

# ANT-8 3.1.0b

## Architecture Reference

February 13, 2003

Copyright 1999-2002 by the President and Fellows of Harvard College.

### 1 Overview

This document describes the architecture of the 8-bit ANT-8 processor.

ANT-8 is a very small and simple processor.

It contains 16 user-visible registers: 14 general-purpose registers and two special registers.

The instruction set consists of 16 instructions.

### 2 ANT-8 Memory Organization

The memory of the ANT-8 processor consists of 256 8-bit bytes. This memory is shared by the instructions and the data.

The ANT-8 architecture is a load/store architecture; the only instructions that can access memory are the *load* and *store* instructions. All other operations access only registers.

### 3 The ANT-8 Register Set

The ANT-8 processor has 16 registers that can be accessed directly by the programmer. In ANT-8 assembler, they are named `r0` through `r15`. In ANT-8 machine language, they are the 4-bit numbers 0 through 15, and are usually written as a single hexadecimal digit (`0x0` through `0xf`).

Registers `r2` through `r15` are general purpose registers. These registers can be used as both the source and destination registers in any of the instructions that use source and destination registers; they are read/write registers.

Registers `r0` and `r1` are not general purpose registers and can be used only as source registers; they are read-only registers. `r0` always contains the constant zero, and `r1` is used to hold values related to the result of the previous instruction. Some instructions do not modify `r1` at all, but several use `r1` to store useful results.

If an instruction attempts to write a value to either `r0` or `r1`, the instruction executes in the normal manner, but no changes are made to the register.

The program counter (or PC) is a special 8-bit register that contains the offset (or index) into memory of the next instruction to execute. Each instruction is 2 bytes long, and each instruction must begin on an even address. Note that the offset is interpreted as an unsigned number and therefore ranges from  $0 \cdots \text{MAX\_UINT8}$ . The PC is not directly accessible to the program.

## 4 Execution of Programs

Programs are executed in the following manner:

### 4.1 Initialization

1. Each location in memory is filled with zero.
2. All of the registers are set to zero.
3. The program counter (PC) is set to zero.
4. The program is loaded into memory from a file.  
See section 6 for information about the program file format.
5. The fetch and execute loop (described in Section 4.2) is executed until the program halts via the `hlt` instruction, or because the execution encounters an error.

Whenever the ANT-8 processor halts due to any error, it dumps core to a file named `ant8.core`.

### 4.2 The Fetch and Execute Loop

1. Fetch the instruction at the offset in memory indicated by the PC.
2. Set  $PC \leftarrow PC + 2$ .
3. Execute the instruction.
  - (a) Get the value of the source registers (if any).
  - (b) Perform the specified operation.
  - (c) Place the result, if any, into the destination register.
  - (d) Update register `r1`, if necessary.
  - (e) Update the PC, if necessary (only for branching or jumping instructions: `beq`, `bgt`, and `jmp` instructions).

## 5 The Instruction Set

### 5.1 Instruction Formats

Most of the instructions have the following general instruction format:

<b>Op</b>	<b>Description</b>	<i>Operator</i> (4 bits)	<i>Register 1</i> (4 bits)	<i>Register 2</i> (4 bits)	<i>Register 3</i> (4 bits)
-----------	--------------------	-----------------------------	-------------------------------	-------------------------------	-------------------------------

The `lc` (load constant), `inc` (increment), and I/O (`in/out`) instructions have the following format:

<b>Op</b>	<b>Description</b>	<i>Operator</i> (4 bits)	<i>Register 1</i> (4 bits)	<i>Constant</i> (8 bits)
-----------	--------------------	-----------------------------	-------------------------------	-----------------------------

The final two exceptions to this are the `ld1` (load) and `st1` (store) instructions, which have the following format:

<b>Op</b>	<b>Description</b>	<i>Operator</i> (4 bits)	<i>Register 1</i> (4 bits)	<i>Register 2</i> (4 bits)	<i>Constant</i> (4 bits)
-----------	--------------------	-----------------------------	-------------------------------	-------------------------------	-----------------------------

The 4 bits of the *Operator* are a 1-digit hexadecimal number that represents the name of the operator or instruction. The 4 bits of the Register(s) are the number of the register (i.e., `0x3` will represent register 3).

