

# ISA Table

Saturday, June 12, 2021 8:31 PM

Category	ASM	OpCode	Layout				Description	Example	
			OpCode	$posA$	$posB$	$posC$			
None	NOP	0x00	NOP	$pad$	$pad$	$pad$	Does nothing; used for attaching a debugger, patching code, timing, byte alignment	NOP	0x00 00 00 00
Math (ALU)	ADD	0x10	ADD	$R_{dest}$	$R_{src1}$	$R_{src2}$	Adds contents of $R_{src1}$ and $R_{src2}$ registers, and places result in $R_{dest}$ register	ADD R1 R2 R3	0x10 01 02 03
Math (ALU)	SUB	0x11	SUB	$R_{dest}$	$R_{src1}$	$R_{src2}$	Subtracts contents of $R_{src2}$ from contents of $R_{src1}$ registers, and places result in $R_{dest}$ register	SUB R1 R2 R3	0x11 01 02 03
Math (ALU)	MUL	0x12	MUL	$R_{dest}$	$R_{src1}$	$R_{src2}$	Multiplies contents of $R_{src1}$ by contents of $R_{src2}$ registers, and places result in $R_{dest}$ register	MUL R1 R2 R3	0x12 01 02 03
Math (ALU)	DIV	0x13	DIV	$R_{dest}$	$R_{src1}$	$R_{src2}$	Divides contents of $R_{src1}$ by contents of $R_{src2}$ registers, and places result in $R_{dest}$ register	DIV R1 R2 R3	0x13 01 02 03
Math (ALU)	MOD	0x14	MOD	$R_{dest}$	$R_{src1}$	$R_{src2}$	Returns the remainder of a division of contents of $R_{src1}$ by contents of $R_{src2}$ registers, and places result in $R_{dest}$ register	MOD R1 R2 R3	0x14 01 02 03
Logic (ALU)	NOT	0x20	NOT	$R_{dest}$	$R_{src}$	$pad$	Performs bitwise negation of contents of $R_{src}$ register, and places result in $R_{dest}$ register	NOT R5 R6	0x20 05 06 00
Logic (ALU)	AND	0x21	AND	$R_{dest}$	$R_{src1}$	$R_{src2}$	Performs a bitwise AND on contents of $R_{src1}$ and $R_{src2}$ registers, and places result in $R_{dest}$ register	AND R2 R3 R5	0x21 02 03 05
Logic (ALU)	OR	0x22	OR	$R_{dest}$	$R_{src1}$	$R_{src2}$	Performs a bitwise OR on contents of $R_{src1}$ and $R_{src2}$ registers, and places result in $R_{dest}$ register	OR R2 R3 R5	0x22 02 03 05
Logic (ALU)	XOR	0x23	XOR	$R_{dest}$	$R_{src1}$	$R_{src2}$	Performs a bitwise XOR on contents of $R_{src1}$ and $R_{src2}$ registers, and places result in $R_{dest}$ register	XOR R2 R3 R5	0x23 02 03 05
Logic (ALU)	EQ	0x24	EQ	$R_{dest}$	$R_{src1}$	$R_{src2}$	Performs a bitwise comparison on contents of $R_{src1}$ and $R_{src2}$ registers, and places result in $R_{dest}$ register; 1 if the two values are equal; 0 if they are not	EQ R2 R3 R5	0x24 02 03 05
Logic (ALU)	NEQ	0x25	NEQ	$R_{dest}$	$R_{src1}$	$R_{src2}$	Performs a bitwise comparison on contents of $R_{src1}$ and $R_{src2}$ registers, and places result in $R_{dest}$ register; 1 if the two values are not equal; 0 if they are	NEQ R2 R3 R5	0x25 02 03 05
Logic (ALU)	GTE	0x26	GTE	$R_{dest}$	$R_{src1}$	$R_{src2}$	Performs a bitwise comparison on contents of $R_{src1}$ and $R_{src2}$ registers, and places result in $R_{dest}$ register; 1 if the value in $R_{src1} \geq$ value in $R_{src2}$ ; 0 otherwise	GTE R2 R3 R5	0x26 02 03 05
Logic (ALU)	LTE	0x27	LTE	$R_{dest}$	$R_{src1}$	$R_{src2}$	Performs a bitwise comparison on contents of $R_{src1}$ and $R_{src2}$ registers, and places result in $R_{dest}$ register; 1 if the value in $R_{src1} \leq$ value in $R_{src2}$ ; 0 otherwise	LTE R2 R3 R5	0x27 02 03 05
Logic (ALU)	GT	0x28	GT	$R_{dest}$	$R_{src1}$	$R_{src2}$	Performs a bitwise comparison on contents of $R_{src1}$ and $R_{src2}$ registers, and places result in $R_{dest}$ register; 1 if the value in $R_{src1} >$ value in $R_{src2}$ ; 0 otherwise	GT R2 R3 R5	0x28 02 03 05
Logic (ALU)	LT	0x29	LT	$R_{dest}$	$R_{src1}$	$R_{src2}$	Performs a bitwise comparison on contents of $R_{src1}$ and $R_{src2}$ registers, and places result in $R_{dest}$ register; 1 if the value in $R_{src1} <$ value in $R_{src2}$ ; 0 otherwise	LT R2 R3 R5	0x29 02 03 05

