

9 INSTRUCTION OPCODES

This chapter lists and describes the opcodes that defines each of the instructions in the ADSP-219x's instruction set. This information is useful for debugging programs.

This chapter covers the following topics:

- [“Opcode Mnemonics” on page 9-1](#)
- [“Opcode Definitions” on page 9-20](#)

Opcode Mnemonics

This section lists, describes, and gives the numeric value for each opcode mnemonic.

Table 9-1. Opcode mnemonics

Mnemonic	Description	Details
AMF	Specifies an ALU or multiplier operation.	page 9-8
AS	Specifies whether ALU saturation mode is 0 = disabled 1 = enabled	page 9-40
B	Specifies whether branch is 0 = immediate 1 = delayed	page 9-32 page 9-41 page 9-42

Instruction Opcodes

Table 9-1. Opcode mnemonics (Cont'd)

Mnemonic	Description	Details
BIT	Specifies which interrupt to enable or disable (0–15).	page 9-60
BO	Specifies whether the supplied 4-bit constant in a type 9 instruction is 01 = as is 11 = negated	page 9-12 page 9-27
BR	Specifies whether bit-reverse addressing on DAG1 is 0 = disabled 1 = enabled	page 9-40
BSR	Specifies whether the secondary DAG address registers are 0 = disabled 1 = enabled	page 9-40
C	Specifies whether a software interrupt is 0 = set 1 = cleared	page 9-60
CC	Specifies the two LSBs of a 4-bit constant value in a type 9 instruction.	page 9-12 page 9-27
CF	Specifies whether to flush the instruction cache 0 = No flush 1 = flush	page 9-50
COND	Specifies one of the condition codes on which to base execution of the instruction.	page 9-11

Table 9-1. Opcode mnemonics (Cont'd)

Mnemonic	Description	Details
D	Specifies the direction of a data move. 0 = read 1 = write	page 9-22 page 9-35 page 9-51 page 9-54 page 9-57 page 9-58
DD	Specifies a destination data register for a DM bus transfer. 00 = AX0 01 = AX1 10 = MX0 11 = MX1	page 9-21
DDREG	Specifies a destination register for a register-to-register move operation.	page 9-13
DREG	Specifies an unrestricted data register (REG0 only).	page 9-13
DMI	Specifies a DAG index address register (I0–I3) for a DM bus transfer.	page 9-18 page 9-21
DMM	Specifies a DAG modify address register (M0–M3) for a DM bus transfer.	page 9-18 page 9-21
DRGP	Specifies a destination register group. 00 = REG0 01 = REG1 10 = REG2 11 = REG3	page 9-39
DRL	Specifies two MSBs of DREG data register address.	page 9-13
DRU	Specifies two LSBs of DREG data register address.	page 9-13
Exponent	Specifies an 8-bit, two's-complement shift value.	page 9-37

Instruction Opcodes

Table 9-1. Opcode mnemonics (Cont'd)

Mnemonic	Description	Details
G	Specifies a DAG register group. 0 = DAG1 1 = DAG2	page 9-17
IREG/MREG	Specifies DAG index and modify registers (I0–I7, M0–M7).	page 9-18
I	Specifies DAG index register (I0–I7).	page 9-17
Idle Value	Specifies a 4-bit value that defines an internal clock divisor.	page 9-53
INT	Specifies whether interrupts are globally 0 = disabled 1 = enabled	page 9-40
LPP	Specifies push/pop of the loop stacks. 0 = disabled 1 = enabled	page 9-50
M	Specifies a DAG modify register.	page 9-17
MM	Specifies whether MAC integer mode is 0 = disabled 1 = enabled	page 9-40
MOD DATA	Specifies an 8-bit, two's-complement immediate data value.	page 9-44 page 9-51
MS	Specifies memory bus for a memory data transfer 0 = 16-bit DM bus 1 = 24-bit PM bus	page 9-54

Table 9-1. Opcode mnemonics (Cont'd)

Mnemonic	Description	Details
OL	Specifies whether ALU overflow mode is 0 = disabled 1 = enabled	page 9-40
PD	Specifies a destination data register for a PM bus transfer. 00 = AY0 01 = AY1 10 = MY0 11 = MY1	page 9-21
PMI	Specifies a DAG index address register (I4–I7) for a PM bus transfer.	page 9-18
PMM	Specifies a DAG modify address register (M4–M7) for a PM bus transfer.	page 9-18
PPP	Specifies push/pop of the PC stack. 0 = disabled 1 = enabled	page 9-50
Q	Specifies the RTI mode. 0 = normal 1 = single-step	page 9-42
R	Specifies a result register. 0 = MR register 1 = SR register	page 9-49
REG	Specifies a core register of RGPx.	page 9-13
REG1	Specifies a register group 1 register	page 9-13 page 9-25

Instruction Opcodes

Table 9-1. Opcode mnemonics (Cont'd)

Mnemonic	Description	Details
REG2	Specifies a register group 2 register	page 9-13 page 9-25
REG3	Specifies a register group 3 register	page 9-13 page 9-56
RGP	Specifies a register group. 00 = REG0 01 = REG1 10 = REG2 11 = REG3.	page 9-13
S	Specifies the branch type. 0 = jump 1 = call	page 9-32 page 9-41
SDREG	Specifies the source data register for a data move operation.	page 9-13
SF	Specifies a shift function.	page 9-15
SPP	Specifies push/pop of the status stack. 0 = disabled 1 = enabled	page 9-50
SR	Specifies whether the secondary data registers are 0 = disabled 1 = enabled	page 9-40

