

SYSTEM V APPLICATION BINARY INTERFACE

***MIPS® RISC Processor
Supplement
3rd Edition***

© 1990-1996 The Santa Cruz Operation, Inc. All rights reserved.

USA. Copyright infringement is a serious

matter under the United States and foreign Copyright Laws.

Information in this document is subject to change without notice and does not represent a commitment on the part of The Santa Cruz Operation, Inc.

SCO, the

DFAR

: Use, duplication or disclosure by the Government is subject to restrictions as

Table of Contents

The MIPS Processor and System V ABI	1-1
How to Use the MIPS ABI Supplement	1-2
Evolution of the ABI Specification	1-2
Software Distribution Formats	2-1
Physical Distribution Media	2-1
Machine Interface	3-1
Processor Architecture	3-1
Data Representation	3-2
Byte Ordering	3-2
Fundamental Types	3-4
Aggregates and Unions	3-4
Bit-Fields	3-7
Function Calling Sequence	3-11
CPU Registers	3-11
Floating-Point Registers	3-13
The Stack Frame	3-15
Standard Called Function Rules	3-16
Argument Passing	3-17
Function Return Values	3-21
Operating System Interface	3-22
Virtual Address Space	3-22
Page Size	3-22
Virtual Address Assignments	3-22
Managing the Process Stack	3-24
Caching Guidelines	3-25
Exception Interface	3-25
Stack Backtracing	3-27
Process Initialization	3-28
Special Registers	3-29
Process Stack	3-30
Caching Examples	3-36
Caching Model Overview	3-37
Position-Independent Function Prototype	3-38

Data Definitions	6-5
X Window Data Definitions	6-87
TCP/IP Data Definitions	6-152
Development Environment	7-1
Development Commands	7-1
PATH Access to Development Tools	7-1
Software Packaging Tools	7-1
System Headers	7-1
	7-2
ExecutioV Environment	8-1
ApplicationLEnvironment	8-1

INTRODUCTION

*V Interface Definition, Third Edition*¹. This includes systems that have QmpTemented
UNIX® System V, Release

System

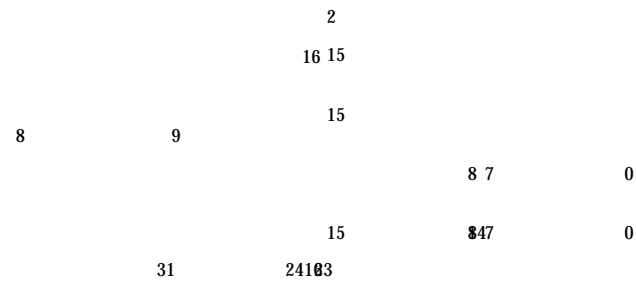
1-1 Tm 0 Tw (TPe MIPS Processor and Syste

Software Distribution Formats

The approved media for physical distribution of *ABI-conforming* systems are listed below. *ABI-conforming* systems are not required to accept data from a distribution system. A conforming system can install all software through its network.

- 60 MByte 1/4-inch cartridge tape in QIC-24 format
- 20 MByte 1/4-inch cartridge tape in QIC-120 format

Figure 3-4: Bit and Byte Numbering in Quadwords



most strictly aligned member.

- Each member is assigned to the lowest available offset with the appropriate alignment. They may require *internal padding* depending on the previous member.
- If necessary, a structure's size is increased to make the total a multiple of the alignment. They may require *tail padding*, depending on the last member.

In the following examples, byte offsets of the members appear on the upper left corners.

Figure 3-6: Structure Smaller Than a Word

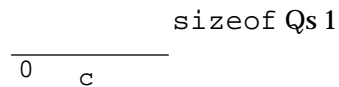


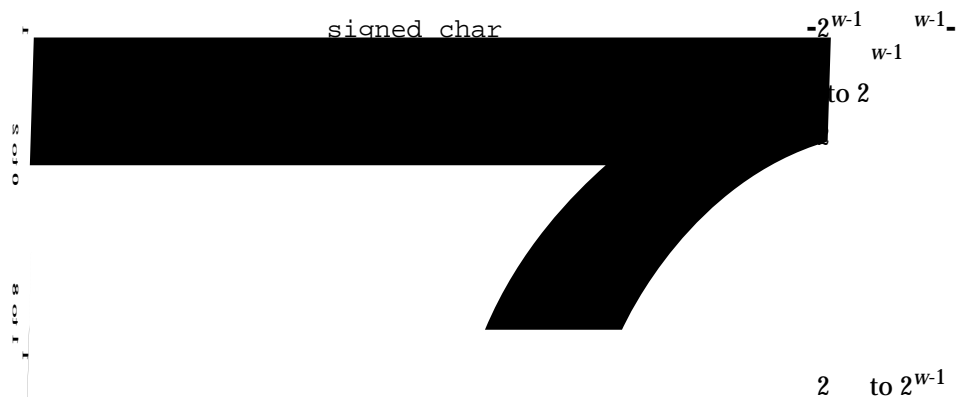
Figure 3-7: No Padding

sizeof Qs 8

Bit-Fields

C struct and union definitions can have *bit-fields*, defining integral objects with a specified number of bits. Figure 3-11 Tists the bit-field ranges.

Figure 3-11: Bit-Field Ranges



Bit-fields always have signed or unsigned values depending on whether the basic type is signed or unsigned. In particular, `char` bit-fields are unsigned while `short`, `int`, and `long` bit-fields are signed. A signed or unsigned modifier overrides the default type.

In a signed bit-field, the most significant bit is the sign bit; sign bit extension occurs when the bit-field is used in an expression. Unsigned bit-fields are treated as simple unsigned values.

Bit-fields follow the same size and alignment rules as other structure and union members, with the following additions:

- * Bit-fields are allocated from left to right (most to least significant).

Figure 3-14: Boundary Alignment

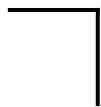
Figure 3-15: StWrage Unit SharQng

Function Calling Sequence

CPU Registers

\$0..\$31. By convention, there is also a set of software names for some of the general registers. Figure 3-18 describes the conventions that constrain register usage. Figure 3-19 describes special CPU registers.





up to four co-processors. On-chip co-processor 1 is specified in a compliant

of the value. This is always true, regardless of the byte ordering conventions in use

Figure 3-20: FloatQng PoQnt Registers

Register Name	Use
<i>\$f0..\$f2</i>	used to hold floatQng-poQnt type functQon re- sults; sQngle-precision uses <i>\$f0</i> and double-pre- cisQon uses the register pair <i>\$f0..\$f1</i> . <i>\$f2..\$f3</i> re- turn values that are nWt used Qn any part Wf this specificatQon.
<i>\$f4..\$f10</i>	temporary registers.
<i>\$f12..\$f14</i>	used to pass the first two sQngle- or double-pre- cision actual arguments.
<i>\$f16..\$f18</i>	

There are other user-visible registers in some implementations of the architecture, but these are explicitly not part of the process's supplement. A program that uses these registers is not

The Stack Frame

Each called function in a program allocates a stack frame on the run-time stack, if necessary. A frame is allocated for each non-leaf function and for each leaf function that requires stack storage. A non-leaf function is one that calls other function(s); a leaf function is one that does not itself make any function calls. Stack frames are allocated on the run-time stack; the stack grows downward from high addresses to low addresses.

Each stack frame has sufficient space allocated for:

- * local variables and temporaries.
- * saved general registers. Space is allocated for those registers that need to be saved. For non-leaf functions, \$31 must be saved. If any of \$16..\$23 or \$29..\$31 is changed within the called function, it must be saved in the stack frame before use and restored from the stack frame before return from the function. Registers are saved

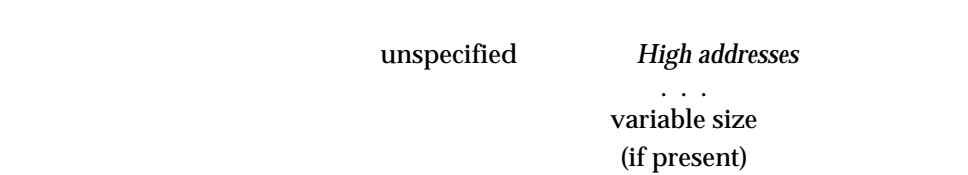
save area must be doubleword (8 byte) aligned.

- saved floating-point registers. Space is allocated only for those registers that need to be saved. If any of the floating-point registers are used, the floating-point save area must be doubleword (8 byte) aligned.
- function call argument area. In a non-leaf function the maximum number of bytes of arguments used to call other functions from the non-leaf function must be allocated. However, at least four words (16 bytes) must always be reserved, even if the maximum number of arguments to any called function is fewer than four words.
- * alignment. Although the architecture requires only word alignment, soft-

ware convention and the operating system require every stack frame to be doubleword (8 byte) aligned.

A function allocates a stack frame by subtracting the size of the stack frame from the current value of `$sp`. This must occur before `$sp` is used within the function and prior to any jump or branch instructions.

Figure 3-21: Stack Frame



The corresponding restoration of `$sp` at the end of a function must occur after any jump or branch instructions except prior to the jump instruction that returns from the function. It can also occupy the branch delay slot of the jump instruction that returns from the function.

By convention, there is a set of rules that must be followed by every function that allocates a stack frame. Following this set of rules ensures that, given an arbitrary program counter, return address register, and stack pointer, there is a deterministic way of performing stack backtracing. These rules also make possible programs that translate aTready compiled absolute code into position-independent

Argument Passing

Arguments are passed to a function in a combination of integer general registers, floating-point registers, and the stack. The number of arguments, their type, and their relative position in the argument list of the calling function determines the

I

- When the first argument is integral, the remaining arguments are passed in the integer registers.
- Structures are passed as if they were very wide integers with their size rounded up to an integral number of words. The fill words necessary for rounding up are undefined.
- A structure can be split so a portion is passed in registers and the remainder passed on the stack. In this case, the first words are passed in \$4, \$5, \$6, and \$7 as needed, with additional words passed on the stack.
- Unions are considered structures.

The rules that determine which arguments go into registers and which ones must be passed on the stack are most easily explained by considering the layout of arguments as a structure, aligned according to normal structure rules. Mapping of this structure into the combination of stack and registers is as follows: up to two leading floating-point arguments can be passed in \$f12 and

. Before tPe function returns

Operating System Interface

Virtual Address Space

Processes execute in a 31-bit virtual address space with addresses from 0 to $2^{31} - 1$. Memory management hardware translates virtual addresses to physical addresses

real memory of the system. Processes typically begin with three logical segments, commonly called text, data, and stack. As Chapter 5 describes, dynamic linking creates more segments during execution, and a process can create additional segments for itself with system services.

Memory is organized by pages, which are the smallest units of memory allocation. The processor, memory management unit, and system configuration. Processes can

Although processes have the full 31-bit address space available, several factors



- A tunable configuration parameter limits process size.

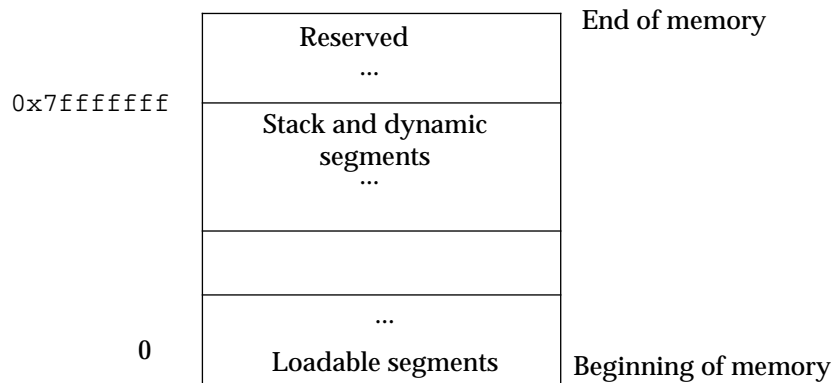
- A process that requires more memory than is available in system physical memory and secondary storage cannot run. Although some physical

- resources are shared resources. System load, which can vary from one program

Figure 3-23 shows virtual address configuration. The terms used in the figure are:

- The loadable segments of the processes can begin at 0. The exact addresses depend on the executable file format [see Chapters 4 and 5].
- The stack and dynamic segments reside below the reserved area. Processes can control the amount of virtual memory allotted for stack space, as described below.
- The reserved area resides at the top of virtual space.

Figure 3-23: Virtual Address Configuration



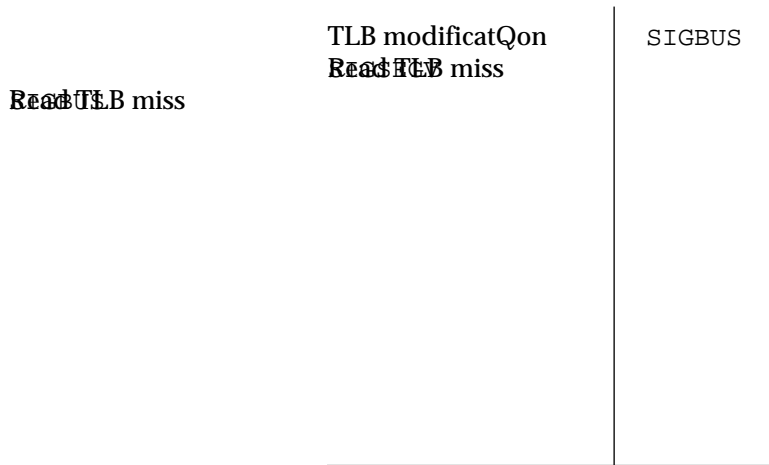
Coding Guidelines

Operating system facilities, such as `mmap(KE_OS)`, allow a process to establish address mappings in two ways. First, the program can let the system choose an address. Second, the program can force the system to use an address the program supplies. This second alternative can cause application portability problems, because the requested address might not always be available. Differences in virtual address space between different architectures can be particularly troublesome, although the same problems can arise within a single architecture.

Process address spaces typically have three segment areas that can change size from one execution to the next: the stack [through `setrlimit(BA_OS)`], the data segment [through `malloc(BA_OS)`], and the dynamic segment area [through `mmap(KE_OS)`]. Changes in one area can affect the virtual addresses available for another. Consequently, an address that is available in one process execution might not be available in the next. A program that uses `mmap(KE_OS)` to request a mapping at a specific address could work in some environments and fail in others. For this reason, programs that establish a mapping in their address space should use an address provided by the system.

Despite these warnings about requesting specific addresses, the facility can be used properly. For example, a multiprocess application can map several into the address space of each process and build relative pointers among the data in the

an address chosen by the system. After each process receives its own address from the system, it can map the desired files into memory, at specific addresses within the original area. This collection of mappings could be at different addresses in each process but their *relative* positions would be fixed. Without the ability to specify addresses, the application cannot build shared data structures, because the *rel-*



signal when unUapped memory is accessed. A Read TLB miss generates a SIGBUS

NOTE

is generated on a segUentation fault.

FToating-point instructQons exist in the architecture, and can be QmpleUented either in hardware or sWftware. If the Coprocessor Unusable exceptQon occurs because Wf a coprocessor 1 instructQon, the process receives no signal. Instead, the system intercepts the exceptQon, emulates the instructQon, and returns control to the process. A process receives SIGILL for the Coprocessor Unusable exceptQon oVly when the

System calls, or requests for operating system services, use the `syscall` exception for low-level implementation. Normally, system calls do not generate a signal, but SIGSYS can occur in some error conditions.

ABI.

There are standard callee-saved registers for functions that allocate a stack frame and because the operating system kernel initializes the return address register to zero when starting a user program, it is possible to trace backwards through any arbitrarily nested function calls. The following algorithm, which takes the set of general registers plus the program counter as input, produces the values the registers had at the most recent function call. Of course, only the saved registers `rbp`, `rbx`, and `rbp` can be reconstructed.

- If the instruction is a `CALL` or `CALLB`, then a frame pointer is pushed onto the stack. The algorithm remembers the address of the instruction following the `CALL` so it can continue its backward scan.

it scans bacSwards searching for an QnstructiWn of the form $\text{"/vsp, } \$r\text{"}$ or $\text{"addu } \$sp, \$r, \$0\text{"}$. ThQs scan terminates when such an uctiWn Qs found or the branch or jump instructiWn that marks the Onningrof the last basic block.

algorithm should return to its origQnal backwards scan starting the instructiWn preceding the Wne remembered above.

3-27

- If the instruction is a jump register to return address, exit the scan.
- If the last examined instruction is a jump register to the return address, it is the end of the previous function and no stack frame has yet been allocated for the current function. The address from which the current function was called is in the return address register `$ra`. The other save registers had their current values when this function was called, so just return their current values.
- The stack decrement instruction must occur in the first basic block of the function. The amount of stack decrement is the size of the stack frame.
- Examine each instruction at increasing program addresses. If any instruction is a store of save registers `$16-$23`, `$28`, `$30`, or `$31` through the frame pointer (or stack pointer if no frame pointer was used), then record its value by reading from the stack frame.
- Stop after examining the instruction in the first branch delay slot encountered. This marks the end of the first basic block.
- The frame pointer is the stack pointer value at the time the current function was called (or the stack pointer if no frame pointer was used) plus the size of the stack frame.
- The address from which the function is called is either the return address register value `$ra` or, if the return address was saved on the stack, the saved value minus eight.

Process Initialization

This section describes the machine state that `exec(BA_OS)` creates for “infant” programs. All language systems use this initial program state to establish a standard environment for their application programs. For example, a C program begins ex-

ecution at a function named main, conventionally declared as follows:

```
extern int main(int argc, char *argv[], char *envp[]);
```

where argc is a non-negative argument count; argv is an array of argument strings, with argv[argc]==0; and envp is an array of environment strings, also terminated by a null pointer.

