# CUDA Binary Utilities

Application Note

DA-06762-001_v11.7 | May 2022

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1. Overview

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

## 1.1. 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.

## 1.2. Differences between `cuobjdump` and `nvdisasm`

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

Here's a quick comparison of the two tools:

Table 1. Comparison of `cuobjdump` and `nvdisasm`

| | **cuobjdump** | **nvdisasm** |
|---|---|---|
| Disassemble cubin | Yes | Yes |
| Extract ptx and extract and disassemble cubin from the following input files:<br>▶ Host binaries | Yes | No |

| | cuobjdump | nvdisasm |
|---|---|---|
| ▶ Executables<br>▶ Object files<br>▶ Static libraries<br>▶ External fatbinary files | | |
| Control flow analysis and output | No | Yes |
| Advanced display options | No | Yes |

# 1.3. Command Option Types and Notation

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

- cuobjdump
- nvdisasm
- 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 : nvdisams --print-raw <file>
Single value   : nvdisasm --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 seperated 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.

```
nvdisasm.exe -b SM70 -b SM75 test.bin
nvdisasm 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 2. 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.

## 2.1. 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)"
/*0000*/      IMAD.MOV.U32 R1, RZ, RZ, c[0x0][0x28] ;  /* 0x00000a00ff017624 */
                                                       /* 0x000fd000078e00ff */
/*0010*/ @!PT SHFL.IDX PT, RZ, RZ, RZ, RZ ;            /* 0x000000fffffff389 */
                                                       /* 0x000fe200000e00ff */
/*0020*/      IMAD.MOV.U32 R2, RZ, RZ, c[0x0][0x160] ; /* 0x00005800ff027624 */
                                                       /* 0x000fe200078e00ff */
/*0030*/      MOV R3, c[0x0][0x164] ;                  /* 0x0000590000037a02 */
                                                       /* 0x000fe20000000f00 */
/*0040*/      IMAD.MOV.U32 R4, RZ, RZ, c[0x0][0x168] ; /* 0x00005a00ff047624 */
                                                       /* 0x000fe200078e00ff */
/*0050*/      MOV R5, c[0x0][0x16c] ;                  /* 0x00005b0000057a02 */
                                                       /* 0x000fcc0000000f00 */
/*0060*/      LDG.E.SYS R2, [R2] ;                     /* 0x0000000002027381 */
                                                       /* 0x000ea800001ee900 */
/*0070*/      LDG.E.SYS R5, [R4] ;                     /* 0x0000000004057381 */
                                                       /* 0x000ea200001ee900 */
/*0080*/      IMAD.MOV.U32 R6, RZ, RZ, c[0x0][0x170] ; /* 0x00005c00ff067624 */
                                                       /* 0x000fe200078e00ff */
/*0090*/      MOV R7, c[0x0][0x174] ;                  /* 0x00005d0000077a02 */
                                                       /* 0x000fe40000000f00 */
/*00a0*/      IADD3 R9, R2, R5, RZ ;                   /* 0x0000000502097210 */
                                                       /* 0x004fd00007ffe0ff */
/*00b0*/      STG.E.SYS [R6], R9 ;                     /* 0x0000000906007386 */
                                                       /* 0x000fe2000010e900 */
/*00c0*/      EXIT ;                                   /* 0x000000000000794d */
                                                       /* 0x000fea0003800000 */
/*00d0*/      BRA 0xd0;                                /* 0xfffffff000007947 */
                                                       /* 0x000fc0000383ffff */
/*00e0*/      NOP;                                     /* 0x0000000000007918 */
                                                       /* 0x000fc00000000000 */
/*00f0*/      NOP;                                     /* 0x0000000000007918 */
                                                       /* 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>;

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:
  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.

## 2.2. 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 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` | `-symbols` | 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` | `-res-usage` | Dump 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. |
| `--function <function name>,...` | `-fun` | Specify names of device functions whose fat binary structures must be dumped. |
| `--function-index <function index>` | `-findex` | 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_35`','`sm_37`','`sm_50`','`sm_52`','`sm_53`','`sm_60`', '`sm_61`','`sm_62`','`sm_70`','`sm_71`','`sm_72`','`sm_73`', '`sm_75`','`sm_80`','`sm_86`'. |
| `--help` | `-h` | Print this help information on this tool. |

| Option (long) | Option (short) | Description |
| --- | --- | --- |
| `--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. |
| `--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 3. nvdisasm

nvdisasm extracts information from standalone cubin files and presents them in human readable format. The output of nvdisasm includes CUDA assembly code for each kernel, listing of ELF data sections and other CUDA specific sections. Output style and options are controlled through nvdisasm command-line options. nvdisasm is also capable to do control flow analysis to annotate jump/branch targets and data flow analysis to analyse register usage.

> **Note:** nvdisasm requires complete relocation information to do control flow analysis. If this information is missing from the CUDA binary, either use the nvdisasm 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.

## 3.1. Usage

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

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

Here's a sample output of nvdisasm:

