

# CPU ISA Green Card v5

## Memory

- **IMEM:** 4 KiB, half-word addressable (12-bit address (32-bit with extension))
- **DMEM:** 4 KiB, byte-addressable (12-bit address (32-bit with extension))
- **Total Memory:** 4 GiB, byte-addressable (32-bit address)

## Instruction Formats

### 16-bit Format

00	Opcode	R1	R2
15 14 13	10 9	5 4	0

### 32-bit Immediate Format

01	Opcode	Rd	R1	Immediate
31 30 29	26 25	21 20	16 15	0

### 32-bit Register Format

01	Opcode	Rd	R1	R2	Family	Function
31 30 29	26 25	21 20	16 15	11 10	9 8	0

### 32-bit Register Format: Mult/Div

01	13 <sub>hex</sub>	Rd1	R1	R2	10	Rd2	Func
31 30 29	26 25	21 20	16 15	11 10 9 8	4 3 0		

## Format Abbreviations

Format	Abbreviation
16-bit	R1
32-bit Immediate	I
32-bit Register	R2
Pseudoinstruction	P

## Register Name, Number, and Use

Name	Number	Use
Z	0	Fixed Constant 0
A	1	Assembler Temporary
R2-R30	2-30	General Purpose Registers
RA	31	Return Address

## Flags

Code	Meaning	Setting Insts
Z	Zero	Add/Sub/Cmp
C	Carry	Add/Sub/Cmp
N	Negative (sign bit)	Add/Sub/Cmp
O	Overflow	Add/Sub/Cmp/Mult/Div
D	Division by Zero	Div

## Branch to Flag

Branch	Flag Check
Equal	Z=1
Not Equal	Z=0
Greater Than	Z=0 and N=0
Greater Than or Equal	N=0
Greater Than Unsigned	Z=0 and C=1
Greater Than or Equal Unsigned	C=1

## Short Forms in Instruction Set

Short Form	Description
PC	Program Counter
Imm	Immediate
SignExtImm	{16{Immediate[15]}, Immediate}
ZeroExtImm	{16'b0, Immediate}
JMPAddr	{PC[31:17], Immediate, 1'b0}
NextInst	PC for next instruction
OpX	Imm, M[Reg], M[Reg + Imm] or R[Reg]

## Memory Format

Format Rule	Setting
Endianness	Little Endian
Bit Alignment	None

## Instruction Set

### Load/Store

Instruction	Mnemonic	Format	Operation (in Verilog)	Assembly	Opcode(hex)
Load Immediate	LDM	I	$R[Rd] = \text{SignExtImm}$	LDM Rd, Imm	0
Load Immediate Upper	LDMU	I	$R[Rd] = \text{Immediate} << 16$	LDMU Rd, Imm	1
Load Word Direct	LDW(D)	I	$R[Rd] = M[R1]$	LDW Rd, [R1]	2
Load Word Indexed	LDW(X)	I	$R[Rd] = M[R1 + \text{SignExtImm}]$	LDW Rd, Imm[R1]	2
Load Half-Word Direct	LDH(D)	I	$R[Rd] = \{16'b0, M[R1](15:0)\}$	LDH Rd, [R1]	3
Load Half-Word Indexed	LDH(X)	I	$R[Rd] = \{16'b0, M[R1 + \text{SignExtImm}](15:0)\}$	LDH Rd, Imm[R1]	3
Load Byte Direct	LDB(D)	I	$R[Rd] = \{24'b0, M[R1](7:0)\}$	LDB Rd, [R1]	4
Load Byte Indexed	LDB(X)	I	$R[Rd] = \{24'b0, M[R1 + \text{SignExtImm}](7:0)\}$	LDB Rd, Imm[R1]	4
Store Word Direct	STW(D)	I	$M[R1] = R[Rd]$	STW Rd, [R1]	5
Store Word Indexed	STW(X)	I	$M[R1 + \text{SignExtImm}] = R[Rd]$	STW Rd, Imm[R1]	5
Store Half-Word Direct	STH(D)	I	$M[R1] = R[Rd](15:0)$	STH Rd, [R1]	6
Store Half-Word Indexed	STH(X)	I	$M[R1 + \text{SignExtImm}] = R[Rd](15:0)$	STH Rd, Imm[R1]	6
Store Byte Direct	STB(D)	I	$M[R1] = R[Rd](7:0)$	STB Rd, [R1]	7
Store Byte Indexed	STB(X)	I	$M[R1 + \text{SignExtImm}] = R[Rd](7:0)$	STB Rd, Imm[R1]	7
Load Jump Address	LDJ	I	$R[Rd] = \text{JMPAddr}$	LDJ Rd, Imm	12