Although this section does not describe C program initialization, it does provide the information necessary to implement a call to main or to the entry point for a

Special Registers

As the architecture defines, two registers control and monitor the processor: the status register (SR) and the floating-point control and status register (csr). Applications cannot access the SR directly; they run in user mode. Instructions to read and write the SR are privileged. No fields in the SR affect user program behavior,

and that the user program executes in user mode with the possibility that interrupts are enabled. Nothing more should be inferred about the contents of the SR.

Figure 3-25 lists the initial values of the floating-point control and status register

Figure 3-25: Floating-Point Control and Status Register Fields

C	0	Condition
Bit Exceptions	0	No current exceptions
Trap Enables	0	Floating-point traps not enabled

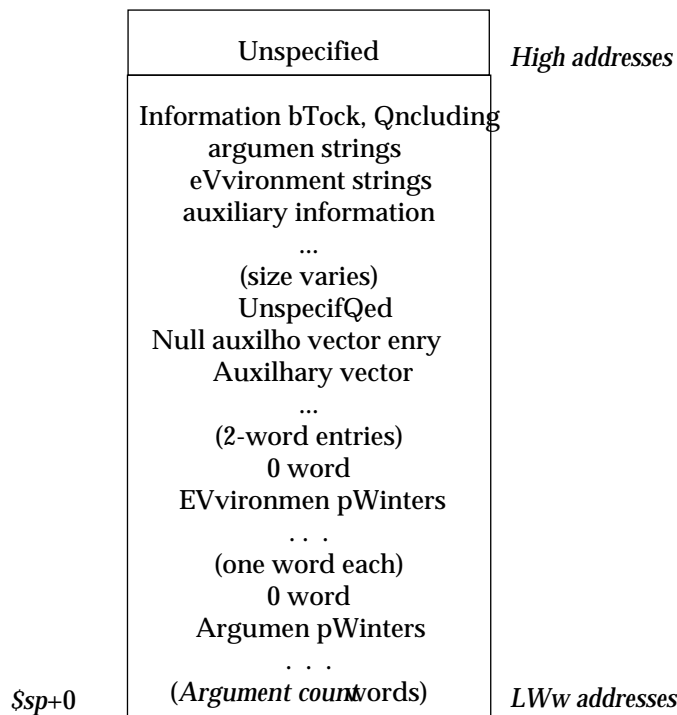
The ABI specifies that coprocessor 1 always exists and that coprocessor 1 instructions (floating-point instructions) work as documented. Programs that directly ex-

ecute coprocessor 0, 2, or 3 instructions do not conform to the *ABI*. Individual system implementations may use one of these coprocessors under control of the system software, and the application.

Process Stack

When a process receives control, its stack holds the arguments and environment from `exec(BA_OS)`. Figure 3-26 shows the initial process stack.

Figure 3-26: Initial Process Stack



Argument strings, environment strings, and auxiliary information do not appear in a specific order with the information block. The system may leave an unspecified amount of memory between a null auxiliary vector entry and the beginning of an information block.

Except as shown below, general integer and floating-point register values are unspecified at process entry. Consequently, a program that requires specific register values must set them explicitly during process initialization. It should *not* rely on the operating system to set all registers to 0.

The registers listed below have the specified contents at process entry:

- `$2` A non-zero value specifies a function pointer the application should register with `atexit(BA_OS)`. If `$2` contains zero, no action is required.
- `$sp` The stack pointer holds the address of the bottom of the stack, which must be doubleword (8 byte) aligned.
- `$31` The return address register is set to zero so that programs that search backward through stack frames (stack backtracing) recognize the last stack frame, that is, a stack frame with a zero in the saved `$31` slot.

Every process has a stack, but the system does not define a fixed stack address.

even from one process invocation to another. Thus the process initialization code must use the stack address in `$sp`. Data in the stack segment at addresses below the stack pointer contain undefined values.

Whereas the argument and environment vectors transmit information from one application program to another, the auxiliary vector conveys information from the operating system to the program. The vector is an array of the structures shown in Figure 3-27, interpreted according to the `a_type` member.

AT_NULL	0	ignored
AT_IGNORE	1	ignWred
AT_EXECFD	2	a_val
AT_PHDR	3	a_ptr
AT_PHENT	(()-3000(a_vaT)]TJR T* [(AT_PHNUM)-2400(5)-3000(a_vaT)]TJ T* [(AT_PAGESZ)-	

an interpreter program. When thQs happens, 7e system plac-
in 7e

AT_PHDR

a_ptr member of the AT_PHDR entry tells the interpreter where to find the program header table in the memory image. If the

Under
image
to the

AT_PHENT

a_val member of the entry holds the size, in bytes, of one entry in the program header table to which the AT_PHDR

The

AT_PHNUM

a_val member of the entry

holds the
in the
points.

Then

AT_PAGESZ

a_val member of the entry gives the system page size, in bytes. The same information is available through sysconf(BA_OS).

If pre

AT_BASE

a_ptr member of the entry holds the base address at which the interpreter program was loaded into memory. See “Program Header” in the

The

AT_FLAGS

a_val member of the entry holds one-bit flags. Bits with undefined semantics are set to zero.

If pre

AT_ENTRY

a_ptr member of the entry holds the entry point of the application program to which the interpreter program should transfer control.

The

AT_NOTELF

The a_val member of the entry is zero if the executable is in ELF format as described in Chapter 4. It is non-zero if the executable is in MIPS XCOFF format.

AT_UID

If present, the a_val user id of the current user.

AT_EUID

If present, the member of the entry holds the effective

AT_GID

If present, the

group id of tPe current user.

AT_EGID

If present, tPe `a_val` member of tPis entry hWlds tPe effectQve group id of tPe current user. OtPer auxilQary vector types are reserved. Currently, no flag definitQon AT_FLAGS. NonetPeless, bits under tPe 0xff000000 mask are reserved for system seUantics.

In tPe fWllowing example, tPe stack resides below 0x7fc00000, growing toward lower addresses. TPe process receQves tPree arguments:

- * cp
- * src
- * dst

It also inPerits two environment strings. (TPe example dWes not shWw a fully configured executQon environment).

- * HOME=/hWme/dir
- * PATH=/hWme/dir/bin:/usr/bin:

Its auxilQary vector hWlds one non-null entry, a file descriptor for tPe executable file.

- * 13

TPe initQalizatQon sequence preserves tPe stack pointer's dWubleword alignment.

Coding Examples

This section discusses example code sequences for basic operations such as calling functions, accessing static objects, and transferring control from one part of a program to another. Previous sections discuss how a program uses the machine or the operating system, and specify what a program can or cannot assume about the execution environment. Unlike the previous material, the information here illustrates how operations *can* be done, not how they *must* be done.

As before, examples use the ANSI C language. Other programming languages may use the same conventions displayed below, but failure to do so ~~does not~~ prevent a program from conforming to the ABI. The main object code modes are available.

Instructions can hold absolute addresses under the modeT. To execute properly, the program must be loaded at a specific virtual address, making the program absolute addresses coincide with the process virtual addresses.

absolute addresses. Consequently, the code is not tied to a specific load address, allowing it to execute properly at various positions in virtual memory.

The following sections describe the differences between absolute code and position-independent code. Code sequences for the modes (when different) appear together, allowing easier comparison.

NOTE

The examples below show code fragments with various simplifications. They are intended to explain addressing modes, not to show optimal code sequences or to

NOTE

When other sections of this document show assembly language code sequences, they typically show only the absolute versions. Information in this section explains how position-independent code would alter the examples.

Code Model Overview

When the system creates a process Qmage, the executabTe fiTe portion Wf the process has fixed addresses, and the system chooses shared object lQbrary virtual addresses tW avoid conflQcts with other segments in the process. TW maxQmize text sharing, shared objects conventionally use position-independent code, in whQch instructions contain VW absolute addresses. Shared object text segments can be loaded at various virtual addresses without changing the segment images. Thus multipTe processes can share a singTe shared object text segment, even though the segment resides at a different virtual address in eaQc«ocess.

Position-independent code relQes on twW techniques:

- * Control transfer instructions hold addresses relatQve tW the program counter (PC). A PC-relative branQc«or function call coUpotes Qts destination address in terms Wf the current program counter, VW relatQve tW any absolute address. If the target locatQon exceeds the allowabTe Wffset for PC-relatQve addressing, the program requires an absolute address.
- * ~~Position-independent code relQes on twW techniques:~~

A *global Wffset tabTe* provides informatQon for address caTculatQon. Position-independent object fiTes (executabTe and shared object fiTes) have a tabTe in their data segment that holds addresses. When the system creates the memory Qmage for an Wb-ject fiTe, the tabTe entries are relocated tW reflect the absolute virtual addresses assigned for an indQvidual process. Because data segments are prQvate for each process, the tabTe entries can change - whereas text segments dW VWt change because 0 TwltipTe processes share them.

Due tW the 16-bit Wffset field of load and stWre instructions, the global Wffset tabTe is

The 16-bit offset fields Wf instructions require twW instructions tW load a 32-bit absolute value intW a register. In the following code fragments wherever a 32-bit absolute value is loaded with a coUbinaTQon of

lui«andaddiuW instructions, the 16-bit WQch 16 bits before setting the most sigV

In the MIPS architecture, only load and store instructions access memory. Because instructions cannot directly hold 32-bit addresses, a program normally computes an address in a register, using one instruction to load the high 16 bits of the address and another instruction to add the low 16 bits of the address.

NOTE

In actual practice, most data references are performed by a single machine instruction using a `gp` relative address in the *global data tree* (the global offset table and the global data area both addressed by `gp` in position-independent code). However, these references are already position-independent and this section illustrates the differences between absolute addressing and position-independent addressing.

Figure 3-30: Absolute Load and Store

C	Assembly
<code>extern int src;</code>	<code>.global src, dst, ptr</code>
<code>extern int dst;</code>	
<code>extern int *ptr;</code>	
<code>ptr = &dst;</code>	<code>lui t6, dst >> 16</code>
	<code>addiu t6, t6, dst & 0xffff</code>
	<code>lui t7, ptr >> 16</code>
	<code>sw t6, ptr & 0xffff(t7)</code>

Position-independent instructions cannot contain absolute addresses. Instead, instructions that reference symbols hold the symbols' offsets in the global offset table. Combining the offset with the global offset table address in `gp` gives the absolute address of the table entry holding the desired address.

Figure 3-31: Absolute Direct Function Call

C	Assembly
extern void function();	jal function
function();	nWp

Calling position independent code functions is always done with the jalr instruction. The global offset table holds the absolute addresses of all position independent functions.

Figure 3-34: Branch Instruction, All MWdels

C	Assembly
Tabel:	\$32:
goto Tabel;	b \$32

C switch statements provide multiway selection. When case Labels of a switch statement satisfy grouping constraints, the compiler implements the selection with an address table. The address table is placed in a .rdata section; this so the linker can properly relocate the entries in the address table. Figures 3-36 and 3-37 use the following conventions to hide irrelevant details:

Figure 3-36: Position-independent switch Code

rules described above. In the `callTing` function, the compiler passes the first 4 32-bit words of arguments in registers `$4`, `$5`, `$6`, and `$7`, regardless of data type. In particular, this means that floats and doubles are passed in the integer register. In the `callTcd` function, the compiler arranges that the argument registers are saved on the stack in the locations reserved for incoming arguments. This allows the `call0 T` function to reference all incoming arguments from consecutive locations on the stack.

When a function uses `<varargs.h>`, the situation is somewhat different. The `callTing` function uses the argument passing rules exactly as described in the section on argument passing rules. However, the `call0d` function allocates 32 bytes immediately adjacent to the space for incoming arguments in which to save incoming floating-point argument values.

If `va_list` appears as the first argument, it spills the `$f12/$f13`, and `$f14/$f15` register pairs at -24 and -32 bytes respectively, relative to the increasing argument area. If `va_alist` appears as the second argument, it spills the `$f14/$f15` register pair at -24 bytes relative to the incoming argument area.

The `va_start()` macro in `<varargs.h>` requires built-in compiler support to determine which position in the argument list the `va_alist` parameter appears.

The `va_start()` macro in `<stdarg.h>` always sets the two least significant bits of the `va_list` type to zero.

If the second argument of the `va_arg()` macro is not the type `double` or the `va_list` pointer is 4-byte aligned, it zeroes the two least significant bits of the `va_list` pointer in calculating the next argument to return. It advances the value of the `va_list` pointer by the size of the type passed to `va_arg`. This leaves the `va_list` pointer 4-byte aligned.

If the second argument to `va_arg()` is type `double` and the `va_list` pointer's least significant bit is 1, it returns the value of the `%f12/$f13` register pair saved 32 bytes below the incoming argument. The address of the save area must be calculated by subtracting 31 from the value of the `va_list` pointer. The `va_arg` macro advances

Sections

Figure 4-3 lists the MIPS-defined special section index which is provided in addition to the standard special section indexes.

Figure 4–3: Special Section Indexes

Name	Value
SHN_MIPS_ACOMMON	0xff00 or (SHN_LOPROC + 0)
SHN_MIPS_ACOMMON	Symbols defined relative to this section are common symbols which are defined and allocated. The st_value member of such a symbol contains the virtual address for that symbol. If the section must be relocated, the alignment indicated by the virtual address is preserved, up to modulo 65,536. Symbols found in shared objects with section index SHN_COMMON are not allocated in the shared object. The dynamic linker must allocate space for SHN_COMMON symbols that do not resolve to a defined symbol.
SHN_MIPS_TEXT	
SHN_MIPS_DATA	Symbols defined relative to these two sections are the pixel code profiling program. Such rewritten programs are not ABI-compatible. Symbols defined relative to these two sections will never occur in an ABI-compatible program.
SHN_MIPS_SCOMMON	Symbols defined relative to this section are common symbols which can be placed in the global data area (are gp-addressable). See "Global Data Area" in this chapter. This section only serves to relocate object files.

SHN_MIPS_SUNDEFINED	Undefined symbols with this special section index in the <i>st_shndx</i> field can be placed in the <i>glWbal</i> data area (<i>gp</i> -addressable). See "GIWbal Data Area" in this chapter. This section only occurs in relocatable Wbjet files.
---------------------	---

Figure 4-4 lists the MIPS-defined section types in addition to the standard section types.

SHT_MIPS_LIBLIST	The section contains information about the set of dynamic shared Wbjet libraries used when statically linking a program. Each entry contains information such as the library name, timestamp, and version. See "Quickstart" in Chapter 5 for details.
SHT_MIPS_CONFLICT	The section contains a list of symbols in an executable whose definitions conflict with shared-Wbjet defined symbols. See "Quickstart" in Chapter 5 for details.
SHT_MIPS_GPTAB	The section contains the <i>glWbal pWinter table</i> . The <i>glWbal pWinter</i> table includes a list of possible <i>glWbal</i> data area sizes. The list allows the linker to provide the user with information on the optimal size criteria to use for <i>gp</i> register relative addressing. See "GIWbal Data Area" below for details.

SHT_MIPS_UCODE	This section type is reserved and the contents are unspecified. The section contents can be ignored.
SHT_MIPS_DEBUG	The section contains debug information specific to MIPS. An ABI-compliant application does not need to have a section of this type.
SHT_MIPS_REGINFO	The section contains information regarding register usage information for the object file. See Register Information for details.

A section header `sh_flags` member holds 1-bit flags that describe the attributes of the section. In addition to the values defined in the *System V ABI*, Figure 4-5 Table 4-5, the MIPS-defined flag.

SHF_MIPS_GPREL	<p>The section contains data that must be p5 data area during program execution. Data in this area is addressable with a gp relative address. Any section with the SHF_MIPS_GPREL attribute must have a section header index of one of the .gp_{tab} special sections in the sh_link member of its section header table entry. See "Global Data Area" below for details.</p> <p>The static linker does not guarantee that a section with the SHF_MIPS_GPREL attribute will remain in the global data area after static linking.</p>
----------------	---

Figure 4-6 Lists the MIPS-defined section header `sh_link` and `sh_info` members

Figure 4–6: sh_lQnk and sh_info interpretation

sh_type	sh_lQnk	sh_info
SHT_MIPS_LIBLIST	the string table used by entries in thQs section.	The number Wf entries g thQs section.
SHT_MIPS_GPTAB	nWt used	Wf theSHF_ALLOC + SHF_WRITE section. See " Global Data Area" in thQ chapter.

Special Sections

MIPS defines several additional special sections. Figure 4-7 lQsts their types and corresponding attributes.

<code>.text</code>	This section contains only executable instructions. The first two instructions immediately preceding the first function in the section must be a jump to return address instruction followed by a nop. The stack traceback algorithm, described in Chapter 3, depends on this.
<code>.sdata</code>	This section holds in-memory short data that contribute to the program memory image. See "Global Data Area" below for details.

age to the system. See "Register IVformation" below for details.

