SM213 Instruction Set Architecture

These two tables describe the SM213 ISA. The first gives a template for instruction machine and assembly language and describes instruction semantics. The second table gives an example of each instruction. With the exception of pc-relative branches and shift, all immediate values stored in instructions are unsigned.

Operation	Machine Language	Semantics / RTL	Assembly
load immediate	`0d-- vvvvvvvv`	$r[d] \leftarrow v$	`ld $v,rd`
load base+offset	`1psd`	$r[d] \leftarrow m[(o = p \times 4) + r[s]]$	`ld o(rs),rd`
load indexed	`2sid`	$r[d] \leftarrow m[r[s] + r[i] \times 4]$	`ld (rs,ri,4),rd`
store base+offset	`3spd`	$m[(o = p \times 4) + r[d]] \leftarrow r[s]$	`st rs,o(rd)`
store indexed	`4sdi`	$m[r[d] + r[i] \times 4] \leftarrow r[s]$	`st rs,(rd,ri,4)`
halt	`F0--`	(stop execution)	`halt`
nop	`FF--`	(do nothing)	`nop`
rr move	`60sd`	$r[d] \leftarrow r[s]$	`mov rs, rd`
add	`61sd`	$r[d] \leftarrow r[d] + r[s]$	`add rs, rd`
and	`62sd`	$r[d] \leftarrow r[d]$ & $r[s]$	`and rs, rd`
inc	`63-d`	$r[d] \leftarrow r[d] + 1$	`inc rd`
inc addr	`64-d`	$r[d] \leftarrow r[d] + 4$	`inca rd`
dec	`65-d`	$r[d] \leftarrow r[d] - 1$	`dec rd`
dec addr	`66-d`	$r[d] \leftarrow r[d] - 4$	`deca rd`
not	`67-d`	$r[d] \leftarrow\sim r[d]$	`not rd`
shift	`7dss` $(ss > 0)$	$r[d] \leftarrow r[d] << (v = ss)$	`shl $v, rd`
	`7dss` $(ss < 0)$	$r[d] \leftarrow r[d] >> (v = -ss)$	`shr $v, rd`
branch	`8-pp`	$pc \leftarrow (a = pc + p \times 2)$	`br a`
branch if equal	`9rpp`	if $r[r] == 0:\ pc \leftarrow (a = pc + p \times 2)$	`beq rr, a`
branch if greater	`Arpp`	if $r[r] > 0:\ pc \leftarrow (a = pc + p \times 2)$	`bgt rr, a`
jump	`B--- aaaaaaaa`	pc $\leftarrow a$	`j a`
get program counter	`6Fpd`	$r[d] \leftarrow$ pc $+ (o = 2 \times p)$	`gpc $o, rd`
jump indirect	`Cdpp`	$pc \leftarrow r[d] + (o = 2 \times p)$	`j o(rd)`
jump double ind, b+off	`Ddpp`	$pc \leftarrow m[(o = 4 \times p) + r[d]]$	`j *o(rd)`
jump double ind, index	`Edi-`	$pc \leftarrow m[4 \times r[i] + r[d]]$	`j *(rd,ri,4)`
atomic exchange	N/A	$r[b] \leftarrow m[r[a]]$ $m[r[a]] \leftarrow r[b]$	`xchg (ra), rb`
system call	`F1nn`	* See section on next page	`sys $n`

Operation	Machine Language Example	Assembly Language Example
load immediate	`0100 00001000`	`ld $0x1000,r1`
load base+offset	`1123`	`ld 4(r2),r3`
load indexed	`2123`	`ld (r1,r2,4),r3`
store base+offset	`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: 8003`	`br 0x1008`
branch if equal	`1000: 9103`	`beq r1, 0x1008`
branch if greater	`1000: a103`	`bgt r1, 0x1008`
jump	`b000 00001000`	`j 0x1000`
get program counter	`6f31`	`gpc $6, r1`
jump indirect	`c104`	`j 8(r1)`
jump double ind, b+off	`d102`	`j *8(r1)`
jump double ind, index	`e120`	`j *(r1,r2,4)`
atomic exchange	N/A	`xchg (r1), r2`

The system call instruction encodes which system call to use. Arguments are passed in the registers `r0, r1, r2` in that order. `fd` refers to a file descriptor: `fd=0` refers to `stdin` and `fd=1` refers to `stdout`. On error, all system calls will return `-1`.

Assembly	Description	Function Signature	Result (in `r0`)
`sys $0`	Read data from file `fd`	`read(fd, buffer, size)`	number of bytes read
`sys $1`	Write data to file `fd`	`write(fd, buffer, size)`	number of bytes written
`sys $2`	Execute program `buf`	`exec(buffer, size)`	0