WRAMP Instruction Set Description

WRAMP General Purpose Registers

The WRAMP general purpose register file consists of 16 registers, each being 32 bits wide. The hardware imposes special uses on only two of these. Certain software register use conventions have been applied to some of the remaining registers, but in essence, they remain true general purpose registers, as the hardware does not restrict their use.

Register	Description
`$0`	Hardwired zero
`$1 - $13`	General purpose registers
`$sp`	Stack pointer
`$ra`	Return address register

Figure 1: The WRAMP General Purpose Registers

Register zero (denoted `$0`) always contains the value zero. Any writes to this register have the value discarded. This provides a constant source of zero that can be used for comparing and initialising registers.

The fourteenth register is denoted `$sp`. This register is defined by the conventions to be the stack pointer. While the hardware imposes no special conditions on this register, failure to follow this convention may affect the ability of code to interoperate with other software.

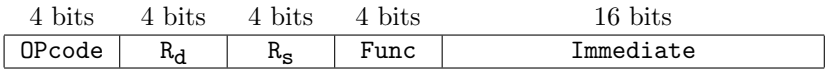
The fifteenth register is denoted `$ra`. It is defined to be the subroutine return address register. When a jump and link instruction is executed this register is loaded with the address of the next instruction after the jump and link. A return from subroutine is performed by executing a jump to register `$ra`, ie. `jr $ra`.

WRAMP Instruction Set Architecture

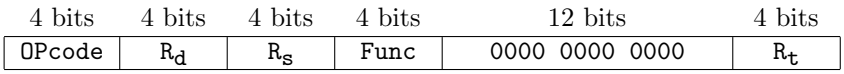
This section contains the details of the WRAMP instruction set. All machine instructions are listed, with their encoding and a brief description of their function. The instructions are grouped into arithmetic instructions, bitwise instructions, test instructions, branch instructions, memory instructions, and special instructions.

Each CPU instruction is a word (32 bits) in length. An instruction is encoded in one of the three formats shown in Figure 2.

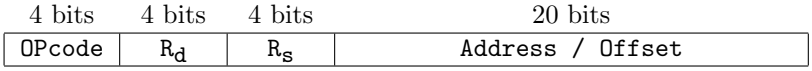
I-Type instruction



R-Type instruction



J-Type instruction



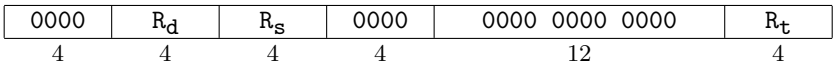
`OPCode`	4 bit operation code
R_d	4 bit destination register specifier
R_s	4 bit source register specifier (first source register)
R_t	4 bit source register specifier (second source register)
`Func`	4 bit function specifier
`Immediate`	16 bit immediate field
`Address / Offset`	20 bit absolute or relative address field

Figure 2: WRAMP Instruction encoding formats

Arithmetic Instructions

Addition

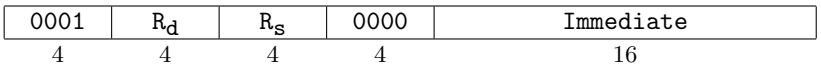
`add` R_d, R_s, R_t



Put the sum of register R_s and register R_t into register R_d. Generate an overflow exception on signed overflow.

Addition, immediate

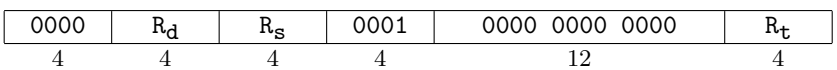
`addi` R_d, R_s, `Immediate`



Put the sum of register R_s and the sign-extended immediate into register R_d. Generate an overflow exception on signed overflow.

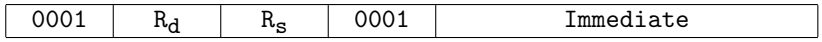
Addition, unsigned

`addu` R_d, R_s, R_t



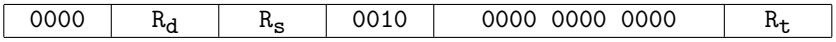
Put the sum of register R_s and register R_t into register R_d. Generate an overflow exception on unsigned overflow.

