TINY

(A Very Small Computer)

Instruction Set Architecture
Reference Manual

Revision

January 1, 2023

The TINY machine is a very small simple computer designed to teach the basics of computer architecture and machine instruction level programming. The basic components of the machine are:

CPU

The CPU consists of a basic ALU capable of performing arithmetic and logic operations on 16-bit words.

Registers

The TINY machine contains sixteen general purpose 16-bit registers: R0, R1, …, R15. The machine uses R15 for the stack pointer (SP).

A 16-bit program counter (PC) contains the address of the next instruction to be executed.

A Program Status Word (PSW) contains three flags (Zero, Negative, and Carry) used to indicate properties of the last arithmetic or logical operation.

Memory

The TINY machine is a stored program computer so the main memory is used for code as well as data. The memory holds of 64K (65535) bytes of data.

TINY is a *little endian* machine, so 16-bit quantities are stored in memory with the least signifiant byte at the lower address.

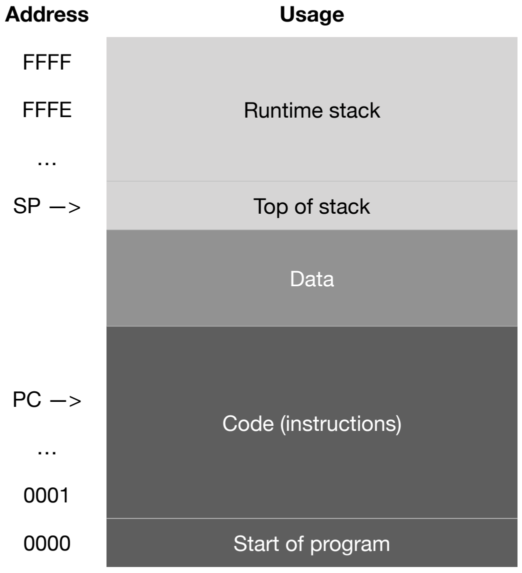
Input/Output

The TINY machine has some very basic I/O capabilities that allow a single character to be read from standard input or written to standard output.

Memory Usage

A stack pointer (SP) register contains the address of the top of the runtime stack. The stack grows down from the highest memory address (0xFFFF) towards lower addresses. The runtime stack is used to keep track of the return location for method calls (by the CALL and RET instructions) as well as for the parameters passing and local variables.

Static (global) data is usually placed between the code and the runtime call stack as shown in the diagram below.



Instructions

Most TINY instructions are single word which includes both the opcode and the operands. A few instructions are two word instructions with the second byte containing an operand (typically a memory address).

Data Transfer

These instructions move data between the main memory and the general purpose registers. The load (LB/LW) and store (STB/STW) instructions use a second instruction word to specify the memory address that is to be read/written. These instructions do not modify the Flags register

Arithmetic

These instructions perform the basic arithmetic operations using the general purpose registers as operands. Flags are set to indicate the status of the arithmetic operation (zero, negative, carry/overflow).

Logic

These instructions perform basic logic operations using the general purpose registers as operands. The logic operation is performed separately on each of the 16 bits in the register(s). Flags are set to indicate the status of the operation (zero, negative).

Branch

These instructions alter the sequence in which instructions are executed allowing for conditional execution and looping. Most of the branch instructions use a second instruction word to specify the address of the next instruction to be executed (if the branch condition is satisfied).

Formats

Instructions consist of an opcode (operation) and up to three operands which are encoded in the various formats described below.

OP	Just the opcode (bits 0..15) and no operands
X	An opcode encoded in bits 0..15 and a 16-bit immediate operand in the second word
R	A single register operand encoded in bits 0..3 The opcode is encoded in bits 4..15
RX	A register encoded in bits 0..3 and a 16-bit immediate operand in the second word The opcode is encoded in bits 4..15
RR	A pair of registers encoded in bits 0..3 and 4..7 The opcode is encoded in bits 8..15
IR	A register encoded in bits 0..3 and an unsigned immediate operand encoded in bits 4..7 The opcode is encoded in bits 8..15
RRR	Three registers encoded in bits 0..3, 4..7 and 8..11 The opcode is encoded in bits 12..15

Timing

Instructions take various amounts of time (clock cycles) to execute which typically depends on just the number of memory references (a few like DIV take longer). The timing for each instruction is given in the description for each opcode.

