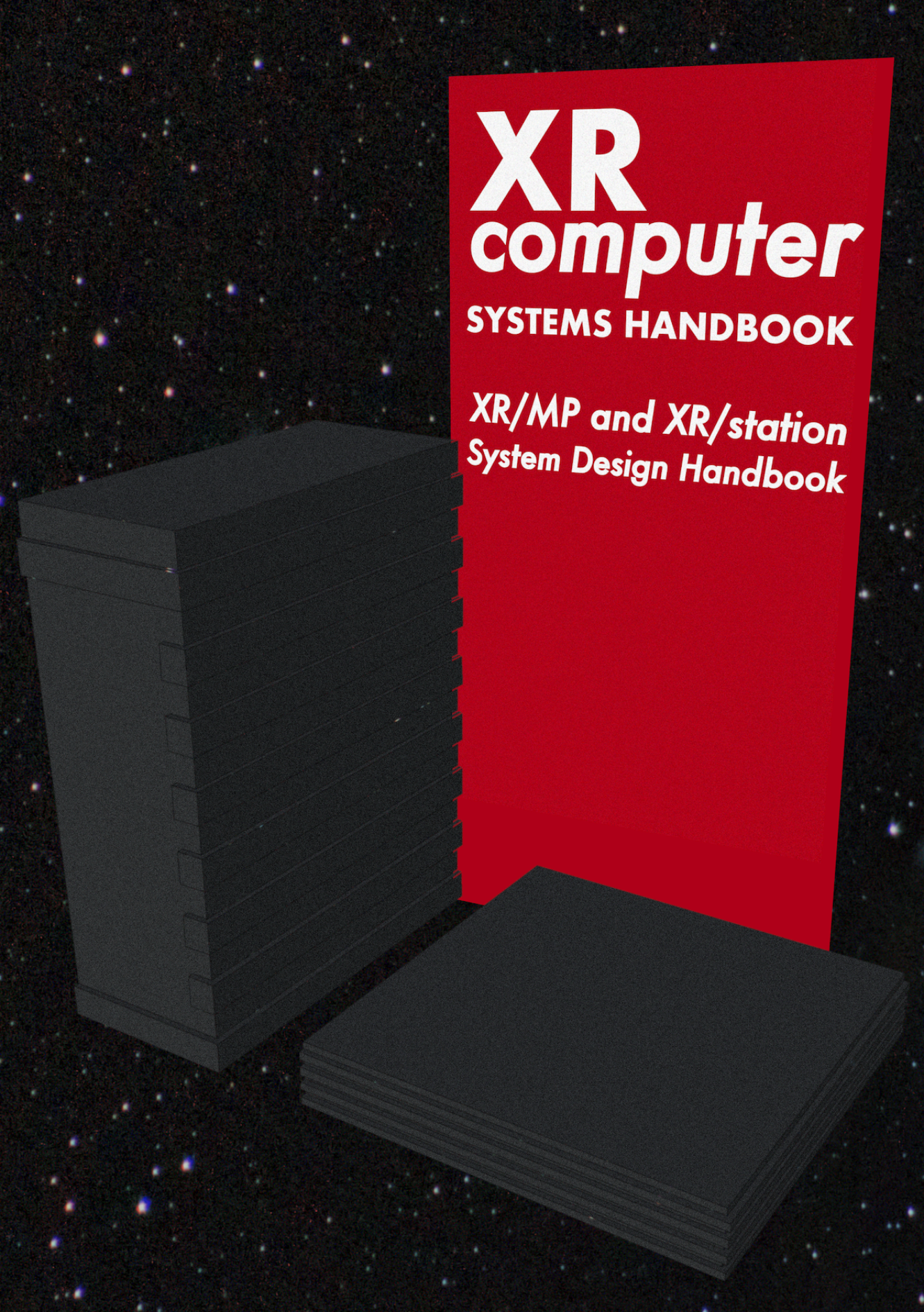
XR computer

SYSTEMS HANDBOOK

XR/MP and XR/station System Design Handbook



XR/station Project
XR/computer Systems Handbook
Revision 1.0, January 5, 2024

Table of Contents

1. Overview

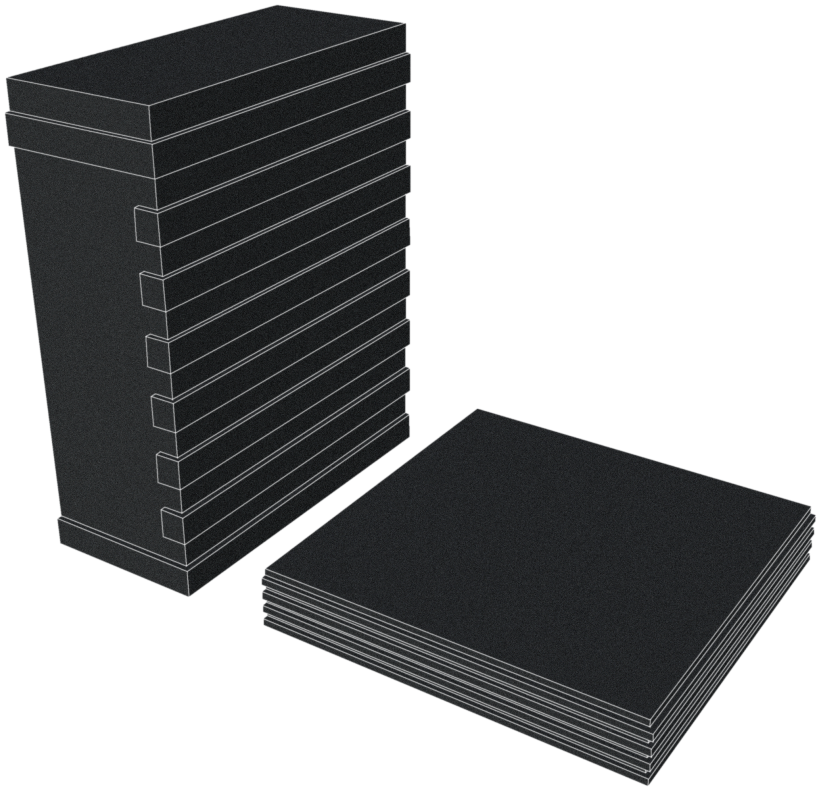
1.1. Introduction

The XR/computer platform is a general hardware design that is shared by the XR/station uniprocessor desktop workstation, and the XR/MP multiprocessor deskside server.

