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 ${\bf FlexGripPlus}~{\bf GPGPU}$ model can be downloaded from:

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













Content

TABLES OF SASS INSTRUCTIONS SUPPORTED IN FLEXGRIPPLUS	5
Table 1 Control-flow instructions supported in FlexGripPlus.	
Table 2 Arithmetic and logic instructions in FlexGripPlus	
Table 3 Data handling and memory instructions in FlexGripPlus	
Table 4 Floating Point Unit (FPU) instructions supported in FlexGripPlus.	
Table 5 Special function unit (SFU) instructions supported in FlexGripPlus.	
INSTRUCTIONS	
Control-flow Instructions	
BRA instruction:	
BAR instruction:	
RET instruction:	
SSY instruction:	
NOP instruction:	12
TRAP instruction:	
CAL instruction:	
Arithmetic and logic instructions	
I2I instruction (CVT):	16
IMUL Instruction:	18
IMUL32 Instruction:	19
IMUL32I Instruction:	20
SHL/SHR Instructions:	21
IADD Instruction:	
IADD32 Instruction:	24
IADD32I Instruction:	25
IMAD Instruction:	26
IMAD32 Instruction:	
IMAD32I Instruction:	
LOP Instruction:	
ISET Instruction:	
Data handling and memory instructions	
MVC Instruction:	
GLD Instruction:	
GST Instruction:	
MOV Instruction: (check final details)	
MOV32 Instruction:	
MVI Instruction:	
R2G Instruction:	40
A2R Instruction:	42
ADA Instruction:	43
Floating point instructions	44
Floating point instructions FADD32 Instruction:	45
FADD Instruction:	
FADD32I Instruction:	47
FMUL Instruction:	
FMUL32 Instruction:	
FMUL32I Instruction:	
FMAD32 Instruction:	
FMAD32I Instruction:	
F2F Instruction:	
F2I Instruction:	
I2F Instruction:	
FSET Instruction:	
RCP Instruction:	
RCP32 Instruction:	
Especial function unit instructions	
SIN instruction:	
COS instruction:	
RRO instruction: (Range Reduction Operation) LG2 instruction:	
EX2 instruction:	
PSO instruction:	00 67

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.

	- -	<u> </u>	
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
121	Integer to integer conversion	I2I.U32.U16/S16 RZ, RX(L H) / g[].U16
		121.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 \mid H), RX(L \mid 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[], lmm
IMAD/	Integer multiply and	IMAD.U16/ S16 RZ, RX(L H), RY(L H), RW
	Add	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	LOP.AND/OR/XOR/PASS_B RZ, RX/ g[], RY
	Operation	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 /	Floating point addition	FADD32 Rx, Ry / g[Ax + Imm], Rz
FADD /		FADD.COND Rx (Cx), Ry, -Rz / c[0xX][Imm]
FADD32I		FADD32I Rx, Ry, Imm
FMUL/	Floating point multiplication	FMUL Rx(Cx.COND), Ry / g [Ax + Imm], Rz / c[0xX][Imm]
FMUL32 /		FMUL32 Rx, Ry / g [Ax+Imm], Rz
FMUL32I		FMUL32I Rx, Ry, Imm
FMAD /	Floating point multiply and addition	FMAD Rx, Ry / - g [Ax+Imm], Rz / c[0xX][Imm], Rw
FMAD32 /		FMAD32I Rx, -Ry, Imm, Rz
FMAD32I		
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 /	Reciprocal value	RCP Ry (Cx.COND), Rx
RCP32		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 CO.NE, 0x1e0

Example (SASS from NVCC):

BRA Oxf0 (1001e003 00000780) BRA CO.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	Commen	tary						
0	instr_is_long	0 = 32 bit lo		1 = 6	64 bit long. (by default)				
1	instr_is_flow	0 = Normal ins.		1 = 5	System ins. (flow control) (by default)				
(2-8)7	Not used	000 0000 (by default)							
(9 – 26)	Branch Address Low Part 18 bits		The Branch Address (24bits) is divided as HB(2)-MB(8bits)-LB(8 bits)						
18			24 down to 9						
(27) 1	Not used	0							
(28 - 31) 4	Instruction Op. Code	BRA_OP :	= 0 x 1 h						
(32 - 33) 2	instr_marker	00 norma	al regist	er access(load or store) (no	ot extra instruction) (by default)				
			_		th Join) (extra instruction)				
			_	er access(load or store) (w	ith Exit)				
		11 immed	diate						
(34 – 37) 4	Not used	0000							
(38)1	modifier 1	0 (by def							
(39 – 43) 5	predicate condition - selects a boolean	encoding		Description	condition formula				
	function of the \$c register	0x00	never	always false	0 (overflow) not used				
		0x01		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		less or greater than / not e					
		0x06	ge	greater than or equal					
		0x07	lge	Ordered	~Z ~S				
		0x08	u	Unordered	Z & S				
		0x09	lu	less than or unordered					
		0x0a	eu	equal or unordered	Z (working)				
		0x0b	leu	not greater than	Z (S ^ O)				
		0x0c	gu	greater than or unorder	` ' '				
		0x0d	Igu	not equal to not less tan	~Z S				
		0x0e 0x0f	geu always		(~S Z) ^ O 1				
		0x10	o O	Overflow	0				
		0x10 0x11	С	carry / unsigned not belo					
		0x11 0x12	a	unsigned above	~Z & C				
		0x12 0x13	S	sign / negative	S				
		0x1c	ns	not sign / positive	~S				
		0x1d	na	unsigned not above	Z I ~C				
		0x1e	nc	not carry / unsigned bel	·				
		0x1f	no	no overflow	~0				
(44 – 45)	Predicate_register	C0 = 00 (I	ov defa						
2		C1 = 01	,	,					
		C2 = 10							
		C3 = 11							
(46-51) 6	Branch Address High Part 6 bits	The Bran	ch Addr	ess (24bits) is divided as: F	IB(6) 51 down to 46 (NOT supported by				
		GPGPU-FLEXGRIP) 000000							
(52 – 63)	Not Used	0000 000	0 0000						

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			
(00 00) 0		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			
(32 - 33) 2	instr_marker	01 normal register access(load or sto	re) (with Join) (extra instruction)		

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:

RFT

RET Cx (COND) (predicate condition)

Example (SASS from NVCC):

RET (00000780 30000003) RET CO.NE (00000280 30000003) RET CO.EQ (00000100 30000003) RET CO.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*.

0	instr_is_long	· · · · · · · · · · · · · · · · · · ·		t long. (By default)			
1	instr_is_flow	•			m ins. (flow contro	l) (By default)	
(2 – 27) 26	Not Used		0000 0000 0000 0000 0000 000				
(28 – 31) 4	Instruction Op. Code	RET = 0x3	3				
(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 - 38) 5	Not used	00000					
(39 – 42) 5	predicate_condition	encoding	name	Description	condition formula		
		0x00	never	always false	0		
		0x01		less than	(S & ~Z) ^ O		
		0x02	е	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		greater than or unordered			
		0x0d	lgu	not equal to	~Z S		
		0x0e	geu	not less tan	(~S Z) ^ O		
			always		1		
		0x10	0	Overflow	0		
		0x11	С	carry / unsigned not below			
		0x12	а	unsigned above	~Z & C		
		0x13	S	sign / negative	S		
		0x1c	ns	not sign / positive	~\$		
		0x1d	na	unsigned not above	Z ~C		
		0x1e		not carry / unsigned below			
		0x1f	no	no overflow	~0		
(45-44)	Predicate_register	C0 = 00 (I	by defa	ult)			
		C1 = 01					
		C2 = 10					
(100		C3 = 11					
(46 - 63)	Not used	0000 0000 0000 0000					

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)

ins.

Not used

(46 - 51) 6

(52 - 63)12

Offset Address HP

Offset: (24-9) 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.	1 = Flow-control System ins. (by default)
(2-8) 7	Not used	0000 000	
(9-24) 16	Offset Address LP	Code address divided as: HP (6 bits) – LP (16	bits)
(25 - 27) 3	Not used	000	
(28-31) 4	Instruction Op. Code	SSY_OP = 0xA	

(32 - 33) 2	instr_marker	00 normal reg Access(load or store) (not extra instruction) (by default)					
		01 norma	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	00000					
(39 – 43) 5	predicate condition for the	encoding	name	Description	condition formula		
	associated \$c Predicate register	0x00	never	always false	0		
l							

	associated \$c Predicate register	0x00	never	always false	0
		0x01	I	less tan	(S & ~Z) ^ O
		0x02	е	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 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	О	Overflow	0
		0x11	С	carry / unsigned not below	С
		0x12	а	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	~0
(44 - 45)	source \$c register, not used in this	00			

(not implemented in SASS_assembly_lib)

0000 0000 0000

High part (Not implemented in the original version of FLEX-GRIP or extended one)

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 (0000000 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) 				
(34 – 60) 26	Not Used	10 normal reg Access(load or store) (with Exi				

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				
		11 immediate				
(34 – 60) 26	Not Used	11 immediate 0000 0000 0000 0000 0000 0000 00				

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:

121.(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_00x7f(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

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)

Ri+(c)	Mnemonics	Commentary					
Bit(s)		0 = 32 bit			1 = 64 bit long. (by default)		
0	instr_is_long			by default)	<u> </u>		
1 (2 8) 7	instr_is_flow			by default)	1 = System ins. (flow control)		
(2-8)7	Destiny_Register_or_memory_space_address	R0= 0000	00, 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 բ			0 = Low part (RxL)		
(10 - 15) 6	Source_1 (register)	R0= 0000					
(16 - 22)	1_Register_Operand (16 bits, 32 bits)	R0= 0000	00, R5=	:000101, R6=000110			
6							
(23) 1	Not used	0					
(24) 1	Not used	0					
(25) 1	Not used	0					
(26 – 27)	Address register low 2 bits	00					
2							
(28 - 31)	Instruction Op. Code	12I = 0xA					
4							
	!asta assalas	000		Asses(leading)	and the desired of the state of		
(32 - 33)2	instr_marker		_	Access(load or store) (not	extra instruction)		
		(by defau	-	Assessition to the second	a tatal factor to the state of		
			_	Access(load or store) (with			
			_	Access(load or store) (with	1 EXIT)		
(2.1)		11 = imm	ediate				
(34)1	Not used	0			1		
(35)1	destination type	0 = Regist			1= Memory destination		
(36-37)2	Predicate register set (enabling a new flag)	CO = 00 (k	y defa	ult)			
		C1 = 01					
		C2 = 10					
		C3 = 11					
(38)1	Write enable / set predicate register			d (just for memory destina	-		
				cate register set, 0 = disab	le predicate register set		
(39 - 43)	predicate_condition	encoding	name		condition formula		
		0x00	never	always false (not used			
		0x01	- 1	less tan	(S & ~Z) ^ O		
		0x02	е	Equal	Z & ~S		
		0x03	le	less than or equal	S ^ (Z O)		
		0x04	g	greater tan	~Z & ~(S ^ O)		
		0x05		less or greater tan / not e			
		0x06	ge	greater than or equal			
		0x07	lge	Ordered	~Z ~S		
		0x08	u	Unordered	Z & S		
		0x09	lu	less than or unordered			
		0x0a	eu	equal or unordered	Z		
		0x0b	leu	not greater than	Z (S ^ O)		
		0x0c		greater than or unorder			
			gu				
		0x0d	lgu	not equal to	~Z S		
		0x0e	geu	not less tan	(~S Z) ^ O		
			always				
		0x10	0	Overflow	0		
		0x11	С	carry / unsigned not belo			
		0x12	а	unsigned above	~Z & C		
		0x13	S	sign / negative	S		
		0x1c	ns	not sign / positive	~\$		
		0x1d	na	unsigned not above	Z ~C		
		OXIU	114		·		
		0x1e	nc	not carry / unsigned bel	·		

(45 - 44) 2	Predicate_register	C0 = 00 (by default)					
		C1 = 01					
		C2 = 10					
		C3 = 11					
(45) 1	Not used	0					
(46 - 48) 3	Source_1_data_type	Source_1_data_type	Mem Type	I2I type			
		000	DT_U8	DT_U16			
		001	DT_U16	DT_U32			
		010	DT_S16	DT_U8			
		011	DT_U32	DT_U32			
		100	DT_U32	DT_S16			
		101	DT_U32	DT_S32			
		110	DT_U32	DT_S8			
		111	DT_U32	DT_U32			
49 – 57	Not used	0000 0000					
58	Not used	0					
(59) 1	Size of operands	0: b16		1: b	032		
(60) 1	Signed or unsigned sources (potentially not used)	0: U16/U32 (Unsigned)		1: 9	516/S32 (Signed)		
(61 – 63)3	Sub_op_code	000 (not used)					

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...

