# YAMS: Yet Another Machine Simulator

Reference Manual
Edition 1.0.0, for version 1.0.0 of YAMS
26 November 2002

**Juha Aatrokoski, Timo Lilja, Leena Salmela,
Teemu Takanen and Aleksi Virtanen**

# 1 Overview

This manual documents `YAMS` version 1.0.0,

`YAMS` is a machine simulator. It contains simulated CPUs, memory and IO-mapped simulated hardware devices such as disks and consoles.

The intended use of `YAMS` is to provide a platform for operating system implementation courses. `YAMS` is very much like a real machine, but it can be used as a normal UNIX process. `YAMS` has also very simple, but still realistic hardware interface. These features make it an easy platform for OS development.

# 2 Install

Installation instructions can be found in file 'INSTALL'.

# 3 Configuration

## 3.1 Configuration Overview

The configuration files are looked (in this order):

in the current directory file `./yams.conf`

in the home directory file `$HOME/.yams.conf`

in `/etc/yams.conf`

Configuration file consists of four kinds of sections. Sections are separated by the following syntax

```
Section "section-name"
    var val
    ...
EndSection
```

Each `var` is an identifier, consisting of letters. Values `val` are either strings (inside quotation marks) or integer values. Integers can be in decimal notation (the default), or in hexadecimal when they are preceded with "0x". E.g., 1234 is in decimal and 0xFFFF is in hexadecimal notation.

Comments begin with the hash mark '#'. Everything up to the trailing newline will be ignored.

The valid section names are:

"simulator" (Section 3.2 [Configuring the Simulator], page 3.)

"disk" (Section 3.3 [Configuring the Disk], page 4.)

"tty" (Section 3.4 [Configuring the Terminal], page 4.)

"nic" (Section 3.5 [Configuring the Network], page 5.)

The `"simulator"` section is mandatory. Other sections are optional and should be specified only if the corresponding devices are to be included into the simulated machine.

## 3.2 Configuring the Simulator

`cpus INTEGER`

This option specifies the number of CPUs in the simulated machine. This can be an integer from 1 to 256. That, is `YAMS` can support up to 256 different CPUs.

`memory INTEGER`

This option specifies the amount of memory in 4 kilobytes pages. So, for example, if this option is set to 1024, this means that YAMS has totally 1024 * 4KB = 4096 KB of memory.

`clock-speed INTEGER` This option specifies the "clock rate" of YAMS simulator.

All options in this section are mandatory.

## 3.3 Configuring the Disk

`filename` STRING
: This option specifies the file name of the disk image. The simulator reads this disk image or creates new if the file doesn't exist.

`sector-size` INTEGER
: This option specifies the sector size (in kilobytes) of the simulated disk device.

`sectors` INTEGER
: This option specifies the number of sectors in the simulated disk device.

`vendor` STRING
: This option specifies the vendor string of the simulated disk device.

`irq` INTEGER
: This option specifies the IRQ of the simulated device. The valid values are from 0 to 4.

`cylinders` INTEGER
: This option specifies the number of cylinders in the simulated disk device.

`rotation-time` INTEGER
: This option specifies the disk rotation time in milliseconds.

`seek-time` INTEGER
: This option specifies the disk seek time in milliseconds.

The options `irq`, `filename`, `sector-size` and `sectors` are mandatory, other are optional.

## 3.4 Configuring the Terminal

`unix-socket` STRING
: Specifies the UNIX domain socket file to where the `YAMS` will connect its simulated terminal.

`tcp-host` STRING
: Specifies the remote host name (either DNS name or IP address) to the host where to connect the simulated terminal device. `YAMS` will block until a connection is established to the remote host.

: The empty string `""` marks that YAMS should start a listening socket in the local machine. `YAMS` will block until a connection is established to this listening socket.

`port` INTEGER
: Specifies the TCP port where to connect `YAMS` or if empty host name was given in `tcp-host` where to start the listening socket.

`vendor` STRING
: This option specifies the vendor string of the simulated terminal device.

`irq` INTEGER
: This option specifies the IRQ of the simulated device. The valid values are from 0 to 4.

The mandatory options are `irq`, either `unix-socket` or `tcp-host` and `port`.

## 3.5 Configuring the Network

`mtu INTEGER`

This option specifies the mtu, maximum transfer unit of the simulated network interface card (NIC) device.

`unix-socket STRING`

This option specifies the file name of the unix domain socket, in which `YAMS` will connect its simulated NIC device.

`udp-host`

Specifies the remote host name (either DNS name or IP address) to the host where to send the network device packets.

`port`

Specifies the udp port where the network packets will be sent.

`send-delay INTEGER`

Specifies the send delay of the network interface card (NIC). This is in milliseconds.

`mac INTEGER`

This option specifies the MAC (Media Access Control) address of the simulated network device.

`reliability INTEGER`

This option specifies the reliability of the network device. The range is from 0 to 100. The value zero means no reliability (everything is dropped), whereas 100 means total reliability. Note that if the UDP socket is used, 100% reliability is not guaranteed, though.

`dma-delay INTEGER`

This option specifies the delay of the direct memory access (DMA). The unit is milliseconds.

`vendor STRING`

This option specifies the vendor string of the simulated terminal device.

`irq INTEGER`

This option specifies the IRQ of the simulated device. The valid values are from 0 to 4.

The options `mtu`, `irq` and either `unix-socket` or `udp-host` and `port` are mandatory.

## 3.6 Config Example

```
# Simulator config file:

Section "simulator"
  clock-speed        1000          # kHz, "milliseconds" in RTC
                                   #  are based on this
# memory             1024          # in 4 kB pages
  memory             4096
  cpus               1
```

```
EndSection

Section "disk"
  vendor               "1MB-disk"
  irq                  3
  sector-size          1024
  cylinders            4
  sectors              1024
  rotation-time        10            # milliseconds
  seek-time            100           # milliseconds, full seek
  filename             "store.file"
EndSection

Section "tty"
  vendor               "Terminal"
  irq                  4
# unix-socket          "tty0.socket" # path and filename
                                     # to unix domain socket
  tcp-host             ""            # empty hostname means
                                     # listen in TCP port 'port'
  port                 9999
# tcp-host             "localhost"   # connect to this host
# port                 1234
  send-delay           1             # in milliseconds

EndSection

Section "nic"
  vendor               "6Com-NIC"
  irq                  2
  mtu                  1324
  mac                  0x0F010203     # in hex
  reliability          100            # in percents
  dma-delay            1              # in milliseconds
  send-delay           1              # in milliseconds
# unix-socket          "nic0.socket"  # path and filename
                                      # to unix domain socket
  udp-host             "239.255.0.0"  # address of the remote host
  port                 31337          # udp port number
EndSection
```

# 4 Invoking YAMS

The format for running the `YAMS` program is:

`yams` *option* `... [binary-name [`*opt*`] ...]`

`YAMS` supports the following options:

'`binary-name`'
: A binary file to be loaded into the memory at startup (for example an operating system kernel). Binary name may be followed by one or more options, that are passed to the binary.

'`--help`'
'`-h`' Print an informative help message describing the options and then exit.

'`--version`'
'`-v`' Print the version number of `YAMS` on the standard error output and then exit.

'`--config` *file*'
'`-c` *file*' Read configuration file *file*. This will override `YAMS` default configuration searching Section 3.1 [Configuration Overview], page 3.

'`--script` *file*'
'`-s` *file*' Read commands from script *file* and after that drop to interactive prompt. This argument can be given multiple times. Up to 255 different script files are supported. The scripts are executed in the order they are specified in the command line.