The platform is designed around the 32-bit XR/17032 RISC microprocessor, which itself is described in the *XR/17032 Architecture Handbook*. This document describes the platform as it is seen from the perspective of a system software writer; that is, details such as physical design and electrical characteristics are not discussed.

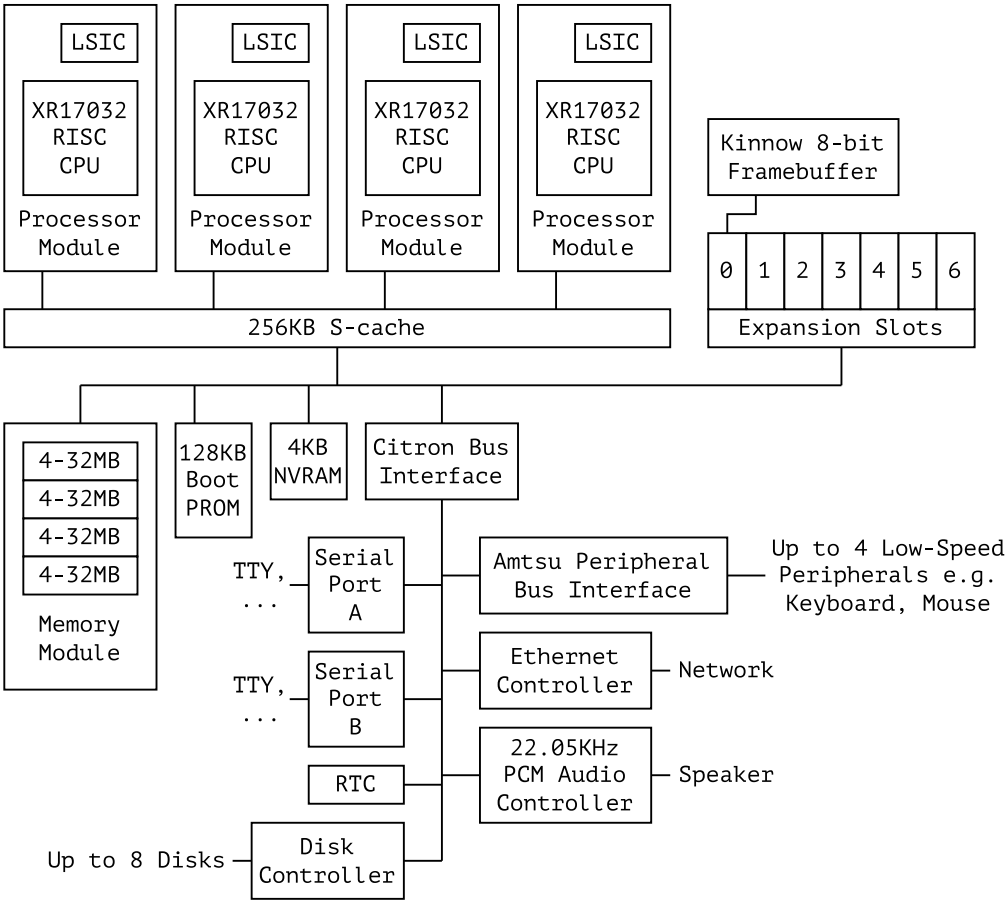
The product range encompassed by the XR/computer platform consists of:

XR/MP	XR/station
Deskside Server	Desktop Workstation
$30K-120K	$8K-30K
2-4 XR/17032 CPUs @ 25 MHz	1 XR/17032 CPU @ 20 MHz
8MB-128MB RAM	4MB-64MB RAM
1-8 200MB-1GB Hard Disks	1-2 200MB-1GB Hard Disks
1024x768 @ 8-bit Color	



1.2. Machine Organization

A simplified diagram of the XR/computer platform follows:



Note that this diagram shows a four-processor XR/MP 1500 model, the maximum extent of what an XR/computer based machine may contain. The actual models implement only a subset of this:

1. XR/station contains only one processor module, and entirely lacks an S-cache. There is only physical space for up to 4 hard disks. Finally, there are only two slots for memory sticks, limiting the machine to 64MB.
2. The XR/MP 1000 model contains only two processor modules, with a 128KB S-cache.

The Kinnow framebuffer card is part of the default configuration of all models, and is therefore described in this document, but can be excluded by order if the customer only requires a headless machine (for a discount!).

Note that since the XR/17032 architecture specifies accesses to physical addresses above 0xC0000000 (3GB) to be noncached, all memory mapped

device registers are placed above this address by this platform. Additionally, the machine is entirely little-endian.

1.3. Memory

There is a memory subsystem which contains up to 4 slots that may each hold one memory stick. A memory stick can have a capacity of 4MB, 8MB, 16MB, or 32MB. The slots are sequentially placed into the physical address space at offsets of 32MB starting at address zero, and the contents of a single stick are presented in a physically contiguous manner beginning at the base of the slot area. That is, the zeroth slot resides at 0x00000000, the first slot resides at 0x02000000, the second is at 0x04000000, etc. This creates a 128MB region in the low physical address space in which memory can be found.

Memory sticks need not be placed into the slots in a manner that is physically contiguous. For instance, it is possible to place a 4MB stick in slot 0 and another 4MB stick in slot 1, leaving a 28MB gap inbetween them in the physical address space. The system software will deal with this correctly. However, slot 0 must always contain some memory for use by the system ROM code.

1.3.1. Probing the Size of Memory

Accesses to empty memory slots will produce bus error exceptions after a short timeout. Therefore, the size of the memory stick in each slot (or absence thereof) can be determined by probing along the slot area until either the end of the slot is reached or a bus error occurs.

Note that the system PROM code will do this automatically, and should be consulted to acquire a map of physical memory at boot time if required.

1.4. System PROM

The 128KB system PROM is placed into the final 128KB of the physical address space (starting at 0xFFFE0000) and contains the reset vector for the XR/17032 microprocessor. This is the first code that runs in the system during startup and is responsible for presenting a simple interface that allows the user to select a boot device, and for booting the operating system.[1]

1.5. NVRAM

Beginning at 0xF8001000, there is a small 4KB battery-backed non-volatile RAM (NVRAM). This is used by the boot firmware to store certain persistent variables, such as the user's preferred boot device.

1.6. S-cache

The secondary cache, or S-cache, is only found on multiprocessor XR/ computer systems.