D:4/a)	Mananics	Commontany					
Bit(s)	Mnemonics		Commentary 0 = 32 bit long. 1 = 64 bit long			ong (Du defects)	
0	instr_is_long					ong. (By default)	
1 (2 (2) 7	instr_is_flow				<u>·</u>	ins. (flow control)	
(2 – 8) 7	Destiny_ Register			=000101, R6 =0			
(9-15) 6	Source_Register_1:	32 bits:		16 bits:		Shared memory case:	
	It could be a GPRS at 32 or 16 bits,		R0 = 0000000 R0[L] = 000000 0 R5 = 0000101 R5[H] = 000010 1			ffset value (9 - 12)	
	or a shared memory location			R5[H] = 000010		(13)	
		R6 = 0000	110	R6[H] = 000011		(14)	
4.5	Libit to and Backton 2 Consent	0 (15			<u> </u>		
16	High-Low part Register 2 Operand	0 = low part (RxL) 1 = High part (RxH) R0 = 000000, R5 =000101, R6 =000110					
(17 – 22) 6	2_ Register_or_shared_mem _Operand (16 bits))00, R5	=000101, R6 =	000110		
23	Not used	0			4 61 1	•	
24	Source_1_Selector	0 = Regist	er Soui	rce	1 = Shared	Mem.	
25	Not used	0					
(26-27) 2	Address register offset used by the shared memory addressing	A0 = 00	(defau	•	A1 = 01		
		A2 = 10			43 = 11		
28	Size of operands			oits operands			
(29 – 31) 3	Instruction Op. Code	IMUL32 =	0x2				
(32 - 33)2	instr_marker		_	Access(load o	r store) (no	t extra instruction)	
		(by defau	•				
			_	Access(load o	r store) (wi	th Join) (extra	
		instructio					
				Access(load o	r store) (wi	th Exit)	
		11 = imm	ediate				
(34)1	Not used	0			_		
(35)1	destination type	0 = Regist	er dest	ination		nal operation	
(36-37)2	Predicate register set (enabling a new flag) or Not used	CO = 00 (I	oy defa	ult)	C1 = 01		
		C2 = 10			C3 = 11		
(38)1	Set predicate register	1 = enabl	ed pred	licate register	set		
		0 = disab	ed pred	dicate register	set		
(39 – 43) 5	predicate_condition	encoding	nama	Doscr	ription	condition	
		encoung	name	Desci	приоп	formula	
		0x00	never	always fals	e (not used) 0	
		0x01	- 1	less	s tan	(S & ~Z) ^ O	
		0x02	е	Eq	lual	Z & ~S	
		0x03	le	less thar	n or equal	S ^ (Z O)	
		0x04	g	great	er tan	~Z & ~(S ^ O)	
		0.05		less or grea	ter tan / no	ot	
		0x05 Ig less or greater tan / not		eq	ıual	~Z	
			·				
			ge	greater th	an or equal	~(S ^ O)	
ĺ		0x06 0x07	ge Ige	greater the Ord	an or equal lered		
		0x06 0x07	ge Ige u	Ord		~Z ~S	
		0x06 0x07 0x08	lge u	Ord Unor	lered dered	~Z ~S Z & S	
		0x06 0x07 0x08 0x09	lge u lu	Ord Unor less than o	lered dered r unordered	~Z ~S Z & S S ^ O	
		0x06 0x07 0x08 0x09 0x0a	lge u lu eu	Ord Unor less than o equal or i	ered dered r unordered unordered	~Z ~S Z & S d S ^ O Z	
		0x06 0x07 0x08 0x09 0x0a 0x0b	lge u lu eu leu	Ord Unor less than o equal or u not grea	lered dered r unordered unordered ater than	~Z ~S Z & S S ^ O Z Z (S ^ O)	
		0x06 0x07 0x08 0x09 0x0a 0x0b	lge u lu eu leu gu	Ord Unor less than o equal or o not grea greater than	ered dered r unordered unordered ater than or unorder	~Z ~S Z & S d S ^ O Z Z (S ^ O) ed ~S ^ (Z O)	
		0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d	lge u lu eu leu gu lgu	Ord Unor less than o equal or o not greater than not eo	ered rdered r unordered unordered ater than or unorder qual to	~Z ~S Z & S d S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S	
		0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e	lge u lu eu leu gu lgu geu	Ord Unor less than o equal or u not greater than not ed not le	ered r unordered unordered ater than or unorder qual to ess tan	~Z ~S Z & S d S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O	
		0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f	lge u lu eu leu gu lgu geu always	Ord Unor less than or equal or or not greater than not ed not le always true	ered r unordered ater than or unorder qual to ess tan e (by defaul	~Z ~S Z & S d S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1	
		0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10	lge u lu eu leu gu lgu geu always	Ord Unor less than o equal or u not greater than not ed not le always true	ered r unordered unordered ater than or unorder qual to ess tan e (by defaul	~Z ~S Z & S S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1	
		0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11	lge u lu eu leu gu lgu geu always	Ord Unor less than o equal or u not greater than not ed not le always true Carry / unsign	ered r unordered unordered ater than or unorder qual to ess tan e (by defaul rflow ned not belo	~Z ~S Z & S S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1 O O O W C	
		0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x12	lge u lu eu leu gu lgu geu always o c	Ord Unor less than o equal or u not grea greater than not le always true Ove carry / unsign	r unordered ater than or unorder qual to ess tan e (by defaul rflow ned not beleed above	~Z ~S Z & S d S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1 O ow C ~Z & C	
		0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x12 0x13	lge u lu eu leu gu lgu geu always o c a	Ord Unor less than o equal or u not greater than not le always true Ove carry / unsign unsigne sign / r	ered r unordered ater than or unorder qual to ess tan e (by defaul rflow ned not bele ed above negative	~Z ~S Z & S S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1 O ow C ~Z & C S	
		0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x12 0x13 0x1c	lge u lu eu leu gu lgu geu always o c a s ns	Ord Unor less than o equal or u not greater than not ed not le always true Ove carry / unsigne sign / r not sign	r unordered unordered ater than or unorder qual to ess tan e (by defaul rflow ned not belied above negative / positive	~Z ~S Z & S S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1 O Ow C ~Z & C S ~S	
		0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x12 0x13 0x1c 0x1d	lge u lu eu leu gu lgu geu always o c a s ns	Ord Unor less than o equal or u not greater than not le always true Ove carry / unsign unsigne sign / r not sign unsigned	r unordered ater than or unorder qual to ess tan e (by defaul rflow ned not beloed above negative / positive not above	~Z ~S Z & S d S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1 O ow C ~Z & C S ~S Z ~C	
		0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x10 0x11 0x12 0x13 0x1c 0x1d 0x1e	lge u lu eu leu gu lgu geu always o c a s ns na	Ord Unor less than o equal or u not greater than not le always true Ove carry / unsign unsigner sign / r not sign unsigned not carry / ur	r unordered unordered ater than or unorder qual to ess tan e (by defaul rflow ned not beled above negative / positive not above nsigned beled	~Z ~S Z & S S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1 O ow C ~Z & C S ~S Z ~C ow ~C	
		0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x10 0x11 0x12 0x13 0x1c 0x1d 0x1e 0x1f	lge u lu eu leu gu lgu geu always o c a s ns	Ord Unor less than o equal or u not greater than not ed always true Ove carry / unsigne sign / r not sign unsigned not carry / ur no ov	r unordered unordered ater than or unorder qual to ess tan e (by defaul rflow ned not belied above negative / positive not above nsigned beliererflow	~Z ~S Z & S d S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1 O ow C ~Z & C S ~S Z ~C	
(44 - 45) 2	Input predicate register to compare before to operate	0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x12 0x13 0x1c 0x1d 0x1e 0x1f CO= 00	lge u lu eu leu gu lgu geu always o c a s ns na	Ord Unor less than o equal or u not greater than not le always true Ove carry / unsigne sign / r not sign unsigned not carry / ur no ov	runordered ater than or unorder qual to ess tan (by defaulted) above negative / positive not above nsigned belove reflow cerflow	~Z ~S Z & S S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1 O ow C ~Z & C S ~S Z ~C ow ~C	
		0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x10 0x11 0x12 0x13 0x1c 0x1d 0x1e 0x1f C0= 00 C2= 10	lge u lu eu leu gu geu always o c a s ns na nc	Ord Unor less than o equal or u not greater than not le always true Ove carry / unsigne sign / r not sign unsigned not carry / ur no ov	r unordered unordered ater than or unorder qual to ess tan e (by defaul rflow ned not belied above negative / positive not above nsigned beliererflow	~Z ~S Z & S S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1 O ow C ~Z & C S ~S Z ~C ow ~C	
(46 – 52) 7	Not used	0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x10 0x11 0x12 0x13 0x1c 0x1d 0x1e 0x1f C0= 00 C2= 10 000 0000	lge u lu eu leu gu geu always o c a s ns na nc	Ord Unor less than o equal or u not greater than not le always true Ove carry / unsigne sign / r not sign unsigned not carry / ur no ov	runordered unordered ater than or unorder qual to ess tan e (by defaul rflow ned not beloed above negative / positive not above nsigned beloerflow ct = 01 c3 = 11	~Z ~S Z & S S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1 O ow C ~Z & C S ~S Z ~C ow ~C	
(46 – 52) 7 53	Not used Shared memory use for Source_2?	0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10 0x11 0x12 0x13 0x1c 0x1d 0x1e 0x1f C0= 00 C2= 10 000 00000 Yes = 1	lge u lu eu leu gu lgu geu always o c a s ns na nc no	Ord Unor less than o equal or u not greater than not le always true Ove carry / unsigne sign / r not sign unsigned not carry / ur no ov	runordered ater than or unorder qual to ess tan (by defaulted) above negative / positive not above nsigned belove reflow cerflow	~Z ~S Z & S S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1 O ow C ~Z & C S ~S Z ~C ow ~C	
(46 – 52) 7	Not used	0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x10 0x11 0x12 0x13 0x1c 0x1d 0x1e 0x1f C0= 00 C2= 10 000 0000	lge u lu eu leu gu lgu geu always o c a s ns na nc no	Ord Unor less than o equal or u not greater than not le always true Ove carry / unsigne sign / r not sign unsigned not carry / ur no ov	runordered unordered ater than or unorder qual to ess tan e (by defaul rflow ned not beloed above negative / positive not above nsigned beloerflow ct = 01 c3 = 11	~Z ~S Z & S S ^ O Z Z (S ^ O) ed ~S ^ (Z O) ~Z S (~S Z) ^ O t) 1 O ow C ~Z & C S ~S Z ~C ow ~C	

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 = Syste	em ins. (flow control)
(2-8)7	Destiny_ Register	R0= 000000, R5=000101, R6=000110			
(9-15) 6	Source_Register_1:	32 bits:	16 bits:		Shared memory case:
	It could be a GPRS at 32 or 16 bits,	R0 = 0000000	RO[L] = 0000	00 0	Offset value (9 - 12)
	or a shared memory location	R5 = 0000101	R5[H] = 0000	10 1	1 (13)
		R6 = 0000110	R6 = 0000110 R6[H] = 00001		0 (14)
					0 (15)
16	High-Low part Register 2 Operand	0 = low part (R	kxL)	1 = High part (RxH)	
(17 – 21) 5	2_ Register_or_shared_mem _Operand (16 bits)	RO = 000000, F	R5 =000101, R6	=000110.	
22	Operand size	0 = 16 bits		1 = 24 bits	
23	Not used	0			
24	Source_1_Selector	0 = Register Sc	ource	1 = Shared Mem.	
25	Not used	0			
(26-27) 2	Address register offset used by the shared memory addressing	A0 = 00 (defa	ault)	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), (lmm)

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	i=000110				
8	Sign of destiny reg	0 = Unsigned	1 = Signed				
(9-15) 6	Source_Register_1:	32 bits:	16 bits:				
	It could be a GPRS at 32 or 16 bits	R0 = 0000000	RO[L] = 000000 0				
		R5 = 0000101	R5[H] = 000010 1				
		R6 = 0000110	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	X XXXX XXXX XXXX XX				
(60) 1	Not used	0					
(61 - 63)3	Sub op code	000	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	Commen	tarv				
0	instr_is_long	0 = 32 bit			1 = 64 bit long. (Default)		
1	instr is flow	0 = Norm		Default)	1	em ins. (flow control)	
(2-8)7	Destiny Register (32 bits) or H/L + Destiny reg (16 bits)	R0 = 000		,		000000 1 , H = high	
, ,		R5 = 000	0101			000101 1 , L = low	
		R6 = 000	0110		R6 (L) = 000110 0		
(9 – 15)7	Source 1 Register (32 bits) or H/L + Source 1 reg (16 bits)	R0 = 000	0000		R1(H) = (000001 1	
		R5 = 0000101			R5(H) =0	000101 1	
		R6 = 000	0110		R6 (L) = 0	000110 0	
(16 - 20)5	Offset_of shift	This can	be a reg	ister or an imme	diate value	e in Hex.	
(5.1)							
(24) 1	Source 1 Selector	1 = Share	d mem	ory op.	0 = Regis	ter operand	
(25-27) 3	Not Used	000					
(28-31)4	Instruction Op. Code	SHL or SH	1R = 0x3				
(32 - 33)2	instr_marker	00 = norr	nal reg	Access(load or sto	ore) (not e	extra instruction) (by	
(32 33)2		default)			0.0, (including (a)	
		_	nal reg	Access(load or sto	ore) (with	Join) (extra	
		instructio	_	,	, ,	, ,	
		10 = norr	nal reg	Access(load or sto	ore) (with	Exit)	
		11 = imm	ediate				
34	Used for	0					
35	destination type	0 = Regis	ter dest	ination			
(36 - 38) 3	Nor used	000		Ī		1	
(39 – 43) 5	predicate condition to operate the instruction	encoding		Descripti		condition formula	
		0x00	never	always fa		0	
		0x01	l	less tar		(S & ~Z) ^ O	
		0x02	е	Equal		Z & ~S	
		0x03	le	less than or	<u> </u>	S ^ (Z O)	
		0x04	g	greater t		~Z & ~(S ^ O)	
		0x05	lg	less or great		~Z	
		0x06	ge	greater than o		~(S ^ O)	
		0x07	lge	Ordere		~Z ~S	
		0x08	u	Unorder		Z & S	
		0x09	lu	less than or un		S ^ O	
		0x0a	eu	equal or uno		Z - (2.1.2)	
		0x0b	leu	not greate		Z (S ^ O)	
		0x0c	Ŭ	greater than or u		` ' ′	
		0x0d	lgu	not equa		~Z S	
		0x0e 0x0f	geu	not less t		(~S Z) ^ O	
		0x01	always	always true (by Overflo		0	
		0x10	0	carry / unsigned			
		0x11	c a	unsigned a		~Z & C	
		0x12	S	sign / nega		S	
		0x1s	ns	not sign / po		~S	
		0x1d	na	unsigned not		Z ~C	
		0x1e	nc	not carry / unsig		'	
		0x1f	no	no overfl		~0	
(44 - 45) 2	Input_predicate_register	CO = 00		373111	C1 = 01	<u> </u>	
(., .5, 2	Used as: precondition to operate the instruction	C2 = 10			C3= 11		
(46-51) 6	Not used	00 0000			_ _		
52	Source 2 selector	1 = Imme	diate va	alue	0 = Regis	ter	
(53-57) 5	Not used	0 0000					
58	Size of destiny and source	1 = 32 bit	:S		0 = 16 bi	ts	
59	Use of the Sign during the shift	1 = Signe	d		0 = Unsig	gned	
60	Not Used	0					
61	Operation code of the shift (Left or Right)	0 = SHL			1 = SHR		
(62 – 63) 2	Sub_opcode	11					

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):

employed

Formats: pending

51:()					
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:	Shared memory case:		
		R0 = 00000	Offset value (9 - 12)		
		R5 = 00101	1 (13)		
		R6 = 00110	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 + So	urce2 + Carry_in		
		Denoted as (IADD.CARRY0)	in the mnemonic		
(23) 1	Not Used	0			
(24) 1	Second_source _operand_type	1 = Constant memory	0 = General purppose		
		C[0x01][0x XX]	Register		
(25-27) 3	Not Used	000			
(28) 1	Carry_in	If carry_in = 1 then perform	the operation:		
		Dest = Source1 + So	urce2 + Carry_in		
		Denoted as (IADD.CARRY0)	in the mnemonic		
(29-31)3	Op_code	(001) IADD			

(22 22)2	inche maniform	00		A/l-		/tt.	in atmosticus)	
(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							
(34)1	destination type	0 1						
(36-37) 2	Predicate register to be set (enabling a new flag, only	0 = Register destination C0 = 00 (by default)						
(30-37) 2	carry)	C1 = 01	by uela	uitj				
	carryy	C1 = 01 C2 = 10						
		C2 = 10 C3 = 11						
(38) 1	Set predicate register	1 = Enabl	e nredi	cate regis	ter set	0 = Disa	able predicate regis	ster set
(39 – 43)5	predicate_condition	encoding		cate regio	Description		condition formula	oter set
,5	predicate_committee	0x00	never	alway	s false (not ι		0	
		0x01	I	aa	less tan	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	(S & ~Z) ^ O	
		0x02	е		Equal		Z & ~S	
		0x03	le	les	s than or equ	ıal	S ^ (Z O)	
		0x04	g		greater tan		~Z & ~(S ^ O)	
		0x05	lg	less or gr	eater tan / n	ot equal	~Z	
		0x06	ge		ter than or e		~(S ^ O)	
		0x07	lge		Ordered	•	~Z ~S	
		0x08	u		Unordered		Z & S	
		0x09	lu	less than or unordered		lered	S ^ O	
		0x0a	eu	equ	al or unorde	red	Z	
		0x0b	leu	nc	t greater tha	n	Z (S ^ O)	
		0x0c	gu	greater	than or uno	rdered	~S ^ (Z O)	
		0x0d	lgu		not equal to		~Z S	
		0x0e	geu		not less than		(~S Z) ^ O	
		0x0f	always	alway	s true (by de	fault)	1	
		0x10	0		Overflow		0	
		0x11	С	carry /	unsigned not	below	С	
		0x12	a	uı	nsigned abov	re	~Z & C	
		0x13	S	S	ign / negative	е	S	
		0x1c	ns		t sign / positi		~\$	
		0x1d	na		igned not ab		Z ~C	
		0x1e	nc		ry / unsigned	below	~C	
		0x1f	no		no overflow		~O	
(44 - 45) 2	Input_predicate_register	CO = 00				C1= 01		
	Used as: precondition to operate the instruction	C2= 10				C3 = 11	-	
(46 - 53) 8	Source_register_2: It could be coming from:	Register	-	6-52) :	Constant m	-	Shared memo	•
	1) GPRS	R0 = 0000			Second part		(46-52): 000 (
	2) Constant memory	R5= 0010			constant me	-		τ snared
	3) Shared memory	R6= 0011	.U		C[0x2][0x16]	-	memory)	
/E/ E7\/	The first part of the Source 2 when constant many and	(53) 0	oort of	the same	(46-53) = 00			
(54 – 57)4	The first part of the Source_2 when constant memory is	The first		the const	ant memory	(1.6.)		