Table 9-1. Opcode mnemonics (Cont'd)

Mnemonic	Description	Details
SRGP	Specifies a source register group for a data move operation. 00 = REG0 01 = REG1 10 = REG2 11 = REG3	page 9-39
SWCD	Specifies a 4-bit nonfunctional value used by ADI tools only.	page 9-52
T	Specifies the return type. 0 = RTS 1 = RTI	page 9-42
TERM	Specifies the terminating condition for the type 11 instruction. 1110 = NOT CE 1111 = TRUE	page 9-34
TI	Specifies whether the timer is 0 = disabled 1 = enabled	page 9-40
U	Specifies whether the DAG index register is 0 = premodified with no update 1 = postmodified with update	page 9-54
XOP	Specifies a restricted data register used to supply the x operand value in a multifunction or conditional instruction.	page 9-19
XREG	Specifies the source register (REG0) in a shift function.	page 9-13

Instruction Opcodes

Table 9-1. Opcode mnemonics (Cont'd)

Mnemonic	Description	Details
Y0	Specifies whether the source of the x-operand is 0 = data register 1 = 0 (explicit value)	page 9-11
YOP	Specifies a restricted data register used to supply the y operand value in a multifunction or conditional instruction.	page 9-19
YREG	Specifies the destination register (REG0) in a shift function.	page 9-13 page 9-11
YY	Specifies the two MSBs of a 4-bit constant value in a type 9 instruction.	page 9-12 page 9-27
Z	Specifies a result or feedback register 0 = result register 1 = feedback register	page 9-23 page 9-26 page 9-27 page 9-11

ALU or Multiplier Function (AMF) Codes

Table 9-2 on page 9-9 lists the AMF codes used by these instruction types:

- “Type 1: Compute | DregX«...DM | DregY«...PM” on page 9-21
- “Type 4: Compute | Dreg «...» DM/PM” on page 9-23
- “Type 8: Compute | Dreg1 «... Dreg2” on page 9-26
- “Type 9: Compute” on page 9-27

Table 9-2. ALU/multiplier function (AMF) codes

Code	Function	Description
Multiplier functions		
00000	NOP	No operation
00001	$X * Y$ (RND)	Multiply
00010	$MR + X * Y$ (RND)	Multiply and accumulate
00011	$MR - X * Y$ (RND)	Multiply and subtract
00100	$X * Y$ (SS)	Multiply
00101	$X * Y$ (SU)	Multiply
00110	$X * Y$ (US)	Multiply
00111	$X * Y$ (UU)	Multiply
01000	$MR + X * Y$ (SS)	Multiply and accumulate
01001	$MR + X * Y$ (SU)	Multiply and accumulate
01010	$MR + X * Y$ (US)	Multiply and accumulate
01011	$MR + X * Y$ (UU)	Multiply and accumulate
01100	$MR - X * Y$ (SS)	Multiply and subtract
01101	$MR - X * Y$ (SU)	Multiply and subtract
01110	$MR - X * Y$ (US)	Multiply and subtract
01111	$MR - X * Y$ (UU)	Multiply and subtract
(RND) = round results; (SS) = both operands signed; (SU) = x operand signed, y operand unsigned; (US) = x operand unsigned, y operand signed; (UU) = both operands unsigned		

Instruction Opcodes

Table 9-2. ALU/multiplier function (AMF) codes (Cont'd)

Code	Function	Description
ALU functions		
10000	Y	PASS/CLEAR
10001	Y + 1	PASS
10010	X + Y + C	Add with carry
10011	X + Y	Add
10100	NOT Y	Negate
10101	- Y	PASS
10110	X - Y + C - 1	Subtract (X-Y) with borrow
10111	X - Y	Subtract
11000	Y - 1	PASS
11001	Y - X	Subtract
11010	Y - X + C - 1	Subtract (Y-X) with borrow
11011	NOT X	Negate
11100	X AND Y	AND/test bit,clear bit
11101	X OR Y	OR/set bit
11110	X XOR Y	XOR/toggle bit
11111	ABS X	Absolute value
(RND) = round results; (SS) = both operands signed; (SU) = x operand signed, y operand unsigned; (US) =x operand unsigned, y operand signed; (UU) = both operands unsigned		

Condition Codes

Table 9-3 on page 9-11 lists the condition codes used by these instruction types:

- “Type 9: Compute” on page 9-27
- “Type 10: Direct Jump” on page 9-32
- “Type 16: Shift Reg0” on page 9-38
- “Type 19: Indirect Jump/Call” on page 9-41
- “Type 20: Return” on page 9-42
- “Type 36: Long Jump/Call” on page 9-59
- “Type 11: Do ... Until” on page 9-34
uses NOT CE and TRUE only for the terminating condition.

Table 9-3. Condition codes

Code	Condition	Description
0000	EQ	Equal to 0 (= 0)
0001	NE	Not equal to 0 (\neq 0)
0010	GT	Greater than 0 (>0)
0011	LE	Less than or equal to 0 (\leq 0)
0100	LT	Less than 0 (<0)
0101	GE	Greater than or equal to 0 (\geq 0)
0110	AV	ALU overflow
0111	NOT AV	Not ALU overflow

Instruction Opcodes

Table 9-3. Condition codes (Cont'd)

Code	Condition	Description
1000	AC	ALU carry
1001	NOT AC	Not ALU carry
1010	SWCOND	CCODE register condition
1011	NOT SWCOND	Not CCODE register condition
1100	MV	MAC overflow
1101	NOT MV	Not MAC overflow
1110	NOT CE	Counter not expired
1111	TRUE	Always true

Constant Codes

Table 9-4 lists the valid constants used by “Type 9: Compute” on page 9-27. As shown, the YY/CC bits determine the constant value and the B0 bits determine the sign of the value.