It is a large cache of recently accessed memory, which is much faster to access than the main DRAM but is still significantly slower than the on-chip primary caches aboard the XR/17032 microprocessors (which can

[1] The *A4X* boot firmware contained within the system PROM is described in the document *A4X Firmware Manual*.

be accessed in a single cycle). It is also used as a single source of truth by the cache coherency protocol.

There is no way to directly access the S-cache, and its existence is completely transparent to system software, as it is kept coherent with external device activity via a snooping write-update scheme. However, the primary caches are not kept coherent with the S-cache or with the rest of the system's autonomous activity (i.e. DMA) and must be manually invalidated by system software in certain situations.

1.7. Expansion Slots

There are 7 slots for expansion cards which can be inserted into an XR/computer system to extend its functionality. Each slot is mapped sequentially into the physical address space at offsets of 128MB beginning at 0xC0000000, and accesses into these 128MB regions are serviced directly by the card. If a card is not present in a slot, an access will result in a short timeout followed by a bus error exception. This can be used to detect if a card is present in a slot or not.

Each slot has only one interrupt line, whose IRQ number is 0x28 + N where N is the slot number from [0, 6]. The usage of this interrupt line is up to the logic on the card. Additionally, the layout and function of the slot space is completely the province of the hardware on the card, except that it must present the following read-only data structure starting at offset zero within its slot space:

```
STRUCT SlotInfo
  // The 32-bit magic number must read 0x0C007CA1.
  Magic : ULONG,

  // The 32-bit board ID number indicates the type of the board.
  BoardId : ULONG,

  // A 15-character, null-terminated string containing a human readable
  // name for the board.
  Name : UBYTE[16],

  // 232 reserved bytes for future expansion.
  Reserved : UBYTE[232],
END
```

The following is a table of the currently defined board identifiers:

	BoardId	Name
Kinnow Framebuffer	0x4B494E36	kinnowfb,8

1.8. Reset Register

Writing the magical 32-bit value 0xAABBCCDD into the "reset register" located at 0xF8800000 will assert the reset line on the bus for several cycles, inducing all devices to enter a quiescent (i.e. non-interrupting) state. Nothing else about the state of the devices may be assumed

except that they will not produce an interrupt until again instructed that they may do so, in whatever device-specific manner. Expansion cards must be sure to respect this.

Note that this is already done by the system PROM at startup time and need not be done again under normal circumstances.

1.9. Revision Register

Reading from the “revision register” located at 0xF8000800 yields a 32-bit revision code for the motherboard which is divided into two 16-bit components. The upper 16 bits indicate the “major” revision, and the low 16 bits indicate the “minor” revision.

2. Interrupt Architecture

2.1. Introduction

Interrupts on the XR/computer platform are mediated between the devices and the processors by the Local Symmetric Interrupt Controller (LSIC), of which there is one for each processor in the system. The XR/17032 microprocessor contains only one IRQ line, which causes a trap to the operating system for interrupt handling when it is asserted. As there is no direct way to determine which device caused the interrupt, an external interrupt controller is required.

2.2. LSIC

The LSIC has 64 interrupt inputs. They are assigned as follows on the XR/computer platform:

0x02	Interval Timer
0x03	Disk Controller
0x04	Serial Port A
0x05	Serial Port B
0x28-0x2E	Expansion Boards
0x30-0x33	Amtsu Devices

Each LSIC has several 32-bit registers which control its behavior. The zeroth LSIC is located at 0xF8030000 in the physical address space, and successive LSICs are arranged at offsets of 32 bytes. In effect, there is an “array” of LSICs which is indexed by the corresponding processor’s number (which can be found in the **WHAMI** control register on that processor, and is in the range of [0-3] on XR/computer systems).

The LSIC registers’ behavior on write and read is enumerated below:

Name	Offset	Function on Write
DISA0	+0x00	Set the disable bits for IRQ0-31.
DISA1	+0x04	Set the disable bits for IRQ32-63.
PEND0	+0x08	Atomically OR the written value into the pending bits for IRQ0-31. However, if the written value is zero, these pending bits are cleared.
PEND1	+0x0C	Atomically OR the written value into the pending bits for IRQ32-63. However, if the written value is zero, these pending bits are cleared.
COMPL	+0x10	Atomically clear the pending bit for the IRQ whose number is written into the register.
IPL	+0x14	Set the current interrupt priority level. Must be of the range [0-63].