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:	Shared memory case:
		R0 = 00000	Offset value (9 - 12)
		R5 = 00101	1 (13)
		R6 = 00110	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

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**

(61 - 63)

Sub_opcode

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

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

Bit(s)	Mnemonics	Commen						
0	instr_is_long	0 = 32 bit		-		long. (Default)		
1 (2 2) 7	instr_is_flow	0 = Norm		Default)		m ins. (flow control)		
(2-8)7	Destiny Register (32 bits)	R0 = 000			• •	00000 1 , H = high		
		R5 = 000				00101 1 , L = low		
(0. 15)7	Source 1 Register (22 hits) or	R6 = 000		P1(H) = 000	R6 (L) = 000110 0 0001 1 Shared memory case:			
(9 – 15)7	Source 1 Register (32 bits) or H/L + Source 1 reg (16 bits)	R5 = 000		R1(H) = 000		Offset value (9 - 12)		
	Or immediate Shared memory	R6 = 000		R5(H) =000 R6 (L) = 000		0 (13) (16 bits address)		
	of infinediate shared memory	NO – 000	3110	KG (L) = 000		1 (14)		
						g[0xoffset]		
(16 – 22) 7	Source 2 Register (32 bits) or	R0 = 000	2000		R1(H) = 0			
(10 22)	H/L + Source 1 reg (16 bits)	R5 = 000			$\mathbf{R5(H)} = 0$			
	1,72 * 554.55 2 1 56 (25 5.15)	R6 = 000			$\mathbf{R6} (\mathbf{L}) = 0$			
24	Not used	0			- () -			
25	Address Register or Imm address	0 = Imme	diate ac	ddress	1 = Addre	ess Register		
(26-27) 4	Address reg [1-0] Ax	A0 = 00	(by defa	ault)	A1 = 01			
		A2 = 10		•	A3 = 11			
28	Size of operands	0 = 16 or	32 bits	operands	1 = 24 or	32 bits operands		
(23 - 31) 3	Instruction Op. Code	IMAD = 0				·		
(32 - 33)2	Instr_marker		nal reg /	Access(load or	store) (no	t extra instruction) (by		
		default)						
				Access(load or	store) (wi	th Join) (extra		
		instructio	-					
			_	Access(load or	store) (wi	th Exit)		
		11 = imm	ediate					
34	Not used (Address reg [2])	0						
35	destination type					rnal operation only		
(36 - 37)2	Register to be set as result of operation if enabled				01 = C1			
20		10 = C2						
38	Set predicate register as result of operation					e predicate register set		
(20, 42) 5	Duadinate andition to annuate the instruction	set						
(39 - 43)5	Predicate condition to operate the instruction	encoding	1	Descri	•	condition formula		
		0x00	never	always		0		
		0x01	ı	less		(S & ~Z) ^ O		
		0x02	е	Equ		Z & ~S		
		0x03	le	less than	<u> </u>	S ^ (Z O)		
		0x04	g	greate		~Z & ~(S ^ O)		
		0x05	lg	less or gr		~Z		
		0x06	ge	greater tha				
		0x07	lge	Orde		~Z ~S		
		0x08	u	Unord	dered	Z & S		
		0x09	lu	less than or	unordered	d S^O		
		0x0a	eu	equal or u	nordered	Z		
		0x0b	leu	not grea	ater tan	Z (S ^ O)		
		0x0c	gu	greater than o	or unorder	ed ~S ^ (Z O)		
		0x0d	lgu	not eq	ual to	~Z S		
		0x0e	geu	not le		(~S Z) ^ O		
		0x0f	always	•				
		0x10	0	Over	flow	0		
		0x11	С	carry / unsign				
		0x12	а	unsigne	d above	~Z & C		
		0x13	S	sign / n	egative	S		
		0x1c	ns	not sign /	positive	~\$		
		0x1d	na	unsigned i	not above	Z ~C		
		0x1e	nc	not carry / un	signed bel	ow ~C		
		0x1f	no	no ove	erflow	~0		
(44 - 45) 2	Input_predicate_register	CO = 00	(by de	fault)	C1 = 01	1		
	Used as: precondition to operate the instruction	C2= 10			C3= 13	1		
(46 -52) 7	3_ Register_Operand (32 bits)	R0= 000000, R5=000101, R6=000			00110			
53	Source 1 selector	1 = Shared memory			0 = Re	gister		
(54 –55) 2	Not used	00						
56	Source 2 Immediate value?	1= Imme	diate va	lue	0 = Re	gister		
57	Not used							
58	Sign of the Source 1	1 = Nega			0 =Pos			
59	Sign of the Source 3	1 = Nega	tive		0 =Pos	sitive		
60	Not Used	0						
(61-63)3	Sub_opcode	000						
			·					

IMAD32 Instruction:



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

PrE: $Rz \leftarrow (Ry * Rx) + Rz$

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	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=0	00101, R6=000110				
(9 – 15)7	Source_Register_1:	32 bits:	16 bits:				
	It should be a general purpose register at 16 or 32 bits	R0 = 0000000	RO(L)= 000000 0				
		R5 = 0000101	R5(H) = 000101 1				
		R6 = 0000110	R6(H)= 000110 1				
(16-22) 7	Source_Register_2:	32 bits:	16 bits:				
	It should be a general purpose register at 16 or 32 bits	R0 = 0000000	RO(L)= 000000 0				
		R5 = 0000101	R5(H)= 000101 1				
		R6 = 0000110	R6(H)= 000110 1				
(23-27) 5	Not used	0 0000					
(28 - 31) 4	Instruction Op. Code	IMAD32 = 0x6					

IMAD32I Instruction:



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, R	6 =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:	16 bits:				
		R0 = 0000000	RO(L) = 000000 0				
		R5 = 0000101	R5(H) = 000101 1				
		R6 = 0000110	R6(H) = 000110 1				
(16-21) 6	The low part of the immediate value $(5-0)$	Low_Imm XX XXXX	Low_Imm XX XXXX				
(22-27) 3	Not Used	00 0000	00 0000				
(28-31)4	Instruction Op. Code	IMADD32I = 0x6					
(32 - 33)2	instr_marker	00 = normal reg Access(loa	d or store) (not extra instruction)				
		01 = normal reg Access(loa	d or store) (with Join) (extra				
		instruction)					
		10 = normal reg Access(load or store) (with Exit)					
		11 = immediate (by defaul	t)				
(34 – 59)	The high part of the immediate value	High_Imm XXXX XXXX XXXX	(XXXX XXXX XX				
26							
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	Commen				(5.6.10)	
0	instr_is_long	0 = 32 bit		Defects)	1 = 64 bit long	<u> </u>	
1 (2 8) 7	instr_is_flow	0 = Norm		:000101, R6 =0		. (flow control)	
(2-8)7 (9-15)7	Destiny_Address_Register (32 bits) Source 1: It could be a GPRS or a shared memory location	Register	•	000101, Kb =0	Shared memo	.rv c2c0:	
(9 – 15) /	Source 1. It could be a GPRS of a Shared memory location	R0 = 0000			Offset value (9	•	
		R5 = 0010			1 (13)	9 - 12)	
		R6 = 0011			1 (13)		
		10-0011	.0		g[0xc	offsetl	
(16 – 22) 7	Source_2:	Register	case:		Second part of		
(10 22)	It can be a general purpose register or a constant memory location	R0 = 0000			memory (i.e.)	the constant	
		R5 = 0010			C[0x2][0x16]		
						(16-22) = 01 0110	
(23 - 27)5	High part of the Source_2 or configuration options:	Shared m	nemory		First part of th		
,		(23-25) 0	-		memory (i.e.)		
		(26-27)	Address	register	C[0x2][0x16]		
		part of th	ie addre	ess (i.e.)	(23-26) = 0010	0 0	
		g [A2 +0x	1] = 10				
		options a					
			= 00, A				
			= 10 , A	3 = 11			
(28 - 31)4	Instruction Op. Code	LOPS = 0x	κD				
100		6.0					
(32 - 33)2	instr_marker				r store) (not ext		
			_	Access(load or	r store) (with Jo	in) (extra	
		instructio	•	A (I I		•• \	
			normal reg Access(load or store) (wi immediate (by default)			it)	
2.4	High Address of 2 anarond	0	ediate (ру аетаціт)			
34 35	High_Address of 2_operand		tor doct	ination	doctination.		
	destination type Prodicate register set (analyting a new flag) or Net used	0 = Regist				destination	
(36 - 37) 2	Predicate register set (enabling a new flag) or Not used	10 = C0 (•	uit)	11 = C1		
(38) 1	Set predicate register			ate register	edicate register		
(30) 1	Set predicate register	set	predica	set		edicate register	
(39 – 43)5	predicate condition to be considered to execute the instruction if	361			361	condition	
(33 – 43)3	the input predicate comparison is active	encoding	name	Desc	ription	formula	
	the input predicate companson is active	0x00	never	always fals	se (not used)	0	
		0x01	ı	,	s tan	(S & ~Z) ^ O	
		0x01 0x02			qual	Z & ~S	
		0x02 0x03	e			S ^ (Z O)	
			le		n or equal	· · · ·	
		0x04	g		ter tan	~Z & ~(S ^ O)	
		0x05	lg	_	nter tan / not qual	~Z	
		0x06	go.		an or equal	~(S ^ O)	
		0x07	ge		lered	~Z ~S	
		0x07	lge 		rdered	Z & S	
			u				
		0x09	lu		r unordered	S ^ O Z	
		0x0a	eu		unordered		
		0x0b	leu		ater than	Z (S ^ O)	
		0x0c	gu	ŭ	or unordered	~S ^ (Z O)	
		0x0d	lgu		qual to	~Z S	
		0x0e	geu		ess tan	(~S Z) ^ O	
			always	-	e (by default)	1	
		0x10	0		erflow	0	
		0x11	С		ned not below	C	
		0x12	а	ŭ	ed above	~Z & C	
		0x13	S	<u> </u>	negative	S	
		0x1c	ns		/ positive	~S	
		0x1d	na		not above	Z ~C	
		0x1e	nc	•	nsigned below	~C	
		0x1f	no	no ov	verflow	~0	
(44 – 45) 2	Input predicate register to compare before to operate	CO = 00			C1 = 01		
1.5		C2 = 10			C3 = 11		
(46 – 47) 2	Logic_operation_selector	AND = 00			OR = 01		
145		XOR = 10			NOT = 11		
(48 – 49) 2	Not used	00					
50	Source 1 inverted			ot inverted no			
51	Source 2 inverted	1= invert	ed 0= n	ot inverted <mark>no</mark>	t working		
52	Not used	V :			A1 2		
53	Shared memory use for Source_2?	Yes = 1			No = 0 regist		
54	Use of constant memory as Source_2?	Yes = 1		11	No = 0 regist	er use	
(55 - 56) 2	Index of the Constant memory space c[xx][xx]	UU (not s	upporte	ed in FLEXGRIF	PLUS)		

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)

Bit(s)	Mnemonics	Commen	itary						
0	instr_is_long	0 = 32 bit				ong. (by default)			
1 (2 (2) 7	instr_is_flow			by default)		ins. (flow control)			
(2 – 8) 7	Destination_register			1 = 000 0001, R2 = 0 l operation	000 0010, R :	15 = 000 1111 or			
(9 – 15)7	Operand Source 1			r or data for Memo	orv				
(5 15),	operana source 1	For Register: Register number in the Core							
		For Memory: Address of memory (ej. [0x7f])							
(16-22) 7	Operand Source 2		•	r or data for Memo	•				
				gister number in th		(low part) [][VVVV	/ 1		
(23) 1	Operand Source 2 Selector	0 = regist	-			(low part) [][XXXXX ry location source	Nj		
(24-27) 4	Not used	0000	ici odai		2 10101101	y location source			
(28 – 31) 4	Instruction Op. Code	ISET_OP	= 0x3						
(32 - 33) 2	instr_marker	00 norma	al reg A	ccess(load or store) (not extra	instruction) (by defa	ıult)		
			•	•		(extra instruction)			
		10 norma	•	ccess(load or store) (with Exit)				
(35)1	Selection of the output comparison register:			ut register	1 = Use the	e internal register o [00x7F1		
(33)1	Destiny Register or Internal Register o[0x7F]	G doc ti	ne outp	at register	2 050 1110	internal register o [oom, ,		
(36 – 37) 2	Predicate register to be set after comparison	CO = 00 (by defa	ult)	C1 = 01				
(22)		C2 = 10	! !		C3 = 11				
(38) 1	Enable the set of the output predicate register	0 = No er	nable		1 = Enable				
(39 – 43) 5	predicate field - selects a boolean function of	encoding	name	Descrip	tion	condition formula			
,-	the \$c register	0x00	never	always false (0	1		
		0x01	L (LT)	less th		(S & ~Z) ^ O			
		0x02	E (EQ)	Equa		Z & ~S			
		0x03	Le	less than o		S ^ (Z O)			
		0x04 0x05	G (GT)	greater t less or greater tha		~Z & ~(S ^ O)			
		0x05 0x06	lg ge	greater than	•	~(S ^ O)			
		0x07	lge	Order	<u> </u>	~Z ~S			
		0x08	u	Unorde	red	Z & S			
		0x09	lu	less than or u	nordered	S ^ O			
		0x0a	eu	equal or und		Z = 1 (2 1 2)			
		0x0b 0x0c	leu	not greate greater than or		Z (S ^ O) ~S ^ (Z O)			
		0x0d	gu Igu	not equa		~Z S			
		0x0e	geu	not less tan		(~S Z) ^ O			
		0x0f	always			1			
		0x10	0	Overflo		0			
		0x11 0x12	c a	carry / unsigned		C ~Z & C			
		0x13	S	sign / neg		S			
		0x1c	ns	not sign / p		~S			
		0x1d	na	unsigned no		Z ~C			
		0x1e	nc	not carry / unsigned below		~C			
(44 – 45)2	Input predicate register to compare before to	0x1f C0 = 00	no	no overt	C1= 01	~0			
(44 – 43)2	operate	C2= 10			C3= 11				
(46-50) 5	Comparison method of the input predicate	encoding	name	Descripti		condition formula			
	condition for operation	0x00	never	always false (n	ot used)	0			
		0x01	1	less tha	n	(S & ~Z) ^ O			
		0x02	e	Equal	ogual	Z & ~S			
		0x03 0x04	le g	less than or greater th	•	S ^ (Z O) ~Z & ~(S ^ O)			
		0x05	lg	less or greater tan					
		0x06	ge	greater than o	•	~(S ^ O)			
		0x07	lge	Ordere		~Z ~S			
		0x08	u	Unorder		Z & S			
		0x09 0x0a	lu eu	less than or un equal or uno		S ^ O Z			
		0x0a 0x0b	leu	not greate		Z (S ^ O)			
		0x0c	gu	greater than or u		~S ^ (Z O)			
		0x0d	lgu	not equa	l to	~Z S			
		0x0e	geu	not less t		(~S Z) ^ O			
		0x0f	always			1			
		0x10 0x11	С	Overflocarry / unsigned		O C			
		0x11 0x12	a	unsigned a		~Z & C			
		0x13	S	sign / nega		S			
		0x1c	ns	not sign / po		~S			
		0x1d	na	unsigned not		Z ~C			
	ĺ	0x1e	nc	not carry / unsig	ned below	~C			
		0x1f	no	no overfl	~···	~0			

(53) 1	Selection of the shared memory as one of the	1= Shared memory used	0 = Shared memory not used
	sources for comparison.		
(54) 1	Selection of constant memory as one of the	1= Constant memory used	0 = Constant memory not used
	sources for comparison.		
(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

(SASS Opcode SM_1.0)

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]</pre>

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...