Addition, unsigned, immediate
`addui` R_d, R_s, `Immediate`



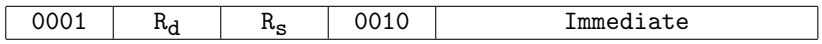
Put the sum of register R_s and the zero-extended immediate into register R_d. Generate an overflow exception on unsigned overflow.

Subtraction
`sub` R_d, R_s, R_t



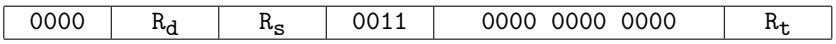
Put the difference of register R_s and register R_t into register R_d. Generate an overflow exception on signed overflow.

Subtraction, immediate
`subi` R_d, R_s, `Immediate`



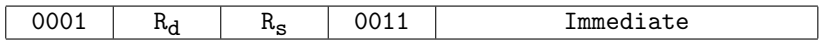
Put the difference of register R_s and the sign-extended immediate into register R_d. Generate an overflow exception on signed overflow.

Subtraction, unsigned
`subu` R_d, R_s, R_t



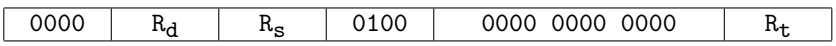
Put the difference of register R_s and register R_t into register R_d. Generate an overflow exception on unsigned overflow.

Subtraction, unsigned, immediate
`subui` R_d, R_s, `Immediate`



Put the difference of register R_s and the zero-extended immediate into register R_d. Generate an overflow exception on unsigned overflow.

Multiplication
`mult` R_d, R_s, R_t



Put the product of the signed multiplication of register R_s and register R_t into register R_d. Generate an overflow exception on signed overflow.

Multiplication, immediate
`multi` R_d, R_s, `Immediate`

0001	R_d	R_s	0100	Immediate
4	4	4	4	16

Put the product of the signed multiplication of register R_s and the sign-extended immediate into register R_d. Generate an overflow exception on signed overflow.

Multiplication, unsigned
`multu` R_d, R_s, R_t

0000	R_d	R_s	0101	0000 0000 0000	R_t
4	4	4	4	12	4

Put the product of the unsigned multiplication of register R_s and register R_t into register R_d. Generate an overflow exception on unsigned overflow.

Multiplication, unsigned, immediate
`multui` R_d, R_s, `Immediate`

0001	R_d	R_s	0101	Immediate
4	4	4	4	16

Put the product of the unsigned multiplication of register R_s and the zero-extended immediate into register R_d. Generate an overflow exception on unsigned overflow.

Division
`div` R_d, R_s, R_t

0000	R_d	R_s	0110	0000 0000 0000	R_t
4	4	4	4	12	4

Put the result of the signed integer division of register R_s by register R_t into register R_d. Generate a divide-by-zero exception if the contents of R_t is zero.

Division, immediate
`divi` R_d, R_s, `Immediate`

0001	R_d	R_s	0110	Immediate
4	4	4	4	16

Put the result of the signed integer division of register R_s by the sign-extended immediate into register R_d. Generate a divide-by-zero exception if the immediate value is zero.

Division, unsigned
`divu` R_d, R_s, R_t

0000	R_d	R_s	0111	0000 0000 0000	R_t
4	4	4	4	12	4

Put the result of the unsigned division of register R_s by register R_t into register R_d. Generate a divide-by-zero exception if the contents of R_t is zero.

Division, unsigned, immediate
`divui` R_d, R_s, `Immediate`

0001	R_d	R_s	0111	Immediate
4	4	4	4	16

Put the result of the unsigned division of register R_s by the zero-extended immediate into register R_d. Generate a divide-by-zero exception if the immediate value is zero.

Remainder
`rem` R_d, R_s, R_t

0000	R_d	R_s	1000	0000 0000 0000	R_t
4	4	4	4	12	4

Put the remainder of the signed division of register R_s by register R_t into register R_d. Generate a divide-by-zero exception if the contents of R_t is zero.

Remainder, immediate
`remi` R_d, R_s, `Immediate`

0001	R_d	R_s	1000	Immediate
4	4	4	4	16

Put the remainder of the signed division of register R_s by the sign-extended immediate into register R_d. Generate a divide-by-zero exception if the immediate value is zero.