Name	Offset	Return on Read
DISA0	+0x00	The disable bits for IRQ0-31.
DISA1	+0x04	The disable bits for IRQ32-63.
PEND0	+0x08	The pending bits for IRQ0-31.
PEND1	+0x0C	The pending bits for IRQ32-63.
CLAIM	+0x10	The current lowest-numbered, unmasked, pending IRQ.
IPL	+0x14	The current interrupt priority level.

The state of the IRQ line is a function of the pending bits, as masked off by the disable bits. Therefore it is given by the following function:

IrqPending := (PEND0 & ~DISA0) ≠ 0 OR (PEND1 & ~DISA1) ≠ 0

With the additional restriction that any interrupt with a number greater than or equal to the value of the **IPL** register is also masked off.

Note that writing a value into the **PEND0** or **PEND1** register of another processor's LSIC can be used to trigger an inter-processor interrupt (IPI) of an arbitrary number, which is useful for tasks such as TB and Dcache shootdown.

2.3. Interrupts

When a device raises an interrupt, it is latched into the corresponding **PEND** register of every LSIC in the system simultaneously. This means that the LSIC interrupts are level-triggered and must be explicitly dismissed by a write of the interrupt number to the **COMPL** register by all processors that took the interrupt.

Note that the input of the device into the LSIC may be either edge-triggered or level-triggered. If it is level-triggered, then clearing the pending bit in the LSIC may have no effect as it will simply be latched again instantaneously if the device is still asserting its interrupt line. In this case, the device must therefore be serviced first in some device-specific manner in order to convince it to drop its interrupt line.

2.4. Interrupt Routing

By "default", all LSICs receive all interrupts. If it is desired that particular interrupts are only serviced by particular processors in the system, then the disable bits for all processors' LSICs should be set such that none will take the interrupt except those which are desired. In this way, arbitrary interrupt routing can be accomplished.

3. Citron Interface

3.1. Introduction

The Citron interface is a small memory-mapped region of 32-bit ports.[2] Many of the integral devices of the XR/computer platform are exposed through this interface. It begins at the physical address 0xF8000000. For example, Citron Port 0x20 would be found at the offset 0x20 * 4 = 0x80 within this space, or 0xF8000080.

3.1.1. Ports

Device ports have some standard behavior in order to simplify drivers somewhat. “Command ports” read zero if the device is completely idle and ready to accept any new command, and a non-zero value with device-specific meaning otherwise. When written, command ports cause the device to perform some action.

“Data ports” may have any device-specific action on reads and writes.

3.1.2. RTC

There is a simple Real Time Clock (RTC) that is responsible for tracking time and for asserting the interval timer interrupt. The time is tracked as a 32-bit Unix epoch timestamp, along with a millisecond part. The current time is stored persistently in a small battery-backed memory. The interval timer can be programmed to periodically interrupt at any 32-bit count of milliseconds.

The IRQ number for the interval timer is 0x02.

The RTC uses two Citron ports, a single command port (0x20) and a single data port (0x21). The data port is readable and writable as a 32-bit datum. The accepted commands are as follows:

#	Function
0x1	Set the timer interval to the number of milliseconds specified in the data port. If zero, the timer interrupt is disabled.
0x2	Reads the current epoch time seconds part into the data port.
0x3	Reads the current epoch time milliseconds part into the data port.
0x4	Sets the current epoch time seconds part from the contents of the data port.
0x5	Sets the current epoch time milliseconds part from the contents of the data port.

3.1.3. Serial Ports

The XR/computer platform supports two serial ports. They are conventionally labeled Serial Port A and Serial Port B.

Each serial port uses two Citron ports, one command port and one data port. The two serial ports are sequential in the Citron port space,

[2] Historically, this was the port I/O space of a previous CISC processor architecture.

beginning at 0x10. For Serial Port A, the command and data ports are 0x10 and 0x11 respectively. For Serial Port B, they are 0x12 and 0x13.

The serial controller collects incoming bytes in a 32-byte receive buffer, and accumulates outgoing characters in a 16-byte transmit buffer. The bytes in the transmit buffer are asynchronously transmitted at 9600 baud.

Reading from the data port will dequeue the next character from the receive buffer. If the receive buffer is empty, 0xFFFF is returned. There is an optional interrupt that is asserted by the serial controller when a character is received.