Bit(s)	Mnemonics	Commenta	ary				
0	instr_is_long	0 = 32 bit le	_		1 = 64 h	oit long. (Default)	
1	instr_is_flow	0 = Norma		efault)		em ins. (flow contr	rol)
(2 – 8) 7	Destiny_Address_Register (32 bits)			0000101, R6 =00		(,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	
(9 – 15) 7	Inmeditate_Value_low_part			address (i.e.)			
(-) .		C[0x2][0x6					
(16–25) 10	Not used	00 0000 00					
(26-27) 2	Address register offset used for the constant memory	A0 = 00			A1 = 01		
(, -	addressing	A2 = 10			A3 = 11		
(28 – 31)4	Instruction Op. Code	MVC = 0x1					
(
(32 - 33) 2	instr_marker	00 = norma	al reg A	ccess(load or sto	re) (not ext	ra instruction)	
(,	- · · ·	(by default	_		-, (,	
			-	ccess(load or sto	re) (with Jo	in) (extra instruction	on)
			_			(it) (another option	
		11 = imme	•	·	, ,	•	
(34 - 35) 2	Not used	00					
(36-37) 2	Predicate register set (enabling a new flag) or Not used	C0 = 00 (b)	/ defaul	t)	C2 = 10		
		C1 = 01			C3 = 11		
38	Set predicate register as result of operation	1: Enable p	redicat	e register set	0 : Disable	e predicate register	rset
(39 - 43)5	Predicate condition to operate the instruction	Encoding	name	Descript	tion	condition formula	
	·	0x00	never	always f		0	
		0x01	1	less ta		(S & ~Z) ^ O	
		0x02	е	Equa		Z & ~S	
		0x03	le	less than or		S ^ (Z O)	•
		0x04	1		•	~Z & ~(S ^ O)	
		0x05	g	greater tan less or greater tan		~Z & (3 × 0)	
			lg 				_
		0x06	ge	greater than	•	~(S ^ O)	-
		0x07	lge	Ordere Unorde		~Z ~S	-
		0x08	u			Z & S	-
		0x09	lu	less than or u		S ^ O	-
		0x0a	eu	equal or und		Z	-
		0x0b	leu	not greate		Z (S ^ O)	-
		0x0c		greater than or		~S ^ (Z O)	_
		0x0d	lgu	not equa		~Z S	
		0x0e	geu	not less		(~S Z) ^ O	
		0x0f	always		•	1	
		0x10	0	Overflo		0	4
		0x11	С	carry / unsigned			
		0x12	а	unsigned a		~Z & C	
		0x13	S	sign / neg		S	
		0x1c	ns	not sign / p		~S	
		0x1d	na	unsigned no		Z ~C	
		0x1e	nc	not carry / unsi	_		
		0x1f	no	no overf	flow	~0	
(44 - 45) 2	Input_predicate_register	CO = 00 (by	defaul	-	: 1 = 01		
	Used as: precondition to operate the instruction	C2= 10			: 3 = 11		
(46-47) 2	Size of movement (source size)	11= 32 bits	5	0	1= 16 bits		
		00= 8 bits					
(48-53) 6	Not used	00 0000					
54	Address Register or Imm address	1 = Immed	iate ado	ress 0	= Address r	egister	
(55-57) 3	Not used	000					
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					
	Sub_opcode						

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 b	it long. (Default)		
1	instr_is_flow	0 = Normal ins.				em ins. (flow contro	ol)	
(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	g0[] - g15[]						
(/ -	32-bit byte-oriented addressing.	0000000 = g0[]	1					
	g	0000001 = g1[]						
			-					
		0001110 = g14	ֈ[] (b	y default)				
(23-27) 5	Not Used	0 0000						
(28-31)4	Instruction Op. Code	GLD = 0xD						
(32 - 33) 2	instr_marker			cess(load or store) (n				
				cess(load or store) (w			n)	
			_	cess(load or store) (w	ith Ex	it)		
/24 20\ 5	Notuced	11 = immediate	:e					
(34 - 38) 5 (39 – 43) 5	Not used Predicate condition to operate the instruction	0 0000	ma -	Deceriation	I	condition formation		
,33 – 43) 5	rredicate condition to operate the instruction	Encoding na		Description		condition formula		
		 	ver	always false		0 (5.8. ~7) A O		
		0x01		less tan		(S & ~Z) ^ O		
		 	e Ia	Equal	.1	Z & ~S		
			le	less than or equa	31	S ^ (Z O)		
			g	greater tan	_	~Z & ~(S ^ O)		
			lg	less or greater ta		~Z		
			ge	greater than or eq	ual	~(S ^ O)		
			ge	Ordered		~Z ~S		
			u	Unordered		Z & S		
		_ 	lu	less than or unorde		S ^ O		
			eu	equal or unordere		Z		
			eu	not greater tan		Z (S ^ O)		
				greater than or unor	dered	~S ^ (Z O)		
			gu	not equal to		~Z S		
			eu	not less tan		(~S Z) ^ O		
			vays		ault)	1		
			0	Overflow		0		
				carry / unsigned not b		С		
		 	a	unsigned above	!	~Z & C		
			S	sign / negative		S		
			าร	not sign / positiv		~\$		
			na	unsigned not above		Z ~C		
				not carry / unsigned l	below	~C		
			10	no overflow		~0		
(44 - 45) 2	Input_predicate_register	CO = 00 (by def	fault	•	C1 = 01			
(46 52) =	Used as: precondition to operate the instruction	C2= 10			C3= 11			
(46 - 52) 7	Not used	000 0000	10)		201 5	T 60 (60)		
(53 – 55) 3	Destiny_move_size	000=DT_U8 (U	-			T_S8 (S8)		
		010=DT_U16 (U				T_S16 (U16) T_U128 (U128) (N 6)T	
		110=DT_U32 (U		• • •		T_U128 (U128) (NC T_S32 (S32)	Ji sup	
(56-60) 5	Not used	0 0000	032		111-D	1_332 (332)		
(61 - 63) 3	Sub_opcode	100 Load						
(01 - 02) 2	Jun_opcode	100 Loau						

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 de	<u> </u>					
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 Not used	00 normal reg Access(load or store) (not extra instruction 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 0 0000	on) (By					
(39 – 43) 5	predicate condition	encoding name Description condition for	rmula					
	, · · · · · <u>-</u> · · · · · ·	0x00 never always false 0						
		0x01 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 4	^ O)					
		0x05 lg less or greater tan ~Z						
		0x06 ge greater than or equal ~(S ^ C))					
		0x07 lge	S					
		0x08 u Unordered Z & S						
		0x09 lu less than or unordered S ^ O						
		OxOa eu equal or unordered Z						
		OxOb leu not greater tan Z (S ^						
		OxOc gu greater than or unordered ~S ^ (Z						
		OxOd Igu not equal to ~Z S						
		OxOe 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						
		Ox12 a unsigned above ~Z & C	-					
		0x13ssign / negativeS0x1cnsnot sign / positive~S						
		Ox1c ns not sign / positive S						
		Ox1e nc not carry / unsigned below ~C	•					
		0x1f no no overflow ~O						
(44 – 45) 2	Input_predicate_register	C0= 00 (by default) C1= 01						
, , , , ,		C2= 10 C3= 11						
(46 – 52) 7	Not used	000 0000						
(53-55) 3	Move_operand_size	U8 = 000 U64 = 100						
		S8 = 001 U128 = 101						
		U16 = 010 U32 = 110 (by default	:)					
		S16 = 011 S32 = 111						
(56 - 60) 5	Not used	0 0000						
(61 - 63) 3	Sub_Opcode	000=DT_U16						
		001=DT_S16	ult)					
		010=DT_S16						
		011=DT_U32						

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, R	6=000110
(9 – 15)7	Source_1:	Register case:	Shared memory location:
	It can be a general-purpose register or a shared memory location	R0 = 00000,	(9-13) offset of the location
		R5 = 00101,	(14) 1
		R6 = 00110	(15) 1
(16-22) 7	Not used	Register case:	Second part of the constant
		R0 = 00000,	memory (i.e.)
		R5 = 00101,	C[0x2][0x16]
		R6= 00110	(16-22) = 01 0110
(23-27) 5	High part of the Source_2 or configuration options:	Shared memory:	First part of the constant
		(23-25) 000	memory (i.e.)
		(26-27) Address register	C[0x2][0 x16]
		part of the address (i.e.)	(23-26) = 0010 0
		g[A2+0x1] = 10	
		options are:	
		A0 = 00 , A1 = 01	
		A2 = 10, A3 = 11	
(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	ς = 0			
(35 - 38) 4	Not used	0000						
(39 - 43)5	predicate_condition	encoding	name	Descripti	ion	condition formula		
		0x00	never	always fa	alse	0		
		0x01	I	less tar	n	(S & ~Z) ^ O		
		0x02	е	Equal		Z & ~S		
		0x03	le	less than or		S ^ (Z O)		
		0x04	g	greater t		~Z & ~(S ^ O)		
		0x05	lg	less or great		~Z		
		0x06	ge	greater than o	<u> </u>	~(S ^ O)		
		0x07	lge	Ordere		~Z ~S		
		0x08	u	Unorder		Z & S		
		0x09	lu	less than or un		S ^ O		
		0x0a	eu	equal or uno	Z / (S A O)			
		0x0b	leu	not greate	Z (S ^ O)			
		0x0c	gu	u not equal to		· ' '		
		0x0d 0x0e	lgu			~Z S		
		0x0e 0x0f	geu alway			(~S Z) ^ O 1		
		0x01	o o	Overflo		0		
		0x10	С	carry / unsigned				
		0x11	a	unsigned a		~Z & C		
		0x12	S	sign / nega		S		
		0x1c	ns	not sign / po		~S		
		0x1d	na	unsigned not		Z ~C		
		0x1e	nc	not carry / unsig		•		
		0x1f	no	no overfl		~0		
(44 – 45) 2	Input predicate register	CO = 00 (by defa	ult)	C1 = 01	-		
,		C2= 10		,	C3 = 11			
(46-48) 3	Size of movement (source size)	000	DT_U8		100 D	T_U32		
		001	OT_U16		101 D	_ T_U32		
		010	OT_S16		110 D	T_U32		
		011	DT_U32		111 D	T_U32		
49	??	1						
(50 – 52) 3	Not used	000						
53	Source 1 selector	1 = Share	ed mem	ory	0 = Regist	er		
(54 – 57) 4	Not used	0000						
58	Size of the destiny	1= 32 bit			0= 16 bits	- 4		
59	Signed or unsigned sources	1=S16/S3	32		D= U16/U3:	2 (Unsigned)		
60	Not used	0						
(61 - 63) 3	Sub_Opcode	000=DT_	_		100=DT_S			
		001=DT_	_		_	32 (by default)		
		010=DT_	_		110=DT_U			
		011=DT_	U32		111=DT_S	32		

MOV32 Instruction:



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:	Shared memory case:			
		R0 = 00000	Offset value (9 - 12)			
		R5 = 00101	1 (13)			
		R6 = 00110	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	000 0000						
(16 - 21) 6	Inmeditate_Value_low_part	XX XXXX	XX XXXX						
(22 - 27) 6	Not used	00 0000	00 0000						
(28 - 31)4	Instruction Op. Code	GLD = 0x1	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 X	XXX XXXX						
60	Not used								
(61 - 63)3	Sub opcode	000	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...

Bit(s)	Mnemonics	Commenta						
0	instr_is_long	0 = 32 bit lo			_	oit long. (Default)		
1	instr_is_flow	0 = Norma	ins. (D	1 = Syst	em ins. (flow contr	rol)		
(2 – 6) 5	Not used	0 0000						
(7 – 19) 13	Address offset	The size capacity of the shared memory is 0x4000						
(20–25) 6	Not used	00 0000			144 04			
(26-27) 2	Address register used for the shared memory addressing	A0 = 00			A1 = 01			
/20 21\4	Instruction On Code	A2 = 10 R2G = 0x0			A3 = 11			
(28 – 31)4	Instruction Op. Code	K2G = 0X0						
(32 - 33) 2	instr_marker	00 = norma	al reg A	ccess(load or stor	e) (not ext	ra instruction)		
,		(by default	•	,	-, (,		
			-	ccess(load or stor	e) (with Jo	in) (extra instruction	on)	
			_			it) (another option		
		11 = imme	diate					
(34 - 35) 2	Not used	00						
(36-37) 2	Predicate register set (enabling a new flag) or Not used	C0 = 00 (by	defaul	t)	C2 = 10			
		C1 = 01			C3 = 11			
38	Set predicate register as result of operation			e register set		predicate register	-	
(39 – 43) 5	Predicate condition to operate the instruction	Encoding		-		condition formula	1	
		0x00	never	always fa		0		
		0x01		less tar	1	(S & ~Z) ^ O		
		0x02	е	Equal		Z & ~S		
		0x03	le	less than or	·	S ^ (Z O)		
		0x04	g	greater t		~Z & ~(S ^ O)		
		0x05	lg	less or great		~Z		
		0x06	ge	greater than c	-	~(S ^ O)		
		0x07	lge	Ordere		~Z ~S		
		0x08	u	Unorder		Z & S		
		0x09	lu	less than or un		S ^ O		
		0x0a	eu	equal or uno		Z		
		0x0b	leu	not greater		Z (S ^ O)		
		0x0c	gu	greater than or u		· · ·		
		0x0d	lgu	not equal		~Z S		
		0x0e	geu	not less t		(~S Z) ^ O		
		0x0f	always			1		
		0x10	0	Overflov		0	-	
		0x11	С	carry / unsigned			-	
		0x12	a	unsigned a		~Z & C	ļ	
		0x13	S	sign / nega		\$ ~\$	4	
		0x1c	ns	not sign / po			-	
		0x1d	na	unsigned not		Z ~C	-	
		0x1e 0x1f	nc no	not carry / unsign no overfl		~C ~O	1	
/// /E\2	Input predicate register					U		
(44 – 45) 2	Input_predicate_register Used as: precondition to operate the instruction	C0= 00 (by C2= 10	uelaul	-	L= 01 B= 11			
(46–52) 7	Source register		00. R5=	0000101, R6=000				
(53–54) 2	Size of movement (source size)	01= 32 bits		5555151, 110-556				
(33 34) 2	and the country (source size)	01= 32 bits 00= 16 bits						
		10= 8 bits						
(55-57) 3	Not used	000						
58	Size of the destiny	1= 32 bits		0=	16 bits			
59	Signed or unsigned sources	1=S16/S32				Unsigned)		
60	Not used	0				•		
		111						

R2A Instruction:

<mark>Checked</mark> 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...

