BioAssembly Definition

March 26, 2024

1 BioAssembly ISA

This chapter lists and explains in detail the available BioAssembly instructions.

1.1 List of instructions

This section lists all the available BioAssembly instructions. The instructions are divided into groups depending on functionality. The table reports the instruction name, a brief description, and a reference to the detailed description.

DMF electrode control instructions

Table 1.1 lists the instructions dedicated to controlling the electrodes of the the DMF platform. Even if electrodes can be considered as DMF devices and thus controlled with the instructions listed in Table 1.2, dedicated instructions are provided since electrodes are the basic and fundamental unit of actuation in DMF platforms.

Instruction	Description	Reference
SETELI	Set electrode immediate	1.2.1
CLRELI	Clear electrode immediate	1.2.2
SETEL	Set electrode	1.2.3
CLREL	Clear electrode	1.2.4
CLRALL	Clear all electrodes	1.2.5

Table 1.1: List of 'DMF electrode control' instructions.

DMF device access instructions

Table 1.2 lists the instructions used to interact with actuation and sensing devices of the DMF platform. Examples of these devices are: heaters, coolers, electro-chemical sensors, optical sensor, etc.

Instruction	Description	Reference
DEVWR	Device write	1.2.6
ADEVRD	Asynchronous device read	1.2.7
ADEVWR	Asynchronous device write	1.2.8
ADEVEX	Execute asynchronous device reads and writes	1.2.9
ADEVCL	Clear queued asynchronous device reads and writes	1.2.10

Table 1.2: List of 'DMF device access' instructions.

Task management and synchronization instructions

Table 1.3 lists the instructions for managing (starting and stopping) and synchronizing tasks.

Instruction	Description	Reference
TSTART	Start a task	1.2.11
TSTOP	Stop the current task	1.2.12
TICK	Apply changes and wait next tick (synchronization)	1.2.13
BARR	Inter-task synchronization barrier	1.2.14

Table 1.3: List of 'Task management and synchronization' instructions.

Memory access instructions

Table 1.4 lists the instructions for memory access.

Instruction	Description	Reference
LI	Load immediate	1.2.15
MOVE	Move (implements memory indirection)	1.2.16

Table 1.4: List of 'Memory access' instructions.

Arithmetic and logic instructions

Table 1.5 lists the instructions for arithmetic and logic operations on integer numbers.

Instruction	Description	Reference
ADD	Add	1.2.17
SUB	Subtract	1.2.18
MULT	Multiply	1.2.19
DIV	Divide	1.2.20
AND	Bitwise AND	1.2.21
OR	Bitwise OR	1.2.22
XOR	Bitwise XOR	1.2.23
NOT	Bitwise NOT	1.2.24
ADDI	Add immediate	1.2.25
SUBI	Subtract immediate	1.2.26
ANDI	Bitwise AND immediate	1.2.27
ORI	Bitwise OR immediate	1.2.28
XORI	Bitwise XOR immediate	1.2.29
SLL	Shift left logical	1.2.30
SRL	Shift right logical	1.2.31
SRA	Shift right arithmetic (propagate MSB)	1.2.32

Table 1.5: List of 'Arithmetic and logic' instructions.

Flow control instructions

Table 1.6 list instruction for program flow control.

Instruction	Description	Reference
JI	Jump immediate	1.2.33
J	Jump	1.2.34
JIAL	Jump immediate and link	1.2.35
BEQ	Branch if equal	1.2.36
BGE	Branch if greater or equal	1.2.37
BLE	Branch if less or equal	1.2.38

Table 1.6: List of 'Flow control' instructions.

Real numbers instructions

Table 1.7 lists the instruction that perform operations related to real numbers. These include arithmetic, memory access, comparison, and conversion instructions.

Instruction	Description	Reference
R_ADD	Add (real)	1.2.39
R_SUB	Subtract (real)	1.2.40
R_MULT	Multiply (real)	1.2.41
R_DIV	Divide (real)	1.2.42
R_NEG	Change sign (real)	1.2.43
R_ABS	Absolute (real)	1.2.44
R_CEQ	Compare equal (real)	1.2.45
R_CGE	Compare greater or equal (real)	1.2.46
R_CLE	Compare less or equal (real)	1.2.47
R_CVTI2F	Convert integer to float (real)	1.2.48
R_CVTF2I	Convert float to integer (real)	1.2.49

Table 1.7: List of 'Real numbers' instructions.