Writing to the data port will enqueue a character into the transmit buffer. If the transmit buffer is full, reading from the command port will yield a non-zero value. This should be done before attempting to push a character. There is an optional interrupt that is asserted by the serial controller when the transmit buffer has some space available.

The IRQ numbers for serial ports A and B are 0x04 and 0x05 respectively.

The accepted commands are as follows:

#	Function
0x3	Enable the interrupt for this serial port.
0x4	Disable the interrupt for this serial port.

3.1.4. Disk Controller

The XR/computer platform supports up to 8 block addressable devices (such as hard disks) via a simple disk controller. The disk controller has one Citron command port (0x19), and two data ports, data port **A** (0x1A) and data port **B** (0x1B).

The controller performs transfers in units of sectors, which are 512 bytes. One transfer can be in progress to each attached disk simultaneously, and can be up to 8 sectors (4KB) in length. The transfers perform DMA to and from arbitrary sector-aligned physical addresses (i.e. the low 9 bits of the physical address are ignored).

Reading the command port yields a bit set of status bits. The Nth bit where N is a disk number of the range [0, 7] indicates whether that disk is busy or not. This can be used for polled operation of the disk controller. An interrupt can also be made to trigger when transfers complete.

The IRQ number for the disk controller is 0x03.

3.1.4. Disk Controller

The accepted commands are as follows:

#	Function
0x1	Select the disk number specified in data port **A**.
0x2	Start a READ transfer from the selected disk beginning at the sector number specified in data port **A**.
0x3	Start a WRITE transfer to the selected disk beginning at the sector number specified in data port **A**.
0x4	Set data port **B** to a bit set of disks whose transfer has completed since the last time this port was read. Atomically clears the bit set.
0x5	Read information about the disk whose number is in data port **A**. Data port **A** is set to 1 if the disk is present, 0 otherwise. Data port **B** is set to a 32 bit count of sectors in the disk.
0x6	Enable the transfer completion interrupt.
0x7	Disable the transfer completion interrupt.
0x8	Set the sector length for the next transfer from the contents of data port **A**.
0x9	Set the sector-aligned physical address for the next transfer from the contents of data port **A**.

4. Audio Controller

4.1. Introduction

To Be Designed: A 22.05KHz interrupt-driven double-buffered audio controller

5. Ethernet Controller

5.1. Introduction

To Be Designed: A simple Ethernet controller

6. Amtsu Peripheral Bus

6.1. Introduction

The XR/computer platform supports up to 4 low-speed peripheral devices connected to the system via the Amtsu peripheral bus. These devices include things such as mice and keyboards.

The Amtsu bus interface is presented as a set of Citron ports. There are four data ports, **SELECT** (0x30), **MID** (0x31), **A** (0x33), and **B** (0x34). There is one command port (0x32).

The **SELECT** port contains an ID number of the currently selected Amtsu device, of the range [0, 5]. Writing into this port selects a different device. ID 0 is reserved for the command set of the Amtsu controller.

The **MID** port is read-only and contains the Model ID of the currently selected device. This is a unique identifier for the types of peripheral devices.

The **A**, **B**, and command ports are mapped to virtual **A**, **B**, and command ports of the selected peripheral device. Note that these are actually transmitted via a simple protocol over a relatively slow serial connection, and therefore take many more cycles to access than most Citron ports.

Note that when interrupts are enabled for an Amtsu peripheral, the IRQ number is 0x30 + N where N is the device ID.

The following is a table of the currently defined Amtsu model identifiers:

Name	MID
AISA Mouse	0x4D4F5553
AISA Keyboard	0x8FC48FC4

When ID 0 is selected, the Amtsu controller itself accepts commands through the Citron ports. It accepts the following commands:

#	Function
0x1	Enable interrupts from the device number specified in data port **B**.
0x2	Reset the devices on the Amtsu peripheral bus.
0x3	Disable interrupts from the device number specified in data port **B**.