1	Pending								
0	Bit(s)	Mnemonics	Commen	itary					
1		instr_is_long				1 =	64 bit long. (by de	fault)	
(9 - 15) 7 Source_Data_Register R0= 000000, R5=000101, R6=00110 (16 - 27) 12 Immediate value Oxyryr (28 - 31) 4 Instruction Op. Code R2A = 0x0 (32 - 33) Instr_marker On normal reg Access(load or store) (with Join) (extra instruction 01 normal reg Access(load or store) (with Join) (extra instruction 11 immediate (35 - 38) 4 Not used O (35 - 38) 4 Not used O (39 - 43) 5 Predicate condition to operate the instruction Encoding name Description Condition fo	1	instr_is_flow					System ins. (flow o		
(32 - 33) Instruction Op. Code R2A = 0x0			A0= 0000	000, A5=	000101, A6=000110				
(32 - 33)		Source_Data_Register	R0= 0000	000, R5=	000101, R6=000110.				
1	(16 – 27) 12	Immediate value	0xYYY						
Oli normal reg Access(load or store) (with Join) (extra ins 10 normal reg Access(load or store) (with Exit)	(28 – 31) 4	Instruction Op. Code	R2A = 0 x	0					
Oli normal reg Access(load or store) (with Join) (extra ins 10 normal reg Access(load or store) (with Exit)									
(35 - 38) 4 (39 - 43) 5 Predicate condition to operate the instruction									
Predicate condition to operate the instruction			•						
Ox00 never always false O				1		·			
No.01	(39 – 43) 5	Predicate condition to operate the instruction			•		condition formula		
0x02 e Equal 2 & ~S 0x03 le less than or equal 5 ^ (Z C 0x04 g greater tan ~Z & ~(S ^ N 0x05 lg less or greater tan ~Z & ~(S ^ N 0x06 ge greater than or equal ~(S ^ N 0x07 lge Ordered ~Z ~S 0x08 u Unordered Z & S 0x09 lu less than or unordered Z & S 0x09 lu less than or unordered Z 0x0a eu equal or unordered Z 0x0b leu not greater tan Z (S ^ N 0x0c gu greater than or unordered ~S ^ N 0x0c gu greater than or unordered ~Z S 0x00 lgu not equal to ~Z S 0x00 geu not less tan (~S Z) 0x00 always always true (by default) 1 0x10 0 0verflow 0 0x11 c carry / unsigned not below C 0x12 a unsigned above ~Z & C 0x13 a unsigned above ~Z & C 0x14 a unsigned not above Z ~C 0x16 ns not sign / positive ~S 0x16 ns not sign / positive ~S 0x16 ns not carry / unsigned below ~C 0x16 ns not carry / unsigned below ~C 0x17 no no overflow ~O 0x18 nc not carry / unsigned below ~C 0x19 nc not carry / unsigned ~C				never					
Dx03 le less than or equal S ^ (Z C Ox04 g greater tan ~ 72 & ~ (S ^ C C Ox05 lg less or greater tan ~ 72 & ~ (S ^ C C Ox05 lg less or greater tan ~ 72 & ~ (S ^ C C Ox06 ge greater than or equal ~ (S ^ C Ox07 lge Ordered ~ 72 ~ 5 Ox08 u Unordered Z & S Ox09 lu less than or unordered Z & S Ox09 lu less than or unordered Z Ox00 leu not greater tan Z (S ^ C Ox06 gu greater than or unordered ~ 72 S Ox06 gu greater than or unordered ~ 75 ^ (Z Ox06 lgu not equal to ~ 72 S Ox06 geu not less tan (~ 5 2) ^ Ox06 geu not less tan (~ 5 2) ^ Ox06 geu not less tan (~ 5 2) ^ Ox06 geu not less tan (~ 5 2) ^ Ox07 laways always true (by default) 1 Ox10 O Ox11 C Carry / unsigned above ~ 72 & C Ox12 a unsigned above ~ 72 & C Ox13 s sign / negative S Ox12 a unsigned above ~ 72 & C Ox12 a unsigned not above Z ~ C Ox16 na unsigned not above Z ~ C Ox16 Ox16 na unsigned not above Z ~ C Ox16			 				(S & ~Z) ^ O		
Ox04 g greater tan ~72 & ~(\$^{\chickness}\$ \\ Ox05 lg less or greater tan ~72 \\ Ox06 ge greater than or equal ~(\$^{\chickness}\$ \\ Ox07 lge Ordered ~72 ~8 \\ Ox08 u Unordered 72 ~8 \\ Ox09 lu less than or unordered 72 ~8 \\ Ox09 lu less than or unordered 73 ~8 \\ Ox09 lu less than or unordered 74 ~8 \\ Ox00 gu greater than or unordered 74 ~8 \\ Ox00 gu greater than or unordered ~7 \\ Ox00 gu greater than or unordered ~7 \\ Ox00 gu not equal to ~72 \$ \\ Ox00 lgu not equal to ~72 \$ \\ Ox00 geu not less tan (~5 2) \\ Ox00 geu not less tan (~5 2) \\ Ox00 geu not less tan (~5 2) \\ Ox01 a unsigned above ~2 \$ \\ Ox12 a unsigned above ~7 \$ \\ Ox12 a unsigned above ~7 \$ \\ Ox12 a unsigned not above ~7 \$ \\ Ox12 a unsigned not above 7 ~C \\ Ox12 a unsigned not above 7 ~C \\ Ox12 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsigned below ~C \\ Ox16 nc not carry / unsi				_					
Ox05 Ig less or greater tan ~7				le		ıal	S ^ (Z O)		
0x06 ge greater than or equal ~(\$ ^ \color 0x07 lge Ordered ~2 ~\color 5 ~\color 0x08 u Unordered Z & \$ \color 5 ~\color 0x09 lu less than or unordered Z & \$ \color 0x09 lu less than or unordered Z & \$ \color 0x09 lu less than or unordered Z & \$ \color 0x00 leu not greater tan Z (\$ ^ \color 0x00 lgu not equal to ~\color 2 \$ \color 0x00 lgu not equal to ~\color 2 \$ \color 0x00 lgu not equal to ~\color 2 \$ \color 0x00 lgu not less tan (~\color 5 2) ^ \color 0x00 lgu not less tan (~\color 5 2) ^ \color 0x00 lgu not less tan (~\color 5 2) ^ \color 0x00 lgu not less tan (~\color 5 2) ^ \color 0x00 lgu not less tan (~\color 5 2) ^ \color 0x11 c carry / unsigned not below C 0x11 c carry / unsigned above ~\color 2 & \color 0x12 a unsigned above ~\color 0x12 a unsigned above ~\colo			I 	+			~Z & ~(S ^ O)		
0x07 Ige 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 0x0b leu not greater tan Z (S ^ O 0x0b leu not greater tan Z (S ^ O 0x0b leu not equal to ~Z S 0x0b leu not equal to ~Z S 0x0b leu not less tan (~S Z 0x0b leu not less tan (~S Z 0x0b leu not equal to ~Z S 0x0b leu not less tan (~S Z 0x1b 0x1				lg	less or greater t	an			
Ox08 u			 	ge		qual	~(S ^ O)		
Ox09 Iu less than or unordered S ^ O Ox00 eu equal or unordered Z (S ^ Ox00 leu not greater tan Z (S ^ Ox00 leu not greater tan Z (S ^ Ox00 leu not equal to ~ C S S Ox00 leu not equal to ~ C S S Ox00 leu not equal to ~ C S Z Ox00 leu not equal to ~ C S Z S Ox00 leu not equal to ~ C S Z S Ox00 leu not equal to ~ C S Z S Ox10 Ox010 Overflow O Ox11 C Carry / unsigned not below C Ox12 a unsigned above ~ C X S Ox12 a unsigned above ~ C X S Ox12 ns not sign / positive ~ S Ox12 ns not sign / positive ~ S Ox12 ns not carry / unsigned below ~ C Ox14 na unsigned not above Z ~ C Ox16 no no overflow ~ C Ox16 no no overflow ~ C Ox16 no no overflow ~ C Ox16 Not used Ox10			0x07	lge	Ordered		~Z ~S		
Ox0a eu equal or unordered Z (S ^ 0)			0x08	u	Unordered		Z & S		
Ox0b leu not greater tan Z (5 ^ 0 0 x 0 c gu greater than or unordered ~ S ^ (Z 0 x 0 d lgu not equal to ~ Z S 0 x 0 d lgu not equal to ~ Z S 0 x 0 e geu not less tan (~ S Z) ^ 0 x 0 e geu not less tan (~ S Z) ^ 0 x 0 e geu not less tan (~ S Z) ^ 0 x 10 o Overflow O Ox11 c carry / unsigned not below C Ox12 a unsigned above ~ Z & C Ox13 s sign / negative S Ox10 ns not sign / positive ~ S Ox10 ns not sign / positive ~ S Ox10 na unsigned not above Z ~ C Ox10 no not carry / unsigned below ~ C Ox10 no no overflow ~ O Ox10 no no overflow ~ Ox10 Ox1			0x09	lu	less than or unord	lered	S ^ O		
Ox0c gu greater than or unordered ~S ^ (Z Ox0d lgu not equal to ~Z S Ox0e geu not less tan (~S Z) Ox0f always always true (by default) 1 Ox10 o Overflow O Ox11 c carry / unsigned not below C Ox12 a unsigned above ~Z & C Ox12 a unsigned above ~Z & C Ox13 s sign / negative S Ox1c ns not sign / positive ~S Ox1d na unsigned not above Z ~C Ox1d na unsigned not above Z ~C Ox1f no no overflow ~C Ox1f			0x0a	eu	equal or unorde	red	Z		
Ox0d igu not equal to ~7 S Ox0e geu not less tan (~S Z) ^ Ox0e geu not less tan (~S Z) ^ Ox0e geu not less tan (~S Z) ^ Ox0e always always true (by default) 1 Ox10 O Overflow O Ox11 C Carry / unsigned not below C Ox12 a unsigned above ~72 & C Ox12 a unsigned above ~72 & C Ox12 a unsigned not above S Ox1c ns not sign / positive ~S Ox1c ns not sign / positive ~S Ox1d na unsigned not above Z ~C Ox1e nc not carry / unsigned below ~C Ox1e nc not carry / unsigned below ~C Ox1f no no overflow ~O Ox1e			0x0b	leu	not greater tar	n	Z (S ^ O)		
Ox0e geu not less tan (~S Z) ^ Ox0f always always true (by default) 1			0x0c	gu	greater than or uno	rdered	~S ^ (Z O)		
Ox0f always always true (by default) 1			0x0d	lgu	not equal to		~Z S		
Ox0f always always true (by default) 1			0x0e	geu	not less tan		(~S Z) ^ O		
Ox11			0x0f	always	always true (by de	fault)			
0x12 a unsigned above ~Z & C			0x10	0	Overflow		0		
Ox13 s sign / negative S			0x11	С	carry / unsigned not	below	С		
Ox1c ns not sign / positive ~S			0x12	а	unsigned abov	e	~Z & C		
Ox1d na unsigned not above Z ~C			0x13	S	sign / negative	e	S		
Ox1e nc not carry / unsigned below ~C			0x1c	ns	not sign / positi	ve	~\$		
Ox1f no no overflow ~O			0x1d	na	unsigned not abo	ove	Z ~C		
(44 - 45) 2 Input_predicate_register C0= 00 (by default) C1= 01 (46 - 60) 14 Not used 00 0000 0000 0000 (61-63) Sub_Opcode 000 DT_U16 001 DT_S16 010 DT_S16			0x1e	nc	not carry / unsigned	below	~C		
C2= 10 C3= 11 (46 - 60) 14 Not used (61-63) Sub_Opcode 001 DT_S16 010 DT_S16			0x1f	no	no overflow		~0		
(46 - 60) 14 Not used 00 0000 0000 0000 (61-63) Sub_Opcode 000 DT_U16 001 DT_S16 010 DT_S16	44 – 45) 2	Input_predicate_register	CO = 00 (by defa	ult)	C1 = 01	l		
(61-63) Sub_Opcode 000 DT_U16 001 DT_S16 010 DT_S16						C3 = 11			
001 DT_S16 010 DT_S16			00 0000 0	000000	00				
100 DT_S32 101 DT_S32 110 DT_U32 (by default) 111 DT_S32	(61-63)	Sub_Opcode	001 C 010 C 011 C 100 C 101 C	OT_S16 OT_S16 OT_U32 OT_S32 OT_S32 OT_U32	(by default)				

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...

51.11									
Bit(s)	Mnemonics	Comment							
0	instr_is_long	0 = 32 bit				1 = 64 bit long. (by default)			
1	instr_is_flow			by default)		System ins. (flow c	ontrol)		
(2 – 8) 7	Destiny_ Address_Register (32 bits)		A0= 000000, A5=000101, A6=000110						
(9 - 15) 7	Not used		000 0000						
(16 - 25) 10	Immediate value	00 0000 0	000						
(26-27) 2	Address register used for the shared memory	A0 = 00			A1 = 0				
	addressing	A2 = 10			A3 = 1	1			
(28 – 31) 4	Instruction Op. Code	R2A = 0x0							
(32 - 33)	instr_marker	01 norma 10 norma	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	Not used	0							
(35 - 38) 4	Not used	0000							
(39 – 43) 5	Predicate condition to operate the instruction	Encoding	name	Description		condition formula			
		0x00	never	always false		0			
		0x01	_	less tan		(S & ~Z) ^ O			
		0x02	e	Equal		z & ~s			
		0x03	le	less than or equ	ıal	S ^ (Z O)			
		0x04	g	greater tan		~Z & ~(S ^ O)			
		0x05	lg	less or greater to	an	~Z			
		0x06	ge	greater than or ed	qual	~(S ^ O)			
		0x07	lge	Ordered		~Z ~S			
		0x08	u	Unordered		Z & S			
		0x09	lu	less than or unord	ered	S ^ O			
		0x0a	eu	equal or unorder	red	Z			
		0x0b	leu	not greater tar		Z (S ^ O)			
		0x0c		greater than or uno					
		0x0d	lgu	not equal to		~Z S			
		0x0e	geu	not less tan		(~S Z) ^ O			
			always		fault)	1			
		0x10	0	Overflow		0			
		0x11	С	carry / unsigned not	below				
		0x12	a	unsigned abov		~Z & C			
		0x13	S	sign / negative		S			
		0x1c	ns	not sign / positi		~S			
		0x1d	na	unsigned not abo		Z ~C			
		0x1e	nc	not carry / unsigned		•			
		0x16	no	no overflow	DEIOW	~0			
44 4E\2	Input prodicate register				C1= 01				
44 – 45) 2	Input_predicate_register	C0 = 00 (b) C2 = 10	у аета	uit)	C1 = 03				
(46 – 60) 14	Not used	00 0000 0	000 00	00	C3- 1.	•			
· · · · · ·			000 00	00					
(61-63)	Sub_Opcode	010							

