

Virtual X68000 Reference Manual

Version 1.1

Kaz Sasayama

Revision 1.21, updated Sat, 16 Dec 2000 00:54:31 +0900.

Copyright © 1999-2000 Hypercore Software Design, Ltd.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Table of Contents

1	Introduction	1
2	Command Reference	2
2.1	Invoking vx68k	2
3	M68000 Architecture	3
3.1	Data Size	3
4	Virtual X68000 Coding Conventions	4
4.1	Basic Types.....	4
5	General Purpose Components	5
5.1	Memory and I/O	5
5.1.1	Class memory	5
5.1.2	Class memory_map	6
5.2	Processor.....	6
5.2.1	Context.....	6
5.2.2	Execution Unit	7
5.2.2.1	Instructions.....	7
6	X68000 Emulation	8
6.1	Machine	8
6.2	Memory Components	8
6.2.1	Main Memory.....	8
6.2.2	The IOCS and the System ROM.....	9
6.2.3	Text Video Memory	9
6.2.4	Graphic Video Memory.....	10
6.2.5	Z8530 SCC.....	10
6.3	Disk I/O.....	10
6.3.1	Floppy Disk.....	10
7	Human68k Emulation	11
7.1	Memory Management.....	11
7.2	Filesystem.....	11
8	GTK+ User Interface	12
9	Sample Application	13
	Appendix A Document Type Definitions	14
A.1	Disk Set Descriptors	14
A.2	Options Setting.....	14
	Function and Variable Index	15

Data Type Index	16
Concept Index	17

1 Introduction

Virtual X68000 is a virtual machine that emulates a Sharp X68000 system.
Virtual X68000 is written in C++ and uses many templates.

2 Command Reference

While Virtual X68000 is written primarily as a set of reusable class libraries, a sample program is also provided for demonstration.

This chapter describes the usage of the sample program.

2.1 Invoking **vx68k**

`vx68k` executes X68000 programs.

`vx68k` [*option*]... [-] *command* [*argument*]...

Options

'-M n'

'--memory-size=n'

Allocate *n* megabytes.

3 M68000 Architecture

Before describing internals of Virtual X68000, let's review the M68000 architecture. Readers who is familiar with this architecture can skip this chapter.

3.1 Data Size

In M68000 architecture, a *byte* is an octet. A *word* is two octets. A *long word* is four octets.

4 Virtual X68000 Coding Conventions

This chapter describes the coding conventions used in the Virtual X68000 programs.

4.1 Basic Types

Virtual X68000 defines several basic types to keep the program portable. These types are defined in namespace `vm68k::types` and imported in namespace `vm68k`.

uint_type Typedef
This is a natural unsigned type that can hold an unsigned word number on M68000. This type is `unsigned int` on all host architectures.

uint32_type Typedef
This is a natural unsigned type that can hold an unsigned long word number on M68000. This type is `unsigned int` on host architectures that have `unsigned int` with at least 32-bit.

sint_type Typedef
`sint_type` is a natural signed type that can hold a signed word number on M68000. This type is `int` on most host architectures, but if the host architecture cannot represent `-0x8000` in that type, i.e. it does not use 2's complement representation, it is `long int` instead.

sint32_type Typedef
`sint32_type` is a natural signed type that can hold a signed long word number on M68000. This type is `int` on host architectures that have `int` with at least 32-bit and that can represent `-0x80000000` in that type. Otherwise, it is `long int` if the type can hold `-0x80000000`, or `long long int` if the compiler is GCC. If no type can hold `-0x80000000` on the architecture, Virtual X68000 cannot be compiled.

5 General Purpose Components

General purpose components are provided by the library `libvm68k`. These components are declared in the namespace `vm68k`.

5.1 Memory and I/O

Memory is an object that can be mapped in an address space. The class `memory` is the abstract base class for all memory.

Virtual X68000 uses a single address space to access memory and peripheral devices.

5.1.1 Class `memory`

memory Abstract Class

This class has methods `get_16`, `get_8`, `get_32`, `put_16`, `put_8`, and `put_32`. The methods `get_16`, `get_8`, `put_16`, and `put_8` are pure virtual and must be overridden in derived classes. Default implementations for `get_32` and `put_32` are provided but a derived class can override those for better performance.

memory::function_code Enumeration

`memory::SUPER_PROGRAM`

`memory::SUPER_DATA`

`memory::USER_PROGRAM`

`memory::USER_DATA`

`uint_type` **get_16** (`function_code fc`, `uint32_type address`) `const` Abstract Method on memory

This method reads a word data from the memory. *address* must be aligned to a word boundary.