# 5 Command Console

When `YAMS` is started for interactive use, the simulation doesn't start automatically. Instead, the system is started into hardware command console. This console can be thought as firmware code that exists in actual hardware.

The main uses of the console are data loading (kernel image loading) and simulator running state control (starting and stopping). In addition to the basic functionality, the console offers some features that are useful for debugging.

When the system is in the console mode a prompt is printed for user. The prompt looks like this:

YAMS [0]>

Console commands can only be entered when command prompt is shown. The number in parenthesis tells the number of hardware clock cycles the system has simulated so far.

The console understands the following commands:

See also:

## 5.1 help

Help command prints a list of available commands. If a command name is given as an argument, extended help for that command is printed instead of the list.

## 5.2 quit

Quit command exits `YAMS`. By default, `YAMS` exists with exit code 0, but if some other code is needed (usually when running scripted tests), exit value in range [0,255] can be given as an argument to the quit-command.

## 5.3 memwrite

Memwrite reads a file and writes it into simulators memory. The first argument to memwrite command must be a valid hardware memory address (for example memory address relative to 0, not a segmented address) where to load the file. The second argument is the name of the file to read in quotation marks.

The following example loads file 'test-binary' into memory starting from address 0x00030000.

```
memwrite 0x00030000 "test-binary"
```

Note that no byte order conversions are done when loading the data. The binary must already be in big-endian byte order.

See Section 5.4 [memread], page 9. See Section 5.16 [numbers], page 12.

## 5.4 memread

Memread reads simulator part of simulator memory and writes it in a file. The first argument to memread command must be a valid hardware memory address (for example memory address relative to 0, not a segmented address) where to start the read from. The second argument is the number of bytes to read. The third argument is the name of the file to be written in quotation marks.

The following example dumps 4 kilobytes (one page) of memory starting from address 0x0003000 to file 'dump-test-file'.

```
memread 0x0003000 4096 "dump-test-file"
```

See Section 5.3 [memwrite], page 8. See Section 5.16 [numbers], page 12.

## 5.5 start

Start command starts the simulation loop. While running the simulation, `YAMS` doesn't take console commands. To return to console and stop the simulation, send interrupt signal to `YAMS` (usually by pressing CTRL-C).

The stopped simulation can be continued with a new start command.

See Section 5.7 [step], page 9.

## 5.6 tlbdump

Tlbdump command prints the contents of translation look-aside buffer for CPU 0. If numeric argument is given to the command, it specifies some other CPU than CPU 0 for printing.

## 5.7 step

Step runs the simulator for one clock cycle and then drops back to the console. If numeric argument is given to step command, given number of clock cycles is simulated before dropping back to the console.

If premature returning is needed, `YAMS` can be forced to drop back to the console by sending interrupt signal (usually by pressing CTRL-C).

See Section 5.5 [start], page 9. See Section 5.16 [numbers], page 12.

## 5.8 break

Break command set hardware breakpoint at the address given as argument to the command. When any CPU in the system loads instruction from the given address, `YAMS` drops to the console.

Only one breakpoint can be active at the same time.

See Section 5.9 [unbreak], page 10. See Section 5.16 [numbers], page 12.

## 5.9 unbreak

Unbreak command clears hardware breakpoints.

See Section 5.8 [break], page 9.

## 5.10 regdump

Regdump command prints contents of CPU and CP0 registers. By default CPU 0 and it's co-processor 0 status is printed. If some print for some other CPU is needed, regdump takes numeric argument which specifies the processor number. Processors are numbered starting from 0.

See Section 5.11 [regwrite], page 10.

## 5.11 regwrite

CPU and CP0 registers can be written with regwrite command. The first argument for the command is the name of the register (register names can be seen with regdump command). The second argument is the new value to store in the given register.

By default CPU 0 registers are affected, but register name can be prefixed by CPU number and colon to store into some other CPU.

Some examples:

```
regwrite s0 0xdeadbeef
regwrite 1:sp 0x00030000
```

See Section 5.10 [regdump], page 10.

## 5.12 interrupt

Hardware and software interrupt lines can be raised with interrupt command. The raising will be valid only for one clock cycle. After that, CPU will automatically clear the interrupt as non-pending.

Interrupt command takes interrupt number as first argument. The second argument specifies the identification number of the CPU which should get the interrupt request. By default all requests go to CPU 0.

The interrupt number number in closed range [0,7]. The meaning of each number is (the numbers correspond to interrupt register bit-fields in CP0):

'0' Software interrupt line 0

'1' Software interrupt line 1

'2' Hardware interrupt line 0

'3' Hardware interrupt line 1

'4' Hardware interrupt line 2

'5' Hardware interrupt line 3

'6' Hardware interrupt line 4

'7' Hardware interrupt line 5

## 5.13 dump

Contents of simulator memory can be seen with the dump command. By default, the command prints 11 words surrounding CPU 0 program counter. This is useful when stepping programs.

Dump takes the beginning address of the dump as an optional first argument. The second, also optional, argument is the number of words to dump. The address argument can be substituted by CPU register name, which may be prefixed by CPU id.

Examples:

```
dump
dump v0
dump 0:v1
dump 0x80010000 20
dump 0:t2 10
```

## 5.14 poke

One word can be written into simulator memory by poke command. Poke takes the memory address as the first argument and value to be stored as second argument. Only full words can be written.

See Section 5.13 [dump], page 11.

## 5.15 boot

Boot command can be used to boot a kernel image. Boot command takes the name of the kernel image file in quotation marks as it's first argument. The second argument is optional quoted string of kernel arguments.

For example, to boot Buenos kernel from "buenos.img" with arguments:

```
boot "buenos.img" "startproc=shell"
```

The exact boot process is:

1. Kernel image file is loaded into memory. This step is equivalent to command `memwrite "image" 0x00010000`.
2. Program counters in all CPUs are set to 0x80010000. This could be done manually by using command `regwrite pc 0x80010000` for each CPU.
3. Kernel argument string is copied into it's memory area. This can't be done without `boot` command.
4. Simulation is started. This step could be done manually with command `start`.

See Section 5.3 [memwrite], page 8. See Section 5.11 [regwrite], page 10. See Section 5.5 [start], page 9.

## 5.16 Entering numbers in the hardware console

When a number is needed as a part of hardware console command (either number of bytes, offset or memory address), `YAMS` always accepts number in either binary, decimal or hexadecimal form.

Decimal numbers (base 10) can be entered the usual way ("1234"). Binary numbers must be prefixed by letter 'b' ("b1010001"). Hexadecimal numbers must be prefixed by either '#' or '0x' ("#a02be").

All numbers must be positive integers in closed range [0, 4294967295] (or 2^32-1). In hex, this range is [#0, #ffffffff] and in binary [b0, b11111111111111111111111111111111].

# 6 Simulated Machine

YAMS simulates a machine with RISC CPUs. The instruction set of the CPU emulates MIPS32 architecture.

Simulation environment simulates an entire computer, including memory, TLB, network interface cards, disks and console devices. Both Direct Memory Access (DMA) and Memory Mapped IO (MMIO) devices are present.

## 6.1 Machine Architecture

## 6.2 CPU

YAMS CPU emulates a big-endian MIPS32 processor. The CPU supports all instructions of the MIPS32 instruction set architecture. The processor also contains a MIPS32 style co-processor 0. See Section 6.3 [CP0], page 15. Coprocessor 1 (Floating Point Unit) is not implemented.

