1

TR-94034

RWC-1 Function-Level Simulator Manual
Release 0.2

Massively Parallel Software Laboratory

Received February 1995

Tsukuba Research Center, Real World Computing Partnership
Tsukuba Mitsui Building 16F, 1-6-1 Takezono
Tsukuba-shi, Ibaraki 305, JAPAN

Abstract This manual describes the RWC-1 user application program simulator, called **rwc1nommu**. In the first section, we give you an overview of this simulator. The command usages are described in the "Guided Tour" section. You can find convenient commands for interactive simulation and debugging. Run time environments for applications running on this system are described in the "Run Time Environment" section. The "Run Time Environment" section forms the programming guide of **rwc1nommu**. Finally, future work and problems are described in the "Conclusion" section.

Copyright ©1994 Real World Computing Partnership All Right Reserved.
DO NOT DISTRIBUTE.

*This document was prepared by Hiroshi Harada, Software Research Associates Inc., as the part of the contract with Massively Parallel Software Laboratory

Contents

1 Overview **1**

2 Guided Tour **1**
2.1 Launching . . . 1
2.2 Partition Size *psize* . . . 2
2.3 Current PE *curpe* . . . 2
2.4 Loading program *load,bload* . . . 3
2.5 Run Program *run* . . . 4
2.6 Set Break Point *break* . . . 4
2.7 Clear Break Point *clear* . . . 5
2.8 Rerun Suspended Program *continue* . . . 5
2.9 Step Execution *step* . . . 6
2.10 Trace Mode *trace* . . . 6
2.11 Disasm *disasm* . . . 7
2.12 Current Execution Mode *cem* . . . 10
2.13 Program Counter *where* . . . 10
2.14 Set Program Cpunter *pc* . . . 10
2.15 General Purpose Registers *gpr,gprd,gprud,gprf* . . . 11
2.16 Special Purpose Registers *spr* . . . 12
2.17 Memmory Dump *dump,adump* . . . 13
2.18 Global Symbol *symbol* . . . 14
2.19 Packet Logging *packet,plog* . . . 15
2.20 Segment *segment* . . . 16
2.21 Instruction Count *icount* . . . 17
2.22 Silent Mode *silent* . . . 18
2.23 Help *help* . . . 19
2.24 Error Code *ecode* . . . 21
2.25 Landing *quit* . . . 21

3 Run Time Environment **22**
3.1 Partition . . . 22
3.2 Virtual System Call . . . 22
3.2.1 Process Termination . . . 23
3.2.2 Ascii Character Write . . . 23
3.2.3 String Write . . . 23
3.2.4 Flush Write Buffer . . . 23
3.2.5 Ascii Character Read . . . 23
3.2.6 String Read . . . 24
3.2.7 Process Suspend . . . 24
3.2.8 Segment Allocation *1* . . . 24
3.2.9 Segment Resize *1* . . . 24
3.2.10 Segment Allocation *2* . . . 25
3.2.11 Segment Resize *2* . . . 25
3.2.12 Shared Segment Allocation . . . 25

3.2.13 Segment Deallocate 26
3.2.14 Get RPE 26
3.2.15 Get Partition Size 26
3.3 Virtual Memory 26
3.4 Environment variable Command Argument 27
3.5 Command Argument 27

4 Conclusion **28**

5 Appendix **29**
5.1 Command 29
5.2 Mode 30
5.3 Error Message 30

1 Overview

rwc1nommu is a multi PE RWC-1 simulator for user mode application programs. It runs on SUNOS and HPUX workstations in a RWCP TRC environment. It simulates user mode application programs, so it does not support privileged instructions and privileged registers including MMU and GTLB. But operating system facilities are needed to run application programs. **rwc1nommu** provides minimum operating system facilities to the user application programs. These include process management, simple I/O, and memory management. These operating system facilities are simulated by executing RWC-1's **trap** instruction. **rwc1nommu** also provides a range of convenient commands for interactive simulation and debugging. It supports many adb like commands such as **run**, **step**, **continue**, and so on. **rwc1nommu**'s instruction set is based on the **RWC-1 Instruction Set Manual Version 0.404**.

2 Guided Tour

This section describes the use of commands in this system. The **rwc1nommu** supports many convenient commands for interactive simulation and debugging. Each subsection has simple examples to guide you through. You can use this section as the operation manual for **rwc1nommu**.

2.1 Launching

When first started, **rwc1nommu** normally executes commands in the **.rwc1simrc** file in your $HOME environment variable directory. If the environment variable $RWC1SIMRC is set, **rwc1nommu** executes commands from $RWC1SIMRC. You can describe all of the **rwc1nommu** commands in this startup file. A line beginning with a # is treated as a comment. Here are some sample startup files for rwc1nommu.

```
#
# Startup file for rwc1nommu
#
plog off
silent on
eload
symbol
```

After the startup file has been processed, **rwc1nommu** can execute commands supplied as arguments on the command line, or commands from a script. One argument is handled as 1 **rwc1nommu** command. If the command is composed of two or more

tokens, you should enclose the command in double quarts when you invoke **rwc1nommu** from the c-shell.

```
% rwc1nommu "psize 16" "load a.out" run quit
% rwc1nommu "load a.out" "disasm $_main" run quit
```

After startup is completed, **rwc1nommu** begins reading commands from the standard input file, with prompt messages. The number enclosed in square brakets "[" and "]" is the current PE number [1].

```
rwc1nommu[0]
```

The simulator then repeatedly reads a command from the standard input file and executes it. The GNU readline library is used for reading commands from a standard input file, So you can use GNU emacs including editing facilities such as TAB(file name completion) and Ctrl-k(delete line). A history is also displayable by Ctrl-P(previous) and Ctrl-N(next).