1.2 Detailed description

1.2.1 SETELI - Set electrode immediate

Syntax:

```
SETELI ELECTRODE_IMMEDIATE
```

Description:

This instruction set the electrode number specified by `ELECTRODE_IMMEDIATE`. The changes are applied at the end of the synchronization period (when the instruction `TICK` or `BARR` is executed).

1.2.2 CLRELI - Clear electrode immediate

Syntax:

```
CLRELI ELECTRODE_IMMEDIATE
```

Description:

This instruction clear the electrode number specified by ELECTRODE_IMMEDIATE. The changes are applied at the end of the synchronization period (when the instruction TICK or BARR is executed).

1.2.3 SETEL - Set electrode

Syntax:

```
SETEL ELECTRODE_POINTER
```

Description:

This instruction set the electrode number specified in the data memory address specified by ELECTRODE_POINTER. The changes are applied at the end of the synchronization period (when the instruction TICK or BARR is executed).

1.2.4 CLREL - Clear electrode

Syntax:

```
CLREL ELECTRODE_POINTER
```

Description:

This instruction clear the electrode number specified in the data memory address specified by ELECTRODE_POINTER. The changes are applied at the end of the synchronization period (when the instruction TICK or BARR is executed).

1.2.5 CLRALL - Clear all electrodes

Syntax:

```
CLRALL
```

Description:

This instruction clears all the electrodes in the DMF platform. The changes are applied at the end of the synchronization period (when the instruction TICK or BARR is executed). It is recommended to use this instruction only to bring the platform in a known state (e.g., initialization, reset after error).

1.2.6 DEVWR - Device write

Syntax:

```
DEVWR DEVICE_ADDRESS_IMMEDIATE SOURCE_POINTER
```

Description:

This instruction is used to access the device memory in a synchronous way. The content of the data memory at the address SOURCE_POINTER is copied to the device memory at the address DEVICE_ADDRESS_IMMEDIATE. The write operation happens at the end of the synchronization period (when the instruction TICK or BARR is executed).

1.2.7 ADEVRD - Asynchronous device read

Syntax:

```
ADEVRD DEVICE_ADDRESS_IMMEDIATE TARGET_POINTER
```

Description:

This instruction is used to access the device memory in an asynchronous way. The content of the device memory at the address `DEVICE_ADDRESS_IMMEDIATE` is copied to the data memory at the address `TARGET_POINTER`. The write operation happens when the instruction `ADEVEX` is executed.

1.2.8 ADEVWR - Asynchronous device write

Syntax:

```
ADEVWR DEVICE_ADDRESS_IMMEDIATE SOURCE_POINTER
```

Description:

This instruction is used to access the device memory in an asynchronous way. The content of the data memory at the address `SOURCE_POINTER` is copied to the device memory at the address `DEVICE_ADDRESS_IMMEDIATE`. The write operation happens when the instruction `ADEVEX` is executed.

1.2.9 ADEVEX - Execute asynchronous device reads and writes

Syntax:

```
ADEVEX
```

Description:

This instruction triggers the queued asynchronous read and write operations (queued by the instructions `ADEVRD` and `ADEVWR`).

1.2.10 ADEVCL - Clear queued asynchronous device reads and writes

Syntax:

```
ADEVCL
```

Description:

This instruction clears the queued asynchronous read and write operations (queued by the instructions `ADEVRD` and `ADEVWR`).

1.2.11 TSTART - Start a task

Syntax:

```
TARGET_PROGRAM_COUNTER
```

Description:

This instruction starts a parallel tasks. The parallel task execution starts from the program memory location `TARGET_PROGRAM_COUNTER`. The tasks starts in the next synchronization period.

1.2.12 TSTOP - Stop the current task

Syntax:

```
TSTOP
```

Description:

This instruction stops the current tasks (where the task is called). The task continues to execute until the instruction `TICK` or `BARR` is executed.

1.2.13 TICK - Apply changes and wait next tick (synchronization)

Syntax:

```
TICK
```

Description:

This instruction implements synchronization. The instruction is called by each task in every synchronization period to signal that execution for the current period is complete.

1.2.14 BARR - Inter-task synchronization barrier

Syntax:

```
BARR BARRIER_ID BARRIER_COUNT
```

Description:

This instruction implements and inter-task synchronization barrier. The barrier is identified by an ID specified by `BARRIER_ID`. The barrier cannot be 'traversed' until an amount of tasks, specified by `BARRIER_COUNT`, has executed the instruction (referring to the same ID). When called, the instruction also signal that execution of the executing task for the current period is complete (same as in the `TICK` instruction).