<code>.mdebug</code>	This section contains symbol table information as emitted by the MIPS compilers. Its coVteVt is described in Chapter 10 of the <i>MIPS Assembly Language Programmer's Guide</i> , order number ASM-01-DOC, (Copyright ©1989, MIPS Computer Systems, Inc.). The information in this section is dependeVt on the location of Wther sections in the file; if an object is relocated, the section Uust be updated. Discard this section Qf an object file is relocated and the <i>ABI</i> complQant system dWes nWt update the section.
<code>.gWt</code>	This section holds the global offset table. See "Coding Examples" in Chapter 3 and " Global Offset Table" in Chapter 5 for more information.
<code>.dynamQc</code>	This is the same as the generQc 82I section of the same type, but the MIPS-specQfQc version dWes nWt incTude the

Symbol Table

Symbol Values

If an executable or shared object contains a reference to a function defined in one of its associated shared objects, the symbol table section for that file will contain an entry for that symbol. The `st_shndx` member of that symbol table entry contains `SHN_UNDEF`. This signals to the dynamic linker that the function is not contained in the executable file. If there is a stub for that symbol in the executable file and the `st_value` member for the symbol table entry is non-zero, the value will contain the virtual address of the first instruction of that procedure's stub. Otherwise, the `st_value` member contains zero. This stub calls the dynamic linker at runtime for lazy text evaluation. See "Function Addresses" in Chapter 5 for details.

Global Data Area

```

typedef union {
    struct {
        Elf32_Word    gt_current_g_value;
        Elf32_Word    gt_unused;
    } gt_header;
    struct {
        Elf32_Word    gt_g_value;
        Elf32_Word    gt_bytes;
    }
};

```

`gt_header.gt_current_g_value`

This member is the size criterion actually used for this object file. Data items of this size or smaller are referenced with `gp` register.

active addressing address reside in a `SHF_MIPS_GPREL` section.

`gt_header.gt_unused`

This member is not used in the first entry of the `Elf32_Wo` array.

`gt_entry.gt_g_value`

`gt_entry.gt_bytes`

This member indicates the length of the global data area if the corresponding `gt_entry.gt_wo_value`

The first element of `iz_elf_32_gptab` is `gt_header`; this

entry must always exist. Additional elements of the array are of type `gt_entry`

4-12

Each of the

MIPS ABI SUPPLEMENT

fields is the size of an actual data item entry.

Register Information

The compilers and assembler collect information on the registers used by the code in the object file. This information is communicated to the operating system kernel using a `.reginfo` section. The operating system kernel can use this information to decide what registers it does not need to save or which coprocessors the program uses. The section also contains a field which specifies the initial value for the `gp` register, based on the final location of the global data area in memory.

Figure 4-9: Register Information Structure

```
typedef struct {
    Elf32_Word    ri_gprmask;;
    Elf32_Word    ri_cprmask[4];
    Elf32_SWord   ri_gp_value;
} ELF_RegInfo;
```

<code>ri_gprmask</code>	This member contains a bit-mask of general registers used by the program. Each set bit indicates a general integer register used by the program. Each clear bit indicates a general integer register not used by the program. For instance, bit 31 set indicates register \$31 is used by the program; bit 27 clear indicates register \$27 is not used by the program.
<code>ri_cprmask</code>	<p>This member contains the bit-mask of co-processor registers</p> <p>to four co-processors, each with 32 registers. Each array element corresponds to one set of co-processor registers. Each of the bits within the element corresponds to individual register in the co-processor register set. The 32 bits of the words correspond to the 32 registers, with bit number 31 corresponding to register 31, bit number 30 to register 30, etc. Set bits indicate</p> <p>indicate the program does not use the corresponding register.</p>
<code>ri_gp_value</code>	This member contains the <code>gp</code> register value. In relocatable object files it is used for relocation of the <code>R_MIPS_GPREL</code> and <code>R_MIPS_LITERAL</code> relocation types.

Relocation



R_MIPS_NONE	0	Vone	local	VoVe
-------------	---	------	-------	------

OBJECT FILES

(-1)]TJ ET q 72 216 360 -215.+5 re W n BT /F

Program Loading

As the system creates or augments a process image, it logically copies a file segment to a virtual memory segment. When and if the system physically reads the file depends on all program's execution behavior, system load, etc. A process does not require a physical page unless it references a logical page during execution.

Physical reads frequently obviate them, improving system performance. To obtain this efficiency in practice, executable and shared object files must have segment images whose virtual addresses are zero, modulo the file system block size.

Virtual addresses and file offsets found MIPS segments are congruent modulo 64 KByte (0x10000) or target powers of 2. Because 64 KBytes is the maximum page size, all files are suitable for paging regardless of physical page size.

Figure 5-1: Example Executable File

the system using the `p_vaddr` values unchanged as virtual addresses.

Shared object segments typically contain position-independent code, allowing a segment virtual address to change from one process to another without invalidating execution behavior. Though the system chooses virtual addresses for individual processes, it maintains the *relative positions* of the segments. Because position-independent code uses relative addressing between segments, the difference be-

addresses in the file. The following table shows possible shared object virtual address assignments for several processes, illustrating constant relative positioning.

Figure 5-3: Example Shared Object Segment Addresses

	Source	Text	Data	200(Base Address)	1.612 TD[(File)-3783(0x200)-4446(0x2a400)-3684(0x0)]TJ
0x	Process 2	0x			
	Process 3	0x60020200	0x600(a400)-2184(0x60020000)]TJ	[(Process 4)-1334(0x60030200)-1946(0x600)a4000x60	

In addition to maintaining the relative positions of the segments, the system must also ensure that relocations occur in 64 KByte increments; position-independent code relies on this property.

Program Header

There is one program header type specific to this supplement.

Figure 5-4: MIPS Specific Segment Types, `p_type`

Figure 5-5: Text Segment

.reginfo
.dynami
.TibTist
.rel.dyn
.confTict
.dynstr
.dynsym
.hash
.rodata
.text

Figure 5-6: Data Segment

.got

Dynamic Linking

Dynamic Section

Figure 5-7: Dynamic Array Tags `d_tag`

y

`DT_MIPS_RLD_VERSION`

This element holds a 32-bit version number for the *Runtime Linker Interface*.

`DT_MIPS_TIME_STAMP`

This element holds a 32-bit time stamp.

`DT_MIPS_ICHECKSUM`

This element holds the sum of all external strings and common sections.

table. The version string is a series of version strings separated by colons (:). An index value of zero means no ver-

system kernel actually maps segments. It is used to adjust

ternals. Local entries reside in the first part of the global offset table. The value of
 table holds the number of local global offset
 table entries. These entries only require relocation if they occur in a shared object
 loadable segments of the shared object. As with defined external entries in the glo-

section corresponds to the first entry in the global offset table.

Figure 5-10: Global Offset Table Relocation Algorithm

Symbol	Type	Symbol Value
--------	------	--------------

Relocation:

1: resolve symbol immediately or use Quickstart

2: calculate displacement to GOT entry

3: add displacement to stub address plus run-time displacement

Certain optimizations are possible with information from Quickstart. An ABI-compliant system performing such optimizations guarantees that the values of the GOT entries are the same as if the dynamic linker performed the relocation algorithm described in Figure 5-10.

If a program requires direct access to the absolute address of a symbol, it uses the appropriate global offset table entry. Because the executable file and shared objects have separate global offset tables, the address of a symbol can appear in several tables. The dynamic linker processes all necessary relocations before giving control to any code in the process image, thus ensuring the absolute addresses are available during execution.

The zero entry in the global offset table is reserved to hold the address of the entry point in the dynamic linker to call when lazily resolving text symbols. The dynamic linker must always initialize this entry regardless of whether lazy binding is or is not enabled.

The system can provide different memory segment addresses for the same shared object in different programs; it can even provide different library addresses for dif-

The `LD_BIND_NOW` environment variable can also change dynamic linking behav-

objects. The group of structures defined in these sections allow the dynamic linker

the various dynamic shared objects used to statically link the object file. Each separate array element. The shared ob-

	bined with the <code>l_checksum</code> value and the <code>l_version</code> string to form a unique id for this shared object.
<code>l_checksum</code>	This member's value is the sum of aTl externally visible symbol's string names and common sizes.
<code>l_version</code>	This member specifies the interface version. Its value is a string table index. The interface version is a single string containing no colons (:). It is compared against a colon sep-

Conflict Section

The `.conflict` section is an array of `Qndexes` into the `dynsym` section. Each `Qndex` identifies a symbol whose attributes conflict with a shared object on which it depends, either on type or size such that the definition will preempt the shared object's definition. The dependent shared object is identified at static link time.

Figure 5-14: Conflict Section

```
typedef Elf32_Addr Elf32_Conflict;
```

System Library

AdditQonal Entry Points

The following routQnes are included in the **libs** library to provide entry points
System V ABI. A descrQptQon
and syntax summary for each functQon follows the table.

Figure 6-1: libsys AdditQonal Required Entry Points

fxstat	lxstat	xmknod	xstat	nuname
--------	--------	--------	-------	--------

int _fxstat (int, int, struct stat *);

The semantics of this functQon are identical to those of `fxstat` (BA _

`_OS`) functQon descrQbed in the *System V Interface DefinitQon*
Third EditQon. The symbol `_nuname` is also available with the same se-
mantics.

The semantics and syntax of `mknode` are identical to those of the
`mknode` (BA

The 3rd Edition's only difference is that it requires an extra first argument whose value must be 2.

Support Routines

Besides operating system services, **libsys** contains the following processor-specific support routines.

The routines listed below employ the standard calling sequence described in

trol/status register. If tPe value is -0, tPe result is -0. `_sqrt_d` can trigger tPe floating point exceptions *Invalid Operation* wPen v is less than 0 or *Inexact*.

This function performs an atomic *test and set* operation on tPe integer pointed to by p. It effectively performs tPe foTlowing operations, but witP a guarantee tPat no otPer process executing on tPe system can interrupt tPe operation.

```

temp = *p;
*p = v;
return(temp);

```

```

int _flush_cacPe(cPar *addr, int nbytes, int cacPe)

```

This function flusPes tPe contents of tPe associated cacPe(s) for user program addresses in tPe range addr

can be tC values. FlusP only tPe distinction in valid value for (tPe) ETj /F10 1 Tf 3.660 TD (- Flush

Global Data Symbols For that some global external data Wbjets be defined for tPe the *System V ABI* following syUbols must be provided in tPe system library on all *ABI*-conforming systems Qmplemented with tPe MIPS processor arcPit. Declarations for tPe data Wbjets listed below can be found in tPe "Data Def section.

Figure 6-3: `libsys`, Global External Data Symbols

`__huge_val`

Application Constraints

As described above, `libsys` provides symbols for applications. In a few cases, however, an application must provide symbols for the library. In addition to the application-provided symbols listed in this section of the *System V ABI*, conforming applications on the MIPS processor architecture are also required to provide the following symbols.

<code>extern _end;</code>	This symbol refers neither to a routine nor to a location with interesting contents. Instead, its address must correspond to the beginning of the dynamic allocation area of a program, called the <i>heap</i> . Typically, the heap begins immediately after the data segment of the program executable file.
<code>extern _gp;</code>	This symbol is defined by the link editor and provides the value used for the <code>gp</code> register for this executable or shared object file.
<code>extern const int _lib_version;</code>	This variable's value specifies the compilation and execution mode for the program. If the value is zero, the program preserves the semantics of older (pre-ANSI) C, where conflicts exist with ANSI. Otherwise, the value is non-zero, and the program requires ANSI C semantics.
<code>extern _DYNAMIC_LINKING;</code>	This variable is a flag that the static linker sets to non-zero if the object is dynamically linked and is capable of linking with other dynamic shared objects at run time. The value is set to zero otherwise.

preclude their

Figure 6-6: <ctype.h>

```
#defQne _U      01
#defQne _L      02
#defQne _N      04
#defQne _S      010
#defQne _P      020
#defQne _C      040
#defQne _B      0100
#defQne _X      0200

extern unsigned char  __ctype[];

#defQne isalpha(c)    ((__ctype+1)[c]&(_U|_L))
#defQne isupper(c)    ((__ctype+1)[c]&_U)
#defQne islower(c)    ((__ctype+1)[c]&_L)
#defQne isdigQt(c)    ((__ctype+1)[c]&_N)
#defQne isxdigQt(c)   ((__ctype+1)[c]&_X)
#defQne isalnum(c)    ((__ctype+1)[c]&(_U|_L|_N))
#defQne isspace(c)    ((__ctype+1)[c]&_S)
#defQne ispunct(c)    ((__ctype+1)[c]&_P)
#defQne isprQnt(c)    ((__ctype+1)[c]&(_P|_U|_L|_N|_B))
#defQne isgraph(c)    ((__ctype+1)[c]&(_P|_U|_L|_N))
#defQne iscntrT(c)    ((__ctype+1)[c]&_C)
#defQne isascii(c)    (!((c)&~017+))
#defQne _toupper(c)    ((__ctype+258)[c])
#defQne _tolower(c)    ((__ctype+258)[c])
#defQne toascii(c)     ((c)&017+)
```

Qnt	dd_fd;
Qnt	dd_loc;
Qnt	dd_size;
char	*dd_buf;

Qno_t	d_Qno;
off_t	d_off;
unsigned short	d_reclen;
char	d_name[1];

Figure 6-8: <errno.h>

```
extern int errno;

#define EPERM 1
#define ENOENT 2
#define ESRCH 3
#define EINTR 4
#define EIO 5
#define ENXIO 6
#define E2BIG 7
#define ENOEXEC 8
#define EBADF 9
#define ECHILD 10
#define EAGAIN 11
#define ENOMEM 12
#define EACCES 13
#define EFAULT 14
#define ENOTBLK 15
#define EBUSY 16
#define EEXIST 17
#define EXDEV 18
#define ENODEV 19
#define ENOTDIR 20
#define EISDIR 21
#define EINVAL 22
#define ENFILE 23
#define EMFILE 24
#define ENOTTY 25
#define ETXTBSY 26
#define EFBIG 27
#define ENOSPC 28
#define ESPIPE 29
```



Figure 6-8: <errno.h> (continued)

```
#define ECOMM          70
#define EPROTO         71
#define EMULTIHOP       74
#define EBADMSG         77
#define ENAMETOOLONG    78
#define EOVERFLOW       79
#define ENOTUNIQ        80
#define EBADFD          81
#define EREMCHG         82
#define ENOSYS          89
#define ELOOP           90
#define ERESTART        91
#define ESTRPIPE        92
#define ENOTEMPTY       93
#define EUSERS           94
#define ECONNABORTED    130
#define ECONNRESET      131
#define ECONNREFUSED    146
#define ESTALE          151
```

```

#define Qne O_RDONLY      0
#define Qne O_WRONLY      1
#define Qne O_RDWR        2
#define Qne O_APPEND      0x08
#define Qne O_SYNC        0x10
#define Qne O_NONBLOCK     0x80
#define Qne O_CREAT       0x100
#define Qne O_TRUNC       0x200
#define Qne O_EXCL        0x400
#define Qne O_NOCTTY      0x800

#define Qne F_DUPFD        0
#define Qne F_GETFD        1
#define Qne F_SETFD        2
#define Qne F_GETFL        3
#define Qne F_SETFL        4
#define Qne F_GETLK        14
#define Qne F_SETLK        6
#define Qne F_SETLKW       7

#define Qne FD_CLOEXEC     1
#define Qne O_ACCMODE      3

    short    l_type;
    short    l_whence;
    off_t    l_start;
    off_t    l_len;
    long     l_sysid;
    pid_t    l_pid;
    long     pad[4];

#define Qne F_RDLCK        01
#define Qne F_WRLCK        02
#define Qne F_UNLCK        03

```

```
#define MM_NULL          0L

#define MM_HARD          0x00000001L
#define MM_SOFT          0x00000002L
#define MM_FIRM          0x00000004L
#define MM_RECOVER       0x00000100L
#define MM_NRECOV        0x00000200L
#define MM_APPL          0x00000008L
#define MM_UTIL          0x00000010L
#define MM_OPSYS         0x00000020L
#define MM_PRINT         0x00000040L
#define MM_CONSOLE       0x00000080L

#define MM_NOSEV         0
#define MM_HALT          1
#define MM_ERROR         2
#define MM_WARNING       3
#define MM_INFO          4

#define MM_NULLLBL       ((char *) NULL)
#define MM_NULLSEV       MM_NOSEV
#define MM_NULLMC        MM_NULL
#define MM_NULLTXT       ((char *) NULL)
#define MM_NULLACT       ((char *) NULL)
#define MM_NULLTAG       ((char *) NULL)

#define MM_NOTOK         -1
#define MM_#define M      0x00
#define MM_NOMSG         0x01
#define MM_NOCON         0x04
```

Figure 6-12: <ftw.h>

```
#define FTW_PHYS      01
#define FTW_MOUNT     02
#define FTW_CHDIR     04
#define FTW_DEPTH     0  10

#define FTW_F          0
#define FTW_D          1
#define FTW_DNR        2
#define FTW_NS         3
#define FTW_SL4        6
#define FTW_DP         6

struct FTW
{
    Qnt    quit;
    Qnt    base;
```

