Bytecode BrainDesigner

Dieser einfache Befehlssatz ist der standardmäßig im BrainDesigner verwendete, der so optimiert ist, dass er auch leicht in Binärcode für andere Plattformen umgewandelt werden kann.

Format

Alle Befehle folgen dem folgenden Format:

```
command dest, src1, src2, …
```

Ein Kommando führt eine Rechenoperation aus und schreibt das Ergebnis in `dest`. Eventuelle Parameter werden dahinter angegeben.

Als Quellen und Ziele stehen dabei je nach Befehl zur Verfügung:

- alle Inputs/Parameter und Internals/Outputs des Moduls
- zwei temporäre Werte `V0` und `V1`

Gibt es nur einen In- und Output (beispielsweise bei Synapsen), werden diese mit „Input“ und „Output“ angesprochen.

Befehlsübersicht

`ABS`	Absolutwert
`ADD`	Addition
`DIV`	Division
`LOAD`	Laden eines festen Wertes bzw. Internals
`MAX`	Größerer von zwei Werten
`MIN`	Kleinerer von zwei Werten
`MUL`	Multiplikation
`SAT`	Sättigung +/-1
`SUB`	Subtraktion
`TANH`	Tangens Hyperbolicus
`WRITE`	Schreiben in Outputs/Internals

ABS

Macht aus einem Wert einen Absolutwert.

Format

```
ABS <src>
```

Erlaubte Werte

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

Beispiel

```
ABS V0
```

Weitere Hinweise

Das Zielregister entspricht dem Quellregister.

Kompilierung ARM

Der Wert wird mit 0 verglichen. Ist er kleiner, wird der Wert negiert.

Beispiel:

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

Bis zu 3 Zyklen.

ADD

Addiert zwei Werte.

Format

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

Erlaubte Werte

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

Beispiel

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

Weitere Hinweise

keine

Kompilierung ARM

Die Addition wird mittels eines einzelnen ADD-Befehls durchgeführt.

Beispiel:

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

1 Zyklus.

DIV

Dividiert den ersten Quellwert (Dividend) durch den zweiten Quellwert (Divisor) und schreibt sie an eine bestimmte Stelle.

Format

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

Erlaubte Werte

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

`<src1>`: Inputs/Parameter, `V0`, `V1`

`<src2>`: Inputs/Parameter, `V0`, `V1`

Beispiel

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

Weitere Hinweise

Die Division durch Null wird abgefangen und das Ergebnis ist dann immer gleich Null.

Auf dem ARM-Prozessor gibt es weitere Einschränkungen für den Wertebereich des Dividenden: dieser muss im Intervall [-64, +64) liegen. Das Ergebnis hat eine geringere Nachkommagenauigkeit von 10 Binärstellen (~ ±0,001).

Kompilierung ARM

Der ARM-Prozessor liefert einen Signed-DIV-Befehl SDIV. Um eine Division durch Null zu vermeiden, wird geprüft, ob der Divisor gleich Null ist und in diesem Fall das Ergebnis auf Null gesetzt. Sollte dies der Fall sein, wird der eigentliche Divisions-Abschnitt übersprungen.

Der SDIV-Befehl liefert ein 32-Bit-Ergebnis. Dieses oder vorher der Dividend müssen bei zwei Festkommawerten mit 15 Nachkommastellen um 15 Binärstellen nach links geshiftet werden. Dadurch geht Genauigkeit verloren: entweder beim Dividenden würden nur zwei Vorkommastellen bleiben oder beim Ergebnis würde es keine Nachkommastellen geben.

Konkret wurde eine Mischung benutzt: Der Dividend wird um 10 Stellen nach links geshiftet, was den Wertebereich auf das Intervall [-64, +64) ein-

schränkt. Das Ergebnis wird zusätzlich um die fehlenden 5 Bit geshiftet, sodass die Nachkommagenauigkeit auf 10 Stellen reduziert wird.

Beispiel:

```
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
```

Bis zu 18 Zyklen.

LOAD

Lädt einen Wert in ein Register.

Format

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

Erlaubte Werte

`<dest>`: V0, V1

`<value>`: Double-Wert, Internals/Outputs, Inputs/Parameters

Beispiel

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

Weitere Hinweise

Auf dem ARM-Prozessor werden Festkommazahlen mit 15 Nachkommastellen verwendet. Die darstellbare Genauigkeit liegt also bei $1/2^{15} \approx 0{,}00003$.

Kompilierung ARM

Der in einen 32-Bit-Wert umgewandelte Festkommawert (17 Vorkommastellen, 15 Nachkommastellen) wird mittels MOVW und MOVT in das gewünschte Register geladen.

Beispiel:

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

2 Zyklen.

Für Internals/Outputs wird der LDR-Befehl verwendet:

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

1 Zyklus.

MAX

Übernimmt den größeren von zwei Werten.