Move Register — MOV

MOV r_s, r_t

The contents of the register r_s is copied to register r_t.

$$r_t \leftarrow r_s$$

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

Bytes:	2
Cycles:	2
Flags:	none
Opcode:	01xx
Format:	RR

Load Quick **LDQ**

LDQ *r*, value

The unsigned 4-bit immediate value is copied into register *r*. The most significant 12 bits of the register are cleared.

$$r[0..3] \leftarrow \text{value}$$
$$r]4..15] \leftarrow 0$$

0	0	0	0	0	0	1	1	*value*				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

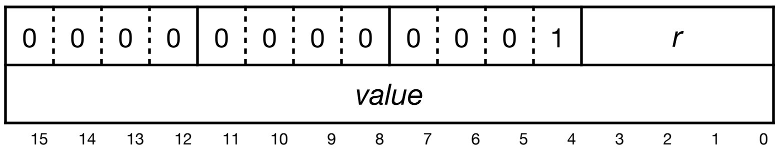
Bytes:	2
Cycles:	2
Flags:	none
Opcode:	03xx
Format:	IR

Load Immediate **LI**

LI *r*, *value*

The 16-bit immediate value contained in the second word of the instruction is copied into register *r*.

$$r \leftarrow \text{value}$$



Bytes:	4
Cycles:	4
Flags:	none
Opcode:	001x
Format:	RX

Load Byte **LB**

LB *r*, *address*

The byte at the memory location given by the second word of the instruction is copied into the low order byte of register *r* and the high order byte of register *r* is set to zero.

$$r[0..7] \leftarrow (\text{address})$$
$$r]8..15] \leftarrow 0$$

0	0	0	0	0	0	0	0	0	0	1	0	*r*			
address															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	4
Cycles:	5
Flags:	none
Opcode:	002x
Format:	RX

Load Word **LW**

LW *r*, *address*

The byte at the memory location given by the second word of the instruction is copied into the low order byte of register *r*. The contents of the next memory location is copied into the high order byte of register *r*.

$$r[0..7] \leftarrow (\text{address})$$
$$r]8..15] \leftarrow (\text{address} + 1)$$

0	0	0	0	0	0	0	0	0	0	1	1	*r*			
address															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	4
Cycles:	6
Flags:	none
Opcode:	003x
Format:	RX

Store Byte STB

STB *r*, *address*

The low order byte of register *r* is stored at the memory location given by the second word of the instruction.

$$(\text{address}) \leftarrow r[0..7]$$

0	0	0	0	0	0	0	0	0	1	0	0	*r*			
address															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	4
Cycles:	5
Flags:	none
Opcode:	004x
Format:	RX

Store Word **STW**

STW *r*, *address*

The low order byte of register *r* is stored at the memory location given by the second word of the instruction. The high order bye of register *r* is stored at the next memory location.

$$(\text{address}) \leftarrow r[0..7]$$
$$(\text{address} + 1) \leftarrow r]8..15]$$

0	0	0	0	0	0	0	0	0	1	0	1	*r*			
address															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	4
Cycles:	6
Flags:	none
Opcode:	005x
Format:	RX

Load Byte Indirect LBX

LBX *r*, r_x, *offset*

The byte at the memory address given by adding the 4-bit unsigned immediate offset to the contents of register r_x is copied into the low byte of register *r*. The high order byte of register *r* is set to zero.

$$r[0..7] \leftarrow (r_x + \text{offset})$$
$$r]8..15] \leftarrow 0$$

0	0	0	1	*offset*				r_x				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	3
Flags:	none
Opcode:	1xxx
Format:	IRR