Category	ASM	OpCode	Layout				Description	Example	
			OpCode	posA	posB	posC			
Flow Control	JMP	0x30	JMP	Addr <sub>hb</sub>	Addr <sub>lb</sub>	pad	Jumps to a memory location denoted by Addr <sub>hb</sub> and Addr <sub>lb</sub> bytes (big-endian). Supports <i>labelName</i> syntax instead of directly entering address.	JMP 0xBEEF	0x30 BE EF 00
								JMP labelName	0x30 ?? ?? 00
Flow Control	JMPi	0x31	JMPi	R <sub>loc</sub>	pad	pad	Jumps to a memory location stored in R <sub>loc</sub> register. Stored location is 2-bytes (big-endian)	JMPi R1	0x31 01 00 00
Flow Control	JMPT	0x32	JMPT	Addr <sub>hb</sub>	Addr <sub>lb</sub>	R <sub>condition</sub>	Jumps to a memory location denoted by Addr <sub>hb</sub> and Addr <sub>lb</sub> bytes (big-endian) if the value in R <sub>condition</sub> register is not zero. Supports <i>labelName</i> syntax instead of directly entering address.	JMPT 0xBEEF R0	0x32 BE EF 00
								JMPT labelName R0	0x32 ?? ?? 00
Flow Control	JMPTi	0x33	JMPTi	R <sub>loc</sub>	R <sub>condition</sub>	pad	Jumps to a memory location stored in R <sub>loc</sub> register if the value in R <sub>condition</sub> register is not zero. Stored location is 2-bytes (big-endian)	JMPTi R1 R0	0x33 01 00 00
Memory	SET	0x40	SET	R <sub>loc</sub>	Val <sub>hb</sub>	Val <sub>lb</sub>	Sets a literal value specified by Val <sub>hb</sub> and Val <sub>lb</sub> (big endian) into R <sub>loc</sub> register.	SET R1 0xBEEF	0x40 01 BE EF
Memory	COPY	0x41	COPY	R <sub>dest</sub>	R <sub>src</sub>	pad	Copies contents of R <sub>src</sub> register into R <sub>dest</sub> register	COPY R5 R6	0x41 05 06 00
Memory	LOAD	0x42	LOAD	R <sub>dest</sub>	Addr <sub>hb</sub>	Addr <sub>lb</sub>	Loads the value from a memory location denoted by Addr <sub>hb</sub> and Addr <sub>lb</sub> bytes (big-endian) into R <sub>dest</sub> register	LOAD R4 0xBEEF	0x42 04 BE EF
Memory	LOADi	0x43	LOADi	R <sub>dest</sub>	R <sub>src</sub>	pad	Loads the value from a memory at the address stored in R <sub>src</sub> into R <sub>dest</sub> register	LOADi R4 R5	0x43 04 05 00
Memory	STR	0x44	STR	Addr <sub>hb</sub>	Addr <sub>lb</sub>	R <sub>src</sub>	Stores the value from R <sub>src</sub> register into memory location denoted by Addr <sub>hb</sub> and Addr <sub>lb</sub> bytes (big-endian)	STR 0xBEEF R1	0x44 BE EF 01
Memory	STRi	0x45	STRi	R <sub>dest</sub>	R <sub>src</sub>	pad	Stores the value from R <sub>src</sub> register into memory location at the address of the value of R <sub>dest</sub> .	STRi R4 R1	0x45 04 01 00
Memory	PUSH	0x46	PUSH	R <sub>src</sub>	pad	pad	Pushes contents of the R <sub>src</sub> register onto the stack	PUSH R5	0x46 05 00 00
Memory	POP	0x47	POP	R <sub>dest</sub>	pad	pad	Pops the value on top of the stack into R <sub>dest</sub> register	POP R6	0x47 06 00 00