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.

This implementation of SIA has a stack – hardware support for a stack data structure that can hold function parameters, local variables, return addresses, and return values. **R15** is the stack register – its current address is the bottom most byte of the stack.

***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

## “3R” instructions

### 3R

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

### 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

### 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

### 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)

### multiply (opcode 4)

Multiplies the value of the first register times the second and places the answer in a third register.

Example: multiply r1 r2 r3 ; r3 🡸 r1 \* r2

Instruction format: 3R

### or (opcode 6)

Performs 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

### 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

## Branch instructions (ALL OPCODE 7)

### br1

|  |  |  |  |
| --- | --- | --- | --- |
| 4 bits | 4 bits | 4 bits | 4 bits |
| OPCODE | Branch type | register 1 | register 2 |

|  |
| --- |
| 16 bits |
| 16 bits of address offset |

OR

### br2

|  |  |  |
| --- | --- | --- |
| 4 bits | 4 bits | 8 bits |
| OPCODE | Branch type | Top 8 bits of address |

|  |
| --- |
| 16 bits |
| Bottom 16 bits of address |

### branchXXXXXXXXX (opcode 7)

Conditionally branches based on register 1 and register 2. There are 6 branch instructions:

|  |  |  |
| --- | --- | --- |
| Command | Branch Type | Condition |
| branchIfLess | 0 | if (r1 < r2) |
| branchIfLessOrEqual | 1 | if (r1 <= r2) |
| branchIfEqual | 2 | if (r1 == r2) |
| branchIfNotEqual | 3 | if (r1 != r2) |
| branchIfGreater | 4 | if (r1 > r2) |
| branchIfGreaterOrEqual | 5 | if (r1 >= r2) |

If the condition is true, jump to an offset from the current program counter. (The offset is the number of **words (2 bytes)** forward or back. PC <= PC + (2 \* offset).

Example: branchifequal r1 r2 1000

Instruction format: br1

### call (opcode 7)

Calls a “function” – pushes the PC of the next instruction onto the stack (R15), then jumps to the address specified by this instruction times 2.

Example: call 444

Instruction format: br2, branch type 6

### jump (opcode 7)

Jumps to the location specified in the instruction times 2

Example: jump 1000

instruction format: br2, branch type 7

### Load/Store instructions

### ls

|  |  |  |  |
| --- | --- | --- | --- |
| 4 bits | 4 bits | 4 bits | 4 bits |
| OPCODE | register to load/store | address register | address offset |

### load (opcode 8)

Loads a register from the memory pointed to by another register plus 2 times the offset (0 to 30). Note that both the address in the register and the offset are in words (memory locations).

Example: load r1 r2 10 ; loads r1 with the value pointed to by r2 plus 20 bytes

instruction format: ls

### store (opcode 9)

Stores a register’s value into memory pointed to by another register plus 2 times the offset (0 to 30). Note that both the address in the register and the offset are in words (memory locations).

Example: store r1 r2 10 ; stores r1’s value into the memory pointed to by r2 plus 20 bytes

instruction format: ls

## Stack Instructions

### stack

|  |  |  |  |
| --- | --- | --- | --- |
| 4 bits | 4 bits | 2 bits | 6 bits |
| OPCODE | register | 00 = return, 01 = push, 10 = pop | Unused (set to 0) |

### pop (opcode 10)

Copies data from stack pointer through stack pointer + 3 to specified register. Adds four to the stack pointer.

Example: pop R1

Instruction format: stack

### push (opcode 10)

Subtracts four from the stack pointer. Takes the value in the specified register and stores it in the memory address indicated by the stack pointer.

Example: push R1

Instruction format: stack

### return (opcode 10)

Pops the top value from the stack and jumps to that address. Register is ignored for return.

Example: return

Instruction format: stack

### Move

move

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

### move (opcode 11)

Sets a register to a **signed** 8 bit value

Example: move -127 r1; sets register R1 to -127

Instruction format: move

### interrupt (opcode 12)

Interrupts the CPU using a particular interrupt number. This could be used to jump between kernel mode and user mode or to support devices. For the virtual machine, two interrupts are supported: 0 (print registers) and 1 (print out memory).

Example: interrupt 17

Instruction format: move Note that the register is ignored.