Load Word Indirect LWX

LWX *r*, r_x, *offset*

The byte at the memory address given by adding the 4-bit unsigned immediate offset to the contents of register r_x is copied into the low byte of register *r*. The byte at the memory next memory location is copied into the high order byte of register *r*.

$$r[0..7] \leftarrow (r_x + \text{offset})$$
$$r]8..15] \leftarrow (r_x + \text{offset} + 1)$$

0	0	1	0	*offset*				r_x				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	4
Flags:	none
Opcode:	2xxx
Format:	IRR

Store Byte Indirect STBX

STBX *r*, r_x, *offset*

The contents of the low order byte of register *r* is stored at the memory address given by adding the 4-bit unsigned immediate offset to the contents of register r_x.

$$(r_x + \text{offset}) \leftarrow r[0..7]$$

0	0	1	1	*offset*				r_x				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

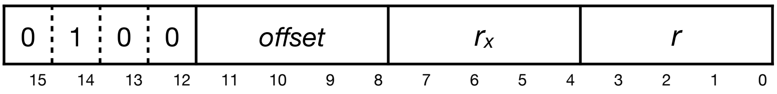
Bytes:	2
Cycles:	3
Flags:	none
Opcode:	3xxx
Format:	IRR

Store Word Indirect **STWX**

STWX *r*, r_x, *offset*

The contents of the low order byte of register *r* is stored at the memory address given by adding the 4-bit unsigned immediate offset to the contents of register r_x. The contents of the high order byte of register *r* is stored at the next memory location.

$$(r_x + \text{offset}) \leftarrow r[0..7]$$
$$(r_x + \text{offset} + 1) \leftarrow r]8..15]$$



Bytes:	2
Cycles:	4
Flags:	none
Opcode:	4xxx
Format:	IRR

Push Register **PUSH**

PUSH *r*

The contents of register *r* is pushed onto the runtime stack.

$$\begin{aligned} (SP-1) &\leftarrow r[8..15] \\ (SP-2) &\leftarrow r[0..7] \\ SP &\leftarrow SP-2 \end{aligned}$$

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

Bytes:	2
Cycles:	4
Flags:	none

Opcode:	00Ax
Format:	R

Pop Register — POP

POP *r*

The value at the top of the runtime stack is popped and stored into register *r*.

$$r[0..7] \leftarrow (SP + 1))$$
$$r[8..15] \leftarrow (SP + 2)$$
$$SP \leftarrow SP + 2$$

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

Bytes:	2
Cycles:	4
Flags:	none
Opcode:	00Bx
Format:	R

Input **IN**

IN *r*

The next byte from standard input is copied into the low order byte of register *r*. The high order byte of register *r* is set to zero.

$$r[0..7] \leftarrow \text{stdin}$$
$$r[8..15] \leftarrow 0$$

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

Bytes:	2
Cycles:	3
Flags:	none

Opcode:	00Cx
Format:	R

Output — OUT

OUT *r*

The low order byte of register *r* is written to standard output.

$$\text{stdout} \leftarrow r[0..7]$$

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

Bytes:	2
Cycles:	3
Flags:	none
Opcode:	00Dx
Format:	R

Negate NEG

NEG r

The contents of register r is replaced with it’s two’s complement.

$$r \leftarrow -r$$

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

Bytes:	2
Cycles:	2
Flags:	Z, N, C
Opcode:	007x
Format:	R

Absolute Value ABS

ABS r

The contents of register r is replaced with it’s absolute value.

$$r \leftarrow |r|$$

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

Bytes:	2
Cycles:	2
Flags:	Z, N, C
Opcode:	008x
Format:	R

Sign Extend **EXT**

EXT *r*

Converts a signed byte to a signed word by propagating the sign bit (bit position 7) of the low order byte to the high order byte (bit positions 8-15) of register.

$$\begin{aligned}
r[8] &\leftarrow r[7]\\
r[9] &\leftarrow r[7]\\
r[10] &\leftarrow r[7]\\
r[11] &\leftarrow r[7]\\
r[12] &\leftarrow r[7]\\
r[13] &\leftarrow r[7]\\
r[14] &\leftarrow r[7]\\
r[15] &\leftarrow r[7]
\end{aligned}$$

