The ChristmasFlow programming manual
With examples

Written by Gabies and littlewho

25th of December 2018

0th Edition
Highly experimental!

Contents

1 Introduction

In the following paragraphs we are going to define the elementary terms and concepts used.

The instruction set is designed to support basic arithmetic and logic operations for **8-bit** integer and **boolean** data types, input/output and casting integers to character representation and execution flow control. Beside usual assembly languages, DataFlow languages contain a few particularities: there are no registers, there is no storage memory, only input data flowing to the output.

Each **instruction** represents a **node** in a **directed graph** that has an index, an input layer and an output layer. A node has an indegree equal to 2 and a variable outdegree, either 0, 1 or 2. A node exists in multiple instances called **iteration levels**. The node on one such level is executed as soon as all the necessary input is provided. The node execution is prioritized by iteration level from highest to lowest, although in the case of **loops** the order is reversed, since the output layer of the nodes with a lower iteration level feed the input layer of the ones with a higher iteration level. When a loop ends, the iteration level must be set back to the state in which it was before the beginning of the loop, in order to allow nested loops.

Conditional execution is realized using **branch** instructions that, based on input, decide to which node the output is directed.

2 Instruction formats

An instruction has the following format: **{index} {operator} {destination}**

1. **index**:

   The index of the node that represents the current instruction.

2. **operator**:

   The operator that defines the current instruction.

3. **destination**:

   The destination of the output layer of the node. There are some exceptions, since for some operators there will be two destinations, and for some there is no destination.

   Each node has two inputs, although the second input for some operands may not be necessary and therefore will be ignored.

   The destination has the following format: **{node}{input side}**

   (a) **node**:
   The index of the destination node.

   (b) **input side**:
   The side (left - abbreviated **L**, or right - abbreviated **R**) of the input layer of the destination node.

3 Operators

List of operators:

1. Input/Output operators:
   (a) REA (Reads integer from input)
   (b) PRI (Prints 8-bit integer to output)
   (c) PRC (Prints character to output)

   Use table:

Operand	Left input	Right input	Left output	Right output
REA	None	None	Integer	None
PRI	Integer	None	None	None
PRC	Integer	None	None	None

2. Arithmetic operators:
   (a) ADD (Adds two integers)
   (b) SUB (Subtracts two integers)
   (c) MUL (Multiplies two integers)
   (d) DIV (Divides two integers and returns quotient and remainder)

   Use table:

Operand	Left input	Right input	Left output	Right output
ADD	Integer	Integer	Integer	None
SUB	Integer	Integer	Integer	None
MUL	Integer	Integer	Integer	None
DIV	Integer	Integer	Integer (quotient)	Integer (remainder)

3. Boolean arithmetic operators:
   (a) AND (Boolean AND operator)
   (b) ORR (Boolean OR operator)
   (c) XOR (Boolean XOR operator)
   (d) NOT (Boolean NOT operator)

Use table:

Operand	Left input	Right input	Left output	Right output
AND	Boolean	Boolean	Boolean	None
ORR	Boolean	Boolean	Boolean	None
XOR	Boolean	Boolean	Boolean	None
NOT	Boolean	None	Boolean	None

4. Comparison operators:

   (a) SML (Smaller comparison operator)

   (b) SME (Smaller or equal comparison operator)

   (c) GRT (Greater comparison operator)

   (d) GRE (Greater or equal comparison operator)

   (e) EQU (equal comparison operator)

   (f) DIF (different comparison operator)

   Use table:

Operand	Left input	Right input	Left output	Right output
SML	Integer	Integer	Boolean	None
SME	Integer	Integer	Boolean	None
GRT	Integer	Integer	Boolean	None
GRE	Integer	Integer	Boolean	None
EQU	Integer	Integer	Boolean	None
DIF	Integer	Integer	Boolean	None

5. Conditional operator:

   (a) BRB (Branch on boolean)

   Use table:

Operand	Left input	Right input	Left output	Right output
BRB	Any	Boolean	Any (if true)	Any (if false)

6. Bitwise operators:

   (a) SHR (Shift to the right)

   (b) SHL (Shift to the left)

   (c) BAN (Bitwise AND)

(d) BOR (Bitwise OR)

(e) BXR (Bitwise XOR)

(f) BNT (Bitwise NOT)

Use table:

Operand	Left input	Right input	Left output	Right output
SHR	Integer	Integer	Integer	None
SHL	Integer	Integer	Integer	None
BAN	Integer	Integer	Integer	None
BOR	Integer	Integer	Integer	None
BXR	Integer	Integer	Integer	None
BNT	Integer	None	Integer	None

7. Index level control operators:

   (a) AIL (Add to index level)

   (b) SIL (Subtract from index level)

   Use table:

Operand	Left input	Right input	Left output	Right output
AIL	Any	Integer	Any	None
SIL	Any	Integer	Any	None

8. DataFlow specific operators:

   (a) CLN (Clones input)

   (b) SND_X (Send a constant to the input layer of another node, X is an 8-bit integer)

   Use table:

Operand	Left input	Right input	Left output	Right output
CLN	Any	None	Any	Any
SND_X	None	None	Integer	None

