

ISA Documentation

Computer Architecture Documentation

Machine Specifications

Registers

8 Registers

- 6 General Purpose Registers
- R7 (RSP) is not directly accessible and is the Stack Pointer
- R8 (RIP) is the Instruction Pointer Register

Memory

- 16-bit addresses (Big-Endian)

Instruction Set

- 16-bit values and memory addresses
- OpCode byte is split into two nibbles (category - instruction)
- False: 0x0000 True: 0xFFFF
 - JMPT jumps as long as the value in $R_{\text{Condition}}$ is NOT 0
- General Purpose Registers labeled R0 - R6
 - RSP: Stack Pointer
 - RIP: Instruction Pointer (inaccessible)

No Operation

<i>OpCode</i>	<i>Name</i>	<i>Format</i>	<i>Description</i>
0x00	NOP	NOP PAD PAD PAD	No Operation

Math

<i>OpCode</i>	<i>Name</i>	<i>Format</i>	<i>Description</i>
0x10	ADD	ADD R_{Dest} R_{Addend} R_{Addend}	Adds the values of two registers and stores the sum in a destination register
0x11	SUB	SUB R_{Dest} R_{Minuend} $R_{\text{Subtrahend}}$	Subtracts the values of two registers and stores the difference in a destination register
0x12	MUL	MUL R_{Dest} R_{Factor} R_{Factor}	Multiplies the values of two registers and stores the product in a destination register
0x13	DIV	DIV R_{Dest} R_{Dividend} R_{Divisor}	Divides the values of two registers and stores the quotient in a destination register

0x14	MOD	MOD R_{Dest} $R_{Dividend}$ $R_{Divisor}$	Divides the values of two registers and stores the remainder in a destination register
------	-----	---	--

Logic

<i>OpCode</i>	<i>Name</i>	<i>Format</i>	<i>Description</i>
0x20	NOT	NOT R_{Dest} R_{Source} PAD	Performs a bitwise NOT operation on the contents of R_{Source} and stores the result in R_{Dest}
0x21	AND	AND R_{Dest} R_{LHS} R_{RHS}	Performs a bitwise AND operation on the contents of R_{LHS} and R_{RHS} registers and stores the result in R_{Dest}
0x22	OR	OR R_{Dest} R_{LHS} R_{RHS}	Performs a bitwise OR operation on the contents of R_{LHS} and R_{RHS} registers and stores the result in R_{Dest}
0x23	XOR	XOR R_{Dest} R_{LHS} R_{RHS}	Performs a bitwise XOR operation on the contents of R_{LHS} and R_{RHS} registers and stores the result in R_{Dest}
0x24	EQ	EQ R_{Dest} R_{LHS} R_{RHS}	Performs a bitwise equality check on the contents of R_{LHS} and R_{RHS} registers, storing 0xFFFF in R_{Dest} if equal and 0x0000 if not equal.
0x25	NEQ	NEQ R_{Dest} R_{LHS} R_{RHS}	Performs a bitwise inequality check on the contents of R_{LHS} and R_{RHS} registers, storing 0x0000 in R_{Dest} if equal and 0xFFFF if not equal.
0x26	GTE	GTE R_{Dest} R_{LHS} R_{RHS}	Performs a bitwise comparison on the contents of R_{LHS} and R_{RHS} registers, storing 0xFFFF in R_{Dest} if $R_{LHS} \geq R_{RHS}$ and 0x0000 otherwise.
0x27	LTE	LTE R_{Dest} R_{LHS} R_{RHS}	Performs a bitwise comparison on the contents of R_{LHS} and R_{RHS} registers, storing 0xFFFF in R_{Dest} if $R_{LHS} \leq R_{RHS}$ and 0x0000 otherwise.
0x28	GT	GT R_{Dest} R_{LHS} R_{RHS}	Performs a bitwise comparison on the contents of R_{LHS} and R_{RHS} registers, storing 0xFFFF in R_{Dest} if $R_{LHS} > R_{RHS}$ and 0x0000 otherwise.
0x29	LT	LT R_{Dest} R_{LHS} R_{RHS}	Performs a bitwise comparison on the contents of R_{LHS} and R_{RHS} registers, storing 0xFFFF in R_{Dest} if $R_{LHS} < R_{RHS}$ and 0x0000 otherwise.

