FLexGripPlus GPGPU
Programmer's Manual

Operational codes – SASS assembly language SM_1.0

Operational codes – SASS assembly language SM_1.0

Authors:
Josie Esteban Rodriguez Condia
Boyang Du
Gianluca Roascio
Eduard Sci
Juan David Guerrero Balaguera

Contact:
josie.rodriguez@polito.it
boyang.du@polito.it
gianluca.roascio@polito.it
edouard.scie@grenoble-inp.org
juandavid.guerrero@uptc.edu.co

All reported Op-codes are fully compatible with the SASS assembly language in the SM_1.0 for GPGPUs using the G80 microarchitecture. The opcodes were specifically determined to support the verification, testing and operation of the **FlexGripPlus GPGPU model**.

The manual was developed by the Electronic CAD & Reliability Group (CAD) in the Department of Control and Computer Engineering (DAUIN).
Politecnico di Torino
Italy, 2020

The Floating Point Unit (FPU) extension and op-codes were developed in collaboration between Politecnico di Torino and the Grenoble Institute of Technology.

The Special Functions Unit (SFU) extension and op-codes were developed in cooperation between Politecnico di Torino and Universidad Pedagogica y Tecnologica de Colombia (UPTC).

All Activities performed in the development of the **FlexGripPlus GPGPU** model were supported with fundings by the European Commission through the Horizon 2020 **RESCUE-ETN** project under grant 722325. For more information: http://rescue-etn.eu/

The **FlexGripPlus GPGPU** model can be downloaded from:

https://github.com/Jerc007/Open-GPGPU-FlexGrip-

CAD Group
POLITECNICO DI TORINO
Grenoble INP
Uptc Universidad Pedagógica y Tecnológica de Colombia
RESCUE-ETN

Content

Glosary:

In the description of the opcodes, the following words are employed to represent the resources in the GPGPU:

- **GPRS:** General Purpose Registers in the Regiter File.
- **Imm:** Immediate value stored in the instruction code.
- **Rx, Ry, Rz:** Registers
- **Ax:** Address registers
- **SRx:** special system-controlled registers
- **U:**Insigned value
- **S:** Singed value
- **Cx:** Conditional or predicate registers
- **global14 r[0xXX]:** global or main memory in the GPGPU
- **g[0xXX]:** Shared memory
- **c[0xXX][0xYY]:** Constant memory
- **local[0xXX]:** Local memory

TABLES OF SASS INSTRUCTIONS SUPPORTED IN FLEXGRIPPLUS

Table 1 Control-flow instructions supported in FlexGripPlus.

Mnemonic	Description	Formats
BRA	Branch	BRA CX.COND Imm BRA Imm
BAR	barrier synchronization	BAR.ARV.WAIT b0, 0xFFF
RET	Return from kernel	RET RET CX.COND
SSY	Set synchronization point	SSY Imm
NOP	No operation	NOP NOP.S
TRAP	Trap interruption	TRAP
CAL	Call to subroutine	CAL.NOINC CAL

Table 2 Arithmetic and logic instructions in FlexGripPlus.

Mnemonic	Description	Formats
I2I	Integer to integer conversion	I2I.U32.U16/S16 RZ, RX(L|H) / g[].U16 I2I.U32.S32 RZ, |RX| / -RX I2I.U32.U16.BEXT RZ, RX(L|H) / g[].U8 I2I.S32.S16.BEXT RZ, RX(L|H) / g[].S8
IMUL/	Integer multiplication	IMUL.U16.U16 RZ, RX(L|H) / g[].U16, RY(L|H) IMUL.S16.S16 RZ, RX(L|H) / g[].S16, RY(L|H)
IMUL32/		IMUL32.U16.U16 RZ, RX(L|H)/g[].U16, RY(L|H)
IMUL32I		IMUL32I.U16.U16 RZ, RX(L|H), Imm IMUL32I.S16.S16 RZ, RX(L|H), Imm
SHL	Shift left	SHL RZ, RX, RY / Imm SHL RZ, g [], Imm SHL.U16 RZ(L|H), RX(L|H), Imm
SHR	Shift right	SHR.S32 RZ, RX, RY / Imm SHR.S32 RZ, g [], Imm SHR.U16 / S16 RZ(L|H), RX(L|H), Imm SHR RZ, g[], Imm SHR RZ, RX, RY / Imm
IADD/	Integer add	IADD RZ, RX / -RX, RY IADD RZ, g[], RX / -RX IADD RZ, RX, c[0x1][]
IADD32/		IADD32 RZ, RX, RY / -RY IADD32 RZ, g [0x..], RX / -RX IADD32.U16 RZ(L|H), RX(L|H), RY(L|H) /-RY(L|H)
IADD32I		IADD32I RZ, RX / -RX, Imm IADD32I RZ, g[], Imm
IMAD/	Integer multiply and Add	IMAD.U16/ S16 RZ, RX(L|H), RY(L|H), RW IMAD.U16/ S16 RZ, RX(L|H), c[0x1][], RY IMAD. RZ, RX(L|H), c[0x1][], RY
IMAD32/		IMAD32.U16 RZ, RXL|H, RYL|H, RZ
IMAD32I		IMAD32I.U16/ S16 RZ, RX(L|H), Imm, RZ
LOP	Bitwise logical Operation	LOP.AND/OR/XOR/PASS_B RZ, RX/ g[], RY LOP.AND/OR/XOR/PASS_B RZ, RX, c[0x1] [] LOP.U16.AND/OR/XOR/PASS_B RZ(L|H), RX(L|H), RY(L|H)
ISET	Integer comparison	ISET RZ, RX, RY / c[0x1][], COMP_TYPE ISET RZ, g[], RX, COMP_TYPE ISET.S32 RZ, RX, RY / c[0x1][], COMP_TYPE ISET.S32 RZ, g[], RX, COMP_TYPE

Table 3 Data handling and memory instructions in FlexGripPlus.

Mnemonic	Description	Formats
MVC	Load from constant memory	MVC RX, c [0x1] []
GLD	Load from global memory	GLD.U32|U16|S16|U8|S8 RZ, global14[]
GST	Store to global Memory	GST.U32|U16|S16|U8|S8 global14[], RX
MOV/	Move register to register/load from shared memory	MOV RZ, RX / g[] MOV.U16 RZ(L|H), RX(L|H) / g[].(U16|U8)
MOV32		MOV32 RZ, RX / g[] MOV32.U16 RZ(L|H), RX(L|H)
MVI	Move immediate to destination	MVI RX, Imm
R2G	Store to shared Memory	R2G.U32.U32 g [], RX R2G.U16.U16 g [], RXL|H R2G.U16.U8 g [], RX
R2A	Move data register to address register	R2A AX, RX
A2R	Move address register to data register	A2R RX, AX
ADA	Movement from address register to address register	ADA Ax, Ax, Offset

Table 4 Floating Point Unit (FPU) instructions supported in FlexGripPlus.

Mnemonic	Description	Formats
FADD32 / FADD / FADD32I	Floating point addition	FADD32 Rx, Ry / g[Ax + Imm], Rz FADD.COND Rx (Cx), Ry, -Rz / c[0xX][Imm] FADD32I Rx, Ry, Imm
FMUL / FMUL32 / FMUL32I	Floating point multiplication	FMUL Rx(Cx.COND), Ry / g [Ax + Imm], Rz / c[0xX][Imm] FMUL32 Rx, Ry / g [Ax+Imm], Rz FMUL32I Rx, Ry, Imm
FMAD / FMAD32 / FMAD32I	Floating point multiply and addition	FMAD Rx, Ry / - g [Ax+Imm], Rz / c[0xX][Imm], Rw FMAD32I Rx, -Ry, Imm, Rz
F2F	Floating point conversion	F2F.F32.F32 Rx (CX.COND), -Ry / |Ry|
F2I	Conversion from Floating point to Integer	F2I.S32.F32.COND Rx, Ry
I2F	Conversion from Integer to Floating point	I2F.F32.S32/U32 Rx (CX.COND), Ry
FSET	Floating point set	FSET.C0 o[0x7f] (Cx.COND), Rx / |Rx|, Ry / c[0xX][Imm], COND
RCP / RCP32	Reciprocal value	RCP Ry (Cx.COND), Rx RCP32 Ry, Rx

Table 5 Special function unit (SFU) instructions supported in FlexGripPlus.

Mnemonic	Description	Formats
SIN	Single precision SIN (32 bits)	SIN Rx, Rx
COS	Single precision COS (32 bits)	COS Rx, Rx
RRO	Range Reduction Operator (phase)	RRO Ry, Rx, (SIN/EX2)
EX2	Find the base-2 exponential of a value.	EX2 Ry, Rx
RSQ	Reciprocal of the square root in single-precision (32 bits)	RSQ Ry, Rx
LG2	Calculates the Log, base 2, of a value	LG2 Ry, Rx

INSTRUCTIONS

Control-flow Instructions

BRA

BAR

RET

SSY

NOP

TRAP

CAL

BRA instruction:

Checked OK

This instruction generates a change in the warp PC in the SM. (Brach operation in the GPU)
PC ← offset address

Mnemonics:
Direct BRA (offset address) **BRA 0x1E0**
Indirect BRA (predicate_register.condition) offset_address **BRA C0.NE, 0x1e0**

Example (SASS from NVCC):

BRA 0xf0 (1001e003 00000780)
BRA C0.NE, 0xe8 (00000280 1001d003)

(SASS_assembly_lib):
Format: BRA(int **offset**, int **condition**, int **pred_reg_cond**, int **marker**)

Offset: bits (51-46) – (26-9)
Condition: (43-39)
Pred_reg_cond: (45-44)
Marker: (1-0)

Note:
The original version of this instruction (in FlexGrip) was implemented with an address limit of 18 bits. **(High part of the memory address is not implemented)** GPGPU-FLEXGRIP instruction memory is limited to 18 bits of address pointer. This condition was repaired in the extended version of the model.

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. (**by default**)
1	instr_is_flow	**0 =** Normal ins.	**1 =** System ins. (flow control) **(by default)**
(2 – 8) 7	Not used	000 0000 **(by default)**	
(9 – 26) 18	Branch Address Low Part 18 bits	The Branch Address (24bits) is divided as HB(2)-MB(8bits)-LB(8 bits) 24 down to 9	
(27) 1	Not used	0	
(28 – 31) 4	Instruction Op. Code	BRA_OP = 0x1h	
(32 - 33) 2	instr_marker	**00** normal register access(load or store) (not extra instruction) **(by default)** 01 normal register access(load or store) (with **Join**) (extra instruction) 10 normal register access(load or store) (with **Exit**) 11 immediate	
(34 – 37) 4	Not used	0000	
(38)1	modifier 1	0 **(by default)**	
(39 – 43) 5	predicate condition - selects a boolean function of the $c register		
(44 – 45) 2	Predicate_register	C0 = 00 **(by default)** C1 = 01 C2 = 10 C3 = 11	
(46-51) 6	Branch Address High Part 6 bits	The Branch Address (24bits) is divided as: HB(6) 51 down to 46 **(NOT supported by GPGPU-FLEXGRIP) 000000**	
(52 – 63)	Not Used	0000 0000 0000	

encoding	name	Description	condition formula
0x00	never	always false	0 (overflow) not used
0x01	l	less than	(S & ~Z) ^ O (working)
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater than	~Z & ~(S ^ O)
0x05	lg	less or greater than / not equal	~Z (**working**)
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z (**working**)
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

BAR instruction:

Checked

This instruction generates a barrier condition to synchronize all thread in a warp

Mnemonics:

BAR.(type)

Example (SASS from NVCC):

BAR.ARV.WAIT b0, 0xfff (00000000 861FFE03)

(SASS_assembly_lib):

Pending…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. (**by default**)
1	instr_is_flow	**0 =** Normal ins.	**1 =** System ins. (flow control) **(by default)**
(2 – 27) 26	BAR	0x61FFE0	
(28 – 31) 4	Instruction Op. Code	BAR = 0x8h	
(32 - 33) 2	instr_marker	**00** normal register access(load or store) (not extra instruction) **(by default)** 01 normal register access(load or store) (with **Join**) (extra instruction) 10 normal register access(load or store) (with **Exit**) 11 immediate	
(34 – 63) 30	Not used	00 0000 0000 0000 0000 0000 0000 0000	

RET instruction:

Checked YES

This instruction returns from a kernel execution or a thread path (*taken-not taken*) in case of divergence.

RET (Return, Employed to finish the operation of the kernel in Flexgrip)

Mnemonics:
RET
RET Cx (COND) (predicate condition)

Example (SASS from NVCC):
RET (00000780 30000003)
RET C0.NE (00000280 30000003)
RET C0.EQ (00000100 30000003)
RET C0.NE (00000280 30000003)

(SASS_assembly_lib):
Format: RET(int **condition**, int **pred_reg_cond**, int **marker**)

Condition: (43-39)
Pred_reg_cond: (45-44)
Marker: (1-0)

Note:
The original version of this instruction (in FlexGrip) was able to stop the kernel execution. The additional feature of returning from a thread path was added in the improved version FlexGrip*.

<table>
<tr><td>0</td><td>instr_is_long</td><td>0 = 32 bit long.</td><td>1 = 64 bit long. (By default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td>0 = Normal ins</td><td>1 = System ins. (flow control) (By default)</td></tr>
<tr><td>(2 – 27) 26</td><td>Not Used</td><td colspan="2">0000 0000 0000 0000 0000 0000 00</td></tr>
<tr><td>(28 – 31) 4</td><td>Instruction Op. Code</td><td colspan="2">RET = 0x3</td></tr>
<tr><td>(32 - 33) 2</td><td>instr_marker</td><td colspan="2">00 normal reg Access(load or store) (not extra instruction) (By default)
01 normal reg Access(load or store) (with Join) (extra instruction)
10 normal reg Access(load or store) (with Exit) (POP from warp stack)
11 immediate</td></tr>
<tr><td>(34 - 38) 5</td><td>Not used</td><td colspan="2">0000 0</td></tr>
<tr><td>(39 – 42) 5</td><td>predicate_condition</td><td colspan="2">
<table>
<tr><th>encoding</th><th>name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>never</td><td>always false</td><td>0</td></tr>
<tr><td>0x01</td><td>l</td><td>less than</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>e</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>g</td><td>greater tan</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>lg</td><td>less or greater tan</td><td>~Z</td></tr>
<tr><td>0x06</td><td>ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>u</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>leu</td><td>not greater tan</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>o</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>c</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>a</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>s</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>no</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(45-44)</td><td>Predicate_register</td><td colspan="2">C0 = 00 (by default)
C1 = 01
C2 = 10
C3 = 11</td></tr>
<tr><td>(46 - 63)</td><td>Not used</td><td colspan="2">0000 0000 0000 0000 000</td></tr>
</table>

SSY instruction:

Checked YES

This instruction defines the convergence point for a potential divergence generation program kernels. This instruction activates the divergence stack module in the GPGPU.

SSY (Returning Address)
Example:

Mnemonics:
SSY 0xd88

Example (SASS from NVCC):
SSY 0xe8 (00000000 a001d003)

(SASS_assembly_lib):

SSY(int **offset**, int **condition**, int **pred_reg_cond**, int **marker**) (The predicate condition is not employed in this instruction, but is present in the code and function description)
Offset: (24-9)
Condition: (43-39)
Pred_reg_cond: (45-44)
Marker: (1-0)

<table>
<tr><th>Bit(s)</th><th>Mnemonics</th><th colspan="2">Commentary</th></tr>
<tr><td>0</td><td>instr_is_long</td><td>0 = 32 bit long.</td><td>1 = 64 bit long. (by default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td>0 = Normal ins.</td><td>1 = Flow-control System ins. (by default)</td></tr>
<tr><td>(2 – 8) 7</td><td>Not used</td><td colspan="2">0000 000</td></tr>
<tr><td>(9 – 24) 16</td><td>Offset Address LP</td><td colspan="2">Code address divided as: HP (6 bits) – LP (16 bits)</td></tr>
<tr><td>(25 - 27) 3</td><td>Not used</td><td colspan="2">000</td></tr>
<tr><td>(28 – 31) 4</td><td>Instruction Op. Code</td><td colspan="2">SSY_OP = 0xA</td></tr>
<tr><td>(32 - 33) 2</td><td>instr_marker</td><td colspan="2">00 normal reg Access(load or store) (not extra instruction) (by default)
01 normal reg Access(load or store) (with Join) (extra instruction)
10 normal reg Access(load or store) (with Exit)
11 immediate</td></tr>
<tr><td>(34 – 38) 5</td><td>Not used</td><td colspan="2">00000</td></tr>
<tr><td>(39 – 43) 5</td><td>predicate condition for the associated $c Predicate register</td><td colspan="2">
<table>
<tr><th>encoding</th><th>name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>never</td><td>always false</td><td>0</td></tr>
<tr><td>0x01</td><td>l</td><td>less tan</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>e</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>g</td><td>greater tan</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>lg</td><td>less or greater tan</td><td>~Z</td></tr>
<tr><td>0x06</td><td>ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>u</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>leu</td><td>not greater than</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>o</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>c</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>a</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>s</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>no</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(44 - 45)</td><td>source $c register, not used in this ins.</td><td colspan="2">00</td></tr>
<tr><td>(46 -51) 6</td><td>Offset Address HP</td><td colspan="2">High part (Not implemented in the original version of FLEX-GRIP or extended one)
(not implemented in SASS_assembly_lib)</td></tr>
<tr><td>(52 – 63)12</td><td>Not used</td><td colspan="2">0000 0000 0000</td></tr>
</table>

NOP instruction:

Checked YES

Not operation instruction or bypass instruction

Mnemonics:
NOP
NOP.S (predicate condition)

Example (SASS from NVCC):
NOP (E0000001 F0000001)
NOP.S (E0000002 F0000001)

(SASS_assembly_lib):
Format: RET(int **condition**, int **pred_reg_cond**, int **marker**)

Condition: (43-39)
Pred_reg_cond: (45-44)
Marker: (1-0)

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. (**By default**)
1	instr_is_flow	**0 =** Normal ins. (**By default**)	**1 =** System ins. (flow control)
(2 – 27) 26	Not Used	0000 0000 0000 0000 0000 0000 00	
(28 – 31) 4	Instruction Op. Code	NOP = 0xF	
(32 - 33) 2	instr_marker	**00** normal reg Access(load or store) (not extra instruction) **(by default)** 01 normal reg Access(load or store) (with **Join**) (extra instruction) (NOP) 10 normal reg Access(load or store) (with **Exit**) (POP from warp stack) (NOP.S) 11 immediate	
(34 – 60) 26	Not Used	0000 0000 0000 0000 0000 0000 00	
(61 - 63) 4	Sub_operand_type	111	

TRAP instruction:

Checked NO, Implemented pending.

