CPU ISA Manual

Revision 0.5

This CPU uses 16-bit instructions and 32-bit registers. There are 16 registers in the CPU. Additional registers in the systolic array and the vector processing unit can be addressed by the extended register address in certain instructions. The data format is little endian.

Register Map:

Register #	Usage	Comments
R0	Accumulator	Result of 2-input operations
R1	General purpose	
R2	General purpose	
R3	General purpose	
R4	General purpose	
R5	General purpose	
R6	General purpose	
R7	General purpose	
R8	General purpose	
R9	General purpose	
R10	General purpose	
R11	General purpose	
R12	Wide shift source L	
R13	Wide shift source H	
R14	Shuffle / wide shift destination L	
R15	Shuffle / wide shift destination H	
R16..R31	Systolic array input buffer (top)	
R32..R47	Systolic array input buffer (left)	Horizontal word
R48..R63	Systolic array input buffer (left)	Vertical word (for matrix transposing)
~~R64..R79~~	~~Systolic array output buffer~~	
R80	Systolic array control register	Enable, valid, matrix_size
R81	Systolic array data address	
R82	Systolic array weight address	
R83	Systolic array save address	
R84	Systolic array ReLU upper limit	
R96..R115	VPU input buffer	
R116	VPU control register	

R117	VPU data address	
R118	VPU weight address	
R119	VPU save address	
R120	VPU ReLU upper limit	

Types of Instructions:

R-type:

Field	OP code	RS1	func	RS2
# of bits	5	4	3	3

I-type:

Field	OP code	Immediate
# of bits	4	12

M-type:

Field	OP code	R / Immediate	RE
# of bits	5	4	7

M-type instructions are used in memory operations. RE is the extended register address. Depending on the instruction, data may be moved to R or RE.

Instruction Descriptions:

SET is an M-type instruction, it sets the address specified in RE with the 4-bit unsigned immediate value.

```
Example:    SET R0 0b0101       // sets R0 to 0101 binary
```

MOV32 is an M-type instruction, it moves 32 bits of data between registers.

```
Example:    MOV32 R0 R1         // moves 32-bit data from R0 to R1
```

MOV64O is an M-type instruction, it moves data from a non-extended register to an extended register. If the extended register is a data buffer register, 64 bits of data is moved. Otherwise, 32 bits of data is moved.

```
Example:    MOV64O R0 R32       // moves 64-bit data {R1, R0} to {R33, R32}
Example:    MOV64O R0 R80       // moves 32-bit data from R0 to R80
```

MOV64I is an M-type instruction, it moves data from an extended register to a non-extended register. If the extended register is a data buffer register, 64 bits of data is moved. Otherwise, 32 bits of data is moved.

```
Example:    MOV64I R0 R32       // moves 64-bit data {R33, R32} to {R1, R0}
```

ADDI is an I-type instruction, it adds a signed 12-bit immediate to the accumulator (R0).

```
Example:    ADDI 0xfff          // adds -1 decimal to R0
```

ADDIU is an I-type instruction, it adds an unsigned 12-bit immediate to the accumulator (R0).

Example: ADDIU 0xfff // adds 4095 decimal to R0

ADDIR is an M-type instruction, it adds an signed 7-bit immediate to the value in RS1 and store the result in R0.
Example: ADDIR R1 -1 // adds -1 to R1, store in R0

ADD RS1 + RS2 → R0
SUB RS1 – RS2 → R0
AND RS1 & RS2 → R0
OR RS1 | RS2 → R0
XOR RS1 ^ RS2 → R0
Example: SUB R2 R1 // perform R2 – R1, store result in R0

LS RS1 << RS2 → R0, negative RS2 value means right shift. Range of RS2 value is -32 to +31.
Example: LS R2 R1 // suppose R1 = 8, shifts R2 left by 8 bits, store result in R0
AS RS1 <<< RS2 → R0, negative RS2 value means right shift. Range of RS2 value is -32 to +31.
Example: AS R2 R1 // suppose R1 = -8, shifts R2 right by 8 bits while preserving the sign bit, store result in R0

JMP is an I-type instruction, it jumps unconditionally to the address specified by the 12-bit immediate.
Example: JMP 0x010 // jumps unconditionally to address 010 hexadecimal in instruction memory

JZ is an I-type instruction, if R0 is 0, then it jumps to the address specified by the 12-bit immediate.
Example: JZ 0x010 // if R0 is 0, then jumps to address 010 hexadecimal in instruction memory

