMA2                       | FP16 Fused Mutiply Add                 |
| HFMA2_32I                   | FP16 Fused Mutiply Add                 |
| HMMA                        | Matrix Multiply and Accumulate         |
| HMUL2                       | FP16 Multiply                          |
| HMUL2_32I                   | FP16 Multiply                          |
| HSET2                       | FP16 Compare And Set                   |
| HSETP2                      | FP16 Compare And Set Predicate         |
| DADD                        | FP64 Add                               |
| DFMA                        | FP64 Fused Mutiply Add                 |
| DMUL                        | FP64 Multiply                          |
| DSETP                       | FP64 Compare And Set Predicate         |
| <b>Integer Instructions</b> |  |
| BMMA                        | Bit Matrix Multiply and Accumulate     |
| BMSK                        | Bitfield Mask                          |
| BREV                        | Bit Reverse                            |
| FLO                         | Find Leading One                       |
| IABS                        | Integer Absolute Value                 |
| IADD                        | Integer Addition                       |
| IADD3                       | 3-input Integer Addition               |
| IADD32I                     | Integer Addition                       |
| IDP                         | Integer Dot Product and Accumulate     |
| IDP4A                       | Integer Dot Product and Accumulate     |
| IMAD                        | Integer Multiply And Add               |
| IMMA                        | Integer Matrix Multiply and Accumulate |

continues on next page

Table 3 – continued from previous page

| <b>Opcode</b>                  | <b>Description</b>                          |
|--------------------------------|---|
| IMNMX                          | Integer Minimum/Maximum                     |
| IMUL                           | Integer Multiply                            |
| IMUL32I                        | Integer Multiply                            |
| ISCADD                         | Scaled Integer Addition                     |
| ISCADD32I                      | Scaled Integer Addition                     |
| ISETP                          | Integer Compare And Set Predicate           |
| LEA                            | LOAD Effective Address                      |
| LOP                            | Logic Operation                             |
| LOP3                           | Logic Operation                             |
| LOP32I                         | Logic Operation                             |
| POPC                           | Population count                            |
| SHF                            | Funnel Shift                                |
| SHL                            | Shift Left                                  |
| SHR                            | Shift Right                                 |
| VABSDIFF                       | Absolute Difference                         |
| VABSDIFF4                      | Absolute Difference                         |
| <b>Conversion Instructions</b> |   |
| F2F                            | Floating Point To Floating Point Conversion |
| F2I                            | Floating Point To Integer Conversion        |
| I2F                            | Integer To Floating Point Conversion        |
| I2I                            | Integer To Integer Conversion               |
| I2IP                           | Integer To Integer Conversion and Packing   |
| FRND                           | Round To Integer                            |
| <b>Movement Instructions</b>   |   |
| MOV                            | Move  |
| MOV32I                         | Move  |
| MOVM                           | Move Matrix with Transposition or Expansion |
| PRMT                           | Permute Register Pair                       |
| SEL                            | Select Source with Predicate                |
| SGXT                           | Sign Extend                                 |
| SHFL                           | Warp Wide Register Shuffle                  |
| <b>Predicate Instructions</b>  |   |

continues on next page



Table 3 – continued from previous page

| <b>Opcode</b>                        | <b>Description</b>   |
|--------------------------------------|--|
| PLOP3                                | Predicate Logic Operation                                  |
| PSETP                                | Combine Predicates and Set Predicate                       |
| P2R                                  | Move Predicate Register To Register                        |
| R2P                                  | Move Register To Predicate Register                        |
| <b>Load/Store Instructions</b>       |  |
| LD                                   | Load from generic Memory                                   |
| LDC                                  | Load Constant  |
| LDG                                  | Load from Global Memory                                    |
| LDL                                  | Load within Local Memory Window                            |
| LDS                                  | Load within Shared Memory Window                           |
| LDSM                                 | Load Matrix from Shared Memory with Element Size Expansion |
| ST                                   | Store to Generic Memory                                    |
| STG                                  | Store to Global Memory                                     |
| STL                                  | Store to Local Memory                                      |
| STS                                  | Store to Shared Memory                                     |
| MATCH                                | Match Register Values Across Thread Group                  |
| QSPC                                 | Query Space  |
| ATOM                                 | Atomic Operation on Generic Memory                         |
| ATOMS                                | Atomic Operation on Shared Memory                          |
| ATOMG                                | Atomic Operation on Global Memory                          |
| RED                                  | Reduction Operation on Generic Memory                      |
| CCTL                                 | Cache Control  |
| CCTLL                                | Cache Control  |
| ERRBAR                               | Error Barrier  |
| MEMBAR                               | Memory Barrier   |
| CCTLT                                | Texture Cache Control                                      |
| <b>Uniform Datapath Instructions</b> |  |
| R2UR                                 | Move from Vector Register to a Uniform Register            |
| S2UR                                 | Move Special Register to Uniform Register                  |
| UBMSK                                | Uniform Bitfield Mask                                      |
| UBREV                                | Uniform Bit Reverse  |
| UCLEA                                | Load Effective Address for a Constant                      |