`unsigned int` **get_8** (`function_code fc`, `uint32_type address`) `const` Abstract Method on memory

This method reads a byte data from the memory.

`uint32_type` **get_32** (`function_code fc`, `uint32_type address`) `const` Method on memory

`void` **put_16** (`function_code fc`, `uint32_type address`, `uint_type value`) Abstract Method on memory

`void` **put_8** (`function_code fc`, `uint32_type address`, `unsigned int value`) Abstract Method on memory

`void` **put_32** (`function_code fc`, `uint32_type address`, `uint32_type value`) Method on memory

default_memory Class

5.1.2 Class `memory_map`

memory_map	Class
This is a class of address spaces for memory.	
memory_map::function_code	Typedef
Alias of <code>memory::function_code</code> .	
<code>int get_8 (uint32_type address, function_code fc) const</code>	Method on <code>memory_map</code>
<code>uint16_type get_16 (uint32_type address, function_code fc) const</code>	Method on <code>memory_map</code>
<code>uint16_type get_16_unchecked (uint32_type address, function_code fc) const</code>	Method on <code>memory_map</code>
<code>uint32_type get_32 (uint32_type address, function_code fc) const</code>	Method on <code>memory_map</code>
<code>void put_8 (uint32_type address, int value, function_code fc)</code>	Method on <code>memory_map</code>
<code>void put_16 (uint32_type address, uint16_type value, function_code fc)</code>	Method on <code>memory_map</code>
<code>void put_16_unchecked (uint32_type address, uint16_type value, function_code fc)</code>	Method on <code>memory_map</code>
<code>void put_32 (uint32_type address, uint32_type value, function_code fc)</code>	Method on <code>memory_map</code>

5.2 Processor

A processor is made of a pair of a context and an execution unit. A context represents the dynamic state, which is updated by program execution. An execution unit represents the static setting that is not changed while program execution.

5.2.1 Context

The state of the processor is stored in a context. Major components of a context are a set of registers and a reference to an address space.

context	Class
This class represents the dynamic part of a processor.	
regs regs	Instance Variable of context
This variable keeps values of the processor registers.	
<code>bool supervisor_state () const</code>	Method on context
This method returns true if this context is in the supervisor state.	

`void set_supervisor_state (bool state)` Method on context
This method sets the supervisor state to *state*.

`memory::function_code data_fc () const` Method on context
This method returns the function code for data.

`memory::function_code program_fc () const` Method on context
This method returns the function code for programs.

5.2.2 Execution Unit

Virtual X68000 encapsulates non-dynamic aspects of a M68000 processor into an execution unit.

exec_unit Class
This class represents the static part of a processor.

`instruction_type set_instruction (uint16_type op, const instruction_type &i)` Method on exec_unit
This method sets an instruction for operation word *op* to *i* and returns the previous value.

`void step (context &c) const` Method on exec_unit
This method executes a single instruction in context *c*.

`void run (context &c) const` Method on exec_unit
This method executes instructions in context *c*.

5.2.2.1 Instructions

An instruction is defined by a function. This function is called *instruction handler*.

6 X68000 Emulation

This chapter describes UI-independent part of Virtual X68000.

They are available in library 'libvx68k'.

6.1 Machine

In Virtual X68000, a machine is an abstraction of X68000 hardware and firmware BIOS.

These definitions are available in '<vx68k/machine.h>'.

machine	Class
This class represents the user-interface independent part of an X68000 hardware and firmware.	
machine (<i>size_t</i> <i>memory_size</i>)	Constructor on machine
This method constructs a machine.	
<i>size_t</i> memory_size () const	Method on machine
Returns the size of main memory.	
<i>class</i> <i>exec_unit</i> * exec_unit () const	Method on machine
Returns the pointer to the execution unit of this object.	
<i>context</i> * master_context () const	Method on machine
Returns the master context.	
<i>void</i> connect (<i>console</i> * <i>c</i>)	Method on machine
Sets a <i>console</i> for this object. A console is an abstract interface to the host system.	
<i>void</i> configure (<i>memory_map</i> & <i>mm</i>)	Method on machine
Configures address space <i>mm</i> for this object.	
console	Abstract Class
This class is an abstract interface to the host system.	

6.2 Memory Components

6.2.1 Main Memory

main_memory	Class
Memory component for the main memory.	