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

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	A0= 000000, A5=000101, A6=000110				
(9 - 24)16	Immediate value (Low part?) (FlexGrip only uses 22-9)	Imm value: 0xXXXX				
25	Source_1_Selector??	0 = Register Source	1 = Shared Mem.			
(26-27)2	Source_Address_Register	Options are:				
		A0: 00 A1: 01				
		A2: 10 A3: 11				
(28 - 31) 4	Instruction Op. Code	ADA = 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	Not used	0								
(35)1	destination type	0 = Regist			1= Memory destination					
(36-37)2	Predicate register set (enabling a new flag) or Not used	CO = 00 (k	y defa	ult)						
		C1 = 01								
		C2 = 10								
(20)4	Mills and the first and the transfer	C3 = 11	1. 1 .	17 (- 1					
(38)1	Write enable / set predicate register			d (just for memory destinati						
		1 = enable	e preak	cate register set, 0 = disable	predicate register set					
		Not used	(0)							
(39 - 43)	predicate_condition	encoding	name	Description	condition formula					
5		0x00	never	always false (not used)	0					
		0x01	_	less tan	(S & ~Z) ^ O					
		0x02	е	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 equ						
		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						
		0x0d	lgu	not equal to	~Z S					
		0x0e	geu	not less tan	(~S Z) ^ O					
			always		1					
		0x10	0	Overflow	0					
		0x11	С	carry / unsigned not below						
		0x12	а	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						
(45 -5)		0x1f	no	no overflow	~0					
(45 - 59)	Not used? High part of the Imm, value, or from the	0000 0000	0000	0000						
(64 63)2	source of destiny register?	004								
(61 – 63)3	Sub_op_code	001								

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:

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:	Shared memory case:			
		R0= 00000	Offset value (9 - 12)			
		R5= 00101	1 (13)			
		R6= 00110	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	A1 = 01			
		A2 = 10	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:

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:	Shared memory case:			
		R0 = 00000	Offset value (9 - 12)			
		R5 = 00101	1 (13)			
		R6 = 00110	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)							
(2.4)4	Her I fee	11 = immediate							
(34)1	Used for	0				1			
(35)1	destination type	0 = Regist				1= Mer	nory destination		
(36-37) 2	Predicate register set (enabling a new flag) or Not used	C0 = 00 (L) $C1 = 01$	оу аета	uit)					
		C1 = 01 C2 = 10							
		C2 = 10 C3 = 11							
(20) 1	Cat pradicate register		o prodi	cata ragio	stor sot	O - Dic	abla prodicata regista	rcot	
(38) 1	Set predicate register	1 = Enabl					able predicate registe	rset	
(39 – 43)5	predicate_condition	encoding			Description		condition formula		
		0x00	never	aiway	ys false (not ι	usea)	0		
		0x01	-		less tan		(S & ~Z) ^ O		
		0x02	e	1	Equal		Z & ~S		
		0x03	le	ies	s than or equ	ıaı	S ^ (Z O)		
		0x04	g	loos and	greater tan		~Z & ~(S ^ O)		
		0x05	lg		reater tan / n		~Z		
		0x06	ge	grea	ter than or e	quai	~(S ^ O)		
		0x07	lge		Ordered		~Z ~S		
		0x08	u	1	Unordered	l a se a d	Z & S		
		0x09	lu		han or unord			S ^ O Z	
		0x0a	eu		ial or unorde				
		0x0b	leu		ot greater ta		Z (S ^ O)		
		0x0c	gu		than or uno	rdered	~S ^ (Z O)		
		0x0d	lgu		not equal to		~Z S		
		0x0e	geu	.	not less tan	C14\	(~S Z) ^ O		
			always	aiway	s true (by de	rauit)	1		
		0x10	0		Overflow	. la a l'acce	0		
		0x11	С	• •	unsigned not		C		
		0x12	a	+	nsigned abov		~Z & C		
		0x13	S	-	ign / negative		<u>S</u>		
		0x1c	ns		t sign / positi		Z ~C		
		0x1d	na		igned not abo		~C		
		0x1e 0x1f	nc		ry / unsigned no overflow	below	~0		
(44 45) 2	Innut wedicate register to compare before to consta		no		no overnow	C1 - 01	U		
(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:	Register	250 114	6-521.	Constant m		Shared memory	<i>,</i> ·	
(40 - 33) 0	4) GPRS	R0 = 0000	-	J-32j.	Second part	-	(46-52): 000 000	•	
	5) Constant memory	R5 = 0000			constant me		, ,		
	6) Shared memory	R6 = 0010			C[0x2][0x16		memory)		
	-,,	(53) 0			(46-53) = 00	-	, ,		
(54 – 57)4	First part of the Source_2 when constant memory is	• •	of the	constant	memory (i.e.				
	employed	C[0x2][0x			, (
		(54-57) =	_						
58	Sign of Source_1	Positive =				Negativ	ve = 1		
59	Sign of Source_2	Positive =	0			Negativ			
60	Not used	0							
(61 – 63)3	Sub_op_code	000							

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:

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	<pre>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)</pre>
(34 - 59) 26	The high part of the immediate value of 32 bits	11 - Illimediate (by deladit)
(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:

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:	Register case:	Shared memory location:
	It can be a general-purpose register or a shared memory location	R0 = 00000,	(9-13) offset of the location
		R5 = 00101,	(14) 1
		R6 = 00110	(15) 1
(16-22) 7	Source_2:	Register case:	Second part of the constant
	It can be a general-purpose register or a constant memory location	R0 = 00000,	memory (i.e.)
		R5 = 00101,	C[0x2][0x16]
		R6 = 00110	(16-22) = 01 0110
(23-27) 5	High part of the Source_2 or configuration options:	Shared memory:	First part of the constant
		(23-25) 000	memory (i.e.)
		(26-27) Address register	C[0x2][0 x16]
		part of the address (i.e.)	(23-26) = 0010 0
		g [A2+0x1] = 10	
		options are:	
		A0 = 00 , A1 = 01	
		A2 = 10 , A3 = 11	
(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 = Regist	er dest	ination		mory destination
(36-37) 2	Predicate register set (enabling a new flag) or Not used	CO = 00 (k			C1 = 01	,
, ,	<i>3</i> . <i>3</i>	C2 = 10	•	,	C3 = 11	
(38) 1	Set predicate register	1 = Enabl	e predi	cate register set	0 = Dis	sable predicate register set
(39 – 43)5	predicate_condition	encoding	name	Description		condition formula
		0x00	never	always false (not u	used)	0
		0x01	I	less tan		(S & ~Z) ^ O
		0x02	е	Equal		Z & ~S
		0x03	le	less than or equ	ıal	S ^ (Z O)
		0x04	g	greater tan		~Z & ~(S ^ O)
		0x05	lg	less or greater tan / n	ot equa	~Z
		0x06	ge	greater than or e	qual	~(S ^ O)
		0x07	lge	Ordered		~Z ~S
		0x08 u Unordered			Z & S	
		0x09 lu less than or unordered			S ^ O	
		0x0a	eu	equal or unorde	red	Z
		0x0b	leu	not greater tha	an	Z (S ^ O)
		0x0c	gu	greater than or uno	rdered	~S ^ (Z O)
		0x0d	lgu	not equal to		~Z S
		0x0e	geu	not less tan		(~S Z) ^ O
		0x0f	always	always true (by de	fault)	1
		0x10	0	Overflow		0
		0x11	С	carry / unsigned not	below	С
		0x12	a	unsigned abov	e e	~Z & C
		0x13	S	sign / negative		S
		0x1c	ns	not sign / positi		~\$
		0x1d	na	unsigned not ab		Z ~C
		0x1e	nc	not carry / unsigned		~C
		0x1f	no	no overflow		~0
(44 - 45) 2	Input predicate register to compare before to operate	CO = 00			C1 = 01	
		C2 = 10			C3 = 11	
(46 - 47) 2	Result round method	Not roun	ding = (00		ed 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	0		No = 0	
(58) 1	Sign of Source_1	Positive =				
(59) 1	Sign of Source_2	Positive = 0, Negative = 1				
(60) 1	Not used	0				
(61 – 63)3	Sub_op_code	000				

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:

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:	Shared memory case:	
		R0= 00000	Offset value (9 - 12)	
		R5= 00101	1 (13)	
		R6= 00110	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	A1 = 01	
		A2 = 10	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:

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):

D::/ \						
Bit(s)	Mnemonics	Commentary				
0	instr_is_long	0 = 32 bit long. 1 = 64			64 bit long. (Default)	
1	instr_is_flow	0 = Normal ins. (Defau	lt)	1 = Syste	1 = System ins. (flow control)	
(2-8)7	Destiny_General_purpose_register	R0= 000000, R5=00010	01, R6=0001	10		
(9 – 15)7	Source_1:	Register case:	Shared me	emory loca	tion:	
	It can be a general-purpose register or a shared memory	R0 = 00000,	(9-13) an o	offset of th	e location	
	location	R5 = 00101,	(14) 1			
		R6 = 00110	(15) 1			
(16-22) 7	Source_2:	Register case:	Second pa	rt of the co	onstant memory (i.e.)	
	It can be a general-purpose register or a constant memory	R0 = 00000,	C[0x2][0x1	[6]		
	location	R5 = 00101,	(16-22) = 0	01 0110		
		R6= 00110				
(23-27) 5	High part of the Source_2 or configuration options:	Shared memory:	Constant r	nemory:	Constant memory:	
		(23-25) 000	First part of	of the	This field also can be	
		(26-27) Address	constant n	nemory	employed as part of	
		register part of the	(i.e.)		the source 3, when	
		address (i.e.)	C[0x2][0 x1	6]	the constant	
		g[A2+0x1] = 10	(23-27) = 0	0010	memory is	
		options are:			employed as SRC3.	
		A0 = 00, A1 = 01			(23) :0	
		A2 = 10 , A3 = 11			(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)						
(2.4)4	Head for	11 = immediate						
(34)1	Used for	0		in a til a m	1 14 2 2 2 2			
(35)1	destination type	0 = Regist				ory destination		
(36-37) 2	Predicate register set (enabling a new flag) or Not used	CO = 00 (I	оу аета	uit)	C1 = 01			
(20) 1	Cat avadianta vanistav	C2 = 10	المحمد م	cate register set		C3 = 11		
(38) 1 (39 – 43)5	Set predicate register predicate_condition	encoding		<u> </u>	0 = Disable predicate register set			
(33 – 43)3		0x00	never	Description always false (no		condition formula 0		
		0x00	ı	less tan	it useuj	(S & ~Z) ^ O		
		0x01	e	Equal		Z & ~S		
		0x03	le	less than or e	onual	S ^ (Z O)		
		0x04	g	greater ta	•	~Z & ~(S ^ O)		
		0x05		less or greater tan		` '		
		0x06	ge	greater than or		~(S ^ O)		
		0x07	lge	Ordered	•	~Z ~S		
		0x08	u	Unordere		Z & S		
		0x09	lu	less than or und		S ^ O		
		0x0a	eu	equal or unor	dered	Z		
		0x0b	leu	not greater t	han	Z (S ^ O)		
		0x0c	gu	greater than or u	nordered	~S ^ (Z O)		
		0x0d	lgu	not equal	to	~Z S		
		0x0e	geu	not less ta	ın	(~S Z) ^ O		
		0x0f	always	always true (by	default)	1		
		0x10	0	Overflow	/	0		
		0x11	С	carry / unsigned n	ot below	С		
		0x12	a	unsigned ab	ove	~Z & C		
		0x13	S	sign / negat		S		
		0x1c	ns	not sign / pos		~S		
		0x1d	na	unsigned not a		Z ~C		
		0x1e	nc	not carry / unsign		~C		
		0x1f	no	no overflo		~0		
(44 - 45) 2	Input predicate register to compare before to operate	CO = 00			C1 = 01			
(46.52).6	Common 2. It was all become interest and an arrangement	C2 = 10			C3= 11			
(46-52) 6	Source 3: It could be a register or a constant memory	Register				nt memory:	mon	
	location	R0 = 00000, R5 = 00101,				rt of the constant me 0x2][0 x16]	:iiioiy	
		R6 = 0010	-			= 00 0010		
(53) 1	Shared memory use for Source_2?	Yes = 1	J		No = 0	30 0010		
(55) 1	A bit indicates if the shared memory is employed	103 - 1			110 = 0			
(54) 1	Use of constant memory for Source_2 or Source_3?	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
(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:

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:

Bit(s)	Mnemonics	Commentary					
0	instr_is_long	0 = 32 bit long.	1 = 64 bit long. (Default)				
1	instr_is_flow	0 = Normal ins. (Defau	lt)	1 = System ins. (flow control)			
(2-8)7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110					
(9 – 15)7	Source_1:	Register:					
	It is a general-purpose register	R0 = 00000, R5 = 00101, R6 = 00110					
(16-22) 7	Source_2:	Register case:	Second pa	rt of the constant memory (i.e.)			
	It can be a general-purpose register or a constant memory	R0 = 00000, C[0x2][0x16]					
	location	R5= 00101, (16-22) = 001 0110			R5= 00101, (16-22) = 001 0110		001 0110
		R6= 00110					
(28 - 31) 4	Instruction Op. Code	F2F = 0xA					

(22 22)2	Santa mandan	00		A/ +	\	instruction \ /bdefect
(32 - 33)2	instr_marker			Access(load or store		
			_			in) (extra instruction)
		10 = nom	_	Access(load or store	e) (WILII EX	it)
(24)1	Used for		ediate		1	
(34)1		0 0 = Bogist	or doct	tination	1 - Mama	ry destination
. ,	destination type Prodicate register set (analyting a new flag) or Net used	0 = Regist			C1 = 01	<u>'</u>
(36-37) 2	Predicate register set (enabling a new flag) or Not used	C0 = 00 (k)	y ueia	uitj	C1 = 01 $C3 = 11$	
(38) 1	Sat pradicate register		o prodi	cate register set		
(39 - 43)5	Set predicate register predicate_condition					le predicate register set condition formula
(39 – 43)5	predicate_condition	encoding		-		
		0x00	never	always false (no	t usea)	0 (6.8 e/7) A O
		0x01	-	less tan		(S & ~Z) ^ O
		0x02	е	Equal		Z & ~S
		0x03	le	less than or e	•	S ^ (Z O)
		0x04	g	greater ta		~Z & ~(S ^ O)
		0x05		less or greater tan /		
		0x06	ge	greater than or	equal	~(S ^ O)
		0x07	lge	Ordered		~Z ~S
		0x08	u	Unordered		Z & S
		0x09	lu	less than or uno		S ^ O
		0x0a	eu	equal or unord		Z
		0x0b	leu	not greater t		Z (S ^ O)
		0x0c	gu	greater than or ur		~S ^ (Z O)
		0x0d	lgu	not equal t		~Z S
		0x0e	geu	not less ta		(~S Z) ^ O
			always			1
		0x10	0	Overflow		0
		0x11	С	carry / unsigned n		С
		0x12	а	unsigned abo		~Z & C
		0x13	S	sign / negat		S
		0x1c	ns	not sign / pos		~\$
		0x1d	na	unsigned not a		Z ~C
		0x1e	nc	not carry / unsigne		~C
		0x1f	no	no overflo		~0
(44 - 45) 2	Input predicate register to compare before to operate	CO = 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, Ne g	gative = 1 (default =	1)	
(60) 1	Not used	0				
(61 – 63)3	Sub_op_code	110				

