

Project:

- To develop a micro RISC ISA processor for IoT and related applications.
- Development includes:
 - Assembler
 - RTL
 - Test bench
 - Test suites including self-checking functional tests
 - Stretch Goal: JTAG debugger
 - Micro architecture and user documents
- Demonstration in Analog Devices simulation environment
- This development is only for educational purposes should not be used for commercial purposes
- Work should not be shared without the permission of Analog Devices
- Analog Devices reserves the right to use this work

Page: 447 of:

https://www.analog.com/media/en/dsp-documentation/processor_manuals/50836807228561ADSP2106xSHARCPProcessorUsersManual_Revision2_1.pdf

Architecture:

Data registers/datapath = 16bit

Address = 16 bit

Number of

Types:

3a. IF condition compute; [//detailed list of compute given later](#)

IF condition $DM(la, Mb) = ureg$;

3c. IF condition compute;

IF condition $ureg = DM(la, Mb)$;

5. IF condition compute, $ureg1 = ureg2$;

7 (modified). IF condition MODIFY (la, Mb) ;

9. IF condition JUMP (Md, lc) ;

17 (modified for data length = 16) $ureg = <data16>$;

20 (modified). PUSH PCSTK = $<ureg>$;

$<ureg> = POP PCSTK$;

(Comment: PCSTK is Return address stack, this is modified to make complete operation as one instruction)

21. NOP;

22. IDLE (optional)

Conditions (A-5):

Condition Description

EQ ALU equal zero

LT ALU less than zero

LE ALU less than or equal zero

AC ALU carry AV ALU overflow

MV Multiplier overflow

MS Multiplier sign

SV Shifter overflow

SZ Shifter zero

NE ALU not equal to zero

GE ALU greater than or equal zero

GT ALU greater than zero

NOT AC Not ALU carry

NOT AV Not ALU overflow

NOT MV Not multiplier overflow

NOT MS Not multiplier sign

NOT SV Not shifter overflow

NOT SZ Not shifter zero

FOREVER Always True

Registers (page: A-6):

Note: All registers are 16bit

Data Register File

R15 - R0 Register file locations, fixed-point

Program Sequencer

PC Program counter (read-only)

PCSTK Top of PC stack

PCSTKP PC stack pointer

Data Address Generators

I7 - I0 DAG1 index registers

M7 - M0 DAG1 modify registers

L7 - L0 DAG1 length registers

B7 - B0 DAG1 base registers

Timer

TPERIOD Timer period

TCOUNT Timer counter (As soon as non-zero value is written, timer will start)

Map 1 Registers:

PC	program counter
PCSTK	top of PC stack
PCSTKP	PC stack pointer
FADDR	fetch address
DADDR	decode address
LADDR	loop termination address
CURLCNTR	current loop counter
LCNTR	loop counter
R15 - R0	register file locations
I15 - I0	DAG1 and DAG2 index registers
M15 - M0	DAG1 and DAG2 modify registers
L15 - L0	DAG1 and DAG2 length registers
B15 - B0	DAG1 and DAG2 base registers

System Registers:

MODE1	mode control 1
MODE2	mode control 2
IRPTL	interrupt latch
IMASK	interrupt mask
IMASKP	interrupt mask pointer
ASTAT	arithmetic status
STKY	sticky status
USTAT1	user status reg 1
USTAT2	user status reg 2

(b7=0)										System Registers			
b7 b6 b5 b4													
b3 b2 b1 b0	0000	0001	0010	0011	0100	0101	0110	0111					
0000	R0	I0	M0	L0	B0		FADDR	USTAT1					
0001	R1	I1	M1	L1	B1		DADDR	USTAT2					
0010	R2	I2	M2	L2	B2								
0011	R3	I3	M3	L3	B3		PC						
0100	R4	I4	M4	L4	B4		PCSTK						
0101	R5	I5	M5	L5	B5		PCSTKP						
0110	R6	I6	M6	L6	B6		LADDR						
0111	R7	I7	M7	L7	B7		CURLCNTR						
1000	R8	I8	M8	L8	B8		LCNTR						
1001	R9	I9	M9	L9	B9					IRPTL			
1010	R10	I10	M10	L10	B10					MODE2			
1011	R11	I11	M11	L11	B11					MODE1			
1100	R12	I12	M12	L12	B12					ASTAT			
1101	R13	I13	M13	L13	B13					IMASK			
1110	R14	I14	M14	L14	B14					STKY			
1111	R15	I15	M15	L15	B15					IMASKP			