Trap interruption to host.

Mnemonics:
TRAP

Example (SASS from NVCC):
TRAP (00000000 90000003)

(SASS_assembly_lib):
Pending…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0** = 32 bit long.	**1** = 64 bit long. **(By default)**
1	instr_is_flow	**0** = Normal ins.	**1** = System ins. **(By default)**
(2 – 27) 26	Not Used	0000 0000 0000 0000 0000 0000 00	
(28 – 31) 4	Instruction Op. Code	TRAP = 0x9	
(32 - 33) 2	instr_marker	**00** normal reg Access(load or store) (not extra instruction) **(by default)** 01 normal reg Access(load or store) (with **Join**) (extra instruction) 10 normal reg Access(load or store) (with **Exit**) (POP from warp stack) 11 immediate	
(34 – 60) 26	Not Used	0000 0000 0000 0000 0000 0000 00	
(61 - 63) 4	Sub_operand_type	000	

CAL instruction:

Checked YES

Cal to the subroutine without context switch or parameters

CAL: PC <- PC(CAL)
Mnemonics:
CAL.(type) (address of the subroutine)

Example (SASS from NVCC):
CAL.NOINC 0xF0 (00000000 2001E003)

(SASS_assembly_lib):
Pending…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0** = 32 bit long.	**1** = 64 bit long. **(By default)**
1	instr_is_flow	**0** = Normal ins.	**1** = System ins. **(By default)**
(2 – 8) 7	Not Used	000 0000	
(9 – 27) 19	Initial address of the subroutine	From 0x00000 to 0x3FFFF	
(28 – 31) 4	Instruction Op. Code	CAL = 0x2	
(32 - 33) 2	instr_marker	**00** normal reg Access(load or store) (not extra instruction) **(by default)** 01 normal reg Access(load or store) (with **Join**) (extra instruction) 10 normal reg Access(load or store) (with **Exit**) (POP from warp stack) 11 immediate	
(34 – 60) 26	Not Used	0000 0000 0000 0000 0000 0000 00	
(61 - 63) 4	Sub_operand_type	000	

Arithmetic and logic instructions

I2I

IMUL

IMUL32

IMUL32I

SHL

SHR

IADD

IADD32

IADD32I

IMAD

IMAD32

IMAD32I

LOP

ISET

I2I instruction (CVT):

Checked YES

This instruction performs the conversion of formats among integer values. It should be noted that this instruction is only available for integer operands.

Mnemonics:
I2I.(destiny operand format).(source operand format) (destiny location of the operand),(source location of the operand)

The formats in destiny and source may be signed (**S**), unsigned (**U**), and 8, 16, and 32 bits wide. The sources and destinies may be registers, shared memory, constant memory, or global memory locations.

Example (SASS from NVCC):
I2I.U32.U16 R0, R0L (04000780 a0000001)
I2I.U32.U16 R1, g [0x1].U16 (04200780 a0004205)
I2I.S32.S32 R1, -R1 (2c014780 a0000205)

(SASS_assembly_lib):
Formats:

I2I_32_16(int **dest_reg**, int **source_reg_1**, char **hilo_1**, char **sigd**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
I2I_U32_S32_abs2(int **dest_reg**, int **source_reg_1**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
I2I_S32_S32_neg2(int **dest_reg**, int **source_reg_1**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
I2I_32_16_shmem(int **dest_reg**, int **addr_reg**, int **offset**, char **sigd**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
I2I_32_32_o0x7f(int **source_reg_1**, char **sigd**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
I2I_32_16_BEXT(int **dest_reg**, int **source_reg_1**, char **hilo_1**, char **sigd**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
I2I_32_16_BEXT_shmem(int **dest_reg**, int **addr_reg**, int **offset**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
I2I_16_16_BEXT_shmem(int **dest_reg**, char **hilo_d**, int **addr_reg**, int **offset**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)

dest_reg: (8-2)
hilo_1: (9)
condition: (39-43)
Pred_reg_cond: (45-44)
set_pred: (38)
pred_reg_set: (38-37)
marker: (1-0)
source_reg_1: (10-15) or (16-22) depending on the function
sigd: (48) or (59)
addr_reg: (26-27) and (34-37) (address register Ax)
hilo_d: (2)
offset: (9-13)

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. (**by default**)
1	instr_is_flow	**0 =** Normal ins. (**by default**)	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_Register_or_memory_space_address	R0= 000000, R5=000101, R6=000110…	Address: (condition code register??) 0x00 to 0x7F {**o [0x7f]**}
9	Source location (High-Low)(source of 16 bit-size case)	1 = High part (RxH)	0 = Low part (RxL)
(10 - 15) 6	Source_1 (register)	R0= 000000, R5=000101, R6=000110…	
(16 – 22) 6	1_Register_Operand (16 bits, 32 bits)	R0= 000000, R5=000101, R6=000110…	
(23) 1	Not used	0	
(24) 1	Not used	0	
(25) 1	Not used	0	
(26 – 27) 2	Address register low 2 bits	00	
(28 – 31) 4	Instruction Op. Code	I2I = 0xA	
(32 - 33)2	instr_marker	00 = normal reg Access(load or store) (not extra instruction) **(by default)** 01 = normal reg Access(load or store) (with **Join**) (extra instruction) 10 = normal reg Access(load or store) (with **Exit**) 11 = immediate	
(34)1	Not used	0	
(35)1	destination type	0 = Register destination	1= Memory destination
(36-37)2	Predicate register set (enabling a new flag)	C0 = 00 **(by default)** C1 = 01 C2 = 10 C3 = 11	
(38)1	Write enable / set predicate register	**1** = write enabled (just for memory destination) **1** = enable predicate register set, **0** = disable predicate register set	
(39 – 43)	predicate_condition		

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

<table>
<tr><td>(45 - 44) 2</td><td>Predicate_register</td><td colspan="3">C0 = 00 (by default)
C1 = 01
C2 = 10
C3 = 11</td></tr>
<tr><td>(45) 1</td><td>Not used</td><td colspan="3">0</td></tr>
<tr><td>(46 - 48) 3</td><td>Source_1_data_type</td><td colspan="3"><table>
<tr><th>Source_1_data_type</th><th>Mem Type</th><th>I2I type</th></tr>
<tr><td>000</td><td>DT_U8</td><td>DT_U16</td></tr>
<tr><td>001</td><td>DT_U16</td><td>DT_U32</td></tr>
<tr><td>010</td><td>DT_S16</td><td>DT_U8</td></tr>
<tr><td>011</td><td>DT_U32</td><td>DT_U32</td></tr>
<tr><td>100</td><td>DT_U32</td><td>DT_S16</td></tr>
<tr><td>101</td><td>DT_U32</td><td>DT_S32</td></tr>
<tr><td>110</td><td>DT_U32</td><td>DT_S8</td></tr>
<tr><td>111</td><td>DT_U32</td><td>DT_U32</td></tr>
</table></td></tr>
<tr><td>49 – 57</td><td>Not used</td><td colspan="3">0000 0000</td></tr>
<tr><td>58</td><td>Not used</td><td colspan="3">0</td></tr>
<tr><td>(59) 1</td><td>Size of operands</td><td>0: b16</td><td>1: b32</td></tr>
<tr><td>(60) 1</td><td>Signed or unsigned sources (potentially not used)</td><td>0: U16/U32 (Unsigned)</td><td>1: S16/S32 (Signed)</td></tr>
<tr><td>(61 – 63)3</td><td>Sub_op_code</td><td colspan="3">000 (not used)</td></tr>
</table>

IMUL Instruction:

Checked YES

This instruction performs the integer multiplication of two operands. The sources can be registers or constant memory locations. The operation of these instructions could be dependable on predicate conditions.

Mnemonics:
IMUL.(destiny operand format).(source operand format) (destiny location of the operand), (source 1), (source 2)

The formats in destiny and source may be signed (**S**), unsigned (**U**) in 16, and 32 bits wide. The sources and destinies may be registers, shared memory, constant memory, or global memory locations.

Example (SASS from NVCC):
IMUL.U16.U16 R2, R2L, R1L (00000780 40020809)
IMUL.U16.U16 R4, R1L, R3H (00000780 40070411)

(SASS_assembly_lib):
Formats:
IMUL_U16_U16_shmem(int **dest_reg**, int **addr_reg**, int **offset**, int **source_reg_2**, char **hilo_2**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
IMUL_S16_S16_regs(int **dest_reg**, int **source_reg_1**, char **hilo_1**, int **source_reg_2**, char **hilo_2**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
IMUL_S16_S16_shmem(int **dest_reg**, int **addr_reg**, int **offset**, int **source_reg_2**, int **hilo_2**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
IMUL_U16_U16_regs(int **dest_reg**, int **source_reg_1**, char **hilo_1**, int **source_reg_2**, char **hilo_2**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)

Pending the description…

Bit(s)	Mnemonics	Commentary
0	instr_is_long	**0 =** 32 bit long. **1 =** 64 bit long. **(By default)**
1	instr_is_flow	**0 =** Normal ins. **(By default)** **1 =** System ins. (flow control)
(2 – 8) 7	Destiny_ Register	**R0**= 000000, **R5**=000101, **R6**=000110…
(9-15) 6	Source_Register_1: It could be a GPRS at 32 or 16 bits, or a shared memory location	**32 bits**: **R0**= 0000000 **R5**= 0000101 **R6**= 0000110 **16 bits:** **R0[L]** = 000000 0 **R5[H]**= 000010 1 **R6[H]**= 000011 0 **Shared memory case:** Offset value (9 - 12) 1 (13) 0 (14) 0 (15)
16	High-Low part Register 2 Operand	**0 =** low part (RxL) **1 =** High part (RxH)
(17 – 22) 6	2_ Register_or_shared_mem _Operand (16 bits)	**R0** = 000000, **R5**=000101, **R6**=000110…
23	Not used	**0**
24	Source_1_Selector	**0 =** Register Source **1 =** Shared Mem.
25	Not used	**0**
(26-27) 2	Address register offset used by the shared memory addressing	**A0 =** 00 **(default)** **A1 =** 01 **A2 =** 10 **A3 =** 11
28	Size of operands	**0 =** 16 or **1 =** 32 bits operands
(29 – 31) 3	Instruction Op. Code	IMUL32 = 0x2
(32 - 33)2	instr_marker	00 = normal reg Access(load or store) (not extra instruction) **(by default)** 01 = normal reg Access(load or store) (with **Join**) (extra instruction) 10 = normal reg Access(load or store) (with **Exit**) 11 = immediate
(34)1	Not used	0
(35)1	destination type	**0 =** Register destination **1 =** Internal operation
(36-37)2	Predicate register set (enabling a new flag) or Not used	**C0** = 00 **(by default)** **C1** = 01 **C2** = 10 **C3** = 11
(38)1	Set predicate register	**1** = enabled predicate register set **0** = disabled predicate register set
(39 – 43) 5	predicate_condition	
(44 - 45) 2	Input predicate register to compare before to operate	**C0=** 00 **C1 =** 01 **C2=** 10 **C3 =** 11
(46 – 52) 7	Not used	000 0000
53	Shared memory use for Source_2?	**Yes =** 1 **No =** 0
(54 – 60) 7	Not used	000 0000
(61 – 63)3	Sub_op_code	000

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

IMUL32 Instruction:

Checked YES

This instruction performs the integer multiplication of two operands.

Mnemonics:
IMUL32.(destiny operand format).(source operand format) (destiny location of the operand), (source 1), (source 2)

The formats in destiny and source may be signed (**S**), unsigned (**U**) in 16, and 32 bits wide. The sources and destinies may be registers, shared memory, constant memory, or global memory locations.

Example (SASS from NVCC):
IMUL32.U16.U16 R8, R6H, R1L (40021A20)
IMUL32.U16.U16 R11, R6H, R2L (40041A2C)
IMUL32.U24.U24 R1, R1, R0 (40400204)

(SASS_assembly_lib):
Formats:
IMUL32_U16_U16_regs(int **dest_reg**, int **source_reg_1**, char **hilo_1**, int **source_reg_2**, char **hilo_2**)
IMUL32_U16_U16_shmem(int **dest_reg**, int **addr_reg**, int **offset**, int **source_reg_2**, char **hilo_2**)

Pending the description…

Bit(s)	Mnemonics	Commentary		
0	instr_is_long	**0 =** 32 bit long. **(By default)**	**1 =** 64 bit long.	
1	instr_is_flow	**0 =** Normal ins. **(By default)**	**1 =** System ins. (flow control)	
(2 – 8) 7	Destiny_ Register	**R0**= 000000, **R5**=000101, **R6**=000110…		
(9-15) 6	Source_Register_1: It could be a GPRS at 32 or 16 bits, or a shared memory location	**32 bits**: **R0**= 0000000 **R5**= 0000101 **R6**= 0000110	**16 bits:** **R0[L]** = 000000 0 **R5[H]**= 000010 1 **R6[H]**= 000011 0	**Shared memory case:** Offset value (9 - 12) 1 (13) 0 (14) 0 (15)
16	High-Low part Register 2 Operand	**0 =** low part (RxL)	**1 =** High part (RxH)	
(17 – 21) 5	2_ Register_or_shared_mem _Operand (16 bits)	**R0** = 000000, **R5**=000101, **R6**=000110…		
22	Operand size	**0 =** 16 bits	**1 =** 24 bits	
23	Not used	**0**		
24	Source_1_Selector	**0 =** Register Source	**1 =** Shared Mem.	
25	Not used	**0**		
(26-27) 2	Address register offset used by the shared memory addressing	**A0 =** 00 **(default)** **A1 =** 01 **A2 =** 10 **A3 =** 11		
(28 – 31) 4	Instruction Op. Code	IMUL32 = 0x4		

IMUL32I Instruction:

Checked YES

This instruction performs the integer multiplication of two operands using an immediate operand. The sources can be registers.

Mnemonics:
IMUL32I.(destiny operand format).(source operand format) (destiny location of the operand), (source 1), (Imm)

The formats in destiny and source may be signed (**S**), unsigned (**U**) in 16, and 32 bits wide. The sources and destinies may be registers, shared memory, constant memory, or global memory locations.

Example (SASS from NVCC):

(SASS_assembly_lib):
Formats:
IMUL_U16_U16_shmem(int **dest_reg**, int **addr_reg**, int **offset**, int **source_reg_2**, char **hilo_2**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
IMUL_S16_S16_regs(int **dest_reg**, int **source_reg_1**, char **hilo_1**, int **source_reg_2**, char **hilo_2**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
IMUL_S16_S16_shmem(int **dest_reg**, int **addr_reg**, int **offset**, int **source_reg_2**, int **hilo_2**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)
IMUL_U16_U16_regs(int **dest_reg**, int **source_reg_1**, char **hilo_1**, int **source_reg_2**, char **hilo_2**, int **condition**, int **pred_reg_cond**, char **set_pred**, int **pred_reg_set**, int **marker**)

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. **(By default)**
1	instr_is_flow	**0 =** Normal ins. **(By default)**	**1 =** System ins. (flow control)
(2 – 7) 6	Destiny_ Register	**R0** = 000000, **R5**=000101, **R6**=000110…	
8	Sign of destiny reg	**0** = Unsigned	**1** = Signed
(9-15) 6	Source_Register_1: It could be a GPRS at 32 or 16 bits	**32 bits**: **R0**= 0000000 **R5**= 0000101 **R6**= 0000110	**16 bits:** **R0[L]** = 000000 0 **R5[H]**= 000010 1 **R6[H]**= 000011 0
(16 – 21) 6	Source 2: Imm operand low part	XX XXXX	
(22-27) 6	Not used	**0000**	
(28 – 31) 4	Instruction Op. Code	IMUL32 = 0x4	
(32 - 33)2	instr_marker	**00** = normal reg Access(load or store) (not extra instruction) **01** = normal reg Access(load or store) (with **Join**) (extra instruction) **10** = normal reg Access(load or store) (with **Exit**) **11** = immediate **(by default)**	
(34 - 59) 26	Source 2: High part of the immediate value of 32 bits	XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX … XX	
(60) 1	Not used	0	
(61 – 63)3	Sub_op_code	000	

SHL/SHR Instructions:

Checked YES

These instructions perform the logic shift operations (Left or Right) into operands of 16 or 32 bits size. The sources can be registers or constant memory locations. The operation of these instructions could be dependable on predicate conditions.

Pre: Rz <- Rx **<<** Ry
Pre: Rz <- Rx **>>** Ry

Mnemonics:
(Predicate) LOP. (Logic Operation).(Size) Destiny, Source_1, Source_2

Destiny and source registers are 16 or 32-bit size. The source_1 can be a register or a shared memory location. The source_2 can be a constant memory location.