1.2.15 LI - Load immediate

Syntax:

```
LI TARGET_POINTER OPERAND_1_IMMEDIATE
```

Description:

This instruction loads the immediate value `OPERAND_1_IMMEDIATE` into the data memory location specified by the address `TARGET_POINTER`.

1.2.16 MOVE - Move (implements memory indirection)

Syntax:

```
MOVE TARGET_POINTER_POINTER SOURCE_POINTER_POINTER
```

Description:

This instruction implements memory indirection. It copies the content from the data memory location of which address is stored in the memory location specified by `SOURCE_POINTER_POINTER` to the data memory location of which address is stored in the memory location specified by `TARGET_POINTER_POINTER`. A C equivalent would be `**TARGET_POINTER_POINTER = **SOURCE_POINTER_POINTER;`.

1.2.17 ADD - Add

Syntax:

```
ADD RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs integer addition. The result is stored in the data memory location specified by `RESULT_POINTER`. The first operand is loaded from the data memory location specified by `OPERAND_1_POINTER`. The second operand is loaded from the data memory location specified by `OPERAND_2_POINTER`.

1.2.18 SUB - Subtract

Syntax:

```
SUB RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs integer subtraction. The result is stored in the data memory location specified by `RESULT_POINTER`. The first operand is loaded from the data memory location specified by `OPERAND_1_POINTER`. The second operand is loaded from the data memory location specified by `OPERAND_2_POINTER`.

1.2.19 MULT - Multiply

Syntax:

```
MULT RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs integer multiplication. The result is stored in the data memory location specified by `RESULT_POINTER`. The first operand is loaded from the data memory location specified by `OPERAND_1_POINTER`. The second operand is loaded from the data memory location specified by `OPERAND_2_POINTER`.

1.2.20 DIV - Divide

Syntax:

```
DIV RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs integer division. The result is stored in the data memory location specified by `RESULT_POINTER`. The first operand is loaded from the data memory location specified by `OPERAND_1_POINTER`. The second operand is loaded from the data memory location specified by `OPERAND_2_POINTER`.

1.2.21 AND - Bitwise AND

Syntax:

```
AND RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs bitwise AND between two integers. The result is stored in the data memory location specified by `RESULT_POINTER`. The first operand is loaded from the data memory location specified by `OPERAND_1_POINTER`. The second operand is loaded from the data memory location specified by `OPERAND_2_POINTER`.

1.2.22 OR - Bitwise OR

Syntax:

```
OR RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs bitwise OR between two integers. The result is stored in the data memory location specified by `RESULT_POINTER`. The first operand is loaded from the data memory location specified by `OPERAND_1_POINTER`. The second operand is loaded from the data memory location specified by `OPERAND_2_POINTER`.

1.2.23 XOR - Bitwise XOR

Syntax:

```
XOR RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs bitwise XOR between two integers. The result is stored in the data memory location specified by `RESULT_POINTER`. The first operand is loaded from the data memory location specified by `OPERAND_1_POINTER`. The second operand is loaded from the data memory location specified by `OPERAND_2_POINTER`.

1.2.24 NOT - Bitwise NOT

Syntax:

```
NOT RESULT_POINTER OPERAND_1_POINTER
```

Description:

This instruction performs bitwise NOT of an integer. The result is stored in the data memory location specified by `RESULT_POINTER`. The operand is loaded from the data memory location specified by `OPERAND_1_POINTER`.

1.2.25 ADDI - Add immediate

Syntax:

```
ADDI RESULT_POINTER OPERAND_1_POINTER OPERAND_2_IMMEDIATE
```

Description:

This instruction performs integer addition. The result is stored in the data memory location specified by `RESULT_POINTER`. The first operand is loaded from the data memory location specified by `OPERAND_1_POINTER`. The second operand is an immediate specified by `OPERAND_2_IMMEDIATE`.

1.2.26 SUBI - Subtract immediate

Syntax:

```
SUBI RESULT_POINTER OPERAND_1_POINTER OPERAND_2_IMMEDIATE
```

Description:

This instruction performs integer subtraction. The result is stored in the data memory location specified by RESULT_POINTER. The first operand is loaded from the data memory location specified by OPERAND_1_POINTER. The second operand is an immediate specified by OPERAND_2_IMMEDIATE.

1.2.27 ANDI - Bitwise AND immediate

Syntax:

```
ANDI RESULT_POINTER OPERAND_1_POINTER OPERAND_2_IMMEDIATE
```

Description:

This instruction performs bitwise and between two integers. The result is stored in the data memory location specified by RESULT_POINTER. The first operand is loaded from the data memory location specified by OPERAND_1_POINTER. The second operand is an immediate specified by OPERAND_2_IMMEDIATE.

1.2.28 ORI - Bitwise OR immediate

Syntax:

