Bytecode BrainDesigner

This simple instruction set is the default used in BrainDesigner. It is optimized to be easily converted into binary code for other platforms.

Format

All commands are in the following format:

```
command dest, src1, src2, …
```

A command performs an operation, and writes the result to `dest`. Any parameters are given after the destination.

Depending on the command the following sources and destinations are available:

- all Inputs/Parameters and Internals/Outputs of the module
- two temporary Values `V0` and `V1`

If only one input and output exist (for example for synapses), those are addressed using “Input” and “Output”.

Command Overview

`ABS`	Absolute value
`ADD`	Addition
`DIV`	Division
`LOAD`	Load a value
`MAX`	Maximum of two values
`MIN`	Minimum of two values
`MUL`	Multiplication
`SAT`	Saturation +/-1
`SUB`	Subtraction
`TANH`	Hyperbolic tangent
`WRITE`	Write to Outputs/Internals

ABS

Converts a value to its absolute value.

Format

```
ABS <src>
```

Allowed Values

```
<src>:     V0, V1
```

Sample

```
ABS V0
```

Additional Information

The destination register corresponds to the source register.

Compiling in ARM architecture

The value is compared to 0. If it is smaller, the value is negated.

Sample:

```
CMP   R8, #0          ; R8 = Source
BPL   #1              ; PL = Plus or Zero
RSB   R8, R8, #0
```

Up to 3 cycles on STM32.

ADD

Adds two values.

Format

```
ADD <dest>, <src1>, <src2>
```

Allowed Values

`<dest>`: `V0`, `V1`
`<src1>`: Inputs/Parameters, `V0`, `V1`
`<src2>`: Inputs/Parameters, `V0`, `V1`

Sample

```
ADD V0, Input, x
```

Additional Information

No additional information.

Compiling in ARM architecture

The addition is done using a single ADD instruction.

Sample:

```
ADD   R8, R8, R9
```

1 cycle on STM32.

DIV

Divides the first source value (dividend) by the second source value (divisor) and writes it to a specific location.

Format

```
DIV <dest>, <src1>, <src2>
```

Allowed Values

`<dest>`:	`V0`, `V1`
`<src1>`:	Inputs/Parameters, `V0`, `V1`
`<src2>`:	Inputs/Parameters, `V0`, `V1`

Sample

```
DIV V0, Input1, Input2
```

Additional Information

The division by zero is catched and the result is always zero.

On the ARM processor, there are additional restrictions on the range of values for the dividend: it must be in the interval [-64, +64). The result has a lower accuracy of only 10 binary digits (~ ±0,001).

Compiling in ARM architecture

The ARM processor provides a command SDIV for signed division. In order to avoid division by zero, it is checked whether the divisor is zero, and in this case the result is set to zero. In this case, the actual division section is skipped.

The SDIV command provides a 32-bit result. The result or the dividend must be shifted to the left by 15 binary digits. So accuracy is lost: either there would be only two digits for the dividend or the result would have no decimal places.

In this case a mixture is used: The dividend is shifted to the left by 10 bits which results in a smaller range of values in the interval [-64, +64). The re-

sult is shifted by the missing 5 bits, so the the number of decimal places is reduced to 10 bit.

Sample:

```
CMP   R1, #0        ; R1 = Source 2 (Divisor)
BNE   #2
MOVW  R8, #0        ; R8 = Destination
B     #5
LSL   R0, R0, #10   ; R0 = Source 1 (Dividend)
SDIV  R8, R0, R1
LSL   R8, R8, #5
```

Up to 18 cycles on STM32.

LOAD

Loads a value to a register.

Format

```
LOAD <dest>, <value>
```

Allowed Values

`<dest>`: V0, V1

`<value>`: double value, Internals/Outputs, Inputs/Parameters

Sample

```
LOAD V0, 0.25
LOAD V0, Internal1
LOAD V0, Input1
```

Additional Information

On the ARM processor fixed point numbers with 15 decimal place are used. The accuracy is therefore $1/2^{15} \approx 0{,}00003$.

Compiling in ARM architecture

The fixed-point value is converted to 32 bit (17 integer places, 15 decimal places) and is loaded to the desired register using MOVW and MOVT.

Sample:

```
MOVW  R8, #0x4000
MOVT  R8, #0x0000
```

2 cycles on STM32.

For Internals/Outputs the LDR command is used:

```
LDR   R8, [R12, #1]
```

1 cycle on STM32.

MAX

Gets the greater value out of two values.

Format

```
MAX <dest>, <src1>, <src2>
```

Allowed Values

`<dest>`:	`V0`, `V1`
`<src1>`:	Inputs/Parameters, `V0`, `V1`
`<src2>`:	Inputs/Parameters, `V0`, `V1`

Sample

```
MAX V0, Input1, Input2
```

Additional Information

No additional information.

Compiling in ARM architecture

src1 and src2 are compared: if src2 is greater, it is written to the destination register, otherwise src1 is written to the destination register.