continues on next page

Table 3 – continued from previous page

| <b>Opcode</b>               | <b>Description</b>  |
|-----------------------------|---|
| UFLO                        | Uniform Find Leading One  |
| UIADD3                      | Uniform Integer Addition  |
| UIADD3.64                   | Uniform Integer Addition  |
| UIMAD                       | Uniform Integer Multiplication                                      |
| UISETP                      | Integer Compare and Set Uniform Predicate                           |
| ULDC                        | Load from Constant Memory into a Uniform Register                   |
| ULEA                        | Uniform Load Effective Address                                      |
| ULOP                        | Logic Operation   |
| ULOP3                       | Logic Operation   |
| ULOP32I                     | Logic Operation   |
| UMOV                        | Uniform Move  |
| UP2UR                       | Uniform Predicate to Uniform Register                               |
| UPLOP3                      | Uniform Predicate Logic Operation                                   |
| UPOPC                       | Uniform Population Count  |
| UPRMT                       | Uniform Byte Permute  |
| UPSETP                      | Uniform Predicate Logic Operation                                   |
| UR2UP                       | Uniform Register to Uniform Predicate                               |
| USEL                        | Uniform Select  |
| USGXT                       | Uniform Sign Extend   |
| USHF                        | Uniform Funnel Shift  |
| USHL                        | Uniform Left Shift  |
| USHR                        | Uniform Right Shift   |
| VOTEU                       | Voting across SIMD Thread Group with Results in Uniform Destination |
| <b>Texture Instructions</b> |   |
| TEX                         | Texture Fetch   |
| TLD                         | Texture Load  |
| TLD4                        | Texture Load 4  |
| TMML                        | Texture MipMap Level  |
| TXD                         | Texture Fetch With Derivatives                                      |
| TXQ                         | Texture Query   |
| <b>Surface Instructions</b> |   |
| SUATOM                      | Atomic Op on Surface Memory   |

continues on next page

Table 3 – continued from previous page

| <b>Opcode</b>                     | <b>Description</b>                                 |
|-----------------------------------|--|
| SULD                              | Surface Load                                       |
| SURED                             | Reduction Op on Surface Memory                     |
| SUST                              | Surface Store                                      |
| <b>Control Instructions</b>       |  |
| BMOV                              | Move Convergence Barrier State                     |
| BPT                               | BreakPoint/Trap                                    |
| BRA                               | Relative Branch                                    |
| BREAK                             | Break out of the Specified Convergence Barrier     |
| BRX                               | Relative Branch Indirect                           |
| BRXU                              | Relative Branch with Uniform Register Based Offset |
| BSSY                              | Barrier Set Convergence Synchronization Point      |
| BSYNC                             | Synchronize Threads on a Convergence Barrier       |
| CALL                              | Call Function                                      |
| EXIT                              | Exit Program                                       |
| JMP                               | Absolute Jump                                      |
| JMX                               | Absolute Jump Indirect                             |
| JMXU                              | Absolute Jump with Uniform Register Based Offset   |
| KILL                              | Kill Thread  |
| NANOSLEEP                         | Suspend Execution                                  |
| RET                               | Return From Subroutine                             |
| RPCMOV                            | PC Register Move                                   |
| RTT                               | Return From Trap                                   |
| WARPSYNC                          | Synchronize Threads in Warp                        |
| YIELD                             | Yield Control                                      |
| <b>Miscellaneous Instructions</b> |  |
| B2R                               | Move Barrier To Register                           |
| BAR                               | Barrier Synchronization                            |
| CS2R                              | Move Special Register to Register                  |
| DEPBAR                            | Dependency Barrier                                 |
| GETLMEMBASE                       | Get Local Memory Base Address                      |
| LEPC                              | Load Effective PC                                  |
| NOP                               | No Operation                                       |

continues on next page

Table 3 – continued from previous page

| Opcode      | Description                       |
|-------------|-----------------------------------|
| PMTRIG      | Performance Monitor Trigger       |
| R2B         | Move Register to Barrier          |
| S2R         | Move Special Register to Register |
| SETCTAID    | Set CTA ID                        |
| SETLMEMBASE | Set Local Memory Base Address     |
| VOTE        | Vote Across SIMD Thread Group     |

## 6.4. Ampere and Ada Instruction Set

The Ampere and Ada architectures (Compute Capability 8.0 and 8.6) have the following instruction set format:

```
(instruction) (destination) (source1), (source2) ...
```

Valid destination and source locations include:

- ▶ RX for registers
- ▶ URX for uniform registers
- ▶ SRX for special system-controlled registers
- ▶ PX for predicate registers
- ▶ UPX for uniform predicate registers
- ▶ c[X][Y] for constant memory

Table 7 lists valid instructions for the Ampere and Ada GPUs.