6.2.2 The IOCS and the System ROM

X68000 uses the *IOCS* for input/output and other primitive services. The name *IOCS* stands for input/output control subsystem. It is stored in the *System ROM*.

system_rom	Class
Memory component for the System ROM. This class derives <code>vm68k::memory</code> .	
This class manages a table of IOCS call handlers, and dispatches IOCS calls to them.	
<code>uint16_type get_16 (function_code fc, uint32_type address) const</code>	Method on <code>system_rom</code>
Returns the 16-bit value at address <i>address</i> using function code <i>fc</i> .	
<code>int get_8 (function_code fc, uint32_type address) const</code>	Method on <code>system_rom</code>
Returns the 8-bit value at address <i>address</i> using function code <i>fc</i> .	
<code>void put_16 (function_code fc, uint32_type address, uint16_type value)</code>	Method on <code>system_rom</code>
Stores 16-bit value <i>value</i> at address <i>address</i> using function code <i>fc</i> .	
<code>void put_8 (function_code fc, uint32_type address, int value)</code>	Method on <code>system_rom</code>
Stores 8-bit value <i>value</i> at address <i>address</i> using function code <i>fc</i> .	
<code>void attach (exec_unit *eu)</code>	Method on <code>system_rom</code>
<code>void detach (exec_unit *eu)</code>	Method on <code>system_rom</code>
<code>void initialize (memory_map &mm)</code>	Method on <code>system_rom</code>
system_rom::iocs_call_type	Typedef
Type for IOCS call handlers.	
<code>void set_iocs_call (int number, const iocs_call_type &handler)</code>	Method on <code>system_rom</code>
Sets the handler for IOCS call <i>number</i> to <i>handler</i> .	
<code>void call_iocs (int number, context &c)</code>	Method on <code>system_rom</code>

6.2.3 Text Video Memory

text_video_memory	Class
Memory component for text video frame buffer.	
<code>uint16_type get_16 (function_code fc, uint32_type address) const</code>	Method on <code>text_video_memory</code>
Returns the 16-bit value at address <i>address</i> using function code <i>fc</i> .	
<code>int get_8 (function_code fc, uint32_type address) const</code>	Method on <code>text_video_memory</code>
Returns the 8-bit value at address <i>address</i> using function code <i>fc</i> .	

<code>void</code> put_16 (<code>function_code</code> <i>fc</i> , <code>uint32_type</code> <i>address</i> , <code>uint16_type</code> <i>value</i>)	Method on <code>text_video_memory</code>
Stores 16-bit value <i>value</i> at address <i>address</i> using function code <i>fc</i> .	
<code>void</code> put_8 (<code>function_code</code> <i>fc</i> , <code>uint32_type</code> <i>address</i> , <code>int</code> <i>value</i>)	Method on <code>text_video_memory</code>
Stores 8-bit value <i>value</i> at address <i>address</i> using function code <i>fc</i> .	
<code>void</code> install_iocs_calls (<code>system_rom</code> & <i>rom</i>)	Method on <code>text_video_memory</code>
<code>void</code> fill_plane (<code>int</code> <i>left</i> , <code>int</code> <i>top</i> , <code>int</code> <i>right</i> , <code>int</code> <i>bottom</i> , <code>int</code> <i>plane</i> , <code>uint16_type</code> <i>pattern</i>)	Method on <code>text_video_memory</code>
Fills a rectangular area in a plane. This method implements the function of an IOCS call <code>_TXFILL</code> .	

6.2.4 Graphic Video Memory

`graphic_video_memory` Class

6.2.5 Z8530 SCC

X68000 uses a Zilog Z8530 SCC, or serial communication controller, for a COM port and a mouse port.

6.3 Disk I/O

`disk_unit` Abtrace Class
Base class for disk units.

<code>uint32_type</code> recalibrate (<code>uint16_type</code> <i>mode</i>)	Abtrace Method on <code>disk_unit</code>
<code>uint32_type</code> seek (<code>uint16_type</code> <i>mode</i> , <code>uint32_type</code> <i>pos</i>)	Abstract Method on <code>disk_unit</code>
<code>uint32_type</code> read (<code>uint16_type</code> <i>mode</i> , <code>uint32_type</code> <i>pos</i> , <code>memory_map</code> & <i>mm</i> , <code>uint32_type</code> <i>buf</i> , <code>uint32_type</code> <i>nbytes</i>)	Abstract Method on <code>disk_unit</code>
<code>uint32_type</code> write (<code>uint16_type</code> <i>mode</i> , <code>uint32_type</code> <i>pos</i> , <code>const memory_map</code> & <i>mm</i> , <code>uint32_type</code> <i>buf</i> , <code>uint32_type</code> <i>nbytes</i>)	Abstract Method on <code>disk_unit</code>
<code>uint32_type</code> verify (<code>uint16_type</code> <i>mode</i> , <code>uint32_type</code> <i>pos</i> , <code>const memory_map</code> & <i>mm</i> , <code>uint32_type</code> <i>buf</i> , <code>uint32_type</code> <i>nbytes</i>)	Abstract Method on <code>disk_unit</code>
<code>uint32_type</code> check (<code>uint16_type</code> <i>mode</i> , <code>int</code> <i>op</i>)	Abstract Method on <code>disk_unit</code>