Remainder, unsigned
`remu` R_d, R_s, R_t

0000	R_d	R_s	1001	0000 0000 0000	R_t
4	4	4	4	12	4

Put the remainder of the unsigned division of register R_s by the register R_t into register R_d. Generate a divide-by-zero exception if the contents of R_t is zero.

Remainder, unsigned, immediate
`remui` R_d, R_s, `Immediate`

0001	R_d	R_s	1001	Immediate
4	4	4	4	16

Put the remainder of the unsigned division of register R_s by the zero-extended immediate into register R_d. Generate a divide-by-zero exception if the immediate value is zero.

Load high immediate
`lhi` R_d, `Immediate`

0011	R_d	0000	1110	Immediate
4	4	4	4	16

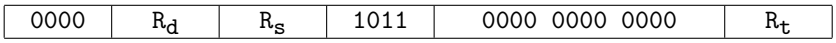
Put the 16 bit immediate into the upper 16 bits of register R_d, and set the lower 16 bits to zero.

Load address
`la` R_d, `Address`

Put the zero-extended 20 bit address into register R_d.

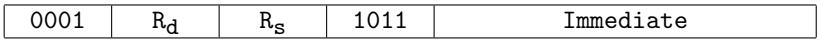
Bitwise instructions

And
`and` R_d, R_s, R_t



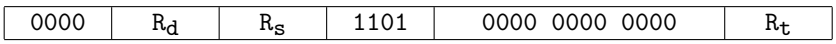
Put the result of the logical AND of registers R_s and R_t into register R_d.

And, immediate
`andi` R_d, R_s, `Immediate`



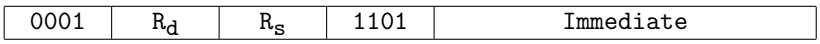
Put the result of the logical AND of register R_s and the zero-extended immediate into register R_d.

Or
`or` R_d, R_s, R_t



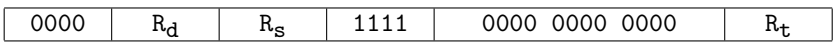
Put the result of the logical OR of registers R_s and R_t into register R_d.

Or, immediate
`ori` R_d, R_s, `Immediate`



Put the result of the logical OR of register R_s and the zero-extended immediate into register R_d.

Xor
`xor` R_d, R_s, R_t



Put the result of the logical exclusive-OR of registers R_s and R_t into register R_d.

Xor, immediate
`xori` R_d, R_s, Immediate

0001	R_d	R_s	1111	Immediate
4	4	4	4	16

Put the result of the logical exclusive-OR of register R_s and the zero-extended immediate into register R_d.

Shift left logical
`sll` R_d, R_s, R_t

0000	R_d	R_s	1010	0000 0000 0000	R_t
4	4	4	4	12	4

Shift the value in register R_s left by the unsigned value given by the least significant 5 bits of register R_t, and put the result in register R_d, inserting zeros into the low order bits.

Shift left logical, immediate
`slli` R_d, R_s, Immediate

0001	R_d	R_s	1010	Immediate
4	4	4	4	16

Shift the value in register R_s left by the unsigned value given by the least significant 5 bits of the immediate, and put the result in register R_d, inserting zeros into the low order bits.

Shift right logical
`srl` R_d, R_s, R_t

0000	R_d	R_s	1100	0000 0000 0000	R_t
4	4	4	4	12	4

Shift the value in register R_s right by the unsigned value given by the least significant 5 bits of register R_t, and place the result in register R_d, inserting zeros into the high order bits.

Shift right logical, immediate
`srli` R_d, R_s, Immediate

0001	R_d	R_s	1100	Immediate
4	4	4	4	16

Shift the value in register R_s right by the unsigned value given by the least significant 5 bits of the immediate, and place the result in register R_d, inserting zeros into the high order bits.

Shift right arithmetic
`sra` R_d, R_s, R_t

0000	R_d	R_s	1110	0000 0000 0000	R_t
4	4	4	4	12	4

Shift the value in register R_s right by the unsigned value given by the least significant 5 bits of register R_t, and place the result in register R_d, sign-extending the high order bits.