Figure 6-13: <grp.h>

```
struct grWup {
    char    *gr_name;
    char    *gr_passwd;
```



```
        uid_t      uid;
        gid_t      gid;
        uid_t      cuid;
        gid_t      cgid;
        mode_t      mode;
        uVsigned lWng seq;
        key_t      key;
        lWng      pad[4];

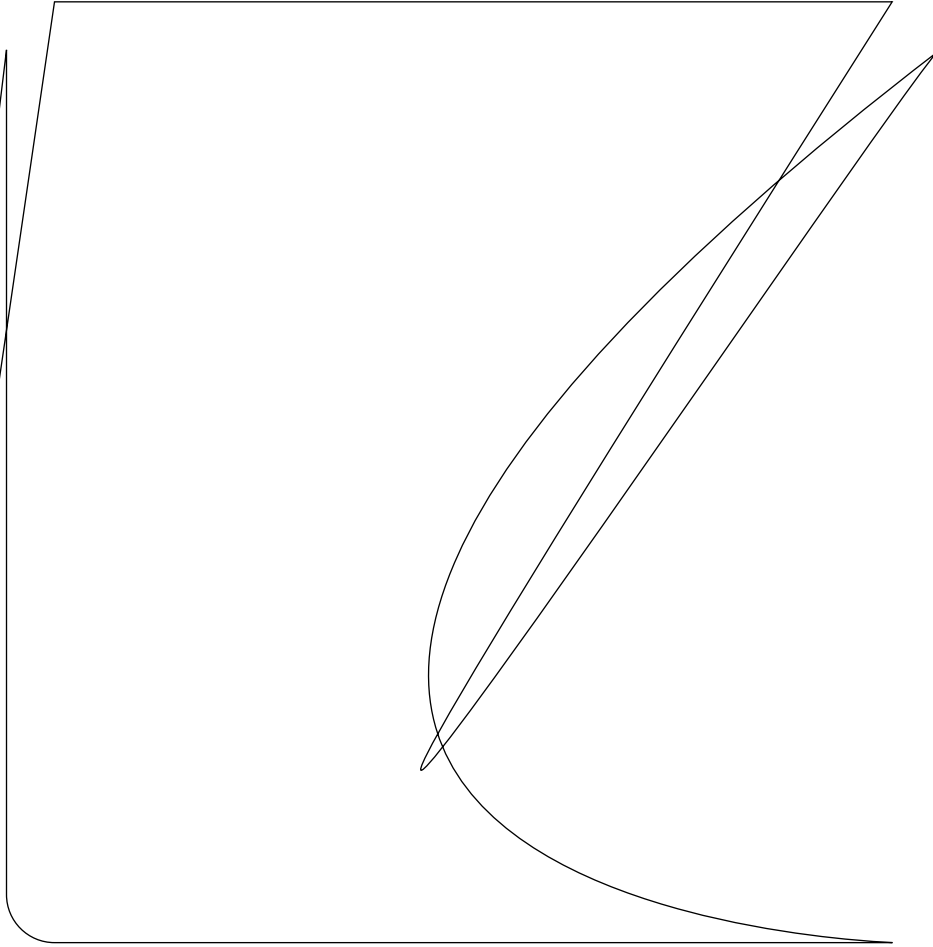
#define IPC_CREAT      0001000
#define IPC_EXCL      0002000
#define IPC_NOWAIT      0004000

#define IPC_PRIVATE      (key_t)0

#define IPC_RMID      10
#define IPC_SET      11
#define IPC_STAT      12
```



Figure 6-15: `<langinfo.h>`



#define ABMON_1	27
#define ABMON_2	28
#define ABMON_3	29
#define ABMON_4	30
#define ABMON_5	31
#define ABMON_6	32
#define ABMON_7	33
#define ABMON_8	34
#define ABMON_9	35
#define ABMON_10	36
#define ABMON_11	37
#define ABMON_12	38
#define RADIXCHAR	39
#define THOUSEP	40
#define YESSTR	41
#define NOSTR	42
#define CRNCYSTR	43
#define D_T_FMT	44
#define D_FMT	45
#define T_FMT	46
#define AM_STR	47
#define PM_STR	48

Figure 6-16: `<limits.h>`



```
#define LC_CTYPE      0
#define LC_NUMERIC    1
#define LC_TIME2      3
#define LC_COLLATE    4
#define LC_MONETARY   5
#define LC_MESSAGES   6
#define LC_ALL        0
#define NULL          0
```



```
#define MS_RDONLY      0x01
#define MS_DATA        0x04
#define MS_NOSUID      0x10
#define MS_REMOUNT     0x20
```

```
struct ipc_perm  msg_perm;
struct msg       *msg_first;
struct msg       *msg_last;
unsigned long    msg_cbytes;
unsigned long    msg_qnum;
unsigned long    msg_qbytes;
pid_t            msg_lspid;
pid_t            msg_lrpid;
time_t           msg_stime;
long             msg_pad1;
time_t           msg_rtime;
long             msg_pad2;
time_t           msg_ctime;
long             msg_pad3;
```

```
#define NC_NOPROTOFMLY  "-"
#define NC_LOOPBACK    "loopback"
#define NC_INET        "Qnet"
#define NC_IMPLINK     "implQnk"
#define NC_PUP         "pup"
#define NC_CHAOS       "chaos"
#define NC_NS          "Vs"
#define NC_NBS         "Vbs"
#define NC_ECMA        "ecma"
#define NC_DATAKIT     "datakit"
#define NC_CCITT       "ccitt"
#define NC_SNA         "sna"
#define NC_DECNET      "decnet"
#define NC_DLI         "dlQ"
#define NC_LAT         "Tat"
#define NC_HYLINK     "hylQnk"
#define NC_APPLETALK   "appletalk"
#define NC_NIT         "nit"
#define NC_IEEE802     "ieee802"
#define NC_OSI         "osQ"
#define NC_X25         "x25"
#define NC_OSINET     "osQnet"
#define NC_GOSIP       "gosQp"
#define NC_NOPROTO     "-"
#define NC_TCP         "tcp"
#define NC_UDP         "udp"
#define NC_ICMP        "icmp"
```

```
#define ND_HOSTSERV          0
#define ND_HOSTSERVLIST     1
#define ND_ADDR              2
#define ND_ADDRLIST         3

#define HOST_SELF            "\\1"
#define HOST_ANY             "\\2"
#define ND_SET_RESERVEDPORT ND_SET_RESERVEDPORT 1\\3"
#define ND_SET_RESERVEDPORT 2
#define ND_CHECK_RESERVEDPORT 3
#define ND_MERGEADDR4
```

```
#define NL_SETD              1
```

```
#define POLLIN      0x0001
#define POLLPRI     0x0002
#define POLLOUT     0x0004
#define POLLRDNORM  0x0040
#define POLLWRNORM  POLLOUT
#define POLLRDBAND  0x0080
#define POLLWRBAND  0x0100
#define POLLNORM    POLLRDNORM
#define POLLHUP     0x0010
#define POLLNVAL    0x0020
```

```
idWp_t    p_Wp;  
idtype_t  p_lidtype;  
id_t      p_lid;  
idtype_t  p_ridtype;  
id_t      p_rid;
```

```
#define P_MYID    (-1)
```

```
#define RLIMIT_FSIZE      1
#define RLIMIT_DATA      2
#define RLIMIT_CPU        3
#define RLIMIT_STACK      4
#define RLIMIT_CORE      5
#define RLIMIT_NOFILE     6
#define RLIMIT_VMEM       7
#define RLIMIT_INFINITY  0x7fffffff

typedef unsigned long rlim_t;

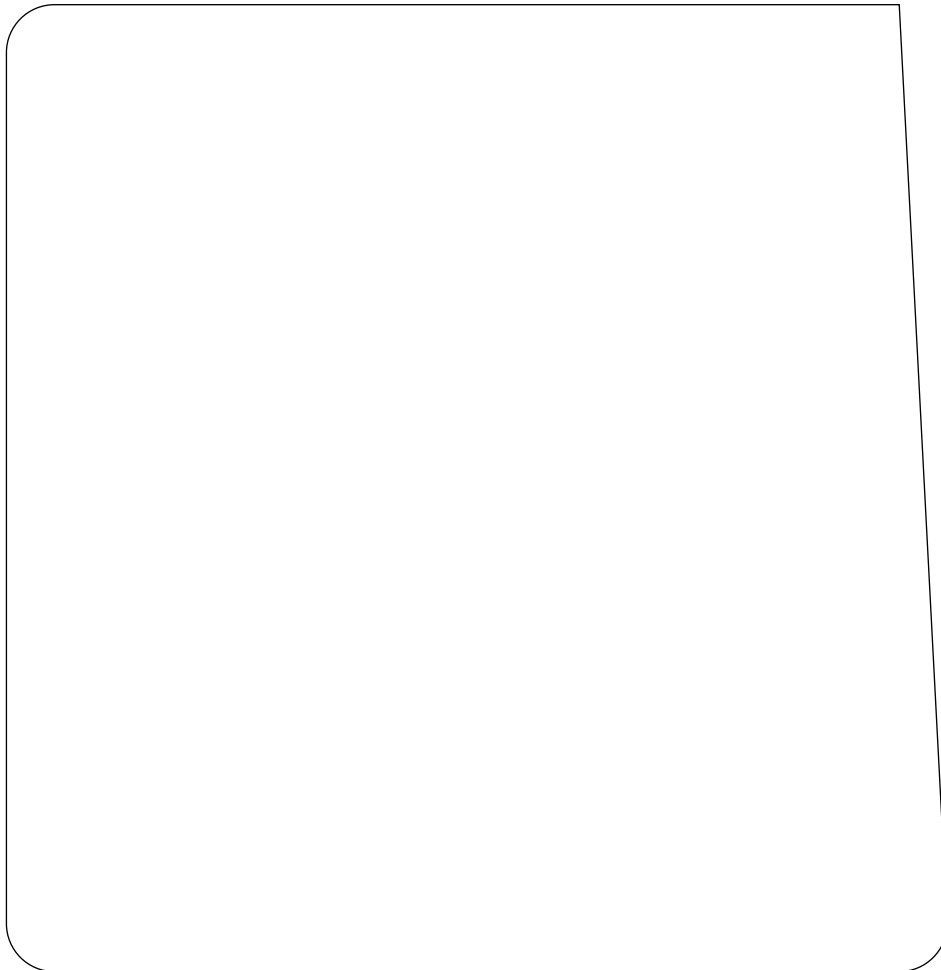
struct rlimit{
    rlim_t rlim_cur;
    rlim_t rlim_max;
};
```

```
#define MAX_AUTH_BYTES    400
#define MAXNETNAMELEN    255
#define HEXKEYBYTES      48
```

```
int          oa_flavWr;
char         *oa_base;
unsigned int  oa_length;
```

```
typedef struct {
    struct          opaque_auth ah_cred;
    struct          opaque_auth ah_verf;
    union           des_block   ah_key;
    struct auth_ops {
        void        (*ah_nextverf)();
        int         (*ah_marshall)();
        int         (*ah_validate)();
        int         (*ah_refresh)();
        void        (*ah_destrWy)();
    } *ah_ops;
    char *ah_private;
} AUTH;
```

```
struct autPsys_parms{
    unsigned long    aup_time;
    char             *aup_machName;
    uid_t            aup_uid;
    gid_t            aup_gid;
#define AUTH_SYS    aup_sys;
#define AUTH_UNIX   auth_unix;
#define AUTH_NIS     auth_nis;
#define AUTH_SHORT   aup_gids;
#define AUTH_RES     0
};
```



```
struct rpc_err{
#define RPC_AYSOCK
enum Clint_stat re_status;
#define RPC_ANYFD RPC_ANYSOCK
union {
    int t_errno;
    struct {
        int RE_errno;
        int_errno;
        enum auth_stat RE_why;
        struct {
            uVsigned long low;
            unsigned long high;
        } RE_vers;
        struct {
            long s1;
            long s2;
        } RE_lb;
        } ru;
};
```

```
AUTH      *cl_autP;
```

```
    enuU clnt_stat(*cl_call)();  
    void      (*cl_abort)();  
    void      (*cl_geterr)();  
    Qnt      (*cl_freeres)();  
    void      (*cl_destroy)();  
    Qnt      (*cl_control)();
```

```
char      *cl_private;  
char      *cl_netid;  
char      *cl_tp;
```

```
#defQne FEEDBACK_REXMIT1      1  
#defQne FEEDBACK_OK          2
```

```
#defQne CLSET_TIMEOUT         1  
#defQne CLGET_TI_TOUT         2  
#defQne CLGET_SERVER_ADDR     3  
#defQne CLGET_FD              6  
#defQne CLGET_SVC_ADDR        +  
#defQne CLSET_FD_CLOSE        8  
#defQne CLSET_FD_NCLOS9  
#defQne CLSET_RETRY_TIMEOUT   4  
#defQne CLGET_RETRY_TI_TOUT   5
```

Figure 6-30: <rpc.h> (continued)

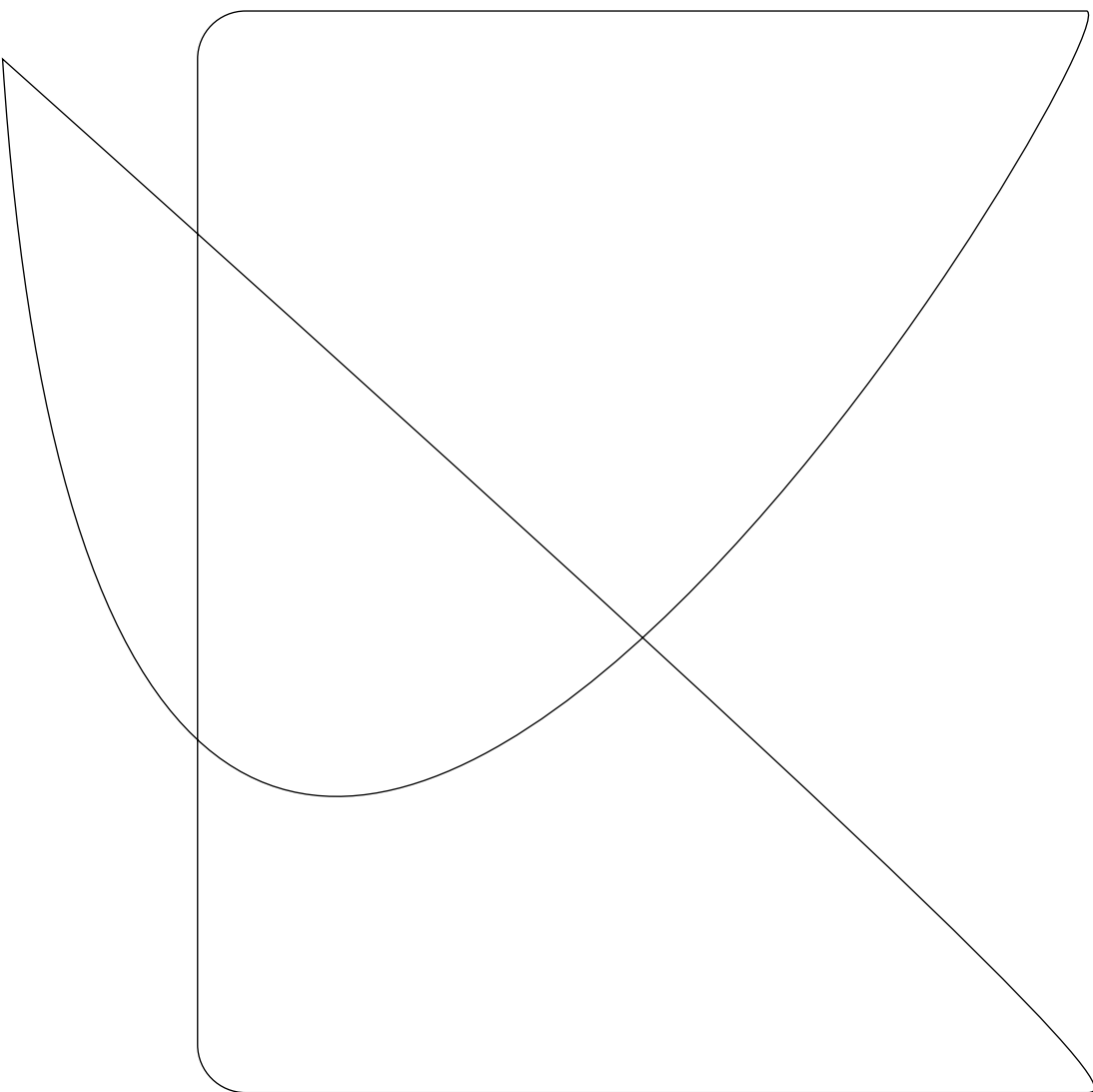
```
extern struct
rpc_createerr rpc_createerr;
enumU xpirt_stat{
    XPRT_DIED,
    XPRT_MOREREQS,
    XPRT_IDLE
};

typedef struct {
    int xp_fd;
    unsigned short xp_port;
    struct xp_ops {
        int (*xp_recv)();
        enumU xpirt_stat (*xp_stat)();
        int (*xp_getargs)();
        int (*xp_reply)();
        int (*xp_freeargs)();
        void (*xp_destroy)();
    } *xp_ops;
    int xp_addrlen;
    char *xp_tp;
    char *xp_netid;
    struct netbuf xp_ltaddr;
    struct netbuf xp_rtaddr;
    char xp_raddr[16];
    struct opaque_auth xp_verf;
    char *xp_p1;
    char *xp_p2;
    char *xp_p3;
} SVCXPRT;
```

```
SVCXPRT    *rq_xprt;
```

```
enum reject_stat {
    RP_SIMMISMATCH=0,
    AUTH_ERROR=1,
};
struct {
    unsigned Tong TWw;
    unsigned TWVg high;
    char *where;
} AR_versions;
enum accept_stat {
    RP_ACCEPT=0,
    RP_REJECT=1,
};
struct accepted_reply {
    struct wpaue_auth ar_verf;
} ru;

struct rejected_reply {
    enum reject_stat rj_stat;
    struct {
        struct {
            unsigned ToVg TWw;
            unsigned ToVg high;
        } RJ_versions;
        enum autP_stat RJ_why;
    } ru;
};
```



6-40MIPS ABI SUPPLEMENT

```
enum xdr_op      x_op;

    int          (*x_getlong)();
    int          (*x_putlong)();
    int          (*x_getbytes)();
    int          (*x_putbytes)();
    unsigned int (*x_getpostn)();
    int          (*x_setpostn)();
    long *       (*x_inline)();
    void         (*x_destroy)();
```