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

Bytes:	2
Cycles:	2
Flags:	Z, N, C*
	*cleared

Opcode:	009x
Format:	R

Compare **CMP**

CMP r_1, r_2

The contents of register r_2 is subtracted from the contents of register r_1. The result is discarded after setting the condition codes according to the result of the subtraction.

$$r_2 - r_1$$

0	0	0	0	0	0	1	0	*r1*				*r2*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	2
Flags:	Z, N, C
Opcode:	02xx
Format:	RR

Compare Quick CMPQ

CMPQ r, *value*

The contents of the 4-bit unsigned immediate value is subtracted from the contents of register r. The result is discarded after setting the condition codes according to the result of the subtraction.

$$r - \text{value}$$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	*value*				*r*			

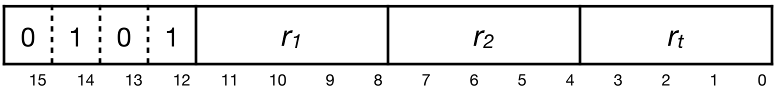
Bytes:	2
Cycles:	2
Flags:	Z, N, C
Opcode:	04xx
Format:	IR

Add ADD

ADD r_1, r_2, r_t

The contents of the register r_2 is added to the contents of register r_1 and the result is placed in register r_t.

$$r_t \leftarrow r_1 + r_2$$



Bytes:	2
Cycles:	2
Flags:	Z, N, C
Opcode:	5xxx
Format:	RRR

Add Quick ADQ

ADDQ r, value

The 4-bit unsigned immediate value is added to the contents of register r.

$$r \leftarrow r + \text{value}$$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	*value*				*r*			

Bytes:	2
Cycles:	2
Flags:	Z, N, C
Opcode:	05xx
Format:	IR

Subtract **SUB**

SUB r_1, r_2, r_t

The contents of the register r_2 is subtracted from the contents of register r_1 and the result is placed in register r_t.

$$r_t \leftarrow r_1 - r_2$$

0	1	1	0	r_1				r_2				r_t			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	2
Flags:	Z, N, C
Opcode:	6xxx
Format:	RRR

Subtract Quick — SUBQ

SUBQ r, *value*

The 4-bit unsigned immediate value is subtracted from the contents of register r.

$$r \leftarrow r - \text{value}$$

0	0	0	0	0	1	1	0	*value*				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	2
Flags:	Z, N, C
Opcode:	06xx
Format:	IR

Multiply **MUL**

MUL r_1, r_2, r_t

The contents of the register r_1 is multiplied by the contents of register r_2 and the result is placed in register r_t.

$$r_t \leftarrow r_1 \times r_2$$

0	1	1	1	r_1				r_2				r_t			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	3
Flags:	Z, N, C
Opcode:	7xxx
Format:	RRR

Multiply Quick MULQ

MULQ r, *value*

The contents of register r is multiplied by the 4-bit unsigned immediate value.

$$r \leftarrow r \times \text{value}$$

0	0	0	0	0	1	1	1	*value*				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	3
Flags:	Z, N, C
Opcode:	07xx
Format:	IR

Divide **DIV**

DIV r_1, r_2, r_t

The contents of the register r_1 is divided by the contents of register r_2 and the quotient is placed in register r_t.

$$r_t \leftarrow r_1 \div r_2$$

1	0	0	0	*r1*				*r2*				*rt*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	4
Flags:	Z, N, C
Opcode:	8xxx
Format:	RRR

Divide Quick **DIVQ**

DIVQ r, *value*

The contents of register r is divided by the 4-bit unsigned immediate value.