Table 4: Table 7. Ampere and Ada Instruction Set

| Opcode                             | Description                 |
|------------------------------------|-----------------------------|
| <b>Floating Point Instructions</b> |                             |
| FADD                               | FP32 Add                    |
| FADD32I                            | FP32 Add                    |
| FCHK                               | Floating-point Range Check  |
| FFMA32I                            | FP32 Fused Multiply and Add |
| FFMA                               | FP32 Fused Multiply and Add |
| FMNMX                              | FP32 Minimum/Maximum        |
| FMUL                               | FP32 Multiply               |
| FMUL32I                            | FP32 Multiply               |

continues on next page

Table 4 – continued from previous page

| <b>Opcode</b>               | <b>Description</b>                 |
|-----------------------------|------------------------------------|
| FSEL                        | Floating Point Select              |
| FSET                        | FP32 Compare And Set               |
| FSETP                       | FP32 Compare And Set Predicate     |
| FSWZADD                     | FP32 Swizzle Add                   |
| MUFU                        | FP32 Multi Function Operation      |
| HADD2                       | FP16 Add                           |
| HADD2_32I                   | FP16 Add                           |
| HFMA2                       | FP16 Fused Mutiply Add             |
| HFMA2_32I                   | FP16 Fused Mutiply Add             |
| HMMA                        | Matrix Multiply and Accumulate     |
| HNMNMX2                     | FP16 Minimum / Maximum             |
| HMUL2                       | FP16 Multiply                      |
| HMUL2_32I                   | FP16 Multiply                      |
| HSET2                       | FP16 Compare And Set               |
| HSETP2                      | FP16 Compare And Set Predicate     |
| DADD                        | FP64 Add                           |
| DFMA                        | FP64 Fused Mutiply Add             |
| DMMA                        | Matrix Multiply and Accumulate     |
| DMUL                        | FP64 Multiply                      |
| DSETP                       | FP64 Compare And Set Predicate     |
| <b>Integer Instructions</b> |                                    |
| BMMA                        | Bit Matrix Multiply and Accumulate |
| BMSK                        | Bitfield Mask                      |
| BREV                        | Bit Reverse                        |
| FLO                         | Find Leading One                   |
| IABS                        | Integer Absolute Value             |
| IADD                        | Integer Addition                   |
| IADD3                       | 3-input Integer Addition           |
| IADD32I                     | Integer Addition                   |
| IDP                         | Integer Dot Product and Accumulate |
| IDP4A                       | Integer Dot Product and Accumulate |
| IMAD                        | Integer Multiply And Add           |

continues on next page

Table 4 – continued from previous page

| <b>Opcode</b>                  | <b>Description</b>                          |
|--------------------------------|---|
| IMMA                           | Integer Matrix Multiply and Accumulate      |
| IMNMX                          | Integer Minimum/Maximum                     |
| IMUL                           | Integer Multiply                            |
| IMUL32I                        | Integer Multiply                            |
| ISCADD                         | Scaled Integer Addition                     |
| ISCADD32I                      | Scaled Integer Addition                     |
| ISETP                          | Integer Compare And Set Predicate           |
| LEA                            | LOAD Effective Address                      |
| LOP                            | Logic Operation                             |
| LOP3                           | Logic Operation                             |
| LOP32I                         | Logic Operation                             |
| POPC                           | Population count                            |
| SHF                            | Funnel Shift                                |
| SHL                            | Shift Left                                  |
| SHR                            | Shift Right                                 |
| VABSDIFF                       | Absolute Difference                         |
| VABSDIFF4                      | Absolute Difference                         |
| <b>Conversion Instructions</b> |   |
| F2F                            | Floating Point To Floating Point Conversion |
| F2I                            | Floating Point To Integer Conversion        |
| I2F                            | Integer To Floating Point Conversion        |
| I2I                            | Integer To Integer Conversion               |
| I2IP                           | Integer To Integer Conversion and Packing   |
| I2FP                           | Integer to FP32 Convert and Pack            |
| F2IP                           | FP32 Down-Convert to Integer and Pack       |
| FRND                           | Round To Integer                            |
| <b>Movement Instructions</b>   |   |
| MOV                            | Move  |
| MOV32I                         | Move  |
| MOVM                           | Move Matrix with Transposition or Expansion |
| PRMT                           | Permute Register Pair                       |
| SEL                            | Select Source with Predicate                |

continues on next page

Table 4 – continued from previous page

| <b>Opcode</b>                        | <b>Description</b>   |
|--------------------------------------|--|
| SGXT                                 | Sign Extend  |
| SHFL                                 | Warp Wide Register Shuffle                                 |
| <b>Predicate Instructions</b>        |  |
| PLOP3                                | Predicate Logic Operation                                  |
| PSETP                                | Combine Predicates and Set Predicate                       |
| P2R                                  | Move Predicate Register To Register                        |
| R2P                                  | Move Register To Predicate Register                        |
| <b>Load/Store Instructions</b>       |  |
| LD                                   | Load from generic Memory                                   |
| LDC                                  | Load Constant  |
| LDG                                  | Load from Global Memory                                    |
| LDGDEPBAR                            | Global Load Dependency Barrier                             |
| LDGSTS                               | Asynchronous Global to Shared Memcopy                      |
| LDL                                  | Load within Local Memory Window                            |
| LDS                                  | Load within Shared Memory Window                           |
| LDSM                                 | Load Matrix from Shared Memory with Element Size Expansion |
| ST                                   | Store to Generic Memory                                    |
| STG                                  | Store to Global Memory                                     |
| STL                                  | Store to Local Memory                                      |
| STS                                  | Store to Shared Memory                                     |
| MATCH                                | Match Register Values Across Thread Group                  |
| QSPC                                 | Query Space  |
| ATOM                                 | Atomic Operation on Generic Memory                         |
| ATOMS                                | Atomic Operation on Shared Memory                          |
| ATOMG                                | Atomic Operation on Global Memory                          |
| RED                                  | Reduction Operation on Generic Memory                      |
| CCTL                                 | Cache Control  |
| CCTLL                                | Cache Control  |
| ERRBAR                               | Error Barrier  |
| MEMBAR                               | Memory Barrier   |
| CCTLT                                | Texture Cache Control                                      |
| <b>Uniform Datapath Instructions</b> |  |