Shift right arithmetic, immediate
`srai` R_d, R_s, `Immediate`

0001	R_d	R_s	1110	Immediate
4	4	4	4	16

Shift the value in register R_s right by the unsigned value given by the least significant 5 bits of the immediate, and place the result in register R_d, sign-extending the high order bits.

Test instructions

Set on less than
`slt` R_d, R_s, R_t

0010	R_d	R_s	0000	0000 0000 0000	R_t
4	4	4	4	12	4

Set register R_d to 1 if register R_s is less than register R_t according to a signed comparison, and 0 otherwise.

Set on less than immediate
`slti` R_d, R_s, `Immediate`

0011	R_d	R_s	0000	Immediate
4	4	4	4	16

Set register R_d to 1 if register R_s is less than the sign-extended immediate according to a signed comparison, and 0 otherwise.

Set on less than, unsigned
`sltu` R_d, R_s, R_t

0010	R_d	R_s	0001	0000 0000 0000	R_t
4	4	4	4	12	4

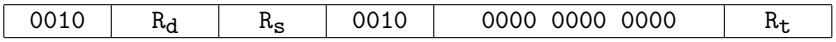
Set register R_d to 1 if register R_s is less than register R_t according to an unsigned comparison, and 0 otherwise.

Set on less than, unsigned, immediate
`sltui` R_d, R_s, `Immediate`

0011	R_d	R_s	0001	Immediate
4	4	4	4	16

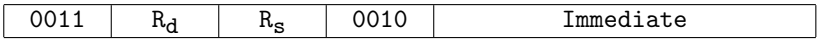
Set register R_d to 1 if register R_s is less than the zero-extended immediate according to an unsigned comparison, and 0 otherwise.

Set on greater than
`sgt` R_d, R_s, R_t



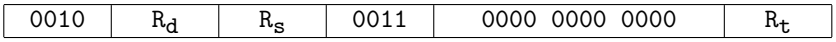
Set register R_d to 1 if register R_s is greater than register R_t according to a signed comparison, and 0 otherwise.

Set on greater than, immediate
`sgti` R_d, R_s, `Immediate`



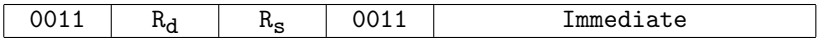
Set register R_d to 1 if register R_s is greater than the sign-extended immediate according to a signed comparison, and 0 otherwise.

Set on greater than, unsigned
`sgtu` R_d, R_s, R_t



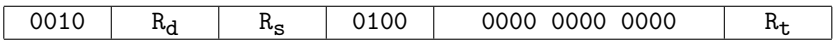
Set register R_d to 1 if register R_s is greater than register R_t according to an unsigned comparison, and 0 otherwise.

Set on greater than, unsigned, immediate
`sgtui` R_d, R_s, `Immediate`



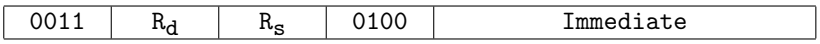
Set register R_d to 1 if register R_s is greater than the zero-extended immediate according to an unsigned comparison, and 0 otherwise.

Set on less than or equal to
`sle` R_d, R_s, R_t



Set register R_d to 1 if register R_s is less than or equal to register R_t according to a signed comparison, and 0 otherwise.

Set on less than or equal to, immediate
`slei` R_d, R_s, `Immediate`



Set register R_d to 1 if register R_s is less than or equal to the sign-extended immediate according to a signed comparison, and 0 otherwise.

Set on less than or equal to, unsigned
`sleu` R_d, R_s, R_t

0010	R_d	R_s	0101	0000 0000 0000	R_t
4	4	4	4	12	4

Set register R_d to 1 if register R_s is less than or equal to register R_t according to an unsigned comparison, and 0 otherwise.

Set on less than or equal to, unsigned, immediate
`sleui` R_d, R_s, Immediate

0011	R_d	R_s	0101	Immediate
4	4	4	4	16

Set register R_d to 1 if register R_s is less than or equal to the zero-extended immediate according to an unsigned comparison, and 0 otherwise.