Format

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

Erlaubte Werte

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

Beispiel

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

Weitere Hinweise

keine

Kompilierung ARM

Es werden src1 und src2 verglichen: ist src2 größer, wird src2 ins Zielregister geschrieben und über den letzten Befehl rüber gesprungen, ansonsten wird mit diesem letzten Befehl src1 ins Zielregister geschrieben.

Beispiel:

```
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 Zyklen.

MIN

Übernimmt den kleineren von zwei Werten.

Format

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

Erlaubte Werte

<dest>: V0, V1

<src1>: Inputs/Parameter, V0, V1

<src2>: Inputs/Parameter, V0, V1

Beispiel

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

Weitere Hinweise

keine

Kompilierung ARM

Es wird zunächst der Wert von src1 ins Zielregister geschrieben. Anschließend werden src1 und src2 verglichen: ist src2 kleiner, wird src2 ins Zielregister geschrieben, ansonsten wird über diesen Befehl gesprungen.

Beispiel:

```
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 Zyklen.

MUL

Multipliziert zwei Werte und schreibt sie an eine bestimmte Stelle.

Format

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

Erlaubte Werte

<dest>: V0, V1

<src1>: Inputs/Parameter, V0, V1

<src2>: Inputs/Parameter, V0, V1

Beispiel

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

Weitere Hinweise

keine

Kompilierung ARM

Die Multiplikation wird mittels des Befehls SMULL durchgeführt, der zwei 32-Bit-Werte multipliziert und ein 64-Bit-Wert als Ergebnis liefert. Dieses wird dann mittels LSR, LSL und ORR entsprechend des Festkommaformats (15 Nachkommastellen) wieder auf 32 Bit gebracht. Neben dem Zielregister (R8 für V0, R9 für V1) wird ein weiteres Zielregister für die oberen 32 Bit benötigt. Hierfür wird ein für diesen Befehl ungenutztes Register verwendet und vorher gepusht und nach der Ausführung wieder gepopt.

Beispiel:

```
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
```

Bis zu 10 Zyklen.

SAT

Sättigt den angegebenen Parameter im Bereich +/-1.

Format

```
SAT <src>
```

Erlaubte Werte

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

Beispiel

```
SAT V0
```

Weitere Hinweise

keine

Kompilierung ARM

Es wird der ARM-eigene Sättigungsbefehl SSAT genutzt, der Immediate-Wert ist dabei 15, sodass auf 15 Nachkommastellen gesättigt wird.

Beispiel:

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

1 Zyklus.

SUB

Subtrahiert den zweiten Source-Wert vom ersten und schreibt ihn in ein Zielregister.

Format

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

Erlaubte Werte

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

Beispiel

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

Weitere Hinweise

keine

Kompilierung ARM

Die Addition wird mittels eines einzelnen SUB-Befehls durchgeführt.

Beispiel:

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

1 Zyklus.

TANH

Berechnet den Tangens Hyperbolicus. Es wird jeweils eine vom System bereitgestellte Funktion genutzt.

Format

`TANH <dest>, <src>`

Erlaubte Werte

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

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

Beispiel

```
TANH V0, Input
```

Weitere Hinweise

keine

Kompilierung ARM

Es wird eine vorbereitete Tanh-Routine mit Lookup-Tabelle verwendet. Der kompilierte Code besteht im Grunde aus einem Sprung zu dieser Funktion. Der Wert muss in Register R0 bereitgestellt werden, weshalb dieses Register zunächst gepusht (und am Ende gepopt wird) und der gewünschte Eingabewert nach R0 kopiert wird (wenn es nicht sowieso R0 ist). Nach der Funktion wird der Wert zum gewünschten Output kopiert. Ist der Output sowieso R0, muss nicht kopiert werden, R0 wird in diesem Fall des Weiteren nicht gepusht/gepopt.

Beispiel:

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

Bis zu 7 Zyklen (inkl. Sprung) + Ausführungszeit des Tanh.

WRITE

Schreibt einen Wert in ein Output.

Format

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

Erlaubte Werte

`<dest>`: Outputs/Internals

`<src>`: Inputs/Parameter, `V0`, `V1`, Internals

Beispiel

```
WRITE Output, V0
```

Weitere Hinweise

keine

Kompilierung ARM

Es ist zu unterscheiden, ob die Quelle sich bereits in einem Register befindet, oder erst in ein Register zwischengespeichert werden muss (bei Internals).

Befindet sich der zu schreibende Wert bereits in einem Register und die Zieladresse ist ein Offset zu Register R12 (Start des eigenen Datenbereichs im RAM), dann wird der STR-Befehl genutzt.

Beispiel:

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

2 Zyklen.

Ist dies nicht der Fall, wird zunächst der Wert gelesen und in Register R0 zwischengespeichert, das vorher auf den Stack gepusht wurde. Dieser wird dann mittels des STR-Befehls geschrieben und R0 wieder gepopt.

Beispiel:

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