```
    .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,@"STO_CUDA_ENTRY STV_DEFAULT"
_Z9acos_main10acosParams:
```

```
.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:
```

## 3.1.1. Control flow graph information

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

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

`nvdisasm` generates control flow in the DOT graph description language format. The output of the control flow from nvdisasm can be imported to a DOT graph visualization tool such as Graphviz.

To generate a PNG image (`cfg.png`) of the control flow of the above cubin (`a.cubin`) with `nvdisasm` and Graphviz:

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

The generated visual graph is:

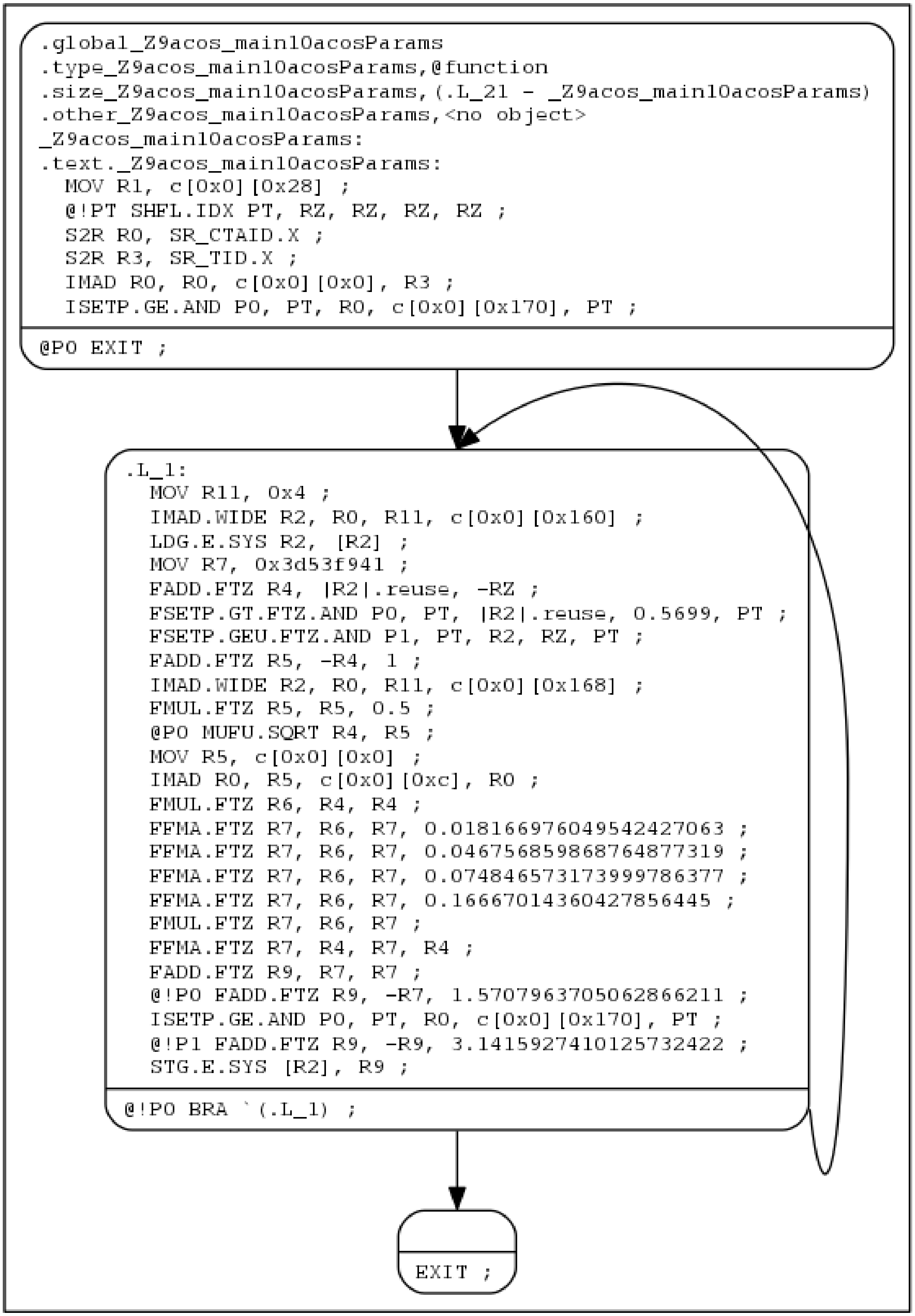
Figure 1. Control Flow Graph


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