continues on next page

Table 4 – continued from previous page

| <b>Opcode</b>               | <b>Description</b>  |
|-----------------------------|---|
| R2UR                        | Move from Vector Register to a Uniform Register                     |
| REDUX                       | Reduction of a Vector Register into a Uniform Register              |
| S2UR                        | Move Special Register to Uniform Register                           |
| UBMSK                       | Uniform Bitfield Mask   |
| UBREV                       | Uniform Bit Reverse   |
| UCLEA                       | Load Effective Address for a Constant                               |
| UF2FP                       | Uniform FP32 Down-convert and Pack                                  |
| UFLO                        | Uniform Find Leading One  |
| UIADD3                      | Uniform Integer Addition  |
| UIADD3.64                   | Uniform Integer Addition  |
| UIMAD                       | Uniform Integer Multiplication                                      |
| UISETP                      | Integer Compare and Set Uniform Predicate                           |
| ULDC                        | Load from Constant Memory into a Uniform Register                   |
| ULEA                        | Uniform Load Effective Address                                      |
| ULOP                        | Logic Operation   |
| ULOP3                       | Logic Operation   |
| ULOP32I                     | Logic Operation   |
| UMOV                        | Uniform Move  |
| UP2UR                       | Uniform Predicate to Uniform Register                               |
| UPLOP3                      | Uniform Predicate Logic Operation                                   |
| UPOPC                       | Uniform Population Count  |
| UPRMT                       | Uniform Byte Permute  |
| UPSETP                      | Uniform Predicate Logic Operation                                   |
| UR2UP                       | Uniform Register to Uniform Predicate                               |
| USEL                        | Uniform Select  |
| USGXT                       | Uniform Sign Extend   |
| USHF                        | Uniform Funnel Shift  |
| USHL                        | Uniform Left Shift  |
| USHR                        | Uniform Right Shift   |
| VOTEU                       | Voting across SIMD Thread Group with Results in Uniform Destination |
| <b>Texture Instructions</b> |   |
| TEX                         | Texture Fetch   |

continues on next page



Table 4 – continued from previous page

| <b>Opcode</b>                     | <b>Description</b>                                 |
|-----------------------------------|--|
| TLD                               | Texture Load                                       |
| TLD4                              | Texture Load 4                                     |
| TMML                              | Texture MipMap Level                               |
| TXD                               | Texture Fetch With Derivatives                     |
| TXQ                               | Texture Query                                      |
| <b>Surface Instructions</b>       |  |
| SUATOM                            | Atomic Op on Surface Memory                        |
| SULD                              | Surface Load                                       |
| SURED                             | Reduction Op on Surface Memory                     |
| SUST                              | Surface Store                                      |
| <b>Control Instructions</b>       |  |
| BMOV                              | Move Convergence Barrier State                     |
| BPT                               | BreakPoint/Trap                                    |
| BRA                               | Relative Branch                                    |
| BREAK                             | Break out of the Specified Convergence Barrier     |
| BRX                               | Relative Branch Indirect                           |
| BRXU                              | Relative Branch with Uniform Register Based Offset |
| BSSY                              | Barrier Set Convergence Synchronization Point      |
| BSYNC                             | Synchronize Threads on a Convergence Barrier       |
| CALL                              | Call Function                                      |
| EXIT                              | Exit Program                                       |
| JMP                               | Absolute Jump                                      |
| JMX                               | Absolute Jump Indirect                             |
| JMXU                              | Absolute Jump with Uniform Register Based Offset   |
| KILL                              | Kill Thread  |
| NANOSLEEP                         | Suspend Execution                                  |
| RET                               | Return From Subroutine                             |
| RPCMOV                            | PC Register Move                                   |
| WARPSYNC                          | Synchronize Threads in Warp                        |
| YIELD                             | Yield Control                                      |
| <b>Miscellaneous Instructions</b> |  |
| B2R                               | Move Barrier To Register                           |

continues on next page

Table 4 – continued from previous page

| Opcode      | Description                       |
|-------------|-----------------------------------|
| BAR         | Barrier Synchronization           |
| CS2R        | Move Special Register to Register |
| DEPBAR      | Dependency Barrier                |
| GETLMEMBASE | Get Local Memory Base Address     |
| LEPC        | Load Effective PC                 |
| NOP         | No Operation                      |
| PMTRIG      | Performance Monitor Trigger       |
| S2R         | Move Special Register to Register |
| SETCTAID    | Set CTA ID                        |
| SETLMEMBASE | Set Local Memory Base Address     |
| VOTE        | Vote Across SIMD Thread Group     |

## 6.5. Hopper Instruction Set

The Hopper architecture (Compute Capability 9.0) has the following instruction set format:

```
(instruction) (destination) (source1), (source2) ...
```

Valid destination and source locations include:

- ▶ RX for registers
- ▶ URX for uniform registers
- ▶ SRX for special system-controlled registers
- ▶ PX for predicate registers
- ▶ UPX for uniform predicate registers
- ▶ c[X][Y] for constant memory
- ▶ desc[URX][RY] for memory descriptors

Table 8 lists valid instructions for the Hopper GPUs.