2.2 Partition Size *psize*

Before loading a program, you can set the partition size, as the PE number. The *psize* command sets the partition size. The partition size must be a power of 2[2]. The default partition size is 32PE, and the maximum size is 64PE. You should choose a reasonable partition size to minimize simulation cost. You cannot change the partition size once you have loaded the program. When the *psize* command is invoked without an argument, the current partition size is printed to standard output.

```
rwc1nommu[0] psize 16
rwc1nommu[0] psize 4
```

[1] See curpe section

[2] RWC-1's partition size is a multiple of 8

2.3 Current PE *curpe*

The **rwc1nommu** supports a multi PE environment, and we have to handle many PEs. So we have introduces a **current PE**. By default, PE related commands, such as **where**,**pc**, are applied to the **current PE**. In other words, the **current PE** is the pointer to a special PE that the command applies. The **current PE** number is displayed at the end of the prompt message. When invoked without an argument, the *curpe* command displays the **current PE** number like this:

```
rwc1nommu[0] curpe
Current PE = 0
rwc1nommu[0]
```

When new current PE number is given, the *curpe* command changes the **current PE** like this:

```
rwc1nommu[0] curpe 3
Current PE = 3
rwc1nommu[3]
rwc1nommu[3] curpe 6
Current PE = 6
rwc1nommu[6]
```

2.4 Loading program *load,bload*

The *load* command loads the elf formatted program file. You can specify the program file path with a command argument. By default, the "a.out" file in the current directory will be loaded. If the specified file is not an elf formatted file, the *load* command will fail and output an error message[3].

```
rwc1nommu[0] load a.out
rwc1nommu[0] bload a.out
rwc1nommu[0] load hello -world -space
```

When the *load* command is invoked, **rwc1nommu** reads the elf header from the specified file, sets the program counter for each PE, reads the global symbols, allocates virtual memory segments, and then loads the program. As described in a later section, you can view these global symbols with the *symbol* command. Each global symbol name can be used as a command argument.

[3] See Error Message Appendix

```
rwc1nommu[0] symbol
00000000:00000000 Object DISP$_mypenum
00000000:00000000 Object SIZE$_others_break
00000000:00000000 Object DISP$_others_break
00000000:00000000 Object SIZE$_mypenum
00000000:00000000 Object DISP$_putchar
00000000:00000000 Object SIZE$_putchar
00000000:00000001 Object DISP$_myexit__Fv
00000000:00000001 Object DISP$_main
rwc1nommu[0] disasm $_main
_main:
[00000000:00002070]:st 0($r30), $r8
[00000000:00002070]:st 0($r30), $r8
[00000000:00002074]:nop
[00000000:00002078]:add $r28, $r31, 0
[00000000:00002080]:jmp.l 2168
[00000000:00002084]:nop
[00000000:00002088]:mkcnt $r0,$r30, 1044, 0, 1, 0, 0, 1,0
[00000000:00002090]:mkpkti $r0,$r28,492,0,1,0,0,1,1
[00000000:00002098]:break 3ffe
[00000000:0000209c]:nop
rwc1nommu[0]
```

2.5 Run Program *run*

The *run* command executes the program. You must load the program before invoking *run*. You can supply command arguments[4] like in a UNIX shell environment as follows:

```
rwc1nommu[1] run arg1 arg2
```

Once the **run** command is invoked, the simulator continues to run the program **break-point** is hit or the program is over. To suspend simulation, type **Ctrl-C**. The suspended program can be restarted with the *continue* command[5]. **Break point** and the *continue* command are described in a later section.

2.6 Set Break Point *break*

The *break* command is available to set break points and to confirm them. Specify the text address when you set a break point. The text address is specified in hexadecimal or as a

[4] See Command Argument Section
[5] See continue

symbol name with a $ prefix. Break point is a partition oriented resource. Enterd break points are shared by all PEs.

```
rwc1nommu[0] break 3a
rwc1nommu[0] break $_Function
```

When the *break* command is invoked without an argument, currently set break points and break point index will be displayed.

```
rwc1nommu[1] break 3c
rwc1nommu[1] break 60
rwc1nommu[1] break
Breakpoint[ 0]:00000000:0000003c
Breakpoint[ 1]:00000000:00000060
rwc1nommu[1]
```

2.7 Clear Break Point *clear*

The *clear* command clears the breakpoints set by the *break* command. You can specify the break point index. The break point indexes can be displayed by entering the *break* command without an argument.

```
rwc1nommu[3] clear 3
```

The *clear* command is invoked without an argument to clear all break points.

```
rwc1nommu[2] break 3c
rwc1nommu[2] break 60
rwc1nommu[2] break 6c
rwc1nommu[2] break
Breakpoint[ 0]:00000000:0000003c
Breakpoint[ 1]:00000000:00000060
Breakpoint[ 2]:00000000:0000006c
rwc1nommu[2] clear 1
rwc1nommu[2] break
Breakpoint[ 0]:00000000:0000003c
Breakpoint[ 2]:00000000:0000006c
rwc1nommu[2] clear
rwc1nommu[2] break
rwc1nommu[2]
```

2.8 Rerun Suspended Program *continue*

The *continue* commnad reruns a suspended program. There is no argument with this command.

```
rwc1nommu[0] continue
```

2.9 Step Execution *step*

The *step* command executes 1 instruction. There is no argument with this command.

```
rwc1nommu[0] step
```

2.10 Trace Mode *trace*

The *trace* command controls the **trace mode**. The trace mode is a PE oriented resource, allowing you to control each PE's **trace mode**. When **trace mode** is **on**, trace information will be printed to the standard output file during simulation. To turn on the current PE's **trace mode**, specify the on argument as below:

```
rwc1nommu[0] trace on
```

To turn on the all PE's trace modes, specify the **on** and **all** arguments like this:

```
rwc1nommu[0] trace on all
```

Once the trace mode is turned on, the simulator executes the program while printing trace information to the standard output file. The trace information consists of PE number, Program Counter, executed instruction, and nearest neighbour's symbol name.

To turn off the current PE's trace mode, specify the **off** argument like this:

```
rwc1nommu[0] trace off
```

To clear the all PE's trace modes, specify the **off** and **all** arguments like this:

```
rwc1nommu[0] trace off all
```