```
nvdisasm -bbcfg a.cubin | dot -obbcfg.png -Tpng
```

Here's the generated graph:

Figure 2. Basic Block Control Flow Graph



## 3.1.2. Register liveness information

`nvdisasm` is capable of showing register (general and predicate) liveness range information. For each line of CUDA assembly, `nvdisasm` 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 |
                                                      // |                 |      |
                                                      // |                 |      |
                                                      // |    000000000011 |      |
                                                      // |  # 012345678901 | # 01 |
                                                      // +-----------------+------+
    .global acos                                      // |                 |      |
    .type   acos,@function                            // |                 |      |
    .size   acos,(.L_21 - acos)                       // |                 |      |
    .other  acos,@"STO_CUDA_ENTRY STV_DEFAULT"        // |                 |      |
acos:                                                 // |                 |      |
.text.acos:                                           // |                 |      |
    MOV R1, c[0x0][0x28] ;                            // |  1  ^           |      |
    NOP;                                              // |  1  ^           |      |
    S2R R0, SR_CTAID.X ;                              // |  2 ^:           |      |
    S2R R3, SR_TID.X ;                                // |  3 :: ^         |      |
    IMAD R0, R0, c[0x0][0x0], R3 ;                    // |  3 x: v         |      |
    ISETP.GE.AND P0, PT, R0, c[0x0][0x170], PT ;      // |  2 v:           | 1 ^  |
@P0 EXIT ;                                            // |  2 ::           | 1 v  |
.L_1:                                                 // |  2 ::           |      |
     MOV R11, 0x4 ;                                   // |  3 ::         ^ |      |
     IMAD.WIDE R2, R0, R11, c[0x0][0x160] ;           // |  5 v:^^       v |      |
     LDG.E.SYS R2, [R2] ;                             // |  4 ::^        : |      |
     MOV R7, 0x3d53f941 ;                             // |  5 :::    ^   : |      |
     FADD.FTZ R4, |R2|.reuse, -RZ ;                   // |  6 ::v ^  :   : |      |
     FSETP.GT.FTZ.AND P0, PT, |R2|.reuse, 0.5699, PT; // |  6 ::v :  :   : | 1 ^  |
     FSETP.GEU.FTZ.AND P1, PT, R2, RZ, PT ;           // |  6 ::v :  :   : | 2 :^ |
     FADD.FTZ R5, -R4, 1 ;                            // |  6 ::  v^ :   : | 2 :: |
     IMAD.WIDE R2, R0, R11, c[0x0][0x168] ;           // |  8 v:^^:: :   v | 2 :: |
     FMUL.FTZ R5, R5, 0.5 ;                           // |  5 ::  :x :     | 2 :: |
 @P0 MUFU.SQRT R4, R5 ;                               // |  5 ::  ^v :     | 2 v: |
     MOV R5, c[0x0][0x0] ;                            // |  5 ::  :^ :     | 2 :: |
     IMAD R0, R5, c[0x0][0xc], R0 ;                   // |  5 x:  :v :     | 2 :: |
     FMUL.FTZ R6, R4, R4 ;                            // |  5 ::  v ^:     | 2 :: |
     FFMA.FTZ R7, R6, R7, 0.018166976049542427063 ;   // |  5 ::  : vx     | 2 :: |
     FFMA.FTZ R7, R6, R7, 0.046756859868764877319 ;   // |  5 ::  : vx     | 2 :: |
     FFMA.FTZ R7, R6, R7, 0.074846573173999786377 ;   // |  5 ::  : vx     | 2 :: |
     FFMA.FTZ R7, R6, R7, 0.16667014360427856445 ;    // |  5 ::  : vx     | 2 :: |
     FMUL.FTZ R7, R6, R7 ;                            // |  5 ::  : vx     | 2 :: |
     FFMA.FTZ R7, R4, R7, R4 ;                        // |  4 ::  v  x     | 2 :: |
     FADD.FTZ R9, R7, R7 ;                            // |  4 ::     v ^   | 2 :: |
@!P0 FADD.FTZ R9, -R7, 1.5707963705062866211 ;        // |  4 ::     v ^   | 2 v: |
     ISETP.GE.AND P0, PT, R0, c[0x0][0x170], PT ;     // |  3 v:       :   | 2 ^: |
@!P1 FADD.FTZ R9, -R9, 3.1415927410125732422 ;        // |  3 ::       x   | 2 :v |
     STG.E.SYS [R2], R9 ;                             // |  3 ::       v   | 1 :  |
@!P0 BRA `(.L_1) ;                                    // |  2 ::           | 1 v  |
     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
```