Table 5: Table 8. Hopper Instruction Set

| Opcode                             | Description |
|------------------------------------|-------------|
| <b>Floating Point Instructions</b> |             |
| FADD                               | FP32 Add    |
| FADD32I                            | FP32 Add    |

continues on next

Table 5 – continued from previous page

| <b>Opcode</b>               | <b>Description</b>                 |
|-----------------------------|------------------------------------|
| FCHK                        | Floating-point Range Check         |
| FFMA32I                     | FP32 Fused Multiply and Add        |
| FFMA                        | FP32 Fused Multiply and Add        |
| FMNMX                       | FP32 Minimum/Maximum               |
| FMUL                        | FP32 Multiply                      |
| FMUL32I                     | FP32 Multiply                      |
| FSEL                        | Floating Point Select              |
| FSET                        | FP32 Compare And Set               |
| FSETP                       | FP32 Compare And Set Predicate     |
| FSWZADD                     | FP32 Swizzle Add                   |
| MUFU                        | FP32 Multi Function Operation      |
| HADD2                       | FP16 Add                           |
| HADD2_32I                   | FP16 Add                           |
| HFMA2                       | FP16 Fused Mutiply Add             |
| HFMA2_32I                   | FP16 Fused Mutiply Add             |
| HMMA                        | Matrix Multiply and Accumulate     |
| HMNMX2                      | FP16 Minimum / Maximum             |
| HMUL2                       | FP16 Multiply                      |
| HMUL2_32I                   | FP16 Multiply                      |
| HSET2                       | FP16 Compare And Set               |
| HSETP2                      | FP16 Compare And Set Predicate     |
| DADD                        | FP64 Add                           |
| DFMA                        | FP64 Fused Mutiply Add             |
| DMMA                        | Matrix Multiply and Accumulate     |
| DMUL                        | FP64 Multiply                      |
| DSETP                       | FP64 Compare And Set Predicate     |
| <b>Integer Instructions</b> |                                    |
| BMMA                        | Bit Matrix Multiply and Accumulate |
| BMSK                        | Bitfield Mask                      |
| BREV                        | Bit Reverse                        |
| FLO                         | Find Leading One                   |
| IABS                        | Integer Absolute Value             |

continues on next

Table 5 – continued from previous page

| <b>Opcode</b>                  | <b>Description</b>                                 |
|--------------------------------|--|
| IADD                           | Integer Addition                                   |
| IADD3                          | 3-input Integer Addition                           |
| IADD32I                        | Integer Addition                                   |
| IDP                            | Integer Dot Product and Accumulate                 |
| IDP4A                          | Integer Dot Product and Accumulate                 |
| IMAD                           | Integer Multiply And Add                           |
| IMMA                           | Integer Matrix Multiply and Accumulate             |
| IMNMX                          | Integer Minimum/Maximum                            |
| IMUL                           | Integer Multiply                                   |
| IMUL32I                        | Integer Multiply                                   |
| ISCADD                         | Scaled Integer Addition                            |
| ISCADD32I                      | Scaled Integer Addition                            |
| ISETP                          | Integer Compare And Set Predicate                  |
| LEA                            | LOAD Effective Address                             |
| LOP                            | Logic Operation                                    |
| LOP3                           | Logic Operation                                    |
| LOP32I                         | Logic Operation                                    |
| POPC                           | Population count                                   |
| SHF                            | Funnel Shift                                       |
| SHL                            | Shift Left   |
| SHR                            | Shift Right  |
| VABSDIFF                       | Absolute Difference                                |
| VABSDIFF4                      | Absolute Difference                                |
| VHMNMX                         | SIMD FP16 3-Input Minimum / Maximum                |
| VIADD                          | SIMD Integer Addition                              |
| VIADDMNMX                      | SIMD Integer Addition and Fused Min/Max Comparison |
| VIMNMX                         | SIMD Integer Minimum / Maximum                     |
| VIMNMX3                        | SIMD Integer 3-Input Minimum / Maximum             |
| <b>Conversion Instructions</b> |  |
| F2F                            | Floating Point To Floating Point Conversion        |
| F2I                            | Floating Point To Integer Conversion               |
| I2F                            | Integer To Floating Point Conversion               |