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:

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_ma	arker	00 - norm					
				Access(load or store			y default)
			_	•	• •	i n) (extra instruction	1)
				Access(load or store	e) (with Exi	it)	
(24)4 Head fam		11 = imm	ediate	T	1		
(34)1 Used for		0			1		
(35)1 destinati		0 = Regist				ry destination	
(36-37) 2 Predicate	e register set (enabling a new flag) or Not used	CO = 00 (b	y deta	uit)	C1 = 01		
(20) 4		C2 = 10	1º		C3 = 11		1
· , ,	icate register			cate register set		le predicate register	set
(39 – 43)5 predicate	e_condition	encoding		Descriptio		condition formula	
		0x00	never	always false (no	t usea)	0	
		0x01	<u> </u>	less tan		(S & ~Z) ^ O	
		0x02	e	Equal		Z & ~S	
		0x03	le	less than or e	•	S ^ (Z O)	
		0x04	g	greater ta		~Z & ~(S ^ O)	
		0x05		less or greater tan /			
		0x06	ge	greater than or	equal	~(S ^ O)	
		0x07	lge	Ordered		~Z ~S	
		0x08	u	Unordered		Z & S	
		0x09	lu	less than or uno		S ^ O	
		0x0a	eu	equal or unord		Z	
		0x0b	leu	not greater t		Z (S ^ O)	
		0x0c	gu	greater than or ur		~S ^ (Z O)	
		0x0d	lgu	not equal t		~Z S	
		0x0e	geu	not less ta		(~S Z) ^ O	
			always			1	
		0x10	0	Overflow		0	
		0x11	С	carry / unsigned n		С	
		0x12	а	unsigned abo		~Z & C	
		0x13	S	sign / negat		S	
		0x1c	ns	not sign / pos		~\$	
		0x1d	na	unsigned not a		z ~C	
		0x1e	nc	not carry / unsigne		~C	
		0x1f	no	no overflo		~0	
(44 - 45) 2 Input pre	edicate register to compare before to operate	CO= 00			C1 = 01		
		C2= 10			C3 = 11		
	lue, purpose?	1					
, ,	g mechanism	00 = not r	oundin	g	11 = to z	zero	
(54) 1 Use of co	onstant memory for Source_2 or Source_3?	Yes = 1			No = 0		
(58) 1 Destiny t	to signed?	No = 0			Yes = 1 ((default = 1)	
(60) 1 Not used	Č	0				-	
` '	code	100					

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:

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	01 = norn	nal reg	Access(load or store Access(load or store Access(load or store	e) (with Jo i	in) (extra instruction	by default) n)
(34)1	Used for	0			1		
(35)1	destination type	0 = Regist	er dest	ination	1= Memo	ry destination	
(36-37) 2	Predicate register set (enabling a new flag) or Not used	CO = 00 (k)	y defa	ult)	C1 = 01		
		C2 = 10			C3 = 11		
(38) 1	Set predicate register	1 = Enabl	e predio	cate register set	0 = Disab	le predicate registe	r set
(39 – 43)5	predicate_condition	encoding	name	Description	n	condition formula	
		0x00	never	always false (no	t used)	0	
		0x01	I	less tan		(S & ~Z) ^ O	
		0x02	е	Equal		Z & ~S	
		0x03	le	less than or e	equal	S ^ (Z O)	
		0x04	g	greater ta		~Z & ~(S ^ O)	
		0x05	lg	less or greater tan ,	/ not equa		
		0x06	ge	greater than o	requal	~(S ^ O)	
		0x07	lge	Ordered		~Z ~S	
		0x08	u	Unordere	d	Z & S	
		0x09	lu	less than or und	ordered	S ^ O	
		0x0a	eu	equal or unor	dered	Z	
		0x0b	leu	not greater t	:han	Z (S ^ O)	
		0x0c	gu	greater than or u	nordered	~S ^ (Z O)	
		0x0d	lgu	not equal		~Z S	
		0x0e	geu	not less ta	ın	(~S Z) ^ O	
			always			1	
		0x10	0	Overflow		0	
		0x11	С	carry / unsigned r		С	
		0x12	a	unsigned ab		~Z & C	
		0x13	S	sign / negat		S	
		0x1c	ns	not sign / pos		~S	
		0x1d	na	unsigned not		Z ~C	
		0x1e	nc	not carry / unsign		~C	
		0x1f	no	no overflo		~0	
(44 - 45) 2	Input predicate register to compare before to operate	CO = 00			C1 = 01		
/AC\ 4	5th od rather annual 2	C2= 10			C3 = 11		
(46) 1	Fixed value, purpose?	1 - Yes			0 - 11 -		
(48) 1	From signed value Source_1	1 = Yes	- : اد میں م	-	0 = No		
(49-50) 2	Rounding mechanism Use of constant memory for Source 2 or Source 3?	00 = not r	oundin	g	11 = to 2	zero	
(54) 1	ose 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					

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], R16, R17, LT (600047C8 B01121FD)

(SASS_assembly_lib):

Formats:

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=00	0110
(9 - 15)7	Source_1: It is a general-purpose register	Register: R0 = 00000, F	R5 = 00101, R6 = 00110
(16-22) 7	Source_2: This can be from general-purpose registers	Register case:	Second part of the constant memory (i.e.)
	or a constant memory location.	R0 = 00000,	C[0x2][0x16]
		R5 = 00101, R6 = 00110	(16-22) = 001 0110
(23-27) 5	The high part of the constant memory location	First part of the constant memo	ry (i.e.) C[0x2][0x16]
		(23-27) = 0 0010	
(28 - 31) 4	Instruction Op. Code	FSET = 0xB	

/22 2010		00		A / I		1.6.5
(32 - 33)2	instr_marker		_	Access(load or store) (not ex	•	y default
			_	Access(load or store) (with Jo)
			_	Access(load or store) (with E	at)	
/2.4\1	Used for	11 = imm 0	ediate	1		
(34)1 (35)1		0 = Regist	or doct	ination 1- Mam	ory destination	
(36-37) 2	destination type Predicate register set (enabling a new flag) or Not used	$\mathbf{C0} = \mathbf{Regist}$			•	
(30-37) 2	Predicate register set (enabling a new hag) or Not used	C0 = 00 (k) $C2 = 10$	y dera	C3 = 1:		
(38) 1	Set predicate register		a predi		ole predicate register	cot
(39 – 43)5	predicate_condition	encoding	•	Description	condition formula	361
(33 43)3	predicate_condition	0x00	never	always false (not used)	0	
		0x01	ı	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		less or greater tan / not equa	· · · · · · · · · · · · · · · · · · ·	
		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	
		I 	always		1	
		0x10	0	Overflow	0	
		0x11	С	carry / unsigned not below	C	
		0x12	a	unsigned above	~Z & C	
		0x13	S	sign / negative	S	
		0v1c	nc	not sign / nositive	~c	
		0x1c	ns na	not sign / positive	~S 7 ~C	
		0x1d	na	unsigned not above	Z ~C	
		0x1d 0x1e	na nc	unsigned not above not carry / unsigned below		
(44 - 45) 2	Input predicate register to compare before to operate	0x1d	na	unsigned not above	Z ~C ~C	
(44 - 45) 2	Input predicate register to compare before to operate	0x1d 0x1e 0x1f	na nc	unsigned not above not carry / unsigned below no overflow	Z ~C ~C	
(44 - 45) 2 (46-50) 5	Input predicate register to compare before to operate Predicate condition to perform between the two main	0x1d 0x1e 0x1f C0 = 00	na nc no	unsigned not above not carry / unsigned below no overflow C1= 01	Z ~C ~C	
,		0x1d 0x1e 0x1f C0 = 00 C2 = 10	na nc no	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11	Z ~C ~C ~O	
	Predicate condition to perform between the two main	0x1d 0x1e 0x1f C0= 00 C2= 10 encoding	na nc no	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description	Z ~C ~C ~O	
,	Predicate condition to perform between the two main	0x1d 0x1e 0x1f C0 = 00 C2 = 10 encoding 0x00	na nc no name never	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal	Z ~C ~C ~O condition formula 0 (S & ~Z) ^ O Z & ~S	
	Predicate condition to perform between the two main	0x1d 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03	na nc no name never L (LT) E (EQ) Le	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal	Z ~C ~C ~O condition formula 0 (S & ~Z) ^ O Z & ~S S ^ (Z O)	
,	Predicate condition to perform between the two main	0x1d 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04	na nc no name never L (LT) E (EQ) Le G (GT)	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan	Z ~C ~C ~O condition formula 0 (S & ~Z) ^ O Z & ~S S ^ (Z O) ~Z & ~(S ^ O)	
	Predicate condition to perform between the two main	0x1d 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05	na nc no name never L (LT) E (EQ) Le G (GT)	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal	Z ~C	
	Predicate condition to perform between the two main	0x1d 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06	na nc no name never L (LT) E (EQ) Le G (GT) Lg ge	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal	Z ~C	
	Predicate condition to perform between the two main	0x1d 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07	na nc no name never L (LT) E (EQ) Le G (GT) Lg ge lge	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered	Z ~C ~C ~O condition formula 0 (S & ~Z) ^ O Z & ~S S ^ (Z O) ~Z & ~(S ^ O) all ~Z ~(S ^ O) ~Z ~S	
	Predicate condition to perform between the two main	0x1d 0x1e 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08	na nc no name never L (LT) E (EQ) Le G (GT) Lg ge lge u	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered Unordered	Z ~C	
	Predicate condition to perform between the two main	0x1d 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09	na nc no name never L (LT) E (EQ) Le G (GT) Lg ge lge u lu	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered Unordered less than or unordered	Z ~C	
,	Predicate condition to perform between the two main	0x1d 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a	na nc no no name never L (LT) E (EQ) Le G (GT) Lg ge lge u lu eu	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered Unordered less than or unordered equal or unordered	Z ~C	
	Predicate condition to perform between the two main	0x1d 0x1e 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b	na nc no name never L (LT) E (EQ) Le G (GT) Lg ge lge u lu eu leu	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered Unordered less than or unordered equal or unordered not greater than	Z ~C	
	Predicate condition to perform between the two main	0x1d 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c	na nc no name never L (LT) E (EQ) Le G (GT) Lg ge lge u lu eu leu gu	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered Unordered less than or unordered equal or unordered not greater than greater than or unordered	Z ~C	
,	Predicate condition to perform between the two main	0x1d 0x1e 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b	na nc no no never L (LT) E (EQ) Le G (GT) Lg ge lge u lu eu leu gu lgu	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered Unordered less than or unordered equal or unordered not greater than	Z ~C	
	Predicate condition to perform between the two main	0x1d 0x1e 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e	na nc no name never L (LT) E (EQ) Le G (GT) Lg ge lge u lu eu leu gu	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered Unordered less than or unordered equal or unordered not greater than greater than or unordered not equal to not less tan	Z ~C	
	Predicate condition to perform between the two main	0x1d 0x1e 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e	na nc no name never L (LT) E (EQ) Le G (GT) Lg ge lge u lu eu lu eu leu gu lgu geu	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered Unordered less than or unordered equal or unordered not greater than greater than or unordered not equal to not less tan	Z ~C	
, ,	Predicate condition to perform between the two main	0x1d 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f	na nc no no name never L (LT) E (EQ) Le G (GT) Lg ge lge u lu eu leu gu leu guu always	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered Unordered less than or unordered equal or unordered equal or unordered not greater than greater than or unordered not less tan always true (by default)	Z ~C	
, ,	Predicate condition to perform between the two main	0x1d 0x1e 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x10	na nc no no name never L (LT) E (EQ) Le G (GT) Lg ge lge u lu eu leu gu leu guu always o	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered Unordered less than or unordered equal or unordered equal or unordered not greater than greater than greater than always true (by default) Overflow	Z ~C	
, ,	Predicate condition to perform between the two main	0x1d 0x1e 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x10 0x11	na nc no no name never L (LT) E (EQ) Le G (GT) Lg ge lge u lu eu leu gu leu guu always o c	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered Unordered less than or unordered equal or unordered equal or unordered not greater than greater than or unordered not less tan always true (by default) Overflow carry / unsigned not below	Z ~C	
, ,	Predicate condition to perform between the two main	0x1d 0x1e 0x1e 0x1f C0= 00 C2= 10 encoding 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x10 0x11 0x12	na nc no no name never L (LT) E (EQ) Le G (GT) Lg ge lge u lu eu leu gu leu always o c a	unsigned not above not carry / unsigned below no overflow C1= 01 C3= 11 Description always false (not used) less tan Equal less than or equal greater tan less or greater tan / not equal greater than or equal Ordered Unordered less than or unordered equal or unordered equal or unordered not greater than greater than greater than always true (by default) Overflow carry / unsigned not below unsigned above	Z ~C	

		0x1e	nc	not carry / unsigne	d below	~C
		0x1f	no	no overflov	/	~0
(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)

Destiny_General_purpose_register

Instruction Op. Code

Source_1: It is a general-purpose register

(SASS_assembly_lib):

Formats:

(2 - 8)7

(9 - 15)7

(28 - 31)4

Not implemented yet...

Not implem	enteu 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)

Register: RCP = 0x9

R0= 000000, R5=000101, R6=000110...

R0= 00000 ..., **R5**= 00101, **R6**= 00110 ...