Computes (Page: 504):

Fixed point computes involve 16-bit sign-extended integer operands in all cases

B.2.1 ALU Operations

The ALU operations are described in this section. Tables B.1 and B.2 summarize the syntax and opcodes for the fixed-point and floating-point ALU operations, respectively. The rest of this section contains detailed descriptions of each operation.

Fixed-Point Operations

$R_n = R_x + R_y$

$R_n = R_x - R_y$

$R_n = R_x + R_y + CI$

$R_n = R_x - R_y + CI - 1$

$COMP(R_x, R_y)$

$R_n = -R_x$

$R_n = ABS\ R_x$

$R_n = R_x AND\ R_y\ //bit\ by\ bit$

$R_n = R_x OR\ R_y\ //bit\ by\ bit$

$R_n = R_x XOR\ R_y\ //bit\ by\ bit$

$R_n = REG_OR\ R_x\ //\ OR\ all\ the\ bits\ of\ R_x - New$

$R_n = REG_AND\ R_x\ //\ AND\ all\ the\ bits\ of\ R_x - New$

$R_n = NOT\ R_x$

$R_n = MIN(R_x, R_y)$

$R_n = MAX(R_x, R_y)$

Floating-Point Operations

$F_n = F_x + F_y$

$F_n = F_x - F_y$

$F_n = \text{ABS}(F_x + F_y)$

$F_n = \text{ABS}(F_x - F_y)$

$F_n = (F_x + F_y)/2$

$\text{COMP}(F_x, F_y)$

$F_n = -F_x$

$F_n = \text{ABS } F_x$

$F_n = \text{PASS } F_x$

$F_n = \text{RND } F_x$

$F_n = \text{SCALB } F_x \text{ BY } R_y$

$R_n = \text{MANT } F_x$

$R_n = \text{LOGB } F_x$

$R_n = \text{FIX } F_x \text{ BY } R_y$

$R_n = \text{FIX } F_x$

$R_n = \text{TRUNC } F_x \text{ BY } R_y$

$R_n = \text{TRUNC } F_x$

$F_n = \text{FLOAT } R_x \text{ BY } R_y$

$F_n = \text{FLOAT } R_x$

$F_n = \text{RECIPS } F_x$

$F_n = \text{RSQRTS } F_x$

$F_n = F_x \text{ COPYSIGN } F_y$

$F_n = \text{MIN}(F_x, F_y)$

$F_n = \text{MAX}(F_x, F_y)$

$F_n = \text{CLIP } F_x \text{ BY } F_y$

B.2.2 Multiplier Operations

The multiplier operations are described in this section. Table B.3 summarizes the syntax and opcodes for the fixed-point and floating-point multiplier operations. The rest of this section contains detailed descriptions of each operation.

MRF is a 40 bit accumulate register made by joining (MR2[7:0]+MR1[15:0]+MR0[15:0]) registers

Fixed-Point Operations

$R_n = R_x * R_y$

$MRF = R_x * R_y$

$R_n = MRF + R_x * R_y$

$MRF = MRF + R_x * R_y$

$R_n = MRF - R_x * R_y$

$MRF = MRF - R_x * R_y$

$R_n = \text{SAT } MRF$

Floating-Point Operations

$F_n = F_x * F_y$

B.2.3 Shifter Operations (page: 556)

$R_n = \text{ASHIFT } R_x \text{ BY } R_y$ //Positive value of R_y will do left shift, negative will do right shift on R_x , page 560

$R_n = \text{ROT } R_x \text{ BY } R_y$ //Positive value of R_y will do left shift, negative will do right shift on R_x , page 561

$R_n = \text{LEFTZ } R_x$ //Function: Extracts the number of leading 0s from the fixed-point operand in R_x .

$R_n = \text{LEFTO } R_x$ //Function: Extracts the number of leading 1s from the fixed-point operand in R_x .