Instruction Set of the C16 CPU

Abbreviations

UL	low byte of 16 bit unsigned operand
UH	high byte of 16 bit unsigned operand
U8	8 bit unsigned operand, will be zero extended
U4	4 bit unsigned operand contained in opcode, will be zero extended
SL	low byte of 16 bit signed operand
SH	high byte of 16 bit signed operand
S8	8 bitnsigned operand, will be sign extended
S4	4 bit signed operand contained in opcode, will be sign extended
u	an unsigned immediate operand (zero extended if < 16 bit)
s	a signed immediate operand (sign extended if < 16 bit)
uRR	RR unsigned (in comparisons)
sRR	RR signed (in comparisons)
RU	RR, but the 8 bit operand is zero extended. Implies 8 buit transfer.
RS	RR, but the 8 bit operand is sign extended. Implies 8 buit transfer.
R	low byte of RR. Implies 8 buit transfer.
uLL	LL unsigned (in comparisons)
sLL	LL signed (in comparisons)
LU	LL, but the 8 bit operand is zero extended. Implies 8 buit transfer.
LS	LL, but the 8 bit operand is sign extended. Implies 8 buit transfer.
L	low byte of LL. Implies 8 buit transfer.
iff	if and only if

Some opcodes exist in a quick, short, or long format. The assembler generates the proper opcode based on the actual operand value.

Opcodes of the S16 CPU

Opcode Byte			**Assembler Syntax**	**Operation**
1	2	3		
00			HALT	HALT until next interrupt (if enabled)
01			NOP	Do nothing
02			JMP u	PC := u

Opcode Byte			Assembler Syntax	Operation
1	2	3		
03	UL	UH	JMP RRNZ, u	PC = u iff (RR != 0)
04	UL	UH	JMP RRZ, u	PC = u iff (RR == 0)
05	UL	UH	CALL u	-(SP) = PC + 3; PC := u
06			CALL (RR)	-(SP) = PC + 3; PC := RR
07			RET	PC := (SP)+
08			MOVE (SP)+, RR	RR := (SP)+
09			MOVE (SP)+, RS	RR := (SP)+ (sign extended)
0A			MOVE (SP)+, RU	RR := (SP)+ (zero extended)
0B			MOVE (SP)+, LL	LL := (SP)+
0C			MOVE (SP)+, LS	LL := (SP)+ (sign extended)
0D			MOVE (SP)+, LU	LL := (SP)+ (zero extended)
0E			MOVE RR, -(SP)	-(SP) := RR
0F			MOVE R, -(SP)	-(SP) := R
10	UL	UH	AND RR, #u	RR := RR & u
11	U8			
12	UL	UH	OR RR, #u	RR := RR | u
13	U8			
14	UL	UH	XOR RR, #u	RR := RR ^ u
15	U8			
16	SL	SH	SEQ RR, #s	RR := -1 iff (RR == s) else 0
17	S8			
18	SL	SH	SNE RR, #s	RR := -1 iff (RR != s) else 0
19	S8			
1A	SL	SH	SGE RR, #s	RR := -1 iff (sRR >= s) else 0
1B	S8			
1C	SL	SH	SGT RR, #s	RR := -1 iff (sRR > s) else 0
1D	S8			

Opcode Byte			Assembler Syntax	Operation
1	2	3		
1E	SL	SH	SLE RR, #s	RR := -1 iff (sRR <= s) else 0
1F	S8			
20	SL	SH	SLT RR, #s	RR := -1 iff (sRR < s) else 0
21	S8			
22	UL	UH	SHS RR, #u	RR := -1 iff (uRR >= u) else 0
23	U8			
24	UL	UH	SHI RR, #u	RR := -1 iff (uRR > u) else 0
25	U8			
26	UL	UH	SLS RR, #u	RR := -1 iff (uRR <= u) else 0
27	U8			
28	UL	UH	SLO RR, #u	RR := -1 iff (uRR < u) else 0
29	U8			
2A	UL	UH	ADD SP, #u	SP := SP + u
2B	U8			
2C			CLRW -(SP)	-(SP) := 0
2D			CLRB -(SP)	-(SP) := 0
2E	U8		IN (u), RU	RR := input data, zero extended
2F	U8		OUT R, (u)	output data := R
30			AND LL, RR	RR := RR and LL
31			OR LL, RR	RR := RR or LL
32			XOR LL, RR	RR ::= RR xor LL
33			SEQ LL, RR	RR ::= -1 iff LL == RR) else 0
34			SNE LL, RR	RR ::= -1 iff LL != RR) else 0
35			SGE LL, RR	RR := -1 iff sLL >= sRR) else 0
36			SGT LL, RR	RR := -1 iff sLL > sRR) else 0
37			SLE LL, RR	RR := -1 iff sLL <= sRR) else 0
38			SLT LL, RR	RR := -1 iff sLL < sRR) else 0