```
ORI RESULT_POINTER OPERAND_1_POINTER OPERAND_2_IMMEDIATE
```

Description:

This instruction performs bitwise OR between two integers. The result is stored in the data memory location specified by RESULT_POINTER. The first operand is loaded from the data memory location specified by OPERAND_1_POINTER. The second operand is an immediate specified by OPERAND_2_IMMEDIATE.

1.2.29 XORI - Bitwise XOR immediate

Syntax:

```
XORI RESULT_POINTER OPERAND_1_POINTER OPERAND_2_IMMEDIATE
```

Description:

This instruction performs bitwise XOR between two integers. The result is stored in the data memory location specified by RESULT_POINTER. The first operand is loaded from the data memory location specified by OPERAND_1_POINTER. The second operand is an immediate specified by OPERAND_2_IMMEDIATE.

1.2.30 SLL - Shift left logical

Syntax:

```
SLL RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs shift left of an integer. When shifting, the new bits are 0. The result is stored in the data memory location specified by RESULT_POINTER. The operand to be shifted is loaded from the data memory location specified by OPERAND_1_POINTER. The shift amount is loaded from the data memory location specified by OPERAND_2_POINTER.

1.2.31 SRL - Shift right logical

Syntax:

```
SRL RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs shift right of an integer. When shifting, the new bits are 0. The result is stored in the data memory location specified by `RESULT_POINTER`. The operand to be shifted is loaded from the data memory location specified by `OPERAND_1_POINTER`. The shift amount is loaded from the data memory location specified by `OPERAND_2_POINTER`.

1.2.32 SRA - Shift right arithmetic (propagate MSB)

Syntax:

```
SRA RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs shift right of an integer. When shifting, the new bits match the most significant bit (MSB). In other words, the MSB is propagated. The result is stored in the data memory location specified by `RESULT_POINTER`. The operand to be shifted is loaded from the data memory location specified by `OPERAND_1_POINTER`. The shift amount is loaded from the data memory location specified by `OPERAND_2_POINTER`.

1.2.33 JI - Jump immediate

Syntax:

```
JI TARGET_PROGRAM_COUNTER_IMMEDIATE
```

Description:

This instruction perform the unconditional branch to the target program counter specified by `TARGET_PROGRAM_COUNTER_IMMEDIATE`.

1.2.34 J - Jump

Syntax:

```
J TARGET_PROGRAM_COUNTER_POINTER
```

Description:

This instruction perform the unconditional branch to the target program counter specified by the content of the memory location specified by `TARGET_PROGRAM_COUNTER_POINTER`.

1.2.35 JIAL - Jump immediate and link

Syntax:

```
JIAL TARGET_PROGRAM_COUNTER_IMMEDIATE LINK_LOCATION_POINTER
```

Description:

This instruction perform the unconditional branch to the target program counter specified by TARGET_PROGRAM_COUNTER_IMMEDIATE. The program counter of the instruction right after JIAL is stored in the memory location specified by LINK_LOCATION_POINTER. This instruction is used in combination with J to return form routines.

1.2.36 BEQ - Branch if equal

Syntax:

```
BEQ TARGET_PROGRAM_COUNTER_IMMEDIATE OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction perform the conditional branch to the target program counter specified by TARGET_PROGRAM_COUNTER_IMMEDIATE if the first integer operand loaded from the data memory location specified by OPERAND_1_POINTER is equal to the second integer operand loaded from the data memory location specified by OPERAND_2_POINTER. If the condition is false, execution continues without any branch.

1.2.37 BGE - Branch if greater or equal

Syntax:

```
BGE TARGET_PROGRAM_COUNTER_IMMEDIATE OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction perform the conditional branch to the target program counter specified by TARGET_PROGRAM_COUNTER_IMMEDIATE if the first integer operand loaded from the data memory location specified by OPERAND_1_POINTER is greater or equal to the second integer operand loaded from the data memory location specified by OPERAND_2_POINTER. If the condition is false, execution continues without any branch.

1.2.38 BLE - Branch if less or equal

Syntax:

```
BLE TARGET_PROGRAM_COUNTER_IMMEDIATE OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction perform the conditional branch to the target program counter specified by TARGET_PROGRAM_COUNTER_IMMEDIATE if the first integer operand loaded from the data memory location specified by OPERAND_1_POINTER is less or equal to the second integer operand loaded from the data memory location specified by OPERAND_2_POINTER. If the condition is false, execution continues without any branch.

1.2.39 R_ADD - Add (real)

Syntax:

```
R_ADD RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs addition between real numbers. The result is stored in the data memory location specified by RESULT_POINTER. The first operand is loaded from the data memory location specified by OPERAND_1_POINTER. The second operand is loaded from the data memory location specified by OPERAND_2_POINTER.