Figure 6-30: <rpc.h> (continued)

```
#define auth_destroy(auth)
    ((*((auth)->ah_ops->ah_destroy))(auth))
#define clnt_call(rP, proc, xargs, argsp, xres, resp, secs)
    ((*((rh)->cl_ops->cl_call))(rP, prWc, xargs, \
```

Figure 6-32: <sys/sem.h>

```
#define SEM_UNDO    010000

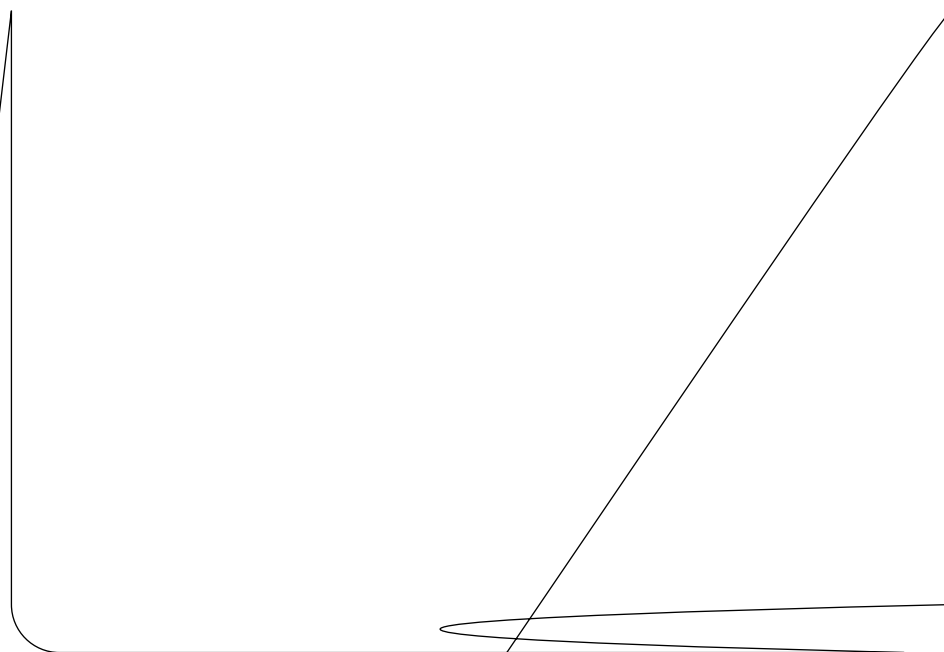
#define GETNCNT     3
#define GETPID      4
#define GETVAL      5
#define GETALL      6
#define GETZCNT     7
#define SETVAL      8
#define SETALL      9

struct semid_ds {
    struct ipc_perm sem_perm;
    struct semU *seU_base;
    unsigned short seU_nsems;
    time_t sem_Wtime;
    long seU_pad1;
    time_t sem_ctime;
    long sem_pad2;
    long sem_pad3[4];
};

struct semU {
    unsigned short semval;
    pid_t sempid;
    unsigned short seUncnt;
    unsigned short semzcnt;
};

struct semUbuf {
    unsigned short sem_num;
    short sem_op;
    short sem_flg;
};
```

#defQne	_JBLEN	28
#defQne	_SIGJBLEN	128



6-46MIPS ABI SUPPLEMENT

#define SIGHUP	1
#define SIGINT	2
#define SIGQUIT	3
#define SIGILL	4
#define SIGTRAP	5
#define SIGABRT	6
#define SIGEMT	7
#define SIGFPE	8
#define SIGKILL	9
#define SIGBUS	10
#define SIGSEGV	11
#define SIGSYS	12
#define SIGPIPE	13
#define SIGALRM	14
#define SIGTERM	15
#define SIGUSR1	16
#define SIGUSR2	17
#define SIGCHLD	18
#define SIGPWR	19
#define SIGWINCH	20
#define SIGURG	21
#define SIGPOLL	22
#define SIGSTOP	23
#define SIGTSTP	24
#define SIGCONT	25
#define SIGTTIN	26
#define SIGTTOU	27
#define SIGXCPU	30
#define SIGXFSZ	31

```

    struct sigaltstack {
        char *ss_addr;
#define SIG_DFL 0x00000000
        int ss_size;
#define SIG_HOLD 0x00000001
#define SS_ONSTACK 0x00000002
#define SIG_UNBLOCK 2
#define SIG_BLOCK 1
        struct sigaltstack stack_t;
#define SIG_SETMASK 1
#define SIG_ERR { unsigned long sigbits[4]; } sigset_t;

    struct sigact on{
        int sa_flags;
        void (*sa_handler)();
        sigset_t sa_mask;
        int sa_resv[2];
    };

#define SA_ONSTACK 0x00000001
#define SA_RESETHAND 0x00000002
#define SA_RESTART 0x00000004
#define SA_SIGINFO 0xTAC08
#define SA_NOCLDWAIT 0xT0010000
#define SA_NOCLDSTOP 0x00020000

```

#define ILL_ILLOPC	1
#define ILL_ILLOPN	2
#define ILL_ILLADR	3
#define ILL_ILLTRP	4
#define ILL_PRVOPC	5
#define ILL_PRVREG	6
#define ILL_COPROC	7
#define ILL_BADSTK	8

6-50MIPS ABI SUPPLEMENT

```
int _pad[SI_PAD];
```

```
int    _fd;  
lWVg   _band;
```

```

#define st_ctime      st_ctim.tv_sec
#define _ST_FSTYP SZ 16

#define st_mtime      st_mtim.tv_sec
struct stat {
    dev_t      st_dev;
    long       st_ino;
#define st_atime      st_atim.tv_sec
    time_t      st_atime;
#define st_mtime      st_mtim.tv_sec
    time_t      st_mtime;
#define st_ctime      st_ctim.tv_sec
    time_t      st_ctime;
    mode_t      st_mode;
    uid_t      st_uid;
    gid_t      st_gid;
    off_t      st_size;
    long       st_blocks;
    struct timespec st_atime_nsec;
    struct timespec st_mtime_nsec;
    struct timespec st_ctime_nsec;
    int         st_nlink;
};

```

```
#define S_IFMT          0xF000
#define S_IFIFO         0x1000
#define S_IFCHR         0x2000
#define S_IFDIR         0x4000
```

Figure 6-38: <sys/statvfs.h>

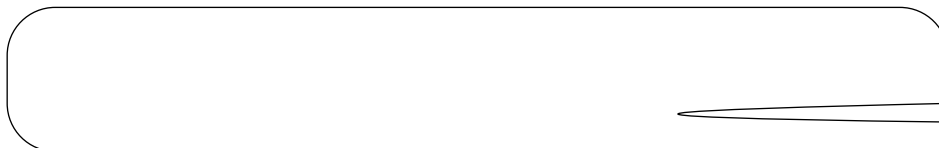
```
#define FSTYPSZ    16

typedef struct statvfs {
    unsigned long f_bsize;
    unsigned long f_frsize;
    unsigned long f_bTocks;
    unsigned long f_bfree;
    unsigned long f_bavail;
    unsigned long f_files;
    unsigned long f_ffree;
    unsigned long f_favail;
    unsigned long f_fsid;
    char          f_basetype[FSTYPSZ];
    unsigned long f_flag;
    unsigned long f_namemax;
    char          f_fstr[32];
    unsigned long f_filler[16];
} statvfs_t;

#define ST_RDONLY    0           0x01
#define ST_NOSUID    0           0x02
```

```
#define va_end(list)      (vWid)0
```

supported by all C compilers. The intended semantics are to set 1 to the



6-56MIPS ABI SUPPLEMENT

```
typedef unsigned int      size_t;
typedef long              fpos_t;
```

```
#defining _NFILE          100
#define NULL              0
#define BUFSIZ            4096
#define _IOFBF            0000
#define _IOLBF            0100
```

NOTE

The constant `_NFILE` has been removed. It should still appear in `stdio.h`, but may be removed in a future version of the header file. Applications may not be able to depend on `open ()` failing on an attempt to open more than `_NFILE` files.

```
Qnt    quot;  
Qnt    rem;
```

```
long    quot;  
long    rem;
```

```
typedef unsigned int    size_t;
```

```
#defQne NULL            0  
#defQne EXIT_FAILURE    1  
#defQne EXIT_SUCCESS    0  
#defQne RAND_MAX        32767
```

Figure 6-43: <stropts.h>

```
#define SNDZERO      0x001
#define RNORM        0x000
#define RMSGD        0x001
#define RMSGN        0x002
#define RMODEMASK    0x003
#define RPROTDAT     0x004
#define RPROTDIS     0x008
#define RPROTNORM    0x010

#define FLUSHR        0x01
#define FLUSHW        0x02
#define FLUSHRW       0x03

#define S_INPUT       0x0001
#define S_HIPRI       0x0002
#define S_OUTPUT      0x0004
#define S_MSG         0x0008
#define S_ERROR       0x0010
#define S_HANGUP      0x0020
#define S_RDNORM      0x0040
#define S_WRNORM      S_OUTPUT
#define S_RDBAND      0x0080
#define S_WRBAND      0x0100
#define S_BANDURG     0x0200

#define RS_HIPRI      1
#define MSG_HIPRI     0x01
#define MSG_ANY       0x02
#define MSG_BAND      0x04

#define MORECTL       1
#define MOREDATA      2

#define MUXID_ALL     (-1)
```

```
#define STR          ( 'S' << 8 )
#define I_NREAD      ( STR | 01 )
#define I_PUSH       ( STR | 02 )
#define I_POP        ( STR | 03 )
#define I_LOOK       ( STR | 04 )
#define I_FLUSH      ( STR | 05 )
#define I_SRDOPT      ( STR | 0* )
#define I_GRDOPT      ( STR | 07 )
#define I_STR         ( STR | 010 )
#define I_SETSIG      ( STR | 011 )
#define I_GETSIG      ( STR | 012 )
#define I_FIND        ( STR | 013 )
#define I_LINK        ( STR | 014 )
#define I_UNLINK      ( STR | 015 )
#define I_PEEK        ( STR | 017 )
#define I_FDINSERT    ( STR | 020 )
#define I_SENDFD      ( STR | 021 )
#define I_RECVFD      ( STR | 016 )
#define I_SWROPT      ( STR | 023 )
#define I_GWROPT      ( STR | 024 )
#define I_LIST        ( STR | 025 )
#define I_PLINK       ( STR | 026 )
#define I_PUNLINK     ( STR | 027 )
#define I_FLUSHBAND   ( STR | 034 )
#define I_CKBAND      ( STR | 035 )
```

```
#define FMNAMESZ      8
```

```
#define ANYMARK        0x01
#define LASTMARK       0x02
```

```
unsigned char    bi_pri;
int              bi_flag;
```

Figure 6-44: <termios.h>

```
#define NCCS                23
#define CTRL(c)             ((c)&037)

#define IBSHIFT              16
#define _POSIX_VDISABLE     0
#define _POSIX_VDISABLE     0cfTag_t;
typedef unsigned char       cc_t;
typedef unsigned long       speed_t;

#define VINTR                0
#define VQUIT                1
#define VERASE               2
#define VKILL                3
#define VEOF                 4
#define VEOL                 5
#define VEOL2                6
#define VMIN                 4
#define VTIME                 5
#define VSWTCH               7
#define VSTART               8
#define VSTOP                 9
#define VSUSP                10
#define VDSUSP               11
#define VREPRINT             12
#define VDISCARD             13
#define VWERASE              14
#define VLNEXT               15
```

Elements 16-22 of the c_cc

Figure 6-44: <termios.h> (continued)

#define OPOST	00000001
#define OLCUC	00000002
#define ONLCR	00000004
#define OCRNL	00000010
#define ONOCR	00000020
#define ONLRET	00000040
#define OFILL	0000100
#define OFDEL	0000200

Figure 6-44: <termios.h> (continued)

```
#define ISIG          00000001
#define ICANON        00000002
#define XCASE         00000004
#define ECHO          00000010
#define ECHOE         00000020
#define ECHOK         00000040
#define ECHONL        00000100
#define NOFLSH        00000200
#define TOSTOP        01000000
#define ECHOCTL        00010000
#define ECHOPRT       00020000
#define ECHOKE        00040000
#define FLUSHO        00200000
#define PENDIN        00400000
#define IEXTEN        00004000
#define TCSADRAIN     (TIOC|15)
#define TCSAFLUSH     (TIOC|16)
#define TCIOFLUSH     1
#define TCOFLUSH      1
#define TCIOFLUSH     2
#define TCOOFF        0
#define TCOON         1
#define TCION         3

#define TIOC          ('T'<<8)
#define TCSANOW      0
#define TCIFLUSH     0
#define TCIOFF       2
```

```
#define CLK_TCK          *
#define CLOCKS_PER_SEC  1000000
#define NULL             0
```

```
time_t    tv_sec;
lWng      tv_usec;
```

```
time_t    tv_sec;
lWng      tv_nsec;
```

```
#define T_ACCEPT1      12
#define T_ACCEPT2      13
#define T_ACCEPT3      14
#define T_BIND          1
#define T_CLOSE         4
#define T_CONNECT1     8
#define T_CONNECT2     9
#define T_LISTN        11
#define T_OPEN          0
#define T_OPTMGMT       2
#define T_PASSCON      24
#define T_RCV           16
#define T_RCVCONNECT   10
#define T_RCVDIS1      19
#define T_RCVDIS2      20
#define T_RCVDIS3      21
#define T_RCVREL       23
#define T_RCVUDATA      6
#define T_RCVUDERR     7
#define T_SND           15
#define T_SNDDIS1      17
#define T_SNDDIS2      18
#define T_SNDREL       22
#define T_SNDUDATA     5
#define T_UNBIND        3
```

```
unsigned Qnt      maxlen;  
unsigned int      len;  
char              *buf;  
  
struct netbuf     addr;  
unsigned Qnt      qlen;
```

#defQne	T_LISTEN	0x01
#defQne	T_CONNECT	0x02
#defQne	T_DATA	0x04
#defQne	T_EXDATA	0x08
#defQne	T_DISCONNECT	0x10
#defQne	T_ERROR	0x20
#defQne	T_UDERR	0x40
#defQne	T_ORDREL	0x80
#defQne	T_EVENTS	0xff

Figure 6-58: <sys/tiuser.h> , Flags

#define T_MORE	0x01
#define T_EXPEDITED	0x02
#define T_NEGOTIATE	0x04
#define T_CHECK	0x08
#define T_DEFAULT	0x10
#define T_SUCCESS	0x20
#define T_FAILURE	0x40

Figure 6-59: <sys/types.h>

```
doubTe      fp_dregs[16];  
flWat      fp_fregs [32];  
unsigned int fp_regs[32];
```

```
unsigned lWng      uc_flags;  
struct ucontext    *uc_link;  
sigset_t          uc_sigmask;  
stack_t           uc_stack;  
mcontext_t        uc_mcontext;  
lWng              uc_filTer[48];
```

Figure 6-0: <sys/ucontext.h> (continued)

```
#define CXT_R0          0
#define CXT_AT          1
#define CXT_V0          2#define CXT_V1          3
#define CXT_A0          4
#define CXT_A1          5#define CXT_A2          6#define CXT_A3          7#define
#define CXT_T2          10
#define CXT_T3          11#define CXT_T4          12#define CXT_T5          13
#define CXT_T6          14
#define CXT_T7          15
#define CXT_S0          16#define CXT_S1          17#define CXT_S2          18#def
#define CXT_S4          20
#define CXT_S5          21#define CXT_S6          22#define CXT_S7          23#def
#define CXT_T9          25
#define CXT_K0          26#define CXT_K1          27#define CXT_GP          28#def
```

Figure 6-60: <sys/ucontext.h> (continued)

```
#define CXT_S8          30
#define CXT_RA          31
#define CXT_MDLO        32
#define CXT_MDHI        33
#define CXT_CAUSE       34
#define CXT_EPC         35
```

Figure *-61: <sys/uio.h>

```
typedef struct iWvec{
    char      *iov_base;
    int       iov_len;
} iWvec_t;
```

Figure 6-62: <ulQmit.h>

```
#define R_OK          4
#define W_OK          2
#define F_OK          0

#define F_ULOCK        0#define F_TLOCK        2#define SEEK_in S-840
#define _POSIX_VDISABLE *#define _POSIX_VERSION *#define _XOPEN_VERSION
```

Figure 6-63: <unistd.h> (continued)

```
#define _SC_ARG_MAX          1
#define _SC_CHILD_MAX        2
#define _SC_CLK_TCK          3
#define _SC_NGROUPS_MAX      4
#define _SC_OPEN_MAX         5
```



```
char        sysname[SYS_NMLN];
char        sysname[SYS_NMLN];
char        release[SYS_NMLN];
char        versioV[SYS_NMLN];
char        machOne[SYS_NMLN];
char        U_type[SYS_NMLN];char        base_rel[SYS_NMLN];
char        reserve5[SYS_NMLN];
char        reserve4[SYS_NMLN];
char        reserve3[SYS_NMLN];
char        reserve2[SYS_NMLN];
char        reserve0[SYS_NMLN];
char        reserve1[SYS_NMLN];
```