continues on next

Table 5 – continued from previous page

| Opcode                         | Description  |
|--------------------------------|--|
| I2I                            | Integer To Integer Conversion                              |
| I2IP                           | Integer To Integer Conversion and Packing                  |
| I2FP                           | Integer to FP32 Convert and Pack                           |
| F2IP                           | FP32 Down-Convert to Integer and Pack                      |
| FRND                           | Round To Integer   |
| <b>Movement Instructions</b>   |  |
| MOV                            | Move   |
| MOV32I                         | Move   |
| MOVM                           | Move Matrix with Transposition or Expansion                |
| PRMT                           | Permute Register Pair                                      |
| SEL                            | Select Source with Predicate                               |
| SGXT                           | Sign Extend  |
| SHFL                           | Warp Wide Register Shuffle                                 |
| <b>Predicate Instructions</b>  |  |
| PLOP3                          | Predicate Logic Operation                                  |
| PSETP                          | Combine Predicates and Set Predicate                       |
| P2R                            | Move Predicate Register To Register                        |
| R2P                            | Move Register To Predicate Register                        |
| <b>Load/Store Instructions</b> |  |
| FENCE                          | Memory Visibility Guarantee for Shared or Global Memory    |
| LD                             | Load from generic Memory                                   |
| LDC                            | Load Constant  |
| LDG                            | Load from Global Memory                                    |
| LDGDEPBAR                      | Global Load Dependency Barrier                             |
| LDGMC                          | Reducing Load  |
| LDGSTS                         | Asynchronous Global to Shared Memcopy                      |
| LDL                            | Load within Local Memory Window                            |
| LDS                            | Load within Shared Memory Window                           |
| LDSM                           | Load Matrix from Shared Memory with Element Size Expansion |
| STSM                           | Store Matrix to Shared Memory                              |
| ST                             | Store to Generic Memory                                    |
| STG                            | Store to Global Memory                                     |

continues on next

Table 5 – continued from previous page

| <b>Opcode</b>                        | <b>Description</b>  |
|--------------------------------------|---|
| STL                                  | Store to Local Memory   |
| STS                                  | Store to Shared Memory  |
| STAS                                 | Asynchronous Store to Distributed Shared Memory With Explicit Synchroniz          |
| SYNCS                                | Sync Unit   |
| MATCH                                | Match Register Values Across Thread Group   |
| QSPC                                 | Query Space   |
| ATOM                                 | Atomic Operation on Generic Memory  |
| ATOMS                                | Atomic Operation on Shared Memory   |
| ATOMG                                | Atomic Operation on Global Memory   |
| REDAS                                | Asynchronous Reduction on Distributed Shared Memory With Explicit Synchronization |
| REDG                                 | Reduction Operation on Generic Memory   |
| CCTL                                 | Cache Control   |
| CCTLL                                | Cache Control   |
| ERRBAR                               | Error Barrier   |
| MEMBAR                               | Memory Barrier  |
| CCTLT                                | Texture Cache Control   |
| <b>Uniform Datapath Instructions</b> |   |
| R2UR                                 | Move from Vector Register to a Uniform Register                                   |
| REDUX                                | Reduction of a Vector Register into a Uniform Register                            |
| S2UR                                 | Move Special Register to Uniform Register   |
| UBMSK                                | Uniform Bitfield Mask   |
| UBREV                                | Uniform Bit Reverse   |
| UCGABAR_ARV                          | CGA Barrier Synchronization   |
| UCGABAR_WAIT                         | CGA Barrier Synchronization   |
| UCLEA                                | Load Effective Address for a Constant   |
| UF2FP                                | Uniform FP32 Down-convert and Pack  |
| UFLO                                 | Uniform Find Leading One  |
| UIADD3                               | Uniform Integer Addition  |
| UIADD3.64                            | Uniform Integer Addition  |
| UIMAD                                | Uniform Integer Multiplication  |
| UISETP                               | Integer Compare and Set Uniform Predicate   |

continues on next

Table 5 – continued from previous page

| <b>Opcode</b>                            | <b>Description</b>  |
|--|---|
| ULDC                                     | Load from Constant Memory into a Uniform Register                   |
| ULEA                                     | Uniform Load Effective Address                                      |
| ULEPC                                    | Uniform Load Effective PC   |
| ULOP                                     | Logic Operation   |
| ULOP3                                    | Logic Operation   |
| ULOP32I                                  | Logic Operation   |
| UMOV                                     | Uniform Move  |
| UP2UR                                    | Uniform Predicate to Uniform Register                               |
| UPLOP3                                   | Uniform Predicate Logic Operation                                   |
| UPOPC                                    | Uniform Population Count  |
| UPRMT                                    | Uniform Byte Permute  |
| UPSETP                                   | Uniform Predicate Logic Operation                                   |
| UR2UP                                    | Uniform Register to Uniform Predicate                               |
| USEL                                     | Uniform Select  |
| USETMAXREG                               | Release, Deallocate and Allocate Registers                          |
| USGXT                                    | Uniform Sign Extend   |
| USHF                                     | Uniform Funnel Shift  |
| USHL                                     | Uniform Left Shift  |
| USHR                                     | Uniform Right Shift   |
| VOTEU                                    | Voting across SIMD Thread Group with Results in Uniform Destination |
| <b>Warpgroup Instructions</b>            |   |
| BGMMA                                    | Bit Matrix Multiply and Accumulate Across Warps                     |
| HGMMA                                    | Matrix Multiply and Accumulate Across a Warpgroup                   |
| IGMMA                                    | Integer Matrix Multiply and Accumulate Across a Warpgroup           |
| QGMMA                                    | FP8 Matrix Multiply and Accumulate Across a Warpgroup               |
| WARPGROUP                                | Warpgroup Synchronization   |
| WARPGROUPSET                             | Set Warpgroup Counters  |
| <b>Tensor Memory Access Instructions</b> |   |
| UBLKCP                                   | Bulk Data Copy  |
| UBLKPF                                   | Bulk Data Prefetch  |
| UBLKRED                                  | Bulk Data Copy from Shared Memory with Reduction                    |
| UTMACCTL                                 | TMA Cache Control   |