$$r \leftarrow r \div \text{value}$$

0	0	0	0	1	0	0	0	*value*				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

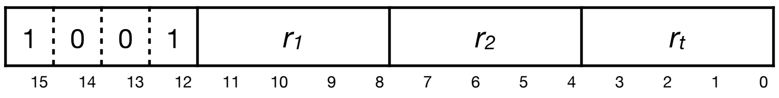
Bytes:	2
Cycles:	4
Flags:	Z, N, C
Opcode:	08xx
Format:	IR

Modulo MOD

MOD r_1, r_2, r_t

The contents of the register r_1 is divided by the contents of register r_2 and the remainder is placed in register r_t.

$$r_t \leftarrow r_1 \,\%\, r_2$$



Bytes:	2
Cycles:	4
Flags:	Z, N, C
Opcode:	9xxx
Format:	RRR

Modulo Quick — MODQ

MODQ r, *value*

The contents of register r is divided by the 4-bit unsigned immediate value and the remainder is placed in register r.

$$r \leftarrow r \ \% \ \text{value}$$

0	0	0	0	1	0	0	1	*value*				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	4
Flags:	Z, N, C
Opcode:	09xx
Format:	IR

Shift (Arithmetic) Left SAL

SAL r_1, r_2, r_t

SLL r_1, r_2, r_t

The contents of the register r_1 is shifted left by the count contained in register r_2 and the result is placed in register r_t. Vacated bit positions are filled with zeros.

$$r_t \leftarrow r_1 << r_2$$

1	1	0	1	r_1				r_2				r_t			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	2
Flags:	Z, N, C
Opcode:	Dxxx
Format:	RRR

Shift (Arithmetic) Left Quick **SALQ**

SLLQ *r*, *count*

SALQ *r*, *count*

The contents of the register *r* is shifted left by the 4-bit unsigned count value. Vacated bit positions are filled with zeros.

$$r \leftarrow r \;\mathbf{<<}\; \text{count}$$

0	0	0	0	1	1	0	1	*count*				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	2
Flags:	Z, N, C
Opcode:	0Dxx
Format:	IR

Shift Arithmetic Right **SAR**

SAR r_1, r_2, r_t

The contents of the register r_1 is shifted left by the count contained in register r_2 and the result is placed in register r_t. Vacated bit positions are filled with the sign bit from register r_1.

$$r_t \leftarrow r_1 \texttt{>>} r_2$$

1	1	1	0	r_1				r_2				r_t			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	2
Flags:	Z, N, C*
	*cleared
Opcode:	Exxx
Format:	RRR

Shift Arithmetic Right Quick **SARQ**

SARQ r, *count*

The contents of the register r is shifted left by the 4-bit unsigned count value. Vacated bit positions are filled with the sign bit from register r.

$$r \leftarrow r \texttt{ >> } \text{count}$$

0	0	0	0	1	1	1	0	*count*				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	2
Flags:	Z, N, C*
	*cleared

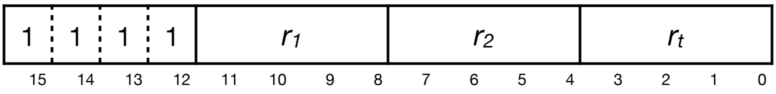
Opcode:	0Exx
Format:	IR

Shift Logical Right SLR

SLR r_1, r_2, r_t

The contents of the register r_1 is shifted right by the count contained in register r_2 and the result is placed in register r_t. Vacated bit positions are filled with zeros.

$$r_t \leftarrow r_1 \mathbin{\texttt{>>>}} r_2$$



Bytes:	2
Cycles:	2
Flags:	Z, N, C*
	*cleared
Opcode:	Fxxx
Format:	RRR

Shift Logical Right Quick **SLRQ**

SLRQ *r*, *count*

The contents of the register *r* is shifted right by the 4-bit unsigned count value. Vacated bit positions are filled with zeros.

$$r \leftarrow r \;\texttt{>>>}\; \text{count}$$

0	0	0	0	1	1	1	1	*count*				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	2
Flags:	Z, N, C*
	*cleared
Opcode:	0Fxx
Format:	IR

Logical Not **NOT**

NOT r_s, r_t

The one’s complement of the contents of register r_s is placed in register r_t.