## 3.1.3. Debug information

`nvdisasm` 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:

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

Here's a sample output of a kernel using `nvdisasm -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:
        /*0000*/                   MOV R1, c[0x0][0x28] ;
        /*0010*/                   NOP;
    //## File "/home/user/cuda/sample/sample.cu", line 25
        /*0020*/                   MOV R0, 0x160 ;
        /*0030*/                   LDC R0, c[0x0][R0] ;
        /*0040*/                   MOV R0, R0 ;
        /*0050*/                   MOV R2, R0 ;
    //## File "/home/user/cuda/sample/sample.cu", line 26
        /*0060*/                   MOV R4, R2 ;
        /*0070*/                   MOV R20, 32@lo((_Z6kernali + .L_1@srel)) ;
        /*0080*/                   MOV R21, 32@hi((_Z6kernali + .L_1@srel)) ;
        /*0090*/                   CALL.ABS.NOINC `(_Z3fooi) ;
.L_1:
        /*00a0*/                   MOV R0, R4 ;
        /*00b0*/                   MOV R4, R2 ;
        /*00c0*/                   MOV R2, R0 ;
        /*00d0*/                   MOV R20, 32@lo((_Z6kernali + .L_2@srel)) ;
        /*00e0*/                   MOV R21, 32@hi((_Z6kernali + .L_2@srel)) ;
        /*00f0*/                   CALL.ABS.NOINC `(_Z3bari) ;
.L_2:
        /*0100*/                   MOV R4, R4 ;
        /*0110*/                   IADD3 R4, R2, R4, RZ ;
        /*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:
```

`nvdisasm` 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 `nvdisasm -g` command.

Here's a sample output of a kernel using `nvdisasm -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
        /*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:
```

# 3.2. Command-line Options

Table 3 contains the supported command-line options of `nvdisasm`, 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 3. nvdisasm Command-line Options

| Option (long) | Option (short) | Description |
|---|---|---|
| `--base-address <value>` | `-base` | Specify the logical base address of the image to disassemble. This option is only valid when disassembling a raw instruction binary (see option '`--binary`'), and is ignored when disassembling an Elf file. Default value: 0. |
| `--binary <SMxy>` | `-b` | When this option is specified, the input file is assumed to contain a raw instruction binary, that is, a sequence of binary instruction encodings |

