Instruction Set Architecture:

Maintaining the conditions provided, the instruction set for a 16 bit microprocessor is given here. The opcode was chosen to be four bits to support all the sixteen instructions. There are four different types of instruction and 8 registers. The instruction types are R-type (Register type), I-type (Immediate type), J-type (Jump type) and S-type (Special type). The name, description and type of 16 instructions have been stated in the table below.

Instruction	Opcode	Type	Description
halt	0000	S-Type	Halts the CPU
add	0001	R-Type	Add two registers
sub	0010	R-Type	Subtract one register from another
and	0011	R-Type	Bitwise AND two registers
or	0100	R-Type	Bitwise OR two registers
slt	0101	R-Type	Set to 1 if one register is less than another
sll	0110	R-Type	Logical/Arithmetic shift left
srl	0111	R-Type	Arithmetic shift right
addi	1000	I-Type	Add immediate operand
lw	1001	I-Type	Load word from memory
sw	1010	I-Type	Store word from memory
bne	1011	I-Type	Branch if two registers are not equal
beq	1100	I-Type	Branch if two registers are equal
jal	1101	J-Type	Jump to location and set link register
j	1110	J-Type	Jump to location within instruction
ja	1111	S-Type	Jump to location in register

Different types of instruction

(i) Machine code of R-type:

Machine code of R-type has 5 segments. First 4 bit is opcode. Then next 3 bit is for source register (rs). Then next 3 bit for target register (rt) and then three bit for destination register (rd). Here 3 bit is used to denote all 8 registers. The last 3 bit of the 16 bit machine code is for ‘shamt’ which denotes shift amount. This is only used for shifting left or right.

opcode	rs	rt	rd	shamt
4-bit	3-bit	3-bit	3-bit	3-bit

(ii) Machine code of I-type:

Machine code of R-type has 4 segments. The first 4 bit is for opcode. Next 3 bit is for source register (rs). Then 3 bit for destination register (rd). Last 6 bit is for immediate value.

opcode	rs	rd	immediate
4-bit	3-bit	3-bit	6-bit

(iii) Machine code of J-type:

Machine code of J-type has only 2 segments. First 4-bit is for opcode and the rest 12 bit is for the address to where jumping is needed.

opcode	address
4-bit	12-bit

(iv) Machine code of S-type:

Machine code of S-type has two segments. The first 4 bit is opcode. The next 3 bit is for a register (rs). The rest 9 bit is don't care bit. If opcode is all zero halt operation will take place and if opcode is all 1 than jump will occur to the address stored in the register denoted in the second segment.

opcode	rs	don't Care
4-bit	3-bit	9-bit

Details of different instructions:

add: Used to add two registers together. The last 4-bit will not affect the operation (don't care). This is a common and important operation for processor. That's why it is included.

Assembly Code: add $rd, $rs, $rt
Function: $rd = $rs + $rt

opcode	rs	rt	rd	shamt
0001	XXX	XXX	XXX	XXX

sub: Used to subtract one from register from another. The last 4-bit will not affect the operation (don't care). This operation has the similar importance as adding and it is a common operation.

Assembly Code: sub $rd, $rs, $rt
Function: $rd = $rs - $rt

opcode	rs	rt	rd	shamt
0010	XXX	XXX	XXX	XXX

and: Used to bitwise AND two registers. The last 4-bit will not affect the operation (don't care). This fulfills the project requirements to support the '&' C construct.

Assembly Code: and $rd, $rs, $rt
Function: $rd = $rs & $rt

opcode	rs	rt	rd	shamt
0011	XXX	XXX	XXX	XXX

or: Used to bitwise OR two registers. The last 4-bit will not affect the operation (don't care). This fulfills the project requirements to support the '|' C construct.

Assembly Code: or $rd, $rs, $rt
Function: $rd = $rs | $rt

opcode	rs	rt	rd	shamt
0100	XXX	XXX	XXX	XXX

slt: It Sets the destination register to '1' if the first source register is less than the second source register. It clears the destination register to "0" otherwise. This is the primary instruction for branch conditions. It is necessary for the four required condition checks: >, <, >=, <=. The last 3-bit is not used in this instruction (don't care).