Example (SASS from NVCC):
SHL R5, R1, R0 (C4000780 30000215)
SHL R1, R3, 0x4 (C4100780 30040605)
SHL R2, R3, 0x5 (C4100780 30050609)
SHL R6, R4, 0x1 (C4100780 30010819)
SHL R0 (C0.EQU), R0, 0x2 (C4100500 30020001)
SHR.S32 R1, R1, 0x1 (EC100780 30010205)
SHR.S32 R2, R2, 0x1 (EC100780 30010409)
SHR.S32 R2, R2, 0x10 (EC100780 30100409)
SHR.U16 R1H, R0H, 0xA (E0100780 300A020D)

(SASS_assembly_lib):
Formats:
Pending

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. **(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny Register (32 bits) or H/L + Destiny reg (16 bits)	**R0** = 0000000 **R5** = 0000101 **R6** = 0000110	**R0(H)** = 000000 1 , **H** = high **R5(H)** =000101 1 , **L** = low **R6 (L)** = 000110 0
(9 – 15)7	Source 1 Register (32 bits) or H/L + Source 1 reg (16 bits)	**R0** = 0000000 **R5** = 0000101 **R6** = 0000110	**R1(H)** = 000001 1 **R5(H)** =000101 1 **R6 (L)** = 000110 0
(16 – 20) 5	Offset_of shift	This can be a register or an immediate value in Hex.	
(24) 1	Source 1 Selector	1 = Shared memory op.	0 = Register operand
(25-27) 3	Not Used	000	
(28-31) 4	Instruction Op. Code	SHL or SHR = 0x3	
(32 - 33)2	instr_marker	00 = normal reg Access(load or store) (not extra instruction) **(by default)** 01 = normal reg Access(load or store) (with **Join**) (extra instruction) 10 = normal reg Access(load or store) (with **Exit**) 11 = immediate	
34	Used for….	0	
35	destination type	0 = Register destination	
(36 – 38) 3	Nor used	000	
(39 – 43) 5	predicate condition to operate the instruction	(see table below)	
(44 - 45) 2	Input_predicate_register Used as: precondition to operate the instruction	**C0=** 00 **C2=** 10	**C1=** 01 **C3=** 11
(46-51) 6	Not used	00 0000	
52	Source 2 selector	1 = Immediate value	0 = Register
(53-57) 5	Not used	0 0000	
58	Size of destiny and source	**1** = 32 bits	**0** = 16 bits
59	Use of the Sign during the shift	**1** = Signed	**0** = Unsigned
60	Not Used	0	
61	Operation code of the shift (Left or Right)	**0** = SHL	**1** = SHR
(62 – 63) 2	Sub_opcode	11	

encoding	name	Description	condition formula
0x00	never	always false	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater tan	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

IADD Instruction:

Checked YES

This instruction performs the Integer addition in (32 or 16 bits) of two sources. The sources can be registers or shared memory locations. The operation of this instruction could be dependable on predicate conditions. Moreover, the operation may modify some of these predicate flags.

Pred: Rx <- Ry + Rz

Mnemonics:
ADD (predicate_condition_out) (Destiny register) (Predicate_condition_in) , (source register 1), (source register 2)

The predicate_condition must be previously set by other instructions to be used as a condition for the addition operation.

Destiny and source registers are 16 or 32-bit size. The source register seems to be selected among (R0 - Rn), where n is the total number of registers employed by the application.

Example (SASS from NVCC):

```
IADD R2, g [0x4], R2                    (04208780 2000c809)
IADD R4, R5, R4                         (04010780 20000a11)
IADD.C0 R0, R0, c[0x1][0x0]             (044007c0 21000001)
IADD.C1 R0, g [0x4], R7                 (0421c7d0 2000c801)
IADD R7 (C3.CARRY), R7, c[0x1][0x2]     (0440b880 21000e1d)
IADD.CARRY1 R1, R1, R124                (041f1780 30400205)
IADD.CARRY0 R5, R5, R6                  (04018780 30400a15)
```

(SASS_assembly_lib):
Formats:
pending

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. **(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_Address_Register	R0= 000000, R5=000101, R6=000110…	
(9 – 15)7	Source_Register_1: It could be a GPRS or a shared memory location	**Register case:** **R0**= 00000 … **R5**= 00101 **R6**= 00110 …	**Shared memory case:** Offset value (9 - 12) 1 (13) 1 (14) g[0xoffset]
(16-21) 6	Not used	00 0000	
(22) 1	Carry_in_enable	**If carry_in_enable = 1 then perform the operation:** Dest = Source1 + Source2 + Carry_in **Denoted as** (IADD.CARRY0) in the mnemonic	
(23) 1	Not Used	0	
(24) 1	Second_source _operand_type	1 = Constant memory C[0x01][0x**XX**]	0 = General purppose Register
(25-27) 3	Not Used	000	
(28) 1	Carry_in	**If carry_in = 1 then perform the operation:** Dest = Source1 + Source2 + Carry_in **Denoted as** (IADD.CARRY0) in the mnemonic	
(29-31)3	Op_code	(001) **IADD**	

(32 - 33)2	instr_marker	00 = normal reg Access(load or store) (not extra instruction) **(by default)** 01 = normal reg Access(load or store) (with **Join**) (extra instruction) 10 = normal reg Access(load or store) (with **Exit**) 11 = immediate	
(34)1	Used for….	0	1
(35)1	destination type	0 = Register destination	
(36-37) 2	Predicate register to be set (enabling a new flag, only carry)	C0 = 00 **(by default)** C1 = 01 C2 = 10 C3 = 11	
(38) 1	Set predicate register	**1** = Enable predicate register set	**0** = Disable predicate register set

(39 – 43)5 predicate_condition

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less than	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

(44 - 45) 2	Input_predicate_register Used as: precondition to operate the instruction	**C0=** 00 **C2=** 10	**C1=** 01 **C3=** 11	
(46 - 53) 8	Source_register_2: It could be coming from: 1) GPRS 2) Constant memory 3) Shared memory	**Register case (46-52):** **R0=** 00000 … **R5=** 00101 **R6=** 00110 … **(53)** 0	**Constant memory:** Second part of the constant memory (i.e.) *C[0x2][**0x16**]* **(46-53)** = 0001 0110	**Shared memory:** **(46-52):** 000 0000 **(53): 1** (use of shared memory)
(54 – 57)4	The first part of the Source_2 when constant memory is employed	The first part of the constant memory (i.e.) *C[**0x2**][0x16]*		

		(54-57) = 0010	
58	W_32 Operation at 32 or 16 bits	**16 bits** = 0	**32 bits=** 1
59	Sign of Source_2	**Positive** = 0	**Negative** = 1
60	Not used	0	
(61 – 63)3	Sub_op_code	000	

IADD32 Instruction:

Checked YES

This instruction performs the Integer addition in (32 bits) of two sources of type register or shared memory locations. The operation of this instruction is not dependable on predicate conditions.

Rz <- Rx + Ry

Mnemonics:
IADD32 (Destiny register), (source register 1), (source register 2)

Destiny and source registers are 32-bit size. The source registers can be selected among (R0 - Rn), where n is the total number of registers employed by the application.

Example (SASS from NVCC):

IADD32 R0, g [0x5], R3 (2103EA00)
IADD32 R2, g [0x6], R3 (2103EC08)

(SASS_assembly_lib):
Formats:
pending

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long. **(Default)**	**1 =** 64 bit long.
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_Address_Register (32 bits)	**R0**= 000000, **R5**=000101, **R6**=000110…	
(9 – 15)7	Source_Register_1: It could be a GPRS or a shared memory location	**Register case:** **R0**= 00000 … **R5**= 00101 **R6**= 00110 …	**Shared memory case:** Offset value (9 - 12) 1 (13) 1 (14) g[0xoffset]
(16-22) 7	Source_2_Data_Register	**R0**= 000000, **R5**=000101, **R6**=000110…	
(23) 1	Not Used	0	
(24) 1	Source 1 Selector	1 = Shared memory operand	0 = Register operand
(25-27) 3	Not Used	000	
(28-31) 4	Instruction Op. Code	IADD32 = 0x2	

IADD32I Instruction:

Checked YES

This instruction performs the Immediate Integer addition in (32 bits) of one source of type register or shared memory locations and one immediate operand.

Rz <- Rx + Imm

Mnemonics:
IADD32 (Destiny register), (source register 1), (Immediate value)

Destiny and source registers are 32-bit size. The source register can be selected among (R0 - Rn), where n is the total number of registers employed by the application.

Example (SASS from NVCC):
IADD32I R5, R5, 0x4 (00000003 20048a15)
IADD32I R1, R1, 0x1 (00000003 20018205)
IADD32I R9, R4, 0x40C (00000043 200C8825)
IADD32I R11, R11, 0x30 (00000003 2030962d)

(SASS_assembly_lib):
Formats:
Pending

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. **(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_Address_Register (32 bits)	**R0**= 000000, **R5**=000101, **R6**=000110…	
(9 – 15) 7	Source_Register_1: It could be a GPRS or a shared memory location	**Register case:** **R0**= 00000 … **R5**= 00101 **R6**= 00110 …	**Shared memory case:** Offset value (9 - 12) 1 (13) 1 (14) g[0xoffset]
(16-21) 6	The low part of the immediate value (5 – 0)	**Low_Imm** XX XXXX	
(22-23) 2	Not Used	00	
(24) 1	Source 1 Selector	1 = Shared memory operand	0 = Register operand
(25-27) 3	Not Used	000	
(28-31) 4	Instruction Op. Code	IADD32I = 0x2	
(32 - 33)2	instr_marker	00 = normal reg Access(load or store) (not extra instruction) 01 = normal reg Access(load or store) (with **Join**) (extra instruction) 10 = normal reg Access(load or store) (with **Exit**) 11 = immediate **(by default)**	
(34) 1	Used for….	0	
(35) 1	destination type	0 = Register destination	
(36 – 59)	The high part of the immediate value (28 – 6)	**High_Imm** XXXX XXXX XXXX XXXX XXXX XX	
60	Not Used	0	
(61 – 63)	Sub_opcode	000	

IMAD Instruction:

Checked YES

This instruction performs the multiply and addition of three operands of 16 or 32 bits size. The sources can be registers or constant memory locations. The operation of these instructions could be dependable on predicate conditions.

Pre: Rz <- (Rx * Ry) + Rw

Mnemonics:
(Predicate) IMAD.(Size) Destiny, Source_1, Source_2, Source_3

Example (SASS from NVCC):

IMAD.U16 R3, R2H, R1L, R3	(0000c780 60020A0D)
IMAD.U16 R1, R2L, R1L, R3	(0000c780 60020805)
IMAD.U16 R1, R5L, R0L, R1	(00004780 60001405)
IMAD.U16 R1, g [0x6].U16, R0H, R1	(00204780 60014C05)
IMAD.U16 R6, R0L, R2L, R4	(00010780 60040019)
IMAD.U16 R11, R7L, R8L, R11	(0002C780 60101C2D)
IMAD.U16.C2 o[0x7f], R0L, R1L, R5	(000147E8 600201FD)
IMAD.U16 R4 (C3.TRUE), -R0H, R1H, R4	(0C012780 60030211)
IMAD.HI.SAT.S24 R1, R2, R1, R0	(00000780 70010405)

(SASS_assembly_lib):
Formats:
Pending

<table>
<tr><th>Bit(s)</th><th>Mnemonics</th><th colspan="4">Commentary</th></tr>
<tr><td>0</td><td>instr_is_long</td><td colspan="2">0 = 32 bit long.</td><td colspan="2">1 = 64 bit long. (Default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td colspan="2">0 = Normal ins. (Default)</td><td colspan="2">1 = System ins. (flow control)</td></tr>
<tr><td>(2 – 8) 7</td><td>Destiny Register (32 bits)</td><td colspan="2">R0 = 0000000
R5 = 0000101
R6 = 0000110</td><td colspan="2">R0(H) = 000000 1 , H = high
R5(H) =000101 1 , L = low
R6 (L) = 000110 0</td></tr>
<tr><td>(9 – 15)7</td><td>Source 1 Register (32 bits) or
H/L + Source 1 reg (16 bits)
Or immediate Shared memory</td><td>R0 = 0000000
R5 = 0000101
R6 = 0000110</td><td>R1(H) = 000001 1
R5(H) =000101 1
R6 (L) = 000110 0</td><td colspan="2">Shared memory case:
Offset value (9 - 12)
0 (13) (16 bits address)
1 (14)
g[0xoffset]</td></tr>
<tr><td>(16 – 22) 7</td><td>Source 2 Register (32 bits) or
H/L + Source 1 reg (16 bits)</td><td colspan="2">R0 = 0000000
R5 = 0000101
R6 = 0000110</td><td colspan="2">R1(H) = 000001 1
R5(H) =000101 1
R6 (L) = 000110 0</td></tr>
<tr><td>24</td><td>Not used</td><td colspan="4">0</td></tr>
<tr><td>25</td><td>Address Register or Imm address</td><td colspan="2">0 = Immediate address</td><td colspan="2">1 = Address Register</td></tr>
<tr><td>(26-27) 4</td><td>Address reg [1-0] Ax</td><td colspan="4">A0 = 00 (by default) A1 = 01
A2 = 10 A3 = 11</td></tr>
<tr><td>28</td><td>Size of operands</td><td colspan="2">0 = 16 or 32 bits operands</td><td colspan="2">1 = 24 or 32 bits operands</td></tr>
<tr><td>(23 – 31) 3</td><td>Instruction Op. Code</td><td colspan="4">IMAD = 011</td></tr>
<tr><td></td><td></td><td colspan="4"></td></tr>
<tr><td>(32 - 33)2</td><td>Instr_marker</td><td colspan="4">00 = normal reg Access(load or store) (not extra instruction) (by default)
01 = normal reg Access(load or store) (with Join) (extra instruction)
10 = normal reg Access(load or store) (with Exit)
11 = immediate</td></tr>
<tr><td>34</td><td>Not used (Address reg [2])</td><td colspan="4">0</td></tr>
<tr><td>35</td><td>destination type</td><td colspan="2">0 = Register</td><td colspan="2">1 = No destination, internal operation only</td></tr>
<tr><td>(36 – 37)2</td><td>Register to be set as result of operation if enabled</td><td colspan="4">00 = C0 (by default) 01 = C1
10 = C2 11 = C3</td></tr>
<tr><td>38</td><td>Set predicate register as result of operation</td><td colspan="2">1: Enable predicate register set</td><td colspan="2">0: Disable predicate register set</td></tr>
<tr><td>(39 – 43) 5</td><td>Predicate condition to operate the instruction</td><td colspan="4">
<table>
<tr><th>encoding</th><th>name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>never</td><td>always false</td><td>0</td></tr>
<tr><td>0x01</td><td>l</td><td>less tan</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>e</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>g</td><td>greater tan</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>lg</td><td>less or greater tan</td><td>~Z</td></tr>
<tr><td>0x06</td><td>ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>u</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>leu</td><td>not greater tan</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>o</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>c</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>a</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>s</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>no</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(44 - 45) 2</td><td>Input_predicate_register
Used as: precondition to operate the instruction</td><td colspan="2">C0= 00 (by default)
C2= 10</td><td colspan="2">C1= 01
C3= 11</td></tr>
<tr><td>(46 -52) 7</td><td>3_ Register_Operand (32 bits)</td><td colspan="4">R0= 000000, R5=000101, R6=000110…</td></tr>
<tr><td>53</td><td>Source 1 selector</td><td colspan="2">1 = Shared memory</td><td colspan="2">0 = Register</td></tr>
<tr><td>(54 –55) 2</td><td>Not used</td><td colspan="2">00</td><td colspan="2"></td></tr>
<tr><td>56</td><td>Source 2 Immediate value?</td><td colspan="2">1= Immediate value</td><td colspan="2">0 = Register</td></tr>
<tr><td>57</td><td>Not used</td><td colspan="4"></td></tr>
<tr><td>58</td><td>Sign of the Source 1</td><td colspan="2">1 = Negative</td><td colspan="2">0 =Positive</td></tr>
<tr><td>59</td><td>Sign of the Source 3</td><td colspan="2">1 = Negative</td><td colspan="2">0 =Positive</td></tr>
<tr><td>60</td><td>Not Used</td><td colspan="4">0</td></tr>
<tr><td>(61 – 63) 3</td><td>Sub_opcode</td><td colspan="4">000</td></tr>
</table>

IMAD32 Instruction:

Checked No

This instruction performs the integer multiply and addition of three operands of 16 or 32-bits size. The sources must be registers.

PrE: Rz <- (Ry * Rx) + Rz

The destiny register should be one of the source operands in the MAD operation. (Source 3 or Rz)

Mnemonics:

IMAD32 (Destiny)(size), (Source_1), (Source_2), (Source_3)

Example (SASS from NVCC):
IMAD32.U16 R1, R3H, R5H, R1 (600A0C04)
IMAD32.U16 R11, R9H, R30H, R11 (603C242C)

(SASS_assembly_lib):
Formats:
Pending…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long.**(Default)**
1	instr_is_flow	**0 =** Normal ins.	**1 =** System ins. (flow control) **(Default)**
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15)7	Source_Register_1: It should be a general purpose register at 16 or 32 bits	**32 bits:** **R0**= 0000000 **R5**= 0000101 **R6**= 0000110	**16 bits:** **R0(L)=** 000000 0 **R5(H)=** 000101 1 **R6(H)=** 000110 1
(16-22) 7	Source_Register_2: It should be a general purpose register at 16 or 32 bits	**32 bits:** **R0**= 0000000 **R5**= 0000101 **R6**= 0000110	**16 bits:** **R0(L)=** 000000 0 **R5(H)=** 000101 1 **R6(H)=** 000110 1
(23-27) 5	Not used	0 0000	
(28 – 31) 4	Instruction Op. Code	IMAD32 = 0x6	

IMAD32I Instruction:

Checked No

This instruction performs the integer multiply and addition of three operands of 16 or 32-bits size when one of the sources is an immediate value. The sources must be registers.

PrE: Rz <- (Ry * Imm) + Rz

The destiny register should be one of the source operands in the MAD operation. (Source 3 or Rz)

Mnemonics:

IMAD32I (Destiny), (Source_1), (IMM), (Source_3)

Example (SASS from NVCC):
IMAD32I.S16 R2, R4H, 0x25634, R2 (00002563 60341109)
IMAD32I.S16 R4, R12H, 0x3fffff, R4 (0003FFFF 603F3111)
IMAD32I.U16 R2, R4H, 0x25634, R2 (00002563 60341009)

(SASS_assembly_lib):
Formats:
Pending…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. **(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 7) 6	Destiny_Address_Register (32 bits)	**R0**= 000000, **R5**=000101, **R6**=000110…	
8	Sign of destiny reg	**0** = Unsigned	**1** = Signed
(9 – 15) 7	Source_Register_1: It could be a GPRS of 32 or 16-bits size	**32 bits:** **R0** = 0000000 **R5** = 0000101 **R6** = 0000110	**16 bits:** **R0(L)** = 000000 0 **R5(H)** = 000101 1 **R6(H)** = 000110 1
(16-21) 6	The low part of the immediate value (5 – 0)	**Low_Imm** XX XXXX	
(22-27) 3	Not Used	00 0000	
(28-31) 4	Instruction Op. Code	IMADD32I = 0x6	
(32 - 33)2	instr_marker	00 = normal reg Access(load or store) (not extra instruction) 01 = normal reg Access(load or store) (with **Join**) (extra instruction) 10 = normal reg Access(load or store) (with **Exit**) 11 = immediate **(by default)**	
(34 – 59) 26	The high part of the immediate value	**High_Imm** XXXX XXXX XXXX XXXX XXXX XX	
60	Not Used	0	
(61 – 63)	Sub_opcode	000	

LOP Instruction:

Checked YES

This instruction performs the logic operations (AND, OR, XOR, PASS, and NOT) into operands of 16 or 32 bits size. The sources can be registers or shared memory locations. The operation of this instruction could be dependable on predicate conditions. Moreover, the process may modify some of these predicate flags.