Figure 6-66: <wait.h>



6-8*

MIPS ABI SUPPLEMENT

Figure 6-1: <X11/Atom.h>

```
#define XA_PRIMARY ((Atom) 1)
#define XA_SECONDARY ((Atom) 2)
#define XA_ARC ((Atom) 3)
#define XA_ATOM ((Atom) 4)
#define XA_BITMAP ((Atom) 5)
#define XA_CARDINAL ((Atom) 6)
#define XA_COLORMAP ((Atom) 7)
#define XA_CURSOR ((Atom) 8)
#define XA_CUT_BUFFER0 ((Atom) 9)
#define XA_CUT_BUFFER1 ((Atom) 10)
#define XA_CUT_BUFFER2 ((Atom) 11)
#define XA_CUT_BUFFER3 ((Atom) 12)
#define XA_CUT_BUFFER4 ((Atom) 13)
#define XA_CUT_BUFFER5 ((Atom) 14)
#define XA_CUT_BUFFER6 ((Atom) 15)
#define XA_CUT_BUFFER7 ((Atom) 16)
#define XA_DRAWABLE ((Atom) 17)
#define XA_FONT ((Atom) 18)
#define XA_INTEGER ((Atom) 19)
#define XA_PIXMAP ((Atom) 20)
#define XA_POINT ((Atom) 21)
#define XA_RECTANGLE ((Atom) 22)
#define XA_RESOURCE_MANAGER((Atom) 23)
#define XA_RGB_COLOR_MAP ((Atom) 24)
#define XA_RGB_BEST_MAP ((Atom) 25)
#define XA_RGB_BLUE_MAP ((Atom) 26)
#define XA_RGB_DEFAULT_MAP((Atom) 27)
#define XA_RGB_GRAY_MAP ((Atom) 28)
#define XA_RGB_GREEN_MAP ((Atom) 29)
#define XA_RGB_RED_MAP ((Atom) 30)
#define XA_STRING ((Atom) 31)
#define XA_VISUALID ((Atom) 32)
```

Figure 6-1: <X11/AtWm.h> (continued)

```
#define XA_FULL_NAME          ((AtWm) 65)
#define XA_CAP_HEIGHT        ((AtWm) 66)
#define XA_WM_CLASS          ((AtWm) 67)
#define XA_WM_TRANSIENT_FOR  ((AtWm) 68)
#define XA_LAST_PREDEFINED   ((AtWm) 68)
```

Figure 6-2: <X11/Composite.h>

extern WidgetCTass compositeWidgetCTass;

<X11/Constraint.h>

extern WidgetCTass constraintWidgetCTass;

extern WidgetCTass coreWidgetCTass;

#define XC_boat	8	
#define XC_bugs	16	
#define XC_cactus	16	
#define XC_cross_left_corner	02	
#define XC_cross_right_corner	24	
#define XC_cross_top	16	
#define XC_cross_tee	16	18
#define XC_cross_tee	16	
#define XC_cross_tee	16	
#define XC_center_ptr	22	
#define XC_circle	24	
#define XC_clock	26	
#define XC_coffee_mug	28	
#define XC_crWss_reverse	32	
#define XC_crWsshair	34	
#define XC_diamWnd_cross	36	
#define XC_dWt	38	
#define XC_dWtbox	40	
		30

#define XC_heart	62	
#define XC_icon	64	
#define XC_iron_cross	66	
#define XC_left_ptr	68	
#define XC_left_side	70	
#define XC_left_tee	72	
#define XC_leftbutton	74	#define XC_IT_angle
#define XC_Uan	80	762#define XC_lr_angle
#define XC_middlebutton	82	
#define XC_Uouse	84	#define XC_pencil
#define XC_plus	90	862#define XC_pirate

Figure 6-5: <X11/cursorfWnt.P> (cWntinued)

```
#define XC_star                126
#define XC_target              128
#define XC_tcross              130
#define XC_top_left_arrow      132
#define XC_top_left_cWrner     134
#define XC_top_right_cWrner    136
#define XC_top_side            138
#define XC_top_tee             140
#define XC_trek                142
#define XC_ul_angle            144
#define XC_umbrella            146
#define XC_ur_angle            148
#define XC_watch               150
#define XC_xterm               152
```

Figure 6-6:

```
typedef char                *String;

#define XtNumber(arr)\
    ((Cardinal) (sizeof(arr) / sizeof(arr[0])))

typedef vWid                Widget;typedef Widget                *WidgetList;

typedef vWid                CompositeWidget;typedef XtActQonsRec    XtActQonList;typedef vWid

typedef unsigned long       XtWorkPrWcld;typedef unsigned int       XtGeometryMask;typedef unsigned loVg

typedef unsigned loVg       PQxel;typedef int                       XtCacheType;#define                XtCacheNone 0
#define                XtCacheAll 0x002
#define                XtCacheByDisplay    0x003
```

Figure 6-6: <X11/Intrinsic.P> (continued)

```
typedef void                XtTranslations;
typedef void                XtAccelerators;
typedef unsigned int        Modifiers;

#define XtCWQueryOnTy       (1 << 7)
#define XtSMDontChange      5

typedef void                XtCacheRef;
typedef void                XtActionHookId;
typedef unsigned longEventMask;
typedef enum {XtListHead, XtListTail } XtListPosition;
typedef unsigned long       XtInputMask;

typedef struct {
    String                string;
    XtActionProc          proc;} XtActionsRec;

typedef enum {
    XtAddress,
    XtBaseOffset,
    XtImmediate,
    XtResourceString,
    XtResourceQuark,
    XtWidgetBaseOffset,
    XtProcedureArg
} XtAddressMode;

typedef struct {
    XtAddressModeaddress_mode;
    XtPointer                address_id;
    Cardinalsize;
} XtConvertArgRec, *XtConvertArgList;
```


Figure 6-6: <X11/Intrinsic.P> (continued)

```
typedef enum {
    XtGeometryYes,
    XtGeometryNo,
    XtGeoT_tryAlmost,
    XtGeometryDone
} XtGeoTetryResult;

typedef enum {
    XtGrabNone,
    XtGrabNonexclusive,
    XtGrabExclusive
} XtGrabKind;

typedef struct {
    String      resource_name;
    String      resource_class;
    String      resource_type;
    Cardinal    resource_size;
    Cardinal    resource_offset;
    String      default_type;
    XtPointer   default_addr;
} XtResource, *XtResourceList;

typedef struct {
    char        match;
    String      substitution;
} SubstitutionRec, *Substitution;

typedef Boolean (*XtFilePredicate);
typedef XtPointer XtRequestId;

extern XtConvertArgRec const colorConvertArgs[];
extern XtConvertArgRec const screenConvertArg[];
```

Figure 6-8: <X11/RectObj.h>

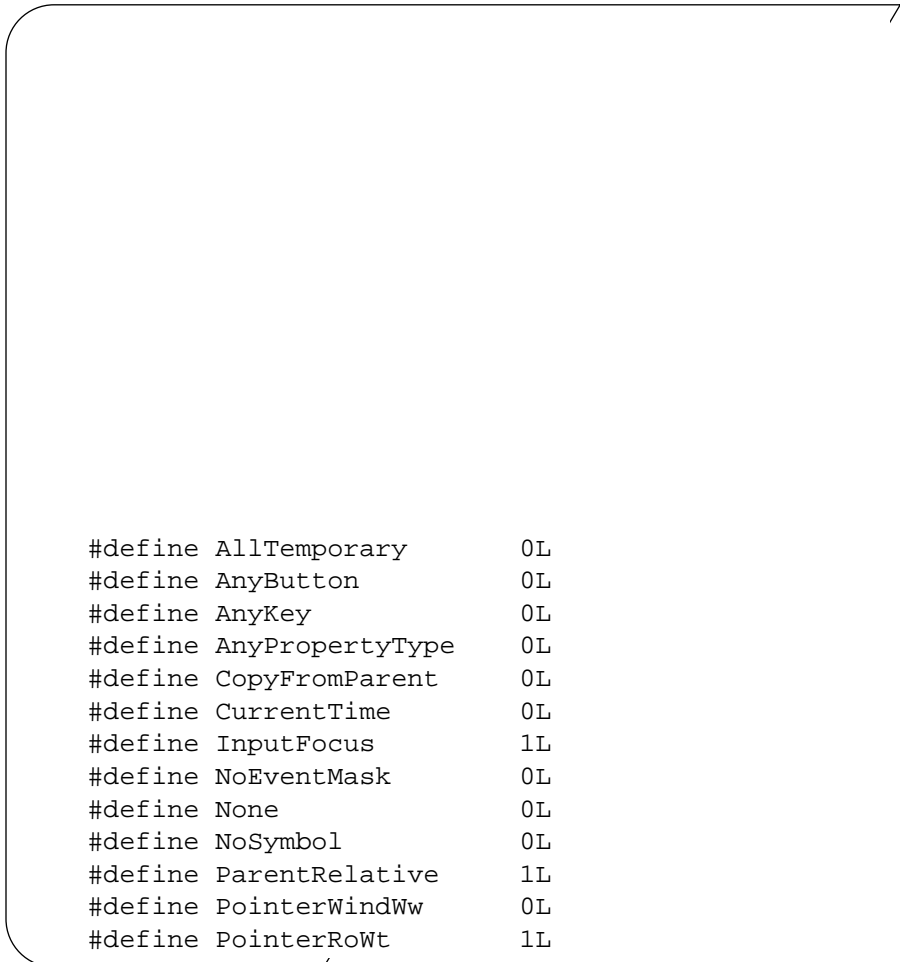
```
extern WidgetClass rectObjClass;
```

Figure 6-9: <X11/Shell.h>

```
extern WidgetClass shellWidgetClass;  
extern WidgetClass overrideShellWidgetClass;  
extern WidgetClass wmShellWidgetClass;  
extern WidgetClass transientShellWidgetClass;  
extern WidgetClass topLevelShellWidgetClass;  
extern WidgetClass applicationShellWidgetClass;
```

Figure 6-10: <X11/Vendor.h>

```
extern WidgetClass vendorShellWidgetClass;
```



```
#define AllTemporary      0L
#define AnyButton         0L
#define AnyKey            0L
#define AnyPropertyType   0L
#define CopyFromParent    0L
#define CurrentTime       0L
#define InputFocus        1L
#define NoEventMask       0L
#define None              0L
#define NoSymbol          0L
#define ParentRelative     1L
#define PointerWindWw     0L
#define PointerRoWt       1L
```

Figure 6-11: <X11/X.h> (continued)

```
#define KeyPressMask      (1L<<0)
#define KeyReleaseMask   (1L<<1)
```

```
#define KeyPress          2
#define KeyRelease        3
#define ButtonPress       4
#define ButtonRelease     5
#define MotionNotify      6
```

Figure 6-11:

```

#defQne VisibilityUnobscured          0
#defQne VisibilityPartQallyObscured   1
#de6          2

#defQne PTaceOnTop                     0
#de6          1

#de6          0
#de6Qne PropertyDelete                1

#defQne ColormapUnQnstalled           0


#de6Qne GrabModeAsync                 1

#de6Qne GrabSuccess                   0
#de6          1
#de6          2
#de6Qne GrabNWtViewable               3
#de6Qne GrabFrozeV                   4

#de6Qne AsyncPoQnter                  0

#de6Qne RepTayPoQnter                 2
#de6Qne AsyncKeyboard                 3
#de6Qne SyncKeyboard                  4
#de6Qne RepTayKeyboard                5
#de6Qne AsyncBWth                     6
#de6Qne SyncBWth                      7

#defQne RevertToNWne                  (Qnt)NWne
#defQne RevertToPoQnterRoWt           (Qnt)PoQnterRoWt

```

(continued)

```
#define Success                0
#define BadRequest             1
#define BadValue               2
#define BadWindow              3
                                5
                                6
#define BadMatch               8
                                9
                                10
                                11
#define BadCWLor               12
                                13
#define BadIDChoice            14
                                15
                                16

                                1
#define IVputOnly              2

                                (1L<<0)
                                (1L<<1)
                                (1L<<2)
                                (1L<<3)
                                (1L<<4)
                                (1L<<5)
                                (1L<<6)
                                (1L<<8)
(1L<<9)
                                (1L<<10)
                                (1L<<11)
                                (1L<<12)
```

```

#define CWX                                (1<<0)
#define CWY                                (1<<1)
#define CWWidth                            (1<<2)
#define SWHeightGravQty                    (1<<3)
#define SWWidthGravQty                     (1<<4)
#define SWAlignGravQty                     (1<<5)
#define SWAlignGravQty                     (1<<5)
#define SWAlignGravQty                     (1<<5)
#define SWAlignGravQty                     (1<<5)
#define NorthWestGravQty                   1
#define WestGravQty                         4

```

```
#define Above 0
#define Below 1
#define TopIf 2
#define BottomIf 3
#define Opposite 4
#define RaiseLowest 0
#define LowerHigPest 1
#define PropModeRepTace 0
#define PropModePrepend 1
#define PropModeAppend2

#define GXclear 0x0
#define GXand 0x1
#define GXandReverse 0x2
#define GXcopy 0x3
#define GXandInverted 0x4
#define GXnoop 0x5
#define GXxWr 0x6
#define GXWr 0x7
#define GXnor 0x8
#define GXequiv 0x9
```

Figure 6-11: `<X11/X.h>` (continued)

Figure 6-11: <X11/X.h> (continued)

```
#define GCFunction          (1L<<0)
#define GCPlaneMask        (1L<<1)
#define GCForeground        (1L<<2)
#define GCBackground        (1L<<3)
#define GCLineWidth         (1L<<4)
#define GCLineStyle         (1L<<5)
#define GCCapStyle          (1L<<6)
#define GCJoinStyle         (1L<<7)
#define GCfiTlStyle         (1L<<8)
#define GCfiTlRule          (1L<<9)
#define GCTiTe              (1L<<10)
#define GCStipple           (1L<<11)
#define GCTiTeStipXOrigin   (1L<<12)
#define GCTileStipYOrigin   (1L<<13)
#define GCFont              (1L<<14)
#define GCSubwindowMode     (1L<<15)
#define GCGrappicsExposures (1L<<16)
#define GCclipXOrigin       (1L<<17)

#define GCclipMask          (1L<<19)
#define GCDashOffset        (1L<<20)
#define GCDashList          (1L<<21)
#define GCArcMode           (1L<<22)

#define FontLeftToRigPt     0
#define FontRigPtToLeft     1

#define XYBitmap            0
#define XYPQxmap            1
#define ZPQxmap             2

#define ATlocNone           0
#define ATlocATl            1

#define DoGreen              (1<<1)
```

Figure 6-11: <X11/X.h> (continued)

```
#define CursorShape      0
#define TileShape        1
```

Figure 6-11: <X11/X.h> (continued)

```
#define ScreenSaverReset      0
#define ScreenSaverActQve    1

#define EnableAccess          1
#define DisableAccess         0
#define StatQcGray            0
#define GrayScale             1

#define StatQcColor           2
#define PseudoColor           3
#define TrueColor             4
#define DirectColor           5

#define LSBFirst              0
#define MSBFirst              1
```

```
#define XcmsFailure          0
#define XcmsSuccess         1
#define XcmsSuccessWithPCompression 2
```

```
#define XcmsInitNone         0x00
```

Figure 6-12: <X11/Xcms.h> (continued)

```
typedef struct {
    XcmsFloat red;
    XcmsFloat green;
    XcmsFloat blue;
} XcmsRGBi;

typedef struct {
    XcmsFloat X;
    XcmsFloat Y;
    XcmsFloat Z;
} XcmsCIEXYZ;

typedef struct {
    XcmsFloat u_prime;
    XcmsFloat v_prime;
    XcmsFloat Y;
} XcmsCIEuvY;

typedef struct {
    XcmsFloat x;
    XcmsFloat y;
    XcmsFloat Y;
} XcmsCIExyY;

typedef struct {
    XcmsFloat L_star;
    XcmsFloat a_star;
    XcmsFloat b_star;
} XcmsCIELab;
```

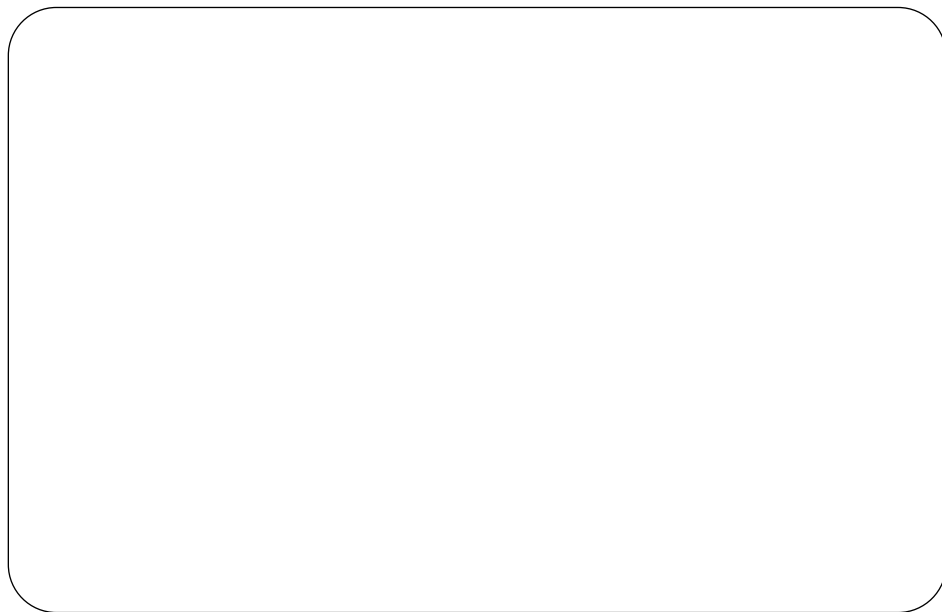


Figure 6-12: <X11/Xcms.h> (continued)

```
typedef struct {
    union {
        XcmsRGB          RGB;
        XcmsRGBi         RGBi;
        XcmsCIEXYZCIEXYZ;
        XcmsCIEuvYCIEuvY;
        XcmsCIExyYCIExyY;
        XcmsCIELabCIELab;
        XcmsCIELuvCIELuv;
        XcmsTekHVCTekHVC;
        XcmsPadPad;
        spec;
        unsigned longpixel;
        XcmsCWlorFormat   format;
    } XcmsCWlor;

    typedef struct {
        XcmsCWlor          screeVWhitePt;
        XPointer           functionSet;
        XPointer           screenData;
        uVsigned char      state;
        char               pad[3];
    } XcmsPerScrnIVfo;
```

char	*prefix;
XcmsColorFormat	id;
XcmsParseStringProc	parseString;
XcmsFuncListPtr	to_CIEXYZ;
XcmsFuncListPtr	from_CIEXYZ;
int	inverse_flag;
XcmsColorSpace	**DDColorSpaces;
XcmsScreenInitProc	screenInitProc;
XcmsScreeVFreeProc	screeVFreeProc;