6.2. Keyboard

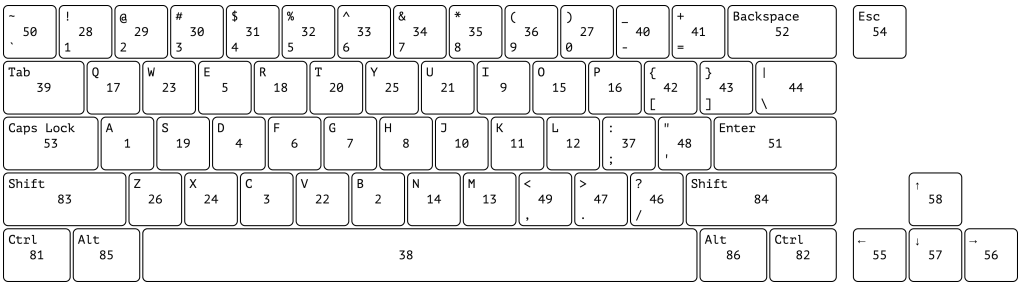
There is a standard keyboard device for the Amtsu bus. The keyboard is a simple input device designed to operate at the speed of a human hand (that is, very slowly relative to the microprocessor).

When the IRQ for a keyboard device is enabled in the Amtsu controller, an interrupt will be signaled whenever a key is pressed or released.

When selected in the Amtsu interface, this device presents several commands:

#	Function
0x1	Pop a scancode from the keyboard into data port **A**. If the 15th bit of the scancode is set, that is, it has been OR'ed with 0x8000, then the key was released and the true scancode is the low 14 bits. Otherwise, it was pressed.
0x2	Reset the keyboard.
0x3	If the scancode in data port **A** is currently pressed, then set data port **A** to 1. Otherwise, set it to 0.

The layout of the keyboard is shown below. Scancodes for each key are labeled in the center of the key:



6.3. Mouse

There is a standard mouse device for the Amtsu bus. The mouse is a simple pointing input device. There are three buttons.

When the IRQ for a mouse device is enabled in the Amtsu controller, an interrupt will be signaled whenever the mouse moves, and whenever one of the buttons is pressed or released.

When selected in the Amtsu interface, this device presents several commands:

#	Function
0x1	Read information from the last event into the data ports. Data port **A** is set to a value that indicates the type of the event. Data port **B** is set to an argument for the event.
0x2	Reset the mouse.

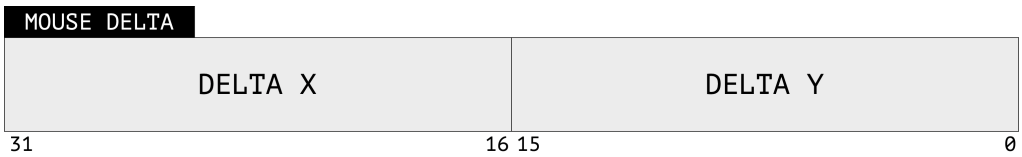
6.3.1. Mouse Events

When command 0x1 is written to the command port, information from the last mouse event is latched into data ports **A** and **B**. The event types reported in data port **A** have the following meaning:

0x1	Button pressed.
0x2	Button released.
0x3	Mouse moved.

When the event type indicates a button press or release, data port **B** reports a number representing the mouse button:

0x1	Left button.
0x2	Right button.
0x3	Middle button.



When the event type indicates mouse movement, the change in mouse position is indicated in a 32-bit value called the “mouse delta” which is latched into data port **B**.

The upper 16 bits of this value contain the change in X position, and the lower 16 bits contain the change in Y position. These are both 16-bit signed (two’s complement) integers. X represents “left-right” and Y represents “up-down”. A negative change indicates a movement to the “left” or “up”, and a positive change represents a movement to the “right” or “down”.

7. Kinnow Framebuffer

7.1. Introduction

The Kinnow framebuffer card provides a very simple linear 1024x768 framebuffer with 8-bit pixels. The slot space is laid out as follows:

Offset	Purpose
+0x000000	SlotInfo
+0x003000	Kinnow Registers
+0x100000	Framebuffer Memory

There are two 32-bit read-only device registers:

Offset	Purpose
+0x0	Display Size
+0x4	Framebuffer Memory Size

The display size register contains two 12-bit fields. The low 12 bits [0:11] contain the width of the display in pixels, and the next 12 bits [12:23] contain the height of the display in pixels. The framebuffer memory size register contains the size in bytes of the framebuffer memory. It will always be at least large enough to contain the display pixel data, calculated by a simple multiplication of width by height as each pixel occupies only 1 byte.