The *trace* command is invoked without argument, current trace mode of each PE will be displayed.

```
rwc1nommu[0] trace
PE::[ 0] Off PE::[ 1] Off PE::[ 2] Off PE::[ 3] Off
PE::[ 4] Off PE::[ 5] Off PE::[ 6] Off PE::[ 7] Off
PE::[ 8] Off PE::[ 9] Off PE::[10] Off PE::[11] Off
PE::[12] Off PE::[13] Off PE::[14] Off PE::[15] Off
rwc1nommu[0] curpe
Current PE = 0
rwc1nommu[0] trace on
rwc1nommu[0] trace
PE::[ 0] On PE::[ 1] Off PE::[ 2] Off PE::[ 3] Off
PE::[ 4] Off PE::[ 5] Off PE::[ 6] Off PE::[ 7] Off
PE::[ 8] Off PE::[ 9] Off PE::[10] Off PE::[11] Off
PE::[12] Off PE::[13] Off PE::[14] Off PE::[15] Off
rwc1nommu[0] step
PE[0]::[00000000:0000267c]:add $r27 , $r27 , 2048 :_getDS256
rwc1nommu[0]
rwc1nommu[0] trace on all
rwc1nommu[0] trace
PE::[ 0] On PE::[ 1] On PE::[ 2] On PE::[ 3] On
PE::[ 4] On PE::[ 5] On PE::[ 6] On PE::[ 7] On
PE::[ 8] On PE::[ 9] On PE::[10] On PE::[11] On
PE::[12] On PE::[13] On PE::[14] On PE::[15] On
rwc1nommu[0] step
PE[0]::[00000000:00002680]:sub $r26, $r26, 1 :_getDS256
PE[1]::[00000000:00002680]:sub $r26, $r26, 1 :_getDS256
PE[2]::[00000000:00002680]:sub $r26, $r26, 1 :_getDS256
PE[3]::[00000000:00002680]:sub $r26, $r26, 1 :_getDS256
PE[4]::[00000000:00002680]:sub $r26, $r26, 1 :_getDS256
PE[5]::[00000000:00002680]:sub $r26, $r26, 1 :_getDS256
PE[6]::[00000000:00002680]:sub $r26, $r26, 1 :_getDS256
PE[7]::[00000000:0000266c]:add $r28, $r31, $r27 :_getDS256
PE[8]::[00000000:0000266c]:add $r28, $r31, $r27 :_getDS256
PE[9]::[00000000:0000266c]:add $r28, $r31, $r27 :_getDS256
PE[10]::[00000000:0000266c]:add $r28, $r31, $r27 :_getDS256
PE[11]::[00000000:0000266c]:add $r28, $r31, $r27 :_getDS256
PE[12]::[00000000:0000266c]:add $r28, $r31, $r27 :_getDS256
PE[13]::[00000000:0000266c]:add $r28, $r31, $r27 :_getDS256
PE[14]::[00000000:0000266c]:add $r28, $r31, $r27 :_getDS256
PE[15]::[00000000:0000266c]:add $r28, $r31, $r27 :_getDS256
rwc1nommu[0]
```

2.11 Disasm *disasm*

The *disasm* command disasembles the program from the specified address. When *disasm* is invoked without an argument, 32 instructions from the current PE's program counter of current execution mode will be disassembled and printed to the standard output file. You can specify the text address as an argument. The text address should be specified in hexadecimal or as a symbol name with a $ prefix. You can also specify the instruction number in decimal. When the disasembled instruction's address matches a symbol value, the symbol value is also printed out.

```
rwc1nommu[0] disasm $_main
_main:
[00000000:00002538]:st 0($r30) , $r8
[00000000:0000253c]:nop
[00000000:00002540]:add $r28 , $r31 , 0
[00000000:00002548]:jmp.l 2638
[00000000:0000254c]:nop
[00000000:00002550]:mkcnt $r0 , $r30, 1197, 0, 1, 0, 0, 1, 0
[00000000:00002558]:mkpkti $r0 , $r28, 52c, 0, 1,0,0,1,3
[00000000:00002560]:break 3ffe
[00000000:00002564]:nop
[00000000:00002568]:add $r1 , $r31 , 2
[00000000:0000256c]:add $r24 , $r31 , 0
[00000000:00002570]:mkcnt $r0 , $r30, 1204, 0, 1, 0, 0, 1, 0
[00000000:00002578]:sll $r24 , $r24 , 54
[00000000:0000257c]:nop
[00000000:00002580]:add $r28 , $r31 , 8208
[00000000:00002588]:add $r24 , $r24 , $r28
[00000000:0000258c]:nop
[00000000:00002590]:mkpkti $r0 , $r24, 5f8, 0, 1,0,0,1,3
[00000000:00002598]:break 3ffe
[00000000:0000259c]:nop
[00000000:000025a0]:add $r1 , $r31 , 145273
[00000000:000025a8]:add $r2 , $r31 , $r0
[00000000:000025ac]:nop
[00000000:000025b0]:add $r28 , $r31 , 3
[00000000:000025b8]:jmp.l 2638
[00000000:000025bc]:nop
[00000000:000025c0]:mkcnt $r0 , $r30, 1211, 0, 1, 0, 0, 1, 0
[00000000:000025c8]:mkpkti $r0 , $r28, 538, 0, 1,0,0,1,7
[00000000:000025d0]:break 3ffe
[00000000:000025d4]:nop
[00000000:000025d8]:ld $r2, 0($r30)
[00000000:000025dc]:mkpkt $r0,$r2, 0(sop), 0(sou), 0(rol), 1(mask)
rwc1nommu[0]
rwc1nommu[0] disasm $_getDS 10
_getDS:
[00000000:00002638]:add.u $r28 , $r31 , 145296
[00000000:00002640]:ld $r28 , 0($r28)
[00000000:00002644]:bcnd 8 , $r28 , 269c(15)
[00000000:00002648]:add $r27 , $r31 , 1048576
[00000000:00002650]:add $r26 , $r31 , 64
[00000000:00002654]:trap 06
[00000000:00002658]:add $r27 , $r31 , 1048576
[00000000:00002660]:add.u $r28 , $r31 , 145296
[00000000:00002668]:st 0($r28) , $r27
[00000000:0000266c]:add $r28 , $r31 , $r27
rwc1nommu[0]
```

2.12 Current Execution Mode *cem*

The *cem* command displays the current execution mode of the specified PE. You can specify the PE number as an argument. When *cem* is invoked without an argument, the current PE's execution mode will be printed out.

```
rwc1nommu[0] cem
User Low Mode
rwc1nommu[0] cem 1 2 3
PE:[1]
User Low Mode
PE:[2]
User Low Mode
PE:[3]
User Low Mode
rwc1nommu[0]
```

2.13 Program Counter *where*

The *where* command displays the program counter. When the *where* command is invoked without an argument, the current PE's program counter of current execution mode will be displayed. You can specify the PE number, and execution mode. The PE number should be specified as a decimal. The execution mode shoud be specified as **userlow**, **userhigh** or **system**.

```
rwc1nommu[0] where
PC:[00000000:000025f4] STATUS:[1] INTERRUPT:[1]
rwc1nommu[0] where userlow userhigh
USER LOW:
PC:[00000000:000025f4] STATUS:[1] INTERRUPT:[1]
USER HIGH:
PC:[00000000:00000000] STATUS:[0] INTERRUPT:[1]
rwc1nommu[0] where 1 2 userlow
PE:[1]
PC:[00000000:000025f4] STATUS:[1] INTERRUPT:[1]
PE:[2]
PC:[00000000:000025f4] STATUS:[1] INTERRUPT:[1]
USER LOW:
PC:[00000000:000025f4] STATUS:[1] INTERRUPT:[1]
rwc1nommu[0]
```