### Integer ALU

Instruction	Mnemonic	Format	Operation (in Verilog)	Assembly	Opcode/Family/ Func(hex)
Add Immediate	ADD(M)	P	$R[Rd] = R[R1] + \text{SignExtImm}$	ADD Rd, R1, Imm	–
Add Direct	ADD(D)	R2	$R[Rd] = R[R1] + M[R2]$	ADD Rd, R1, [R2]	13/0/0
Add Indexed	ADD(X)	P	$R[Rd] = R[R1] + M[R2 + \text{SignExtImm}]$	ADD Rd, R1, Imm[R2]	–
Add Register (destructive)	ADD(R)	R1	$R[R1] = R[R1] + R[R2]$	ADD R1, R2	8
Add Register (non-destructive)	ADD(R)	R2	$R[Rd] = R[R1] + R[R2]$	ADD Rd, R1, R2	13/0/1
Add PC	ADDPC	R2	$R[Rd] = PC + R[R1]$	ADDPC Rd, R1	13/3/3
Subtract Immediate	SUB(M)	P	$R[Rd] = R[R1] - \text{SignExtImm}$	SUB Rd, R1, Imm	–
Subtract Direct	SUB(D)	R2	$R[Rd] = R[R1] - M[R2]$	SUB Rd, R1, [R2]	13/0/2
Subtract Indexed	SUB(X)	P	$R[Rd] = R[R1] - M[R2 + \text{SignExtImm}]$	SUB Rd, R1, Imm[R2]	–
Subtract Register (destructive)	SUB(R)	R1	$R[R1] = R[R1] - R[R2]$	SUB R1, R2	9
Subtract Register (non-destructive)	SUB(R)	R2	$R[Rd] = R[R1] - R[R2]$	SUB Rd, R1, R2	13/0/3

## Shift and Logical

Instruction	Mnemonic	Format	Operation (in Verilog)	Assembly	Opcode/Family/ Func(hex)
Not Register	NOT	R1	$R[R1] = \sim R[R2]$	NOT R1, R2	7
And Immediate	AND(M)	P	$R[Rd] = R[R1] \& \text{ZeroExtImm}$	AND Rd, R1, Imm	–
And Direct	AND(D)	R2	$R[Rd] = R[R1] \& M[R2]$	AND Rd, R1, [R2]	13/1/0
And Indexed	AND(X)	P	$R[Rd] = R[R1] \& M[R2 + \text{ZeroExtImm}]$	AND Rd, R1, Imm[R2]	–
And Register	AND(R)	R2	$R[Rd] = R[R1] \& R[R2]$	AND Rd, R1, R2	13/1/1
Or Immediate	OR(M)	I	$R[Rd] = R[R1]   \text{ZeroExtImm}$	OR Rd, R1, Imm	8
Or Direct	OR(D)	R2	$R[Rd] = R[R1]   M[R2]$	OR Rd, R1, [R2]	13/1/2
Or Indexed	OR(X)	P	$R[Rd] = R[R1]   M[R2 + \text{ZeroExtImm}]$	OR Rd, R1, Imm[R2]	–
Or Register	OR(R)	R2	$R[Rd] = R[R1]   R[R2]$	OR Rd, R1, R2	13/1/3
Xor Immediate	XOR(M)	P	$R[Rd] = R[R1] \oplus \text{ZeroExtImm}$	XOR Rd, R1, Imm	–
Xor Direct	XOR(D)	R2	$R[Rd] = R[R1] \oplus M[R2]$	XOR Rd, R1, [R2]	13/1/4
Xor Indexed	XOR(X)	P	$R[Rd] = R[R1] \oplus M[R2 + \text{ZeroExtImm}]$	XOR Rd, R1, Imm[R2]	–
Xor Register	XOR(R)	R2	$R[Rd] = R[R1] \oplus R[R2]$	XOR Rd, R1, R2	13/1/5
Arithmetic Shift Right Immediate	ASR	R2	$R[Rd] = R[R1] >>> R2$	ASR Rd, R1, R2	13/1/6
Arithmetic Shift Right Register	ASRR	R2	$R[Rd] = R[R1] >>> R[R2]$	ASRR Rd, R1, R2	13/1/7
Logical Shift Left Immediate	LSL	R2	$R[Rd] = R[R1] << R2$	LSL Rd, R1, R2	13/1/8
Logical Shift Left Register	LSLR	R2	$R[Rd] = R[R1] << R[R2]$	LSLR Rd, R1, R2	13/1/9
Logical Shift Right Immediate	LSR	R2	$R[Rd] = R[R1] >> R2$	LSR Rd, R1, R2	13/1/10
Logical Shift Right Register	LSRR	R2	$R[Rd] = R[R1] >> R[R2]$	LSRR Rd, R1, R2	13/1/11
Circular Shift Left Immediate	CSL	R2	$R[Rd] = \{R[R1], R[R1]\} >> (32 - R2)$	CSL Rd, R1, R2	13/1/12
Circular Shift Left Register	CSLR	R2	$R[Rd] = \{R[R1], R[R1]\} >> (32 - R[R2])$	CSLR Rd, R1, R2	13/1/13
Circular Shift Right Immediate	CSR	R2	$R[Rd] = \{R[R1], R[R1]\} >> R2$	CSR Rd, R1, R2	13/1/14
Circular Shift Right Register	CSRR	R2	$R[Rd] = \{R[R1], R[R1]\} >> R[R2]$	CSRR Rd, R1, R2	13/1/15