Table 9-4. Constants

Code		Decimal / Hex	Decimal / Hex
YY	CC	B0 = 01	B0 = 11
00	00	1 / 0x0001	-2 / 0xFFFE
00	01	2 / 0x0002	-3 / 0xFFFD
00	10	4 / 0x0004	-5 / 0xFFFB
00	11	8 / 0x0008	-9 / 0xFFF7

Table 9-4. Constants (Cont'd)

Code		Decimal / Hex	Decimal / Hex
YY	CC	BO = 01	BO = 11
01	00	16 / 0x0010	-17 / 0xFFEF
01	01	32 / 0x0020	-33 / 0xFFDF
01	10	64 / 0x0040	-65 / 0xFFBF
01	11	128 / 0x0080	-129 / 0xFF7F
10	00	256 / 0x0100	-257 / 0xFEFF
10	01	512 / 0x0200	-513 / 0xFDFE
10	10	1024 / 0x0400	-1025 / 0xFBFF
10	11	2048 / 0x0800	-2049 / 0xF7FF
11	00	4096 / 0x1000	-4097 / 0xEFFF
11	01	8192 / 0x2000	-8193 / 0xDFFF
11	10	16384 / 0x4000	-16385 / 0xBFFF
11	11	-32768 / 0x8000	+32767 / 0x7FFF

Core Register Codes

Table 9-5 on page 9-14 list the core registers and their addresses. The complete address of any individual register is formed by appending the register's address bits to its RGP bits, so, for example, the address of the I2 register is 010010. The opcode mnemonics DREG, DDREG, SDREG, XREG, and YREG and the following instruction types reference these registers by their address bits:

- “Type 3: Dreg/Ireg/Mreg «…» DM/PM” on page 9-22

Instruction Opcodes

- “Type 4: Compute | Dreg «…» DM/PM” on page 9-23
- “Type 6: Dreg «…» Data16” on page 9-24
- “Type 8: Compute | Dreg1 «…» Dreg2” on page 9-26
- “Type 9: Compute” on page 9-27
- “Type 12: Shift | Dreg «…» DM/PM” on page 9-35
- “Type 14: Shift | Dreg1 «…» Dreg2” on page 9-36
- “Type 15: Shift Data8” on page 9-37
- “Type 16: Shift Reg0” on page 9-38
- “Type 17: Any Reg «…» Any Reg” on page 9-39
- “Type 34: Dreg «…» IOreg” on page 9-57
- “Type 35: Dreg «…»Sreg” on page 9-58

Table 9-5. Core registers

RGP/Address	Register Groups (RGP)			
Address	00 (REG0)	01 (REG1)	10 (REG2)	11 (REG3)
0000	AX0	I0	I4	ASTAT
0001	AX1	I1	I5	MSTAT
0010	MX0	I2	I6	SSTAT
0011	MX1	I3	I7	LPSTACKP
0100	AY0	M0	M4	CCODE
0101	AY1	M1	M5	SE
0110	MY0	M2	M6	SB

Table 9-5. Core registers (Cont'd)

RGP/Address	Register Groups (RGP)			
Address	00 (REG0)	01 (REG1)	10 (REG2)	11 (REG3)
0111	MY1	M3	M7	PX
1000	MR2	L0	L4	DMPG1
1001	SR2	L1	L5	DMPG2
1010	AR	L2	L6	IOPG
1011	SI	L3	L7	IJPG
1100	MR1	IMASK	Reserved	Reserved
1101	SR1	IRPTL	Reserved	Reserved
1110	MR0	ICNTL	CNTR	Reserved
1111	SR0	STACKA	LPSTACKA	STACKP

SF Function Codes

Table 9-6 list the shift function (SF) codes used by these instruction types:

- “Type 12: Shift | Dreg «...» DM/PM” on page 9-35
- “Type 14: Shift | Dreg1 «... Dreg2” on page 9-36
- “Type 15: Shift Data8” on page 9-37
—shift functions (codes 0000-0111) only
- “Type 16: Shift Reg0” on page 9-38

Instruction Opcodes

Table 9-6. SF codes

Code	Function
0000	LSHIFT (HI)
0001	LSHIFT (HI, OR)
0010	LSHIFT (LO)
0011	LSHIFT (LO, OR)
0100	ASHIFT (HI)
0101	ASHIFT (HI, OR)
0110	ASHIFT (LO)
0111	ASHIFT (LO, OR)
1000	NORM (HI)
1001	NORM (HI, OR)
1010	NORM (LO)
1011	NORM (LO, OR)
1100	EXP (HI)
1101	EXP (HIX)
1110	EXP (LO)
1111	Derive Block Exponent

I and M Codes

Table 9-7 on page 9-17 lists the DAG index and modify register codes used by the following instruction types. The G bit (DAG1/DAG2) determines which group of I (index) and M (modify) registers.