4 Examples and visual representations

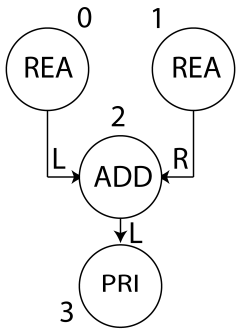
The first example is a program that represents the addition of two integers that are being read from input. The result is then printed to output.
Code:

```
0 REA 2L -
1 REA 2R -
2 ADD 3L -
3 PRI - -
```

C equivalent:

```
int a, b;
scanf("%d %d", &x, &y);
int s = a + b;
printf("%d", s);
```

Graph:



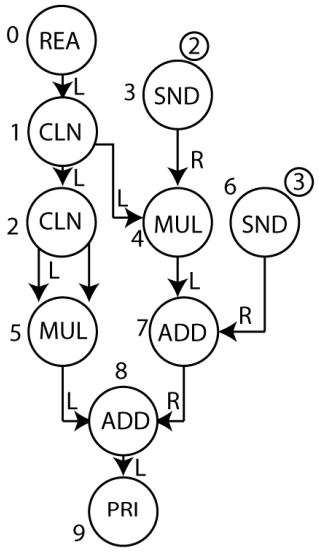
The second example shows how to use basic instructions and how they interact which each other.
Code:

```
0 REA 1L -
1 CLN 2L 4L
2 CLN 5L 5R
3 SND_2 4R -
4 MUL 7L -
5 MUL 8L -
6 SND_3 7R -
7 ADD 8R -
8 ADD 9L -
9 PRI
```

C equivalent:

```
int x;
scanf("%d", &x);
int y = x * x + 2 * x + 3;
printf("%d", y);
```

Graph:



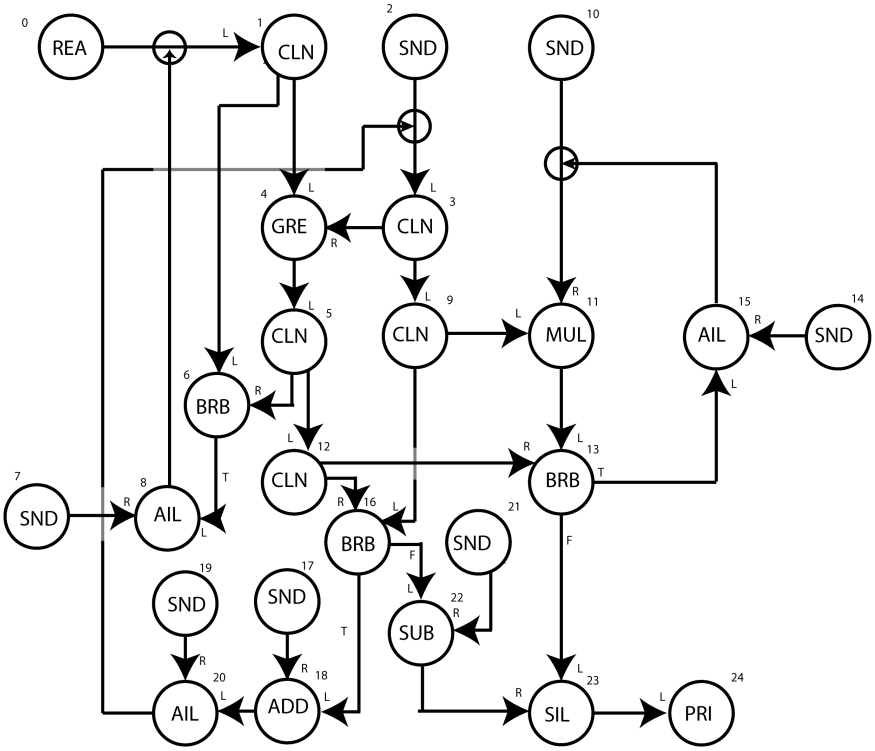
The third example illustrates a **for** loop and how iteration levels are changed. The case for nested fors is treated in a similar way.
Code:

```
0 REA 1L -
1 CLN 4L 6L
2 SND_1 3L -
3 CLN 4R 9L
4 GRE 5L -
5 CLN 6R 12L
6 BRB 8L -
7 SND_1 8R -
8 AIL 1L -
9 CLN 11L 16L
10 SND_1 11R
11 MUL 13L -
12 CLN 13R 16R
13 BRB 15L 23L
14 SND_1 15R -
15 AIL 11R -
16 BRB 18L 22L
17 SND_1 18R -
18 ADD 20L -
19 SND_1 20R -
20 AIL 3L -
21 SND_1 22R -
22 SUB 23R -
23 SIL 24L -
24 PRI - -
```

C equivalent:

```
int x,y;
scanf("%d", &x);
y=1;
for(int i = 1; i <= x; i++)
{
    y *= i;
}
printf("%d", y);
```

Graph:


0
REA
1
CLN
2
SND
10
SND
4
GRE
3
CLN
5
CLN
9
CLN
11
MUL
15
AIL
14
SND
6
BRB
12
CLN
13
BRB
7
SND
8
AIL
16
BRB
21
SND
19
SND
17
SND
22
SUB
20
AIL
18
ADD
23
SIL
24
PRI