CRiSAM (CISC RISC y Stack Accumulator Machine) James C. Brakefield (2020)

Op	description	Forth/uCODE	Stack/Accum		RISC		CISC-BIS	
		1 byte	2 bytes	2 byte w/imm	3 byte	3 byte w/imm	sz1: data, sz2: delta	isz: imm size
	5-bit register fields	00xxxxxx	01xxxxxx 0prssssss	01xxxxxx 1pnnnISZ 01xxxxxx cccccISZ	10xxxxxx 0dsrrrrr ddddssss (most, not all)	10xxxxxx 1dsnnISZ ddddssss (most, not all)	11xxxxxx 0szrrrrr 0szbbbbb 11xxxxxx 0szrrrrr 1szbbbbb SHFiiiii 11xxxxxx 1sznnISZ 0szbbbbb 11xxxxxx 1sznnISZ 1szbbbbbSHFiiiii	
	4-bit register fields	1 byte	2 bytes	2 byte w/imm	3 byte	3 byte w/imm	sz1: data, sz2: delta	isz: imm size
		00xxxxxx	01xxxxxx 0prunnnn	01xxxxxx 1nnnnnnn	10xxxxxxdddd ssssrrrr0000	10xxxxxxdddd sssssnnnnnn1 10xxxxxxdddd sssssnnnISZ0	11xxxxxxrrrr bbbbiiiiszsz	11xxxxxxxisz bbbbiiiiszsz

x: op bits, p: pop/push, r: replace op, n: imm, d: dest reg, r&s: src reg, b: base reg, i: index reg, sz: mem/off size fld, ISZ imm size, SHF index shift amount

x: op bits, p: pop/push, r: replace op, u: update CCR, n: imm, d: dest reg, r&s: src reg, b: base reg, i: index reg, sz: size fld

Four data sizes possible (eg 8, 16, 32 & 64 or 8, 16, 24 & 48 or 8, 16, 24 & 32 or ...), registers are atleast the size of the largest supported data or address
Initially CALL shall push PC, CCR & Frame pointer as a single word, RTN may or may not restore CCR (defualt is non-restore), RTN does not restore FP, RTNF does restore FP
Register file configurations: straight reg file (30+zero+PC), single stack(4+1) with frame(12) + regs(12+zero+PC), dual stack(4+1 & 4+1) with frame(12) + regs(8+zero+PC)

Forth	Uses register file as data and return stacks, spill/restore TBD, can be implemented as a compressed encoding of Stack/RISC/CISC ops with register file in stack mode	`00xxxxxx`
uCode	Use lookup table to abbreviate 16 & 24-bit instructions, lookup table supplies prefix bits as well	`00xxxxxx`
	Defining of uCodes not a user level activity, likewise switching between stack mode and register mode	
Stack	Uses register file as data and return stacks and frame area, indexing into these areas, p:push/pop, r:replace op, u:CCR update, imm encodes its size, spill/restore TBD	`01xxxxxx 0prssssss`
RISC	Three registers specified, use per RISC/Stack mode, imm encoded by ISZ, both ID2D & IS2D ops	`10xxxxxx 0sdrrrrr ssssdddd`
CISC	x86 addressing, use per RISC/Stack mode, delta sz: 0, 8, 24 & 64, imm size: 8,16,24… 32 registers in reg file, PC & CCR separate	`11xxxxxx 0szrrrrr 0szbbbbb & 11xxxxxx 0szrrrrr`
Accum	When not in stack mode reg #15 is accum, #14 is return adr 16 index locations including pop/push data/return stack, frame indexing and zero and ?	