The framebuffer memory is laid out in row-major order. That is, the first row of a 1024 pixel wide display is stored as a contiguous sequence of 1024 bytes, the next row is the next 1024 bytes, and so on. Therefore, the offset for a particular (X,Y) pair can be calculated by:

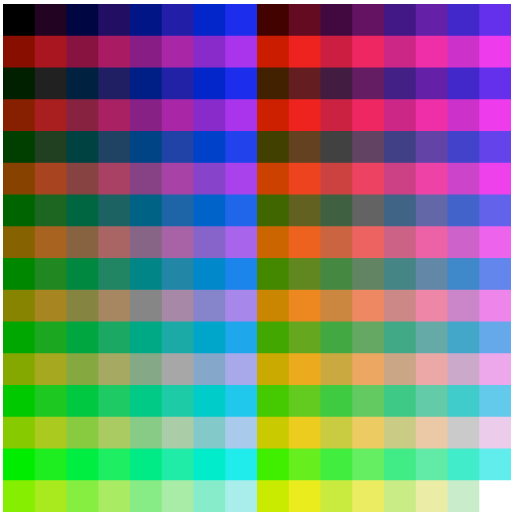
```
Offset := (y * Width) + x
```

And, likewise, an offset can be converted to an (X,Y) pair by:

```
Y = Floor(Offset / Width)
X = Offset % Width
```

7.2. Palette

The pixel color is computed by looking up the 8-bit pixel value in a hardwired palette of 256 colors:



The RGB values are reproduced below:

```
000000 220022 000044 220066 000088 2200aa 0000cc 2200ee 440000 660022
440044 660066 440088 6600aa 4400cc 6600ee 880000 aa0022 880044 aa0066
880088 aa00aa 8800cc aa00ee cc0000 ee0022 cc0044 ee0066 cc0088 ee00aa
cc00cc ee00ee 002200 222222 002244 222266 002288 2222aa 0022cc 2222ee
442200 662222 442244 662266 442288 6622aa 4422cc 6622ee 882200 aa2222
882244 aa2266 882288 aa22aa 8822cc aa22ee cc2200 ee2222 cc2244 ee2266
cc2288 ee22aa cc22cc ee22ee 004400 224422 004444 224466 004488 2244aa
0044cc 2244ee 444400 664422 444444 664466 444488 6644aa 4444cc 6644ee
884400 aa4422 884444 aa4466 884488 aa44aa 8844cc aa44ee cc4400 ee4422
cc4444 ee4466 cc4488 ee44aa cc44cc ee44ee 006600 226622 006644 226666
006688 2266aa 0066cc 2266ee 446600 666622 446644 666666 446688 6666aa
4466cc 6666ee 886600 aa6622 886644 aa6666 886688 aa66aa 8866cc aa66ee
cc6600 ee6622 cc6644 ee6666 cc6688 ee66aa cc66cc ee66ee 008800 228822
008844 228866 008888 2288aa 0088cc 2288ee 448800 668822 448844 668866
448888 6688aa 4488cc 6688ee 888800 aa8822 888844 aa8866 888888 aa88aa
8888cc aa88ee cc8800 ee8822 cc8844 ee8866 cc8888 ee88aa cc88cc ee88ee
00aa00 22aa22 00aa44 22aa66 00aa88 22aaaa 00aacc 22aaee 44aa00 66aa22
44aa44 66aa66 44aa88 66aaaa 44aacc 66aaee 88aa00 aaaa22 88aa44 aaaa66
88aa88 aaaaaa 88aacc aaaaee ccaa00 eeaa22 ccaa44 eeaa66 ccaa88 eeaaaa
ccaacc eeaaee 00cc00 22cc22 00cc44 22cc66 00cc88 22ccaa 00cccc 22ccee
44cc00 66cc22 44cc44 66cc66 44cc88 66ccaa 44cccc 66ccee 88cc00 aacc22
88cc44 aacc66 88cc88 aaccaa 88cccc aaccee cccc00 eecc22 cccc44 eecc66
cccc88 eeccaa cccccc eeccee 00ee00 22ee22 00ee44 22ee66 00ee88 22eeaa
00eecc 22eeee 44ee00 66ee22 44ee44 66ee66 44ee88 66eeaa 44eecc 66eeee
88ee00 aaee22 88ee44 aaee66 88ee88 aaeeaa 88eecc aaeeee ccee00 eeee22
ccee44 eeee66 ccee88 eeeeaa cceecc ffffff
```