1.2.40 R_SUB - Subtract (real)

Syntax:

```
R_SUB RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs subtraction between real numbers. The result is stored in the data memory location specified by `RESULT_POINTER`. The first operand is loaded from the data memory location specified by `OPERAND_1_POINTER`. The second operand is loaded from the data memory location specified by `OPERAND_2_POINTER`.

1.2.41 R_MULT - Multiply (real)

Syntax:

```
R_MULT RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs multiplication between real numbers. The result is stored in the data memory location specified by `RESULT_POINTER`. The first operand is loaded from the data memory location specified by `OPERAND_1_POINTER`. The second operand is loaded from the data memory location specified by `OPERAND_2_POINTER`.

1.2.42 R_DIV - Divide (real)

Syntax:

```
R_DIV RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction performs division between real numbers. The result is stored in the data memory location specified by `RESULT_POINTER`. The first operand is loaded from the data memory location specified by `OPERAND_1_POINTER`. The second operand is loaded from the data memory location specified by `OPERAND_2_POINTER`.

1.2.43 R_NEG - Change sign (real)

Syntax:

```
R_NEG RESULT_POINTER OPERAND_1_POINTER
```

Description:

This instruction performs sign inversion on a real number. The result is stored in the data memory location specified by `RESULT_POINTER`. The operand is loaded from the data memory location specified by `OPERAND_1_POINTER`.

1.2.44 R_ABS - Absolute (real)

Syntax:

```
R_ABS RESULT_POINTER OPERAND_1_POINTER
```

Description:

This instruction performs the absolute value operation on a real number. The result is stored in the data memory location specified by RESULT_POINTER. The operand is loaded from the data memory location specified by OPERAND_1_POINTER.

1.2.45 R_CEQ - Compare equal (real)

Syntax:

```
R_CEQ RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction perform the comparison between two real number operands. The result of the comparison is stored in the data memory location specified by RESULT_POINTER. If the first operand loaded from the data memory location specified by OPERAND_1_POINTER is equal to the second operand loaded from the data memory location specified by OPERAND_2_POINTER, the result is 1. Otherwise, the result is 0. This instruction can be used in combination with BEQ to perform conditional branches using real numbers as operands.

1.2.46 R_CGE - Compare greater or equal (real)

Syntax:

```
R_CGE RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction perform the comparison between two real number operands. The result of the comparison is stored in the data memory location specified by RESULT_POINTER. If the first operand loaded from the data memory location specified by OPERAND_1_POINTER is greater or equal to the second operand loaded from the data memory location specified by OPERAND_2_POINTER, the result is 1. Otherwise, the result is 0. This instruction can be used in combination with BEQ to perform conditional branches using real numbers as operands.

1.2.47 R_CLE - Compare less or equal (real)

Syntax:

```
R_CLE RESULT_POINTER OPERAND_1_POINTER OPERAND_2_POINTER
```

Description:

This instruction perform the comparison between two real number operands. The result of the comparison is stored in the data memory location specified by RESULT_POINTER. If the first operand loaded from the data memory location specified by OPERAND_1_POINTER is less or equal to the second operand loaded from the data memory location specified by OPERAND_2_POINTER, the result is 1. Otherwise, the result is 0. This instruction can be used in combination with BEQ to perform conditional branches using real numbers as operands.

1.2.48 R_CVTI2F - Convert integer to real

Syntax:

```
R_CVTI2F RESULT_POINTER OPERAND_1_POINTER
```

Description:

This instruction performs conversion of a number from the integer representation to the real representation. The value of the number is maintained during conversion (only the representation changes). The result (real) is stored in the data memory location specified by `RESULT_POINTER`. The operand (integer) is loaded from the data memory location specified by `OPERAND_1_POINTER`.

1.2.49 R_CVTF2I - Convert real to integer

Syntax:

```
R_CVTF2I RESULT_POINTER OPERAND_1_POINTER
```

Description:

This instruction performs conversion of a number from the real representation to the integer representation. The value of the number truncated during conversion. Thus, only the integer part is kept. The result (integer) is stored in the data memory location specified by `RESULT_POINTER`. The operand (real) is loaded from the data memory location specified by `OPERAND_1_POINTER`.