Assembly Code: slt $rd, $rs, $rt
Function: $rd = $rs < $rt ? 1 : 0

opcode	rs	rt	rd	shamt
0101	XXX	XXX	XXX	XXX

sll: Performs a left shift by the given amount. Useful in performing calculations involving powers of two, and helps in manipulating bit patterns. Here 'rs' is the source register, 'rd' is the destination register and 'shamt' is shift amount. 'rt' is not used in this case (don't care).

Assembly Code: sll $rd, $rs, [0 to 16]
Function: $rd = $rs << [0 to 16]

opcode	rs	rt	rd	shamt
0110	XXX	XXX	XXX	XXX

srl: Performs a right shift by the given amount. Useful in performing calculations involving powers of two, and helps in manipulating bit patterns. Here 'rs' is the source register, 'rd' is the destination register and 'shamt' is shift amount. 'rt' is not used in this case (don't care).

Assembly Code: srl $rd, $rs, [0 to 16]
Function: $rd = $rs >> [0 to 16]

opcode	rs	rt	rd	shamt
0111	XXX	XXX	XXX	XXX

addi: Adds an immediate operand to the given register. Useful in simple calculations, as memory and registers are not required to store small constants.

Assembly Code: addi $rd, $rs, [-32 to 31]
Function: $rd = $rs + [-32 to 31]

opcode	rs	rd	immediate
1000	XXX	XXX	XXXXXX

lw: Load a word from memory into a register. A necessary operation, as registers cannot hold all the necessary data in most programs.

Assembly Code: lw $rd, [-32 to 31]($rs)
Function: $rd = *($rs + [-32 to 31])

opcode	rs	rd	immediate
1001	XXX	XXX	XXXXXX

sw: Store a word from a register into memory. A necessary operation, as registers cannot hold all the necessary data in most programs.

Assembly Code: sw $rd, [-32 to 31]($rs)
Function: *($rs + [-32 to 31]) = $rd

opcode	rs	rd	immediate
1010	XXX	XXX	XXXXXX

bne: Add an offset to the PC counter if the two operands are not equal. Necessary to fulfill the branch condition!=, and used along with slt for > and < conditions.

Assembly Code: bne $rs, $rd, [-32 to 31]
Function: $pc = ($rs != $rd) ? $pc + [-32 to 31] : $pc + 1

opcode	rs	rd	immediate
1011	XXX	XXX	XXXXXX

beq: Add an offset to the PC counter if the two operands are equal. Necessary to fulfill the branch condition ==, and used along with SLT for >= and <= conditions.

Assembly Code: beq $rs, $rd, [-32 to 31]
Function: $pc = ($rs == $rd) ? $pc + [-32 to 31] : $pc + 1

opcode	rs	rd	immediate
1100	XXX	XXX	XXXXXX

jal: Jumps to the location given in the opcode by loading the immediate into the bottom twelve bits of the program counter. It also stores the current value of $pc + 1 into the link register to enable a return from a function call.

Assembly Code: jal [0 to 4095]
Function: $pc = concatenate ($pc[15:12], [0 to 4095])

opcode	address
1101	XXXXXXXXXXXX

j: Jumps to the location given in the opcode by loading the immediate into the bottom twelve bits of program counter.

Assembly Code: j [0 to 4095]
Function: $pc = concatenate ($pc [15:12], [0 to 4096])

opcode	address
1110	XXXXXXXXXXXX

jr: Jumps to the location in the given register. Necessary for returning from a function call, as you need to jump to the location in the link register.

Assembly Code: jr $rs
Function: $pc = $rs

opcode	rs	don't Care
1111	XXX	XXXXXXXXX

halt: Stops the program.

Assembly Code: halt
Function: $pc = $pc

opcode	don't Care
0000	XXXXXXXXXXXX

Register Roles:

The architecture was chosen to have only eight registers. As it is 16 bit processor the instruction size is also 16 bit. So if the register number is greater than 8, more bits would be needed to denote them. Register number, their name and role is stated in the table below.

Register	Name	Role
$0	$zero	Zero register that holds all zeros
$1	$v0	Function return and used in pseudo instructions
$2	$a0	First Argument
$3	$a1	Second Argument
$4	$t0	Temporary register
$5	$t1	Temporary register
$6	$s0	Saved register
$7	$lr	Link register; where the return address is stored during a function call