Pre: Rz <- Rx **and** Ry
Pre: Rz <- Rx **or** Ry
Pre: Rz <- Rx **xor** Ry

Mnemonics:
(Predicate) LOP. (Logic Operation).(Size) Destiny, Source_1, Source_2

Destiny and source registers are 16 or 32-bit size. The source_1 can be a register or a shared memory location. The source_2 can be a constant memory location.

Example (SASS from NVCC):
LOP.AND.U16 R0H, R0H, c[0x1][0x0] (00400780 D0800205)
LOP.XOR R7, R7, R8 (04008780 D0080E1D)
LOP.PASS_B R0 (C0.EQU), R0, ~R4 (0402C500 D0040001)
LOP.AND R5, R3, R2 (04000780 D0020615)

(SASS_assembly_lib):
Formats:
Pending

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. **(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_Address_Register (32 bits)	**R0**= 000000, **R5**=000101, **R6**=000110…	
(9 – 15) 7	Source 1: It could be a GPRS or a shared memory location	**Register case:** **R0**= 00000 … **R5**= 00101 **R6**= 00110 …	**Shared memory case:** Offset value (9 - 12) 1 (13) 1 (14) g[0xoffset]
(16 – 22) 7	Source_2: It can be a general purpose register or a constant memory location	**Register case:** **R0=** 00000 …, **R5=** 00101, **R6=** 00110 …	Second part of the constant memory (i.e.) *C[0x2][**0x16**]* **(16-22)** = 01 0110
(23 – 27) 5	High part of the Source_2 or configuration options:	**Shared memory:** **(23-25)** 000 **(26-27)** Address register part of the address (i.e.) g [**A2**+0x1] = 10 options are: **A0** = 00**, A1** = 01 **A2** = 10**, A3** = 11	First part of the constant memory (i.e.) *C[**0x2**][0x16]* (23-26) = 0010 0
(28 – 31)4	Instruction Op. Code	LOPS = 0xD	
(32 – 33)2	instr_marker	00 = normal reg Access(load or store) (not extra instruction) 01 = normal reg Access(load or store) (with **Join**) (extra instruction) 10 = normal reg Access(load or store) (with **Exit**) 11 = immediate **(by default)**	
34	High_Address of 2_operand	0	
35	destination type	0 = Register destination	1= Memory destination
(36 – 37) 2	Predicate register set (enabling a new flag) or Not used	**00** = C0 **(by default)** 10 = C2	01 = C1 11 = C3
(38) 1	Set predicate register	**1:** Enable predicate register set	**0**: Disable predicate register set
(39 – 43)5	predicate_condition to be considered to execute the instruction if the input predicate comparison is active		
(44 – 45) 2	Input predicate register to compare before to operate	**C0 =** 00 **C2 =** 10	**C1 =** 01 **C3 =** 11
(46 – 47) 2	Logic_operation_selector	**AND** = 00 **XOR** = 10	**OR** = 01 **NOT** = 11
(48 – 49) 2	Not used	00	
50	Source 1 inverted	1= inverted 0= not inverted not working	
51	Source 2 inverted	1= inverted 0= not inverted not working	
52	Not used		
53	Shared memory use for Source_2?	**Yes =** 1	**No =** 0 register use
54	Use of constant memory as Source_2?	**Yes =** 1	**No =** 0 register use
(55 – 56) 2	Index of the Constant memory space c[**xx**][xx]	00 **(not supported in FLEXGRIPPLUS)**	

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

57	Not used	0	
58	Size selector, Modifier 1	0: b16	1: b32
59	Size selector, Modifier 2	0: u16/u32	1: s16/s32
(60) 1	Not used	0	
(61 – 63)3	Sub_op_code	000	

ISET Instruction:

Checked YES

This instruction performs the integer comparison of two integer sources. A destiny register can be affected if selected. This instruction affects one flag of a predicate flag as a consequence of the comparison. This instruction can also require an input predicate condition as a precondition for its execution.

Pre: Rx vs Ry

Mnemonics:
ISET Destiny. (Predicate condition) , Source_1, Source_2

Source_1, Source_2, and Destiny are general purpose registers or constant memory parameters.

Example (SASS from NVCC):
ISET.S32.C0 o [0x7f], R2, R124, GT (307C05FD 6C0107C8)
ISET.S32.C0 o [0x7f], R0, R124, GT (307C01FD 6C0107C8)

(SASS_assembly_lib):
Formats:
ISET_regs(char sigd, int dest_reg, int source_reg_1, int source_reg_2, int comparison, int condition, int pred_reg_cond, char set_pred, int pred_reg_set, char output_reg, int marker)

<table>
<tr><th>Bit(s)</th><th>Mnemonics</th><th colspan="2">Commentary</th></tr>
<tr><td>0</td><td>instr_is_long</td><td>0 = 32 bit long.</td><td>1 = 64 bit long. (by default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td>0 = Normal ins. (by default)</td><td>1 = System ins. (flow control)</td></tr>
<tr><td>(2 – 8) 7</td><td>Destination_register</td><td colspan="2">R0= 000 0000, R1= 000 0001, R2= 000 0010, R15= 000 1111… or
R127 for internal operation</td></tr>
<tr><td>(9 – 15)7</td><td>Operand Source 1</td><td colspan="2">Could be register or data for Memory
For Register: Register number in the Core
For Memory: Address of memory (ej. [0x7f])</td></tr>
<tr><td>(16-22) 7</td><td>Operand Source 2</td><td colspan="2">Could be register or data for Memory
For Register: Register number in the Core
For Memory: Address of memory (ej. [0x7f]) (low part) [][XXXXX]</td></tr>
<tr><td>(23) 1</td><td>Operand Source 2 Selector</td><td>0 = register Source</td><td>1 = Memory location source</td></tr>
<tr><td>(24-27) 4</td><td>Not used</td><td colspan="2">0000</td></tr>
<tr><td>(28 – 31) 4</td><td>Instruction Op. Code</td><td colspan="2">ISET_OP = 0x3</td></tr>
<tr><td>(32 - 33) 2</td><td>instr_marker</td><td colspan="2">00 normal reg Access(load or store) (not extra instruction) (by default)
01 normal reg Access(load or store) (with Join) (extra instruction)
10 normal reg Access(load or store) (with Exit)
11 immediate</td></tr>
<tr><td>(35)1</td><td>Selection of the output comparison register: Destiny Register or Internal Register o[0x7F]</td><td>0 = use the output register</td><td>1 = Use the internal register o[00x7F]</td></tr>
<tr><td>(36 – 37) 2</td><td>Predicate register to be set after comparison</td><td colspan="2">C0 = 00 (by default) C1 = 01
C2 = 10 C3 = 11</td></tr>
<tr><td>(38) 1</td><td>Enable the set of the output predicate register</td><td>0 = No enable</td><td>1 = Enable</td></tr>
<tr><td>(39 – 43) 5</td><td>predicate field - selects a boolean function of the $c register</td><td colspan="2">
<table>
<tr><th>encoding</th><th>name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>never</td><td>always false (not used)</td><td>0</td></tr>
<tr><td>0x01</td><td>L (LT)</td><td>less than</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>E (EQ)</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>Le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>G (GT)</td><td>greater than</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>lg</td><td>less or greater than / not equall</td><td>~Z</td></tr>
<tr><td>0x06</td><td>ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>u</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>leu</td><td>not greater tan</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>o</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>c</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>a</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>s</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>no</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(44 – 45)2</td><td>Input predicate register to compare before to operate</td><td>C0= 00
C2= 10</td><td>C1= 01
C3= 11</td></tr>
<tr><td>(46-50) 5</td><td>Comparison method of the input predicate condition for operation</td><td colspan="2">
<table>
<tr><th>encoding</th><th>name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>never</td><td>always false (not used)</td><td>0</td></tr>
<tr><td>0x01</td><td>l</td><td>less than</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>e</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>g</td><td>greater than</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>lg</td><td>less or greater tan / not equal</td><td>~Z</td></tr>
<tr><td>0x06</td><td>ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>u</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>leu</td><td>not greater tan</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>o</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>c</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>a</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>s</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>no</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
</table>

(53) 1	Selection of the shared memory as one of the sources for comparison.	**1=** Shared memory used	**0 =** Shared memory not used
(54) 1	Selection of constant memory as one of the sources for comparison.	**1=** Constant memory used	**0 =** Constant memory not used
(55 – 57) 3	The high part of the second memory operand	[**XXXX**][]	
(58) 1	Size of operands	0: b16	1: b32
(59) 1	Signed or unsigned selection for destiny	0: u16/u32 **(Unsigned)**	1: s16/s32 **(Signed)**
(60) 1	Not used	0	
(63 – 61) 3	Secondary Operation Code	011	

Data handling and memory instructions

MVC

GLD

GST

MOV

MOV32

MVI

R2G

R2A

A2R

ADA

MVC Instruction:

Checked YES

This instruction performs the movement of an immediate operand in the opcode of the instruction. The immediate value can be combined with one address register.

PrE: Rx <- Constant[Imm]

Mnemonics:

MVC (Destiny).(size)(predicate) c[offset + Address_reg]

Example (SASS from NVCC):

MVC R1 (C3.EQU), c [0x1] [0x1] (2440F500 10000205)
MVC R1 (C2.NE), c [0x1] [0x1] (2440E280 10000205)
MVC R1, c[0x0] [A1+0x0] (2400C780 14000005)
MVC.U16 R1L, c[0x0] [A1+0x0].U8; (20000780 14000009)
MVC R2, c[0x0] [A2+0x0].U8 (24000780 18000009)
MVC.U16 R1L, c[0x0] [A2+0x0].U8 (20000780 18000009)
MVC R2, c[0x0] [A2+0x0].U16 (24004780 18000009)

(SASS_assembly_lib):
Formats:
Pending…

<table>
<tr><th>Bit(s)</th><th>Mnemonics</th><th colspan="2">Commentary</th></tr>
<tr><td>0</td><td>instr_is_long</td><td>0 = 32 bit long.</td><td>1 = 64 bit long. (Default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td>0 = Normal ins. (Default)</td><td>1 = System ins. (flow control)</td></tr>
<tr><td>(2 – 8) 7</td><td>Destiny_Address_Register (32 bits)</td><td colspan="2">R0= 0000000, R5=0000101, R6=0000110…</td></tr>
<tr><td>(9 – 15) 7</td><td>Inmeditate_Value_low_part</td><td colspan="2">constant memory address (i.e.)
C[0x2][0x6] = 000 0110</td></tr>
<tr><td>(16–25) 10</td><td>Not used</td><td colspan="2">00 0000 0000</td></tr>
<tr><td>(26-27) 2</td><td>Address register offset used for the constant memory addressing</td><td>A0 = 00
A2 = 10</td><td>A1 = 01
A3 = 11</td></tr>
<tr><td>(28 – 31)4</td><td>Instruction Op. Code</td><td colspan="2">MVC = 0x1</td></tr>
<tr><td></td><td></td><td colspan="2"></td></tr>
<tr><td>(32 - 33) 2</td><td>instr_marker</td><td colspan="2">00 = normal reg Access(load or store) (not extra instruction) (by default)
01 = normal reg Access(load or store) (with Join) (extra instruction)
10 = normal reg Access(load or store) (with Exit) (another option)
11 = immediate</td></tr>
<tr><td>(34 – 35) 2</td><td>Not used</td><td colspan="2">00</td></tr>
<tr><td>(36-37) 2</td><td>Predicate register set (enabling a new flag) or Not used</td><td>C0 = 00 (by default)
C1 = 01</td><td>C2 = 10
C3 = 11</td></tr>
<tr><td>38</td><td>Set predicate register as result of operation</td><td>1: Enable predicate register set</td><td>0: Disable predicate register set</td></tr>
<tr><td>(39 – 43) 5</td><td>Predicate condition to operate the instruction</td><td colspan="2">
<table>
<tr><th>Encoding</th><th>name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>never</td><td>always false</td><td>0</td></tr>
<tr><td>0x01</td><td>l</td><td>less tan</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>e</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>g</td><td>greater tan</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>lg</td><td>less or greater tan</td><td>~Z</td></tr>
<tr><td>0x06</td><td>ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>u</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>leu</td><td>not greater tan</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>o</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>c</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>a</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>s</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>no</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(44 - 45) 2</td><td>Input_predicate_register
Used as: precondition to operate the instruction</td><td>C0= 00 (by default)
C2= 10</td><td>C1= 01
C3= 11</td></tr>
<tr><td>(46-47) 2</td><td>Size of movement (source size)</td><td>11= 32 bits
00= 8 bits</td><td>01= 16 bits</td></tr>
<tr><td>(48-53) 6</td><td>Not used</td><td colspan="2">00 0000</td></tr>
<tr><td>54</td><td>Address Register or Imm address</td><td colspan="2">1 = Immediate address 0 = Address register</td></tr>
<tr><td>(55-57) 3</td><td>Not used</td><td colspan="2">000</td></tr>
<tr><td>58</td><td>Size of the destiny</td><td colspan="2">1= 32 bits 0= 16 bits</td></tr>
<tr><td>59</td><td>Signed or unsigned sources</td><td colspan="2">1=S16/S32 0= U16/U32 (Unsigned)</td></tr>
<tr><td>60</td><td>Not used</td><td colspan="2">0</td></tr>
<tr><td>(61 – 63) 3</td><td>Sub_opcode</td><td colspan="2">001</td></tr>
</table>

GLD Instruction:

Checked YES

This instruction performs the load of an operand of 8, 16 or 32-bits size from the main memory (global) in the GPGPU.

PrE: Rx <- Global_mem[Rz]

Mnemonics:

GLD (Destiny).(size) (Source_1)

Example (SASS from NVCC):
GLD.U8 R0, global14[R0] (80000780 D00E0001)
GLD.U8 R3, global14[R1] (80000780 D00E020D)
GLD.U8 R1, global14[R4] (80000780 D00E0805)
GLD.U32 R11, global14[R5] (80c00780 D00E0A2D)
GLD.U32 R0, global14[R6] (80c00780 D00E0C01)
GLD.S8 R0, global14[R0] (80200780 D00E0001)

(SASS_assembly_lib):
Formats:
Pending…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. **(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_Address_Register (32 bits)	**R0**= 0000000, **R5**=0000101, **R6**=0000110…	
(9 – 15) 7	Source_Register: GPRS	**R0** = 0000000, **R5** = 0000101, **R6** = 0000110…	
(16-22) 7	Main memory (Global) space 32-bit byte-oriented addressing.	g0[] - g15[] 0000000 = g0[] 0000001 = g1[] ... **0001110** = g14[] **(by default)**	
(23-27) 5	Not Used	0 0000	
(28-31) 4	Instruction Op. Code	GLD = 0xD	
(32 - 33) 2	instr_marker	00 = normal reg Access(load or store) (not extra instruction) **(by default)** 01 = normal reg Access(load or store) (with **Join**) (extra instruction) 10 = normal reg Access(load or store) (with **Exit**) 11 = immediate	
(34 - 38) 5	Not used	0 0000	
(39 – 43) 5	Predicate condition to operate the instruction	(see table below)	
(44 - 45) 2	Input_predicate_register Used as: precondition to operate the instruction	**C0=** 00 **(by default)** **C2=** 10	**C1=** 01 **C3=** 11
(46 - 52) 7	Not used	000 0000	
(53 – 55) 3	Destiny_move_size	000=DT_U8 (U8) 010=DT_U16 (U16) 100=DT_U64 (U64) **(NOT supported)** 110=DT_U32 (U32)	001=DT_S8 (S8) 011=DT_S16 (U16) 101=DT_U128 (U128) **(NOT sup.)** 111=DT_S32 (S32)
(56-60) 5	Not used	0 0000	
(61 – 63) 3	Sub_opcode	100 Load	

Encoding	name	Description	condition formula
0x00	never	always false	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater tan	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

GST Instruction:

This instruction performs the storage into the global memory of one operand coming from a general purpose register.