| Option (long) | Option (short) | Description |
|---|---|---|
| | | 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: 'SM35','SM37','SM50','SM52','SM53','SM60','SM61', 'SM62','SM70','SM72','SM73','SM75','SM80','SM86'. |
| `--cuda-function-index <symbol index>,...` | `-fun` | 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. |
| `--help` | `-h` | Print this help information on this tool. |
| `--life-range-mode` | `-lrm` | This option implies option '`--print-life-ranges`', and determines how register live range info should be printed. '`count`': Not at all, leaving only the # column (number of live registers); '`wide`': Columns spaced out for readability (default); '`narrow`': A one-character column for each register, economizing on table width. Allowed values for this option: '`count`','`narrow`','`wide`'. |
| `--no-dataflow` | `-ndf` | 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 nvelf/cubin are not met. |
| `--no-vliw` | `-novliw` | Conventional mode; disassemble paired instructions in normal syntax, instead of VLIW syntax. |
| `--options-file <file>,...` | `-optf` | Include command line options from specified file. |
| `--output-control-flow-graph` | `-cfg` | When specified, output the control flow graph where each node is a hyperblock, in a format consumable by graphviz tools (such as dot). Example use briefed in Control flow graph information. |
| `--output-control-flow-graph-with-basic-blocks` | `-bbcfg` | When specified, output the control flow graph where each node is a basicblock, in a format consumable by graphviz tools (such as dot). Example use briefed in Control flow graph information. |
| `--print-code` | `-c` | Only print code sections. |
| `--print-instr-offsets-cfg` | `-poff` | When specified, print instruction offsets in the control flow graph. This should be used along with the option `--output-control-flow-graph` or `--output-control-flow-graph-with-basic-blocks`. |
| `--print-instruction-encoding` | `-hex` | When specified, print the encoding bytes after each disassembled operation. |

| Option (long) | Option (short) | Description |
|---|---|---|
| `--print-life-ranges` | `-plr` | Print register life range information in a trailing column in the produced disassembly. Example use briefed in Register liveness information. |
| `--print-line-info` | `-g` | Annotate disassembly with source line information obtained from .debug_line section, if present. Example use briefed in Debug information. |
| `--print-line-info-inline` | `-gi` | Annotate disassembly with source line information obtained from .debug_line section along with function inlining info, if present. Example use briefed in Debug information |
| `--print-line-info-ptx` | `-gp` | 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. |
| `--separate-functions` | `-sf` | Separate the code corresponding with function symbols by some new lines to let them stand out in the printed disassembly. |
| `--version` | `-V` | Print version information on this tool. |

# Chapter 4. Instruction Set Reference

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

## 4.1. Kepler Instruction Set

The Kepler architecture (Compute Capability 3.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 condition registers
- c[X][Y] for constant memory

Table 4 lists valid instructions for the Kepler GPUs.

Table 4. Kepler Instruction Set

| Opcode | Description |
|---|---|
| **Floating Point Instructions** | |
| FFMA | FP32 Fused Multiply Add |
| FADD | FP32 Add |
| FCMP | FP32 Compare |
| FMUL | FP32 Multiply |
| FMNMX | FP32 Minimum/Maximum |
| FSWZ | FP32 Swizzle |
| FSET | FP32 Set |
| FSETP | FP32 Set Predicate |
| FCHK | FP32 Division Test |

| Opcode | Description |
|---|---|
| RRO | FP Range Reduction Operator |
| MUFU | FP Multi-Function Operator |
| DFMA | FP64 Fused Multiply Add |
| DADD | FP64 Add |
| DMUL | FP64 Multiply |
| DMNMX | FP64 Minimum/Maximum |
| DSET | FP64 Set |
| DSETP | FP64 Set Predicate |
| **Integer Instructions** | |
| IMAD | Integer Multiply Add |
| IMADSP | Integer Extract Multiply Add |
| IMUL | Integer Multiply |
| IADD | Integer Add |
| ISCADD | Integer Scaled Add |
| ISAD | Integer Sum Of Abs Diff |
| IMNMX | Integer Minimum/Maximum |
| BFE | Integer Bit Field Extract |
| BFI | Integer Bit Field Insert |
| SHR | Integer Shift Right |
| SHL | Integer Shift Left |
| SHF | Integer Funnel Shift |
| LOP | Integer Logic Op |
| FLO | Integer Find Leading One |
| ISET | Integer Set |
| ISETP | Integer Set Predicate |
| ICMP | Integer Compare and Select |
| POPC | Population Count |
| **Conversion Instructions** | |
| F2F | Float to Float |
| F2I | Float to Integer |
| I2F | Integer to Float |
| I2I | Integer to Integer |
| **Movement Instructions** | |
| MOV | Move |
| SEL | Conditional Select/Move |
| PRMT | Permute |
| SHFL | Warp Shuffle |
| **Predicate/CC Instructions** | |