- “Type 4: Compute | Dreg «…» DM/PM” on page 9-23.
- “Type 12: Shift | Dreg «…» DM/PM” on page 9-35
- “Type 19: Indirect Jump/Call” on page 9-41
- “Type 21: Modify DagI” on page 9-43
- “Type 21a: Modify DagI” on page 9-44
- “Type 22: DM/PM «… Data16” on page 9-45
- “Type 29: Dreg «…» DM” on page 9-51
- “Type 32: Any Reg «…» PM/DM” on page 9-54

Table 9-7. I and M codes

Code	DAG1 (G=0)		DAG2 (G=1)	
	I	M	I	M
00	I0	M0	I4	M4
01	I1	M1	I5	M5
10	I2	M2	I6	M6
11	I3	M3	I7	M7

Instruction Opcodes

DMI, DMM, PMI, and PMM Codes

Table 9-8 lists the DAG index and modify register codes used by “Type 1: Compute | DregX«…DM | DregY«…PM” on page 9-21.

Table 9-8. DMI, DMM, PMI, PMM codes

Code	DMI	DMM	PMI	PMM
00	I0	M0	I4	M4
01	I1	M1	I5	M5
10	I2	M2	I6	M6
11	I3	M3	I7	M7

IREG/MREG Codes

Table 9-9 lists the Ireg and Mreg codes used by “Type 3: Dreg/Ireg/Mreg «…» DM/PM” on page 9-22 to specify a DAG index or modify register.

Table 9-9. Ireg, Mreg codes

Code	Register	Code	Register
0000	I0	1000	M0
0001	I1	1001	M1
0010	I2	1010	M2
0011	I3	1011	M3
0100	I4	1100	M4
0101	I5	1101	M5

Table 9-9. Ireg, Mreg codes (Cont'd)

Code	Register	Code	Register
0110	I6	1110	M6
0111	I7	1111	M7

XOP and YOP Codes

Table 9-10 on page 9-19 lists the XOP and YOP codes used by these instructions:

- “Type 1: Compute | DregX«...DM | DregY«...PM” on page 9-21
- “Type 4: Compute | Dreg «...» DM/PM” on page 9-23
- “Type 8: Compute | Dreg1 «... Dreg2” on page 9-26
- “Type 9: Compute” on page 9-27
- “Type 12: Shift | Dreg «...» DM/PM” on page 9-35
- “Type 23: Divide primitive, DIVQ” on page 9-47
- “Type 24: Divide primitive, DIVS” on page 9-48

Table 9-10. XOP/YOP codes

Code	XOP			Code	YOP	
	ALU	MAC	Shift		ALU	MAC
000	AX0	MX0	SI	00	AY0	MY0
001	AX1	MX1	SR2	01	AY1	MY1
010	AR	AR	AR	10	AF	SR1

Instruction Opcodes

Table 9-10. XOP/YOP codes

Code	XOP			Code	YOP	
	ALU	MAC	Shift		ALU	MAC
011	MR0	MR0	MR0	11	0	0
100	MR1	MR1	MR1			
101	MR2	MR2	MR2			
110	SR0	SR0	SR0			
111	SR1	SR1	SR1			

Opcode Definitions

For each instruction opcode, this section provides the following information:

- Opcode bits
- Syntax
- See also (related instruction reference pages)

For mnemonics definitions, see [“Opcode Mnemonics” on page 9-1](#).

Type 1: Compute | DregX«...DM | DregY«...PM

Multifunction ALU/MAC with DM and PM dual read with DAG1 and DAG2 postmodify

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12	11
1	1	PD		DD		AMF					YOP	

10	9	8	7	6	5	4	3	2	1	0
XOP			PMI		PMM		DMI		DMM	

or for NOP only:

23	22	21	20	19	18	17	16	15	14	13	12	11
1	1	PD		DD		0	0	0	0	0		

10	9	8	7	6	5	4	3	2	1	0
			PMI		PMM		DMI		DMM	

SYNTAX

|<ALU>, <MAC>|, Xop = DM(Ia += Mb), Yop = PM(Ic += Md);

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Compute with Dual Memory Read” on page 6-3](#)
- [“Dual Memory Read” on page 6-7](#)

Instruction Opcodes

Type 3: Dreg/Ireg/Mreg «...» DM/PM

Register read/write to immediate 16-bit address

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12	11
1	0	1	D	16-bit address								

10	9	8	7	6	5	4	3	2	1	0
16-bit address							IREG/MREG			

or:

23	22	21	20	19	18	17	16	15	14	13	12	11
1	0	0	D	16-bit address								

10	9	8	7	6	5	4	3	2	1	0
16-bit address							DREG			

SYNTAX

|DM(<Addr16>| = |Dreg, Ireg, Mreg|;

|Dreg, Ireg, Mreg| = |DM(<Addr16>|;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Direct Memory Read/Write—Immediate Address” on page 7-24](#)

Type 4: Compute | Dreg «…» DM/PM

Multifunction ALU/MAC with memory read or write using DAG
postmodify

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12	11
0	1	1	G	D	Z			AMF				YOP

10	9	8	7	6	5	4	3	2	1	0
XOP			DREG				I		M	

SYNTAX

|<ALU>, <MAC> |, Dreg = |DM(Ia += Mb), PM(Ic += Md)|;

|<ALU>, <MAC> |, |DM(Ia += Mb), PM(Ic += Md)| = Dreg;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Compute with Memory Read” on page 6-10](#)
- [“Compute with Memory Write” on page 6-14](#)

Instruction Opcodes

Type 6: Dreg «… Data16

Immediate register group 0 (Dreg) register load

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12	11
0	1	0	0	16-bit data								

10	9	8	7	6	5	4	3	2	1	0
16-bit data							DREG			

SYNTAX

<Dreg> = <Data16>;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Direct Register Load” on page 7-27](#)

