This is a 16 bit computer built on a Xilinx Zynq 7020 FPGA. Here are how the instructions are represented in binary, which are loaded into the instruction memory from a USB serial port.

	Instruction Bits																														
Instruction	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Halt	■	■	■	■																												
Return From Trap	■	■	■																													
Make Atomic	■	■		■			s/e																									
OS Interrupt	■	■						i/r	Type Reg				Immediate																			
Conditional Jump	■		■	■	Type			i/r	Address Reg				Immediate																			
Timer	■		■		Type			i/r	Reg A				Immediate																			
Move	■			■	Type				Source Reg				Immediate															Dest Reg			
Conditional Set	■				Type			i/r																				Dest Reg			
Add/Subtract		■	■	■			a/s	i/r	Reg A				Immediate / Reg B															Dest Reg			
Right Shift		■	■				l/a	i/r	Reg A				Immediate / Reg B															Dest Reg			
Left Shift		■		■				i/r	Reg A				Immediate / Reg B															Dest Reg			
Extract Bit		■						i/r	Reg A				Immediate / Reg B															Dest Reg			
Bitwise And			■	■				i/r	Reg A				Immediate / Reg B															Dest Reg			
Bitwise Or			■					i/r	Reg A				Immediate / Reg B															Dest Reg			
Bitwise Xor				■				i/r	Reg A				Immediate / Reg B															Dest Reg			
Count 1s									Reg A																			Dest Reg			

Registers (16 bits)	
r0	zero register
r1	GP
r2	GP
r3	GP
r4	GP
r5	GP
r6	GP
r7	GP
r8	Func. Arg 1
r9	Func. Arg 2
r10	Func. Arg 3
r11	Base Pointer
r12	Stack Pointer
r13	Cycle Counter
r14	Flags
r15	Program Counter

Flags
Zero - 0
Negative - 1
Carry - 2
Overflow - 3

Special bits
Atomic
Kernel

Halt
Stops the program on the next clock cycle.
HLT
Instruction Binary:
0000 0000 0000 0000 0000 0000 0000 0000

Jump
Jumps based on the current flags.

Instruction	Behavior	Type code	Flags
JEQ	Jump if equal	0 0 0	Z
JNE	Jump if not equal	0 0 1	!Z
JGT	Jump if greater than	0 1 0	!Z & !N
JGE	Jump if greater than or equal	0 1 1	Z | !N
JLT	Jump if less than	1 0 0	N
JLE	Jump if less than or equal	1 0 1	Z | N
JOV	Jump if overflow	1 1 0	O
JMP	Unconditional jump	1 1 1	Z | !Z

Examples:
Jump to the address stored in r1.
    JEQ r1
Jump to the address 0x41F9.
    JGE $0x41F9

Instruction Binary:
0001 [type code (3)] [immediate or register (1)] [address register (4)] [immediate (16)]

No Op
Does nothing, costs one clock cycle. Often used for pipeline stalls.
NOP
This is a pseudoinstruction. It is implemented as follows:
MOV r0 r0

Make Atomic
Makes the instructions between the ATS and ATE instructions atomic, preventing the operating system from initiating a context switch when these instructions are running. Useful for implementing mutexes, but abusable. Sets and resets the Atomic bit, and can only be used in kernel mode.

Examples:
Make the next 3 instructions atomic.

```
ATS
MOV [r1] r2
ADD r2 $0x1 r2
MOV r2 [r1]
ATE
```

Instruction Binary:
0010 00 [immediate or register (1)] 0 0000 0000 0000 0000 0000 0000

Move

Format	Behavior	Type code
MOV $imm r1	Moves an immediate into a register.	0 0 0
MOV r1 r2	Copies register 1 into register 2	0 0 1
MVB $imm[r1] r2	Copies the byte at mem[r1 + imm] into r2	0 1 0
MVB r1 $imm[r2]	Copies the lower half of r1 into the byte at mem[r2 + imm]	0 1 1
MVW $imm[r1] r2	Copies the bytes at mem[r1 + imm] and mem[r1 + imm + 1] into r2	1 0 0
MVW r1 $imm[r2]	Copies r1 into mem[r2 + imm] and mem[r2 + imm + 1]	1 0 1

Examples:
Move the value 5 (byte) into r3.

```
MVB $0x5 r3
```

Move the two bytes stored at mem[r15 + 8] into r1.

```
MVW $0x8[r15] r1
```

Instruction Binary:
0110 [type code (3)] 0 [source reg (4)] [immediate (16)] [dest reg(4)]

Push

Pushes a register or immediate onto the stack, decreasing the stack pointer. You can push either one or two bytes.