PrE: Global_mem[Rx] <- Ry

Mnemonics:

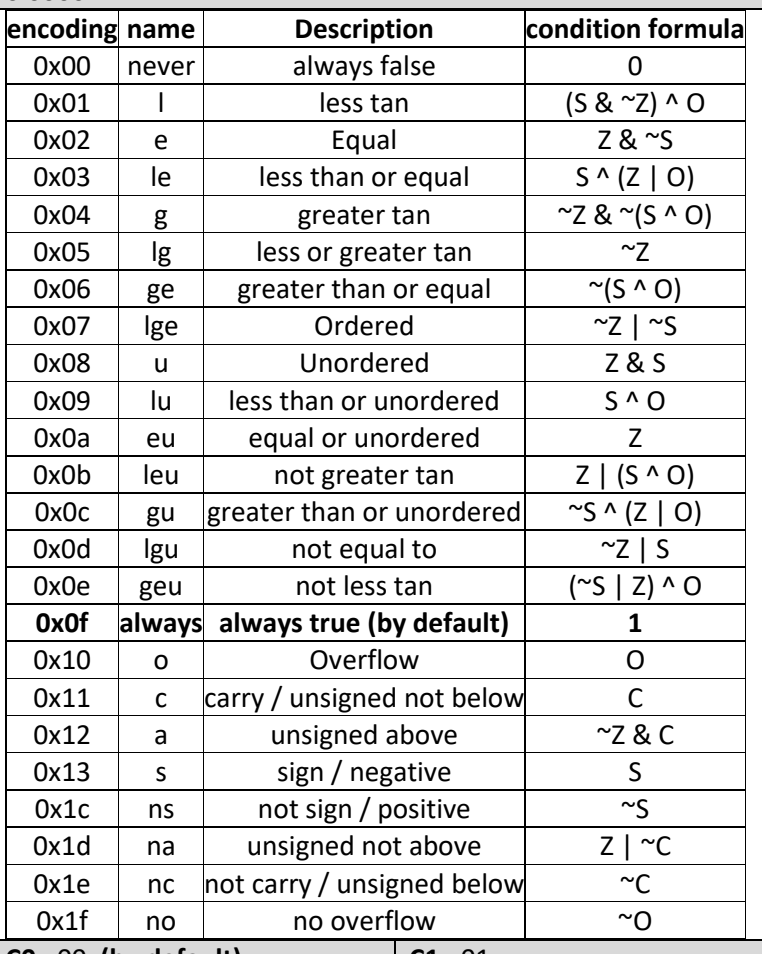
GST global14[Destiny_reg], Source_reg

Example (SASS from NVCC):

GST.U32 global14[R0], R10 (A0C00781 D00E0029)
GST.U32 global14[R1], R0 (A0C00781 D00E0201)
GST.U32 global14[R6], R5 (A0C00780 D00E0C15)
GST.U32 global14[R5], R3 (A0C00780 D00E0A0D)
GST.U32 global14[R4], R5 (A0C00780 D00E0815)

(SASS_assembly_lib):

Formats:
Pending…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. (**By default**)
1	instr_is_flow	**0 =** Normal ins. (**By default**)	**1 =** System ins. (flow control)
(2 – 8) 7	Source_ Data_Register (32 bits)	R0= 000000, R5=000101, R6=000110…	
(9-15)7	Destiny_Register_to_global_memory (32 bits)	R0= 000000, R5=000101, R6=000110…	
(16 – 21) 6	Global_memory_id	Global14 = 001110	
(22 - 27) 6	Not used	00 0000	
(28 – 31) 4	Instruction Op. Code	GST = 0xD	
(32 - 33) 2	instr_marker	00 normal reg Access(load or store) (not extra instruction) (**By default**) 01 normal reg Access(load or store) (with **Exit**) 10 normal reg Access(load or store) (with **Join**) (extra instruction) (33=1, 32=0) 11 immediate	
(34-38) 5	Not used	0 0000	
(39 – 43) 5	predicate_condition		
(44 – 45) 2	Input_predicate_register	**C0=** 00 **(by default)** **C2=** 10	**C1=** 01 **C3=** 11
(46 – 52) 7	Not used	000 0000	
(53-55) 3	Move_operand_size	U8 = 000 S8 = 001 U16 = 010 S16 = 011	U64 = 100 U128 = 101 U32 = 110 (**by default**) S32 = 111
(56 - 60) 5	Not used	0 0000	
(61 - 63) 3	Sub_Opcode	000=DT_U16 001=DT_S16 010=DT_S16 011=DT_U32	100=DT_S32 101=DT_S32 **(by default)** 110=DT_U32 111=DT_S32

encoding	name	Description	condition formula
0x00	never	always false	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater tan	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

MOV Instruction: (check final details)

Checked YES

This instruction performs the movement of an operand from a general purpose register into another.

PrE: Rx <-Ry / Shared_mem[Rx] / Imm

Mnemonics:

MOV (Destiny).(size) (Source)

Example (SASS from NVCC):

MOV R10, R124 (0403C780 1000F829)
MOV R0, g [A4+0x0] (0423C784 1000C001)
MOV R0, g [A3+0x0] (0423C780 1C00C001)
MOV R0, g [0x8] (0423C780 1000D001)
MOV.U16 R0H, g [0x2].U16 (0023C780 10004405)
MOV.U16 R0H, g [0x1].U16 (0023C780 10004205)

(SASS_assembly_lib):
Formats:
Pending…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. **(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15)7	Source_1: It can be a general-purpose register or a shared memory location	**Register case:** **R0**= 00000 …, **R5**= 00101, **R6**= 00110 …	**Shared memory location:** **(9-13)** offset of the location **(14) 1** **(15) 1**
(16-22) 7	Not used	**Register case:** **R0=** 00000 …, **R5=** 00101, **R6=** 00110 …	Second part of the constant memory (i.e.) *C[0x2][**0x16**]* **(16-22)** = 01 0110
(23-27) 5	High part of the Source_2 or configuration options:	**Shared memory:** **(23-25)** 000 **(26-27)** Address register part of the address (i.e.) g [**A2**+0x1] = 10 options are: **A0** = 00**, A1** = 01 **A2** = 10**, A3** = 11	First part of the constant memory (i.e.) *C[**0x2**][**0**x16]* (23-26) = 0010 0
(28 – 31) 4	Instruction Op. Code	MOV = 0x1	

(32 - 33) 2	instr_marker	00 normal reg Access(load or store) (not extra instruction) (**By default**) 01 normal reg Access(load or store) (with **Exit**) 10 normal reg Access(load or store) (with **Join**) (extra instruction) (33=1, 32=0) 11 immediate
34	Address register high part	A4 = 1 Ax = 0
(35 - 38) 4	Not used	0000
(39 – 43) 5	predicate_condition	

encoding	name	Description	condition formula
0x00	never	always false	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater tan	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

(44 – 45) 2	Input_predicate_register	**C0=** 00 **(by default)** **C2=** 10	**C1=** 01 **C3=** 11
(46-48) 3	Size of movement (source size)	000 DT_U8 001 DT_U16 010 DT_S16 011 DT_U32	100 DT_U32 101 DT_U32 110 DT_U32 111 DT_U32
49	??	1	
(50 – 52) 3	Not used	000	
53	Source 1 selector	**1 =** Shared memory	**0 =** Register
(54 – 57) 4	Not used	0000	
58	Size of the destiny	1= 32 bits	0= 16 bits
59	Signed or unsigned sources	1=S16/S32	0= U16/U32 **(Unsigned)**
60	Not used	0	
(61 - 63) 3	Sub_Opcode	000=DT_U16 001=DT_S16 010=DT_S16 011=DT_U32	100=DT_S32 101=DT_S32 **(by default)** 110=DT_U32 111=DT_S32

MOV32 Instruction:

Checked YES

This instruction performs the movement of an operand from a general purpose register into another

PrE: Rx <- Ry

Mnemonics:

MOV32 Destiny_reg, Source_reg or Shared_mem [Source_address_reg]

Example (SASS from NVCC):

MOV32 R1, g [0x8] (1100F004)
MOV32 R0, g [0x7] (1100EE00)
MOV32 R0, R1 (10008200)
MOV32 R4, R3 (10008610)

(SASS_assembly_lib):
Formats:
Pending…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long. **(Default)**	**1 =** 64 bit long.
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_Address_Register (32 bits)	**R0**= 000000, **R5**=000101, **R6**=000110…	
(9 – 15)7	Source_Register_1: It could be a GPRS or a shared memory location	**Register case:** **R0**= 00000 … **R5**= 00101 **R6**= 00110 …	**Shared memory case:** Offset value (9 - 12) 1 (13) 1 (14) g[0xoffset]
(16-22) 7	Not used	000 0000	
(23) 1	Not used	0	
(24) 1	Source 1 Selector	1 = Shared memory operand	0 = Register operand
(25-27) 3	Not Used	000	
(28-31) 4	Instruction Op. Code	MOV32 = 0x1	

MVI Instruction:

Checked YES

This instruction performs the movement of an immediate operand in the opcode of the instruction.

PrE: Rx <- Imm

Mnemonics:

MVI (Destiny).(size) (Imm)

Example (SASS from NVCC):

MVI R11, 0x1 (00000003 1001802D)
MVI R2, 0x1 (00000003 10018009)
MVI R11, 0x17 (00000003 1017802D)

(SASS_assembly_lib):
Formats:
Pending…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. **(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_Address_Register (32 bits)	**R0**= 0000000, **R5**=0000101, **R6**=0000110…	
(9 – 15) 7	Not used	000 0000	
(16 – 21) 6	Inmeditate_Value_low_part	**XX XXXX**	
(22 – 27) 6	Not used	00 0000	
(28 – 31)4	Instruction Op. Code	GLD = 0x1	
(32 - 33) 2	instr_marker	00 = normal reg Access(load or store) (not extra instruction) 01 = normal reg Access(load or store) (with **Join**) (extra instruction) 10 = normal reg Access(load or store) (with **Exit**) 11 = immediate **(by default)**	
(34 - 59)26	Immediate high part 26 bits	**XX XXXX XXXX XXXX XXXX XXXX XXXX**	
60	Not used		
(61 – 63) 3	Sub_opcode	000	

R2G Instruction:

Checked YES

This instruction performs the movement of an operand from a general purpose register to one shared memory location. The location in the share memory can be combined with an address register and one immediate (or address offset) value.

PrE: Shared_mem[Ax + offset] <- Rx

Mnemonics:

R2G.(size destiny).(size source) g[Address_reg + offset], Source_reg

Example (SASS from NVCC):

R2G.U32.U32 g[A1+0xc], R11 (E422c780 04001801)
R2G.U32.U32 g[A1+0x40c], R0 (E4200780 04081801)

(SASS_assembly_lib):
Formats:
Pending…

<table>
<tr><th>Bit(s)</th><th>Mnemonics</th><th colspan="2">Commentary</th></tr>
<tr><td>0</td><td>instr_is_long</td><td>0 = 32 bit long.</td><td>1 = 64 bit long. (Default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td>0 = Normal ins. (Default)</td><td>1 = System ins. (flow control)</td></tr>
<tr><td>(2 – 6) 5</td><td>Not used</td><td colspan="2">0 0000</td></tr>
<tr><td>(7 – 19) 13</td><td>Address offset</td><td colspan="2">The size capacity of the shared memory is 0x4000</td></tr>
<tr><td>(20–25) 6</td><td>Not used</td><td colspan="2">00 0000</td></tr>
<tr><td>(26-27) 2</td><td>Address register used for the shared memory addressing</td><td>A0 = 00
A2 = 10</td><td>A1 = 01
A3 = 11</td></tr>
<tr><td>(28 – 31)4</td><td>Instruction Op. Code</td><td colspan="2">R2G = 0x0</td></tr>
<tr><td></td><td></td><td colspan="2"></td></tr>
<tr><td>(32 - 33) 2</td><td>instr_marker</td><td colspan="2">00 = normal reg Access(load or store) (not extra instruction) (by default)
01 = normal reg Access(load or store) (with Join) (extra instruction)
10 = normal reg Access(load or store) (with Exit) (another option)
11 = immediate</td></tr>
<tr><td>(34 – 35) 2</td><td>Not used</td><td colspan="2">00</td></tr>
<tr><td>(36-37) 2</td><td>Predicate register set (enabling a new flag) or Not used</td><td>C0 = 00 (by default)
C1 = 01</td><td>C2 = 10
C3 = 11</td></tr>
<tr><td>38</td><td>Set predicate register as result of operation</td><td>1: Enable predicate register set</td><td>0: Disable predicate register set</td></tr>
<tr><td>(39 – 43) 5</td><td>Predicate condition to operate the instruction</td><td colspan="2">
<table>
<tr><th>Encoding</th><th>name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>never</td><td>always false</td><td>0</td></tr>
<tr><td>0x01</td><td>l</td><td>less tan</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>e</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>g</td><td>greater tan</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>lg</td><td>less or greater tan</td><td>~Z</td></tr>
<tr><td>0x06</td><td>ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>u</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>leu</td><td>not greater tan</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>o</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>c</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>a</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>s</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>no</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(44 – 45) 2</td><td>Input_predicate_register
Used as: precondition to operate the instruction</td><td>C0= 00 (by default)
C2= 10</td><td>C1= 01
C3= 11</td></tr>
<tr><td>(46–52) 7</td><td>Source register</td><td colspan="2">R0= 0000000, R5=0000101, R6=0000110…</td></tr>
<tr><td>(53–54) 2</td><td>Size of movement (source size)</td><td colspan="2">01= 32 bits
00= 16 bits
10= 8 bits</td></tr>
<tr><td>(55-57) 3</td><td>Not used</td><td colspan="2">000</td></tr>
<tr><td>58</td><td>Size of the destiny</td><td>1= 32 bits</td><td>0= 16 bits</td></tr>
<tr><td>59</td><td>Signed or unsigned sources</td><td>1=S16/S32</td><td>0= U16/U32 (Unsigned)</td></tr>
<tr><td>60</td><td>Not used</td><td colspan="2">0</td></tr>
<tr><td>(61 – 63) 3</td><td>Sub_opcode</td><td colspan="2">111</td></tr>
</table>

R2A Instruction:

Checked YES

This instruction performs the movement of an operand from a general purpose register to one address register that is used to address the shared or constant memories in the GPGPU.

PrE: Ax <- Rx + Imm

Mnemonics:

R2A Address_reg, Source_reg, Imm

Example (SASS from NVCC):

R2A A1, R10, 0x2 (C0000780 00021405)
R2A A2, R11 (C0000780 00001609)
R2A A3, R9, 0x2 (C0000780 0002120D)

(SASS_assembly_lib):
Formats:
Pending…

<table>
<tr><th>Bit(s)</th><th>Mnemonics</th><th colspan="2">Commentary</th></tr>
<tr><td>0</td><td>instr_is_long</td><td>0 = 32 bit long.</td><td>1 = 64 bit long. (by default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td>0 = Normal ins. (by default)</td><td>1 = System ins. (flow control)</td></tr>
<tr><td>(2 – 8) 7</td><td>Destiny_ Address_Register (32 bits)</td><td colspan="2">A0= 000000, A5=000101, A6=000110…</td></tr>
<tr><td>(9 - 15) 7</td><td>Source_Data_Register</td><td colspan="2">R0= 000000, R5=000101, R6=000110…</td></tr>
<tr><td>(16 – 27) 12</td><td>Immediate value</td><td colspan="2">0xYYY</td></tr>
<tr><td>(28 – 31) 4</td><td>Instruction Op. Code</td><td colspan="2">R2A = 0x0</td></tr>
<tr><td></td><td></td><td colspan="2"></td></tr>
<tr><td>(32 - 33)</td><td>instr_marker</td><td colspan="2">00 normal reg Access(load or store) (not extra instruction) (by default)
01 normal reg Access(load or store) (with Join) (extra instruction)
10 normal reg Access(load or store) (with Exit)
11 immediate</td></tr>
<tr><td>34</td><td>Not used</td><td colspan="2">0</td></tr>
<tr><td>(35 – 38) 4</td><td>Not used</td><td colspan="2">0000</td></tr>
<tr><td>(39 – 43) 5</td><td>Predicate condition to operate the instruction</td><td colspan="2">
<table>
<tr><th>Encoding</th><th>name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>never</td><td>always false</td><td>0</td></tr>
<tr><td>0x01</td><td>l</td><td>less tan</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>e</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>g</td><td>greater tan</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>lg</td><td>less or greater tan</td><td>~Z</td></tr>
<tr><td>0x06</td><td>ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>u</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>leu</td><td>not greater tan</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>o</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>c</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>a</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>s</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>no</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(44 – 45) 2</td><td>Input_predicate_register</td><td>C0= 00 (by default)
C2= 10</td><td>C1= 01
C3= 11</td></tr>
<tr><td>(46 – 60) 14</td><td>Not used</td><td colspan="2">00 0000 0000 0000</td></tr>
<tr><td>(61-63)</td><td>Sub_Opcode</td><td colspan="2">000 DT_U16
001 DT_S16
010 DT_S16
011 DT_U32
100 DT_S32
101 DT_S32
110 DT_U32 (by default)
111 DT_S32</td></tr>
</table>

A2R Instruction:

Checked YES

This instruction performs the movement of an operand from an address register to one general purpose register.

PrE: Rx <- Ax + Imm

Mnemonics:

A2R Destiny_reg, Address_reg

Example (SASS from NVCC):

A2R R3, A1 (40000780 0400000d)

(SASS_assembly_lib):

Formats:
Pending…

<table>
<tr><th>Bit(s)</th><th>Mnemonics</th><th colspan="2">Commentary</th></tr>
<tr><td>0</td><td>instr_is_long</td><td>0 = 32 bit long.</td><td>1 = 64 bit long. (by default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td>0 = Normal ins. (by default)</td><td>1 = System ins. (flow control)</td></tr>
<tr><td>(2 – 8) 7</td><td>Destiny_ Address_Register (32 bits)</td><td colspan="2">A0= 000000, A5=000101, A6=000110…</td></tr>
<tr><td>(9 - 15) 7</td><td>Not used</td><td colspan="2">000 0000</td></tr>
<tr><td>(16 – 25) 10</td><td>Immediate value</td><td colspan="2">00 0000 0000</td></tr>
<tr><td>(26-27) 2</td><td>Address register used for the shared memory addressing</td><td>A0 = 00
A2 = 10</td><td>A1 = 01
A3 = 11</td></tr>
<tr><td>(28 – 31) 4</td><td>Instruction Op. Code</td><td colspan="2">R2A = 0x0</td></tr>
<tr><td></td><td></td><td colspan="2"></td></tr>
<tr><td>(32 - 33)</td><td>instr_marker</td><td colspan="2">00 normal reg Access(load or store) (not extra instruction) (by default)
01 normal reg Access(load or store) (with Join) (extra instruction)
10 normal reg Access(load or store) (with Exit)
11 immediate</td></tr>
<tr><td>34</td><td>Not used</td><td colspan="2">0</td></tr>
<tr><td>(35 – 38) 4</td><td>Not used</td><td colspan="2">0000</td></tr>
<tr><td>(39 – 43) 5</td><td>Predicate condition to operate the instruction</td><td colspan="2">
<table>
<tr><th>Encoding</th><th>name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>never</td><td>always false</td><td>0</td></tr>
<tr><td>0x01</td><td>l</td><td>less tan</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>e</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>g</td><td>greater tan</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>lg</td><td>less or greater tan</td><td>~Z</td></tr>
<tr><td>0x06</td><td>ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>u</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>leu</td><td>not greater tan</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>o</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>c</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>a</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>s</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>no</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(44 – 45) 2</td><td>Input_predicate_register</td><td>C0= 00 (by default)
C2= 10</td><td>C1= 01
C3= 11</td></tr>
<tr><td>(46 – 60) 14</td><td>Not used</td><td colspan="2">00 0000 0000 0000</td></tr>
<tr><td>(61-63)</td><td>Sub_Opcode</td><td colspan="2">010</td></tr>
</table>

ADA Instruction:

Checked yes

This instruction performs the addition of immediate value in the address registers (These registers are employed to address the shared memory in the GPGPU)

Ax <- Ay + Imm

Mnemonics:
ADA (Destiny register), (source register), Imm

Destiny and source registers are 32-bit size. The source register seems to be selected among (A0 - A3) instead the destiny may be (A0 – A15)

