Cake++ Virtual Machine Details

- This is a stack machine that uses byte addressing.
- Basic data types supported for arithmetic are
  - `uint64_t`, `int64_t`
- 8 integer types are supported for loads and stores:
  - `uint8_t`, `int8_t`, `uint16_t`, `int16_t`, `uint32_t`, `int32_t`, `uint64_t`, `int64_t`
- Registers:
  - **`pc`**: program counter
  - **`sp`**: stack pointer
  - **`fp`**: frame pointer (points to the start of the stack frame)
- Stack frame:
  - **`fp - arg_space - 8`**: return value
  - **`fp - arg_space`**: arguments to this specific call of the function, in reverse order.
    - Example: for a function with two arguments, the first argument to the function is at (**`fp - 8`**), and the second argument to the function is at (**`fp - 16`**)
  - **`fp`**: return address
  - **`fp + 8`**: old **fp**
  - **`fp + 16`**: local variables of this specific call of the function
  - **`fp + 16 + var_space`**: temporaries (basically just stuff on the stack)
- Instructions:
  - Group 0: Non-immediate Arithmetic/Logic instructions, Non-immediate-indexed loads and stores, non-immediate-indexed jumps
    - Encoding: `0000 0000 oooo oooo`
      - o: opcode
    - List:
      - **add**
        - Opcode: `0x00`
        - Effect: **`push(pop() + pop());`**
      - **sub**
        - Opcode: `0x01`
        - Effect: **`push(pop() - pop());`**
      - **mul**
        - Opcode: `0x02`
        - Effect: **`push(pop() * pop());`**
      - **udiv**
        - Opcode: `0x03`
        - Note: divide unsigned
        - Effect: **`push(pop() udiv pop());`**
      - **sdiv**
        - Opcode: `0x04`
        - Note: divide signed
        - Effect: **`push(pop() sdiv pop());`**
      - **umod**
        - Opcode: `0x05`

    - Note: modulo unsigned
    - Effect: `push(pop() umod pop());`
- **smod**
    - Opcode: `0x05`
    - Note: modulo signed
    - Effect: `push(pop() smod pop());`
- **uslt**
    - Opcode: `0x06`
    - Note: set less than unsigned
    - Effect: `push(pop() uslt pop());`
- **sslt**
    - Opcode: `0x06`
    - Note: set less than signed
    - Effect: `push(pop() sslt pop());`
- **and**
    - Opcode: `0x07`
    - Note: bitwise and
    - Effect: `push(pop() & pop());`
- **or**
    - Opcode: `0x08`
    - Note: bitwise or
    - Effect: `push(pop() | pop());`
- **xor**
    - Opcode: `0x09`
    - Note: bitwise xor
    - Effect: `push(pop() ^ pop());`
- **lsl**
    - Opcode: `0x0a`
    - Note: logical shift left
    - Effect: `push(pop() << pop());`
- **lsr**
    - Opcode: `0x0b`
    - Note: logical shift right
    - Effect: `push(pop() lsr pop());`
- **asr**
    - Opcode: `0x0c`
    - Note: arithmetic shift right
    - Effect: `push(pop() asr pop());`
- **ldxu8**
    - Opcode: `0x0d`
    - Effect: `push(zero_extend_to_64(mem8[pop() + pop()]));`
- **ldxs8**
    - Opcode: `0x0e`
    - Effect: `push(sign_extend_to_64(mem8[pop() + pop()]));`
- **ldxu16**
    - Opcode: `0x0f`
    - Effect: `push(zero_extend_to_64(mem16[pop() + pop()]));`
- **ldxs16**
    - Opcode: `0x10`