| Opcode | Description |
|---|---|
| P2R | Predicate to Register |
| R2P | Register to Predicate |
| CSET | CC Set |
| CSETP | CC Set Predicate |
| PSET | Predicate Set |
| PSETP | Predicate Set Predicate |
| **Texture Instructions** | |
| TEX | Texture Fetch |
| TLD | Texture Load |
| TLD4 | Texture Load 4 Texels |
| TXQ | Texture Query |
| **Compute Load/Store Instructions** | |
| LDC | Load from Constant |
| LD | Load from Memory |
| LDG | Non-coherent Global Memory Load |
| LDL | Load from Local Memory |
| LDS | Load from Shared Memory |
| LDSLK | Load from Shared Memory and Lock |
| ST | Store to Memory |
| STL | Store to Local Memory |
| STS | Store to Shared Memory |
| STSCUL | Store to Shared Memory Conditionally and Unlock |
| ATOM | Atomic Memory Operation |
| RED | Atomic Memory Reduction Operation |
| CCTL | Cache Control |
| CCTLL | Cache Control (Local) |
| MEMBAR | Memory Barrier |
| **Surface Memory Instructions** | |
| SUCLAMP | Surface Clamp |
| SUBFM | Surface Bit Field Merge |
| SUEAU | Surface Effective Address |
| SULDGA | Surface Load Generic Address |
| SUSTGA | Surface Store Generic Address |
| **Control Instructions** | |
| BRA | Branch to Relative Address |
| BRX | Branch to Relative Indexed Address |
| JMP | Jump to Absolute Address |
| JMX | Jump to Absolute Indexed Address |

| Opcode | Description |
|---|---|
| CAL | Call to Relative Address |
| JCAL | Call to Absolute Address |
| RET | Return from Call |
| BRK | Break from Loop |
| CONT | Continue in Loop |
| SSY | Set Sync Relative Address |
| PBK | Pre-Break Relative Address |
| PCNT | Pre-Continue Relative Address |
| PRET | Pre-Return Relative Address |
| BPT | Breakpoint/Trap |
| EXIT | Exit Program |
| **Miscellaneous Instructions** | |
| NOP | No Operation |
| S2R | Special Register to Register |
| B2R | Barrier to Register |
| BAR | Barrier Synchronization |
| VOTE | Query condition across threads |

# 4.2. 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 5 lists valid instructions for the Maxwell and Pascal GPUs.

Table 5. Maxwell and Pascal Instruction Set

| Opcode | Description |
|---|---|
| **Floating Point Instructions** | |
| FADD | FP32 Add |

| Opcode | Description |
|---|---|
| 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 |
| 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 |
| **Integer Instructions** | |
| 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 |
| ISET | Integer Compare And Set |
| ISETP | Integer Compare And Set Predicate |
| LEA | Compute Effective Address |
| LOP | Logic Operation |
| LOP3 | 3-input Logic Operation |

| Opcode | Description |
|---|---|
| POPC | Population count |
| SHF | Funnel Shift |
| SHL | Shift Left |
| SHR | Shift Right |
| XMAD | Integer Short Multiply Add |
| **Conversion Instructions** | |
| F2F | Floating Point To Floating Point Conversion |
| F2I | Floating Point To Integer Conversion |
| I2F | Integer To Floating Point Conversion |
| I2I | Integer To Integer Conversion |
| **Movement Instructions** | |
| MOV | Move |
| PRMT | Permute Register Pair |
| SEL | Select Source with Predicate |
| SHFL | Warp Wide Register Shuffle |
| **Predicate/CC Instructions** | |
| 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 |
| **Texture Instructions** | |
| 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 |
| **Compute Load/Store Instructions** | |
| LD | Load from generic Memory |
| LDC | Load Constant |
| 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 |

| Opcode | Description |
|---|---|
| STL | Store within Local or Shared Window |
| STS | Store within Local or Shared Window |
| 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 |
| CCTLT | Texture Cache Control |
| **Surface Memory Instructions** | |
| SUATOM | Atomic Op on Surface Memory |
| SULD | Surface Load |
| SURED | Reduction Op on Surface Memory |
| SUST | Surface Store |
| **Control Instructions** | |
| 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 |
| CONT | Continue |
| PCNT | Pre-continue |
| EXIT | Exit Program |
| PEXIT | Pre-Exit |
| BPT | BreakPoint/Trap |
| **Miscellaneous Instructions** | |
| NOP | No Operation |
| CS2R | Move Special Register to Register |
| S2R | Move Special Register to Register |
| B2R | Move Barrier To Register |
| BAR | Barrier Synchronization |

| Opcode | Description |
|---|---|
| R2B | Move Register to Barrier |
| VOTE | Vote Across SIMD Thread Group |