### 6.2.1 CPU registers

| Name | Number | Description |
|---|---|---|
| zero | 0 | Always contains 0 |
| at | 1 | Reserved for assembler |
| v0 | 2 | Function return |
| v1 | 3 | Function return |
| a0 | 4 | Argument register |
| a1 | 5 | Argument register |
| a2 | 6 | Argument register |
| a3 | 7 | Argument register |
| t0 | 8 | Temporary (Caller saves) |
| t1 | 9 | Temporary (Caller saves) |
| t2 | 10 | Temporary (Caller saves) |
| t3 | 11 | Temporary (Caller saves) |

| | | |
|---|---|---|
| t4 | 12 | Temporary (Caller saves) |
| t5 | 13 | Temporary (Caller saves) |
| t6 | 14 | Temporary (Caller saves) |
| t7 | 15 | Temporary (Caller saves) |
| s0 | 16 | Saved temporary (Callee saves) |
| s1 | 17 | Saved temporary (Callee saves) |
| s2 | 18 | Saved temporary (Callee saves) |
| s3 | 19 | Saved temporary (Callee saves) |
| s4 | 20 | Saved temporary (Callee saves) |
| s5 | 21 | Saved temporary (Callee saves) |
| s6 | 22 | Saved temporary (Callee saves) |
| s7 | 23 | Saved temporary (Callee saves) |
| t8 | 24 | Temporary (Caller saves) |
| t9 | 25 | Temporary (Caller saves) |
| k0 | 26 | Reserved for operating system |
| k1 | 27 | Reserved for operating system |
| gp | 28 | Global pointer |
| sp | 29 | Stack pointer |
| fp | 30 | Frame pointer |
| ra | 31 | Return address |
| pc | | Program counter |
| hi | | Register used by multiply and divide instructions |
| lo | | Register used by multiply and divide instructions |

## 6.3 CP0

The CP0 registers discussed below are implemented in `YAMS`. Note that all the registers shown in the `YAMS` hardware console are not implemented. Note also that all the registers are writable through the hardware console. However, illegal values entered through the hardware console can result in unpredictable behavior of `YAMS`.

`YAMS` supports two operating modes, kernel mode and user mode. The processor is in kernel mode when the UM bit in the `Status` register is zero or when the EXL bit in the `Status` register is one or when the ERL bit in the `Status` register is 1. Otherwise the processor is in user mode.

### 6.3.1 Exceptions

When an exception occurs, the following steps are performed by the processor. The `EPC` register and the BD field in `Cause` register are loaded appropriately if the EXL bit in `Status` register is not set. The CE and ExcCode fields in `Cause` register are loaded. The EXL bit in `status` register is set and execution is started at the exception vector. Some exceptions load additional information to CP0 registers.

The base for the exception vector is 0x80000000 if the BEV bit in `Status` register is zero. Otherwise the base is 0xbfc00000.

The exception codes (found in field ExcCode in `Cause` register See Section 6.3.14 [Cause], page 21.) and vector offsets for different exceptions are as follows:

0 0x00
: Interrupt. An interrupt has occurred. If the IV field in `Cause` register is zero, the vector offset is 0x180. Otherwise the vector offset is 0x200.

1 0x01
: TLB modification exception. Software has attempted to store to a mapped address but the D bit in TLB is set indicating that the page is not writable. When this exception occurs, `BadVAd`, `Context` and `EntryHi` registers contain the appropriate bits of the faulting address. The vector offset is 0x180.

2 0x02
: TLB exception (load or instruction fetch). The desired entry either was not in the TLB or it was not valid. When this exception occurs, `BadVAd`, `Context` and `EntryHi` registers contain the appropriate bits of the faulting address. If the entry was not in the TLB and the EXL bit in the `Status` register was zero, the vector offset is 0x000. Otherwise the vector offset is 0x180.

3 0x03
: TLB exception (store). Behaves in exactly the same way as the load or instruction fetch one.

4 0x04
: Address error exception (load or instruction fetch). An address exception occurs when memory reference was unaligned or when an attempt to reference kernel address space is made in user mode. When this exception occurs the faulting address is loaded to the `BadVAd` register. The vector offset is 0x180.

5 0x05
: Address error exception (store). Behaves in exactly the same way as the load or instruction fetch one.

6 0x06
: Bus error exception (instruction fetch). Bus error exception occurs when the bus request is terminated in an error. The vector offset is 0x180.

| | |
|---|---|
| 7 0x07 | Bus errpr exception (load or store). Behaves in exactly the same way as the instruction fetch one. |
| 8 0x08 | Syscall exception. A syscall instruction was executed. The vector offset is 0x180. |
| 9 0x09 | Breakpoint exception. A break instruction was executed. The vector offset is 0x180. |
| 10 0x0a | Reserved instruction exception. An instruction which is not defined was executed. The vector offset is 0x180. |
| 11 0x0b | Coprocessor unusable exception. Software attempted to execute a coprocessor instruction but the corresponding coprocessor is not implemented in `YAMS` or a coprocessor 0 instruction when the processor was running in user mode. The vector offset is 0x180. |
| 12 0x0c | Arithmetic overflow. Arithmetic overflow occurred when executing an arithmetic instruction. The vector offset is 0x180. |
| 13 0x0d | Trap exception. The condition of a trap instruction was true. The vector offset is 0x180. |

### 6.3.2 TLB

`YAMS` TLB contains 16 entries. Each entry contains an even entry and an odd entry. For each pair of entries TLB contains the following fields:

| | | |
|---|---|---|
| VPN2 | 19 bits | The virtual page number is actually virtual page number/2. The even entry maps the page VPN2\|0 and the odd entry VPN2\|1, where \| denotes concatenation of bits. |
| G | 1 bit | The global bit of the entry indicates if the entry is available to all processes. |
| ASID | 8 bits | The address space id field is used to distinguish between entries of different processes. The ASID bit in the TLB entry and in the `EntryHi` register must be the same for the entry to be valid. |

Both the even and the odd entry contain the following fields:

| | | |
|---|---|---|
| PFN | 20 bits | The physical page frame number. |
| C | 3 bits | The cache coherence bits. Since there is no cache in `YAMS`, this field is not very useful. |
| D | 1 bit | Dirty bit. This bit tells whether it is allowed to write to the page. |

| | | |
|---|---|---|
| V | 1 bit | Valid bit. This bit tells if the mapping is valid. |

See Section 6.3.5 [EntLo0 and EntLo1], page 18.

See Section 6.3.11 [EntrHi], page 20.

### 6.3.3 Index

Register number: 0

Selection field: 0

The `Index` register contains the index of the TLB used by the `TLBP`, `TLBWI` and `TLBR`. There are two fields in the `Index` register:

| Field name | Bits | Description |
|---|---|---|
| P | 31 | Probe Failure. This field is written by hardware during the `TLBP` instruction to indicate whether the entry is found in TLB (1) or not (0). This field is not writable by software. |
| Index | 15..0 | The index to the TLB. Written by software to give the TLB index used by `TLBW` and `TLBR` instructions. Written by hardware during the `TLBP` instruction if a matching entry is found. This is a read-write field. |
| | 30...16 | Must be written as zero, returns zero when reading. |

### 6.3.4 Random

Register number: 1

Selection field: 0

The value of `Random` register is used to index the TLB by the `TLBWR` instruction. `Random` register is a read-only register. The `YAMS` hardware updates the value of `Random` register after each `TLBWR` instruction. The value of `Random` register varies between 15 (number of TLB entries minus one) and the lower bound set by the `Wired` register. See Section 6.3.8 [Wired], page 19. At start-up and, when the `Wired` register is written, `Random` register is initialized to its upper bound, 15. There is only one field in the `Random` register:

| Field name | Bits | Description |
|---|---|---|
| Random | 15...0 | The random index to the TLB. This is a read-only field. |
| | 31...16 | Must be written as zero, returns zero when reading. |

### 6.3.5 EntLo0 and EntLo1

Register number: 2 and 3

Selection field: 0

The `EntLo` registers are used in the TLB instructions. The data is either moved from TLB to these registers or vice versa. The fields of `EntLo0` and `EntLo1` registers are the same.

| Field name | Bits | Description |
|---|---|---|
| PFN | 25...6 | Page frame number. This is a read-write field. |
| C | 5...3 | Cache coherency bits. These are not very useful in `YAMS` since there is no cache. This is a read-write field. |
| D | 2 | Dirty bit. The page is writable if this bit is set. Otherwise the page is not writable. This is a read-write field. |
| V | 1 | Valid bit. Indicates whether the entry is valid.This is a read-write field. |
| G | 0 | Global bit. Indicates whether this entry is usable for all processes. When writing an entry to the TLB the G bit has to set in both `EntLo0` and `EntLo1` registers for the G bit to be set in the TLB. This is a read-write field. |
| | 31..26 | Ignored when writing, returns zero when reading. |

### 6.3.6 Contxt

Register number: 4

Selection field: 0

`Context` register can be used by the operating system to reference a page table entry array, if the size of the entry is 16 bytes.

| Field name | Bits | Description |
|---|---|---|
| PTEBase | 31...23 | The base address of the page table entry array. This field should be written by software. |
| BadVPN2 | 22...4 | This field contains the upper 19 bits of the virtual address that caused a TLB exception. This field is written by hardware when a TLB exception occurs and from the software's point of view read-only. |

| | | |
|---|---|---|
| | 3..0 | Must be written as zero, returns zero when reading. |

### 6.3.7 PgMask

Register number: 5

Selection field: 0

The `PgMask` (PageMask) register is used in the MIPS32 architecture to allow variable page sizes. Since `YAMS` only supports 4 kB pages the `PgMask` register is a read-only register containing the value 0.

### 6.3.8 Wired

Register number: 6

Selection field: 0

The `Wired` register specifies the lower bound for `Random` register contents. Thus, TLB indexes less than the `Wired` cannot be replaced with the `TLBWR` instruction. `TLBWI` instruction can be used to replace the wired entries. The `Wired` register is initialized to zero.

There is only one field in the `Wired` register:

| Field name | Bits | Description |
|---|---|---|
| Wired | 15...0 | The boundary of wired TLB entries. |
| | 31...16 | Must be written as zero, returns zero when reading. |

### 6.3.9 BadVAd

Register number: 8

Selection field: 0

The read-only register `BadVAd` is written by `YAMS` when address error, TLB refill, TLB invalid or TLB modified exception occur.

The fields of the `BadVAd` are as follows:

| Field name | Bits | Description |
|---|---|---|
| BadVAddr | 31...0 | Bad virtual address. This field is read-only. |

### 6.3.10 Count

Register number: 9

Selection field: 0

The `Count` register is a timer, which is incremented by `YAMS` on every cycle. The `Count` register is a read-write register.

| Field name | Bits | Description |
| --- | --- | --- |
| Count | 31...0 | Counter. This is a read-write field. |

### 6.3.11 EntrHi

Register number: 10

Selection field: 0

The `EntrHi` register contains the data used for matching a TLB entry when writing to, reading from or accessing the TLB.

| Field name | Bits | Description |
| --- | --- | --- |
| VPN2 | 31...13 | The upper 19 bits of the virtual address. |
| ASID | 7...0 | Address space identifier. |
| | 12...8 | Must be written as zero, returns zero when reading. |

### 6.3.12 Compar

Register number: 11

Selection field: 0

The `Compar` register implements a timer and timer interrupt together with the `Count` register. An interrupt is raised when the values of `Count` and `Compar` registers are equal. The timer interrupt uses interrupt line 5. The timer interrupt is cleared by writing a value to the Compar register.

| Field name | Bits | Description |
| --- | --- | --- |
| Comapre | 31...0 | Counter compare value. This is a read-write field. |

### 6.3.13 Status

Register number: 12

Selection field: 0

The `Status` register contains various fields to indicate the current status of the processor.

| Field name | Bits | Description |
| --- | --- | --- |

| | | |
|---|---|---|
| CU | 28 | Indicates whether access to the co-processor 0 is enabled. This field is initialized to one, indicating access to co-processor 0. This bit is a read-write bit. |
| BEV | 22 | Controls the locations of the exception vectors. The value of this field is zero when normal exception vectors are used and one when bootstrap exception vectors are used. This bit is a read-write bit. |
| IM | 15...8 | Interrupt mask. Controls the enabling of individual interrupt lines. This field is a read-write field. |
| UM | 4 | Indicates the base operating mode of the processor. The encoding is zero for kernel mode and one for user mode. This field is a read-write field. |
| ERL | 2 | The error level field. The value of this field is zero when `YAMS` is operating in normal level and one when `YAMS` is operating in error level. When this bit is set the processor is running in kernel mode, all interrupts are disabled and `ERET` instruction will use the `ErrEPC` instead of the `EPC` register for return address. This field is a read-write field. |
| EXL | 1 | The exception level field. The value of this field is zero when `YAMS` is running in normal level and one when in exception level. When the EXL bit is set, the processor is running in kernel mode, all interrupts are disabled, the TLB Refill exceptions use the general exception vector instead of the TLB Refill vector and the `EPC` register and the `BD` field of the `Cause` register will not be updated. This field is a read-write field. |
| IE | 0 | Interrupt enable. When this bit is zero all interrupts are disabled. |
| | 18 | Must be written as zero, returns zero when reading. |
| | 31..29, 27...23, 21...19, 17...16, 7...5, 3 | Ignored when writing, returns zero when reading. |

### 6.3.14 Cause

Register number: 13

Selection field: 0

The `Cause` register can be used to query the cause of the most recent exception. There are also fields which control software interrupt requests and the entry vector for interrupts.

| Field name | Bits | Description |
|---|---|---|
| BD | 31 | This bit is set if the last exception occurred in branch delay slot. Otherwise this bit is zero. The BD field is not updated if the EXL bit in `Status` register is set. This field is read-only. |
| CE | 29 | This field contains the number of the faulting coprocessor when a coprocessor unusable exception occurs. This field is read-only. |
| IV | 23 | This field can be used to control the entry vector for interrupt exceptions. When this bit is set, interrupt exceptions are vectored to the special interrupt vector (0x200). When this bit is not set, the interrupt exceptions are vectored to the general exception vector (0x180). This is a read-write field. |
| HardIP | 15...10 | This field indicates which interrupts are pending. Bit 15 is for hardware interrupt 5, bit 14 for hardware interrupt 4 and so on. This field is read-only. |
| SoftIP | 9...8 | This field controls the requests for software interrupts. Bit 9 is for software interrupt 1 and bit 8 for software interrupt 0. This is a read-write field. |
| ExcCode | 6...2 | This read-only field contains the exception code. See Section 6.3.1 [Exceptions], page 15. |
| | 22 | Ignored when writing, returns zero when reading. |
| | 30, 27...24, 21...16, 7, 1...0 | Must be written as zero, returns zero when reading. |

### 6.3.15 EPC

Register number: 14

Selection field: 0

The read-write register `EPC` (Exception Program Counter) contains the address at which the execution of a program will continue after an exception is serviced. The `EPC` register contains the virtual address of the instruction that caused the exception or, if that instruction is in branch delay slot, the virtual address of the branch or jump instruction preceding that instruction. When the EXL bit in `Status` register is set, the `YAMS` will not write to the `EPC` register.

