Instruction Set Architecture

User Programmable Registers

Number	Nickname	Purpose
R0	zero	Zero
R1	-	General Purpose (Convention: Function Return Value)
R2-R12	-	General Purpose
R13	sp	Stack Pointer
R14	gp	Global Pointer
R15	pc	Program Counter

Instruction Formats

Register Type

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode				Destination Register				Source Register 1				Source Register 2			

Immediate Type

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode				Destination Register				Immediate/Register							

* Register used for LD/ST operations

Jump Type

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode				Address											

Instruction Set

Operand encodings are consistent across each format type. All register encoding results in a 4-bit binary number (since there are 16 registers). For the I-Type format there are 8-bit, signed binary numbers saved as the immediate/offset value. And for J-Type there are 12-bit, unsigned binary numbers saved for the address.

Instruction	Assembly	Format	Op-Code
No Operation	NOP	J-Type	0b0000
Addition	ADD	R-Type	0b0001
Subtraction	SUB	R-Type	0b0010
And	AND	R-Type	0b0011
Or	OR	R-Type	0b0100
Not	NOT	R-Type	0b0101
Move	MOV	I-Type	0b0110
Load	LD	R-Type	0b0111
Store	ST	R-Type	0b1000
Compare	CMP	R-Type	0b1001
Branch Equal	BEQ	I-Type	0b1010
Branch Not Equal	BNE	I-Type	0b1011
Jump	JMP	J-Type	0b1100
Jump Register	JR	R-Type	0b1101
Set Less Than	SLT	R-Type	0b1111

Examples

C	**Assembly**
short a = 5;	MOV r1, #5

short a,b=5,c=5;
a=b+c;
	MOV r2, #5
	MOV r3, #5
	ADD r1, r2, r3

short a,b=5,c=5;
a=b-c;
	MOV r2, #5
	MOV r3, #5
	SUB r1, r2, r3

short a = 3&5;	MOV r2, #3
	MOV r3, #5
	AND r1, r2, r3

short a = 3|5;	MOV r2, #3
	MOV r3, #5
	OR r1, r2, r3

short a = ~5;	MOV r2, #5 NOT r1, r2

short a = 3 == 3 ? 1 : 0;	MOV r1, #3
	MOV r2, #3
	CMP r1, r2
	BEQ Equals
	MOV r3, #0
	JMP Exit
	Equals:
	MOV r3, #1
	Exit:

short a = 3 != 3 ? 1 : 0;	MOV r1, #3
	MOV r2, #3
	CMP r1, r2
	BNE NotEquals
	MOV r3, #0
	JMP Exit
	NotEquals:
	MOV r3, #1
	Exit:

short a = 2 > 1 ? 1 : 0;	MOV r1, #1
	MOV r2, #2
	SLT r3, r1, r2
	BNE TrueConditional
	MOV r1, #0
	JMP Exit
	TrueConditional:
	MOV r1, #1
	Exit:

short a = 2 <= 1 ? 1 : 0;	MOV r1, #1
	MOV r2, #2
	SLT r3, r1, r2
	BEQ TrueConditional
	MOV r1, #0
	JMP Exit
	TrueConditional:
	MOV r1, #1
	Exit:

short a = 2 < 1 ? 1 : 0;	MOV r1, #1
	MOV r2, #2
	SLT r3, r2, r1
	BNE TrueConditional
	MOV r1, #0
	JMP Exit
	TrueConditional:
	MOV r1, #1
	Exit:

short a = 2 >= 1 ? 1 : 0;	MOV r1, #1
	MOV r2, #2
	SLT r3, r2, r1
	BEQ TrueConditional
	MOV r1, #0
	JMP Exit
	TrueConditional:
	MOV r1, #1
	Exit:

function my_function(int a) {
  return a;
}
int main() {
  short a = my_function(5);
}
	MOV r2, #5
	PUSH r2
	CALL my_function
	ADD r2, r1, zero
	
	my_function:
	POP r2
	ADD r1, r2, zero
	RET

function my_function(int *a) {
  return *a;
}
int main() {
  short a = 5;
  short b = my_function(&a);
}
	# addr_a is the address of variable a
	# (named constant in this example code)
	MOV r2, #5
	ST [addr_a], r2
	ADD r3, zero, addr_a
	PUSH r3
	CALL my_function
	ADD r3, r1, zero
	JMP Exit
	
	my_function:
	POP r3
	LD r3, r3
	MOV r1, r3
	RET
	
	Exit: