# 1 Instructions

A general jouette instruction consists of an operation code followed by one or more operands. They can be broken in to two categories, *expression* instructions and *command* instructions. Expression instructions produce a value and place it into either a data or address register.

A typical expression instruction is written as:

`opne` $[o], i_1, \ldots, i_n$

where the $[o]$ represents the output and the list of $i$s the inputs.

A command instruction has no output. It is a side-effecting operation, either changing main memory or program flow. It is typically written as:

`opnc1` $L, i_1, \ldots, i_n$

or

`opnc2` $i_1, \ldots, i_n$

where $L$ is a label and the $i$s are input to the command.

Table 1: Jouette Instruction list

| Instruction | Cst | Effect/CIRL | Description |
|---|---|---|---|
| `add` $[d_i], d_j, d_k$ | 2 | $d_i := d_j + d_k$ | Expression: Adds the contents of two data registers and places it in a third. |
| `mul` $[d_i], d_j, d_k$ | 5 | $d_i := d_j * d_k$ | Expression: Multiplies the contents of two data registers and places it in a third. |
| `sub` $[d_i], d_j, d_k$ | 2 | $d_i := d_j - d_k$ | Expression: Subtracts the contents of two data registers and places it in a third. |
| `div` $[d_i], d_j, d_k$ | 10 | $d_i := d_j / d_k$ | Expression: Divides the contents of two data registers and places it in a third. |

*Continued on next page*

| Instruction | Cst | Effect/CIRL | Description |
|---|---|---|---|
| `addi` $[d_i], d_j, c$ | 1 | $d_i := d_j + c$ | Expression: Adds a constant to a data register and places it in another. |
| `subi` $[d_i], d_j, c$ | 1 | $d_i := d_j - c$ | Expression: Subtracts a constant from a data register and places it in another. |
| `movea` $[d_i], a_j$ | 2 | $d_i := ref(a_j)$ | Expression: Moves contents of an address register to a data register. |
| `moved` $[a_i], d_j$ | 2 | $a_i := deref(d_j)$ | Expression: Moves contents of a data register to an address register. |
| `load` $[d_i], a_j, c$ | 10 | $d_i := mem[a_j \backslash c]$ | Expression: Moves memory pointed to by an address register + constant to a data register. |
| `store` $a_i, c, d_j$ | 10 | $a_i \backslash c \leftarrow d_j$ | Command: Store data register to memory pointed to by an address register + constant. |
| `movem` $a_i, a_j$ | 20 | $a_i \leftarrow mem[a_j]$ | Command: Moves memory pointed to by one address register to memory pointed to by another address register. |
| `bge` $L, d_j$ | 3 | JUMP to $L$ if $d_j \geq 0$ | Command: Jump to label (change program counter) if data register greater than or equal to 0. |
| `blt` $L, d_j$ | 3 | JUMP to $L$ if $d_j < 0$ | Command: |
| `beq` $L, d_j$ | 3 | JUMP to $L$ if $d_j = 0$ | Command: |

*Continued on next page*

| Instruction | Cst | Effect/CIRL | Description |
|---|---|---|---|
| `bne` $L, d_j$ | 3 | JUMP to $L$ if $d_j \neq 0$ | Command: |
| `jump` $L$ | 2 | JUMP to $L$ | Command: Jump to label. |
| `call` $[d_0], L, d_1, \ldots, d_n$ | 5 | JUMP to $L$, remember return location | Expression: Jump to label and remember address of next instruction. $d_0$ is the output register and $d_1, \ldots, d_n$ are the input arguments. |
| `return` $d$ | 3 | Return from last `call` | Command: return to instruction just after last call. |
| $L$: | 0 | Label next instruction | Command: Provide a destination for jump, branch and call. |

# 2 Machine

Word size is always 1 for the jouette. Registers are grouped into data registers and address registers.

Table 2: Registers

| Register(s) | Description |
|---|---|
| $d_0$ to $d_M$ | Data registers (Expressions in CIRL) |
| $a_0$ to $a_N$ | Address registers (Locations in CIRL |
| $d_0$ | ALWAYS zero. |
| $pc$ | The program counter, which is not directly referencable. Holds the location of the *next* instructions. |
| $a_0$ | The stack pointer. |