| Field name | Bits | Description |
|---|---|---|
| EPC | 31...0 | Exception Program Counter. This is a read-write field. |

### 6.3.16 PRId

Register number: 15

Selection field: 0

The read-only register `PRId` (Processor id) contains information about the processor.

| Field name | Bits | Description |
|---|---|---|
| Processor number | 31...24 | The number of the processor in this installation. The first processor is numbered zero, the second one and so on. The last processor's number is the number of processors minus one. |
| Company ID | 23...16 | The company ID number for `YAMS` is 255. |
| Processor ID | 15...8 | The processor ID number for `YAMS` is 0. |
| Revision | 7...0 | The revision number for `YAMS` is 0. |

### 6.3.17 Conf0

Register number: 16

Selection field: 0

The `Conf0` register is a read-only register providing information about the processor. All fields of the `Conf0` register are constant.

| Field name | Bits | Description |
|---|---|---|
| M | 31 | Tells that Conf1 is implemented at a select field 1. The value of this field is one. |
| BE | 15 | Denotes the endianess of the processor. The value of this field is one for a big endian processor like `YAMS`. |
| AT | 14...13 | Indicates that `YAMS` emulates the MIPS32 architecture. The value of this field is zero. |

| | | |
|---|---|---|
| AR | 12...10 | Indicates that `YAMS` emulates the revision 1 of the MIPS32 architecture. The value of this field is zero. |
| MT | 9...7 | Indicates the MMU type used by `YAMS`. Since `YAMS` emulates the standard TLB model of the MIPS32 architecture, the value of this field is one. |

### 6.3.18 Conf1

Register number: 16

Selection field: 1

The read-only `Conf1` register provides more information about the capabilities of the processor.

| Field name | Bits | Description |
|---|---|---|
| M | 31 | The value of this field is zero indicating that there is no `Conf2` register. |
| MMU size | 30...25 | Number of TLB entries minus one. Thus 15 for `YAMS`. |
| IS | 24...22 | Icache setes per way. The value of this field is zero, because `YAMS` does not support caches. |
| IL | 21...19 | Icache line size. The value of this field is zero, because `YAMS` does not support caches. |
| IA | 18...16 | Icache associativity. The value of this field is zero, because `YAMS` does not support caches. |
| DS | 15...13 | Dcache sets per way. The value of this field is zero, because `YAMS` does not support caches. |
| DL | 12...10 | Dcache line size. The value of this field is zero, because `YAMS` does not support caches. |
| DA | 9...7 | Dcache associativity. The value of this field is zero, because `YAMS` does not support caches. |
| C2 | 6 | Indicates whether co-processor 2 is implemented. The value of this field is zero, because `YAMS` does not support co-processor 2. |
| PC | 4 | Indicates whether performance counter registers are implemented. The value of this field is zero, because `YAMS` does not support perforamnce counter registers. |

| | | |
|---|---|---|
| WR | 3 | Indicates whether watch registers are implemented. The value of this field is zero, because `YAMS` does not support watch registers. |
| CA | 2 | Indicates whether code compression is implemented. The value of this field is zero, because `YAMS` does not support code compression. |
| EP | 1 | Indicates whether EJTAG is implemented. The value of this field is zero, because `YAMS` does not support EJTAG. |
| FP | 0 | Indicates whether FPU is implemented. The value of this field is zero, because `YAMS` does not support FPU. |

### 6.3.19 LLAddr

Register number: 17

Selection field: 0

The `LLAddr` register contains the physical address of the most recent LL instruction. This register is not used by software in normal operation.

### 6.3.20 ErrEPC

Register number: 30

Selection field: 0

The read-write register `ErrEPC` functions like the `EPC` register except that it is used on error exceptions.

| Field name | Bits | Description |
|---|---|---|
| ErrorEPC | 31...0 | Error exception program counter |

## 6.4 Memory

### 6.4.1 Architecture

`YAMS` provides 2^32 bytes virtual address space. Virtual address space is divided in five segments shown in table below.

| Virtual Address Range | Size | Description |
|---|---|---|
| 0xFFFFFFFF - 0xE0000000 | 512 MB | Kernel Mapped |
| 0xC0000000 - 0xDFFFFFFF | 512 MB | Supervisor Mapped |

| | | | |
|---|---|---|---|
| 0xA0000000 - 0xBFFFFFFF | | 512 MB | Kernel Unmapped Uncached |
| 0x80000000 - 0x9FFFFFFF | | 512 MB | Kernel Unmapped |
| 0x00000000 - 0x7FFFFFFF | | 2 GB | User Mapped |

Addresses in mapped segments are translated through TLB. Unmapped kernel segments generate physical addresses to lowest 512 MB of physical memory. Cache is not implemented so there are no differences between the two kernel unmapped segments.

### 6.4.2 Kernel Unmapped Uncached Segment

Kernel unmapped segment is further divided into following memory areas:

| Virtual Address Range | Description |
|---|---|
| 0xB0008000 - 0xBFFFFFFF | Memory mapped device IO-area |
| 0xB0001000 - 0xB0001FFF | Kernel Boot parameters string |
| 0xB0000000 - 0xB0000FFF | Device descriptors |
| 0xA0000000 - 0xAFFFFFFF | Kernel binary and stack |

### 6.4.3 Accessing segments

All segments are accessible while processor is in kernel mode. In user mode only User mapped segment is accessible. Accessing other segments generates Address Error. Since no supervisor mode is implemented Supervisor Mapped segment is accessible only in kernel mode.

### 6.4.4 Address Translation

Unmapped Segments

Kernel Unmapped segments generate physical addresses in the following way:

| Segment | Virtual Address | Physical Address |
|---|---|---|
| Kernel Unmapped Uncached | 0xA0000000 - 0xAFFFFFFF | 0x00000000 - 0x0FFFFFFF |
| Kernel Unmapped | 0x80000000 - 0x9FFFFFFF | 0x00000000 - 0x1FFFFFFF |

Addresses 0xB0000000-0xBFFFFFFF are used in memory mapped io-devices and are not threated as normal physical memory.

TLB-address translation

See Section 6.3.11 [EntrHi], page 20.

See Section 6.3.5 [EntLo0 and EntLo1], page 18.

See Section 6.3.2 [TLB], page 16.

See Section 6.3.1 [Exceptions], page 15.

Memory is mapped in 4096-byte pages in `YAMS`. Bits 31-12 of the virtual address refer to the page. Bits 11-0 are used indexing inside the page. Address translation is performed in the following way:

1 Find TLB-entry, that VPN2 field matches to the bits 31-13 of the virtual address and G bit is set or ASID field matches the current process ASID (obtainde from EntryHi register). If TLB-entry is not found and reference type is load raise TLB exception 0x02. Otherwise, if TLB-entry is not found, raise TLB exception 0x03.

2 Check bit 12 (EvenOddBit) in virtual address. If zero use mapping for even page (TLB-entry fields PFN0, C0, D0, V0), otherwise use mapping for even page (TLB-entry fields PFN1, C1, D1, V1).

3 Check validity bit of page (V-field of even/odd page mapping in TLB-entry). If one page is valid and accesses to the page are permitted. Otherwise raise TLB exception 0x02 (ref. type load) or 0x03 (ref. type store).

4 Check dirty bit of page (D-field of even/odd page mapping in TLB-entry). If zero and reference type is store, raise TLB modification exception 0x01.