## Multiply/Divide

Instruction	Mnemonic	Format	Operation (in Verilog)	Assembly	Func(hex)
Multiply Immediate	MULT(M)	P	$\{R[Rd2], R[Rd1]\} = R[R1] * \text{SignExtImm}$	MULT Rd1, Rd2, R1, Imm	–
Multiply Direct	MULT(D)	R2	$\{R[Rd2], R[Rd1]\} = R[R1] * M[R2]$	MULT Rd, R1, [R2]	0
Multiply Indexed	MULT(X)	P	$\{R[Rd2], R[Rd1]\} = R[R1] * M[R2] + \text{SignExtImm}$	MULT Rd1, Rd2, R1, Imm[R2]	–
Multiply Register	MULT(R)	R2	$\{R[Rd2], R[Rd1]\} = R[R1] * R[R2]$	MULT Rd1, Rd2, R1, R2	1
Multiply Immediate Unsigned	MULT(M)U	P	$\{R[Rd2], R[Rd1]\} = R[R1] * \text{ZeroExtImm}$	MULTU Rd1, Rd2, R1, Imm	–
Multiply Direct Unsigned	MULT(D)U	R2	$\{R[Rd2], R[Rd1]\} = R[R1] * M[R2]$	MULTU Rd1, Rd2, R1, [R2]	2
Multiply Indexed Unsigned	MULT(X)U	P	$\{R[Rd2], R[Rd1]\} = R[R1] * M[R2] + \text{SignExtImm}$	MULTU Rd1, Rd2, R1, Imm[R2]	–
Multiply Register Unsigned	MULT(R)U	R2	$\{R[Rd2], R[Rd1]\} = R[R1] * R[R2]$	MULTU Rd1, Rd2, R1, R2	3
Divide Immediate	DIV(M)	P	$R[Rd1] = R[R1] / \text{SignExtImm}$	DIV Rd1, Rd2, R1, Imm	–
Divide Direct	DIV(D)	R2	$R[Rd1] = R[R1] / M[R2]$	DIV Rd1, Rd2, R1, [R2]	4
Divide Indexed	DIV(X)	P	$R[Rd1] = R[R1] / M[R2] + \text{SignExtImm}$ $R[Rd2] = R[R1] \% M[R2] + \text{SignExtImm}$	DIV Rd1, Rd2, R1, Imm[R2]	–
Divide Register	DIV(R)	R2	$R[Rd1] = R[R1] / R[R2]$ $R[Rd2] = R[R1] \% R[R2]$	DIV Rd1, Rd2, R1, R2	5
Divide Immediate Unsigned	DIV(M)U	P	$R[Rd1] = R[R1] / \text{ZeroExtImm}$	DIVU Rd1, Rd2, R1, Imm	–
Divide Direct Unsigned	DIV(D)U	R2	$R[Rd1] = R[R1] / M[R2]$ $R[Rd2] = R[R1] \% M[R2]$	DIVU Rd1, Rd2, R1, [R2]	6
Divide Indexed Unsigned	DIV(X)U	P	$R[Rd1] = R[R1] / M[R2] + \text{SignExtImm}$ $R[Rd2] = R[R1] \% M[R2] + \text{SignExtImm}$	DIVU Rd1, Rd2, R1, Imm[R2]	–
Divide Register Unsigned	DIV(R)U	R2	$R[Rd1] = R[R1] / R[R2]$ $R[Rd2] = R[R1] \% R[R2]$	DIVU Rd1, Rd2, R1, R2	7
Mult Low Half	MULT	P	$R[Rd] = [R[R1] * \text{Op2}](31:0)$	MULT Rd, R1, Op2	–
Mult High Half	MULTH	P	$R[Rd] = [R[R1] * \text{Op2}](63:32)$	MULTH Rd, R1, Op2	–
Mult Low Half Unsigned	MULTU	P	$R[Rd] = [R[R1] * U(\text{Op2})](31:0)$	MULTU Rd, R1, Op2	–
Mult High Half Unsigned	MULTHU	P	$R[Rd] = [R[R1] * U(\text{Op2})](63:32)$	MULTHU Rd, R1, Op2	–
Div Quotient	DIV	P	$R[Rd] = R[R1] / \text{Op2}$	DIV Rd, R1, Op2	–
Div Remainder	DIVR	P	$R[Rd] = R[R1] \% \text{Op2}$	DIVR Rd, R1, Op2	–
Div Quotient Unsigned	DIVU	P	$R[Rd] = R[R1] / U(\text{Op2})$	DIVU Rd, R1, Op2	–
Div Remainder Unsigned	DIVRU	P	$R[Rd] = R[R1] \% U(\text{Op2})$	DIVRU Rd, R1, Op2	–