Example (SASS from NVCC):
ADA A4, A2, 0x1b0 (20000780 d8036011)
ADA A4, A3, 0x1618 (20000780 dc2c3011)

(SASS_assembly_lib):
Formats:
Not implemented yet… pending to describe

<table>
<tr><th>Bit(s)</th><th>Mnemonics</th><th colspan="2">Commentary</th></tr>
<tr><td>0</td><td>instr_is_long</td><td>0 = 32 bit long.</td><td>1 = 64 bit long. (Default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td>0 = Normal ins. (Default)</td><td>1 = System ins. (flow control)</td></tr>
<tr><td>(2 – 8) 7</td><td>Destiny_Address_Register</td><td colspan="2">A0= 000000, A5=000101, A6=000110…</td></tr>
<tr><td>(9 – 24)16</td><td>Immediate value (Low part?) (FlexGrip only uses 22-9)</td><td colspan="2">Imm value: 0xXXXX</td></tr>
<tr><td>25</td><td>Source_1_Selector??</td><td>0 = Register Source</td><td>1 = Shared Mem.</td></tr>
<tr><td>(26-27)2</td><td>Source_Address_Register</td><td colspan="2">Options are:
A0: 00 A1: 01
A2: 10 A3: 11</td></tr>
<tr><td>(28 – 31) 4</td><td>Instruction Op. Code</td><td colspan="2">ADA = 0xD</td></tr>
<tr><td>(32 - 33)2</td><td>instr_marker</td><td colspan="2">00 = normal reg Access(load or store) (not extra instruction) (by default)
01 = normal reg Access(load or store) (with Join) (extra instruction)
10 = normal reg Access(load or store) (with Exit)
11 = immediate</td></tr>
<tr><td>(34)1</td><td>Not used</td><td colspan="2">0</td></tr>
<tr><td>(35)1</td><td>destination type</td><td>0 = Register destination</td><td>1= Memory destination</td></tr>
<tr><td>(36-37)2</td><td>Predicate register set (enabling a new flag) or Not used</td><td colspan="2">C0 = 00 (by default)
C1 = 01
C2 = 10
C3 = 11</td></tr>
<tr><td>(38)1</td><td>Write enable / set predicate register</td><td colspan="2">1 = write enabled (just for memory destination)
1 = enable predicate register set, 0 = disable predicate register set
Not used (0)</td></tr>
<tr><td>(39 – 43) 5</td><td>predicate_condition</td><td colspan="2">
<table>
<tr><th>encoding</th><th>name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>never</td><td>always false (not used)</td><td>0</td></tr>
<tr><td>0x01</td><td>l</td><td>less tan</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>e</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>g</td><td>greater tan</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>lg</td><td>less or greater tan / not equal</td><td>~Z</td></tr>
<tr><td>0x06</td><td>ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>u</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>leu</td><td>not greater than</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>o</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>c</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>a</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>s</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>no</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(45 - 59)</td><td>Not used? High part of the Imm, value, or from the source of destiny register?</td><td colspan="2">0000 0000 0000 0000</td></tr>
<tr><td>(61 – 63)3</td><td>Sub_op_code</td><td colspan="2">001</td></tr>
</table>

Floating point instructions

FADD32

FADD

FADD32I

FMUL

FMUL32

FMUL32I

FMAD

FMAD32

FMAD32I

F2F

F2I

I2F

FSET

RCP

RCP32

FADD32 Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the floating-point addition in single-precision (32 bits) of two sources. The sources can be registers or shared memory locations.

FRx <- FRy + FRz

Mnemonics:
FADD32 (Destiny register), (source register), (source register)

Destiny and source registers are 32-bit size. The source register seems to be selected among (R0 - Rn), where n is the total number of registers employed by the application.

Example (SASS from NVCC):
FADD32 R3, R3, R0 (B000060C)
FADD32 R9, -g [A1+0xd], R3 (B503FA24)
FADD32 R6, g [A2+0x1], -R2 (B9426218)

(SASS_assembly_lib):
Formats:
Not implemented yet…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long. (Default)	**1 =** 64 bit long.
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_Register	R0= 000000, R5=000101, R6=000110…	
(9 – 14)5	Source_Register_1: It could be a GPRS or a shared memory location	**Register case**: R0= 00000 … R5= 00101 R6= 00110 …	**Shared memory case:** Offset value (9 - 12) 1 (13) 1 (14)
(15) 1	Source_1_sign	**0 =** Positive.	**1 =** Negative.
(16-21)5	Source_Register_2: It should be a GPRS.	**Register case**: R0= 00000 … R5= 00101, R6= 00110 …	
(22) 1	Source_2_sign	**0 =** Positive.	**1 =** Negative.
(24)1	Source_1_using_shared_memory	**0 =** No, Source 1 is register.	**1 =** Yes, Source 1 comes from Shared memory.
(26-27) 2	Address register offset used by the shared memory addressing	**A0 =** 00 **A2 =** 10	**A1 =** 01 **A3 =** 11
(28-31) 4	Instruction Opcode	FADD32 = 0xD	

FADD Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the floating-point addition in single-precision (32 bits) of two sources. The sources can be registers or shared memory locations. The operation of this instruction could be dependable on predicate conditions. Moreover, the operation may modify some of these predicate flags.

Pred: FRx <- FRy + FRz

Mnemonics:
FADD (predicate_condition) (Destiny register), (source register), (source register)

The predicate_condition must be previously set by other instructions to be used as a condition for the addition operation.

Destiny and source registers are 32-bit size. The source register seems to be selected among (R0 - Rn), where n is the total number of registers employed by the application.

Example (SASS from NVCC):

FADD R6, R7, -R6 (08018780 B0000E19)
FADD R0 (C1.EQU), R0, R4 (00011500 B0000001)
FADD.TRUNC R1, R1, c[0x1][0x16] (00458780 B1030205)

(SASS_assembly_lib):
Formats:
Not implemented yet…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.**(Default)**	**1 =** 64 bit long.
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_Address_Register	R0= 000000, R5=000101, R6=000110…	
(9 – 15)7	Source_Register_1: It could be a GPRS or a shared memory location	**Register case:** **R0**= 00000 … **R5**= 00101 **R6**= 00110 …	**Shared memory case:** Offset value (9 - 12) 1 (13) 1 (14)
(16-17) 2	Rounding Options	**Not rounded** = 00 **Truncate (rounded to zero)** = 11	
(24) 1	Source_register_is_constant_memory (Cmem)	**Yes =** 1	**No =** 0
(28 – 31) 4	Instruction Op. Code	FADD = 0xD	

(32 - 33)2	instr_marker	00 = normal reg Access(load or store) (not extra instruction) **(by default)** 01 = normal reg Access(load or store) (with **Join**) (extra instruction) 10 = normal reg Access(load or store) (with **Exit**) 11 = immediate		
(34)1	Used for….	0	1	
(35)1	destination type	0 = Register destination	1= Memory destination	
(36-37) 2	Predicate register set (enabling a new flag) or Not used	C0 = 00 **(by default)** C1 = 01 C2 = 10 C3 = 11		
(38) 1	Set predicate register	**1** = Enable predicate register set	**0** = Disable predicate register set	
(39 – 43)5	predicate_condition			
(44 - 45) 2	Input predicate register to compare before to operate	**C0=** 00 **C2=** 10	**C1=** 01 **C3=** 11	
(46 - 53) 8	Source_register_2: It could be coming from: 4) GPRS 5) Constant memory 6) Shared memory	**Register case (46-52):** **R0=** 00000 … **R5=** 00101 **R6=** 00110 … **(53)** 0	**Constant memory:** Second part of the constant memory (i.e.) *C[0x2][**0x16**]* **(46-53)** = 0001 0110	**Shared memory:** **(46-52):** 000 0000 **(53): 1** (use of shared memory)
(54 – 57)4	First part of the Source_2 when constant memory is employed	First part of the constant memory (i.e.) *C[**0x2**][0x16]* (54-57) = 0010		
58	Sign of Source_1	**Positive =** 0	**Negative =** 1	
59	Sign of Source_2	**Positive =** 0	**Negative =** 1	
60	Not used	0		
(61 – 63)3	Sub_op_code	000		

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater tan	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

FADD32I Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the floating point addition in single precision (32 bits) between one source and one immediate value. The source and destiny can be registers.

FRx <- FRy + Imm

Mnemonics:
FADD32I (Destiny register), (source register), (Immediate value)

The immediate value is a 32 bits single-precision operand.

Destiny and source registers are 32-bit size. The source register seems to be selected among (R0 - Rn), where n is the total number of registers employed by the application.

Example (SASS from NVCC):

FADD32I R7, R7, 0x3f000000 (03F00003B0000E1D)
FADD32I R0, R0, 0x49be9b7c (049BE9B7B03C0001)
FADD32I R2, R2, -0x41000000 (0BF00003B0000409)

(SASS_assembly_lib):
Formats:
Not implemented yet…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.**(Default)**	**1 =** 64 bit long.
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15)7	Source_Register_1: it should be a general purpose register.	**Register case:** **R0**= 00000 …, **R5**= 00101, **R6**= 00110 …	
(16-21) 6	The low part of the immediate value of 32 bits (lowest 6 bits)	Immediate value, low part	
(24) 1	Source_register_is_constant_memory (Cmem)	**Yes =** 1	**No =** 0
(28 – 31) 4	Instruction Op. Code	FADD32I = 0xD	
(32 - 33)2	instr_marker	**00** = normal reg Access(load or store) (not extra instruction) **01** = normal reg Access(load or store) (with **Join**) (extra instruction) **10** = normal reg Access(load or store) (with **Exit**) **11** = immediate **(by default)**	
(34 - 59) 26	The high part of the immediate value of 32 bits		
(60) 1	Not used	0	
(61 – 63)3	Sub_op_code	000	

FMUL Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the floating-point multiplication in single-precision (32 bits) between two sources. The sources and destiny can be registers, shared memory locations, constant memory locations, or immediate values. A predicate condition can be present as a precondition for executing the operation.

PRE: FRx <- FRy * FRz

Mnemonics:
FMUL. (Predicate condition) (Destiny), (Source_1), (Source_2)

Source_1 and Source_2 can be the immediate value, shared memory location, or constant memory element. In most cases (*Source_1 can be shared memory location. Similarly, Source_2 can be the constant memory location*)

Example (SASS from NVCC):

FMUL R6, R7, R6 (00000780C0060E19)
FMUL R4, -R4, R3 (04000780C0030811)
FMUL.TRUNC R6 (C0.NEU), R6, c[0x1][0x1] (0040C680C0810C19)
FMUL.TRUNC R4, g [A2+0x1], -R2 (0820C780C802C211)

(SASS_assembly_lib):
Formats:
Not implemented yet…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.**(Default)**	**1 =** 64 bit long.
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15)7	Source_1: It can be a general-purpose register or a shared memory location	**Register case:** **R0**= 00000 …, **R5**= 00101, **R6**= 00110 …	**Shared memory location:** **(9-13)** offset of the location **(14) 1** **(15) 1**
(16-22) 7	Source_2: It can be a general-purpose register or a constant memory location	**Register case:** **R0=** 00000 …, **R5=** 00101, **R6=** 00110 …	Second part of the constant memory (i.e.) *C[0x2][**0x16**]* **(16-22)** = 01 0110
(23-27) 5	High part of the Source_2 or configuration options:	**Shared memory:** **(23-25)** 000 **(26-27)** Address register part of the address (i.e.) g [**A2**+0x1] = 10 options are: **A0** = 00**, A1** = 01 **A2** = 10**, A3** = 11	First part of the constant memory (i.e.) *C[**0x2**][**0**x16]* (23-26) = 0010 0
(28 – 31) 4	Instruction Op. Code	FMUL = 0xC	

(32 - 33)2	instr_marker	00 = normal reg Access(load or store) (not extra instruction) **(by default)** 01 = normal reg Access(load or store) (with **Join**) (extra instruction) 10 = normal reg Access(load or store) (with **Exit**) 11 = immediate	
(34)1	Used for….	0	1
(35)1	destination type	0 = Register destination	1= Memory destination
(36-37) 2	Predicate register set (enabling a new flag) or Not used	**C0** = 00 **(by default)** **C1** = 01 **C2** = 10 **C3** = 11	
(38) 1	Set predicate register	**1** = Enable predicate register set	**0** = Disable predicate register set
(39 – 43)5	predicate_condition	(see table below)	
(44 - 45) 2	Input predicate register to compare before to operate	**C0=** 00 **C1=** 01 **C2=** 10 **C3=** 11	
(46 - 47) 2	Result round method	**Not rounding =** 00	**Rounded to zero =** 11
(53) 1	Shared memory use for Source_2?	**Yes =** 1	**No =** 0
(54) 1	Use of constant memory as Source_2?	**Yes =** 1	**No =** 0
(58) 1	Sign of Source_1	**Positive** = 0, **Negative** = 1	
(59) 1	Sign of Source_2	**Positive** = 0, **Negative** = 1	
(60) 1	Not used	0	
(61 – 63)3	Sub_op_code	000	

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

FMUL32 Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the floating-point multiplication in single-precision (32 bits) between two sources. The sources and destiny can be registers, shared memory locations, constant memory locations, or immediate values. Predicate conditions are not included as a precondition to operate this instruction.

FRx <- FRy * FRz

Mnemonics:
FMUL32 (Destiny), (Source_1), (Source_2)

Source_1 and Source_2 can be the immediate value, shared memory location, or constant memory element. In most cases (*Source_1 can be shared memory location. Similarly, Source_2 can be the constant memory location*)

Example (SASS from NVCC):

FMUL32 R3, R3, R0 (C000060C)
FMUL32 R7, R8, R7 (C007101C)
FMUL32 R2, g [A1+0x6], R0 (C5006C08)

(SASS_assembly_lib):
Formats:
Not implemented yet…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long. **(Default)**	**1 =** 64 bit long.
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_Register	R0= 000000, R5=000101, R6=000110…	
(9 – 14)5	Source_Register_1: It could be a GPRS or a shared memory location	**Register case**: R0= 00000 … R5= 00101 R6= 00110 …	**Shared memory case:** Offset value (9 - 12) 1 (13) 1 (14)
(15) 1	Source_1_sign	**0 =** Positive.	**1 =** Negative.
(16-21)5	Source_Register_2: It should be a GPRS.	**Register case**: R0= 00000 … R5= 00101, R6= 00110 …	
(22) 1	Source_2_sign	**0 =** Positive.	**1 =** Negative.
(24)1	Source_1_using_shared_memory	**0 =** No, Source 1 is register.	**1 =** Yes, Source 1 comes from Shared memory.
(26-27) 2	Address register offset used by the shared memory addressing	**A0 =** 00 **A2 =** 10	**A1 =** 01 **A3 =** 11
(28-31) 4	Instruction Opcode	FMUL32 = 0xC	

FMUL32I Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the floating-point multiplication in single-precision (32 bits) between two sources. The sources and destiny can be registers, shared memory locations, constant memory locations, or immediate values. Predicate conditions are not included as a precondition to operate this instruction.

FRx <- FRy * Imm

Mnemonics:
FMUL32 (Destiny), (Source_1), Imm

Source_1 and Destiny are general-purpose registers.

Example (SASS from NVCC):

FMUL32I R7, R7, 0x3f000000 (03F00003C0000E1D)
FMUL32I R1, R2, 0x40510005 (04051003C0050405)
FMUL32I R1, R0, 0x3f22f983 (03F22F9BC0030005)

(SASS_assembly_lib):
Formats:
Not implemented yet…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long.**(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15)7	Source_Register_1: it should be a general purpose register.	**Register case:** **R0**= 00000 …, **R5**= 00101, **R6**= 00110 …	
(16-21) 6	The low part of the immediate value of 32 bits (lowest 6 bits)	Immediate value, low part	
(24) 1	Source_register_is_constant_memory (Cmem)	**Yes =** 1	**No =** 0
(28 – 31) 4	Instruction Op. Code	FMUL2I = 0xC	
(32 - 33)2	instr_marker	**00** = normal reg Access(load or store) (not extra instruction) **01** = normal reg Access(load or store) (with **Join**) (extra instruction) **10** = normal reg Access(load or store) (with **Exit**) **11** = immediate **(by default)**	
(34 - 59) 26	The high part of the immediate value of 32 bits		
(60) 1	Not used	0	
(61 – 63)3	Sub_op_code	000	

FMAD32 Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the floating-point multiplication and addition in single-precision (32 bits) between three sources. The sources and destiny can be registers, shared memory locations, constant memory locations, or immediate values. Predicate conditions are not included as a precondition to operate this instruction.

PrE: FRx <- ((FRy) * (FRx))+ (FRz)

The destiny register should be one of the source operands in the MAD operation.

Mnemonics:

FMAD (Destiny), (Source_1), (Source_2), (Source_3)

Source_1 and Destiny are general-purpose registers.