JNZ is an I-type instruction, if R0 is not 0, then it jumps to the address specified by the 12-bit immediate.
Example: JZ 0x010 // if R0 is not 0, then jumps to address 010 hexadecimal in instruction memory

JNEG is an I-type instruction, if R0 < 0, then it jumps to the address specified by the 12-bit immediate.
Example: JZ 0x010 // if R0 < 0, then jumps to address 010 hexadecimal in instruction memory

JREG is an M-type instruction, it jumps unconditionally to the address specified in R.
Example: JREG R1 // jumps unconditionally to the address stored in R1

JAL is an I-type instruction, it jumps unconditionally to the address specified by the 12-bit immediate. It will also store the PC before jumping into R1.
Example: JAL 0x010 // jumps unconditionally to address 010 hexadecimal in instruction memory, store current PC in R1

SHFL (shuffle) is an R-type instruction. It shuffles the data in {RS1+1, RS1} based on the setting in RS2. The shuffled data is stored in {R15, R14}. The source data is indexed as the following table.

Byte idx	7	6	5	4	3	2	1	0
Bit idx	(RS1+1) [31:24]	(RS1+1) [23:16]	(RS1+1) [15:8]	(RS1+1) [7:0]	(RS1) [31:24]	(RS1) [23:16]	(RS1) [15:8]	(RS1) [7:0]

Each byte in {R15, R14} is controlled by 4 bits in RS1 {src_idx (3-bit), enable (1-bit)}. The i^{th} byte in {R15, R14} is controlled by bits [4i+3 : 4i] in RS2. For example, the byte of R15[23:16] is controlled by RS2[27:24].

Example: SHFL R12 R11 // shuffles {R13, R12} based on the settings in R11, store result in {R15, R14}

WS is an R-type instruction. It shift data by bytes in {R15..R12} by the number of bytes specified in RS1. The valid range of RS1 is -8 to +7. The remaining bits in the registers are filled with 0. The registers are indexed as the following table.

Byte idx	15	14	13	12	11	10	9	8
Bit idx	(R15) [31:24]	(R15) [23:16]	(R15) [15:8]	(R15) [7:0]	(R14) [31:24]	(R14) [23:16]	(R14) [15:8]	(R14) [7:0]
Byte idx	7	6	5	4	3	2	1	0
Bit idx	(R13) [31:24]	(R13) [23:16]	(R13) [15:8]	(R13) [7:0]	(R12) [31:24]	(R12) [23:16]	(R12) [15:8]	(R12) [7:0]

Example: WS R0 // suppose R0 = 3, shifts {R15..R12} left by 3 bytes

WAIT is an R-type instruction. It stalls the CPU and waits for peripheral's done signal.

Peripheral	func
Systolic Array	0b001
Vector Processing Unit	0b010

Example: WAIT 0b001 // wait for systolic array

L32 is an M-type instruction. It loads 32 bits of data using three cycles from the instruction memory. The data should be stored immediately after the instruction.

Example: L32 R2 0xffffffff // loads 0xffffffff into R2. The constant 0xffffffff will be stored into the next two instruction addresses by the assembler.

L64 is an M-type instruction. It loads 64 bits of data using three cycles from the data memory.

Example: L64 R2 R8 // loads the 64-bit data in the data memory from the address stored in R2 into {R9, R8}

S64 is an M-type instruction. It stores 64 bits of data to the data memory.

Example: S64 R2 R8 // stores 64-bit data in {R9,R8} into the data memory at the address stored in R2

Assembler functions:

Numeric literals. The assembler supports binary, decimal and hexadecimal numbers. Binary numbers should start with “0b”, hexadecimal numbers should start with “0x” and decimal numbers should not have any prefix. Binary and hexadecimal literals are viewed as unsigned numbers.

Comments. The syntax for comment is “//” as in C language.

Labels. The syntax for labels must end with “:”. The string of the label must start with a letter.