Type 7: Reg1/2 «… Data16

Immediate register group 1 or 2 (Ireg, Mreg, Lreg, IMASK, IRPTL, ICNTL, CNTR, STACKA, LPCSTACKA) register load

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12	11	
0	1	0	1	16-bit data									

10	9	8	7	6	5	4	3	2	1	0
16-bit data							REG1			

or:

23	22	21	20	19	18	17	16	15	14	13	12	11	
0	0	1	1	16-bit data									

10	9	8	7	6	5	4	3	2	1	0
16-bit data							REG2			

SYNTAX

| <Reg1>, <Reg2> | = <Data16>;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Direct Register Load” on page 7-27](#)

Instruction Opcodes

Type 8: Compute | Dreg1 «... Dreg2

ALU/MAC with data register move

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	1	0	1	Z							YOP
				AMF								

10	9	8	7	6	5	4	3	2	1	0
XOP			DDREG				SDREG			

or, generate ALU/MAC status only

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	1	0	1	Z							YOP
				AMF								

10	9	8	7	6	5	4	3	2	1	0
XOP			1	0	1	0	1	0	1	0

SYNTAX

| <ALU>, <MAC> |, Dreg = Dreg;

NONE = ALU (Xop, Yop);

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Compute with Register to Register Move” on page 6-18](#)
- [“Generate ALU Status Only: NONE” on page 3-44](#)

Type 9: Compute

Conditional ALU/MAC

OPCODE

Conditional ALU/MAC

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	1	0	0	Z							YOP
								AMF				

10	9	8	7	6	5	4	3	2	1	0
		XOP	0	0	0	0				COND

Conditional ALU/MAC operations using constant YOP

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	1	0	0	Z							YY
								AMF				

10	9	8	7	6	5	4	3	2	1	0
		XOP		CC		BO				COND

Conditional ALU/MAC operations with YOP=0

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	1	0	0	Z						1	1
								AMF				

10	9	8	7	6	5	4	3	2	1	0
		XOP	0	0	0	0				COND

Instruction Opcodes

Conditional MAC squaring operations only

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	1	0	0	Z			AMF			0	0

10	9	8	7	6	5	4	3	2	1	0
	XOP		0	0	0	1		COND		

SYNTAX

```
[IF Cond] |AR, AF| = Xop + |Yop, Yop + C, C, Const, Const + C|;  
[IF Cond] |AR, AF| = Xop - |Yop, Yop+C-1, +C-1, Const, Const+C-1|;  
[IF Cond] |AR, AF| = Yop - |Xop, Xop+C-1|;  
[IF Cond] |AR, AF| = - |Xop+C-1, Xop+Const, Xop+Const+C-1|;  
[IF Cond] |AR, AF| = Xop |AND, OR, XOR| |Yop, Const|;  
[IF Cond] |AR, AF| = PASS |Xop, Yop, Const|;  
[IF Cond] |AR, AF| = NOT |Xop, Yop|;  
[IF Cond] |AR, AF| = ABS Xop;  
[IF Cond] |AR, AF| = Yop +1;  
[IF Cond] |AR, AF| = Yop -1;  
[IF Cond] |MR, SR| = Xop * Yop [(|RND, SS, SU, US, UU|)];  
[IF Cond] |MR, SR| = Yop * Xop [(|RND, SS, SU, US, UU|)];  
[IF Cond] |MR, SR| = |MR, SR| + Xop * Yop [(|RND, SS, SU, US, UU|)];  
[IF Cond] |MR, SR| = |MR, SR| + Yop * Xop [(|RND, SS, SU, US, UU|)];  
[IF Cond] |MR, SR| = |MR, SR| - Xop * Yop [(|RND, SS, SU, US, UU|)];  
[IF Cond] |MR, SR| = |MR, SR| - Yop * Xop [(|RND, SS, SU, US, UU|)];  
[IF Cond] |MR, SR| = 0;  
[IF Cond] MR = MR [(RND)];  
[IF Cond] SR = SR [(RND)];
```

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Add/Add with Carry” on page 3-5](#)
- [“Subtract X-Y/Subtract X-Y with Borrow” on page 3-8](#)
- [“Subtract Y-X/Subtract Y-X with Borrow” on page 3-12](#)

- “Bitwise Logic: AND, OR, XOR” on page 3-15
- “Bit Manipulation: TSTBIT, SETBIT, CLRBIT, TGLBIT” on page 3-18
- “Clear: PASS” on page 3-20
- “Negate: NOT” on page 3-23
- “Absolute Value: ABS” on page 3-26
- “Increment” on page 3-29
- “Decrement” on page 3-32
- “Multiply” on page 4-8
- “Multiply with Cumulative Add” on page 4-11
- “Multiply with Cumulative Subtract” on page 4-14
- “MAC Clear” on page 4-17
- “MAC Round/Transfer” on page 4-19