Flow Control

<i>OpCode</i>	<i>Name</i>	<i>Format</i>	<i>Description</i>
---------------	-------------	---------------	--------------------

0x30	JMP	JMP pad Addr _{High} Addr _{Low} JMP #labelName	Jumps to a memory location denoted by a 16-bit value (Addr _{High} Addr _{Low}). Labels are supported in ASM, replacing the address
0x31	JMPi	JMPi pad R _{Location} pad	Indirectly jumps to a memory location denoted by a 16-bit value stored in R _{Location} .
0x32	JMPT	JMPT R _{Condition} Addr _{High} Addr _{Low} JMPT R _{Condition} #labelName	Jumps to a memory location denoted by a 16-bit value (Addr _{High} Addr _{Low}) if the value of a given register is not 0. Labels are supported in ASM, replacing the address
0x33	JMPTi	JMPTi R _{Condition} R _{Location} pad	Indirectly jumps to a memory location denoted by a 16-bit value (Addr _{High} Addr _{Low}) if the value of a given register is not 0.

Memory

OpCode	Name	Format	Description
0x40	SET	SET R _{Dest} Value _{High} Value _{Low}	Sets the contents of R _{Dest} to the literal value represented by Value _{High} Value _{Low}
0x41	COPY	COPY R _{Dest} R _{Source} PAD	Copies the contents of a R _{Dest} to R _{Source}
0x42	LOAD	LOAD R _{Dest} Addr _{High} Addr _{Low}	Loads the value from memory at Addr _{High} Addr _{Low} into R _{Dest}
0x43	LOADi	LOADi R _{Dest} R _{Source} PAD	Loads the value from memory at Addr _{High} Addr _{Low} into R _{Dest}
0x44	STR	STR R _{Source} Addr _{High} Addr _{Low}	Stores the contents of R _{Source} into memory at Addr _{High} Addr _{Low}
0x45	STRi	STRi R _{Source} R _{Dest} PAD	Stores the contents of R _{Source} into the memory location represented specified by the contents of R _{Dest}
0x46	PUSH	PUSH R _{Source} PAD PAD	Pushes the contents of R _{Source} onto the stack
0x47	POP	POP R _{Dest} PAD PAD	Pops the value on top of the stack into R _{Dest}

Display

<i>OpCode</i>	<i>Name</i>	<i>Format</i>	<i>Description</i>
0x50	CLR	CLR PAD PAD PAD	Clears the screen
0x51	DRAW	DRAW X _{Location} Y _{Location} Color	Sets the color of a pixel at the specified X,Y coordinates to a given color (XTerm256 format).
0x52	DRAWi	DRAWi R _{XLocation} R _{YLocation} R _{Color}	Sets the color of a pixel at the specified X,Y coordinates to a given color (XTerm256 format).

Compiler Notes

**Work In Progress*

Components of a Compiler

- Syntax
 1. Lexer / Tokenizer → Syntax Analysis & Lexemes
 2. Parser → CST & AST
- ???
 3. Semantic Analysis
 4. IR Code Gen → CIL
 5. Optimizer → CIL
 6. Code Gen → Target

Potential Oddities:

- Missing return statements in void functions are added in the codegen stage

Language Features

Has

- ☒ ~~Strong, Static Types~~
- ☒ ~~Reserved Keywords~~
- ☐ Inheritance
- ☐ Templates
 - ☐ Template Specialization
 - ☐ Template Metaprogramming???
 - ☐ Constexpr's
 - ☐ Template Declaration
 - ☐ SFINAE
 - ☐ Type Traits
- ☐ Limited Type Inference
 - ☐ var / new variables
 - ☐ var return types?
- ☐ Function Overloading?
- ☐ Named Loops?
- ☐ Nullable Context?
- ☐ Aliasing?

Doesn't Have:

- Structs
- Generics
- Operator overloading
- Multithreading
- Multiple inheritance