Sample:

```
CMP   R0, R1   ; R1 = Input2
B     10, #1   ; Condition 10: Greater than
               ; or equal
MOV   R8, R1   ; R1 = Input2, R8 = Destination
B     10, #0   ; Unconditional Branch
MOV   R8, R0   ; R0 = Input1
```

5 cycles on STM32.

MIN

Gets the lesser value out of two values.

Format

```
MIN <dest>, <src1>, <src2>
```

Allowed Values

`<dest>`: `V0`, `V1`
`<src1>`: Inputs/Parameters, `V0`, `V1`
`<src2>`: Inputs/Parameters, `V0`, `V1`

Sample

```
MIN V0, Input1, Input2
```

Additional Information

No additional information.

Compiling in ARM architecture

src1 and src2 are compared: if src2 is lesser, it is written to the destination register, otherwise src1 is written to the destination register.

Sample:

```
CMP   R0, R1   ; R1 = Input2
B     10, #1   ; Condition 11: Less than
MOV   R8, R1   ; R1 = Input2, R8 = Destination
B     10, #0   ; Unconditional Branch
MOV   R8, R0   ; R0 = Input1
```

5 cycles on STM32.

MUL

Multiplies two values and writes them to a specific location.

Format

```
MUL <dest>, <src1>, <src2>
```

Allowed Values

`<dest>`: `V0`, `V1`
`<src1>`: Inputs/Parameters, `V0`, `V1`
`<src2>`: Inputs/Parameters, `V0`, `V1`

Sample

```
MUL V0, Input, w
```

Additional Information

No additional information.

Compiling in ARM architecture

The multiplication is done using the command SMULL, which multiplies two 32-bit values and provides a 64-bit value as a result. This is brought back to 32 bits using LSR, LSL and ORR according to the fixed-point format (15 decimal places). In addition to the destination register (R8 for V0, R9 for V1) another destination register for the upper 32 bits is needed. For this purpose, an unused register is used. It is pushed to the stack before the operation and popped from the stack afterwards.

Sample:

```
PUSH  R9
SMULL R9, R8, R0, R1   ; R0 = Input; R1 = w
LSR   R8, R8, #15
LSL   R9, R9, #17
ORR   R8, R8, R9
POP   R9
```

Up to 10 cycles on STM32.

SAT

Saturates the specified parameter in the range +/-1.

Format

```
SAT <src>
```

Allowed Values

```
<src>:     V0, V1
```

Sample

```
SAT V0
```

Additional Information

No additional information.

Compiling in ARM architecture

The ARM saturation command SSAT is used. The immediate value is 15, therefore the value is saturated to 15 decimal places (+/-1).

Sample:

```
SSAT  R8, R8, #15
```

1 cycle on STM32.

SUB

Subtracts the second source value from the first and writes it into a destination register.

Format

```
SUB <dest>, <src1>, <src2>
```

Allowed Values

`<dest>`: `V0`, `V1`
`<src1>`: Inputs/Parameters, `V0`, `V1`
`<src2>`: Inputs/Parameters, `V0`, `V1`

Sample

```
SUB V0, Input, x
```

Additional Information

No additional information.

Compiling in ARM architecture

The addition is done using a single SUB command.

Sample:

```
SUB   R8, R8, R9
```

1 cycle on STM32.

TANH

Calculates the hyperbolic tangent. A system function is used.

Format

```
TANH <dest>, <src>
```

Allowed Values

`<dest>`: `V0`, `V1`

`<src>`: Inputs/Parameters, `V0`, `V1`

Sample

```
TANH V0, Input
```

Additional Information

No additional information.

Compiling in ARM architecture

A prepared tanh routine, which uses a lookup table, is used. The compiled code is basically a jump to this function. The value must be provided in register R0. So this register is pushed to stack and the desired input value is copied to R0 (if it isn't R0 anyway). After the branch the output value is copied to the desired register.

Sample:

```
PUSH  R0
MOV   R0, R2   ; R2 = Input
BL    tanh
MOV   R8, R0   ; R8 = V0
POP   R0
```

Up to 7 cycles on STM32 (incl. branch) + execution time of tanh function.

WRITE

Writes a value to an Output.

Format

```
WRITE <dest>, <src>
```

Allowed Values

`<dest>`:	Outputs/Internals
`<src>`:	Inputs/Parameters, `V0`, `V1`, Internals

Sample

```
WRITE Output, V0
```

Additional Information

No additional information.

Compiling in ARM architecture

It is necessary to distinguish whether the source is already in a register, or has to be temporarily stored in a register (for Internals).

If the source value is already in a register and the destination address is an offset to Register R12 (which is the start address of the own data area in RAM), the STR command is used.

Sample:

```
STR   R8, [R12, #0]
```

2 cycles on STM32.

If this is not the case, the value is loaded to register R0, which was pushed to stack before. This value is written using the STR command and R0 is popped from stack.

Sample:

```
PUSH  R0
LDR   R0, [R12, #1]
STR   R8, [R12, #0]
POP   R0
```