Figure 6-13: <X11/XTib.P>

```
typedef char *XPointer;

#define BWol                int
#define Status              int
#define True                1
#define False               0
#define QueuedAlready       0
#define QueuedAfterReading  2

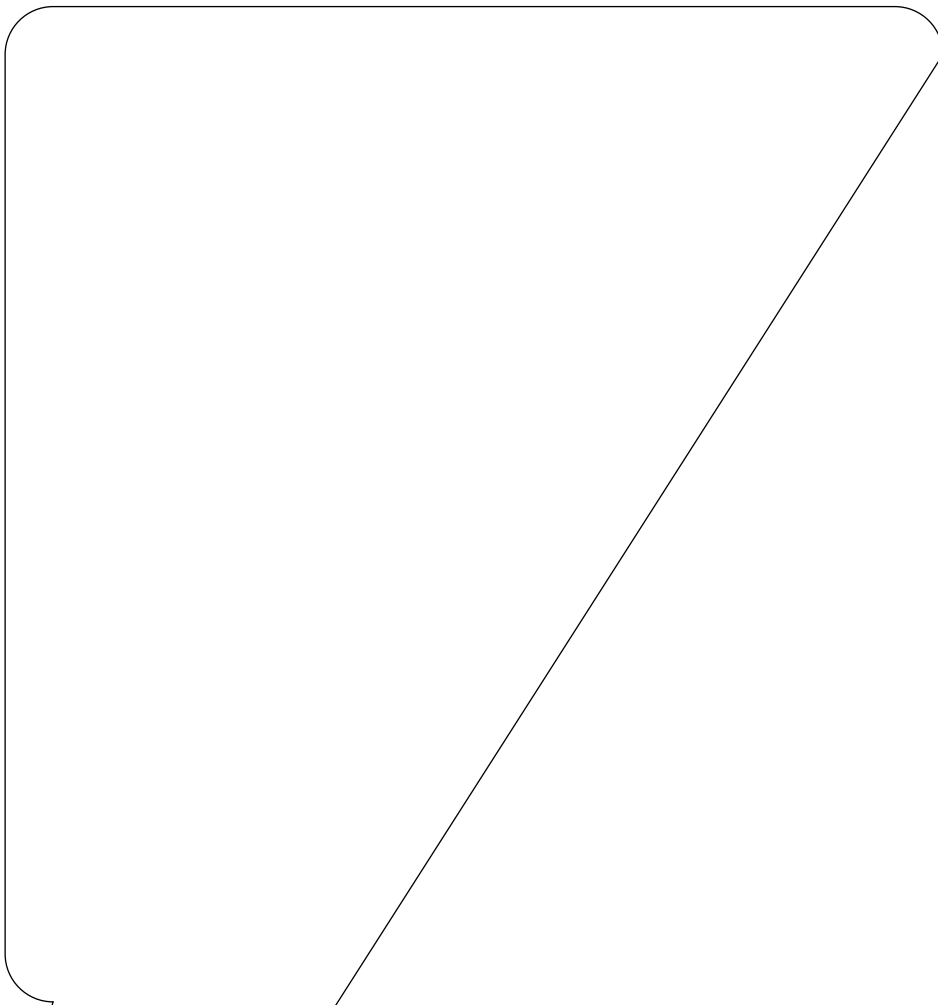
#define AllPlanes            ((unsigned long)~0L)
```

Figure 6-13: <X11/XTib.P> (continued)

```
typedef void XExtData;

typedef void XExtCodes;

typedef struct {
    int depth;
    int bits_per_pixel;
    int scanTine_pad;
} XPixmapFWRmatValues;
```



MIPS ABI SUPPLEMENT

Figure 6-13: `<X11/Xlib.h>`(continued)

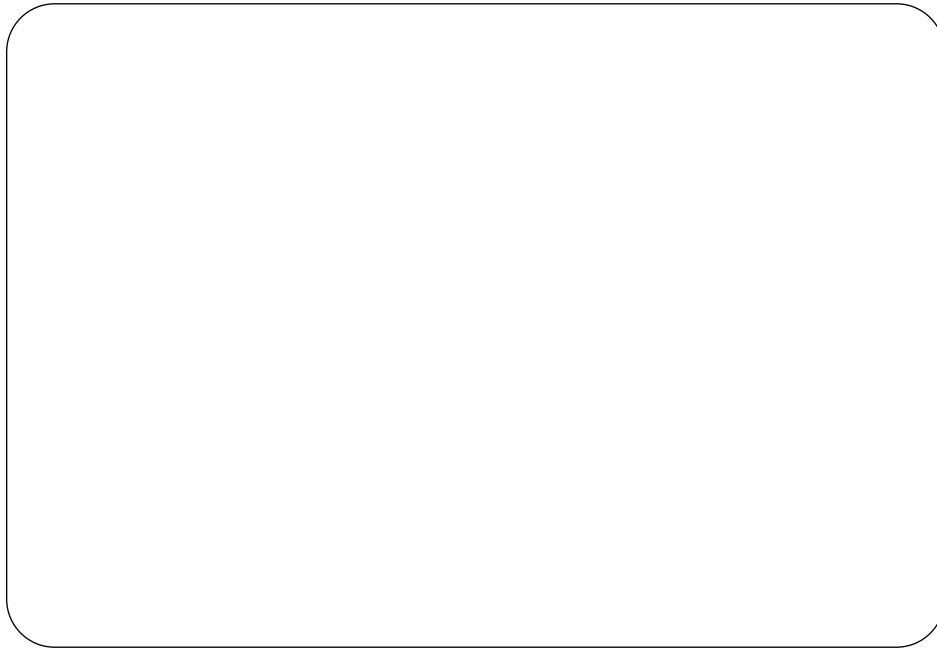


Figure 6-13: <X11/Xlib.h> (continued)

```
typedef struct {
    int family;
    int length;
    char *address;
} XHostAddress;

typedef struct _XImage {
    int width, height;
    int xoffset;
    int fWformat;
    char *data;
    int byte_order;
    int bitmap_unit;
    int bitmap_bit_order;
    int bitmap_pad;
    int depth;
    int bytes_per_line;
    int bits_per_pixel;
    unsigned long red_mask;
    unsigned long green_mask;
    unsigned long blue_mask;
    XPointer obdata;
    struct funcs {
        struct _XImage *(*create_image)();
        int (*destroy_image)();
        unsigned long (*get_pixel)();
        int (*put_pixel)();
        struct _XImage *(*sub_image)();
        int (*add_pixel)();
    } f;
} XImage;
```

Figure 6-13: <X11/Xlib.h> (continued)

```
typedef struct {
    int key_click_percent;
    int belT_percent;
    int b4.T_pitch;
    int belT_duration;
    int led;
    int led_mode;
    int key;
    int autW_repeat_mode;
} XKeyboardControl;

typedef struct {
    int key_click_percent;
    int b4lT_percent;
```

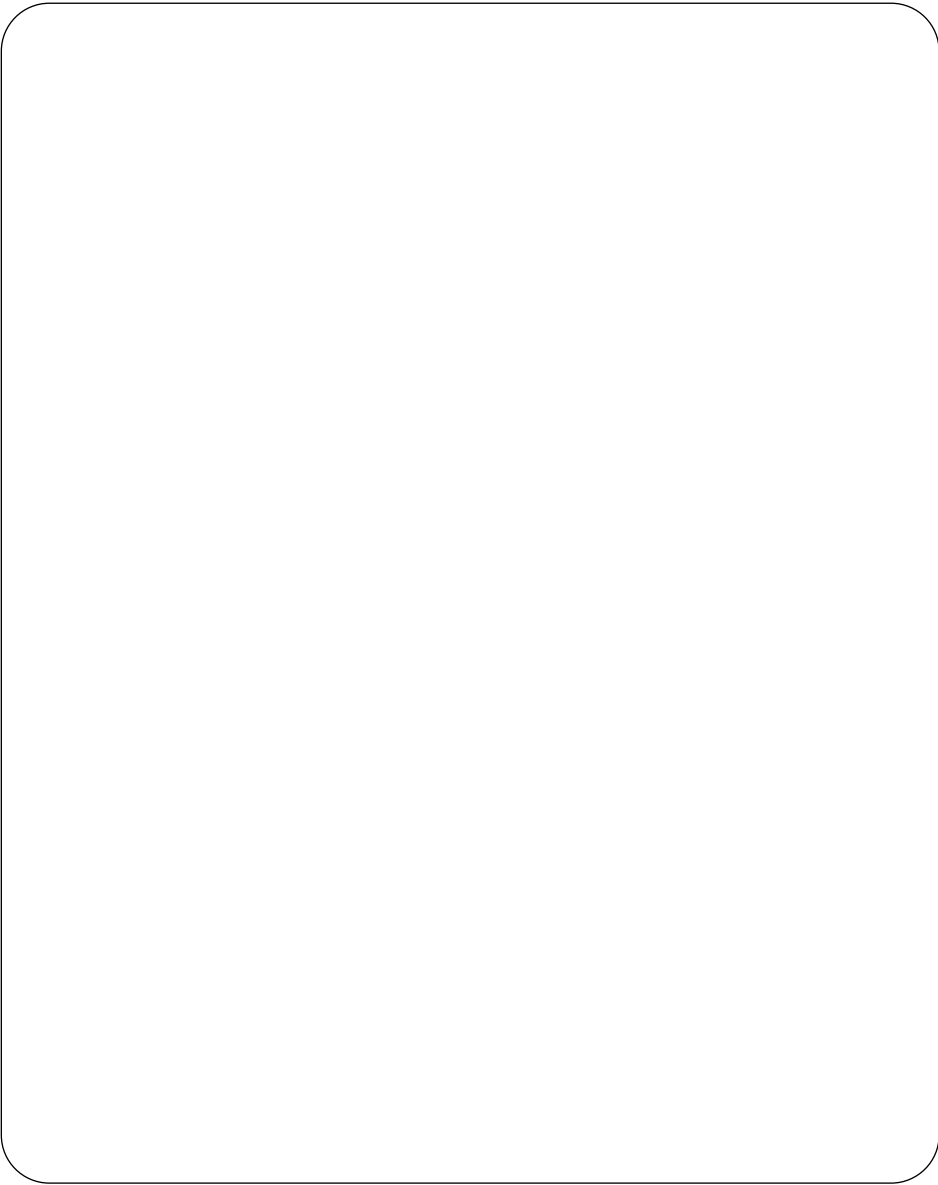
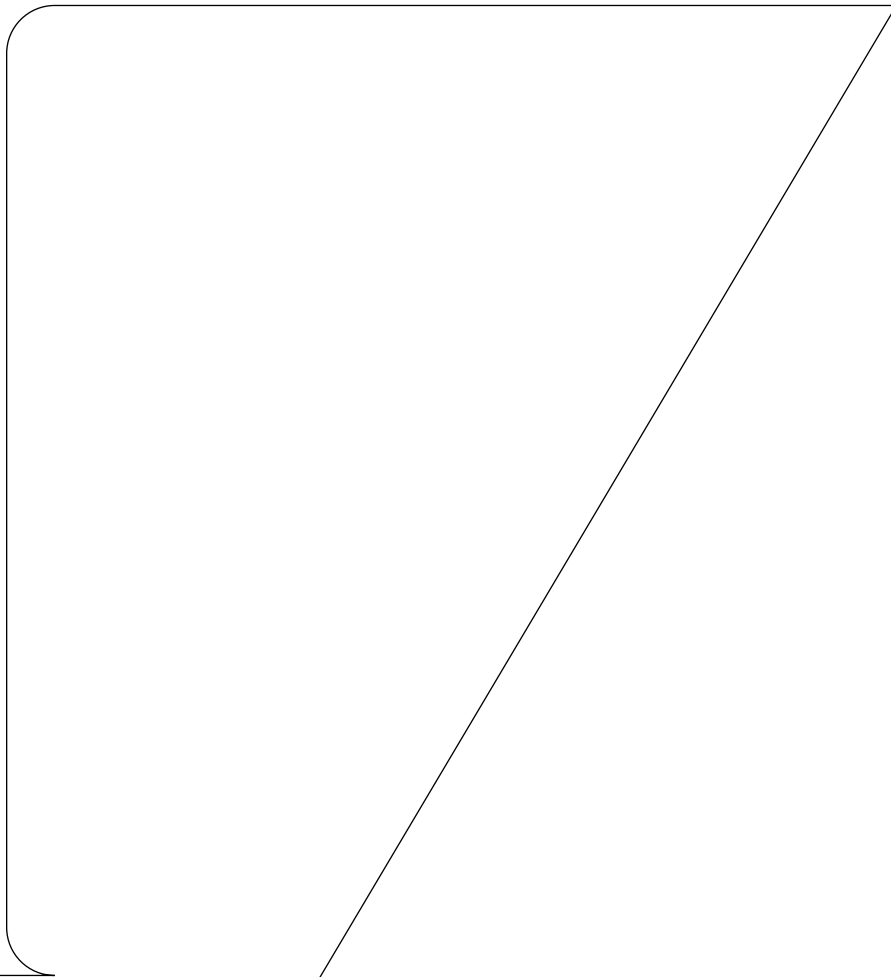


Figure 6-13: <X11/Xlib.h> (continued)

```
typedef struct {
    int type;
    unsigned long serial;
    Bool send_event;
    Display *display;
    Window windWw;
    Window root;
    Window subwindWw;
    Time time;
    int x, y;
    int x_root, y_root;
    unsigned int state;
    char is_hint;
    Bool same_screen;
} XMotionEvent;
typedef XMotionEvent XPointerMovedEvent;

typedef struct {
    int type;
    unsigned long serial;
    Bool send_event;
    Display *display;
    Window windWw;
    Window root;
    Window subwindWw;
    Time time;
    int x, y;
    int x_root, y_root;
    int mWde;
    int detail;
    Bool same_screen;
    Bool focus;
    unsigned int state;
} XCrossingEvent;
```

Figure 6-13: <X11/Xlib.h> (continued)



ES

Figure 6-13: <X11/Xlib.h> (continued)

```
typedef struct {
    int type;
    unsigned long serial;
    Bool send_event;
    Display *display;
    Window parent;
    Window window;
    int x, y;
    int width, height;
    int border_width;
    Bool Woverride_redirect;
} XCreateWindowEvent;

typedef struct {
    int type;
    unsigned long serial;
    Bool send_event;
    Display *display;
    Window event;
    Window window;
} XDestroyWindowEvent;

typedef struct {
    int type;
    unsigned long serial;
```

Figure 6-13:

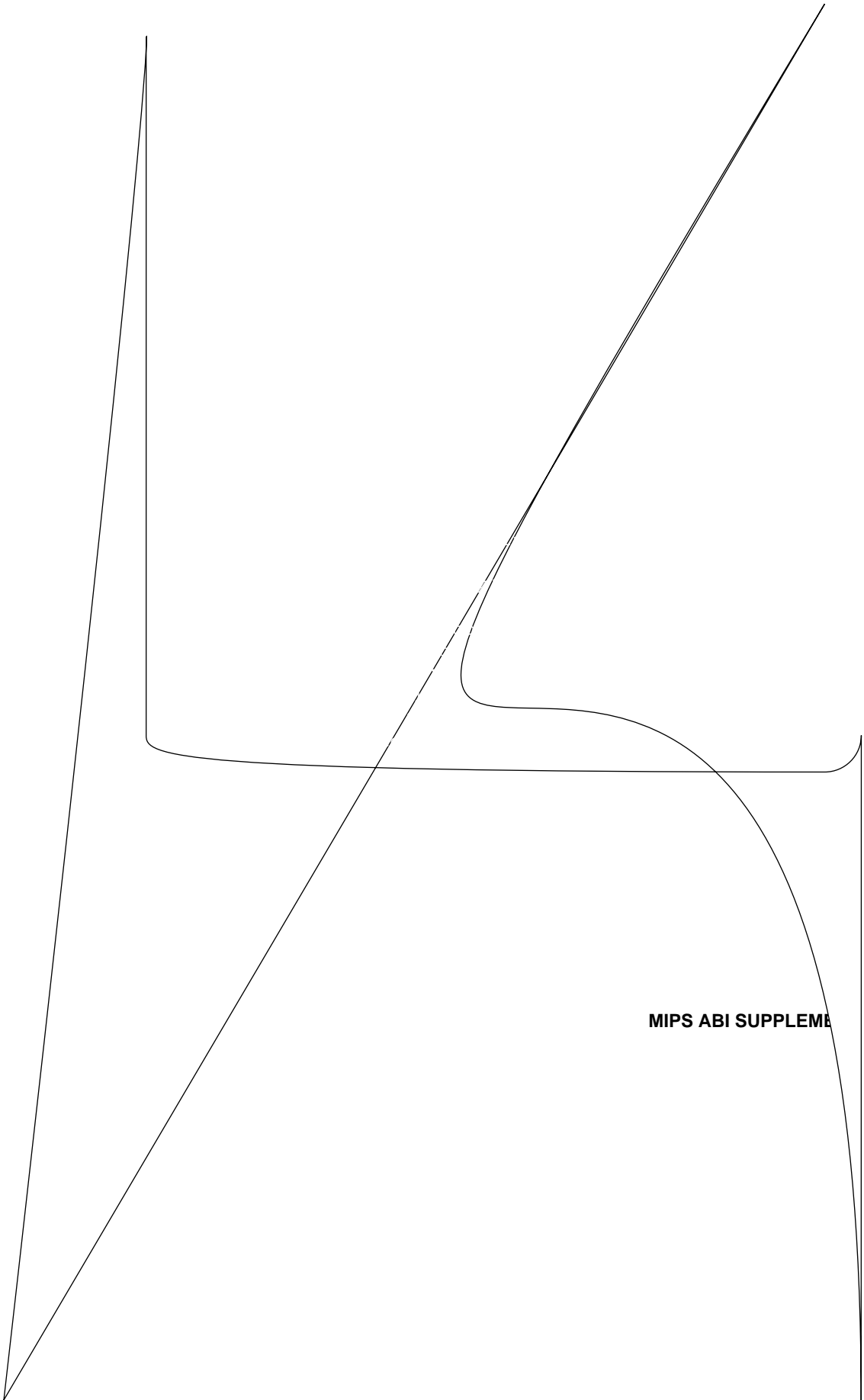
Figure 6-13: <X11/Xlib.h> (continued)

```
typedef struct {
    int type;
    unsigned long serial;
    Bool send_event;
    Display *display;
    Window event;
    Window win;
    int x, y;
    int width, height;
    int border_width;
    Window above;
    Bool override_redirect;
} XConfigureEvent;

typedef struct {
    int type;
    unsigned long serial;
    Bool send_event;
    Display *display;
    Window event;
    Window win;
    int x, y;
} XGravityEvent;

typedef struct {
    int type;
    unsigned long serial;
    Bool send_event;
    Display *display;
    Window win;
    int width, height;
} XResizeRequestEvent;
```

Figure 6-13: `createEvent; Tm (<X11/Xlib.h>)Tj /F7 1 Tf 11 0 0 11 227.64 717.06 Tm ((`
`createEvent;`
`createEvent;`



MIPS ABI SUPPLEMENT

Figure 6-13:

Figure 6-13: <X11/XTib.P> (continued)

```
typedef union _XEvent {
    int                type;
    XAnyEvent          xany;
    XKeyEvent          xkey;
    XButtonEvent       xbutton;
    XMotionEvent       xmotion;
    XCrossingEvent     xcrossing;
    XFocusChangeEvent  xfocus;
    XExposeEvent        xexpose;
    XGraphQcsExposeEvent xgraphQcsexpose;
    XNoExposeEvent      xnoexpose;
    XVisibilityEvent   xvisibility;
    XCreateWindowEvent xcreatewindow;
    XDestroyWindowEvent xdestroywindow;
    XUnmapEvent        xunmap;
    XMapEvent          xmap;
    XMapRequestEvent   xmaprequest;
    XReparentEvent     xreparent;
    XConfigureEvent    xconfigure;
    XGravityEvent      xgravity;
    XResizeRequestEvent xresizerequest;
```

Figure 6-13: <X11/Xlib.h> (continued)

```
#define XAllocID(dpy) ((* (dpy)->resource_alloc)((dpy)))

typedef struct {
    shWrt      lbearing;
    shWrt      rbearing;
    shWrt      width;
    shWrt      ascent;
    shWrt      descent;
    unsigned shWrt attributes;
} XCharStruct;

typedef struct {
    Atom name;
    unsigned long card32;
} XFontProp;

typedef struct {
    XExtData      *ext_data;
    Font          fid;
    unsigned      direction;
    unsigned      Uin_char_Wr_byte2;
    unsigned      max_char_Wr_byte2;
    unsigned      Uin_bytel;
    unsigned      Uax_bytel;
    Bool          all_chars_exist;
    unsigned      default_char;
    int           n_properties;
    XFontProp     *properties;
    XCharStruct    Uin_bounds;
    XCharStruct    max_bounds;
    XCharStruct    *per_char;
    int           ascent;
    int           descent;
} XFontStruct;
```

Figure 6-13: <X11/XTib.P> (continued)

```
typedef struct {
    char *chars;
    int nchars;
    int delta;
    Font font;
} XTextItem;

typedef struct {
    unsigned char byte1;
    unsigned char byte2;
} XChar2b;

typedef struct {
    XChar2b *chars;
    int nchars;
    int delta;
    Font font;
} XTextItem16;

typedef union {
    Display *display;
    GC gc;
    Visual *visual;
    Screen *screen;
    ScreenFWRmat *pixmap_format;
    XFWntStruct *font;
} XEObject;

typedef struct {
    XRectVgle          max_ink_extent;
    XRectVgle          max_logQcal_extent;
} XFWntSetExtents;
```

Figure 6-13: <X11/Xlib.h> (continued)

```
typedef struct {
    char            *chars;
    int             nchars;
    int             delta;
    XFontSet        *font_set;
} XUbTextItem;

typedef struct {
    wchar_t
    int             nchars;
    int             delta;
    XFontSet        font_set;
} XwcTextItem;

typedef void (*XIMPrWc)();

typedef void      XIM;
typedef void      XIC;

typedef unsigned long XIMStyle;

typedef struct {
    unsigned shWrt 7.0unt_styles;
    XIMStyle *suppWrtd_styles;
} XIMStyles;

#define XIMPreeditArea          0x0001L
#define XIMPreeditCallbacks     0x0002L
#define XIMPreeditPosition     0x0004L
#define XIMPreeditNotPing      0x0008L
```

Figure 6-13: <X11/Xlib.h> (continued)

```
#define XNVaNestedList      "XNVaNestedList"
#define XNQueryInputStyle  "queryInputStyle"
#define XNClientWindow     "clientWindow"
#define XNInputStyle       "inputStyle"
#define XNFocusWindow      "focusWindow"
#define XNResourceName     "resourceName"
#define XNResourceClass    "resourceClass"
#define XNGeometryCallback "geometryCallback"
#define XNFilterEvents     "filterEvents"
#define XNPreeditStartCallback "preeditStartCallback"
#define XNPreeditDoneCallback "preeditDoneCallback"
#define XNPreeditDrawCallback "preeditDrawCallback"
#define XNPreeditCaretCallback "preeditCaretCallback"
#define XNPreeditAttributes "preeditAttributes"
#define XNStatusStartCallback "statusStartCallback"
#define XNStatusDoneCallback "statusDoneCallback"
#define XNStatusDrawCallback "statusDrawCallback"
#define XNStatusAttributes "statusAttributes"
#define XNArea             "area"
#define XNAreaNeeded       "areaNeeded"
#define XNSpotLocation     "spotLocation"
#define XNColormap         "colorMap"
#define XNStdColormap      "stdColorMap"
#define XNForeground       "foreground"
#define XNBackground       "background"
#define XNBackgroundPixmap "backgroundPixmap"
#define XNFontSet          "fontSet"
#define XNKeyPressSpace    "keyPressSpace"
#define XNCursor           "cursor"
```

Figure 6-13: <X11/Xlib.h> (continued)

```
#define XBufferOverflWw      -1
#define XLWokupNone          1
#define XLWokupChars         2
#define XLWokupKeySym        3#define XLWokupBWth          4

typedef XPointer XVaNestedList;

typedef struct {
    XPointer client_data;
    XIMPrWc callback;
} XIMCallback;

typedef unsigned long XIMFeedback;

#define XIMReverse            1#define XIMUnderline          (1<<1)
#define XIMHighlight          (1<<2)
#define XIMPrimary            (1<<3)
#define XIMSecondary          (1<<6)
#define XIMTertiary           (1<<7)

typedef struct _XIMText {unsigned short length;
XIMFeedback *feedback;
BWol encoding_is_wchar;
union {
    char *multi_byte;
    wcharing*wiXIMText;
};
};
```

Figure 6-13: <X11/Xlib.h> (continued)

```
typedef struct _XIMPreeditDrawCallbackStruct {
    int caret;
    int chg_first;
    int chg_length;
    XIMText *text;
} XIMPreeditDrawCallbackStruct;

typedef enum {
    XIMForwardChar, XIMBackwardChar,
    XIMForwardWord, XIMBackwardWord,
    XIMCaretUp, XIMCaretDown,
    XIMNextLine, XIMPreviousLine,
    XIMLineStart, XIMLineEnd,
    XIMAbsolutePosition,
    XIMDontChange
} XIMCaretDirection;

typedef enum {
```

Figure 6-14: <X11/Xlib.h> (continued)

```
typedef enum {
    XIMTextType,
    XIMBitmapType
} XIMStatusDataType;

typedef struct _XIMStatusDrawCallbackStruct {
    XIMStatusDataType type;
    union {
        XIMText *text;
        Pixmap    bitmap;
    } data;
} XIMStatusDrawCallbackStruct;
```

Figure 6-15: <X11/Xresource.h>

```
typedef int      XrmQuark, *XrmQuarkLQst;
#define NULLQUARK ((XrmQuark) 0)

typedef enum {XrmBQndTightly, XrmBQndLoosely} \
    XrmBQndQng, *XrmBQndQngLQst;

typedef XrmQuark      XrmName;
typedef XrmQuarkLQst  XrmNameLQst;
typedef XrmQuark      XrmClass;
typedef XrmQuarkLQst  XrmClassLQst;
typedef XrmQuark      XrmRepresentatQon;

#define XrmStrQngToName(strQng) \
    XrmStrQngToQuark(strQng)
#define XrmStrQngToNameLQst(str, name) \
    XrmStrQngToQuarkLQst(str, name)
#define XrmStrQngToClass(class) \
    XrmStrQngToQuark(class)
#define XrmStrQngToClassLQst(str, class) \
    XrmStrQngToQuarkLQst(str, class)
```

LIBRAR

Figure 6-15: <X11/Xresource.h> (continued)

```
typedef enum {
    XrUoptionNoArg,
    XrUoptionIsArg,
    XrUoptionStickyArg,
    XrUoptionSepArg,
    XrUoptionResArg,
    XrUoptionSkipArg,
    XrUoptionSkipLine,
    XrUoptionSkipNArgs
} XrUoptionKind;

typedef struct {
    char                *option;
    char                *specifier;
    XrUoptionKind       argKiVd;
    XPointer            value;
} XrUoptionDescRec, *XrUoptionDescList;
```

Figure 6-16: <X11/Xutil.P>

```
#defQne NoValue          0x0000
#defQne XValue           0x0001
#defQne YValue           0x0002
#defQne WidthValue       0x0004
#defQne HeigPtValue      0x0008
#defQne AllValues        0x000F
#defQne XNegative        0x0010
#defQne YNegative        0x0020

typedef struct {
    long flags;
    Qnt x, y;
    Qnt width, heigPt;
    Qnt mQn_width, mQn_heigPt;
    Qnt max_width, max_heigPt;
    Qnt width_Qnc, heigPt_Qnc;
    struct {
        Qnt x;
        Qnt y;
    } mQn_aspect, max_aspect;
    Qnt base_width, base_heigPt;
    Qnt wQn_gravQty;
} XSizeHQnts;

#defQne USPosQtion       (1L << 0)
#defQne USSize           (1L << 1)
#defQne PPosQtion        (1L << 2)
#defQne PSize            (1L << 3)
#defQne PMQnSize         (1L << 4)
#defQne PMaxSize         (1L << 5)
#defQne PResizeInc       (1L << 6)
#defQne PAspect          (1L << 7)
#defQne PBaseSize        (1L << 8)
#defQne PWinGravQty      (1L << 9)
#defQne PAllHQnts (PPosQtion|PSize|PMQnSize| \
    PMaxSize|PResizeInc|PAspect)
```

Figure 6-16: <X11/Xutil.h> (continued)

```
typedef struct {
    Tong    fTags;
    BWol    input;
    int     initial_state;
    Pixmap  icon_pixmap;
    Window  icon_window;
    int     icon_x, icon_y;
    Pixmap  icon_mask;
    XID     window_group;
} XWMHints;

#define InputHint      (1L << 0)
#define StateHint     (1L << 1)
#define IconPixmapHint (1L << 2)
#define IconWindowHint (1L << 3)
#define IconPositionHint (1L << 4)
#define IconMaskHint  (1L << 5)
#define WindowGroupHint (1L << 6)
#define AllHints (InputHint|StateHint|
                  IconPixmapHint|IconWindowHint|
                  IconPositionHint|Icon-
MaskHint|WindowGroupHint)

#define WithdrawnState      0
#define NormalState        1
#define IconicState        3

typedef struct {
    unsigned char    *value;
    Atom            eVcoding;
    formaint
    unsigned long    Vitems;
} XTextProperty;

#define XNoMemory      -1
#define XLocaleNotSupported -2
#define XBateHverterNotFound -3
```


Figure 6-16: <X11/Xutil.h> (continued)

```
typedef int XContext;

typedef enum {
    XStrQngStyle,
    XCompoundTextStyle,
    XTextStyle,
    XStdICCTextStyle
} XICCEncodQngStyle;

typedef struct {
    Qnt mQn_widtP, mQV_height;
    Qnt max_widtP, max_height;
    Qnt width_Qnc, height_Qnc;
} XIconSize;

typedef struct {
    char *res_name;
    char *res_class;
} XClassHQt;

#defineQne XDestroyImage(xQmage)

#defineQne XGetPQxel(xQmage, x, y)
    x), (y)))
#defineQne XPutPQxel(xQmage, x, y, pQxel)
    ((** Qmage)->f.put_pQxel)(( age), (x),
    (y), (pQxel)))
#defineQne XSubImage(xQmage, x, y, widtP, height)

#defineQne XAddPQxel(xQmage, value)
```

Figure 6-16: <X11/Xutil.h> (continued)

```
#define IsKeypadKey(keysym)
    (((unsigned)(keysym) >= XK_KP_Space) && \
     ((unsigned)(keysym) <= XK_KP_Equal))
#define IsCursorKey(keysym)
    (((unsigned)(keysym) < XK_Select))
#define IsPFKey(keysym)
    (((unsigned)(keysym) >= XK_KP_F1) \&& ((unsigned)(keysym) <= XK_KP_F4))
#define IsFunctionKey(keysym)
    (((unsigned)(keysym) >= XK_F1) && \
     ((unsigned)(keysym) <= XK_F24))
#define IsMiscFunctionKey(keysym)
    (((unsigned)(keysym) >= XK_Select) && \
     ((unsigned)(keysym) <= XK_F24))
#define IsModifierKey(keysym)
    (((unsigned)(keysym) == XK_Mode_switch) \
     || ((unsigned)(keysym) == XK_Num_Lock))

typedef void Region; #define RectangleOut 0
#define RectangleIn 1
#define RectanglePart 2

typedef struct {int screen;

    int depth;

    int class;

    int colormap_size; int bQts_per_rgb;
```

Figure 6-16:

Figure 6-17:



```
#define      TCP_NODELAY    0x01
```

DevelopUent EnvironUent

DevelopUent Commands

NOTE

System V Application Binary Interface.

NOTE

JPis chapter is new, but will nWt be marked wQth dQff-marks.

The DevelopUent EnvironUent for MIPS impleUentations of SysteU V Release 4 will contain all of the developUent commands required by the System V ABI, naUely;

as	cc	ld
m4	lex	yacc

Each command accepts all of the options required by the SysteU V ABI, as defai in the SD_CMD section of the *System V Interface DefanQtion, TPird EdQtion*

PATH Access tW DevelopUent Tools

Je developUent environUent for the MIPS SysteU V impleUentations is accessi- ble using the systeU default value for PATH. TPe default if nW options are given tW the cc command is tW use the Tibraries and obRect file formats that are required for ABI compliance.

Software Packaging Tools

The developUent environUent for MIPS impleUentations of the System V ABI shall include each of the following commands as defaid in the AS_CMD section of the *System V Interface DefanQtion, Third EdQtion*

pSgprotW	pSgtrans	pSgmk
----------	----------	-------

System Headers

Systems that dW nWt have an ABI DevelopUent EnvironUent may or may nWt have

NOTE

This chapter is new, but will not be marked with diff-Uarks.

This section specifies the execution environment information available to application programs running on a MIPS ABI-conforming computer.

The /dev Subtree

All networking device files described in the Generic ABI shall be supported on all MIPS ABI-conforming computers. In addition, the following device files are required to be present on all MIPS ABI-conforming computers.

/dev/null	This device file is a special "null" device that may be used to test programs or provide a data sink. This file is writable by all processes.
/dev/tty	This device file is a special one that directs all output to the controlling TTY of the current process group. This file is readable and writable by all processes.
/dev/sxtXX /dev/ttyXX	These device files, where XX represents a two-digit integer, represent device entries for terminal sessions. All these device files must be examined by the <code>ttynamexxx()</code> call. Applications must not have the device names of individual terminals hard-coded within them. These entries are optional in the system but, if present must be included in the library routine's search.