## 4.3. 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 6 lists valid instructions for the Volta GPUs.

Table 6. Volta Instruction Set

| Opcode | Description |
|---|---|
| **Floating Point Instructions** | |
| 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 |
| 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 |

| Opcode | Description |
|---|---|
| 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 |
| **Integer Instructions** | |
| 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 |
| 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 |
| **Conversion Instructions** | |

| Opcode | Description |
|---|---|
| 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 |
| **Movement Instructions** | |
| MOV | Move |
| MOV32I | Move |
| PRMT | Permute Register Pair |
| SEL | Select Source with Predicate |
| SGXT | Sign Extend |
| SHFL | Warp Wide Register Shuffle |
| **Predicate Instructions** | |
| PLOP3 | Predicate Logic Operation |
| PSETP | Combine Predicates and Set Predicate |
| P2R | Move Predicate Register To Register |
| R2P | Move Register To Predicate Register |
| **Load/Store Instructions** | |
| 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 within Local or Shared Window |
| STS | Store within Local or Shared Window |
| 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 |

| Opcode | Description |
|---|---|
| CCTLT | Texture Cache Control |
| **Texture Instructions** | |
| TEX | Texture Fetch |
| TLD | Texture Load |
| TLD4 | Texture Load 4 |
| TMML | Texture MipMap Level |
| TXD | Texture Fetch With Derivatives |
| TXQ | Texture Query |
| **Surface Instructions** | |
| SUATOM | Atomic Op on Surface Memory |
| SULD | Surface Load |
| SURED | Reduction Op on Surface Memory |
| SUST | Surface Store |
| **Control Instructions** | |
| 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 |
| **Miscellaneous Instructions** | |
| B2R | Move Barrier To Register |
| BAR | Barrier Synchronization |
| CS2R | Move Special Register to Register |
| DEPBAR | Dependency Barrier |
| GETLMEMBASE | Get Local Memory Base Address |

| 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 |

# 4.4. Turing Instruction Set

The Turing architecture (Compute Capability 7.5) 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
- c[X][Y] for constant memory

Table 7 lists valid instructions for the Turing GPUs.

Table 7. Turing Instruction Set

| Opcode | Description |
|---|---|
| **Floating Point Instructions** | |
| 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 |
| FSEL | Floating Point Select |
| FSET | FP32 Compare And Set |

| Opcode | Description |
|---|---|
| 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 |
| **Integer Instructions** | |
| 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 |
| 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 |

| Opcode | Description |
|---|---|
| LOP32I | Logic Operation |
| POPC | Population count |
| SHF | Funnel Shift |
| SHL | Shift Left |
| SHR | Shift Right |
| VABSDIFF | Absolute Difference |
| VABSDIFF4 | Absolute Difference |
| **Conversion Instructions** | |
| 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 |
| **Movement Instructions** | |
| 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 |
| **Predicate Instructions** | |
| PLOP3 | Predicate Logic Operation |
| PSETP | Combine Predicates and Set Predicate |
| P2R | Move Predicate Register To Register |
| R2P | Move Register To Predicate Register |
| **Load/Store Instructions** | |
| 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 within Local or Shared Window |
| STS | Store within Local or Shared Window |

| Opcode | Description |
| --- | --- |
| 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 |
| **Uniform Datapath Instructions** | |
| 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 |
| 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 |

| Opcode | Description |
|---|---|
| USHR | Uniform Right Shift |
| VOTEU | Voting across SIMD Thread Group with Results in Uniform Destination |
| **Texture Instructions** | |
| TEX | Texture Fetch |
| TLD | Texture Load |
| TLD4 | Texture Load 4 |
| TMML | Texture MipMap Level |
| TXD | Texture Fetch With Derivatives |
| TXQ | Texture Query |
| **Surface Instructions** | |
| SUATOM | Atomic Op on Surface Memory |
| SULD | Surface Load |
| SURED | Reduction Op on Surface Memory |
| SUST | Surface Store |
| **Control Instructions** | |
| 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 |
| **Miscellaneous Instructions** | |
| B2R | Move Barrier To Register |

| 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 |
| 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 |