2.14 Set Program Cpunter *pc*

The *pc* command changes the program counter. You must specify a new PC value. When no other arguments are given, **pc** changes the current PE's PC of current execution mode.

```
rwc1nommu[0] where
PC:[00000000:000025f0] STATUS:[1] INTERRUPT:[1]
rwc1nommu[0] pc 2ff0
rwc1nommu[0] where
PC:[00000000:00002ff0] STATUS:[1] INTERRUPT:[1]
rwc1nommu[0] cem
User Low Mode
rwc1nommu[0]
```

When an execution mode option is specified, **pc** command changes the current PE's PC of specified execution mode. The execution mode option must be 'userhigh", "userlow", "system" or "all".

```
rwc1nommu[0] pc userhigh 20f0
rwc1nommu[0] where userhigh
USER HIGH:
PC:[00000000:000020f0] STATUS:[0] INTERRUPT:[1]
rwc1nommu[0]
rwc1nommu[0] pc all 3030
rwc1nommu[0] where userlow userhigh system
USER LOW:
PC:[00000000:00003030] STATUS:[1] INTERRUPT:[1]
USER HIGH:
PC:[00000000:00003030] STATUS:[0] INTERRUPT:[1]
SYSTEM:
PC:[00000000:00003030] STATUS:[0] INTERRUPT:[0]
rwc1nommu[0]
```

When specifying the PE execution mode option, you can use one more option, PE id. The **pc** command, invoked with these 2 options, changes the specified PE's PC of specified execution mode.

```
rwc1nommu[0] pc userlow 12 3030
rwc1nommu[0] where 12 userlow
PE:[12]
PC:[00000000:00003030] STATUS:[1] INTERRUPT:[1]
USER LOW:
PC:[00000000:00003030] STATUS:[1] INTERRUPT:[1]
rwc1nommu[0]
```

2.15 General Purpose Registers *gpr,gprd,gprud,gprf*

The *gpr*, *gprd*, *gprud*, *gprf* command displays the value of the specified general purpose registers. The difference between these commands is output number format. Specify the register number as decimal. You can also specify the current execution mode. When these commands are invoked without an argument, all the current PE's general purpose registers of current execution mode will be displayed. The output number formats are described in the table below.

Command	Number format
gpr	Unsigned hexadecimal
gprd	Signed decimal
gprud	Unsigned decimal
gprf	Floationg Point

```
rwc1nommu[0] gpr
R[ 0]:00000000:00000001 R[ 1]:00000000:00000000
R[ 2]:00000000:00000000 R[ 3]:00000000:00000000
R[ 4]:00000000:00000000 R[ 5]:00000000:00000000
R[ 6]:00000000:00000000 R[ 7]:00000000:00000000
R[ 8]:00000000:00000000 R[ 9]:00000000:00000000
R[10]:00000000:00000000 R[11]:00000000:00000000
R[12]:00000000:00000000 R[13]:00000000:00000000
R[14]:00000000:00000000 R[15]:00000000:00000000
R[16]:00000000:00000000 R[17]:00000000:00000000
R[18]:00000000:00000000 R[19]:00000000:00000000
R[20]:00000000:00000000 R[21]:00000000:00000000
R[22]:00000000:00000000 R[23]:00000000:00000000
R[24]:00000000:00000000 R[25]:00000000:00000000
R[26]:00000000:00000000 R[27]:00000000:00000000
R[28]:00000000:00000000 R[29]:00000000:00000000
R[30]:00000000:00000000 R[31]:00000000:00000000
rwc1nommu[0]
rwc1nommu[0] gpr 1 2 3
R[ 1]:00000000:00000000
R[ 2]:00000000:00000000
R[ 3]:00000000:00000000
rwc1nommu[0] gprd 1
R[ 1]:0
rwc1nommu[0] gprf 1
R[ 1]:0.000000
rwc1nommu[0]
rwc1nommu[0] gpr userhigh 1 2
USER HIGH:
R[ 1]:00000000:00000000
R[ 2]:00000000:00000000
rwc1nommu[0]
```

2.16 Special Purpose Registers *spr*

The *spr* command displays special purpose registers. Specify the PE number as a decimal. You can also specify the execution mode. When the *spr* command is invoked without an argument, the current PE's special purpose registers of current execution mode will be displayed.

```
rwc1nommu[0] spr
[PC] :00000000:00002678 [STATUS:1] [INTERRUPT:1]
[EIAR] :00000000:00000000 [STATUS:0] [INTERRUPT:0]
[ELCSR]:[00000000:00000000]
[ELR] :[OV:0][ZD:0][OF:0][UF:0][IF:0][AL:0][PT:0][RP:0] ↙
[PP:0][MEB:0][MEI:0][EC:0][GTM:0][UE:0]
[CSR] :[PEM:0][TB:0][C:0][PM:0]
[MTAGR]:[SOC:0][ROL:0][PS:0]
rwc1nommu[0] spr userhigh
USER HIGH:
[PC] :00000000:00000000 [STATUS:0] [INTERRUPT:1]
[EIAR] :00000000:00000000 [STATUS:0] [INTERRUPT:0]
[ELCSR]:[00000000:00000000]
[ELR] :[OV:0][ZD:0][OF:0][UF:0][IF:0][AL:0][PT:0][RP:0] ↙
[PP:0][MEB:0][MEI:0][EC:0][GTM:0][UE:0]
[CSR] :[PEM:0][TB:0][C:0][PM:0]
[MTAGR]:[SOC:0][ROL:0][PS:0]
rwc1nommu[0]
rwc1nommu[0] spr 2 userhigh
PE:[2]
[PC] :00000000:00002678 [STATUS:1] [INTERRUPT:1]
[EIAR] :00000000:00000000 [STATUS:0] [INTERRUPT:0]
[ELCSR]:[00000000:00000000]
[ELR] :[OV:0][ZD:0][OF:0][UF:0][IF:0][AL:0][PT:0][RP:0] ↙
[PP:0][MEB:0][MEI:0][EC:0][GTM:0][UE:0]
[CSR] :[PEM:0][TB:0][C:0][PM:0]
[MTAGR]:[SOC:0][ROL:0][PS:0]
USER HIGH:
[PC] :00000000:00000000 [STATUS:0] [INTERRUPT:1]
[EIAR] :00000000:00000000 [STATUS:0] [INTERRUPT:0]
[ELCSR]:[00000000:00000000]
[ELR] :[OV:0][ZD:0][OF:0][UF:0][IF:0][AL:0][PT:0][RP:0] ↙
[PP:0][MEB:0][MEI:0][EC:0][GTM:0][UE:0]
[CSR] :[PEM:0][TB:0][C:0][PM:0]
[MTAGR]:[SOC:0][ROL:0][PS:0]
rwc1nommu[0]
```

2.17 Memmory Dump *dump, adump*

The *dump* and *adump* commands dump the contents of memory for the current PE. The *dump* command dumps memory in hexadecimal, and the *adump* command dumps in

hexadecimal and ascii characters. You can specify the memory address in hexadecimal or as a symbol name with a $prefix.

```
rwc1nommu[0] dump $_alphabet
[00000000:000001e0]:          61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70
[00000000:000001f0]:          71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f 80
.....
rwc1nommu[0]
rwc1nommu[0] adump $_alphabet
[00000000:000001e0]:          61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70
                              a b c d e f g h i j k l m n o p
[00000000:000001f0]:          71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f 80
                              q r s t u v w x y z  —
.....
```

2.18 Global Symbol *symbol*

The *symbol* command displays the global symbol information. The global symbol information consists of value(address), attribute, and name.

```
rwc1nommu[0] load elf_formatted_aout
rwc1nommu[0] symbol
00000000:000000f0 Text _main
00000000:00020c20 Object _extern_str
00000000:000001f8 Text _freeDS
00000000:00000220 Text _printf
00000000:00000170 Text _getDS64
00000000:00000140 Text start
00000000:00000170 Text _getDS128
00000000:00000680 Text _putchar
00000000:00000690 Text _printfloat
00000000:00000650 Text .udiv
00000000:00000660 Text .urem
00000000:00000670 Text .rem
00000000:00000658 Text .div
00000000:00000648 Text .mul
00000000:00000640 Text .umul
00000000:00000890 Text _ftoi
00000000:000009f0 Text _itof
00000000:00000b70 Text _convdouble
00000000:00000ba0 Object _etext
00000000:00020c20 Object _edata
00000000:00020c20 Object __bss_start
00000000:00020c28 Object _end
00000000:00020c28 Object end
rwc1nommu[0]
```

You can specify a symbol name with a $ prefix like this:

```
rwc1nommu[0] symbol $_main
00000000:000000f0 Text _main
rwc1nommu[0]
```

2.19 Packet Logging *packet,plog*

You can log the RWC-1 packet made by the **make packet** instruction if you need. To start logging packets, invoke the *plog* command with the **on** option. To reset, use the **off** option. When the *plog* command is invoked without an argument, the current packet logging mode will be displayed. If you want to save simulation time and virtual memory resources, it is suggested that you shoud not log packets.

```
rwc1nommu[0] plog
Packet Logging Off
rwc1nommu[0] plog on
rwc1nommu[0] plog
Packet Logging On
rwc1nommu[0] plog off
rwc1nommu[0] plog
Packet Logging Off
rwc1nommu[0]
```

The *packet* command displays the logged packet information. You can specify the packet serial number as the argument. Packet information consists of the serial number, PE number, text address invoked by the **make packet** instruction, and packet fields. You can specify the packet serial number. When invoked without an argument, all logged packets will be displayed.

```
rwc1nommu[0] packet 3

-Packet[3] generated at PE::[0][00000000:00002590]-
trc:[0] k_u:[0] priority:[0] apid:[1]
ppe:[0] lvia:[5f8 -¿ 00000000:00002fc0] wcf:[ 1] lvoa:[402] argc[2]
arg[0]:000004b4:00100002
arg[1]:00000000:00000002
rwc1nommu[0]
rwc1nommu[0] packet

-Packet Count[2]-

-Packet[0] generated at PE::[0][00000000:00002610]-
trc:[0] k_u:[0] priority:[0] apid:[1]
ppe:[0] lvia:[4a7 -¿ 00000000:00002538] wcf:[ 1] lvoa:[20000] argc[2]
arg[0]:000004c4:00000002
arg[1]:00000000:00000000

-Packet[1] generated at PE::[0][00000000:00002558]-
trc:[0] k_u:[0] priority:[0] apid:[1]
ppe:[0] lvia:[52c -¿ 00000000:00002960] wcf:[ 1] lvoa:[20100] argc[2]
arg[0]:000004ad:00100002
arg[1]:00000000:00000000
```

2.20 Segment *segment*

rwc1nommu provides a simple virtual memory management facility to run application program. The basic unit of virtual memory is called a **segment**, and each PE's virtual memory consists of 1 or more **segments** [6]. The *segment* command displays the **segment** information for the current PE.

```
rwc1nommu[0] segment
Segment:[ 0]
Start_Addr[00000000:00022000]
Size[8192:0x00002000]
Page[1]
Protection[1]
Segment:[ 1]
Start_Addr[00000000:00002000]
Size[8192:0x00002000]
Page[1]
Protection[0]
Segment:[ 2]
Start_Addr[00000000:40000000]
Size[16384:0x00004000]
Page[2]
Protection[1]
Segment:[ 3]
Start_Addr[00000000:00100000]
Size[524288:0x00080000]
Page[64]
Protection[1]
rwc1nommu[0]
```

2.21 Instruction Count *icount*

The **rwc1nommu** records the executed inctruction count for each PE. The *icount* command displays the instruction count for the specified PE. When no argumen is specified, the instruction count for the current PE will be displayed.

[6] See "Run Time Environment" section

```
rwc1nommu[0] icount
Instruction Count statistics
                                    Load_Word:[ 1]   hspace2cm Store_Word:[ 126
                                     Add_S_RR:[ 2]   hspace2cm Add_S_RI:[ 256]
                                     Add_U_RI:[ 2]   hspace2cm Sub_S_RI:[ 125]
                                Branch_Conditional:[ 126]   hspace2cm Jump:[ 1]
                                           Trap:[ 2]
PE[0]::Total [641 / 641 Clock]
```

When the **all** argument is specified, the instruction count will be displayed for all PE's.

```
rwc1nommu[0] icount all
Instruction Count statistics
                                    Load_Word:[ 1]   hspace2cm Store_Word:[ 126
                                     Add_S_RR:[ 2]   hspace2cmAdd_S_RI:[ 256]
                                     Add_U_RI:[ 2]   hspace2cmSub_S_RI:[ 125]
                                Branch_Conditional:[126]   hspace2cmJump:[ 1]
                                           Trap:[ 2]
PE[0]::Total [641 / 641 Clock]
Instruction Count statistics
                                    Load_Word:[ 1]   hspace2cmStore_Word:[ 126]
                                     Add_S_RR:[ 2]   hspace2cmAdd_S_RI:[ 256]
                                     Add_U_RI:[ 2]   hspace2cmSub_S_RI:[ 125]
                                Branch_Conditional:[ 126]   hspace2cmJump:[ 1]
                                           Trap:[ 2]
PE[1]::Total [641 / 641 Clock]
Instruction Count statistics
                                    Load_Word:[ 1]   hspace2cmStore_Word:[126]
                                     Add_S_RR:[ 2]   hspace2cmAdd_S_RI:[256]
                                     Add_U_RI:[ 2]   hspace2cmSub_S_RI:[125]
                                Branch_Conditional:[126]   hspace2cmJump:[ 1]
                                           Trap:[ 2]
PE[2]::Total [641 / 641 Clock]
Instruction Count statistics
                                    Load_Word:[ 1]   hspace2cmStore_Word:[ 126]
                                     Add_S_RR:[ 2]   hspace2cmAdd_S_RI:[ 256]
                                     Add_U_RI:[ 2]   hspace2cmSub_S_RI:[ 125]
                                Branch_Conditional:[ 126]   hspace2cmJump:[ 1]
                                           Trap:[ 2]
PE[3]::Total [641 / 641 Clock]
```