Example (SASS from NVCC):

FMAD R0, -g [A2+0x1], R2, R0 (04200780 E802C201)
FMAD R3, g [A1+++0x1], R6, R3 (0020C780 E606C20D)
FMAD R5, R7, R6, R5 (00014780 E0060E15)
FMAD R2, -R6, c[0x1][0xc], R3 (0440C780 E08C0C09)

(SASS_assembly_lib):
Not implemented yet…

Bit(s)	Mnemonics	Commentary		
0	instr_is_long	**0 =** 32 bit long.		**1 =** 64 bit long. **(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**		**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…		
(9 – 15)7	**Source_1**: It can be a general-purpose register or a shared memory location	**Register case:** **R0**= 00000 …, **R5**= 00101, **R6**= 00110 …	**Shared memory location:** **(9-13) an** offset of the location **(14) 1** **(15) 1**	
(16-22) 7	**Source_2**: It can be a general-purpose register or a constant memory location	**Register case:** **R0=** 00000 …, **R5=** 00101, **R6=** 00110 …	Second part of the constant memory (i.e.) *C[0x2][**0x16**]* **(16-22)** = 001 0110	
(23-27) 5	High part of the Source_2 or configuration options:	**Shared memory:** **(23-25)** 000 **(26-27)** Address register part of the address (i.e.) g [**A2**+0x1] = 10 options are: **A0** = 00**, A1** = 01 **A2** = 10**, A3** = 11	**Constant memory:** First part of the constant memory (i.e.) *C[**0x2**][**0**x16]* **(23-27)** = 0 0010	**Constant memory:** This field also can be employed as part of the source 3, when the constant memory is employed as SRC3. **(23)**:0 **(24-27):** First part (lower of the address for constant memory)
(28 – 31) 4	Instruction Op. Code	FMAD = 0xE		

(32 - 33)2	instr_marker	00 = normal reg Access(load or store) (not extra instruction) **(by default)** 01 = normal reg Access(load or store) (with **Join**) (extra instruction) 10 = normal reg Access(load or store) (with **Exit**) 11 = immediate	
(34)1	Used for….	0	1
(35)1	destination type	0 = Register destination	1= Memory destination
(36-37) 2	Predicate register set (enabling a new flag) or Not used	**C0** = 00 **(by default)** **C1** = 01 **C2** = 10 **C3** = 11	
(38) 1	Set predicate register	**1** = Enable predicate register set	**0** = Disable predicate register set
(39 – 43)5	predicate_condition	(see table below)	
(44 - 45) 2	Input predicate register to compare before to operate	**C0=** 00 **C2=** 10	**C1=** 01 **C3=** 11
(46-52) 6	Source 3: It could be a register or a constant memory location	**Register case:** **R0=** 00000 …, **R5=** 00101, **R6=** 00110 …	**Constant memory:** High part of the constant memory (i.e.) *C[**0x2**][**0**x16]* **(46-52)** = 00 0010
(53) 1	Shared memory use for Source_2? A bit indicates if the shared memory is employed	**Yes =** 1	**No =** 0
(54) 1	Use of constant memory for Source_2 or Source_3?	**Yes =** 1	**No =** 0

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

(58) 1	Sign of Source_1	**Positive** = 0, **Negative** = 1	
(59) 1	Sign of Source_3	**Positive** = 0, **Negative** = 1	
(60) 1	Not used	0	
(61 – 63)3	Sub_op_code	000	

FMAD32I Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the floating-point multiplication and addition in single-precision (32 bits) among two sources and one immediate value. The sources and the destiny most of the time are general-purpose registers. Predicate values are not included as preconditions to execute the instruction.

PrE: FRx <- ((FRy) * (Imm))+ (FRz)

The destiny register should be one of the source operands in the MAD operation.

Mnemonics:

FMAD (Destiny), (Source_1), (Immediate), (Source_3)

Source_1 and Destiny are general-purpose registers.

Example (SASS from NVCC):

FMAD32I R1, -R3, 0x39fd8000, R1 (039FD803 E0008605)
FMAD32I R0, R1, 0x3fc00000, R0 (03FC0003 E0000201)
FMAD32I R2, R3, 0x3b86d46d, R2 (03B86D47 E02d0609)

(SASS_assembly_lib):
Formats:
Not implemented yet…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long.**(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 14)6	Source_Register_1: it should be a general purpose register.	**Register case:** **R0**= 00000 …, **R5**= 00101, **R6**= 00110 …	
(15)1	Sign of Source_1	**1 =** Negative	**0 =** Positive
(16-21) 6	The low part of the immediate value of 32 bits (lowest 6 bits)	Immediate value, low part	
(24) 1	Source_register_is_constant_memory (Cmem)	**Yes =** 1	**No =** 0
(28 – 31) 4	Instruction Op. Code	FMAD32I = 0xE	

(32 - 33)2	instr_marker	**00** = normal reg Access(load or store) (not extra instruction) **01** = normal reg Access(load or store) (with **Join**) (extra instruction) **10** = normal reg Access(load or store) (with **Exit**) **11** = immediate **(by default)**
(34 - 59) 26	The high part of the immediate value of 32 bits	
(60) 1	Not used	0
(61 – 63)3	Sub_op_code	000

F2F Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the floating conversion between two floating-point elements. This instruction is used to change the format or to move among floating-point sources. A predicate condition can be employed as part of preconditions.

Pre: FRx <- (FRy)

Mnemonics:

FMAD (Destiny), (Source_1), (Immediate), (Source_3)

Source_1 and Destiny are general-purpose registers.

Example (SASS from NVCC):

```
F2F.F32.F32 R4, -R4                (E4004780 A0000811)
F2F.F32.F32 R1, -R2                (E4004780 A0000405)
F2F.F32.F32 R0 (C0.NEU), |R2|      (C4104680 A0000401)
F2F.F32.F32 R11, R11               (C4004780 A000162D)
```

(SASS_assembly_lib):
Formats:
Not implemented yet…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. **(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15)7	**Source_1**: It is a general-purpose register	**Register:** **R0**= 00000 …, **R5**= 00101, **R6**= 00110 …	
(16-22) 7	**Source_2**: It can be a general-purpose register or a constant memory location	**Register case:** **R0=** 00000 …, **R5=** 00101, **R6=** 00110 …	Second part of the constant memory (i.e.) *C[0x2][**0x16**]* **(16-22)** = 001 0110
(28 – 31) 4	Instruction Op. Code	F2F = 0xA	
(32 - 33)2	instr_marker	**00** = normal reg Access(load or store) (not extra instruction) **(by default)** **01** = normal reg Access(load or store) (with **Join**) (extra instruction) **10** = normal reg Access(load or store) (with **Exit**) **11** = immediate	
(34)1	Used for….	0	1
(35)1	destination type	0 = Register destination	1= Memory destination
(36-37) 2	Predicate register set (enabling a new flag) or Not used	**C0** = 00 **(by default)** **C1** = 01 **C2** = 10 **C3** = 11	
(38) 1	Set predicate register	**1** = Enable predicate register set	**0** = Disable predicate register set
(39 – 43)5	predicate_condition	(see table below)	
(44 - 45) 2	Input predicate register to compare before to operate	**C0=** 00 **C1=** 01 **C2=** 10 **C3=** 11	
(46) 1	Fixed value, purpose?	**1**	
(52) 1	Absolute value in source_1	**Yes =** 1	**No =** 0
(54) 1	Use of constant memory for Source_2 or Source_3?	**Yes =** 1	**No =** 0
(58) 1	Sign of Source_1??	**Positive** = 0, **Negative** = 1 **(default = 1)**	
(60) 1	Not used	0	
(61 – 63)3	Sub_op_code	110	

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

F2I Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the floating conversion into an integer (from float to integer). Predicate conditions can be employed as part of the preconditions to execute the instruction.

Pre: (Int)Rx <- ((Float)Ry)

Mnemonics:

F2I. (Predicate condition) (Destiny), (Source_1)

Source_1 and Destiny are general-purpose registers.

Example (SASS from NVCC):

F2I.S32.F32 R1, R0 (8C004780 A0000005)
F2I.S32.F32.TRUNC R2, R2 (8C064780 A0000409)
F2I.U32.F32.TRUNC R5, R5 (84064780 A0000a15)

(SASS_assembly_lib):
Formats:
Not implemented yet…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. **(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15)7	**Source_1**: It is a general-purpose register	**Register:** **R0**= 00000 …, **R5**= 00101, **R6**= 00110 …	
(28 – 31) 4	Instruction Op. Code	F2I = 0xA	
(32 - 33)2	instr_marker	**00** = normal reg Access(load or store) (not extra instruction) **(by default)** **01** = normal reg Access(load or store) (with **Join**) (extra instruction) **10** = normal reg Access(load or store) (with **Exit**) **11** = immediate	
(34)1	Used for….	0	1
(35)1	destination type	0 = Register destination	1= Memory destination
(36-37) 2	Predicate register set (enabling a new flag) or Not used	**C0** = 00 **(by default)** **C1** = 01 **C2** = 10 **C3** = 11	
(38) 1	Set predicate register	**1** = Enable predicate register set	**0** = Disable predicate register set
(39 – 43)5	predicate_condition	(see table below)	
(44 - 45) 2	Input predicate register to compare before to operate	**C0=** 00 **C1=** 01 **C2=** 10 **C3=** 11	
(46) 1	Fixed value, purpose?	**1**	
(49-50) 2	Rounding mechanism	**00 =** not rounding	**11 =** to zero
(54) 1	Use of constant memory for Source_2 or Source_3?	**Yes =** 1	**No =** 0
(58) 1	Destiny to signed?	**No** = 0	**Yes** = 1 **(default = 1)**
(60) 1	Not used	0	
(61 – 63)3	Sub_op_code	100	

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

I2F Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the integer conversion into a floating-point value in single-precision (32 bits). Predicate conditions can be employed as part of the preconditions to execute the instruction.

Pre: (Float) Rx <- ((Int) Ry)

Mnemonics:

I2F. (Predicate condition) (Destiny), (Source_1)

Source_1 and Destiny are general-purpose registers.

Example (SASS from NVCC):

```
I2F.F32.S32 R2, R4                      (44014780 A0000809)
I2F.F32.U32.TRUNC R3 (C0.EQU), R2       (44064500 A000040D)
I2F.F32.S32 R6, R1                      (44014780 A0000219)
I2F.F32.U32.TRUNC R3, R4                (44064780 A000080D)
```

(SASS_assembly_lib):
Formats:
Not implemented yet…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. **(Default)**
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15)7	**Source_1**: It is a general-purpose register	**Register:** **R0**= 00000 …, **R5**= 00101, **R6**= 00110 …	
(28 – 31) 4	Instruction Op. Code	F2I = 0xA	
(32 - 33)2	instr_marker	**00** = normal reg Access(load or store) (not extra instruction) **(by default)** **01** = normal reg Access(load or store) (with **Join**) (extra instruction) **10** = normal reg Access(load or store) (with **Exit**) **11** = immediate	
(34)1	Used for….	0	1
(35)1	destination type	0 = Register destination	1= Memory destination
(36-37) 2	Predicate register set (enabling a new flag) or Not used	**C0** = 00 **(by default)** **C1** = 01 **C2** = 10 **C3** = 11	
(38) 1	Set predicate register	**1** = Enable predicate register set	**0** = Disable predicate register set
(39 – 43)5	predicate_condition	(see table below)	
(44 - 45) 2	Input predicate register to compare before to operate	**C0=** 00 **C2=** 10	**C1=** 01 **C3=** 11
(46) 1	Fixed value, purpose?	**1**	
(48) 1	From signed value Source_1	**1** = Yes	**0** = No
(49-50) 2	Rounding mechanism	**00 =** not rounding	**11 =** to zero
(54) 1	Use of constant memory for Source_2 or Source_3?	**Yes =** 1	**No =** 0
(58) 1	Destiny to signed?	**No** = 0	**Yes** = 1 **(default = 1)**
(60) 1	Not used	0	
(61 – 63)3	Sub_op_code	010	

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

FSET Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs a comparison between two floating-point values and modifies one of the predicate flags on one predicate registers as the effect of the comparison. A predicate condition could be part of the preconditions to execute the instruction. This instruction does not generate changes in the comparable values, but may change a destiny register is select as logical output.

Pre: (FRx vs. FRy)

Mnemonics:

FSET (Affected predicate register and condition) (Source_1), (Source_2), ((Input predicate condition)

Source_1, Source_2, and Destiny are general purpose registers or constant memory parameters.

Example (SASS from NVCC):

```
FSET.C0 o[0x7f], |R2|, c[0x1][0xb], EQ        (605087C8 B08b05FD)
FSET.C0 o[0x7f], |R2|, c[0x1][0x10], GT       (605107C8 B09005FD)
FSET.C0 o[0x7f] (C0.NE), R1, R124, EQ         (600082C8 B07c03FD)
FSET.C0 o[0x7f] (C0.NE), R1, R124, EQ         (600082C8 B07c03FD)
FSET.C0 o[0x7f], R16, R17, LT                 (600047C8 B01121FD)
```

(SASS_assembly_lib):
Formats:
Not implemented yet…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0** = 32 bit long.	**1** = 64 bit long. **(Default)**
1	instr_is_flow	**0** = Normal ins. **(Default)**	**1** = System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15)7	**Source_1:** It is a general-purpose register	**Register:** **R0**= 00000 …, **R5**= 00101, **R6**= 00110 …	
(16-22) 7	**Source_2:** This can be from general-purpose registers or a constant memory location.	**Register case:** **R0=** 00000 …, **R5=** 00101, **R6=** 00110 …	Second part of the constant memory (i.e.) *C[0x2][**0x16**]* **(16-22)** = 001 0110
(23-27) 5	The high part of the constant memory location	First part of the constant memory (i.e.) C[**0x2**][0x16] **(23-27)** = 0 0010	
(28 – 31) 4	Instruction Op. Code	FSET = 0xB	

(32 - 33)2	instr_marker	**00** = normal reg Access(load or store) (not extra instruction) **(by default)** **01** = normal reg Access(load or store) (with **Join**) (extra instruction) **10** = normal reg Access(load or store) (with **Exit**) **11** = immediate	
(34)1	Used for….	0	1
(35)1	destination type	0 = Register destination	1= Memory destination
(36-37) 2	Predicate register set (enabling a new flag) or Not used	**C0** = 00 **(by default)** **C1** = 01 **C2** = 10 **C3** = 11	
(38) 1	Set predicate register	**1** = Enable predicate register set	**0** = Disable predicate register set

(39 – 43)5 predicate_condition

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater tan	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

(44 - 45) 2 Input predicate register to compare before to operate — **C0=** 00 **C1=** 01 **C2=** 10 **C3=** 11

(46-50) 5 Predicate condition to perform between the two main Sources.

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	L (LT)	less tan	(S & ~Z) ^ O
0x02	E (EQ)	Equal	Z & ~S
0x03	Le	less than or equal	S ^ (Z | O)
0x04	G (GT)	greater tan	~Z & ~(S ^ O)
0x05	Lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C

		0x1e	nc	not carry / unsigned below	~C
		0x1f	no	no overflow	~O
(52) 1	Absolute or signed value in Source_1	**1** = Yes		**0** = No	
(54) 1	Use of constant memory for Source_2 or Source_3?	**Yes =** 1		**No =** 0	
(60) 1	Not used	0			
(61 – 63)3	Sub_op_code	011			

RCP Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the reciprocal operation of a floating-point value in single-precision (32 bits). A predicate condition could be part of the preconditions to execute the instruction.

Pre: FRx <- reciprocal (FRy)

Mnemonics:

RCP.(predicate condition) Destiny, Source_1

Source_1, Source_2, and Destiny are general purpose registers or constant memory parameters.

Example (SASS from NVCC):

RCP R0, R0 (00000780 90000001)
RCP R4 (C0.NEU), R2 (00000680 90000411)

(SASS_assembly_lib):
Formats:
Not implemented yet…

<table>
<tr><th>Bit(s)</th><th>Mnemonics</th><th colspan="2">Commentary</th></tr>
<tr><td>0</td><td>instr_is_long</td><td>0 = 32 bit long.</td><td>1 = 64 bit long. (Default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td>0 = Normal ins. (Default)</td><td>1 = System ins. (flow control)</td></tr>
<tr><td>(2 – 8) 7</td><td>Destiny_General_purpose_register</td><td colspan="2">R0= 000000, R5=000101, R6=000110…</td></tr>
<tr><td>(9 – 15)7</td><td>Source_1: It is a general-purpose register</td><td colspan="2">Register: R0= 00000 …, R5= 00101, R6= 00110 …</td></tr>
<tr><td>(28 – 31) 4</td><td>Instruction Op. Code</td><td colspan="2">RCP = 0x9</td></tr>
<tr><td>(32 - 33)2</td><td>instr_marker</td><td colspan="2">00 = normal reg Access(load or store) (not extra instruction) (by default)
01 = normal reg Access(load or store) (with Join) (extra instruction)
10 = normal reg Access(load or store) (with Exit)
11 = immediate</td></tr>
<tr><td>(34)1</td><td>Used for….</td><td>0</td><td>1</td></tr>
<tr><td>(35)1</td><td>destination type</td><td>0 = Register destination</td><td>1= Memory destination</td></tr>
<tr><td>(36-37) 2</td><td>Predicate register set (enabling a new flag) or Not used</td><td colspan="2">C0 = 00 (by default) C1 = 01
C2 = 10 C3 = 11</td></tr>
<tr><td>(38) 1</td><td>Set predicate register</td><td>1 = Enable predicate register set</td><td>0 = Disable predicate register set</td></tr>
<tr><td>(39 – 43)5</td><td>predicate_condition</td><td colspan="2">
<table>
<tr><th>encoding</th><th>name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>never</td><td>always false (not used)</td><td>0</td></tr>
<tr><td>0x01</td><td>l</td><td>less tan</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>e</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>g</td><td>greater tan</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>lg</td><td>less or greater tan / not equal</td><td>~Z</td></tr>
<tr><td>0x06</td><td>ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>u</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>leu</td><td>not greater tan</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>o</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>c</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>a</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>s</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>no</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(44 - 45) 2</td><td>Input predicate register to compare before to operate</td><td>C0= 00
C2= 10</td><td>C1= 01
C3= 11</td></tr>
<tr><td>(60) 1</td><td>Not used</td><td colspan="2">0</td></tr>
<tr><td>(61 – 63)3</td><td>Sub_op_code</td><td colspan="2">011</td></tr>
</table>

RCP32 Instruction:

Checked No, partially implemented, and checking in progress.

This instruction performs the reciprocal operation of a floating-point value in single-precision (32 bits). This instruction does not require a predicate condition to start the execution.

FRx <- reciprocal (FRy)

Mnemonics:

RCP32 Destiny, Source_1

Source_1, Source_2, and Destiny are general purpose registers or constant memory parameters.

Example (SASS from NVCC):

RCP32 R1, R1 (90000204)
RCP32 R4, R4 (90000810)

(SASS_assembly_lib):
Formats:
Not implemented yet…

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.**(Default)**	**1 =** 64 bit long.
1	instr_is_flow	**0 =** Normal ins. **(Default)**	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15)7	Source_Register_1: it should be a general purpose register.	**Register case:** **R0**= 00000 …, **R5**= 00101, **R6**= 00110 …	
(28 – 31) 4	Instruction Op. Code	RCP32 = 0x9	

Especial function unit instructions

SIN

COS

RRO

EX2

RSQ

LG2

SIN instruction:

Checked Not implemented

This instruction generates the approximate SIN operation of an input operand in the format of 32 bits floating-point.
Destiny_f ← SIN (Source_f)

Mnemonics:
Direct SIN: **SIN** R**x**, R**x**

Example (SASS from NVCC):

SIN R1, R1 (80000780 90000205)
SIN R12, R12 (80000780 90001831)

(SASS_assembly_lib):
Not_available

Note:
No comments.

