CCSE ISA V2

Nicholas Miehlbradt

October 2022

1 Overview

1.1 Memory

- Minimum addressable unit is 16-bit words
- Memory addresses are 16 bits long
- Total addressable memory is 128 KB (64K words)

1.2 Registers

All registers are 16 bits wide and initialised to `0x0000`.

There are eight registers. Register 0 always reads the value 0, even after being written to. The remaining 7 registers are numbered sequentially from 1 and are general purpose registers. Registers r6 and r7 can be identified as such or as the stack pointer and link register. Specific instructions will use these as implicit register arguments.

Code	Mnemonic	Behaviour
000	rz	Always reads zero, ignores writes
001	r1	General purpose register
010	r2	General purpose register
011	r3	General purpose register
100	r4	General purpose register
101	r5	General purpose register
110	sp	Stack pointer
111	lr	Link register

1.3 Instruction Encoding

There are three instruction formats. Each instruction has one of these formats:

- Immediate Format (I-Format),
- Jump Format (J-Format),
- Register Operand Format (R-Format) or,
- Long Immediate Format (L-Format).

These formats describe the segmentation and mapping of bits in an instruction into the different data fields.

1.3.1 Immediate Format (I-Format)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op				0	rd			imm8							

1.3.2 Jump Format (J-Format)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op				0000				imm8							

1.3.3 Register Operand Format (R-Format)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op				0	rd			0	ra			0	rb		

1.3.4 Long Immediate Format (L-Format)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op				imm12											

1.4 Hardware Instructions

Syntax	Semantics	Machine Code	Flags
I-Format Instructions			
`put rd, imm8`	`rd := #imm8`	`0000 0 <rd> <imm8>`	-
J-Format Instructions			
`j<c> imm8`	`if (c) pc := pc + signExt(imm8)`	`0010 <c> <imm8>`	-
Function Call Instructions			
`call imm12`[1]	`lr := pc + 1; pc := pc + signExt(imm12)`	`0011 <imm12>`	
`ret`	`pc := lr`	`0011 0000 0000 0000`	
R-Format Memory Instructions			
`ldr rd, [ra]`	`rd := [ra]`	`0100 0 <rd> 0 <ra> 0000`	-
`str rd, [ra]`	`[ra] := rd`	`0101 0 <rd> 0 <ra> 0000`	-
R-Format ALU Instructions			
`add rd, ra, rb`	`rd := ra + rb`	`1000 0 <rd> 0 <ra> 0 <rb>`	ZNCV
`orr rd, ra, rb`	`rd := ra | rb`	`1001 0 <rd> 0 <ra> 0 <rb>`	ZN
`and rd, ra, rb`	`rd := ra & rb`	`1010 0 <rd> 0 <ra> 0 <rb>`	ZN
`not rd, ra`	`rd := ~ra`	`1011 0 <rd> 0 <ra> 0 000`	ZN
`sub rd, ra, rb`	`rd := ra - rb`	`1100 0 <rd> 0 <ra> 0 <rb>`	ZNCV

1.4.1 Condition Codes

These codes can be put after `j` instructions to change the condition on which it will branch.

Mnemonic	Condition	Code
`nz`	`Z = 0`	`0000`
`z`	`Z = 1`	`0001`
`al` or none	always	`1111`

1.4.2 Pseudo Instructions

These instructions are special cases of the instructions above.

Pseudo	Translation
`nop`	`put rz, 0`
`cmp ra, rb`	`sub rz, ra, rb`
`mov rd, ra`	`add rd, ra, rz`

1.5 Flags

Flags are set by flag-setting instructions. They are read by the `j<c>` instructions and used to conditionally determine whether to add the immediate value to the program counter.

Flag	Condition
Z	Set if the result is zero
N	Set if the result is negative (MSB=1)
C	Set on arithmetic carry
V	Set on arithmetic over- or underflow