Specify the **clear** argument to clear the instruction count.

```
rwc1nommu[0] icount clear
Clear Instruction Counter
rwc1nommu[0] icount
Instruction Count statistics
Total [0]
```

2.22 Silent Mode *silent*

To help with debugging, **rwc1nommu** usually prints many messages such as loading program information and receiving packet information. In **rwc1nommu**, the **silent** mode turns off these messages. When **silent** mode is on, these messages are not printed. To turn on **silent** mode, invoke the *silent* command with the **on** argument. When **silent** mode is off, these messages will be printed during simulation. To turn off **silent** mode, invoke the *silent* command with the **off** argument. The *silent* command without an argument prints the current silent mode. The *silent* command applies to all PE.

```
rwc1nommu[0] silent off
Clear Silent mode
rwc1nommu[0] silent
Silent Off
rwc1nommu[0] silent on
```

2.23 Help *help*

The *help* command displays the following help screen.

```
rwc1nommu[0]  help
quit          :quit – quit simulation
help          :help – this command
load          :load [file_name] – load binary file
eload         :eload [file_name] – load elf file
break         :break [text address] – set break point
clear         :clear [break point index] – clear break points
step          :step – step execution
run           :run [command arg...] – start execution
continue      :continue – continue execution
where         :where [peid [userlow — userhigh — system] ... ] ↙
              – show current program counter
disasm        :disasm [text address [line count]] – disasm text
gpr           :gpr [userhigh — userlow — system] [gpr index...] ↙
              – gpr hex dump
gprd          :gprd [userhigh — userlow — system] [gpr index...] ↙
              – gpr dec dump
gprud         :gprud [userhigh — userlow — system] [gpr index...] ↙
              – gpr unsigned dec dump
gprf          :gprf [userhigh — userlow — system] [gpr index...] ↙
              – gpr floating point dump
spr           :spr [ peid [userhigh — userlow — system] ...]– spr dump
prr           :prr – prr dump
dump          :dump [memory address] – memory hex dump
adump         :adump [memory address] – memory ascii dump
trace         :trace [on,off] – set trace or reset trace
silent        :silent [on,off] – set silent or resetr silent
icount        :icount [clear] – display executed instructions
packet        :packet [packet serial no] – dump packet
plog          :plog [on,off] – set packet logging or reset packet logging
ecode         :ecode – display error code
segment       :segment – display virtual memory segment informatrion
symbol        :symbol [symbol name] – display external symbols
cem           :cem [peid] – display current exec mode
dmacr         :dmacr – display DMA Control Registers
gtlb          :gtlb – display Global TLB
ltlb          :ltlb – display Local TLB
mmap          :mmap – display memory map
isc           :isc – display Instruction Step Counter
curpe         :curpe pe_id – change current PE
psize         :psize [1,2,4,8,16,32] – set partition size
pc            :psize pc [pe_id [exec_mode]] – set Program Counter
rwc1nommu[0]
```

2.24 Error Code *ecode*

If **rwc1nommu** finds an error such as a memory alignment error, it prints an error message to the standard output file. The *ecode* command prints the error code to the standard output file.

```
rwc1nommu[0] ecode
E_ADDRESS                -1
E_ALIGN                  -2
E_PROTECTION             -3
E_INSTRUCTION            -4
E_OVERFLOW               -5
E_BREAKPOINT             -6
E_CONDITION_CODE         -7
E_BRANCH_CONDITION       -8
E_SUPERVISOR_INSTRUCTION -10
E_OPCODE                 -11
E_OPCODE2                -12
E_TRAP_NO                -13
E_REGISTER_PROTECTION    -14
E_INSTRUCTION_FORMAT     -15
E_LOCK_FAULT_TRAP        -16
E_STATUS                 -17
E_SEGMENT_ALLOC          -18
E_ELF_PROGRAM_TYPE       -19
E_ELF_PERMISSION         -20
E_ELF_PROGRAM_SIZE       -21
E_ELF_PROGRAM_ADDR       -22
E_ELF_PROGRAM_ALIGN      -23
E_REGISTER               -24
E_SYNC_WCF               -25
E_PAGE_ALIGN             -26
rwc1nommu[0]
```

2.25 Landing *quit*

The *quit* command terminates the simulation. The *quit* command also can be executed from the command line. You can use this simulator as a RWC-1 interpreter like this:

```
your_shell% rwc1nommu "psize 4" "silent on" "plog off" "eload hello" run quit
PE[0]: hello world
PE[1]: hello world
PE[2]: hello world
PE[3]: hello world
your_shell%
```

3 Run Time Environment

In this section, we describe the run time environment for application programs. The run time environment covers **virtual system calls**, **virtual memory management**, **environment variables** and **command argument**. If you are not going to write RWC-1 programs to run on this simulator, you can skip this section.

3.1 Partition

rwc1nommu supports multi PE simulation. You can choose and refer partition size with the *psize* command[7]. The minimum partition size is 1PE, and the maximum is 64PE. The partition size must be a power of 2. In the **rwc1nommu** run time environment, relative PE numbers(RPE) begin at 0, and end at (partition size - 1). Each PE can get the PE's RPE number by executing **trap** [8].

3.2 Virtual System Call

rwc1nommu does not support privileged instructions and registers, because it is a user application simulator. But minimum operating system facilities are needed to run the user application. Then **rwc1nommu** provides these facilities. These are implemented as RWC-1's **trap** instruction. They are called **Virtual systemcalls** in this manual. The following are the virtual system calls provided by the **trap** instruction.

- process termination
- write 1 ascii character to standard output file
- write null terminated string to standard output file
- flush write buffer
- read 1 character from standard input file
- read string from standard input file
- suspend process
- virtual memory allocation(size is specified by the number of page)
- virtual memory resizeing(size is specified by the number of page)
- virtual memory allocation(size is specified by the number of byte)
- virtual memory resizeing(size is specified by the number of byte)
- virtual memory deallocation
- get RPE number

[7] See psize
[8] See virtual system call

- get partition size

The **trap number** distinguishes each **virtual system call** [9]. **Tap numbers** used by a **virtual system call** are defined in the *rwc-1/include/arch/opcode.h* file. Some **virtual system calls** needs arguments, and these arguments shuld be set in the appropriate general purpose register. Usually a **virtual system call** receives 26th and 27th gpr as arguments, and returns it's return value in the 27th gpr. Indivisual virtual systemcalls are described in the following.