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0** = 32 bit long.	**1** = 64 bit long. (**by default**)
1	instr_is_flow	**0** = Normal ins. (by default)	**1** = System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15) 7	Source_operand_register: it should be a general purpose register.	R0= 000000…, R5= 00101, R6= 00110 …	
(16-27) 12	Not used	000000000000	
(28 – 31) 4	Instruction Op. Code	SIN_OP = 0x9h	
(32 - 33) 2	instr_marker	**00** normal register access(load or store) (not extra instruction) **(by default)** 01 normal register access(load or store) (with **Join**) (extra instruction) 10 normal register access(load or store) (with **Exit**) 11 immediate	
(34)1	Used for….	0	
(35)1	destination type	0 = Register destination	
(36-37) 2	Predicate register set (enabling a new flag) or Not used	C0 = **00 (by default)** C1 = 01 C2 = 10 C3 = 11	
(38) 1	Set predicate register	**1** = Enable the setting of a predicate register	
(39 – 43) 5	predicate_condition	(see table below)	
(44 - 45) 2	Input predicate register to compare before to operate	**C0=** 00 **C2=** 10	
(46 – 60)	Not used	000 0000 0000 000	
(61 – 63) 3	Sub_op_code	**100 SIN**	

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

COS instruction:

Checked Not implemented

This instruction generates the approximate COS operation of an input operand in the format of 32 bits floating-point.
Destiny_f ← COS (Source_f)

Mnemonics:
Direct COS: **COS** R**x**, R**x**

Example (SASS from NVCC):

COS R11, R11 (A0000780 9000162d)
COS R1, R1 (A0000780 90000205)

(SASS_assembly_lib):
Not_available

Note:
No comments.

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0** = 32 bit long.	**1** = 64 bit long. (**by default**)
1	instr_is_flow	**0** = Normal ins. (by default)	**1** = System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15) 7	Source_operand_register: it should be a general purpose register.	R0= 000000…, R5= 00101, R6= 00110 …	
(16-27) 12	Not used	000000000000	
(28 – 31) 4	Instruction Op. Code	COS_OP = 0x9h	
(32 - 33) 2	instr_marker	**00** normal register access(load or store) (not extra instruction) **(by default)** 01 normal register access(load or store) (with **Join**) (extra instruction) 10 normal register access(load or store) (with **Exit**) 11 immediate	
(34)1	Used for….	0	
(35)1	destination type	0 = Register destination	
(36-37) 2	Predicate register set (enabling a new flag) or Not used	C0 = **00 (by default)** C1 = 01 C2 = 10 C3 = 11	
(38) 1	Set predicate register	**1** = Enable the setting of a predicate register	
(39 – 43) 5	predicate_condition	(see table below)	
(44 - 45) 2	Input predicate register to compare before to operate	**C0=** 00 **C2=** 10	
(46 – 60)	Not used	000 0000 0000 000	
(61 – 63) 3	Sub_op_code	**101 COS**	

encoding	name	Description	condition formula
0x00	never	always false (not used)	0
0x01	l	less tan	(S & ~Z) ^ O
0x02	e	Equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater tan	~Z & ~(S ^ O)
0x05	lg	less or greater tan / not equal	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	Ordered	~Z | ~S
0x08	u	Unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	o	Overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

RRO instruction: (Range Reduction Operation)

Checked Not implemented

This instruction reduces the range and adjusts the phase to operate a transcendent operation in the SFU. The operands are in 32 bits floating-point single precision.
Destiny_f ← RRO (Source_f, method_of_reduction)

Mnemonics:
Direct RRO: **RRO** R**x**, R**x**, method (SIN, Exp)

Example (SASS from NVCC):
RRO R12, R12, SIN (C0000780 b0001831)
RRO R3, R2, EX2; (C0004780 b000040d)

(SASS_assembly_lib):
Not_available

Note:
No comments.

<table>
<tr><th>Bit(s)</th><th>Mnemonics</th><th colspan="2">Commentary</th></tr>
<tr><td>0</td><td>instr_is_long</td><td>0 = 32 bit long.</td><td>1 = 64 bit long. (by default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td>0 = Normal ins. (by default)</td><td>1 = System ins. (flow control)</td></tr>
<tr><td>(2 – 8) 7</td><td>Destiny_General_purpose_register</td><td colspan="2">R0= 000000, R5=000101, R6=000110…</td></tr>
<tr><td>(9 – 15) 7</td><td>Source_operand_register: it should be a general purpose register.</td><td colspan="2">R0= 000000…, R5= 00101, R6= 00110 …</td></tr>
<tr><td>(16-27) 12</td><td>Not used</td><td colspan="2">000000000000</td></tr>
<tr><td>(28 – 31) 4</td><td>Instruction Op. Code</td><td colspan="2">RRO_OP = 0xB_h</td></tr>
<tr><td>(32 - 33) 2</td><td>instr_marker</td><td colspan="2">00 normal register access(load or store) (not extra instruction) (by default)
01 normal register access(load or store) (with Join) (extra instruction)
10 normal register access(load or store) (with Exit)
11 immediate</td></tr>
<tr><td>(34)1</td><td>Used for….</td><td colspan="2">0</td></tr>
<tr><td>(35)1</td><td>destination type</td><td colspan="2">0 = Register destination</td></tr>
<tr><td>(36-37) 2</td><td>Predicate register set (enabling a new flag) or Not used</td><td colspan="2">C0 = 00 (by default)
C1 = 01
C2 = 10
C3 = 11</td></tr>
<tr><td>(38) 1</td><td>Set predicate register</td><td colspan="2">1 = Enable the setting of a predicate register</td></tr>
<tr><td>(39 – 43) 5</td><td>predicate_condition</td><td colspan="2">
<table>
<tr><th>encoding</th><th>Name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>Never</td><td>always false (not used)</td><td>0</td></tr>
<tr><td>0x01</td><td>L</td><td>less tan</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>E</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>Le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>G</td><td>greater tan</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>Lg</td><td>less or greater tan / not equal</td><td>~Z</td></tr>
<tr><td>0x06</td><td>Ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>Lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>U</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>Lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>Eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>Leu</td><td>not greater than</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>Gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>Lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>Geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>O</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>C</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>A</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>S</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>Ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>Na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>Nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>No</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(44 - 45) 2</td><td>Input predicate register to compare before to operate</td><td colspan="2">C0 = 00
C2 = 10</td></tr>
<tr><td>(46-47) 2</td><td>Selector of the phase corrector</td><td colspan="2">00 = SIN (quadrant 1)
01 = Exp2 (quadrant 2)
10 = ? (quadrant 3)
11 = ? (quadrant 4)</td></tr>
<tr><td>(46 – 60)</td><td>Not used</td><td colspan="2">000 0000 0000 000</td></tr>
<tr><td>(61 – 63) 3</td><td>Sub_op_code</td><td colspan="2">110 RRO</td></tr>
</table>

LG2 instruction:

Checked Not implemented

This instruction calculates the logarithm in a binary base of an input operand.
Destiny_f ← Log_2 (Source_f)

Mnemonics:
Direct LG2: **LG2** R**y,** R**x**

Example (SASS from NVCC):
LG2 R0, R0; (60000780 90000001)
LG2 R2, R2; (60000780 90000409)

(SASS_assembly_lib):
Not_available

Note:
No comments.

<table>
<tr><th>Bit(s)</th><th>Mnemonics</th><th colspan="2">Commentary</th></tr>
<tr><td>0</td><td>instr_is_long</td><td>0 = 32 bit long.</td><td>1 = 64 bit long. (by default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td>0 = Normal ins. (by default)</td><td>1 = System ins. (flow control)</td></tr>
<tr><td>(2 – 8) 7</td><td>Destiny_General_purpose_register</td><td colspan="2">R0= 000000, R5=000101, R6=000110…</td></tr>
<tr><td>(9 – 15) 7</td><td>Source_operand_register: it should be a general purpose register.</td><td colspan="2">R0= 000000…, R5= 00101, R6= 00110 …</td></tr>
<tr><td>(16-27) 12</td><td>Not used</td><td colspan="2">000000000000</td></tr>
<tr><td>(28 – 31) 4</td><td>Instruction Op. Code</td><td colspan="2">LG2_OP = 0x9h</td></tr>
<tr><td>(32 - 33) 2</td><td>instr_marker</td><td colspan="2">00 normal register access(load or store) (not extra instruction) (by default)
01 normal register access(load or store) (with Join) (extra instruction)
10 normal register access(load or store) (with Exit)
11 immediate</td></tr>
<tr><td>(34)1</td><td>Used for….</td><td colspan="2">0</td></tr>
<tr><td>(35)1</td><td>destination type</td><td colspan="2">0 = Register destination</td></tr>
<tr><td>(36-37) 2</td><td>Predicate register set (enabling a new flag) or Not used</td><td colspan="2">C0 = 00 (by default)
C1 = 01
C2 = 10
C3 = 11</td></tr>
<tr><td>(38) 1</td><td>Set predicate register</td><td colspan="2">1 = Enable the setting of a predicate register</td></tr>
<tr><td>(39 – 43) 5</td><td>predicate_condition</td><td colspan="2">
<table>
<tr><th>encoding</th><th>Name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>Never</td><td>always false (not used)</td><td>0</td></tr>
<tr><td>0x01</td><td>L</td><td>less tan</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>E</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>Le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>G</td><td>greater tan</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>Lg</td><td>less or greater tan / not equal</td><td>~Z</td></tr>
<tr><td>0x06</td><td>Ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>Lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>U</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>Lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>Eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>Leu</td><td>not greater than</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>Gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>Lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>Geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>O</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>C</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>A</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>S</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>Ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>Na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>Nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>No</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(44 - 45) 2</td><td>Input predicate register to compare before to operate</td><td colspan="2">C0 = 00
C2 = 10</td></tr>
<tr><td>(46 – 60)</td><td>Not used</td><td colspan="2">000 0000 0000 000</td></tr>
<tr><td>(61 – 63) 3</td><td>Sub_op_code</td><td colspan="2">011 LG2</td></tr>
</table>

EX2 instruction:

Checked Not implemented

This instruction calculates the logarithm in a binary base of an input operand.
Destiny_f ← Log_2 (Source_f)

Mnemonics:
Direct EX2 **EX2** R**y,** R**x**

Example (SASS from NVCC):
EX2 R3, R3; (C0000780 9000060d)
EX2 R1, R2; (C0000780 90000405)

(SASS_assembly_lib):
Not_available

Note:
No comments.

<table>
<tr><th>Bit(s)</th><th>Mnemonics</th><th colspan="2">Commentary</th></tr>
<tr><td>0</td><td>instr_is_long</td><td>0 = 32 bit long.</td><td>1 = 64 bit long. (by default)</td></tr>
<tr><td>1</td><td>instr_is_flow</td><td>0 = Normal ins. (by default)</td><td>1 = System ins. (flow control)</td></tr>
<tr><td>(2 – 8) 7</td><td>Destiny_General_purpose_register</td><td colspan="2">R0= 000000, R5=000101, R6=000110…</td></tr>
<tr><td>(9 – 15) 7</td><td>Source_operand_register: it should be a general purpose register.</td><td colspan="2">R0= 000000…, R5= 00101, R6= 00110 …</td></tr>
<tr><td>(16-27) 12</td><td>Not used</td><td colspan="2">000000000000</td></tr>
<tr><td>(28 – 31) 4</td><td>Instruction Op. Code</td><td colspan="2">EX2_OP = 0x9h</td></tr>
<tr><td>(32 - 33) 2</td><td>instr_marker</td><td colspan="2">00 normal register access(load or store) (not extra instruction) (by default)
01 normal register access(load or store) (with Join) (extra instruction)
10 normal register access(load or store) (with Exit)
11 immediate</td></tr>
<tr><td>(34)1</td><td>Used for….</td><td colspan="2">0</td></tr>
<tr><td>(35)1</td><td>destination type</td><td colspan="2">0 = Register destination</td></tr>
<tr><td>(36-37) 2</td><td>Predicate register set (enabling a new flag) or Not used</td><td colspan="2">C0 = 00 (by default)
C1 = 01
C2 = 10
C3 = 11</td></tr>
<tr><td>(38) 1</td><td>Set predicate register</td><td colspan="2">1 = Enable the setting of a predicate register</td></tr>
<tr><td>(39 – 43) 5</td><td>predicate_condition</td><td colspan="2">
<table>
<tr><th>encoding</th><th>Name</th><th>Description</th><th>condition formula</th></tr>
<tr><td>0x00</td><td>Never</td><td>always false (not used)</td><td>0</td></tr>
<tr><td>0x01</td><td>L</td><td>less tan</td><td>(S & ~Z) ^ O</td></tr>
<tr><td>0x02</td><td>E</td><td>Equal</td><td>Z & ~S</td></tr>
<tr><td>0x03</td><td>Le</td><td>less than or equal</td><td>S ^ (Z | O)</td></tr>
<tr><td>0x04</td><td>G</td><td>greater tan</td><td>~Z & ~(S ^ O)</td></tr>
<tr><td>0x05</td><td>Lg</td><td>less or greater tan / not equal</td><td>~Z</td></tr>
<tr><td>0x06</td><td>Ge</td><td>greater than or equal</td><td>~(S ^ O)</td></tr>
<tr><td>0x07</td><td>Lge</td><td>Ordered</td><td>~Z | ~S</td></tr>
<tr><td>0x08</td><td>U</td><td>Unordered</td><td>Z & S</td></tr>
<tr><td>0x09</td><td>Lu</td><td>less than or unordered</td><td>S ^ O</td></tr>
<tr><td>0x0a</td><td>Eu</td><td>equal or unordered</td><td>Z</td></tr>
<tr><td>0x0b</td><td>Leu</td><td>not greater than</td><td>Z | (S ^ O)</td></tr>
<tr><td>0x0c</td><td>Gu</td><td>greater than or unordered</td><td>~S ^ (Z | O)</td></tr>
<tr><td>0x0d</td><td>Lgu</td><td>not equal to</td><td>~Z | S</td></tr>
<tr><td>0x0e</td><td>Geu</td><td>not less tan</td><td>(~S | Z) ^ O</td></tr>
<tr><td>0x0f</td><td>always</td><td>always true (by default)</td><td>1</td></tr>
<tr><td>0x10</td><td>O</td><td>Overflow</td><td>O</td></tr>
<tr><td>0x11</td><td>C</td><td>carry / unsigned not below</td><td>C</td></tr>
<tr><td>0x12</td><td>A</td><td>unsigned above</td><td>~Z & C</td></tr>
<tr><td>0x13</td><td>S</td><td>sign / negative</td><td>S</td></tr>
<tr><td>0x1c</td><td>Ns</td><td>not sign / positive</td><td>~S</td></tr>
<tr><td>0x1d</td><td>Na</td><td>unsigned not above</td><td>Z | ~C</td></tr>
<tr><td>0x1e</td><td>Nc</td><td>not carry / unsigned below</td><td>~C</td></tr>
<tr><td>0x1f</td><td>No</td><td>no overflow</td><td>~O</td></tr>
</table>
</td></tr>
<tr><td>(44 - 45) 2</td><td>Input predicate register to compare before to operate</td><td colspan="2">C0 = 00
C2 = 10</td></tr>
<tr><td>(46 – 60)</td><td>Not used</td><td colspan="2">000 0000 0000 000</td></tr>
<tr><td>(61 – 63) 3</td><td>Sub_op_code</td><td colspan="2">110 EX2</td></tr>
</table>

RSQ instruction:

Checked Not implemented

This instruction calculates the reciprocal of the square root of an input operand on 32 bits single-precision floating-point.
Destiny_f ← SRQ (Source_f)

Mnemonics:
Direct **RSQ** Ry, Rx

Example (SASS from NVCC):
RSQ R0, R0; (40000780 90000009)
RSQ R3, R0; (40000780 9000000d)

(SASS_assembly_lib):
Not_available

Note:
No comments.

Bit(s)	Mnemonics	Commentary	
0	instr_is_long	**0 =** 32 bit long.	**1 =** 64 bit long. (**by default**)
1	instr_is_flow	**0 =** Normal ins. (by default)	**1 =** System ins. (flow control)
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110…	
(9 – 15) 7	Source_operand_register: it should be a general purpose register.	R0= 000000…, R5= 00101, R6= 00110 …	
(16-27) 12	Not used	000000000000	
(28 – 31) 4	Instruction Op. Code	RSQ_OP = 0x9h	
(32 - 33) 2	instr_marker	**00** normal register access(load or store) (not extra instruction) **(by default)** 01 normal register access(load or store) (with **Join**) (extra instruction) 10 normal register access(load or store) (with **Exit**) 11 immediate	
(34)1	Used for….	0	
(35)1	destination type	0 = Register destination	
(36-37) 2	Predicate register set (enabling a new flag) or Not used	C0 = **00 (by default)** C1 = 01 C2 = 10 C3 = 11	
(38) 1	Set predicate register	**1** = Enable the setting of a predicate register	
(39 – 43) 5	predicate_condition	(see table below)	
(44 - 45) 2	Input predicate register to compare before to operate	**C0 = 00** **C2 =** 10	
(46 – 60)	Not used	000 0000 0000 000	
(61 – 63) 3	Sub_op_code	**100 RSQ**	

encoding	Name	Description	condition formula
0x00	Never	always false (not used)	0
0x01	L	less tan	(S & ~Z) ^ O
0x02	E	Equal	Z & ~S
0x03	Le	less than or equal	S ^ (Z | O)
0x04	G	greater tan	~Z & ~(S ^ O)
0x05	Lg	less or greater tan / not equal	~Z
0x06	Ge	greater than or equal	~(S ^ O)
0x07	Lge	Ordered	~Z | ~S
0x08	U	Unordered	Z & S
0x09	Lu	less than or unordered	S ^ O
0x0a	Eu	equal or unordered	Z
0x0b	Leu	not greater than	Z | (S ^ O)
0x0c	Gu	greater than or unordered	~S ^ (Z | O)
0x0d	Lgu	not equal to	~Z | S
0x0e	Geu	not less tan	(~S | Z) ^ O
0x0f	**always**	**always true (by default)**	**1**
0x10	O	Overflow	O
0x11	C	carry / unsigned not below	C
0x12	A	unsigned above	~Z & C
0x13	S	sign / negative	S
0x1c	Ns	not sign / positive	~S
0x1d	Na	unsigned not above	Z | ~C
0x1e	Nc	not carry / unsigned below	~C
0x1f	No	no overflow	~O