Instruction Set

TM provides two kinds of instructions: **register-only** and **register-memory**.

Register-only (RO) instructions are of the form

```
opcode r1,r2,r3
```

where the `ri` are legal registers. These are the RO opcodes:

- `IN` read an integer from stdin and place result in `r1`; ignore operands `r2` and `r3`
- `OUT` write contents of `r1` to stdout; ignore operands `r2` and `r3`
- `ADD` add contents of `r2` and `r3` and place result in `r1`
- `SUB` subtract contents of `r3` from contents of `r2` and place result in `r1`
- `MUL` multiply contents of `r2` and contents of `r3` and place result in `r1`
- `DIV` divide contents of `r2` by contents of `r3` and place result in `r1`
- `HALT` ignore operands and terminate the machine

Register-memory (RM) instructions are of the form

```
opcode r1,offset(r2)
```

where the `ri` are legal registers and `offset` is an integer offset. `offset` may be negative. With the exception of the `LDC` instruction, the expression `offset(r2)` is used to compute the address of a memory location:

```
address = (contents of r2) + offset
```

There are four RM opcodes for memory manipulation:

- `LDC` place the constant `offset` in `r1`; ignore `r2`
- `LDA` place the address `address` in `r1`
- `LD` place the contents of data memory location `address` in `r1`
- `ST` place the contents of `r1` to data memory location `address`

There are six RM opcodes for branching. If the value of `r1` satisfies the opcode's condition, then branch to the instruction at instruction memory location `address`.

- `JEQ` equal to 0
- `JNE` not equal to 0
- `JLT` less than 0
- `JLE` less than or eual to 0
- `JGT` greater than 0
- `JGE` greater than or equal to 0

Note:

- All arithmetic is done with registers (not memory locations) and on integers. Floating-point numbers must be simulated in the run-time system.
- There are no restrictions on the usage of registers. For example, the source and target registers for an operation can be the same.
- This is also true of the program counter, Register 7. For example,
  - To branch unconditionally to an instruction, a program can load the target address into the PC using an `LDA` instruction.
  - To branch unconditionally to an instruction whose address is stored in data memory, a program can load the target address into the PC using an `LD` instruction.
  - To branch conditionally to an instruction whose address is relative to the current position in the program, a program can use the PC as `r2` in any of the `Jxx` instructions.