5 Generate physical address by concatenating bits 19-0 of PFN-field and bits 11-0 of virtual address.

## 6.5 Boot Sequence

## 6.6 Memory mapped I/O devices

All I/O operations in `YAMS` are memory-mapped. The I/O address space is the upper half of the kernel unmapped uncached segment, ie. the first byte is at the address 0xb0000000 and the last byte at 0xbfffffff. Reads or writes to this area will not cause an exception provided that the CPU is in kernel mode and the read/write is naturally aligned.

Reads from unused portions of the I/O area return 0. However, the operating system should not rely this to be so and instead consider the result is undefined. Writes to unused portions have no effect.

Reads from the I/O area function just as normal memory reads. However, writing anything other than a word (e.g. a byte or a half word) to an I/O 'port' of a device will give unpredicted results. So writing to I/O ports should be restricted to whole words.

The I/O address space is partitioned as follows:

0xb0000000 - 0xb0000fff
: This area holds the 128 device descriptors which describe the hardware devices that are available in the system. For details: See Section 6.6.1 [Device descriptors], page 28. This area is read-only, meaning that writes have no effect.

0xb0001000 - 0xb0001fff
: This area holds the kernel boot parameters as a 0-terminated (C-style) string. This area is read-only, meaning that writes have no effect.

0xb0002000 - 0xb0007fff
: This area is reserved for future use. This area is read-only, meaning that writes have no effect.

0xb0008000 - 0xbfffffff
: This area holds the actual I/O ports (memory areas) for the devices. Whether writing to a certain address has any effect depends on the device and port in question.

Each of the I/O devices is documented in the following sections:

### 6.6.1 Device descriptors

In the memory range from 0xb0000000 to 0xb0000fff are located 128 device descriptors that describe the hardware devices. Each of the descriptors is 32 bytes long and has the following structure:

| OFFSET | SIZE | DESCRIPTION |
|---|---|---|
| 0x00 | 1 word | Device type code. Type code 0 is not used by any devices, and it means that this descriptor is unused and should be ignored. See Section 6.6.2 [Device type codes], page 28. |
| 0x04 | 1 word | Device I/O address base. All I/O port offsets of a device are relative to this address. |
| 0x08 | 1 word | The length of the device's I/O address area in bytes. This will always be a multiple of 4, since all ports are 32 bits wide. |
| 0x0C | 1 word | The number of the IRQ that the device generates. Possible values are from 0 to 5. A value of -1 (0xffffffff) means the device will not generate any IRQs. |
| 0x10 | 8 bytes | Vendor string. These bytes are used to describe the model of the device or some other information intended to be read by humans. The operating system may safely ignore the contents of these bytes. These bytes may contain any values and need not be 0-terminated. |
| 0x18 | 2 words | Reserved. The contents of these word should be considered undefined. |

When starting, the operating system should read through *all* device descriptors, ignoring those with device type code of 0. In practise there will be no more devices after the first descriptor with type code 0, but the OS must not rely on this as it may very well change in the future.

### 6.6.2 Device type codes

A device is identified by its type code. The type codes have the following meaning and grouping:

| | |
|---|---|
| 0x100 | The 0x100 series is for so-called meta-devices, such as those that are integrated into the motherboard chipset. |
| 0x101 | System memory information. See Section 6.6.4 [Meminfo], page 29. |
| 0x102 | System real-time clock device (RTC). See Section 6.6.5 [RTC], page 29. |
| 0x103 | System software shutdown device. See Section 6.6.6 [Shutdown], page 30. |
| 0x200 | The 0x200 series is for TTYs and other character-buffered devices. |
| 0x201 | the basic TTY as described in this document. See Section 6.6.8 [Terminals], page 30. |
| 0x300 | The 0x300 series is for disks and other block-buffered devices. |
| 0x301 | Hard disk as described in this document. See Section 6.6.9 [Disks], page 32. |
| 0x400 | The 0x400 series is for network devices. |
| 0x401 | NIC as described in this document. See Section 6.6.10 [NIC], page 33. |
| 0xC00 | CPU status "devices". The last two hexadecimal digits indicate the number of the CPU, from 0 to 255. See Section 6.6.7 [CPU status], page 30. |

### 6.6.3 Hardware interrupts

Interrupts (IRQs) caused by hardware devices are distributed evenly to all CPUs since they are not CPU specific (unlike other exceptions).

If `YAMS` is configured with more than one CPU, the operating system *must* support all processors and initialize them symmetrically or some device IRQs may be lost (more correctly never noticed or handled rather than lost).

See Section 3.2 [Configuring the Simulator], page 3.

### 6.6.4 System memory information device

The system memory information device has device type code 0x101 and it has the following port:

| OFFSET | NAME | R/W | DESCRIPTION |
|---|---|---|---|
| 0x00 | PAGES | R | This port contains the number of physical memory pages in the system. Each page is 4096 bytes (4kB) in size. |

### 6.6.5 System real-time clock device

The RTC device (device type code 0x102) contains information about the speed and uptime of the system. It has the following ports:

| OFFSET | NAME | R/W | DESCRIPTION |
|---|---|---|---|
| 0x00 | MSEC | R | Milliseconds elapsed since the machine started. |

| | | | |
|---|---|---|---|
| 0x04 | CLKSPD | R | Machine clock speed in Hz. |

The milliseconds are calculated from elapsed clock cycles and the simulator's virtual clock speed, and have no relatioin whatsoever with real word time.

See Section 3.2 [Configuring the Simulator], page 3.

### 6.6.6 Software shutdown device

The software shutdown device (device type code 0x103) is used to exit from `YAMS` from within the running program (OS). It has the following port:

| OFFSET | NAME | R/W | DESCRIPTION |
|---|---|---|---|
| 0x00 | SHUTDN | W | Writing the magic word to this port will shut down the machine. |

The magic word is 0x0badf00d. Writing the magic word to the port will immediately (after the clock cycle is finished) cause the simulator to exit.

### 6.6.7 CPU status devices

Each CPU in the system has a status metadevice associated with it. The device type codes for CPU status devices range from 0xC00 to 0xCFF, the last two hexadecimal digits indicating the number of the CPU. The device has the following port:

| OFFSET | NAME | R/W | DESCRIPTION |
|---|---|---|---|
| 0x00 | STATUS | W | CPU status word. |

Presently the status word consists only of the lowest order bit (0-bit), which is set when the CPU is running. Since all existing CPUs are always running, the status word is always 1.

**Caution:** Since the maximum number of device descriptors is 128, configuring `YAMS` with too many processors will cause undesirable effects.

See Section 3.2 [Configuring the Simulator], page 3.

### 6.6.8 Terminal devices

Only terminals with device type code 0x201 are covered in this section.

A terminal (TTY) is a character buffered I/O device from which data can be read when it is available and to which data can be written in certain speed. Reads and writes are done one byte (character) at a time. A terminal device has the following ports:

| OFFSET | NAME | R/W | DESCRIPTION |
|---|---|---|---|
| 0x00 | STATUS | R | Status bits for the TTY device. |
| 0x04 | COMMAND | W | Port for giving commands to the TTY device. |

| | | | |
|---|---|---|---|
| 0x08 | DATA | RW | Data port for reading from and writing to the TTY. Only the 8 lowest bits are used. |

Operating the TTY is based mostly on interpreting the status bits, which are described in the following table. Reading from or writing to DATA will update the status bits before the next clock cycle.

| BIT | NAME | DESCRIPTION |
|---|---|---|
| 0 | RAVAIL | There is meaningful (read: real) data available in DATA. If this bit is not set, reads from DATA will return 0. |
| 1 | WBUSY | The TTY is writing out its internal buffer. When this bit is set, all writes to DATA will be ignored. |
| 2 | RIRQ | The TTY has generated an IRQ because there is new data available. |
| 3 | WIRQ | The TTY has generated an IRQ because WBUSY has been cleared. |
| 29 | ICOMM | The last command issued to the COMMAND port was unrecognized. |
| 30 | EBUSY | The last command issued to the COMMAND port could not be handled because the TTY was busy. |
| 31 | ERROR | Undefined error in the device. The TTY is to be considered unusable if this bit is set. |

The following commands are available to control a TTY device:

0x01 Reset RIRQ. Will zero the RIRQ bit, indicating that the IRQ generated has been handled.