        - Effect: `push(sign_extend_to_64(mem16[pop() + pop()]));`
    - **ldxu32**
        - Opcode: `0x11`
        - Effect: `push(zero_extend_to_64(mem32[pop() + pop()]));`
    - **ldxs32**
        - Opcode: `0x12`
        - Effect: `push(sign_extend_to_64(mem32[pop() + pop()]));`
    - **ldx64**
        - Opcode: `0x13`
        - Effect: `push(mem64[pop() + pop()]);`
    - **stx8**
        - Opcode: `0x14`
        - Effect: `mem8[pop() + pop()] = (pop()[7:0]);`
    - **stx16**
        - Opcode: `0x15`
        - Effect: `mem16[pop() + pop()] = (pop()[15:0]);`
    - **stx32**
        - Opcode: `0x16`
        - Effect: `mem32[pop() + pop()] = (pop()[31:0]);`
    - **stx64**
        - Opcode: `0x17`
        - Effect: `mem64[pop() + pop()] = pop();`
    - **jmpx**
        - Opcode: `0x18`
        - Effect: `pc = (pop() + pop());`
    - **jfal**
        - Opcode: `0x19`
        - Effect:
```
cond = pop(); temp = pop();
if (cond == 0) { pc = temp; }
```
    - **jtru**
        - Opcode: `0x1a`
        - Effect:
```
cond = pop(); temp = pop();
if (cond != 0) { pc = temp; }
```
- Group 1: Immediate arithmetic/logic instructions, immediate-indexed loads and stores, and jumps that use immediates
    - Encoding: `0000 0001 oooo oooo iiii iiii iiii iiii`
        - o: opcode
        - i: sign-extended 16-bit immediate
    - List:
        - **addi simm16**
            - Opcode: `0x00`
            - Effect: `push(pop() + sign_extend_to_64(simm16));`
        - **subi simm16**
            - Opcode: `0x01`
            - Effect: `push(pop() - sign_extend_to_64(simm16));`
        - **muli simm16**
            - Opcode: `0x02`

        - Effect: **push**(**pop**() * **sign_extend_to_64**(simm16));
- **udivi simm16**
    - Opcode: 0x03
    - Note: divide unsigned
    - Effect: **push**(**pop**() udiv **sign_extend_to_64**(simm16));
- **sdivi simm16**
    - Opcode: 0x04
    - Note: divide signed
    - Effect: **push**(**pop**() sdiv **sign_extend_to_64**(simm16));
- **umodi simm16**
    - Opcode: 0x05
    - Note: modulo unsigned
    - Effect: **push**(**pop**() umod **sign_extend_to_64**(simm16));
- **smodi simm16**
    - Opcode: 0x05
    - Note: modulo signed
    - Effect: **push**(**pop**() smod **sign_extend_to_64**(simm16));
- **uslti simm16**
    - Opcode: 0x06
    - Note: set less than unsigned
    - Effect: **push**(**pop**() uslt **sign_extend_to_64**(simm16));
- **sslti simm16**
    - Opcode: 0x06
    - Note: set less than signed
    - Effect: **push**(**pop**() sslt **sign_extend_to_64**(simm16));
- **andi simm16**
    - Opcode: 0x07
    - Note: bitwise and
    - Effect: **push**(**pop**() & **sign_extend_to_64**(simm16));
- **ori simm16**
    - Opcode: 0x08
    - Note: bitwise or
    - Effect: **push**(**pop**() | **sign_extend_to_64**(simm16));
- **xori simm16**
    - Opcode: 0x09
    - Note: bitwise xor
    - Effect: **push**(**pop**() ^ **sign_extend_to_64**(simm16));
- **lsli simm16**
    - Opcode: 0x0a
    - Note: logical shift left
    - Effect: **push**(**pop**() << **sign_extend_to_64**(simm16));

- **lsri simm16**
  - Opcode: `0x0b`
  - Note: logical shift right
  - Effect: **push**(**pop**() lsr **sign_extend_to_64**(simm16));
- **asri simm16**
  - Opcode: `0x0c`
  - Note: arithmetic shift right
  - Effect: **push**(**pop**() asr **sign_extend_to_64**(simm16));
- **ldxu8i simm16**
  - Opcode: `0x0d`
  - Effect: **push**(**zero_extend_to_64**(**mem8**[**pop**()
  - **sign_extend_to_64**(simm16)]));
- **ldxs8i simm16**
  - Opcode: `0x0e`
  - Effect: **push**(**sign_extend_to_64**(**mem8**[**pop**()
  - **sign_extend_to_64**(simm16)]));
- **ldxu16i simm16**
  - Opcode: `0x0f`
  - Effect: **push**(**zero_extend_to_64**(**mem16**[**pop**()
  - **sign_extend_to_64**(simm16)]));
- **ldxs16 simm16**
  - Opcode: `0x10`
  - Effect: **push**(**sign_extend_to_64**(**mem16**[**pop**()
  - **pop**()]));
- **ldxu32i simm16**
  - Opcode: `0x11`
  - Effect: **push**(**zero_extend_to_64**(**mem32**[**pop**()
  - **sign_extend_to_64**(simm16)]));
- **ldxs32i simm16**
  - Opcode: `0x12`
  - Effect: **push**(**sign_extend_to_64**(**mem32**[**pop**()
  - **sign_extend_to_64**(simm16)]));
- **ldx64i simm16**
  - Opcode: `0x13`
  - Effect: **push**(**mem64**[**pop**() + **sign_extend_to_64**(simm16)]);
- **stx8i simm16**
  - Opcode: `0x14`
  - Effect: **mem8**[**pop**() + **sign_extend_to_64**(simm16)] = (**pop**()[7:0]);
- **stx16i simm16**
  - Opcode: `0x15`
  - Effect: **mem16**[**pop**() + **sign_extend_to_64**(simm16)] = (**pop**()[15:0]);
- **stx32i simm16**
  - Opcode: `0x16`
  - Effect: **mem32**[**pop**() + **sign_extend_to_64**(simm16)] = (**pop**()[31:0]);
- **stx64i simm16**
  - Opcode: `0x17`