$$r_t \leftarrow \sim r_s$$

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

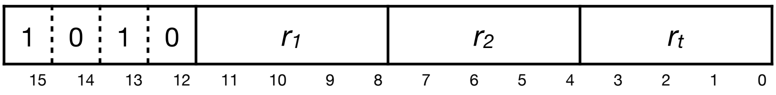
Words:	1
Clocks:	2
Flags:	Z, N, C*
	*cleared

Logical And **AND**

AND r_1, r_2, r_t

The contents of the register r_1 is (bitwise) and'ed with the contents of register r_2 and the result is placed in register r_t.

$$r_t \leftarrow r_1 \mathbin{\&} r_2$$



Bytes:	2
Cycles:	2
Flags:	Z, N, C*
	*cleared
Opcode:	Axxx
Format:	RRR

Logical And Quick — ANDQ

ANDQ *r*, *value*

The contents of the register *r* is and'ed with the 4-bit unsigned immediate value.

$$r \leftarrow r \ \& \ \text{value}$$

0	0	0	0	1	0	1	0	*value*				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

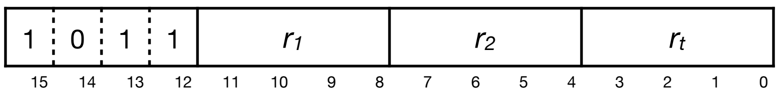
Bytes:	2
Cycles:	2
Flags:	Z, N, C*
	*cleared
Opcode:	0Axx
Format:	IR

Logical Or **OR**

OR r_1, r_2, r_t

The contents of the register r_1 is (bitwise) or'ed with the contents of register r_2 and the result is placed in register r_t.

$$r_t \leftarrow r_1 \mid r_2$$



Bytes:	2
Cycles:	2
Flags:	Z, N, C*
	*cleared
Opcode:	Bxxx
Format:	RRR

Logical Or Quick ORQ

ORQ r, *value*

The contents of the register r is or'ed with the 4-bit unsigned immediate value.

$$r \leftarrow r \mid \text{value}$$

0	0	0	0	1	0	1	1	*value*				*r*			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bytes:	2
Cycles:	2
Flags:	Z, N, C*
	*cleared
Opcode:	0Bxx
Format:	IR

Logical Exclusive Or XOR

XOR r_1, r_2, r_t

The contents of the register r_1 is (bitwise) exclusive or'ed with the contents of register r_2 and the result is placed in register r_t.

$$r_t \leftarrow r_1 \oplus r_2$$

1	1	0	0	r_1				r_2				r_t			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

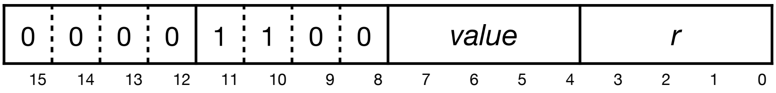
Bytes:	2
Cycles:	2
Flags:	Z, N, C*
	*cleared
Opcode:	Cxxx
Format:	RRR

Logical Exclusive Or Quick **XORQ**

XORQ r, *value*

The contents of the register r is exclusive or'ed with the 4-bit unsigned immediate value.

$$r \leftarrow r \oplus \text{value}$$



Bytes:	2
Cycles:	2
Flags:	Z, N, C*
	*cleared
Opcode:	0Cxx
Format:	IR

Branch (always) B

B *address*

Control is transferred to the instruction located at the address specified by the second word of the instruction.

$$PC \leftarrow \text{address}$$

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

Bytes:	4
Cycles:	4
Flags:	none
Opcode:	0001
Format:	X

Branch Equal BEQ

BEQ	*address*
BE	*address*
BZ	*address*

Control is transferred to the instruction located at the address specified by the second word of the instruction if the Zero flag is set.

if Z **then** $PC \leftarrow \text{address}$

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

Bytes:	4
Cycles:	4
Flags:	none
Opcode:	0002
Format:	X

Branch Not Equal BNE

BNE *address*
BNZ *address*

Control is transferred to the instruction located at the address specified by the second word of the instruction if the Zero flag is not set.