0x02 Reset WIRQ. Acts similarly to the RIRQ resetting.

Reading from a TTY device by the operating system would typically be done as follows.

When there is input data available, the TTY will raise an IRQ. The handler should check just in case that RAVAIL is really set (should always be if RIRQ is set) before reading. It will then read one byte from DATA into its own buffer. After reading the byte, it should check if more data is available by checking the RAVAIL bit. Data can be read as long as RAVAIL is set, and all of it should be read too or the IRQ will be raised again after exiting the handler. When all available data is read, the handler should reset the RIRQ bit (command 0x01).

Writing to a TTY device would typically be implemented by the OS as follows.

First check WBUSY. If WBUSY is set, the thread should go to sleep. When WBUSY is cleared an interrupt is raised. The handler should wake up the writing thread and reset WIRQ (command 0x02). The writing thread should write the output one byte at a time as long as WBUSY is not set. When WBUSY becomes set, the thread should go to sleep again. This cycle is repeated until all output is written.

See Section 3.4 [Configuring the Terminal], page 4.

### 6.6.9 Hard disk devices

Only disks with device type code 0x301 are covered in this section.

A disk device transfers data between disk and memory using DMA. It generates interrupts when it has completed a DMA transfer. The data is stored on an image file in the directory from where `YAMS` is run.

A disk device has the following I/O ports:

| OFFSET | NAME | R/W | DESCRIPTION |
|---|---|---|---|
| 0x00 | STATUS | R | Status bits for the disk device. |
| 0x04 | COMMAND | W | Port for issuing commands to the disk. |
| 0x08 | DATA | R | Return value port for query commands. The data will be available before the next clock cycle after the query command is written to the COMMAND port. |
| 0x0C | TSECTOR | RW | Number of the disk sector which should be read/written. |
| 0x10 | DMAADDR | RW | Start address of the memory buffer which will be used for sector reads and writes. The size of the buffer is the same as the size of the disk sector and addressing is 0x00000000-based unmapped. |

The following table describes the status bits of a disk device:

| BIT | NAME | DESCRIPTION |
|---|---|---|
| 0 | RBUSY | The disk is busy reading from disk to memory. |
| 1 | WBUSY | The disk is busy writing from memory to disk. |
| 2 | RIRQ | The disk has finished a read operation and generated an IRQ. The IRQ line is held raised by the disk while this bit is set. |
| 3 | WIRQ | The disk has finished a write operation and generated an IRQ. The IRQ line is held raised by the disk while this bit is set. |
| 27 | ISECT | The sector number given to a read/write request is invalid. |
| 28 | IADDR | The address given to a read/write request did not reside entirely in physical memory. |
| 29 | ICOMM | The last command issued to the COMMAND port was unrecognized. |

30 EBUSY The last command issued to the COMMAND port could not be handled because the disk was busy.

31 ERROR Undefined error in the device. The disk is to be considered unusable if this bit is set.

The commands that can be issued to a disk device through the COMMAND port are listed in the following table. Status changes caused by the command will be visible in the status register before the next clock cycle (like in normal memory writes on MIPS32 architecture).

0x01 Begin read operation. Will begin a transfer from the sector TSECTOR to the buffer addresses by DMAADDR. An IRQ is generated on completion.

0x02 Begin write operation. Will begin a transfer from the buffer addressed by DMAADDR to the sector TSECTOR. An IRQ is generated on completion.

0x03 Reset RIRQ. Will clear the RIRQ bit, indicating that the IRQ generated has been handled. This will cause the disk to not raise the IRQ line any further unless there is another IRQ pending (should never happen).

0x04 Reset WIRQ. Acts similarly to the RIRQ resetting.

0x05 Get number of sectors in the disk, returned in DATA.

0x06 Get sector size in bytes, returned in DATA.

0x07 Get sectors per cylinder, returned in DATA.

0x08 Get disk rotation period in milliseconds, returned in DATA.

0x09 Get disk full seek time in milliseconds, returned in DATA.

Using a disk in the OS is very simple. A thread wanting to write to a disk will first reserve the disk for itself. Then it will write the disk sector and the DMA transfer buffer address to TSECTOR and DMAADDR and issue a request for write operation to COMMAND. It should then check if there were any errors. If no errors occured, the thread will go to sleep.

When the operation is finished, the disk will raise an interrupt. The interrupt handler should then wake up the thread that has reserved the disk and reset the WIRQ bit. The thread will then release the disk reservation and go about its business.

Reading from the disk is done similarly.

See Section 3.3 [Configuring the Disk], page 4.

### 6.6.10 Network interface cards

Only network cards with device type code 0x401 are covered in this section.

A network interface card functions very much like the disk, except of course it will make IRQs on its own when packets arrive.

A NIC is "fully full duplex", meaning it has both a receive and a send buffer which can be used simultaneously ie. a frame can be received while sending is in progress. When a frame is received in the receive buffer it must be then DMA transferred to main memory before the next frame can be received.

A network interface card has the following I/O ports:

| OFFSET | NAME | R/W | DESCRIPTION |
|---|---|---|---|
| 0x00 | STATUS | R | Status bits for the network device. |
| 0x04 | COMMAND | W | Port for issuing commands to the NIC. |
| 0x08 | HWADDR | R | Link level address of the NIC. |
| 0x0C | MTU | R | Maximum transfer unit of the NIC in bytes. |
| 0x10 | DMAADDR | RW | Start address of the memory buffer which will be used for frame sends and receives. The size of the buffer is the size of the MTU and addressing is 0x00000000-based unmapped. |

The frames (or packets, since there is no trailer, but the term frame is used in this document) send to the network have the structure defined in the following table. Note that the addresses are in network byte order, which is big-endian (since `YAMS` is also big-endian, this is no problem).

| OFFSET | SIZE | NAME | DESCRIPTION |
|---|---|---|---|
| 0x00 | 1 word | DSTADDR | Link level address of the destination in network byte order. |
| 0x04 | 1 word | SRCADDR | Link level address of the sender in network byte order. |
| 0x08 | Variable | PAYLOAD | Link level payload, can be up to MTU - 8 bytes. The payload length is not defined here, it can be defined in the headers of the higher level protocol. |

Network device status bits are described in the following table

| BIT | NAME | DESCRIPTION |
|---|---|---|
| 0 | RXBUSY | The receive buffer is either receiving a frame or one has been received but not yet transferred to memory. If this bit is set new frames cannot be received. This bit must be cleared manually with the ready to receive command. |
| 1 | RBUSY | The NIC is transferring a frame from the receive buffer to memory. |
| 2 | SBUSY | The NIC is transferring a frame from memory to the send buffer. |

| | | |
|---|---|---|
| 3 | RXIRQ | The NIC has received a frame and generated an IRQ. The frame is ready to be transferred from the receive buffer. |
| 4 | RIRQ | A DMA transfer from the receive buffer to memory has completed and an IRQ was generated. |
| 5 | SIRQ | A DMA transfer from memory to the send buffer was completed and an IRQ was generated. |
| 6 | PROMISC | The NIC is in promiscuous mode, receiving all frames instead of just those addressed to it. |
| 27 | NOFRAME | There is no frame available in the receive buffer but a read transfer was requested. |
| 28 | IADDR | The DMA address given did not reside entirely in physical memory. |
| 29 | ICOMM | The last command issued to the COMMAND port was unrecognized. |
| 30 | EBUSY | The last command issued to the COMMAND port could not be handled because the NIC was busy. |
| 31 | ERROR | Undefined error in the device. The NIC is to be considered unusable if this bit is set. |

When a DMA transfer from memory to the send buffer is requested, the NIC will wait for the send buffer to be available (the previous transmit completed) before doing the actual transfer and then begin transmitting the transferred frame. That is why there is no IRQ after the frame has actually been transmitted into the network.

