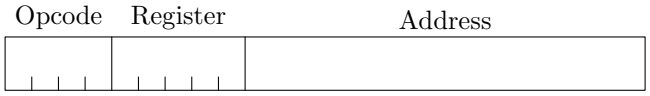
S20 Computer Instruction Set

The S20 is a small 24-bit computer aimed at IoT applications, and includes 32KW of memory. Only integer operations are supported. It has 32 registers with r0 always containing the value 0, r31 containing the program counter, and r30 containing the stack pointer. The following table summarizes the instruction set:

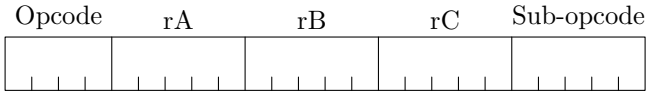
Opcode	Sub-opcode	Instruction	Description
0x1		ld	Load from memory to register
0x2		st	Store from register to memory
0x3		br	Unconditional branch (register is ignored)
0x4		bsr	Branch to subroutine (register is ignored)
0x5		brz	Branch if zero
0x6		bnz	Branch if non-zero
0x7		brn	Branch if negative
0x8		bnn	Branch if non-negative
0x0	0x00	nop	No operation (registers ignored)
0x0	0x01	ldi	Load indirect
0x0	0x02	sti	Store indirect
0x0	0x03	add	Add registers
0x0	0x04	sub	Subtract registers
0x0	0x05	and	Bit-wise AND
0x0	0x06	or	Bit-wise OR
0x0	0x07	xor	Bit-wise XOR
0x0	0x08	shl	Logical left shift
0x0	0x09	sal	Arithmetic left shift
0x0	0x0a	shr	Logical right shift
0x0	0x0b	sar	Arithmetic right shift
0x0	0x10	rts	Return from subroutine (registers ignored)
0x0	0x1f	halt	Halt (registers ignored)

For those instructions that do not list a sub-opcode, instructions are encoded according to the following figure:



The most significant four bits (20–23) are the opcode found in the first column of the table. Bits 15–19 specify a register number. For the `ld` and `st` instructions, this register is the source or the destination of the move. For the conditional branch instructions, this register is the one tested for the condition on which to branch. The remaining bits (0–14) are the relevant memory address for the copy or the destination of the branch.

All other instructions are encoded as follows:



In this instruction format, bits 20–23 are all 0s. Bits 5–19 are three fields of five bits each, specifying three registers. The low-order five bits (0–4) specify the sub-opcode given in the second column of the table. We designate the register specified in bits 15–19 as rA, the one in bits 10–14 as rB, and the one in bits 5–9

as rC. For `nop`, `rts`, and `halt`, none of the three registers are specifed. For the `ldi` and `sti` instructions, the memory address of the load or store is given by the sum of the contents of registers rA and rB, and rC is the destination register of an `ldi` and the source register of an `sti`. For `add`, `sub`, `and`, `or`, and `xor`, rA and rB specify the two registers used in the operation, and rC specifies the register where the result is stored. The shift instructions also use rA and the source register, rC as the destination register, but the bits for rB give the number of bits the word is shifted.

The following is an example of some code and it's assembled machine code (all values in hex):

```
0000  10800a           ld    x, r1
0001  11000b           ld    y, r2
0002  008863           add   r1, r2, r3
0003  019488           shl   r3, 5, r4
0004  720007           brn   r4, skip
0005  22000c           st    r4, z
0006  00001f           halt
0007  001084  skip     sub   r0, r4, r4
0008  22000c           st    r4, z
0009  00001f           halt
000a  000000  x        data  0
000b  000000  y        data  0
000c  000000  z        data  0
```

The S20 is a bit-endian machine. Thus in the binary the most significan byte comes first. The code in the program above would be stored in the executable file as:

```
10 80 0a 11 00 0b 00 88 63 01 94 88 72 00 07 22
00 0c 00 00 1f 00 10 84 22 00 0c 00 00 1f 00 00
00 00 00 00 00 00 00
```