### 5.2 Notation

The notation we will use to describe the operands of the instructions is given in Figure 1.

Note that the same register can serve as both a source and destination in one command. For instance, you can double the contents of a register by adding that register to itself and putting the result back in that register, all in one command.

### 5.3 Instruction Descriptions

<b>hlt</b>	<b>Halt</b>	0 0 0 0	unused	unused
------------	-------------	---------	--------	--------

Dump core to `ant8.core`, and halt the processor.

<b>lc</b>	<b>Load constant</b>	0 0 0 1	<i>des</i>	<i>const8</i>
-----------	----------------------	---------	------------	---------------

Figure 1: ANT-8 Machine Language Operand Types

<i>des</i>	Must always be a register index.
<i>reg</i>	Must always be a register index.
<i>src1</i>	Must always be a register index.
<i>src2</i>	Must always be a register index.
<i>const8</i>	Must be an 8-bit constant (MIN_INT8...MAX_INT8).
<i>uconst8</i>	Must be an 8-bit constant (MIN_UINT8...MAX_UINT8).
<i>uconst4</i>	Must be a 4-bit constant integer (MIN_UINT4...MAX_UINT4).

MAX_INT8	127	Maximum signed 8-bit integer.
MIN_INT8	-128	Minimum signed 8-bit integer.
MAX_UINT8	255	Maximum unsigned 8-bit integer.
MIN_UINT8	0	Minimum unsigned 8-bit integer.
MAX_UINT4	15	Maximum unsigned 4-bit integer.
MIN_UINT4	0	Minimum unsigned 4-bit integer.

Load an 8-bit constant into a register.

$R(des) \leftarrow const8$ .

<b>inc</b>	<b>Increment</b>	0 0 1 0	<i>des</i>	<i>const8</i>
------------	------------------	---------	------------	---------------

Increment the value of a register by a constant.

1. Let  $temp \leftarrow R(des) + const8$ .  
 $R(des)$  and  $const8$  are treated as 8-bit 2's complement integers.
2.  $R(des) \leftarrow$  the lower eight bits of  $temp$ .
3. Detect whether overflow/underflow has taken place:  
 $R(1) \leftarrow 1$  if  $MAX\_INT8 < temp$   
 $R(1) \leftarrow 0$  if  $MIN\_INT8 \leq temp \leq MAX\_INT8$   
 $R(1) \leftarrow -1$  if  $temp < MIN\_INT8$

<b>jmp</b>	<b>Jump</b>	0 0 1 1	<i>(ignored)</i>	<i>const8</i>
------------	-------------	---------	------------------	---------------

Jump to a constant address.

1.  $R(1) \leftarrow PC$ .

Note that PC has already been incremented, and so the value left in  $R(1)$  is set to the address of the instruction following the currently executing instruction.

2.  $PC \leftarrow const8$ .

$const8$  is treated as an unsigned 8-bit integer.

**beq**

**Branch if equal**

0 1 0 0	$reg1$	$reg2$	$reg3$
---------	--------	--------	--------

Branch if equal.

1. Branch to  $R(reg1)$  if  $R(reg2)$  is equal to  $R(reg3)$ :  
 $PC \leftarrow R(reg1)$  if  $R(reg2)$  is equal to  $R(reg3)$   
 $PC \leftarrow PC$  otherwise.

**bgt**

**Branch if greater**

0 1 0 1	$reg1$	$reg2$	$reg3$
---------	--------	--------	--------

Branch if greater than.

1. Branch to  $R(reg1)$  if  $R(reg2) > R(reg3)$ :  
 $PC \leftarrow R(reg1)$  if  $R(reg2) > R(reg3)$   
 $PC \leftarrow PC$  otherwise.

The contents of registers  $reg2$  and  $reg3$  are treated as 8-bit two's complement integers.

