INSTRUCTION SET ARCHITECTUE (ISA) DESIGN FOR A 16-bits microprocessor

This ISA was designed in conformity with the RISC (Reduced Instruction Set Computer) design principles, with careful consideration of simplicity and making the common case fast inorder to achieve the set goals of low cost and minimal CPI (Clock Cycle Per Instruction). I came up with **14 unique Instructions (4-bits opcode)**, with few of the instructions having multiple formats, **3 types of instruction formats (16-bits word)** based on the defined operations my instructions must support, and **8 register types (3-bits)** to make the common cases fast.

Registers and their roles

Name	Register number	Function	Address
$L	0	Holds constant value 0	000
$v0	1	Holds value for results and expression evaluation	001
$t0 - $t1	2 - 3	Temporary registers	010 -011
$s0- $s2	4 - 6	Saved registers	100-110
$Ra	7	Holds return address	111

Instruction format types

Register-type instruction format: this format is used for logical, relational, jump and arithmetic operations carried out in the registers. This instruction format for the operations has a single opcode, but different 3-bit function codes (000 - 111) to decipher the operation being done.

4-bits Opcode	3-bits destination register (Rd)	3-bits source register (Rt)	3-bits source register (Rs)	3-bits function code

Immediate-type instruction format: this format handles operations with constants, decision making instructions, data transfer instructions, and relative addressing.

4-bits opcode	3-bits source/destination register (Rt)	3-bits source/destination register (Rs)	6-bits immediate address

Jump-type instruction format: this supports absolute addressing and halt operation.

4-bits opcode	12-bits address

Instruction Set Architecture

R-type format

Name	Mneumonic & operands	Machine code			Justification of each instruction in the ISA
		opcode	Rd|Rs|Rt	Function code	
Add	add $s1, $s0,$t0	0000	101100010	000	This instructions adds two values two registers and stores in a register
Subtract	sub $s2, $s1,$t0	0000	110101010	001	This instruction subtracts two values and save in a register

And	and $2 , $t1,$s1	0000	110011101	010	This instruction does logic and operation on two values
Or	or $t0, $s0,$s1	0000	010100101	011	This instruction does logic or operation on two registers ($s0 | $s1)
Set less than	Slt $s1, $s0,$s2	0000	011100110	100	This instruction sets the destination register to 1 if $s0 < $s2.
Jump to return address	Jr $ra	0000	111000000	101	This instruction jumps to the register $ra holding the return address
Nor	nor $t1, $s0,$s1	0000	011100101	110	This instruction does logical nor operation on the two values in the two registers ~($s0 | $s1)
Set greater than	Sgt $s2, $s1,$s0	0000	110101100	111	This instruction sets the destination register to 1 if $s1 > $s0

I-type format

Name	Mneumonics & operand	Machine code			Justification of instruction in the ISA
		opcode	Rt | Rs	6-bit address	
Add immediate	addi $s1, $s0, 30	0001	101 100	011110	This instruction adds a constant to the value in a register and stores the result in another register
Subtract immediate	subi $s1, $s0, 30	0010	101 100	011110	This instruction performs a subtraction operation between a constant and the value in a register and stores in another register
Load word	lw $t0 20($s2)	0011	010 110	010100	This instruction simply loads a content in the memory to a register
Store word	sw $s0 20($s2)	0100	100 110	010100	This instruction stores a value from a register to the memory
Branch not equal	Bne $t0, $s2 label(10)	0101	010 110	001010	This instruction tests the status of the two registers before branching to
Branch equal	Beq $t0, $s2 label(10)	0110	010 110	001010	This instruction tests the status of the two registers before branching to a specific address.

And immediate	andi $s2, $t1, 20	0111	110 011	010100	This instruction performs a logical and operation a register and a constant.
Or immediate	andi $s2, $t1, 20	1000	110 011	010100	This instruction performs a logical or operation a register and a constant.
Set less than immediate	slti $s2, $t0 , 10	1001	110 011	001010	This instruction sets the destination register ($s2) to 1 if $t0 < constant.
Set greater than immediate	sgti $s2, $t0 , 10	1010	110 011	001010	This instruction sets the destination register ($s2) to 1 if $t0 >constant

J-type format

Name	Mneumonic & operands	Machine code		Justification of instruction in the ISA
		Opcode	12-bit address	
Jump	J address	1100	000001100100	This instruction is meant to jump to an address
Halt	Halt	1111	000000000000	This instruction is meant to stop the program
Jump and Link	Jal address ($ra<-PC+1;address)	1101	000001000000	This instruction is meant to transfer the current address in program counter (PC) to the $ra and load the PC with the address

C Language VS assembly language

Name	C- language	Assembly language
add	`$s2 = $s1 + $t0;`	`add $s2, $s1, $t0`
subtract	`$s2 = $s1 - $t0;`	`sub $s2, $s1, $t0`
assignment	`$s2 = $s1;`	`add $s2, $s1, $L`
and	`$s2 = $s1 & $t0;`	`and $s2, $s1, $t0`
or	`$s2 = $s1 | $t0;`	`or $s2, $s1, $t0`

for loop	`for ($t0= 1; $t0<=10; $t0++){`
	`$s1= $t0 + 2;`
	`}`
		`addi $s0, $L, 10`
		`loop: addi $t0, $L 1`
		`beq $s0, $t0`
		`exit`
		`addi $s1, $t0, 2`
		`j loop`
		`Halt`

While loop	`while($t0 >= 10)`
	`{ $t0 = $t0 -1;}`
		`addi $s0, $L, 10`
		`addi $t0, $L, 20`
		`loop: sgt $s1, $so, $t0`
		`bne $s1, $L`
		`exit`

		subi $t0, $t0, 1 j loop Halt
If-else	If ($s1 < 6) { $s2 = $t0 - $s1; } else { $s2 = $t0 - $s1; }	loop: sgti $s0, $t1, 6 be $s0, $L label sub $s2, $t0, $s1 j loop label: add $s2, $t0, $s1

While loop	while($s2 != 6)
	{ $s2 = $s2 + 2;}
		addi $t0, $L, 6
		loop: beq $s2, $t0 label
		addi $s2.$t1,2
		j loop
		label:

	while($t0 ==$t1){ $s1 = $t0 + $t1}	Loop: bne $t0, $t1 label add $s1, $t0, $t1 j loop label:
If else	If ($s0 > 10) {$t0 = $s0} else {$t0 = $s0 + 1}	Loop: slti $s1,$s0,10 label add $t0, $L, $s0 j loop label: addi $t0, $s0, 1

Function (call and return)	Int addition(int $s1, int $s0){
	Int $s2 = $s1 + $s0;
	Return ($s2)}
		addi $t1, $L, 10
		sw $s0, 0($t1)
		sw $s1, 1($t1)
		sw $s2, 2($t1)
		
		jal *proc*
		
		lw $s0, 0($t1)
		lw $s1, 1($t1)
		lw $s2, 2($t1)
		
		jr $ra
		
		proc:
		add $s2, $s1,$s0
		add $v0, $L, $s2

Arrays	Int t0, s0,s1;
	t0 = s0[3] + s1[2];
		lw Ss0,3($t1)
		lw Ss1,2($t1)
		add $t0, $s0, $s1
		Halt