PFX0	prefix "000" to op-code				PFX0/NOP	PFX0	RADD	RADDI	8-bit PreFix op-codes 0b10000xxx
PFX1	prefix "001" to op-code				PFX1	PFX1	RSUB	RSUBI	provides op-code expansion capability
PFX2	prefix "010" to op-code				PFX2	PFX2	RADC	RADCI	allocated to ISA with least crowding
PFX3	prefix "011" to op-code				PFX3	PFX3	RSBC	RSBCI	
PFX4	prefix "100" to op-code				PFX4	PFX4	RAND	RANDI	
PFX5	prefix "101" to op-code				PFX5	PFX5	ROR	RORI	
PFX6	prefix "110" to op-code				PFX6	PFX6	RXOR	RXORI	
PFX7	prefix "111" to op-code				PFX7	PFX7	RMUL	RMULI	
AUG0	augment n or dsr or rbi				AUG0	AUG0	RMULU	RMULUI	8-bit AUGment op-codes 0b10100nnn
AUG1	augment n or dsr or rbi				AUG1	AUG1	RMULUS	RMULSUI	provides register field expansion capability
AUG2	augment n or dsr or rbi				AUG2	AUG2	RDIV	RDIVI	allocated to ISA with least crowding
AUG3	augment n or dsr or rbi				AUG3	AUG3	RDIVI	RDIVSI	
AUG4	augment n or dsr or rbi				AUG4	AUG4	RLSL	RLSLI	
AUG5	augment n or dsr or rbi				AUG5	AUG5	RROTL	RROTRI	
AUG6	augment n or dsr or rbi				AUG6	AUG6	RLSR	RLSRI	
AUG7	augment n or dsr or rbi				AUG7	AUG7	RASR	RASRI	
LDB	Load byte zero extended	LDB (C@)	LDB/STB (~C@/C!)	LDBI	LDB	LDBN	RFADD	RFADDI	
LDBS	Load byte sign extended	LDBS	LDBS	LDBSI	LDBS	LDBSN	RFSUB	RFSUBI	
LDH	Load half zero extended	LDH	LDH/STH	LDHI	LDH	LDHN	RFMUL	RFMULI	
LDHS	Load half sign extended	LDHS	LDHS	LDHSI	LDHS	LDHSN	RFDIV	RFDIVI	
LDW	Load word zero extended	LDW	LDW/STW	LDWI	LDW	LDWN	D2D ops		D2D ops: TST, CLR, SET, INC, DEC, JMP, CALL
LDWS	Load word sign extended	LDWS	LDWS	LDWSI	LDWS	LDWSN	ST	STI	
LDL	Load long	LDL (@)	LDL/STL (~@/!)	LDLI	LDL	LDLN			
STB	Store byte	STB (C!)			STB	STBN	LD		
STH	Store half	STH			STH	STHN	LDS		
STW	Store word	STW			STW	STWN			
STL	Store long	STL (!)			STL	STLN			
ADD	Add	ADD (+)	ADD/RADD	ADDI	ADD	ADDI	ADD		
SUB	Subtract	SUB (-)	SUB/RSUB	SUBI	SUB	SUBI	SUB		
ADC	Add with carry	ADC	ADC/RADC	ADCI	ADC	ADCI	ADC		
SBC	Subtract with carry	SBC	SBC/RSBC	SBCI	SBC	SBCI	SBC		
AND	And	AND	AND/RAND	ANDI	AND	ANDI	AND		
OR	Or	OR	OR/ROR	ORI	OR	ORI	OR		
XOR	Exclusive or	XOR	XOR/RXOR	XORI	XOR	XORI	XOR		
MUL	Multiply	MUL	MUL/RMUL	MULI	MUL	MULI	MUL		
MULU	Multiply upper unsigned	MULU	MULU	MULUI	MULU	MULUI	MULU		
MULUS	Multiply upper signed	MULUS	MULUS	MULUSI	MULUS	MULUSI	MULUS		
DIV	Divide unsigned	DIV	DIV	DIVI	DIV	DIVI	DIV		Divide to be implemented by co-processor, started by DIV inst. and
DIVI	Divide signed	DIVI	IDIV	IDIVI	DIVI	IDIVI	DIVI		
LSL	Shift left	LSL	LSL/RLSL	LSLI	LSL	LSLI	LSL		