$$\textbf{if } !Z \textbf{ then } PC \leftarrow \text{address}$$

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

Bytes:	4
Cycles:	4
Flags:	none
Opcode:	0003
Format:	X

Branch Less BLT

BLT *address*
BL *address*

Control is transferred to the instruction located at the address specified by the second word of the instruction if the Negative flag is set.

$$\textbf{if } N \textbf{ then } PC \leftarrow \text{address}$$

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

Bytes:	4
Cycles:	4
Flags:	none
Opcode:	0004
Format:	X

Branch Less Equal BLE

BLE *address*

Control is transferred to the instruction located at the address specified by the second word of the instruction if the Zero flag or the Negative flag is set.

$$\textbf{if } N \,|\, Z \textbf{ then } PC \leftarrow \text{address}$$

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

Bytes:	4
Cycles:	4
Flags:	none
Opcode:	0005
Format:	X

Branch Greater **BGT**

BGT *address*
BG *address*

Control is transferred to the instruction located at the address specified by the second word of the instruction if neither the Zero flag nor the Negative flag is set.

$$\textbf{if } !(N\,|\,Z) \textbf{ then } PC \leftarrow \text{address}$$

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

Bytes:	4
Cycles:	4
Flags:	none
Opcode:	0007
Format:	X

Branch Greater Equal BGE

BGE *address*

Control is transferred to the instruction located at the address specified by the second word of the instruction if the Negative flag is not set.

$$\textbf{if } !N \textbf{ then } PC \leftarrow \text{address}$$

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

Bytes:	4
Cycles:	4
Flags:	none

Opcode:	0006
Format:	X

Branch Less (unsigned) **BLTU**

BLTU *address*
BLU *address*
BC *address*

Control is transferred to the instruction located at the address specified by the second word of the instruction if the Carry flag is set.

if C **then** $PC \leftarrow \text{address}$

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

Bytes:	4
Cycles:	4
Flags:	none
Opcode:	0008
Format:	X

Branch Less Equal (unsigned) BLEU

BLEU *address*

Control is transferred to the instruction located at the address specified by the second word of the instruction if the Carry or the Zero flag is set.

$$\textbf{if } C \,|\, Z \textbf{ then } PC \leftarrow \text{address}$$

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

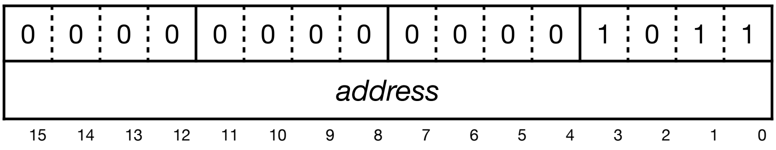
Bytes:	4
Cycles:	4
Flags:	none
Opcode:	0009
Format:	X

Branch Greater (unsigned) BGTU

BGTU *address*
BGU *address*

Control is transferred to the instruction located at the address specified by the second word of the instruction if neither the Carry nor the Zero flag is set.

$$\textbf{if } !(C\,|\,Z) \textbf{ then } PC \leftarrow \text{address}$$



Bytes:	4
Cycles:	4
Flags:	none
Opcode:	000B
Format:	X

Branch Greater Equal (unsigned) **BGEU**

BGEU *address*
BNC *address*

Control is transferred to the instruction located at the address specified by the second word of the instruction if the Carry flag is not set.

$$\textbf{if } !C \textbf{ then } PC \leftarrow \text{address}$$

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

Bytes:	4
Cycles:	4
Flags:	none
Opcode:	000A
Format:	X

Jump (Indirect) JUMP

JUMP *r*

Control is transferred to the instruction located at the address specified by the contents of the register *r*.

$$PC \leftarrow r$$

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

Bytes:	2
Cycles:	2
Flags:	none
Opcode:	00Ex
Format:	R

Halt Execution HALT

HALT

Execution of instructions is suspended; the PC and flags are cleared. The halted flag is set.