Opcode Byte			Assembler Syntax	Operation
1	2	3		
39			SHS LL, RR	RR := -1 iff uLL >= uRR) else 0
3A			SHI LL, RR	RR := -1 iff uLL > uRR) else 0
3B			SLS LL, RR	RR := -1 iff uLL <= uRR) else 0
3C			SLO LL, RR	RR := -1 iff uLL < uRR) else 0
3D			LNOT RR	RR := -1 iff (RR == 0) else 0
3E			NEG RR	RR := - RR
3F			NOT RR	RR := not RR
40			MOVE LL, RR	RR := LL
41			MOVE LL, (RR)	(RR) := LL
42			MOVE L, (RR)	(RR) := LL
43			MOVE RR, LL	LL := RR
44			MOVE RR, (LL)	(LL) := RR
45			MOVE R, (LL)	(LL) := R
46			MOVE (RR), RR	(RR) := RR
47			MOVE (RR), RS	(RR) := R sign extended
48			MOVE (RR), RU	(RR) := R zero extended
49	UL	UH	MOVE (u), RR	RR := (u)
4A	UL	UH	MOVE (u), RS	RR := (u) sign extended
4B	UL	UH	MOVE (u), RU	RR := (u) zero extended
4C	UL	UH	MOVE (u), LL	LL := (u)
4D	UL	UH	MOVE (u), LS	LL := (u) sign extended
4E	UL	UH	MOVE (u), LU	LL := (u) zero extended
4F			MOVE RR, SP	SP := RR
50 to 51			Undefined.(NOP)	
52	U8		LSL RR, #u	RR := RR >> u
53	U8		ASR RR, #u	RR := sRR << u
54	U8		LSR RR, #u	RR := uRR << u

Opcode Byte			Assembler Syntax	Operation
1	2	3		
55			LSL LL, RR	RR := RR >> LL
56			ASR LL, RR	RR := sRR << LL
57			LSR LL, RR	RR := uRR << LL
58			ADD LL, RR	RR := LL + RR
59			SUB LL, RR	RR := LL - RR
5A	UL	UH	MOVE RR, (u)	(u) := RR
5B	UL	UH	MOVE R, (u)	(u) := R
5C	UL	UH	MOVE RR, u(SP)	(SP + u) := RR
5D	U8		MOVE RR, u(SP)	
5E	UL	UH	MOVE R, u(SP)	(SP + u) := R
5F	U8			
60	UL	UH	MOVE u(SP), RR	RR := (SP + u)
61	U8			
62	UL	UH	MOVE u(SP), RS	RR := (SP + u) sign extended
63	U8			
64	UL	UH	MOVE u(SP), RU	RR := (SP + u) zero extended
65	U8			
66	UL	UH	MOVE u(SP), LL	LL := (SP + u)
67	U8			
68	UL	UH	MOVE u(SP), LS	LL := (SP + u) sign extended
69	U8			
6A	UL	UH	MOVE u(SP), LU	LL := (SP + u) zero extended
6B	U8			
6C	UL	UH	LEA u(SP), RR	RR := SP + u
6D	U8			
6E			MOVE -(RR), -(LL)	-(RR) := -(LL) (byte)
6F			MOVE (RR)+, (LL)+	(RR)+ := (LL)+ (byte)

Opcode Byte			Assembler Syntax	Operation
1	2	3		
70			MUL_IS	Initialize signed/unsigned multiplication/ division (modulo == division) Preparation for: LL * RR, LL / RR or LL % RR
71			MUL_IU	
72			DIV_IS	
73			DIV_IU	
74			MD_STP	Do this 16 times after initialization
75			MD_FIN	RR := multiplication/division result
76			MOD_FIN	RR := modulo result
77			EI	Increment interrupt enable counter
78			RETI	Atomar RET followed by EI
79			DI	Decrement interrupt enable counter
7A to 9F			Undefined.(NOP)	
A U4			ADD RR, #u	RR := RR + u
B U4			SUB RR, #u	RR := RR - u
C S4			MOVE #s, RR	RR := s
D S4			SEQ LL, #s	RR := -1 iff (LL == s) else 0 (for C case:)
E S4			MOVE #s, LL	LL := s
F0 to F3			Undefined.(NOP)	
F4	UL	UH	ADD RR, #u	RR := RR + u
F5	U8			
F6	UL	UH	SUB RR, #u	RR := RR - u
F7	U8			
F8	SL	SH	MOVE #s, RR	RR := s
F9	S8			
FA	SL	SH	SEQ LL, #s	RR := -1 iff (LL == s) else 0 (for C case:)
FB	S8			
FC	SL	SH	MOVE #s, LL	LL := s
FD	S8			

Opcode Byte			Assembler Syntax	Operation
1	2	3		
FE to FF			Undefined.	