(32 - 33)2	instr_marker			Access(load or store			(by default)
			_	Access(load or store			n)
			•	Access(load or store	e) (with Exi	t)	
		11 = imm	ediate				
(34)1	Used for	0			1		
(35)1	destination type	0 = Regist	ter dest	ination	1= Memoi	ry destination	
(36-37) 2	Predicate register set (enabling a new flag) or Not used	CO = 00 (I	oy defa	ult)	C1 = 01		
		C2 = 10			C3 = 11		
(38) 1	Set predicate register	1 = Enabl	e predic	cate register set	0 = Disabl	e predicate registe	er set
(39 - 43)5	predicate_condition	encoding	name	Descriptio	n	condition formula	
		0x00	never	always false (no	t used)	0	
		0x01	I	less tan		(S & ~Z) ^ O	
		0x02	е	Equal		z & ~s	
		0x03	le	less than or e	qual	S ^ (Z O)	
		0x04	g	greater ta	n	~Z & ~(S ^ O)	
		0x05		less or greater tan /	not equal	~Z	
		0x06	ge	greater than or	equal	~(S ^ O)	
		0x07	lge	Ordered		~Z ~S	
		0x08	u	Unordered	d	Z & S	
		0x09	lu	less than or uno	rdered	S ^ O	
		0x0a	eu	equal or unor	dered	Z	
		0x0b	leu	not greater	tan	Z (S ^ O)	
		0x0c	gu	greater than or ur	nordered	~S ^ (Z O)	
		0x0d	lgu	not equal t	:0	~Z S	
		0x0e	geu	not less ta	n	(~S Z) ^ O	
		0x0f	always	always true (by	default)	1	
		0x10	О	Overflow		0	
		0x11	С	carry / unsigned n	ot below	С	
		0x12	а	unsigned abo	ove	~Z & C	
		0x13	S	sign / negat	ive	S	
		0x1c	ns	not sign / pos	itive	~S	
		0x1d	na	unsigned not a	bove	Z ~C	
		0x1e	nc	not carry / unsign	ed below	~C	
		0x1f	no	no overflo	w	~O	
(44 - 45) 2	Input predicate register to compare before to operate	CO = 00			C1 = 01		
		C2 = 10			C3 = 11		
(60) 1	Not used	0					
(61 - 63)3	Sub_op_code	011					

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:

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** \leftarrow **SIN (Source_f)**

Mnemonics:

Direct SIN: SIN Rx, Rx

Example (SASS from NVCC):

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

(SASS_assembly_lib):

Not_available

Note:

Bit(s)	Mnemonics	Commen	tary						
0	instr_is_long	0 = 32 bit	long.	1=	64 bit long. (by default)			
1	instr_is_flow	0 = Norm	al ins. (l	by default) 1 =	System ins. (flow contr	ol)			
(2 – 8) 7	Destiny_General_purpose_register	R0= 0000	000, R5=	000101, R6=000110					
(9 – 15)	Source_operand_register: it should be a	R0= 0000	000, R5	5= 00101, R6= 00110					
7	general purpose register.								
(16-27)	Not used	0000000	0000000000						
12									
(28 – 31) 4	Instruction Op. Code	SIN_OP =							
(32 - 33) 2	instr_marker		_	er access(load or store) (•			
			•	er access(load or store) (• •	ion)			
			•	er access(load or store) (with Exit)				
(2.4).4	Head for	11 imme	aiate						
(34)1	Used for	0	h	in a tila a					
(35)1	destination type	0 = Regis							
(36-37) 2	Predicate register set (enabling a new	C0 = 00 (by defai	uit)					
	flag) or Not used	C1 = 01							
		C2 = 10 C3 = 11							
(38) 1	Set predicate register		a tha sa	etting of a predicate regis	tor				
(39-43)5	predicate condition	encoding		Description	condition formula				
(33 43) 3	predicate_condition	0x00	never	always false (not use					
		0x01	I	less tan	(S & ~Z) ^ O				
		0x02	e	Equal	Z & ~S				
		0x02	le	less than or equal	S ^ (Z O)				
		0x04	g	greater tan	~Z & ~(S ^ O)				
		0x05		less or greater tan / not o	<u>'</u>				
		0x06	ge	greater than or equa					
		0x07	lge	Ordered	~Z ~S				
		0x08	u	Unordered	Z & S				
		0x09	lu	less than or unordere					
		0x0a	eu	equal or unordered					
		0x0b	leu	not greater than	Z (S ^ O)				
		0x0c	gu	greater than or unorde					
		0x0d	lgu	not equal to	~Z S				
		0x0e	geu	not less tan	(~S Z) ^ O				
		0x0f	always						
		0x10	0	Overflow	0				
		0x11	С	carry / unsigned not be					
		0x12	а	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 be	low ~C				
		0x1f	no	no overflow	~0				
(44 - 45) 2	Input predicate register to compare	CO = 00							
	before to operate	C2= 10							
(46 – 60) (61 – 63) 3	Not used Sub_op_code	100 SIN	0000 0	00					

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** \leftarrow **COS (Source_f)**

Mnemonics:

Direct COS: COS Rx, Rx

Example (SASS from NVCC):

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

(SASS_assembly_lib):

Not_available

Note:

Bit(s)	Mnemonics	Commentary								
0	instr_is_long	0 = 32 bit long. 1 = 64 bi			it long. (by default)					
1	instr_is_flow	0 = Norm	al ins. (tem ins. (flow contr	ol)					
(2-8)7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110								
(9 – 15)	Source_operand_register: it should be a	R0= 0000	R0= 000000, R5= 00101, R6= 00110							
7	general purpose register.									
(16-27)	Not used	0000000	0000000000							
12										
28 – 31) 4	Instruction Op. Code	COS_OP	= 0x9h							
(32 - 33) 2	instr_marker	00 norma	al regist	er access(load or store) (not	extra instruction) (b	/ default)				
		01 norma	al regist	er access(load or store) (with	Join) (extra instruct	ion)				
		10 norma	al regist	er access(load or store) (with	Exit)					
		11 imme	diate							
(34)1	Used for	0								
(35)1	destination type	0 = Regis	ter dest	ination						
(36-37) 2	Predicate register set (enabling a new	CO = 00 (by defa	ult)						
	flag) or Not used	C1 = 01								
		C2 = 10								
		C3 = 11								
(38) 1	Set predicate register	1 = Enable the setting of a predicate register								
39 – 43) 5	predicate_condition	encoding	name	Description	condition formula					
		0x00	never	always false (not used)	0					
		0x01	1	less tan	(S & ~Z) ^ O					
		0x02	е	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 equ	al ~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	0	Overflow	0					
		0x11	С	carry / unsigned not below						
		0x12	а	unsigned above	~Z & C					
		0x13	S	sign / negative	S					
		0x1c	ns	not sign / positive	~\$					
		0x1d	na	unsigned not above	Z ~C					
		0x1e	nc	not carry / unsigned below						
		0x1f	no	no overflow	~0					
(44 - 45) 2	Input predicate register to compare	CO = 00								
	before to operate	C2 = 10								
(46 - 60)	Not used	000 0000	0000 0	00						
(61 - 63)3	Sub_op_code	101 COS								

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 Rx, Rx, 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:

		_							
Bit(s)	Mnemonics	Commen				<u>.</u>			
0	instr_is_long	0 = 32 bit			= 64 bit long. (by defaul				
1	instr_is_flow				= System ins. (flow cont	rol)			
(2-8)7	Destiny_General_purpose_register			000101, R6=000110					
(9 – 15)	Source_operand_register: it should be a	R0= 0000	00, R5	5= 00101, R6= 00110					
7	general purpose register.	0000000	20000						
(16-27) 12	Not used	00000000	00000						
(28 - 31) 4	Instruction Op. Code	RRO_OP	RRO_OP = 0xBh						
(32 - 33) 2	instr_marker	00 norma	al registe	er access(load or store) (not extra instruction) (k	y default)			
		01 normal register access(load or store) (with Join) (extra instruction)							
		10 norma	al registe	er access(load or store) (with Exit)				
		11 immed	diate						
(34)1	Used for	0							
(35)1	destination type	0 = Regist	ter dest	ination					
(36-37) 2	Predicate register set (enabling a new	CO = 00 (I	by defai	ult)					
	flag) or Not used	C1 = 01							
		C2 = 10							
		C3 = 11							
(38) 1	Set predicate register			tting of a predicate regis					
(39 - 43)5	predicate_condition	encoding		Description	condition formula				
		0x00	Never	always false (not use	•				
		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		less or greater tan / not					
		0x06	Ge	greater than or equa	al ~(S ^ O)				
		0x07	Lge	Ordered	~Z ~S				
		0x08	U	Unordered	Z & S				
		0x09	Lu	less than or unordere	ed S ^ O				
		0x0a	Eu	equal or unordered					
		0x0b	Leu	not greater than	Z (S ^ O)				
		0x0c	Gu	greater than or unorde					
		0x0d	Lgu	not equal to	~Z S	_			
		0x0e	Geu	not less tan	(~S Z) ^ O	_			
			always			_			
		0x10	0	Overflow	0				
		0x11	С	carry / unsigned not be					
		0x12	A	unsigned above	~Z & C	-			
		0x13	S	sign / negative	S				
		0x1c	Ns	not sign / positive	~\$				
		0x1d	Na	unsigned not above		-			
		0x1e	Nc	not carry / unsigned be		-			
/ / / / / / / / / / / / / / / / / / / /		0x1f	No	no overflow	~0				
(44 - 45) 2	Input predicate register to compare before to operate	C0 = 00 C2 = 10							
(46-47) 2	Selector of the phase corrector	00 = SIN	(quadra	nt 1)					
		01 = Exp 2	2 (quad	rant 2)					
		10 = ? (quadrant 3)							
		11 = ? (quadrant 4)							
(46 – 60)	Not used	000 0000 0000 000							
(61 - 63) 3	Sub_op_code	110 RRO							

LG2 instruction:

Checked Not implemented

This instruction calculates the logarithm in a binary base of an input operand. $\textbf{Destiny_f} \leftarrow \textbf{Log_2} \ \textbf{(Source_f)}$

Mnemonics:

Direct LG2: LG2 Ry, Rx

Example (SASS from NVCC):

LG2 R0, R0; (60000780 90000001) LG2 R2, R2; (60000780 90000409)

(SASS_assembly_lib):

Not_available

Note:

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. (flo				ol)				
(2 – 8) 7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110								
(9 – 15)	Source_operand_register: it should be a	R0= 000000, R5= 00101, R6= 00110								
7	general purpose register.	, ,								
(16-27)	Not used	00000000	0000000000							
12										
28 – 31) 4	Instruction Op. Code	LG2_OP =	= 0x9h							
32 - 33) 2	instr_marker	00 norma	al regist	er access(load or store) (not	extra instruction) (b	default)				
·		01 norma	al regist	er access(load or store) (with	Join) (extra instruct	ion)				
		10 norma	al regist	er access(load or store) (with	Exit)					
		11 immed	diate							
(34)1	Used for	0								
(35)1	destination type	0 = Regist	ter dest	ination						
(36-37) 2	Predicate register set (enabling a new	CO = 00 (I	by defa	ult)						
	flag) or Not used	C1 = 01								
		C2 = 10								
		C3 = 11								
(38) 1	Set predicate register	1 = Enabl		1						
39 – 43) 5	predicate_condition	encoding		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		less or greater tan / not equ						
		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						
		0x0d	Lgu	not equal to	~Z S					
		0x0e	Geu	not less tan	(~S Z) ^ O					
			always		1					
		0x10	0	Overflow	0					
		0x11	С	carry / unsigned not below						
		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						
44 45\0	Land and distance of the target	0x1f	No	no overflow	~0					
44 - 45) 2	Input predicate register to compare	CO = 00								
(46 - 60)	before to operate	C2 = 10	0000	00						
(46 – 60)	Not used	000 0000	0000 0	00						
61 – 63) 3	Sub_op_code	011 LG2								

EX2 instruction:

Checked Not implemented

This instruction calculates the logarithm in a binary base of an input operand. $\textbf{Destiny_f} \leftarrow \textbf{Log_2} \ \textbf{(Source_f)}$

Mnemonics:

Direct EX2 EX2 Ry, Rx

Example (SASS from NVCC):

EX2 R3, R3; (C0000780 9000060d) EX2 R1, R2; (C0000780 90000405)

(SASS_assembly_lib):

 $Not_available$

Note:

Bit(s)	Mnemonics	Commentary							
0	instr_is_long				it long. (by default)				
1	instr_is_flow	0 = Normal ins. (by default) 1 = Syste			em ins. (flow control)				
(2-8)7	Destiny_General_purpose_register	R0= 000000, R5=000101, R6=000110							
(9 – 15)	Source_operand_register: it should be a	R0= 000000, R5= 00101, R6= 00110							
7	general purpose register.	,							
(16-27)	Not used	00000000	0000000000						
12									
(28 – 31) 4	Instruction Op. Code	EX2_OP =	0x9h						
(32 - 33) 2	instr_marker	00 norma	l registe	er access(load or store) (not	extra instruction) (b	y default)			
		01 norma	l registe	er access(load or store) (with	Join) (extra instruct	ion)			
		10 norma	l registe	er access(load or store) (with	Exit)				
		11 immed	liate						
(34)1	Used for	0							
(35)1	destination type	0 = Regist	er dest	ination					
(36-37) 2	Predicate register set (enabling a new	CO = 00 (k	y defai	ult)					
	flag) or Not used	C1 = 01							
		C2 = 10							
(0.5)		C3 = 11							
(38) 1	Set predicate register			tting of a predicate register					
(39 – 43) 5	predicate_condition	encoding		Description	condition formula				
		0x00	Never	always false (not used)	0				
		0x01	L	less tan	(S & ~Z) ^ O				
		0x02	Ε .	Equal	Z & ~S				
		0x03	Le	less than or equal	S ^ (Z O)				
		0x04	G	greater tan	~Z & ~(S ^ O)				
		0x05		less or greater tan / not equa					
		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 Z L (S A O)				
		0x0b	Leu	not greater than	Z (S ^ O)				
		0x0c	Gu	greater than or unordered					
		0x0d	Lgu	not equal to	~Z S				
		0x0e	Geu	not less tan	(~S Z) ^ O				
		0x0f 0x10	always	always true (by default) Overflow	1				
		0x10 0x11	O C	carry / unsigned not below	O C				
		0x11 0x12	A	unsigned above	~Z & C				
		0x12 0x13	S	sign / negative	S				
		0x13 0x1c	Ns		~S				
		0x1c 0x1d	Na Na	not sign / positive					
		0x10 0x1e	Nc Nc	unsigned not above not carry / unsigned below	Z ~C ~C				
		0x1e 0x1f	No	not carry / unsigned below no overflow	~0				
(AA AE\ 2	Input prodicate register to sample		NO	no overnow	1				
(44 - 45) 2	Input predicate register to compare	C0 = 00 C2 = 10							
(46 60)	Not used	000 0000	0000	00					
(46 – 60) (61 – 63) 3	Sub_op_code	110 EX2	00000	00					
DI - D313	SUD OD COUR	TTO CX5							

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** \leftarrow **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:

Bit(s)	Mnemonics	Commentary									
0	instr_is_long				oit 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)	Source_operand_register: it should be a	R0= 000000, R5= 00101, R6= 00110									
7	general purpose register.										
(16-27)	Not used	0000000000									
12											
(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 norma	al regist	er access(load or store	e) (with J	oin) (extra instruct	tion)				
		10 normal register access(load or store) (with Exit)									
		11 immed	diate								
(34)1	Used for	0									
(35)1	destination type	0 = Regist									
(36-37) 2	Predicate register set (enabling a new	CO = 00 (I	by defa	ult)							
	flag) or Not used	C1 = 01									
		C2 = 10									
1		C3 = 11									
(38) 1	Set predicate register			etting of a predicate re	_						
(39 – 43) 5	predicate_condition	encoding				condition formula					
		0x00	Never	always false (not u	used)	0					
		0x01	L	less tan		(S & ~Z) ^ O					
		0x02	Е	Equal		Z & ~S					
		0x03	Le	less than or equ	ual	S ^ (Z O)					
		0x04	G	greater tan		~Z & ~(S ^ O)					
		0x05		less or greater tan / n		~Z					
		0x06	Ge	greater than or ed	qual	~(S ^ O)					
		0x07	Lge	Ordered		~Z ~S					
		0x08	U	Unordered		Z & S					
		0x09	Lu	less than or unord		S ^ O					
		0x0a	Eu	equal or unorder		ZZ					
		0x0b	Leu	not greater tha		Z (S ^ O)					
		0x0c	Gu	greater than or uno	rdered	~S ^ (Z O)					
		0x0d	Lgu	not equal to		~Z S					
		0x0e	Geu	not less tan	6	(~S Z) ^ O					
			always		rault)	1					
		0x10	0	Overflow	. la a ! .	0					
		0x11	С	carry / unsigned not		C ~7.8.C					
		0x12	A	unsigned abov		~Z & C					
		0x13	S	sign / negative		S					
		0x1c	Ns	not sign / positi		~S					
		0x1d	Na	unsigned not abo		Z ~C					
		0x1e	Nc	not carry / unsigned	below	~C					
/44 45 6		0x1f	No	no overflow		~0					
(44 - 45) 2	Input predicate register to compare	C0 = 00									
(46 60)	before to operate	C2 = 10	00000	00							
(46-60)	Not used	000 0000	00000	00							
(61 – 63) 3	Sub_op_code	100 RSQ									