continues on next

Table 5 – continued from previous page

| <b>Opcode</b>               | <b>Description</b>                                       |
|-----------------------------|--|
| UTMACMDFLUSH                | TMA Command Flush  |
| UTMALDG                     | Tensor Load from Global to Shared Memory                 |
| UTMAPF                      | Tensor Prefetch  |
| UTMAREDG                    | Tensor Store from Shared to Global Memory with Reduction |
| UTMASTG                     | Tensor Store from Shared to Global Memory                |
| <b>Texture Instructions</b> |  |
| TEX                         | Texture Fetch  |
| TLD                         | Texture Load   |
| TLD4                        | Texture Load 4   |
| TMML                        | Texture MipMap Level                                     |
| TXD                         | Texture Fetch With Derivatives                           |
| TXQ                         | Texture Query  |
| <b>Surface Instructions</b> |  |
| SUATOM                      | Atomic Op on Surface Memory                              |
| SULD                        | Surface Load   |
| SURED                       | Reduction Op on Surface Memory                           |
| SUST                        | Surface Store  |
| <b>Control Instructions</b> |  |
| ACQBULK                     | Wait for Bulk Release Status Warp State                  |
| BMOV                        | Move Convergence Barrier State                           |
| BPT                         | BreakPoint/Trap  |
| BRA                         | Relative Branch  |
| BREAK                       | Break out of the Specified Convergence Barrier           |
| BRX                         | Relative Branch Indirect                                 |
| BRXU                        | Relative Branch with Uniform Register Based Offset       |
| BSSY                        | Barrier Set Convergence Synchronization Point            |
| BSYNC                       | Synchronize Threads on a Convergence Barrier             |
| CALL                        | Call Function  |
| CGAERRBAR                   | CGA Error Barrier  |
| ELECT                       | Elect a Leader Thread                                    |
| ENDCOLLECTIVE               | Reset the MCOLLECTIVE mask                               |
| EXIT                        | Exit Program   |

continues on next



Table 5 – continued from previous page

| <b>Opcode</b>                     | <b>Description</b>                               |
|-----------------------------------|--|
| JMP                               | Absolute Jump                                    |
| JMX                               | Absolute Jump Indirect                           |
| JMXU                              | Absolute Jump with Uniform Register Based Offset |
| KILL                              | Kill Thread                                      |
| NANOSLEEP                         | Suspend Execution                                |
| PREEXIT                           | Dependent Task Launch Hint                       |
| RET                               | Return From Subroutine                           |
| RPCMOV                            | PC Register Move                                 |
| WARPSYNC                          | Synchronize Threads in Warp                      |
| YIELD                             | Yield Control                                    |
| <b>Miscellaneous Instructions</b> |  |
| B2R                               | Move Barrier To Register                         |
| BAR                               | Barrier Synchronization                          |
| CS2R                              | Move Special Register to Register                |
| DEPBAR                            | Dependency Barrier                               |
| GETLMEMBASE                       | Get Local Memory Base Address                    |
| LEPC                              | Load Effective PC                                |
| NOP                               | No Operation                                     |
| PMTRIG                            | Performance Monitor Trigger                      |
| S2R                               | Move Special Register to Register                |
| SETCTAID                          | Set CTA ID                                       |
| SETLMEMBASE                       | Set Local Memory Base Address                    |
| VOTE                              | Vote Across SIMT Thread Group                    |



---

# Chapter 7. `cu++filt`

`cu++filt` decodes (demangles) low-level identifiers that have been mangled by CUDA C++ into user readable names. For every input alphanumeric word, the output of `cu++filt` is either the demangled name if the name decodes to a CUDA C++ name, or the original name itself.

## 7.1. Usage

`cu++filt` accepts one or more alphanumeric words (consisting of letters, digits, underscores, dollars, or periods) and attempts to decipher them. The basic usage is as following:

```
cu++filt [options] <symbol(s)>
```

To demangle an entire file, like a binary, pipe the contents of the file to `cu++filt`, such as in the following command:

```
nm <input file> | cu++filt
```

To demangle function names without printing their parameter types, use the following command :

```
cu++filt -p <symbol(s)>
```

To skip a leading underscore from mangled symbols, use the following command:

```
cu++filt -_ <symbol(s)>
```

Here's a sample output of `cu++filt`:

```
$ cu++filt _Z1fIiEb1
bool f<int>(long)
```

As shown in the output, the symbol `_Z1fIiEb1` was successfully demangled.

To strip all types in the function signature and parameters, use the `-p` option:

```
$ cu++filt -p _Z1fIiEb1
f<int>
```

To skip a leading underscore from a mangled symbol, use the `-_` option:

```
$ cu++filt -_ __Z1fIiEb1
bool f<int>(long)
```

To demangle an entire file, pipe the contents of the file to `cu++filt`:

```
$ nm test.sm_70.cubin | cu++filt
0000000000000000 t hello(char *)
0000000000000070 t hello(char *)::display()
0000000000000000 T hello(int *)
```

Symbols that cannot be demangled are printed back to stdout as is:

```
$ cu++filt _ZD2
_ZD2
```

Multiple symbols can be demangled from the command line:

```
$ cu++filt _ZN6Scope15Func1Enez _Z3fooIiPFYneEiEvv _ZD2
Scope1::Func1(__int128, long double, ...)
void foo<int, __int128 (*)>(long double, int>()
_ZD2
```

## 7.2. Command-line Options

Table 9 contains supported command-line options of `cu++filt`, along with a description of what each option does.