3.2.1 Process Termination

This call terminates execution.

Trap Number	SIM_EXIT 0x01
1st argument	None
2nd argument	None
Return value	Error Code

3.2.2 Ascii Character Write

This call prints one ascii character to the standard output file.

Trap Number	SIM_PRINTC 0x02
1st argument	Ascii Code
2nd argument	None
Return Value	Error Code

3.2.3 String Write

This call prints a null terminated string to the standard output file.

Trap Number	SIM_PRINTS 0x03
1st argument	String's Address
2nd argument	None
Return Value	Error Code

3.2.4 Flush Write Buffer

This call flushs and prints the write buffer to the standard output file. If the write buffer is empty, nothing is done.

[9] For datails of the trap instruction, see RWC-1 Instruction Set Manual

Trap Number	SIM_FLUSH 0x20
1st argument	None
2nd argument	None
Return Value	Error Code

3.2.5 Ascii Character Read

This call reads a one byte ascii character from the standard input file, and stores it in the virtual memory space. The address is specified as the 1st argument. You should specify a valid address.

Trap Numbner	SIM_GETC 0x04
1st arguments	Character's Address
2nd argument	None
Return Value	Error Code

3.2.6 String Read

This call reads a string from the standard input file, and stores it in the virtual memory space. The address is specified as the 1st argument. You should specify a valid address.

Trap Number	SIM_GETLINE 0x05
1st argument	String's argument
2nd argument	None
Return Value	Error Code

3.2.7 Process Suspend

This call suspends simulation and returns the simulator to the prompt. This call is used to implement the break point suspend facility for this simulator. It is not for general use. This call has no argument and no return value; it keeps all register values for current simulation.

Trap Number	SIM_BREAKPOINT_HIT 0x08
1st argument	None
2nd argument	None
Return Value	None

3.2.8 Segment Allocation *1*

This call allocates a segment in virtual memory space. The Allocated segment is PE local. The 1st argument specifies the segment's start address, and the 2nd argument specifies the size of the segment in pages. Page size of a segment is 8K bytes. If for example you are going to allocate a 1 mega byte segments at 0x40000000(1 giga), The 1st argument should be 0x40000000, and the 2nd argument should be 128(1M/8K). You should specify

a valid start address where a segment is not allocated. If another segment is already allocated, this **virtual system call** returns **E_ADDRESS**.

Trap Number	SIM_SEG_ALLOC 0x06
1st argument	start address
2nd argument	number of page
Return Value	Error Code

3.2.9 Segment Resize *1*

This **virtual system call** resizes the segment allocated at the specified address. The 1st argument specifies the address, and the 2nd specifies the size. The size is specified in pages. If no segment is allocated at the specified address, this **virtual system call** will return **E_ADDRESS**.

Trap Number	SIM_SEG_RESIZE 0x0A
1su argument	Segment Address
2nd argument	size(number of pages)
Return	Error Code

3.2.10 Segment Allocation *2*

This call allocates a segment in virtual memory space. The Allocated segment is PE local. The 1st argument specifies the segment's start address, and the 2nd argument specifies the byte size to be allocated for the segment. If for example, you are going to allocate a 1 mega byte segment at 1¡¡30 (1 giga), the 1st argument should be 1¡¡30, and the 2nd argument should be 1¡¡20(1M). You should specify a valid start address where no segment is allocated. If another segment is already allocated, this **system call** returns **E_ADDRESS**. If the size is not multiple of pagesize, segment size will be extended by multiples of pagesize.

Trap Number	SIM_SEG_ALLOC2 0x09
1st argument	Segment Address
2nd argument	Byte Size
Return Value	Error Code

3.2.11 Segment Resize *2*

This **virtual system call** resizes the segment allocated at the specified address. The 1st argument specifies the address, and the 2nd specifies the size. The size is specified in bytes. If no segment is allocated at the specified address, this **virtual system call** will return **E_ADDRESS**. If the size is not a multiple of pagesize, the segment size will be extended by multiples of pagesize.

Trap Number	SIM_SEG_RESIZE2 0x0B
1st argument	Segment Address
2nd argument	Byte Size
Return Value	Error Code

3.2.12 Shared Segment Allocation

This call allocates segments in virtual memory space. The Allocated segment is shared by all PEs. The 1st argument specifies the segment's start address, and the 2nd argument specifies the byte size to be allocated to the segment. If for example, you are going to allocate a 1 mega byte segment at 1¡¡30 (1 giga), the 1st argument should be 1¡¡30, and the 2nd argument should be 1¡¡20(1M). You should specify a valid start address where no segment is allocated. If another segment is already allocated, this **system call** returns **E_ADDRESS**. If the size is not a multiple of pagesize, the segment size will be extended by multiples of pagesize.

Trap Number	SIM_SHARED_SEG_ALLOC 0x0F
1st argument	Segment Address
2nd argument	Byte Size
Return Value	Error Code

3.2.13 Segment Deallocate

This **virtual system call** deallocates the segment allocated at a specified address. The 1st argument specifies the address. If no segment is allocated at the specified address, it returns **E_ADDRESS**.

Trap Number	SIM_SEG_REMOVE 0x07
1st argument	Segment Address
2bd Argument	None
Return Value	Error Code

3.2.14 Get RPE

This **virtual system call** returns the RPE number.

Trap Number	SIM_GET_RPE 0x0C
1st argument	None
2bd Argument	None
Return Value	RPE Number

3.2.15 Get Partition Size

This **virtual system call** returns the partition size.

Trap Number	SIM_GET_PSIZE 0x21
1st argument	None
2bd Argument	None
Return Value	Partition Size

3.3 Virtual Memory

rwc1nommu provides a minimum virtual memory management facility to application programs. The basic unit of virtual memory is called a **segment**, and the virtual memory consists of 1 or more **segments**. The segment has 3 attributes:

- Start Address
- Size
- Protection

The segment consists of 1 or more pages, with a page size of 8K bytes. The size of a segment must be a multiple of the page size. The application program is able to manage segments by executing a **virtual system call**. The maximum number of segments per PE is 16. When an elf formatted program is loaded, the **load** command is invoked, and the simulator allocates segments accoding to the elf header in the loaded program. The text segment is shared by all PEs. When a simple binary file is loaded(**bload** is invoked), the simulator allocates 3 **segments**, Text, Data, Stack. Following are the attributes of these segments.

