Appendix 4 Instruction Set Architecture

ISA Overview

The SD-8516 has core, extended, and CISC instructions.

- CORE operates like a RISC instruction set. It is designed to be completely sufficient while remaining small. This is what you should target first.
- EXTENDED are quality of life instructions. A great example is ADD register immediate. You do not need this instruction; you can load values into registers and add them. But we provide ADD register, immediate for quality of life. Technically, MUL is a QOL as well since you can loop with ADD.
- CISC instructions are special instructions, usually VAX-isms, designed to be quality of life for assembly language programmers.

Core instruction set

27 RISC-style instructions. A small core.

00	LD_IMM	LDA $5	Load register with immediate value	
02	LD_REG	LDA [X]	load register from memory location	
06	ST_REG	STA [X]	Store A in memory location using register as pointer	
09	MOV	MOV Y, A	Copy A up on Y	
0B	PUSH	PUSH A	Push A onto stack – ye scurvy dog!	
0C	POP	POP Y	Pull stack value an' hand it over to register	
0F	PUSHF	PUSHF	Push “pirate” flags/status	
10	POPF	POPF	Pop “pirate” flags back	
15	INC	INC X	Increment register by 1 (any size: byte/word/etc.)	
16	DEC	DEC Y	Decrement register by 1 (any size)	
1E	ADD	ADD X, Y	Add X = X + Y	
1F	SUB	SUB X, Y	Subtract X = X - Y	
32	AND	AND dst, src	Bitwise AND (compare two integers bit by bit)	
33	OR	OR dst, src	Bitwise OR (same)	
34	XOR	OR dst, src	Bitwise XOR	
35	NOT	NOT reg	Invert all bits in an integer word	
46	SHL	SHL A	Shift left	Z, N, C
47	SHR	SHR A	Shift right	Z, N, C
5A	CMP	CMP A, B	Compare (subtract, discard result)	Z, N, C, V
5B	CMP_IMM	CMP A, 0x0001	Compare (subtract, discard result)	Z, N, C, V
64	JMP		Unconditional jump	None
65	JZ		Jump if zero	None
84	CALL		Call subroutine (push IP, jump)	
85	RET		Return from subroutine (pop IP)	
B6	SETF	SETF 0x80	Set bits in flags	*

B7	CLRF	CLRF 0x80	Clear bits in flags	*
B8	TESTF	TESTF 0x80	Non-destructive AND	Z C

Core-2 instructions

10 instructions. Some instructions are not strictly RISC but still considered core. For example, LD_MEM occurs too many times for it to be considered an extended opcode. MUL is another one included here; MUL is technically an extended opcode (can loop over ADD) but it is a fundamental operation, so it is included here.

01	LD_MEM	LDA [$5]	Load register from memory location	
05	ST_MEM	STA [$10]	Store register in memory location	
20	MUL	MUL X, Y	Multiply X = X * Y	
21	DIV	DIV X, Y	Divide X = X / Y	
22	MOD	MOD X, Y	Modulo X = X % Y	
66	JNZ		Jump if not zero	None
67	JC		Jump if carry set	None
68	JNC		Jump if carry clear	None
86	INT		Software interrupt	
87	RTI		Return from Interrupt	

Extended instruction set

53 instructions. Some of these are more extended than others. For example, load from a memory location is, in fact

03	LD_REG24	LDA [X:Y]	Load register from memory location using [low_byte:word]	
04	LD_IMM24	LDA [$1:$C000]	Load byte from memory location [bank:addr]	
07	ST_REG24	STA [X:Y]	Store A in memory using [low_byte:word] registers.	
08	ST_IMM24	STA [$0:$A0]	Store register in memory location [bank:addr]	
0A	XCHG	XCHG X, Y	Swap dem two – X an Y trade place, quick quick	
0D	PUSHA	PUSHA	Save all registers in ye treasure chest	
0E	POPA	POPA	Get the registers back	
23	ADD_REG_IMM	ADD X, $1234	Add immediate word value; X = X + immediate	
24	SUB_REG_IMM	SUB X, $ABCD	Subtract immediate; X = X - immediate	
25	MUL_REG_IMM	MUL X, $100	Multiply by immediate; X:Y = X * immediate	
26	DIV_REG_IMM	DIV X, $10	Divide by immediate; X = quotient, Y = remainder	
27	MOD_REG_IMM	MOD X, $FF	Modulo by immediate; X = X % immediate	
28	ADDC	ADDC X, Y	Add with carry; X = X + Y + carry flag	
29	SUBC	SUBC X, Y	Subtract with borrow; X = X - Y - borrow	
30	ADDC_REG_IMM	ADDC X, $5	Add imm w/carry; X = X + imm + carry	
31	SUBC_REG_IMM	SUBC X, $1	Subtract w/carry X = X - imm - borrow	
36	TEST	TEST dst, src	Non-destructive AND	
37	AND_IMM	AND dst, imm	Bitwise AND with immediate	
38	OR_IMM	OR dst, imm	Bitwise OR with immediate	