$$\begin{aligned} PC &\leftarrow 0 \\ H &\leftarrow \text{true} \end{aligned}$$

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

Bytes:	2
Cycles:	2
Flags:	H, Z*, N*, C*
	*cleared
Opcode:	0000
Format:	OP

Call CALL

CALL *address*

The address of the instruction following the call instruction is pushed onto the top of the stack. Then control is transferred to the instruction located at the address specified in the second word of the instruction.

$$\begin{aligned} (SP-1) &\leftarrow PC[8..15] \\ (SP-2) &\leftarrow PC[0..7] \\ SP &\leftarrow SP-2 \\ PC &\leftarrow \text{address} \end{aligned}$$

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

Bytes:	4
Cycles:	6
Flags:	none
Opcode:	000C
Format:	X

Execute (Call Indirect) EXEC

EXEC *r*

The address of the instruction following the call instruction is pushed onto the top of the stack. Then control is transferred to the instruction located at the address specified by the contents of register *r*.

$$\begin{aligned} (SP-1) &\leftarrow PC[8..15] \\ (SP-2) &\leftarrow PC[0..7] \\ SP &\leftarrow SP-2 \\ PC &\leftarrow r \end{aligned}$$

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

Bytes:	2
Cycles:	4
Flags:	none
Opcode:	00Fx
Format:	R

Return RET

RET

Control is transferred to the address obtained by popping a value off the top of the stack.

$$\begin{aligned} PC[0..7] &\leftarrow (SP + 1)) \\ PC[8..15] &\leftarrow (SP + 2) \\ SP &\leftarrow SP + 2 \end{aligned}$$

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

Bytes:	2
Cycles:	4
Flags:	none
Opcode:	000D
Format:	OP

Register Display REGS

REGS

If the virtual machine is in debugging mode, then then contents of the registers is displayed on standard output.

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

Bytes:	2
Cycles:	2
Flags:	none
Opcode:	000E
Format:	OP

Memory Dump DUMP

DUMP

If the virtual machine is in debugging mode, then then contents of memory is displayed on standard output (lines that are all zeros are skipped).

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

Bytes:	2
Cycles:	2
Flags:	none
Opcode:	000F
Format:	OP

TINY Instruction Set

Quick Reference

Encoding	Opcode	Encoding	Opcode
`0000`	HALT	`01xx`	MOV
`0001`	B	`02xx`	CMP
`0002`	BEQ	`03xx`	LDQ
`0003`	BNE	`04xx`	CMPQ
`0004`	BLT	`05xx`	ADDQ
`0005`	BLE	`06xx`	SUBQ
`0006`	BGE	`07xx`	MULQ
`0007`	BGT	`08xx`	DIVQ
`0008`	BLTU	`09xx`	MODQ
`0009`	BLEU	`0Axx`	ANDQ
`000A`	BGEU	`0Bxx`	ORQ
`000B`	BGTU	`0Cxx`	XORQ
`000C`	CALL	`0Dxx`	SALQ
`000D`	RET	`0Exx`	SARQ
`000E`	REGS	`0Fxx`	SLRQ
`000F`	DUMP	`1xxx`	LBX
`001x`	LI	`2xxx`	LWX
`002x`	LB	`3xxx`	STBX
`003x`	LW	`4xxx`	STWX
`004x`	STB	`5xxx`	ADD
`005x`	STW	`6xxx`	SUB
`006x`	NOT	`7xxx`	MUL
`007x`	NEG	`8xxx`	DIV
`008x`	ABS	`9xxx`	MOD
`009x`	EXT	`Axxx`	AND
`00Ax`	PUSH	`Bxxx`	OR
`00Bx`	POP	`Cxxx`	XOR
`00Cx`	IN	`Dxxx`	SAL
`00Dx`	OUT	`Exxx`	SAR
`00Ex`	JUMP	`Fxxx`	SLR
`00Fx`	EXEC	–	—