SM213 Instruction Set Architecture

2012-02-02 02:55:30

SM213 machine language instructions are 2 or 6 bytes. The first 2 bytes are split into 4 hex digits of 4 bits each for the opcode and the three operands: `Opcode`, `Op0`, `Op1`, `Op2`. There are 16 possible opcodes, numbered `0` through `F` in hex. The meaning and use of the three operands is different for each opcode, and is given in the first table using these letters: for registers 0-7, s for source, d for destination, i for index. The letters p and o for offsets represents an actual number, not a register, and may use either one or two hex digits. Offsets indicated with o in assembly are stored in compressed form as p in machine code (meaning $o \times 2$ or $o \times 4$, as in the semantics column). The placeholder `-` means the hex digit is ignored. The second table contains examples.

Operation	Machine Language	Semantics/RTL	Assembly
load immediate	`0`d`--` *vvvvvvvv*	r[d] ← *vvvvvvvv*	`ld $`*vvvvvvvv*`, r`d
load base+dis	`1`psd	r[d] ← m[($o = p \times 4$) + r[s]]	`ld `o`(r`s`), r`d
load indexed	`2`sid	r[d] ← m[r[s] + r[i] × 4]	`ld (r`s`,r`i`,4), r`d
store base+dis	`3`spd	m[($o = p \times 4$) + r[d]] ← r[s]	`st r`s`, `o`(r`d`)`
store indexed	`4`sdi	m[r[d] + r[i] × 4] ← r[s]	`st r`s`, (r`d`,r`i`,4)`
halt	`F000`	(stop execution)	`halt`
nop	`FF00`	(do nothing)	`nop`
rr move	`60`sd	r[d] ← r[s]	`mov r`s`, r`d
add	`61`sd	r[d] ← r[d] + r[s]	`add r`s`, r`d
and	`62`sd	r[d] ← r[d] & r[s]	`and r`s`, r`d
inc	`63-`d	r[d] ← r[d] + 1	`inc r`d
inc addr	`64-`d	r[d] ← r[d] + 4	`inca r`d
dec	`65-`d	r[d] ← r[d] − 1	`dec r`d
dec addr	`66-`d	r[d] ← r[d] − 4	`deca r`d
not	`67-`d	r[d] ←~r[d]	`not r`d
shift	`7`dvv	r[d] ← r[d] << vv	`shl $`vv`, r`d
		r[d] ← r[d] >> $-vv$ (if vv is negative)	`shr $-`vv`, r`d
branch	`8-`pp	pc ← (*aaaaaaaa* = pc + $pp \times 2$)	`br` *aaaaaaaa*
branch if equal	`9`spp	if r[s] == 0 : pc ← (*aaaaaaaa* = pc + $pp \times 2$)	`beq r`s`,` *aaaaaaaa*
branch if greater	`A`spp	if r[s] > 0 : pc ← (*aaaaaaaa* = pc + $pp \times 2$)	`bgt r`s`,` *aaaaaaaa*
jump	`B---` *aaaaaaaa*	pc ← *aaaaaaaa* with `.pos` *aaaaaaaa* *label*:	`j` *label*
get program counter	`6F`pd	r[d] ← pc + ($o = 2 \times p$)	`gpc $`o`, r`d
jump indirect	`C`spp	pc ← r[s] + ($o = 2 \times pp$)	`j `o`(r`s`)`
jump double ind,b+disp	`D`spp	pc ← m[($o = 4 \times pp$) + r[s]]	`j *`o`(r`s`)`
jump double ind,index	`E`si`-`	pc ← m[4 × r[i] + r[s]]	`j *(r`s`,r`i`,4)`

Operation	Machine Language Example	Assembly Language Example
load immediate	`0100 00001000`	`ld $0x1000, r1`
load base+dis	`1123`	`ld 4(r2), r3`
load indexed	`2123`	`ld (r1,r2,4), r3`
store base+dis	`3123`	`st r1, 8(r3)`
store indexed	`4123`	`st r1, (r2,r3,4)`
halt	`F000`	`halt`
nop	`FF00`	`nop`
rr move	`6012`	`mov r1, r2`
add	`6112`	`add r1, r2`
and	`6212`	`and r1, r2`
inc	`6301`	`inc r1`
inc addr	`6401`	`inca r1`
dec	`6501`	`dec r1`
dec addr	`6601`	`deca r1`
not	`6701`	`not r1`
shift	`7102`	`shl $2, r1`
	`71FE`	`shr $2, r1`
branch	`1000: 8004`	`br 0x100a`
branch if equal	`1000: 9104`	`beq r1, 0x100a`
branch if greater	`1000: A104`	`bgt r1, 0x100a`
jump	`B000 00001000`	`j label` and elsewhere `.pos 0x1000 label:`
get program counter	`6F31`	`gpc $6, r1`
jump indirect	`C102`	`j 8(r1)`
jump double ind, b+disp	`D102`	`j *8(r1)`
jump double ind, index	`E120`	`j *(r1,r2,4)`