48	SHLC	SHLC A	Shift left	Z, N, C
49	SHRC	SHRC A	Shift right	Z, N, C
4A	ROL	ROL A	Rotate left	Z, N, C
4B	ROR	ROR A	Rotate right	Z, N, C
4C	ROLC	ROLC A	Rotate left	Z, N, C
4D	RORC	RORC A	Rotate right	Z, N, C

A0	SEZ	SEZ	Set zero flag	Z
A1	SEN	SEN	Set negative flag	N
A2	SEC	SEC	Set carry flag	C
A3	SEV		Set overflow flag	V
A4	SEE		Set Extra Flag (User flag)	E
A5	SEF		Set Free Flag (User flag)	F
A6	SEB		Set Bonus Flag (User flag)	B
A7	SEU		Set User Flag (User flag)	U
A8	SED		Set Debug Flag	D
A9	SEI		Set Enable Interrupt	I
AA	SSI		Enable Sound System Interrupts	S
AB	CLZ		Clear zero flag	Z
AC	CLN		Clear negative flag	N
AD	CLC		Clear carry flag	C
AE	CLV		Clear overflow flag	V
AF	CLE	CLE	Clear E	E
B0	CLF	CLF	Clear F Flag (user flag)	F
B1	CLB			B
B2	CLU			U
B3	CLD			D
B4	CLI	CLI	Clear Interrupt Flag	I
B5	CSI	CSI	Clear Sound Interrupt	S

FB	BASIC	Reserved	
FC	YIELD	Yield thread priority	Y
FD	BREAK	Reserved	
FE	NOP	No operation	
FF	HALT	Halt CPU (sets HALT flag)	H

CISC instruction set

7 instructions. CISC style, usually inspired by VAX, 680×0, or other CISC-leaning processors.

8C	MEMCOPY	MEMCOPY ELM, FLD, I	Copy memory from ptr to ptr.	
8D	SCAN	SCAN H, N	Scan ptr H for needle N	
8E	CMPC3	CMPC3 ELM, FLD, C	Compare Characters	Z C

8F	SKPC	SKPC ELM, AL	Skip characters	
90	SKPC_IMM	SKPC ELM, $20	Skip characters (immediate)	
98	PAB	PAB	Pack low 4 bytes of A and low 4 bytes of B into AL	
99	UAB	UAB	Unpack AL into low 4 bytes of AL and low 4 bytes of BL	

`#200 CASE base, selector, limit`
switch-case. Index an address from the table at `base` and jump to it. If selector >= limit it will fall-through (not jump). Requires AL is in the range 0-limit. Table format: [addr][addr][addr]...

`#201 CASEB base, selector, limit`
switch-case-on-byte. Index an address from the table at `base` and jump to it. Checks each selector byte/word/etc. Checks a maximum of `limit` number of records. Will fall-through if not found. Table format: [selector][addr][selector][addr][selector][addr]...

Table of Instructions

Load/Store Instructions

Byte	Code	Example	Description
00	LD_IMM	LDA $5	Load register with immediate value
01	LD_MEM	LDA [$5]	Load register from memory location
02	LD_REG	LDA [X]	load register from memory location
03	LD_REG24	LDA [X:Y]	Load register from memory location using [low_byte:word]
04	LD_IMM24	LDA [$1:$C000]	Load byte from memory location [bank:addr]
05	ST_MEM	STA [$10]	Store register in memory location
06	ST_REG	STA [X]	Store A in memory location using register as pointer
07	ST_REG24	STA [X:Y]	Store A in memory using [low_byte:word] registers.
08	ST_IMM24	STA [$0:$A0]	Store register in memory location [bank:addr]

Data Movement Instructions of the Caribbean

Byte	Code	Example	Description
09	MOV	MOV Y, A	Copy A up on Y
0A	XCHG	XCHG X, Y	Swap dem two – X an Y trade place, quick quick

Pirate Stack Operations

Byte	Code	Example	Description
0B	PUSH	PUSH A	Push A onto stack – ye scurvy dog!
0C	POP	POP Y	Pull stack value an' hand it over to register
0D	PUSHA	PUSHA	Save all registers in ye treasure chest
0E	POPA	POPA	Get the registers back
0F	PUSHF	PUSHF	Push “pirate” flags/status
10	POPF	POPF	Pop “pirate” flags back

Boring, Normal increment operations

Byte	Code	Example
15	INC	INC X
16	DEC	DEC Y

Arithmetic Operations

1E	ADD	ADD X, Y	Add X = X + Y
1F	SUB	SUB X, Y	Subtract X = X - Y
20	MUL	MUL X, Y	Multiply X:Y = X * Y *(result may be 32-bit wide)*
21	DIV	DIV X, Y	
22	MOD	MOD X, Y	
23	ADD_REG_IMM	ADD X, $1234	
24	SUB_REG_IMM	SUB X, $ABCD	
25	MUL_REG_IMM	MUL X, $100	
26	DIV_REG_IMM	DIV X, $10	
27	MOD_REG_IMM	MOD X, $FF	
28	ADDC	ADDC X, Y	
29	SUBC	SUBC X, Y	
30	ADDC_REG_IMM	ADDC X, $5	
31	SUBC_REG_IMM	SUBC X, $1	