## 4.5. Ampere Instruction Set

The Ampere architecture (Compute Capability 8.0and 8.6) 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
- c[X][Y] for constant memory

Table 8 lists valid instructions for the Ampere GPUs.

### Table 8. Ampere Instruction Set

| Opcode | Description |
|---|---|
| **Floating Point Instructions** | |
| FADD | FP32 Add |
| FADD32I | FP32 Add |
| FCHK | Floating-point Range Check |
| FFMA32I | FP32 Fused Multiply and Add |
| FFMA | FP32 Fused Multiply and Add |

| Opcode | Description |
|---|---|
| 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 |
| **Integer Instructions** | |
| 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 |
| IMNMX | Integer Minimum/Maximum |
| IMUL | Integer Multiply |

| Opcode | Description |
|---|---|
| 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 |
| **Conversion Instructions** | |
| 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 |
| **Movement Instructions** | |
| 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 |
| **Predicate Instructions** | |
| PLOP3 | Predicate Logic Operation |
| PSETP | Combine Predicates and Set Predicate |
| P2R | Move Predicate Register To Register |
| R2P | Move Register To Predicate Register |
| **Load/Store Instructions** | |
| LD | Load from generic Memory |

| Opcode | Description |
|---|---|
| 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 within Local or Shared Window |
| STS | Store within Local or Shared Window |
| 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 |
| **Uniform Datapath Instructions** | |
| 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 |

| Opcode | Description |
|---|---|
| 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 |
| **Texture Instructions** | |
| TEX | Texture Fetch |
| TLD | Texture Load |
| TLD4 | Texture Load 4 |
| TMML | Texture MipMap Level |
| TXD | Texture Fetch With Derivatives |
| TXQ | Texture Query |
| **Surface Instructions** | |
| SUATOM | Atomic Op on Surface Memory |
| SULD | Surface Load |
| SUQUERY | Surface Query |
| SURED | Reduction Op on Surface Memory |
| SUST | Surface Store |
| **Control Instructions** | |
| 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 |

| Opcode | Description |
|---|---|
| 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 |
| **Miscellaneous Instructions** | |
| 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 |
| 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 |

# Chapter 5. 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.

## 5.1. Usage

cu++filt accepts one or more alphanumeric words (consisting of letters, digits, underscores, dollars, or periods) and attepts 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 _Z1fIiEbl
bool f<int>(long)
```

As shown in the output, the symbol _Z1fIiEbl was successfully demangled.

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

```
$ cu++filt -p _Z1fIiEbl
f<int>
```

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

```
$ cu++filt -_ __Z1fIiEbl
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
```

# 5.2. Command-line Options

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

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

| Option | Description |
|---|---|
| `-_` | Strip underscore. On some systems, the CUDA compiler puts an underscore in front of every name. This option removes the initial underscore. Whether cu++filt 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 cu++filt and exit. |
| `-v` | Print the version information of this tool. |

## 5.3. Library Availability

`cu++filt` is also available as a static library (libcufilt) that can be linked against an existing project. The following interface describes it's 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 6. nvprune

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

## 6.1. 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.

## 6.2. 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 10. nvprune Command-line Options

| Option (long) | Option (short) | Description |
|---|---|---|
| `--arch <gpu architecture name>,...` | `-arch` | Specify the name of the NVIDIA GPU architecture which will remain in the object or library. |
| `--generate-code` | `-gencode` | This option is same format as nvcc `--generate-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`'. |
| `--no-relocatable-elf` | `-no-relocatable-elf` | Don't keep any relocatable ELF. |
| `--output-file` | `-o` | Specify name and location of the output file. |
| `--help` | `-h` | Print this help information on this tool. |
| `--options-file <file>,...` | `-optf` | Include command line options from specified file. |
| `--version` | `-V` | Print version information on this tool. |