Instruction Opcodes

Type 9a: Compute

Unconditional ALU/MAC

OPCODE

Register file ALU/MAC

23	22	21	20	19	18	17	16	15	14	13	12
0	0	1	0	0	Z			AMF			Y0

11	10	9	8	7	6	5	4	3	2	1	0
	XREG					1	0			YREG	

Register file ALU/MAC with XREG=0

23	22	21	20	19	18	17	16	15	14	13	12
0	0	1	0	0	Z			AMF			Y0

11	10	9	8	7	6	5	4	3	2	1	0
						1	0			YREG	

SYNTAX

```
|AR, AF| = Dreg1 + |Dreg2, Dreg2 + C, C| ;  
|AR, AF| = Dreg1 - |Dreg2, Dreg2 + C -1, +C -1| ;  
|AR, AF| = Dreg2 - |Dreg1, Dreg1 + C -1| ;  
|AR, AF| = Dreg1 |AND, OR, XOR| Dreg2 ;  
|AR, AF| = PASS |Dreg1, Dreg2, Const| ;  
|AR, AF| = PASS 0 ;  
|AR, AF| = NOT |Dreg| ;  
|AR, AF| = ABS Dreg ;  
|AR, AF| = Dreg +1 ;  
|AR, AF| = Dreg -1 ;  
|MR, SR| = Dreg1 * Dreg2 [(|RND, SS, SU, US, UU|)] ;  
|MR, SR| = |MR, SR| + Dreg1 * Dreg2 [(|RND, SS, SU, US, UU|)] ;  
|MR, SR| = |MR, SR| - Dreg1 * Dreg2 [(|RND, SS, SU, US, UU|)] ;
```

SEE ALSO

- “Opcode Mnemonics” on page 9-1
- “Add/Add with Carry” on page 3-5
- “Subtract X-Y/Subtract X-Y with Borrow” on page 3-8
- “Subtract Y-X/Subtract Y-X with Borrow” on page 3-12
- “Bitwise Logic: AND, OR, XOR” on page 3-15
- “Clear: PASS” on page 3-20
- “Negate: NOT” on page 3-23
- “Absolute Value: ABS” on page 3-26
- “Increment” on page 3-29
- “Decrement” on page 3-32
- “Multiply” on page 4-8
- “Multiply with Cumulative Add” on page 4-11
- “Multiply with Cumulative Subtract” on page 4-14

Instruction Opcodes

Type 10: Direct Jump

13-bit relative conditional/unconditional jump with delayed branch option

OPCODE

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	0	1	1	0	B	13-bit address					

10	9	8	7	6	5	4	3	2	1	0
13-bit address							COND			

SYNTAX

[IF Cond] JUMP <Reladdr13> [(DB)];

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Direct JUMP \(PC relative\)” on page 8-27](#)

Type 10a: Direct Jump/Call

16-bit relative conditional/unconditional jump with delayed branch option

OPCODE

23	22	21	20	19	18	17	16	15	14	13	12	11	
0	0	0	1	1	1	14-bit address							

10	9	8	7	6	5	4	3	2	1	0
14-bit address							B	S	2MSBs	

SYNTAX

CALL <Reladdr16> [(DB)];

JUMP <Reladdr16> [(DB)];

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“CALL \(PC relative\)” on page 8-30](#)
- [“JUMP \(PC relative\)” on page 8-34](#)

Instruction Opcodes

Type 11: Do ... Until

12-bit relative conditional DO

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	0	1	0	1	1	0	12-bit address				

10	9	8	7	6	5	4	3	2	1	0
12-bit address							TERM			

SYNTAX

DO <Reladdr12> UNTIL [CE, FOREVER];

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“DO UNTIL \(PC relative\)” on page 8-22](#)

Type 12: Shift | Dreg «…» DM/PM

Shift with memory read/write using DAG postmodify

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	0	1	0	0	1	G			SF		D

10	9	8	7	6	5	4	3	2	1	0	
XOP			DREG				I		M		

SYNTAX

<SHIFT> , Dreg = |DM(Ia += Mb), PM(Ic += Md)|;

<SHIFT> , |DM(Ia += Mb), PM(Ic += Md)| = Dreg;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Compute with Memory Read” on page 6-10](#)
- [“Compute with Memory Write” on page 6-14](#)
- [“XOP/YOP codes” on page 9-19](#)

Instruction Opcodes

Type 14: Shift | Dreg1 «... Dreg2

Register file shift with data register move

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	1	0	1	0	0	SF			

11	10	9	8	7	6	5	4	3	2	1	0
XREG				DDREG				SDREG			

SYNTAX

<SHIFT>, Dreg = Dreg;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Compute with Register to Register Move” on page 6-18](#)

Type 15: Shift Data8

Immediate register file shift

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	1	1	1	1	SF			

11	10	9	8	7	6	5	4	3	2	1	0
XREG				Exponent							

SYNTAX

SR = [SR OR] ASHIFT BY <Imm8> [(|HI, LO|)];

SR = [SR OR] LSHIFT BY <Imm8> [(|HI, LO|)];

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Arithmetic Shift Immediate” on page 5-8](#)
- [“Logical Shift Immediate” on page 5-12](#)

Instruction Opcodes

Type 16: Shift Reg0

Conditional register file shift

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	1	1	1	0	SF			
11	10	9	8	7	6	5	4	3	2	1	0
XREG								COND			

SYNTAX

[IF Cond] SR = [SR OR] ASHIFT Dreg [(|HI, LO|)];
[IF Cond] SR = [SR OR] LSHIFT Dreg [(|HI, LO|)];
[IF Cond] SR = [SR OR] NORM Dreg [(|HI, LO|)];
[IF Cond] SE = [SR OR] EXP Dreg [(|HIX, HI, LO|)];
[IF Cond] SB = [SR OR] EXPADJ Dreg;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Arithmetic Shift” on page 5-6](#)
- [“Logical Shift” on page 5-10](#)
- [“Normalize” on page 5-14](#)
- [“Exponent Derive” on page 5-20](#)
- [“Exponent \(Block\) Adjust” on page 5-23](#)