This is a pseudoinstruction. The implementation of PSB (push byte) is as follows:

```
SUB 0x8 r12
MVB (value) [r12]
```

PSW (push word) is as follows:

```
SUB 0x10 r12
MVW (value) [r12]
```

Examples:

Push the byte $0x51 onto the stack.

```
PSB $0x51
```

Push the two bytes $0xBEEF onto the stack.

```
PSW $0xBEEF
```

Push r1 onto two bytes of the stack.

```
PSW r1
```

Pop

Pops the value pointed to by the stack pointer, then increases the stack pointer.

This is a pseudoinstruction. The implementation of PPB (pop byte) is as follows:

```
MVB [r12] reg
ADD 0x8 r12
```

PPW (pop word) is implemented as such:

```
MVW [r12] reg
ADD 0x10 r12
```

Examples:

Pop one byte into r2.

```
PPB r2
```

Pop two bytes into r4.

PPW r4

Call
Calls a function. Pushes the current program counter onto the stack, then jumps to the address of the function.
This is a pseudoinstruction. The implementation of CAL is as follows:

```
PSW r15
JMP [reg/address]
```

Examples:
Call a function by its label.

```
CAL .add_two_numbers
```

Call a function by its address.

```
CAL $0x11CE
```

Return
Exits a function, and returns control to the caller by popping the value pointed to by r12 (the stack pointer) into r15 (program counter)
This is a pseudoinstruction. The implementation of RET is as follows:

```
PPW r15
```

Warning: failing to return the stack pointer to its original place at the beginning of the function will cause the return instruction to return to the wrong address.

Interrupt/Syscall
Triggers an interrupt, giving control to the operating system. A table of interrupts can be found in the Operating System section. Registers 8-10 are the arguments to the interrupt.

Examples:
Print to the screen. The address of the string to be printed is stored in r1, and its length is 6 characters. The syscall ID for printing is 4.

```
MOV r1 r8
MOV $0x6 r9
INT $0x4
```

Instruction Binary:
0011 0000 0000 [immediate (16)] 0000

ALU instructions

Instruction	Opcode	Type code	Behavior
ADD	1 0 0 0	0 0 0	Addition
SUB	1 0 0 0	0 0 1	Subtraction
RSL	1 0 0 1	0 0 0	Logical Right Shift
RSA	1 0 0 1	0 0 1	Arithmetic Right Shift
LSH	1 0 1 0	0 0 0	Left Shift
EXB	1 0 1 1	0 0 0	Extract Bit
AND	1 1 0 0	0 0 0	Bitwise AND
ORR	1 1 0 1	0 0 0	Bitwise OR
XOR	1 1 1 0	0 0 0	Bitwise XOR
CNT	1 1 1 1	0 0 0	Count 1s

The above instructions are all formatted the same. They can be formatted as follows:

Format	Behavior
OP r1 r2 r3	Calculates r1 OP r2, then stores the result in r3
OP r1 $imm r2	Calculates r1 OP immediate, stores the result in r2
OP r1 r2	Calculates r1 OP r2, stores the result in r2
OP r1 $imm	Calculates r1 OP immediate, stores the result in r1
OP $imm r1	Calculates immediate OP r1, stores the result in r1

Instruction Binary:
[Opcode (4)] [Typecode (3)] [reg/immediate (1)] [Reg A (4)] [Immediate (16)] [Dest reg (4)]
[Opcode (4)] [Typecode (3)] [reg/immediate (1)] [Reg A (4)] [Reg B (4)] 0000 0000 0000 [Dest reg (4)]

These are the set instructions. They will set the destination register to 1 if the condition is met.

Instruction	Behavior	Type code	Flags
SEQ	Set if equal	0 0 0	Z
SNE	Set if not equal	0 0 1	!Z
SGT	Set if greater than	0 1 0	!Z & !N
SGE	Set if greater than or equal	0 1 1	Z | !N
SLT	Set if less than	1 0 0	N
SLE	Set if less than or equal	1 0 1	Z | N
SOV	Set if overflow	1 1 0	O
SCR	Set if carry	1 1 1	C

The format is as follows:

Format	Behavior
SET r1 r2	Sets r2 equal to r1 if the condition is met
SET $imm r1	Sets r1 equal to the immediate if the condition is met

Instruction Binary:
0111 [type code (3)] [immediate/reg (1)] 0000 [immediate (16)] [dest reg (4)]
0111 [type code (3)] [immediate/reg (1)] 0000 [reg (4)] 0000 0000 0000 [dest reg (4)]

Bitwise NOT
This is a pseudoinstruction. It is implemented as:
XOR reg $0xFFFF dest
Usage: NOT r1 r2 -> calculates the inverted bits of r1 and stores them in r2

Comparison (CMP)
This is a pseudoinstruction. It is implemented as:
SUB q1 q2, formatted like the ALU instructions

Test (TST)
This is a pseudoinstruction. It is implemented as:
AND q1 q2, formatted like the ALU instructions