Set on greater than or equal to
`sge` R_d, R_s, R_t

0010	R_d	R_s	0110	0000 0000 0000	R_t
4	4	4	4	12	4

Set register R_d to 1 if register R_s is greater than or equal to register R_t according to a signed comparison, and 0 otherwise.

Set on greater than or equal to immediate
`sgei` R_d, R_s, Immediate

0011	R_d	R_s	0110	Immediate
4	4	4	4	16

Set register R_d to 1 if register R_s is greater than or equal to the sign-extended immediate according to a signed comparison, and 0 otherwise.

Set on greater than or equal to, unsigned
`sgeu` R_d, R_s, R_t

0010	R_d	R_s	0111	0000 0000 0000	R_t
4	4	4	4	12	4

Set register R_d to 1 if register R_s is greater than or equal to register R_t according to an unsigned comparison, and 0 otherwise.

Set on greater than or equal to, unsigned, immediate
`sgeui` R_d, R_s, Immediate

0011	R_d	R_s	0111	Immediate
4	4	4	4	16

Set register R_d to 1 if register R_s is greater than or equal to the zero-extended immediate according to an unsigned comparison, and 0 otherwise.

Set on equal to
`seq` R_d, R_s, R_t

0010	R_d	R_s	1000	0000 0000 0000	R_t
4	4	4	4	12	4

Set register R_d to 1 if register R_s is equal to register R_t according to a signed comparison, and 0 otherwise.

Set on equal to immediate
`seqi` R_d, R_s, Immediate

0011	R_d	R_s	1000	Immediate
4	4	4	4	16

Set register R_d to 1 if register R_s is equal to the sign-extended immediate according to a signed comparison, and 0 otherwise.

Set on equal to, unsigned
`sequ` R_d, R_s, R_t

0010	R_d	R_s	1001	0000 0000 0000	R_t
4	4	4	4	12	4

Set register R_d to 1 if register R_s is equal to register R_t according to an unsigned comparison, and 0 otherwise.

Set on equal to, unsigned, immediate
`sequi` R_d, R_s, Immediate

0011	R_d	R_s	1001	Immediate
4	4	4	4	16

Set register R_d to 1 if register R_s is equal to the zero-extended immediate according to an unsigned comparison, and 0 otherwise.

Set on not equal to
`sne` R_d, R_s, R_t

0010	R_d	R_s	1010	0000 0000 0000	R_t
4	4	4	4	12	4

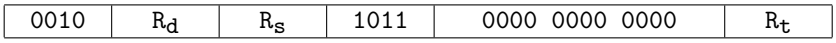
Set register R_d to 1 if register R_s is not equal to register R_t according to a signed comparison, and 0 otherwise.

Set on not equal to immediate
`snei` R_d, R_s, Immediate

0011	R_d	R_s	1010	Immediate
4	4	4	4	16

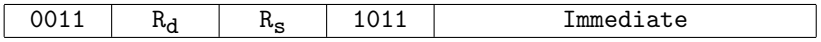
Set register R_d to 1 if register R_s is not equal to the sign-extended immediate according to a signed comparison, and 0 otherwise.

Set on not equal to, unsigned
`sneu` R_d, R_s, R_t



Set register R_d to 1 if register R_s is not equal to register R_t according to an unsigned comparison, and 0 otherwise.

Set on not equal to, unsigned, immediate
`sneui` R_d, R_s, `Immediate`



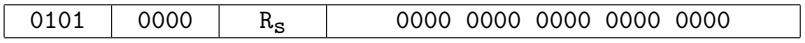
Set register R_d to 1 if register R_s is not equal to the zero-extended immediate according to an unsigned comparison, and 0 otherwise.

Branch instructions

Jump
`j Address`

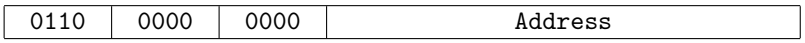
Unconditionally jump to the instruction whose address is given by the address field.

Jump to register
`jr` R_s



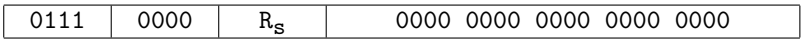
Unconditionally jump to the instruction whose address is in the least significant 20 bits of register R_s.