**ld1**

**Load one byte**

0 1 1 0	$des$	$src1$	$uconst4$
---------	-------	--------	-----------

Load a value from memory into  $R(des)$ .

Note that  $R(src1)$  is treated as an unsigned value ( $MIN\_UINT8 \dots MAX\_UINT8$ ).

1. If  $R(src1) + uconst4 > MAX\_UINT8$ , an "invalid address" error occurs and the processor halts.  $uconst4$  is treated as an unsigned 4-bit integer, while  $R(src1)$  is treated as an unsigned 8-bit value.
2. The 8-bit number located at data memory position ( $R(src1) + uconst4$ ) is loaded into  $R(des)$ .

**st1**

**Store one byte**

0 1 1 1	$reg$	$src1$	$uconst4$
---------	-------	--------	-----------

Store  $R(reg)$  to memory.

Note that  $R(src1)$  is treated as an unsigned value ( $MIN\_UINT8 \dots MAX\_UINT8$ ).

1. If  $R(src1) + uconst4 > MAX\_UINT8$ , an "invalid address" error occurs and the processor halts.  $uconst4$  is treated as an unsigned 4-bit integer, while  $R(src1)$  is treated as an unsigned 8-bit value.
2. Store the contents of  $R(reg)$  into data memory location  $(R(src1) + uconst4)$ .

**add**

**Addition**

1 0 0 0	<i>des</i>	<i>src1</i>	<i>src2</i>
---------	------------	-------------	-------------

8-bit integer addition (with overflow).

1. Let  $temp \leftarrow R(src1) + R(src2)$ .  
 $R(src1)$  and  $R(src2)$  are treated as 8-bit 2's complement integers.
2. Detect whether overflow/underflow has taken place:
 
$$R(1) \leftarrow 1 \text{ if } MAX\_INT8 < temp$$

$$R(1) \leftarrow 0 \text{ if } MIN\_INT8 \leq temp \leq MAX\_INT8$$

$$R(1) \leftarrow -1 \text{ if } temp < MIN\_INT8$$
3.  $R(des) \leftarrow$  the lower eight bits of  $temp$ .

**sub**

**Subtraction**

1 0 0 1	<i>des</i>	<i>src1</i>	<i>src2</i>
---------	------------	-------------	-------------

8-bit integer subtraction (with underflow).

1. Let  $temp \leftarrow R(src1) - R(src2)$ .  
 $R(src1)$  and  $R(src2)$  are treated as 8-bit 2's complement integers.
2. Detect whether overflow/underflow has taken place:
 
$$R(1) \leftarrow 1 \text{ if } MAX\_INT8 < temp$$

$$R(1) \leftarrow 0 \text{ if } MIN\_INT8 \leq temp \leq MAX\_INT8$$

$$R(1) \leftarrow -1 \text{ if } temp < MIN\_INT8$$
3.  $R(des) \leftarrow$  the lower eight bits of  $temp$ .

**mul**

**Multiplication**

1 0 1 0	<i>des</i>	<i>src1</i>	<i>src2</i>
---------	------------	-------------	-------------

Integer multiplication (with overflow).

1.  $temp \leftarrow R(src1) \times R(src2)$ .  
 $R(src1)$  and  $R(src2)$  are treated as 8-bit 2's complement integers.  $temp$  is a 16-bit 2's complement integer.
2.  $R(des) \leftarrow$  the lower 8 bits of  $temp$ .
3.  $R(r1) \leftarrow$  the upper 8 bits of  $temp$ .  
 Note that  $(MIN\_INT8 \times -1)$  leaves  $R(des)$  with  $MIN\_INT8$  and  $R(1)$  with 0.

<b>shf</b>	<b>Bit shift</b>	1 0 1 1	<i>des</i>	<i>src1</i>	<i>src2</i>
------------	------------------	---------	------------	-------------	-------------

Bit shift.

1.  $R(des) \leftarrow R(src1) \ll R(src2)$  if  $R(src2) > 0$   
 $R(des) \leftarrow R(src1) \gg -R(src2)$  otherwise

The contents of registers *src1* and *src2* are treated as 8-bit two's complement integers.