        - Effect: `mem64[pop() + sign_extend_to_64(simm16)] = pop();`
    - **jmpxi simm16**
        - Opcode: `0x18`
        - Effect: `pc = (pop() + sign_extend_to_64(simm16));`
    - **bfal simm16**
        - Opcode: `0x19`
        - Effect: `if (pop() == 0) { pc = pc + sign_extend_to_64(simm16); }`
    - **btru simm16**
        - Opcode: `0x1a`
        - Effect: `if (pop() != 0) { pc = pc + sign_extend_to_64(simm16); }`
- Group 2: Constants
    - List:
        - **constu8 uimm8**
            - Encoding: `0000 0010 0000 0000 iiii iiii`
            - Effect: `push(zero_extend_to_64(uimm8));`
        - **consts8 simm8**
            - Encoding: `0000 0010 0000 0001 iiii iiii`
            - Effect: `push(sign_extend_to_64(simm8));`
        - **constu16 uimm16**
            - Encoding: `0000 0010 0000 0010 iiii iiii iiii iiii`
            - Effect: `push(zero_extend_to_64(uimm16));`
        - **consts16 simm16**
            - Encoding: `0000 0010 0000 0011 iiii iiii iiii iiii`
            - Effect: `push(sign_extend_to_64(simm16));`
        - **constu32 uimm32**
            - Encoding: `0000 0010 0000 0100 iiii iiii iiii iiii iiii iiii iiii iiii`
            - Effect: `push(zero_extend_to_64(uimm32));`
        - **consts32 simm32**
            - Encoding: `0000 0010 0000 0101 iiii iiii iiii iiii iiii iiii iiii iiii`
            - Effect: `push(sign_extend_to_64(simm32));`
        - **const imm64**
            - Encoding: `0000 0010 0000 0110 iiii iiii iiii iiii iiii iiii iiii iiii iiii iiii iiii iiii iiii iiii iiii`
            - Effect: `push(imm64);`
- Group 3: `arg`, `var`, `get_pc`
    - Encoding: `0000 0011 oooo oooo`
        - o: opcode
    - List:
        - **arg**
            - Opcode: `0x00`
            - Effect: `push(fp - 8);`
        - **var**
            - Opcode: `0x01`

        - Effect: `push(fp + 16);`
    - **get_pc**
        - Opcode: `0x02`
        - Effect: `push(pc);`
- Group 4: `argx`, `varx`, `add_to_sp`, `call`, `ret`
    - Encoding: `0000 0100 oooo oooo`
        - o: opcode
    - List:
        - **argx**
            - Opcode: `0x00`
            - Effect: `push(fp - 8 + pop());`
        - **varx**
            - Opcode: `0x01`
            - Effect: `push(fp + 16 + pop());`
        - **add_to_sp**
            - Opcode: `0x02`
            - Effect: `sp = sp + pop();`
        - **call**
            - Opcode: `0x03`
            - Effect:

```
{
  address = pop();
  push(pc);
  old_fp = fp;
  fp = sp;
  push(old_fp)
  pc = address;
}
```

        - **ret**
            - Opcode: `0x04`
            - Effect: `return_sequence();`
        - **syscall**
            - Opcode: `0x05`
            - Effect: `exec_syscall(pop());`