Logic Ops

Byte	Instruction	Example	Description	Flags Affected
32	AND	AND dst, src	Bitwise AND (compare two integers bit by bit)	
33	OR	OR dst, src	Bitwise OR (same)	
34	XOR	OR dst, src	Bitwise XOR	
35	NOT	NOT reg	Invert all bits in an integer word	
36	TEST	TEST dst, src	Non-destructive AND	
37	AND_IMM	AND dst, imm	Bitwise AND with immediate	
38	OR_IMM	OR dst, imm	Bitwise OR with immediate	

Shift/Rotate Operations

Byte	Instruction	Example	Description	Flags Affected
46	SHL	SHL A	Shift left	Z, N, C
47	SHR	SHR A	Shift right	Z, N, C
48	SHLC	SHLC A	Shift left	Z, N, C
49	SHRC	SHRC A	Shift right	Z, N, C
4A	ROL	ROL A	Rotate left	Z, N, C
4B	ROR	ROR A	Rotate right	Z, N, C
4C	ROLC	ROLC A	Rotate left	Z, N, C
4D	RORC	RORC A	Rotate right	Z, N, C

Comparison & Branching

Byte	Instruction	Example	Description	Flags Affected
5A	CMP	CMP A, B	Compare (subtract, discard result)	Z, N, C, V
5B	CMP_IMM	CMP A, 0x0001	Compare (subtract, discard result)	Z, N, C, V
JMP	Unconditional jump	None		
JZ	Jump if zero	None		
JNZ	Jump if not zero	None		
JC	Jump if carry set	None		
JNC	Jump if carry clear	None		

Subroutine Operations

Byte	Instruction	Example	Description	Flags Affected
	CALL		Call subroutine (push IP, jump)	
	RET		Return from subroutine (pop IP)	
	INT		Software interrupt	
	RTI		Return from Interrupt	

MMU/Block Operations

8C	MEMCOPY	MEMCOPY ELM, FLD, I	Copy memory from ptr to ptr.	
8D	SCAN	SCAN H, N	Scan ptr H for needle N	
8E	CMPC3	CMPC3 ELM, FLD, C	Compare Characters	Z C
8F	SKPC	SKPC ELM, AL	Skip characters	
90	SKPC_IMM	SKPC ELM, $20	Skip characters (immediate)	

These instructions are mainly inspired by instructions on the VAX architecture.

- MEMCOPY – The VAX has MOVC3 and MOVC5 instructions.
- SCAN – The VAX has SCANC and LOCC instructions.
- CMPC3 – *Compare Characters* is a VAX instruction; CMPC3 and CMPC5.
- SKPC – *Skip Character* is a VAX instruction.

The VAX was indeed ahead of it's time. This is not surprising as it was the most successful Super-Minicomputer of it's time. In the later microcomputer era, the 680×0 brought over a lot of functionality, but remained more RISC like, requiring tight loops to replicate instructions like the above.

Bit Packing

98	PAB	PAB	Pack low 4 bytes of A and low 4 bytes of B into AL
99	UAB	UAB	Unpack AL into low 4 bytes of AL and low 4 bytes of BL

Flag Operations

Byte	Instruction	Example	Description	Flags Affected
A0	SEZ	SEZ	Set zero flag	Z

Byte	Instruction	Example	Description	Flags Affected
A1	SEN	SEN	Set negative flag	N
A2	SEC	SEC	Set carry flag	C
A3	SEV		Set overflow flag	V
A4	SEE		Set Extra Flag (User flag)	E
A5	SEF		Set Free Flag (User flag)	F
A6	SEB		Set Bonus Flag (User flag)	B
A7	SEU		Set User Flag (User flag)	U
A8	SED		Set Debug Flag	D
A9	SEI		Set Enable Interrupt	I
AA	SSI		Enable Sound System Interrupts	S
AB	CLZ		Clear zero flag	Z
AC	CLN		Clear negative flag	N
AD	CLC		Clear carry flag	C
AE	CLV		Clear overflow flag	V
AF	CLE	CLE	Clear E	E
B0	CLF	CLF	Clear F Flag (user flag)	F
B1	CLB			B
B2	CLU			U
B3	CLD			D
B4	CLI	CLI	Clear Interrupt Flag	I
B5	CSI	CSI	Clear Sound Interrupt	S
B6	SETF	SETF 0x80	Set bits in flags	*
B7	CLRF	CLRF 0x80	Clear bits in flags	*
B8	TESTF	TESTF 0x80	Non-destructive AND	Z C

System Operations

Instruction	Description
YIELD	Poll UI, System Clock, Sound Chip, Video Chip, and others
NOP	No operation
HALT	Halt CPU (set H flag)