6.3.1 Floppy Disk

`floppy_disk_unit` Class

7 Human68k Emulation

Virtual X68000 is unique as it also offers functions of the basic operating system, Human68k.

7.1 Memory Management

7.2 Filesystem

8 GTK+ User Interface

This chapter describes the GTK+ implementation of user interface components.

All components described in this chapter is declared in namespace `vx68k::gtk`.

gtk_console

Implements a console using the GTK+ user interface toolkit.

Class

9 Sample Application

This chapter describes the implementation of the sample application `vx68k`.

Appendix A Document Type Definitions

A.1 Disk Set Descriptors

[Insert DTD here.]

A.2 Options Setting

[Insert DTD here.]

Function and Variable Index

A

attach on system_rom..... 9

C

call_iocs on system_rom..... 9

check on disk_unit..... 10

configure on machine..... 8

connect on machine..... 8

D

data_fc on context..... 7

detach on system_rom..... 9

E

exec_unit on machine..... 8

F

fill_plane on text_video_memory..... 10

G

get_16 on memory..... 5

get_16 on memory_map..... 6

get_16 on system_rom..... 9

get_16 on text_video_memory..... 9

get_16_unchecked on memory_map..... 6

get_32 on memory..... 5

get_32 on memory_map..... 6

get_8 on memory..... 5

get_8 on memory_map..... 6

get_8 on system_rom..... 9

get_8 on text_video_memory..... 9

I

initialize on system_rom..... 9

install_iocs_calls on text_video_memory
..... 10

M

machine on machine..... 8

master_context on machine..... 8

memory::SUPER_DATA..... 5

memory::SUPER_PROGRAM..... 5

memory::USER_DATA..... 5

memory::USER_PROGRAM..... 5

memory_size on machine..... 8

P

program_fc on context..... 7

put_16 on memory..... 5

put_16 on memory_map..... 6

put_16 on system_rom..... 9

put_16 on text_video_memory..... 10

put_16_unchecked on memory_map..... 6

put_32 on memory..... 5

put_32 on memory_map..... 6

put_8 on memory..... 5

put_8 on memory_map..... 6

put_8 on system_rom..... 9

put_8 on text_video_memory..... 10

R

read on disk_unit..... 10

recalibrate on disk_unit..... 10

regs of context..... 6

run on exec_unit..... 7

S

seek on disk_unit..... 10

set_instruction on exec_unit..... 7

set_iocs_call on system_rom..... 9

set_supervisor_state on context..... 7

step on exec_unit..... 7

SUPER_DATA of memory..... 5

SUPER_PROGRAM of memory..... 5

supervisor_state on context..... 6

U

USER_DATA of memory..... 5

USER_PROGRAM of memory..... 5

V

verify on disk_unit..... 10

W

write on disk_unit..... 10

Data Type Index

C

console 8
context 6

D

default_memory 5
disk_unit 10

E

exec_unit 7

F

floppy_disk_unit 10
function_code of memory 5

G

graphic_video_memory 10
gtk_console 12

M

machine 8
main_memory 8
memory 5
memory::function_code 5
memory_map 6
memory_map::function_code 6

S

sint_type 4
sint32_type 4
system_rom 9
system_rom::iocs_call_type 9

T

text_video_memory 9

U

uint_type 4
uint32_type 4

Concept Index

C

coding convention	4
COM port	10
context	6

D

disk	10
Disk Operating System	11
DOS	11

E

execution unit	7
----------------------	---

F

filesystem	11
floppy disk	10

G

graphic video memory	10
----------------------------	----

H

Human68k	11
----------------	----

I

input/output control sybssystem	9
instruction handler	7
IOCS	9

M

machine	8
memory	5
memory management, DOS	11
mouse port	10

O

Operating System, Disk	11
------------------------------	----

P

processor	6
-----------------	---

S

SCC, Z8530	10
serial communication controller, Z8530	10
System ROM	9

T

text video memory	9
-------------------------	---

V

video memory, graphic	10
video memory, text	9

Z

Z8530 SCC	10
-----------------	----