Type 17: Any Reg «... Any Reg

General register move

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	1	1	0	1				

11	10	9	8	7	6	5	4	3	2	1	0
DRGP		SRGP		DDREG				SDREG			

SYNTAX

Reg = Reg;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Register to Register Move” on page 7-22](#)

Instruction Opcodes

Type 18: Mode Change

Mode control

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	1	1	0	0	TI		MM	

11	10	9	8	7	6	5	4	3	2	1	0
AS		OL		BR		SR		SD		INT	

SYNTAX

ENA | TI, MM, AS, OL, BR, SR, SD, INT | ;

DIS | TI, MM, AS, OL, BR, SR, SD, INT | ;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Mode Control” on page 8-69](#)

Type 19: Indirect Jump/Call

Conditional indirect jump/call with delayed branch option

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	1	0	1	1	B	S	G	

11	10	9	8	7	6	5	4	3	2	1	0	
				COND				I				

SYNTAX

[IF Cond] CALL <Ireg> [(DB)];

[IF Cond] JUMP <Ireg> [(DB)];

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Indirect CALL” on page 8-42](#)
- [“Indirect JUMP” on page 8-45](#)

Instruction Opcodes

Type 20: Return

Conditional return from interrupt/return from subroutine with delayed branch option

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	1	0	1	0	B	T	Q	

11	10	9	8	7	6	5	4	3	2	1	0
				COND							

SYNTAX

[IF Cond] RTI [(DB)];

[IF Cond] RTS [(DB)];

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Return from Interrupt” on page 8-48](#)
- [“Return from Subroutine” on page 8-52](#)

Type 21: Modify Dagl

DAG modify

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	0	0	1	1		G	

11	10	9	8	7	6	5	4	3	2	1	0
								I		M	

SYNTAX

```
|MODIFY (Ia += Mb), MODIFY (Ic += Md)|;
```

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Modify Address Register—indirect” on page 7-63](#)

Instruction Opcodes

Type 21a: Modify Dagi

DAG modify immediate value

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	0	0	1	0		G	

11	10	9	8	7	6	5	4	3	2	1	0
MOD DATA								I			

SYNTAX

MODIFY (Ireg += <Imm8>);

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Modify Address Register—direct” on page 7-65](#)

Type 22: DM/PM «··· Data16

16-bit immediate data indirect memory write (two-word instruction)
using DAG postmodify addressing

OPCODE

First word: 16-bit data write

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	1	1	1	1	0	G	
11	10	9	8	7	6	5	4	3	2	1	0
8 DATA LSBs								I	M		

Second word: 16-bit data write

23	22	21	20	19	18	17	16	15	14	13	12
				8 DATA MSBs							
11	10	9	8	7	6	5	4	3	2	1	0

SYNTAX

```
|DM(Ia += Mb), DM (Ic += Md)| = <Data16>;
```

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Indirect 16-bit Memory Write—immediate data” on page 7-55](#)

Instruction Opcodes

Type 22a: DM/PM «... Data24

24-bit immediate data indirect memory write (two-word instruction)
using DAG postmodify addressing

OPCODE

First word: 24-bit data write

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	1	1	1	1	1	G	

11	10	9	8	7	6	5	4	3	2	1	0
8 DATA MidSBs								I	M		

Second word: 24-bit data write

23	22	21	20	19	18	17	16	15	14	13	12
				8 DATA MSBs							

11	10	9	8	7	6	5	4	3	2	1	0
				8 DATA LSBs							

SYNTAX

|PM (Ia += Mb), PM (Ic += Md)| = <Data24>:24;



The :24 syntax at the end of the line is required for 24-bit data. If omitted, 24-bit data is truncated by the assembler.

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Indirect 24-bit Memory Write—immediate data” on page 7-57](#)

Type 23: Divide primitive, DIVQ

DIVQ divide primitive

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	0	1	1	1	1	0	1

11	10	9	8	7	6	5	4	3	2	1	0
0	XOP										

SYNTAX

DIVQ Xop;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Divide Primitives: DIVS and DIVQ” on page 3-35](#)

Instruction Opcodes

Type 24: Divide primitive, DIVS

DIVS divide primitive

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	0	0	0	0	1	1	1	0	0	YOP	

10	9	8	7	6	5	4	3	2	1	0
XOP										

SYNTAX

DIVS Yop, Xop;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Divide Primitives: DIVS and DIVQ” on page 3-35](#)

Type 25: Saturate

Saturate MR/SR on overflow

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	0	1	1	0	R		

11	10	9	8	7	6	5	4	3	2	1	0

SYNTAX

SAT MR;

SAT SR;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“MAC Saturate” on page 4-21](#)

Instruction Opcodes

Type 26:Push/Pop/Cache

Stack control

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	1	0	0				

11	10	9	8	7	6	5	4	3	2	1	0
				CF	PPP		LPP			SPP	

SYNTAX

PUSH |PC, LOOP, STS|;

POP |PC, LOOP, STS|;

FLUSH CACHE;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“PUSH or POP Stacks” on page 8-55](#)
- [“FLUSH CACHE” on page 8-61](#)

Type 29: Dreg «...» DM

Memory read/write with immediate modify (postmodify with update or premodify offset)