ROTL	Rotate left	ROTL	ROTL/RROTL	ROTLI	ROTL	ROTLI	ROTL	
LSR	Unsigned shift right	LSR	LSR/RLSR	LSRI	LSR	LSRI	LSR	
ASR	Signed shift right	ASR	ASR/RASR	ASRI	ASR	ASRI	ASR	
FADD	Floating add	FADD	FADD/RFADD	FADDI	FADD	FADDI	FADD	
FSUB	Floating subtract	FSUB	FSUB/RFSUB	FSUBI	FSUB	FSUBI	FSUB	
FMUL	Floating multiply	FMUL	FMUL/RFMUL	FMULI	FMUL	FMULI	FMUL	
FDIV	Floating divide	FDIV	FDIV/RFDIV	FDIVI	FDIV	FDIVI	FDIV	
CMP	Integer compare	CMP	CMP	CMPI			CMP	CMPI
TST	Compare to zero	TST	TST					
BIT	Logical compare (AND)	BIT	BIT	BITI			BIT	BITI
BRcc	Branch relative on condition		BRcc	BRcc				
CALL	Call subroutine	CALL	CALL	CALLR	CALL	CALLN	CALL	
CALLF	Call subroutine with frame		CALLF	CALLRF	CALLF	CALLFN		
RTN	Return	RTN						
RTNF	Return with frame			RTNF				
BR	exit IF code segment		BR	BR				
BREQ	skip IFN code segment		BREQ	BREQ				
BRNE	skip IF code segment		BRNE	BRNE				
LIT	Length encoded immediate		LIT	LIT	(32) S2D ops	LDI	LEA	
LITN	Length encoded immediate		LITN	LITN				
LITR	PC + len encoded signed imm		LITR	LITR				
DROP		DROP	LDB++/STB++	(--)LDB/STB	EXTRACT	EXTRACTI		
SWAP		SWAP	LDBS++	(--)LDBS	INSERT	INSERTI		
PICK0S (DUP)		PICK0S (DUP)	LDH++/STH++	(--)LDH/STH				
PICK1S (OVER)		PICK1S (OVER)	LDHS++	(--)LDHS				
PICK2S		PICK2S	LDW++/STW++	(--)LDW/STW				
PICK3S		PICK3S	LDWS++	(--)LDWS				
PICKR		PICKR	LDL++/STL++	(--)LDL/STL				
PULLR		PULLR	LDB/STB+offset	LDB/STB+offset++				
PUSH2R		PUSH2R	LDBS+offset	LDBS+offset++				
ZERO		ZERO	LDH/STH+offset	LDH/STH+offset++				
ONE		ONE	LDHS+offset	LDHS+offset++				
TWO		TWO	LDW/STW+offset	LDW/STW+offset++				
			LDL/STL+offset	LDL/STL+offset++				
Plus-store (+!)		Plus-store (+!)						
PC2RS	setup loop start address	PC2RS						
DO-LOOP	unconditional loop repeat	DO-LOOP						
DO-LOOPN	conditional loop repeat	DO-LOOPNE						
DO-LOOP+	count up loop with limit	DO-LOOP+						
DO-LOOP+	count up loop with limit	DO-LOOP+1						
DO-LOOP-	count down loop	DO-LOOP-						
PICK	Push Indexed item to TOS		PICK					
POKE	TOS to indexed location, pop TOS		POKE					
POPN	Drop N elements from DS/RS		POPN					
IN	Input			IN				
OUT	Output			OUT				
DEFUC	Define uCode			DEFUCI				
86	Total not empty	**55**	**53**	53	**52**	52	**48**	23

?unspecified details
n specifies initial size of frame

n specifies # of return values
branch relative with length encoded displacement following op-cod
relative branch conditional on TOS, TOS popped
relative branch conditional on TOS, TOS popped
S2D ops:
POPCT
LDG0S
LDG1S
TRL0S
TRL1S
NEG
NOT
ABS
SWAP
INT2FLT
FLT2INT
MANT
FEXP
FABS
FNEG

push address of next instruction to return stack
branch to adr at TORS
branch to adr at TOR conditional on TOS NE zero, pop TORS if zero,
increment SORS by TOS & pop TOS, branch to adr at TORS if sum <
increment SORS by one, branch to adr at TORS if sum < 3ORS, if >=
decrement SORS, branch to adr at TORS if >0, else double pop RS
from DS if >=0, from RS otherwise
to DS if >=0, to RS otherwise
drop N elements for DS if >0, from RS otherwise

Some other Forth ops

NIP	delete SOS
ABORT	similar to RESET
QUIT	similar to TRAP
/MOD	divide with both quotient and remainder
*/MOD	multiply followed by divide of full product
NEG	
ABS	
MAX	
MIN	
SKIP	skip to next 32-bit word
LIT8	8-bit literal push

Immediate length encodings for 4-bit register fields (currently doing ISZ in 2nd inst. byte with two additional bits preceeding)

CISC/BIS offset	`rrrrbbbb iiiisz--`	0: unsigned byte, 1: signed half, 2: none, 3: long
CISC/BIS data size	`rrrrbbbb iiii--sz`	0: byte, 1: half, 2: word, 3: long
CISC/BIS immediate size	`xISZbbbb iiii----`	ISZ: # of bytes, no unused bit
RISC Bcc	`cISZcccc snnnnnnn. . .`	ISZ specifies length of displacement
RISC 7-bit imm	`ddddssss snnnnnn1`	seven bit signed 2's complement
RISC ISZ imm	`ddddssss snnnISZ0 nnnnnnnn.`	ISZ >0: 12, 20, 28, 36, 44, 52 & 64 bit immediate
Stack/Accum immediate	`sccccccc`	Signed imm, comma coded length, see LIT
Ibid	`s1nnnnnn`	Signed 7-bit
Ibid	`s0ccccnn nnnnnnnn . . .`	Signed 14, 21, 28, 35
Ibid	`s00000sz nnnnnnnn . . .`	Signed 40, 48, 56, 64
Stack Alt signed imm, full 8-bits	`1nnnsnnn`	7-bit