Table 1: Table 9. `cu++filt` Command-line Options

| Op-tion | Description  |
|---------|--|
| -_      | Strip underscore. On some systems, the CUDA compiler puts an underscore in front of every name. This option removes the initial underscore. Whether <code>cu++filt</code> removes the underscore by default is target dependent. |
| -p      | When demangling the name of a function, do not display the types of the function's parameters.   |
| -h      | Print a summary of the options to <code>cu++filt</code> and exit.  |
| -v      | Print the version information of this tool.  |

## 7.3. Library Availability

`cu++filt` is also available as a static library (`libcufilt`) that can be linked against an existing project. The following interface describes its usage:

```
char* __cu_demangle(const char *id, char *output_buffer, size_t *length, int *status)
```

This interface can be found in the file “`nv_decode.h`” located in the SDK.

### Input Parameters

*id* Input mangled string.

*output\_buffer* Pointer to where the demangled buffer will be stored. This memory must be allocated with malloc. If output-buffer is NULL, memory will be malloc'd to store the demangled name and returned through the function return value. If the output-buffer is too small, it is expanded using realloc.

*length* It is necessary to provide the size of the output buffer if the user is providing pre-allocated memory. This is needed by the demangler in case the size needs to be reallocated. If the length is non-null, the length of the demangled buffer is placed in length.

*status* \*status is set to one of the following values:

- ▶ 0 - The demangling operation succeeded
- ▶ -1 - A memory allocation failure occurred
- ▶ -2 - Not a valid mangled id
- ▶ -3 - An input validation failure has occurred (one or more arguments are invalid)

### Return Value

A pointer to the start of the NUL-terminated demangled name, or NULL if the demangling fails. The caller is responsible for deallocating this memory using free.

**Note:** This function is thread-safe.

### Example Usage

```
#include <stdio.h>
#include <stdlib.h>
#include "nv_decode.h"

int main()
{
    int    status;
    const char *real_mangled_name="_ZN8clstmp01I5cls01E13clstmp01_mf01Ev";
    const char *fake_mangled_name="B@d_iDentiFier";

    char* realname = __cu_demangle(fake_mangled_name, 0, 0, &status);
    printf("fake_mangled_name:\t result => %s\t status => %d\n", realname, status);
    free(realname);

    size_t size = sizeof(char)*1000;
    realname = (char*)malloc(size);
    __cu_demangle(real_mangled_name, realname, &size, &status);
    printf("real_mangled_name:\t result => %s\t status => %d\n", realname, status);
    free(realname);

    return 0;
}
```

This prints:

```
fake_mangled_name:   result => (null)      status => -2
real_mangled_name:  result => clstmp01<cls01>.:clstmp01_mf01()  status => 0
```



---

# Chapter 8. nvprune

*nvprune* prunes host object files and libraries to only contain device code for the specified targets.

## 8.1. Usage

*nvprune* accepts a single input file each time it's run, emitting a new output file. The basic usage is as following:

```
nvprune [options] -o <outfile> <infile>
```

The input file must be either a relocatable host object or static library (not a host executable), and the output file will be the same format.

Either the `-arch` or `-generate-code` option must be used to specify the target(s) to keep. All other device code is discarded from the file. The targets can be either a `sm_NN` arch (cubin) or `compute_NN` arch (ptx).

For example, the following will prune `libcublas_static.a` to only contain `sm_70` cubin rather than all the targets which normally exist:

```
nvprune -arch sm_70 libcublas_static.a -o libcublas_static70.a
```

Note that this means that `libcublas_static70.a` will not run on any other architecture, so should only be used when you are building for a single architecture.

## 8.2. Command-line Options

[Table 10](#) contains supported command-line options of *nvprune*, along with a description of what each option does. Each option has a long name and a short name, which can be used interchangeably.

Table 1: Table 10. nvprune Command-line Options

| Option (long)  | Option (short)     | Description   |
|--|--------------------|---|
| <code>--arch &lt;gpu architecture name&gt;, ...</code> | <code>-arch</code> | Specify the name of the NVIDIA GPU architecture which will remain in the object or library.   |
| <code>--generate-code</code>                           | <code>-code</code> | This option is same format as <code>nvcc -generate-code</code> option, and provides a way to specify multiple architectures which should remain in the object or library. Only the 'code' values are used as targets to match. Allowed keywords for this option: 'arch','code'. |
| <code>--no-relocatable-elf</code>                      | <code>-r</code>    | Do not create relocatable ELF.  |
| <code>--output-file</code>                             | <code>-o</code>    | Specify name and location of the output file.   |
| <code>--help</code>                                    | <code>-h</code>    | Print this help information on this tool.   |
| <code>--options-file &lt;file&gt;, ...</code>          | <code>-optf</code> | Include command line options from specified file.   |
| <code>--version</code>                                 | <code>-V</code>    | Print version information on this tool.   |



---

# Chapter 9. Notices

## 9.1. Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or

services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

## 9.2. OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

## 9.3. Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

## Copyright

©2013-2023, NVIDIA Corporation & affiliates. All rights reserved