Available commands for a NIC are listed in the following table

| | |
|---|---|
| 0x01 | Start a DMA transfer from the receive buffer to the memory buffer addressed by DMAADDR. |
| 0x02 | Start a DMA transfer from the memory buffer addressed by DMAADDR into the send buffer. |
| 0x03 | Clear the RXIRQ bit, indicating that the interrupt has been handled and the NIC need not generate it any more for this frame. |
| 0x04 | Clear the RIRQ bit. |
| 0x05 | Clear the SIRQ bit. |
| 0x06 | Clear the RXBUSY bit. This tells the NIC that it can now receive a new frame into the receive buffer. |
| 0x07 | Enter promiscuous mode. |
| 0x08 | Exit promiscuous mode. |

A typical interrupt handler for a NIC works as follows. When a frame is received (RXIRQ) the handler will request a DMA transfer from the NIC into the memory buffer allocated for incoming frames. It will then clear the RXIRQ bit. When the DMA transfer is completed and the NIC generates an IRQ (RIRQ), the handler will do with the received frame whatever it needs to and then clear both RXBUSY and RIRQ bits.

When a frame needs to be sent, the sending thread will reserve the NIC and check if SBUSY bit is set. If set, the thread will go to sleep. When SBUSY is cleared (frame send complete), the interrupt handler will wake up the waiting thread. The thread will then request a send operation and check for errors. It can then exit, there is no need for the sending thread to wait for anything after this.

See Section 3.5 [Configuring the Network], page 5.

# 7 Copying

# GNU GENERAL PUBLIC LICENSE

Version 2, June 1991



## Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program," below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification.") Each licensee is addressed as "you."

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version," you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software

which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

## NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

## END OF TERMS AND CONDITIONS

## How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and an idea of what it does.
Copyright (C) 19yy  name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc.,
59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 20yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type 'show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type 'show c'
for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program 'Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

# Concept Index


## B

booting ........ 27
booting kernel image ........ 11
bootstrapping ........ 27

## C

Co-processor 0 ........ 15
co-processor registers ........ 10
command prompt ........ 8
config, file ........ 7
console commands ........ 8
CPU ........ 13
CPU registers ........ 10, 13

## D

device descriptors ........ 27, 28
device generated interrupts ........ 27, 29
device type codes ........ 28
DMA transfers ........ 27
dumping memory ........ 11
dumping memory to file ........ 9

## E

exiting simulator ........ 8

## G

general description of the simulated machine ... 13
getting help ........ 7

## H

hard disk programming ........ 32
hardware breakpoints ........ 9
help ........ 7

## I

I/O address space ........ 27
identifying system hardware ........ 28
interactive console ........ 8
interrupts ........ 10
interrupts in simulated machine ........ 15
IRQ distribution among processors ........ 29

## K

kernel argument string ........ 25
kernel loading ........ 27
kernel mode ........ 15

## L

loading binaries ........ 8

## M

memory architecture ........ 25
memory mapped devices ........ 27
memory mapping ........ 25
memory protection ........ 25
memory segments ........ 25

## N

network programming ........ 33
numbers ........ 12
numeric constants ........ 12

## O

online help ........ 8
options ........ 7

## P

powering off the simulator from the OS ........ 30
programming instructions ........ 13

## R

registers ........ 10
running one instruction at a time ........ 9

## S

simulated hardware ........ 13
simulation environment ........ 13
starting simulation ........ 9
supported devices ........ 28

## T

TLB ........ 9
TLB handling ........ 16
TTY programming ........ 30

## U

usage ........ 7
user mode ........ 15

## V

version ........ 7
virtual clockspeed ........ 29

## W

writing memory ........ 11
writing registers ........ 10

# Table of Contents


1 Overview .................................... 1

2 Install ....................................... 2

3 Configuration ............................... 3

3.1 Configuration Overview ................................ 3
3.2 Configuring the Simulator ................................ 3
3.3 Configuring the Disk ................................ 4
3.4 Configuring the Terminal ................................ 4
3.5 Configuring the Network ................................ 5
3.6 Config Example ................................ 5

4 Invoking YAMS ........................... 7

5 Command Console ........................ 8

5.1 help ................................ 8
5.2 quit ................................ 8
5.3 memwrite ................................ 8
5.4 memread ................................ 9
5.5 start ................................ 9
5.6 tlbdump ................................ 9
5.7 step ................................ 9
5.8 break ................................ 9
5.9 unbreak ................................ 10
5.10 regdump ................................ 10
5.11 regwrite ................................ 10
5.12 interrupt ................................ 10
5.13 dump ................................ 11
5.14 poke ................................ 11
5.15 boot ................................ 11
5.16 Entering numbers in the hardware console ................ 12

6 Simulated Machine ....................... 13

6.1 Machine Architecture ................................ 13
6.2 CPU ................................ 13
6.2.1 CPU registers ................................ 13
6.3 CP0 ................................ 15
6.3.1 Exceptions ................................ 15
6.3.2 TLB ................................ 16
6.3.3 Index ................................ 17
6.3.4 Random ................................ 17

6.3.5 EntLo0 and EntLo1 .... 18
6.3.6 Contxt .... 18
6.3.7 PgMask .... 19
6.3.8 Wired .... 19
6.3.9 BadVAd .... 19
6.3.10 Count .... 19
6.3.11 EntrHi .... 20
6.3.12 Compar .... 20
6.3.13 Status .... 20
6.3.14 Cause .... 21
6.3.15 EPC .... 22
6.3.16 PRId .... 23
6.3.17 Conf0 .... 23
6.3.18 Conf1 .... 24
6.3.19 LLAddr .... 25
6.3.20 ErrEPC .... 25
6.4 Memory .... 25
6.4.1 Architecture .... 25
6.4.2 Kernel Unmapped Uncached Segment .... 26
6.4.3 Accessing segments .... 26
6.4.4 Address Translation .... 26
6.5 Boot Sequence .... 27
6.6 Memory mapped I/O devices .... 27
6.6.1 Device descriptors .... 28
6.6.2 Device type codes .... 28
6.6.3 Hardware interrupts .... 29
6.6.4 System memory information device .... 29
6.6.5 System real-time clock device .... 29
6.6.6 Software shutdown device .... 30
6.6.7 CPU status devices .... 30
6.6.8 Terminal devices .... 30
6.6.9 Hard disk devices .... 32
6.6.10 Network interface cards .... 33

**7 Copying .... 37**

**GNU GENERAL PUBLIC LICENSE .... 38**

Preamble .... 38
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION .... 39
How to Apply These Terms to Your New Programs .... 43

**Concept Index .... 45**