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	1	0	0	U	DRU		G	D

11	10	9	8	7	6	5	4	3	2	1	0
MOD DATA								I		DRL	

SYNTAX

```
Dreg = DM(Ireg += <Imm8>);      /* postmodify read */
DM(Ireg += <Imm8>) = Dreg;      /* postmodify write */
Dreg = DM(Ireg + <Imm8>);      /* premodify read */
DM(Ireg + <Imm8>) = Dreg;      /* premodify write */
```

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Indirect Memory Read/Write—immediate postmodify” on page 7-49](#)
- [“Indirect Memory Read/Write—immediate premodify” on page 7-52](#)

Instruction Opcodes

Type 30: NOP

No operation

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	0	0	0				

11	10	9	8	7	6	5	4	3	2	1	0
								SWCD			

SYNTAX

NOP;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“No Operation” on page 8-66](#)

Type 31: Idle

Idle

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	0	1	0				

11	10	9	8	7	6	5	4	3	2	1	0
								IDLE VALUE			

SYNTAX

IDLE;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Idle” on page 8-67](#)

Instruction Opcodes

Type 32: Any Reg «…» PM/DM

DAG memory read/write with premodify offset or postmodify update

OPCODE

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	0	1	0	1	0	1	MS	U	G	D	0

10	9	8	7	6	5	4	3	2	1	0
	RGP		REG				I	M		

SYNTAX

```
|DM(Ia += Mb), DM(Ic += Md)| = Reg; /*postmodify write,16-bit*/  
Reg = |DM(Ia += Mb), DM(Ic += Md)|; /*premodify read,16-bit*/  
|DM(Ia + Mb), DM(Ic + Md)| = Reg; /*premodify write,16-bit*/  
Reg = |DM(Ia + Mb), DM(Ic + Md)|; /*postmodify read,16-bit*/  
|PM(Ia += Mb), PM(Ic += Md)| = Reg; /*postmodify write,24-bit*/  
Reg = |PM(Ia += Mb), PM(Ic += Md)|; /*premodify read,24-bit*/  
|PM(Ia + Mb), PM(Ic + Md)| = Reg; /*premodify write,24-bit*/  
Reg = |PM(Ia + Mb), PM(Ic + Md)|; /*postmodify read,24-bit*/
```

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Indirect 16-bit Memory Read/Write—postmodify” on page 7-30](#)
- [“Indirect 16-bit Memory Read/Write—premodify” on page 7-34](#)
- [“Indirect 24-bit Memory Read/Write—postmodify” on page 7-37](#)
- [“Indirect 24-bit Memory Read/Write—premodify” on page 7-41](#)

Type 32a: DM«···DAG Reg | DAG Reg«···Ireg

DAG register store with register transfer

OPCODE

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	0	1	0	1	0	1	0	U	G	1	1

10	9	8	7	6	5	4	3	2	1	0
0	RGP		DAG REG				I		M	

SYNTAX

```
DM(Ireg1 += Mreg1) = |Ireg2, Mreg2, Lreg2|,  
|Ireg2, Mreg2, Lreg2| = Ireg1 ;
```

```
DM(Ireg1 += Mreg1) = |Ireg2, Mreg2, Lreg2|,  
|Ireg2, Mreg2, Lreg2| = Ireg1 ;
```

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Indirect DAG Register Write \(premodify or postmodify\), with DAG Register Move” on page 7-45](#)

Instruction Opcodes

Type 33: Reg3 «··· Data12

Load short constants

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12	11
0	0	0	1	0	0	0	0	12-bit data				

10	9	8	7	6	5	4	3	2	1	0
12-bit data							REG3			

SYNTAX

Reg3 = <Data12>;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Direct Register Load” on page 7-27](#)

Type 34: Dreg «···» IOreg

I/O register read/write

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	1	1	0	1	2MSBs addr		D

11	10	9	8	7	6	5	4	3	2	1	0
8-bit Address								DREG			

SYNTAX

```
IO(<Addr10>) = Dreg;
```

```
Dreg = IO (<Addr10>);
```

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“External IO Port Read/Write” on page 7-59](#)

Instruction Opcodes

Type 35: Dreg «···»Sreg

System control register read/write

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	1	1	0	0			D

11	10	9	8	7	6	5	4	3	2	1	0
8-bit Address								DREG			

SYNTAX

REG(<Addr8>) = Dreg;

Dreg = REG(<Addr8>);

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“System Control Register Read/Write” on page 7-61](#)

Type 36: Long Jump/Call

Conditional long jump/call (two-word instruction)

OPCODE BITS

- First word

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	1	0	1				S

11	10	9	8	7	6	5	4	3	2	1	0
8 Address MSBs								COND			

- Second word

23	22	21	20	19	18	17	16	15	14	13	12
				16 Address LSBs							

11	10	9	8	7	6	5	4	3	2	1	0
16 Address LSBs											

SYNTAX

[IF Cond] LJUMP <Addr24>;

[IF Cond] LCALL <Addr24>;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Long CALL” on page 8-37](#)
- [“Long JUMP” on page 8-40](#)

Instruction Opcodes

Type 37: Interrupt

Software interrupt

OPCODE BITS

23	22	21	20	19	18	17	16	15	14	13	12
0	0	0	0	0	1	1	1	0			

11	10	9	8	7	6	5	4	3	2	1	0
						C		BIT			

SYNTAX

SETINT <Imm4>;

CLRINT <Imm4>;

SEE ALSO

- [“Opcode Mnemonics” on page 9-1](#)
- [“Set Interrupt” on page 8-62](#)
- [“Clear Interrupt” on page 8-64](#)

Instruction Opcodes