Segment	Address	Size	Protection
Text	0x00	Depends on the file size	READ_ONLY
Data	0x40000000(1G)	4Mega Byte	READ_WRITE
Stack	0x7FC00000	4Mega Byte	READ_WRITE

bf segment command prints the allocated segment information.

3.4 Environment variable Command Argument

Like in a UNIX environment, environment variables are passed and copied to the 1 giga address of application's virtual memory. Environment variables are shared by all PEs. The "**adump 40000000**" command prints passed environment variables like this:

```
rwc1nommu[0]          eload ../MAKE.rwc1nommu.sparc-sun-sunos4.1.3/a.out
rwc1nommu[0]          run
Ctrl-C
rwc1nommu[0]          adump 40000000
[00000000:40000000]:  44 49 53 50 4c 41 59 3d 75 32 31 3a 30 2e 30 00
                      D I S P L A Y = u 2 1 : 0 . 0
[00000000:40000010]:  4a 53 45 52 56 45 52 3d 6a 73 65 72 76 65 72 00
                      J S E R V E R = j s e r v e r
[00000000:40000020]:  4d 41 49 4c 3d 2f 75 73 72 2f 73 70 6f 6f 6c 2f
                      M A I L = / u s r / s p o o l /
[00000000:40000030]:  6d 61 69 6c 2f 68 2d 68 61 72 61 64 61 00 4c 4f
                      m a i l / h - h a r a d a L O
[00000000:40000040]:  47 4e 41 4d 45 3d 68 2d 68 61 72 61 64 61 00 43
                      G N A M E = h - h a r a d a C
[00000000:40000050]:  4f 46 46 45 45 48 4f 53 54 3d 75 33 38 00 57 57
                      O F F E E H O S T = u 3 8 W W
[00000000:40000060]:  57 5f 48 4f 4d 45 3d 2f 68 6f 6d 65 2f 68 2d 68
                      W _ H O M E = / h o m e / h - h
[00000000:40000070]:  61 72 61 64 61 2f 57 57 57 2f 4d 6f 73 61 69 63
                      a r a d a / W W W / M o s a i c
rwc1nommu[0]
```

3.5 Command Argument

Like in a UNIX environment, command arguments are also passed to the application program. The command arguments are always located from 0x40002000(1giga + 8kilo) of virtual memory address, and passed arguments are shared by all PEs. The "**adump 40002000**" command prints passed command variables like this:

```
rwc1nommu[0]          run in rwc1nommu environment, arguments is passed like unix.
Ctrl-C
rwc1nommu[0]          adump 40002000
[00000000:40002000]:  2e 2e 2f 4d 41 4b 45 2e 72 77 63 31 6e 6f 6d 6d
                      . . / M A K E . r w c 1 n o m m
[00000000:40002010]:  75 2e 73 70 61 72 63 2d 73 75 6e 2d 73 75 6e 6f
                      u . s p a r c - s u n - s u n o
[00000000:40002020]:  73 34 2e 31 2e 33 2f 61 2e 6f 75 74 00 69 6e 00
                      s 4 . 1 . 3 / a . o u t i n
[00000000:40002030]:  72 77 63 31 6e 6f 6d 6d 75 00 65 6e 76 69 72 6f
                      r w c 1 n o m m u e n v i r o
[00000000:40002040]:  6e 6d 65 6e 74 2c 00 61 72 67 75 6d 65 6e 74 73
                      n m e n t , a r g u m e n t s
[00000000:40002050]:  00 69 73 00 70 61 73 73 65 64 00 6c 69 6b 65 00
                      i s p a s s e d l i k e
[00000000:40002060]:  75 6e 69 78 2e 00 00 01 01 5f 80 00 00 0f ac 00
                      u n i x . _
[00000000:40002070]:  21 5a c0 36 1d ff be 00 03 cf e0 10 00 00 00 00
                      ! Z 6
rwc1nommu[0]
```

4 Conclusion

rwc1nommu does not support previleged registers and instructions. These facilities will be suppoted by **rwc1mmu**, a kernel mode RWC-1 simulator, which will be released in the near future. A X-Window based user interface is going to be supported until the next release. The new user interface will probably be almost same as the maintenance monitor's interface. The GDB based debugger, supporting symbolic debugging, is being implemented by MRI. Questions, bug reports and suggestions are welcome.

5 Appendix

5.1 Command

Command	Arguments	Description
load	[File path]	Load elf formatted file
bload	[File path]	Load simple binary file
run	[command argument...]	run program
break	[text address]	set or print breakpoint
clear	[breakpoint index]	clear breakpoint
continue		rerun program
step		execute 1 instruction
trace	[on,off]	Control Trace Mode
disasm	[text address [number of instructions]]	disassemble text
where	[PE id [exec mode]]	prints Program Counter
pc	[exec mode [PE id] new program counter]	update program counter
gpr	[[exec mode] gpr index ...	print gpr in hexadecimal
gprd	[[exec mode] gpr index...]	print gpr in signed decimal
gprud	[[exec mode] gpr index...]	print gpr in unsigned decimal
gprf	[[exec mode] gpr index...]	print gpr in floating point
spr	[PE id [exec mode]]	prints spr
dump	[Memory Address]	Dump memory
adump	[Memory]	Ascii dump memory
symbol	[$symbol]	print symbol information
packet	[packet serial...]	print packet information
plog	[on,off]	Control packet logging mode
segment	[PE id...]	print segment information
icount	[clear]	print executed instruction count
silent	[on,off]	Control siulent mode
help		Command help
ecode		print error code
quit		quit system

5.2 Mode

Mode	Description	Command	Default Value
Trace	Control Trace simulation	**trace**	off
Packet Log	Control Packet Logging	**plog**	off
Silent	Control messages	**silent**	off

5.3 Error Message

Message	Description
Address Error	Invalid address is specified
Alignment Error	Alignment error is occoured be Load/Store instructions
Protection Error	written in the READ_ONLY protection segment
Branch Condition Code Error	Invalid Conditional Code of **bbc**is specified
Supervisor Instruction Error	privciledged instruction is invoked
Opcode Error	Invalid opcode is specified
Opcode2 Error	Invalid op2 code is specified
Trap No Error	Invalid trap number is specified
Instruction Format Error	Instruction format is not valid
Segment Allocation Error	Invalid segment address is specified
Elf Program Type Error	Invalid elf program type is specified
Elf Protection Error	Invalid permission file is specified
Elf Program Size Error	Invalid elf program size is specified
Elf Program Address Error	Invalid elf program address is specified
Do nothing	This instruction is not implemented