## Branch

Instruction	Mnemonic	Format	Operation (in Verilog)	Assembly	Opcode/Family/ Func(hex)
Compare Immediate	CMP	P	SetFlags(R[R1] - SignExtImm)	CMP R1, Imm	–
Compare Direct	CMD	R2	SetFlags(R[R1] - M[R2])	CMP R1, [R2]	13/3/2
Compare Indexed	CMX	P	SetFlags(R[R1] - M[R2 + SignExtImm])	CMP R1, Imm[R2]	–
Compare Register	CMPR	R1	SetFlags(R[R1] - R[R2])	CMP R1, R2	6
Jump	JMP	P	PC = JMPAddr	JMP Immediate	–
Jump Register	JMPR	R1	PC = R[R1]	JMPR R1	10
Jump and Link	JAL	P	PC = JMPAddr; R[RA] = NextInst	JAL Immediate	–
Jump and Link Register	JAL(R)	P	PC = JMPAddr; R[R1] = NextInst	JAL R1, Imm	–
Jump Register and Link	J(R)AL	P	PC = R[R1]; R[RA] = NextInst	JAL R1	11
Jump Register and Link Register	J(R)AL(R)	R1	PC = R[R1]; R[R2] = PC + 2	JAL R1, R2	11
Jump if Not Equal	JPN	I	If Flags, PC = PC + 4 + (SignExtImm << 1)	JPN Immediate	9
Jump if Not Equal Register	JPN(R)	R1	If Flags, PC = R[R1]	JPN R1	12
Jump if Equal	JPE	I	If Flags, PC = PC + 4 + (SignExtImm << 1)	JPE Immediate	10
Jump if Equal Register	JPE(R)	R1	If Flags, PC = R[R1]	JPE R1	13
Jump if Greater Than	JGT	I	If Flags, PC = PC + 4 + (SignExtImm << 1)	JGT Immediate	11
Jump if Greater Than Register	JGT(R)	R1	If Flags, PC = R[R1]	JGT R1	14
Jump if Greater Than Unsigned	JGTU	P	If Flags, PC = PC + 4 + (SignExtImm << 1)	JGTU Immediate	–
Jump if Greater Than Register Unsigned	JGT(R)U	R2	If Flags, PC = R[R1]	JGTU R1	13/3/0
Jump if Greater Than or Equal	JGE	P	If Flags, PC = NextInst + (SignExtImm << 1)	JGE Immediate	–
Jump if Greater Than or Equal Register	JGE(R)	R1	If Flags, PC = R[R1]	JGE R1	15
Jump if Greater Than or Equal Unsigned	JGEU	P	If Flags, PC = PC + 4 + (SignExtImm << 1)	JGEU Immediate	–
Jump if Greater Than or Equal Register Unsigned	JGE(R)U	R2	If Flags, PC = R[R1]	JGEU R1	13/3/1

## Additional

Instruction	Mnemonic	Format	Operation (in Verilog)	Assembly	Opcode/(hex)
No Operation	NOOP	R1	–	NOOP	0
End Program	END	R1	$PC = PC$	END	1
Increment	INC	R1	$R[R1] = R[R1] + 1$	INC R1	2
Decrement	DEC	R1	$R[R1] = R[R1] - 1$	DEC R1	3
Move Register	MOV	R1	$R[R1] = R[R2]$	MOV R1, R2	4
Swap Registers	SWAPR	R1	$temp = R[R2]; R[R2] = R[R1];$ $R[R1] = temp$	SWAPR R1, R2	5