Simple Instruction Architecture

The simple instruction architecture (SIA) is designed to be an architecture that is easy to assemble, easy to create a virtual machine for and easy for beginners to computer architecture to understand. While a typical RISC (reduced instruction set computer/chip) has dozens of instructions. SIA has 16 instructions.

SIA has 16 registers, numbered from 0-15 (4 bit selector). Unlike many RISC chips, register 0 is not a constant 0; it is a general-purpose register. All registers are 32 bits wide. All registers are interpreted as signed integers for the purpose of mathematical operations.

To keep the instruction count low, a number of instructions that are traditionally available on most systems are omitted in SIA. Most of these are easily replaced with one or more SIA instructions. For example – there is no "clear" instruction to set a register to 0. This is easily fixed by using "sub":
sub r1 r1 r1 will subtract r1 from itself, giving 0.
There is no load immediate to set a register to a particular value. This can be replaced with 2 instructions:
sub r1 r1 r1 ; set r1 to 0
addimmediate r1 25 ; adds 25 to r1, making r1 = 25

NOP (no operation) is not available in SIA, but can be emulated with addimmediate:
addimmediate r0 0 ; adds 0 to R0, doing nothing

Branch if less than and branch if equal are provided; branch if greater than can be performed by swapping the operands of branch if less. branch if not equal can be performed with branch if less than and branch if greater than pointing to the same address. Likewise, branch if less than or equal to can be performed with a branch if less than and a branch if equal pointing to the same address.

IMPORTANT NOTE:
Since all instructions are on a 16-bit boundary, the lowest bit of addresses is not stored, since it will always be 0.

Instructions

add (opcode 1)

Adds the values of 2 registers and places the answer in a third register.
Example: add r1 r2 r3 ; r3 ← r1 + r2
Instruction format: 3R

addimmediate (opcode 9)

Adds a **signed** 8 byte value to a register
Example: addimediate r1 -127 ; adds -127 to r1
Instruction format: ai

and (opcode 2)

Preforms a bitwise and on 2 registers and stores the result in a third register
Example: and r1 r2 r3 ; r3 ← r1 & r2
Instruction format: 3R

branchifequal (opcode 10)

Compares two registers – if they hold the same value, jump to an offset from the current program counter. The offset must be divisible by 2 and can be between -524,286 and 524,286
Example: branchifequal r1 r2 1000
Instruction format: br

branchIfless (opcode 11)

Compares two registers – if the first is less than the second, jump to an offset from the current program counter. The offset must be divisible by 2 and can be between -524,286 and 524,286
Example: branchifless r1 r2 1000
Instruction format: br

divide (opcode 3)

Divides the value of the first register by the second and places the answer in a third register. This is integer math with the fractional portion discarded.
Example: divide r1 r2 r3 ; r3 ← r1 / r2
Instruction format: 3R

halt (opcode 0)

Stops the CPU.
Example: halt
Instruction format: 3R (the register values don't matter)

interrupt (opcode 8)

Interrupts the CPU using a particular interrupt number. This could be used to jump between kernel mode and user mode or to support devices.

Example: interrupt 17
Instruction format: int

iterateover (opcode 13)

Iterateover is for walking linked lists. One register is selected as the pointer to the current node. An offset is added (unsigned) to the contents of that register and that address is fetched from main memory. If that address is not null (0), the current register receives the value fetched from memory and the program jumps BACKWARDS delta instructions.
Example: interateover r1, 4, 20
If memory at address 52 looks like this:

Address	Value	Comment
52	88	value 1 in a linked list
56	60	next node in our linked list
60	71	value 2 in a linked list
64	0	next node (null) in a linked list

This program will "walk" the linked list:

```
sub r1 r1 r1 ; clear r1
addimmediate r1 52 ; set r1 to the start address of the linked list
load r2 r1 0 ; set r2 = what r1 points to (88 for the first iteration)
interrupt 0 ; print all of the registers
iterateover r1 4 4 ; Loop using r1, with the "next" pointer 4 bytes from the
beginning, jumping 4 bytes (two instructions in this case) back for each
successful iteration
```

instruction format: iter

jump (opcode 12)

Jumps to the location specified in the instruction (0 – 536,870,911)
Example: jump 1000
instruction format: jmp

leftshift (opcode 7)

Shifts the contents of the specified register left by the specified number of bits (0-31).
Example: leftshift r1 10
instruction format: sft

load (opcode 14)

Loads a register from the memory pointed to by another register offset by a value (-7 to 7)
Example: load r1 r2 10 ; loads r1 with the value pointed to by r2 plus 10 bytes
instruction format: ls

multiply (opcode 4)

Multiplies the value of the first register times the second and places the answer in a third register.
Example: multply r1 r2 r3 ; r3 ← r1 * r2
Instruction format: 3R

or (opcode 6)

Preforms a bitwise OR on 2 registers and stores the result in a third register
Example: or r1 r2 r3 ; r3 ← r1 | r2
Instruction format: 3R

rightshift (opcode 7)

Shifts the contents of the specified register right by the specified number of bits (0-31).
Example: rightshift r1 10
instruction format: sft

store (opcode 15)

Stores a register's value into memory pointed to by another register offset by a value (-7 to 7)
Example: store r1 r2 10 ; stores r1's value into the memory pointed to by r2 plus 10 bytes
instruction format: ls

subtract (opcode 5)

Subtracts the value of the second register from the first and places the answer in a third register.
Example: subtract r1 r2 r3 ; r3 ← r1 - r2
Instruction format: 3R

Instruction Formats

3R

4 bits	4 bits	4 bits	4 bits
OPCODE	register 1	register 2	destination (register 3)

ai

4 bits	4 bits	8 bits
OPCODE	register 1	immediate value (signed)

br

4 bits	4 bits	4 bits	4 bits
OPCODE	register 1	register 2	top 4 bits of address offset

16 bits
16 bits of address offset

iter

4 bits	4 bits	8 bits
OPCODE	register	next pointer offset

16 bits
16 bits of jump address offset

int

4 bits	12 bits
OPCODE	interrupt

jmp

4 bits	12 bits
OPCODE	top 12 bits of jump address

16 bits
lower 16 bits of jump address

ls

4 bits	4 bits	4 bits	4 bits
OPCODE	register to load/store	address register	address offset (signed)

sft

4 bits	4 bits	2 bits	1 bit	5 bits
OPCODE	register	unused	set for right shift, clear for left shift	shift amount