2.  $R(1) \leftarrow 0$ .

Right shifts are done without sign extension, mimicking the behavior of the C operator  $\gg$  for unsigned integers.

<b>and</b>	<b>Bitwise AND</b>	1 1 0 0	<i>des</i>	<i>src1</i>	<i>src2</i>
------------	--------------------	---------	------------	-------------	-------------

Bitwise logical AND.

1.  $temp \leftarrow R(src1) \& R(src2)$  (the bitwise AND of  $R(src1)$  and  $R(src2)$ ).
2.  $R(des) \leftarrow temp$
3.  $R(1) \leftarrow \sim temp$  (the bitwise negation of  $temp$ ).

<b>nor</b>	<b>Bitwise NOR</b>	1 1 0 1	<i>des</i>	<i>src1</i>	<i>src2</i>
------------	--------------------	---------	------------	-------------	-------------

Bitwise logical NOR.

1.  $temp \leftarrow \sim (R(src1) \mid R(src2))$  (the bitwise OR of  $R(src1)$  and  $R(src2)$ ).
2.  $R(des) \leftarrow temp$ ;
3.  $R(1) \leftarrow \sim temp$  (the bitwise negation of  $temp$ ).

<b>in</b>	<b>Input</b>	1 1 1 0	<i>reg</i>	unused	<i>uconst4</i>
-----------	--------------	---------	------------	--------	----------------

The **in** instruction is used to perform input of a single character from a peripheral device.

The 4 bits of the *uconst4* determine the “peripheral” from which the byte is read. The standard Ant-8 implementation has three peripherals:

0000 Hexadecimal – the value is treated as a two-digit hexadecimal number.

0001 Binary – the value is treated as an eight-digit binary number.

0010 ASCII – the value is interpreted as an ASCII character code.

- $R(reg) \leftarrow$  one byte read from the input, in one of the formats.
- $R(1) \leftarrow 1$  if end-of-input (EOI) has been reached, 0 otherwise.

out
-----

## Output

1 1 1 1	unused	reg	uconst4
---------	--------	-----	---------

The `out` instruction is used to perform output of a single character to a peripheral device. `r1` is set to zero.

The 4 bits of the `uconst4` determine the “peripheral” to which the byte is written. The standard Ant-8 implementation has three peripherals:

0000 Hexadecimal – the value is treated as a two-digit hexadecimal number.

0001 Binary – the value is treated as a eight-digit binary number.

0010 ASCII – the value is interpreted as an ASCII character code.

## 6 ANT-8 Executable Files

ANT-8 program files are stored as text, as a sequence of hexadecimal numbers, one per line. Anything that appears after the number on each line is ignored by the program loader, although it may contain information that is used by the debugger. Empty blank lines or lines that begin with a `#` are ignored during the execution of the program, although they may contain information used by the ANT-8 debugger. Each line in a program file must be less than 512 characters in length, and each line must end with a newline.

The program begins with the instructions, which are written as pairs of 8-bit hexadecimal numbers. If any of the hexadecimal numbers that are supposed to represent instructions are too large to fit into 8 bits, then the program is invalid.

The loader reads bytes from the file until either the end of file is reached, or 256 bytes have been read.

If there are fewer than 256 bytes specified in the file, the bytes not specified are implicitly `0x00`.

## Index

`add`  
for Ant-8, 6

`and`  
for Ant-8, 7

`beq`  
for Ant-8, 5

`bgt`  
for Ant-8, 5

`halt` for Ant-8, 3

`in` for Ant-8, 7

`inc`  
for Ant-8, 4

`jmp`  
for Ant-8, 4

`lc`  
for Ant-8, 3

`ld1`  
for Ant-8, 5

`mul`  
for Ant-8, 6

`nor`  
for Ant-8, 7

`out` for Ant-8, 8

`shf`  
for Ant-8, 7

`st1`  
for Ant-8, 5

`sub`  
for Ant-8, 6