Instruction	Source Reg	Dest Reg	Type	Description	OP code	Func	Cycle	Note
SET	no	any	M-type	Set register value to immediate (4-bit)	“01000”		1	
MOV32	one	any	M-type	Move 32-bit from register to register	“01001”		1	
MOV64O	one	any extended	M-type	Move 64-bit from register to extended register	“01011”		1	Destination register updates one more cycle later
MOV64I	one extended	any	M-type	Move 64-bit from extended register to register	“01010”		2	
ADDI	R0	R0	I-type	Add R0 with immediate (12-bit) and store in R0	“1111”		1	
ADDIU	R0	R0	I-type	Add R0 with unsigned immediate (12-bit) and store in R0	“1101”		1	
ADDIR	one	R0	M-type	Add RS1 with signed immediate (7-bit) and store in R0	“00001”		1	
ADD	two	R0	R-type	Add two registers and store in R0	“00010”	“000”	1	
SUB	two	R0	R-type	Subtract two registers and store in R0	“00010”	“001”	1	
AND	two	R0	R-type	Bitwise AND two registers and store in R0	“00010”	“010”	1	
OR	two	R0	R-type	Bitwise OR two registers and store in R0	“00010”	“011”	1	
XOR	two	R0	R-type	Bitwise XOR two registers and store in R0	“00010”	“100”	1	
LS	two	R0	R-type	Logic shift	“00011”	“000”	1	
AS	two	R0	R-type	Arithmetic shift	“00011”	“001”	1	
JMP	no		I-type	Unconditional jump	“1000”		2	
JZ	R0		I-type	Jump if R0 is 0	“1010”		2	If jump is not taken, it only uses 1 cycle.
JNZ	R0		I-type	Jump if R0 is not 0	“1110”		2	If jump is not taken, it only uses 1 cycle.
JNEG	R0		I-type	Jump if R0 < 0	“1011”		2	If jump is not taken, it only uses 1 cycle.
JREG	one		M-type	Jump to the instruction address specified by the register	“01111”		2	
JAL	no	R1	I-type	Unconditional jump, store original PC in R1	“1001”		2	
SHFL	two	fixed	R-type	Shuffle 64-bit data from {RS1, RS1+1} to {R14, R15} depending on setting in RS2	“00100”	not used	1	
WS	one	fixed	R-type	Shift left 128-bit data in {R12,..,R15}	“00110”	not used	1	
WAIT	no	no	R-type	Wait for done signals from peripherals	“00101”	MUX input	variable	
L64	two	any extended	M-type	Load 64-bit data from data memory	“01100”		3	writes to two consecutive registers at once
L32	no	any extended	M-type	Load 32-bit data from instruction memory	“01101”		3	data should be stored in the next two addresses
S64	two	any extended	M-type	Store 64-bit data to data memory	“01110”		1	Destination memory updates one more cycle later

VPU & SYSARR control

VPU arguments

mode	ch_in	dim	stride	other (tbd)
4-bits	8-bits	8-bits	2-bits	10-bits

mode

deactive	DW4	DW8	AVG	FC	BP
0000	0100	0101	0001	0010	0011

ch_in & dim & stride

- unsigned binary for value

SYSARR arguments

ch_in	dim	ch_out	activate	other (tbd)
8-bits	8-bits	8-bits	1-bit	7-bits

activate

- val = 0 -> deactivate
- val = 1 -> activate

ch_in & dim & ch_out

unsigned binary for value

process image 0

DW-0

```
L32 R5 0x0000E100   // R106 DW-0 weight
L32 R6 0x0000E333   // R107 output 0
MOV640 R5 R106      // setup weight_addr_head and save_addr_head

L32 R5 0x40460800   // R104 VPU setting for DW-0
L32 R6 0x00000000   // R105 image 0
MOV640 R5 R104      // setup data_addr_head and VPU arguments
```

```
L32 R5 0x00000000   // R104 VPU setting for disable
MOV640 R5 R104      // disable VPU
```

`control arguments: 0x40460800`

mode	ch_in	dim	stride	other (tbd)
0100	0000 0100	0110 0000	10	00 0000 0000
DW4	4	96	2	-

PW-0

```
L32 R5 0x0000E10A   // R82 PW-0 weight
L32 R6 0x0000ECF7   // R83 output 1
MOV640 R5 R82       // write weight_addr_head and save_addr_head

L32 R5 0x03300880   // R80 SYSARR setting for PW-0
L32 R6 0x0000E333   // R81 output 0
MOV640 R5 R80       // write data_addr_head and SYSARR arguments

L32 R5 0x00000000   // R80 SYSARR setting for disable
MOV640 R5 R80       // disable SYSARR
```

`control arguments: 0x03300880`

ch_in	dim	ch_out	activate	other (tbd)
0000 0011	0011 0000	0000 1000	1	000 0000
3	48	8	activate	-