Jump and link
`jal Address`



Unconditionally jump to the instruction whose address is given by the address field. Save the address of the next instruction in register `$ra`.

Jump and link register
`jalr` R_s

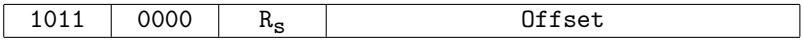


Unconditionally jump to the instruction whose address is in the least significant 20 bits of register R_s. Save the address of the next instruction in register `$ra`.

Branch on equal to zero
`beqz` R_s`, Offset`

Conditionally branch the number of instructions specified by the sign-extended offset if register R_s is equal to 0.

Branch on not equal to zero
`bnez` R_s`, Offset`



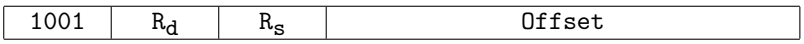
Conditionally branch the number of instructions specified by the sign-extended offset if register R_s is not equal to 0.

Memory instructions

Load word
`lw` R_d`, Offset(`R_s`)`

Add the contents of register R_s and the sign-extended offset to give an effective address. Load the contents of the memory location specified by this effective address into register R_d.

Store word
`sw` R_d`, Offset(`R_s`)`

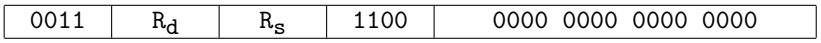


Add the contents of register R_s and the sign-extended offset to give an effective address. Store the contents of register R_d into the memory location specified by this effective address.

Special instructions

Move general register to special register

`movgs` R_d, R_s

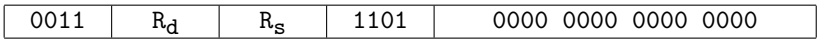


0011	R_d	R_s	1100	0000 0000 0000 0000
4	4	4	4	16

Copy the contents of general purpose register R_s into special purpose register R_d.

Move special register to general register

`movsg` R_d, R_s

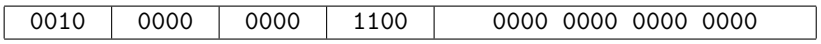


0011	R_d	R_s	1101	0000 0000 0000 0000
4	4	4	4	16

Copy the contents of special purpose register R_s into general purpose register R_d.

Break

`break`

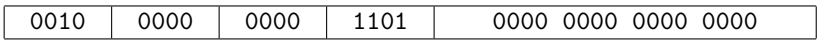


0010	0000	0000	1100	0000 0000 0000 0000
4	4	4	4	16

Generate a breakpoint exception, transferring control to the exception handler.

System call

`syscall`

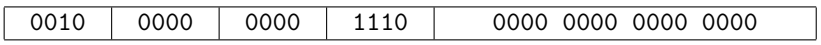


0010	0000	0000	1101	0000 0000 0000 0000
4	4	4	4	16

Generate a system call exception, transferring control to the exception handler.

Return from exception

`rfe`



0010	0000	0000	1110	0000 0000 0000 0000
4	4	4	4	16

Restore the saved interrupt enable and kernel/user mode bits and jump to the instruction at the address specified in the special register `$ear`.

Instruction Set Reference Table

Assembler	Machine code	Function	Description
`add R_d, R_s, R_t`	`0000 dddd ssss 0000 0000 0000 0000 tttt`	$R_d \leftarrow R_s + R_t$	Add
`addi R_d, R_s, immed`	`0001 dddd ssss 0000 iiii iiii iiii iiii`	$R_d \leftarrow R_s + \int(immed)$	Add Immediate
`addu R_d, R_s, R_t`	`0000 dddd ssss 0001 0000 0000 0000 tttt`	$R_d \leftarrow R_s + R_t$	Add Unsigned
`addui R_d, R_s, immed`	`0001 dddd ssss 0001 iiii iiii iiii iiii`	$R_d \leftarrow R_s + immed$	Add Unsigned Immediate
`sub R_d, R_s, R_t`	`0000 dddd ssss 0010 0000 0000 0000 tttt`	$R_d \leftarrow R_s - R_t$	Subtract
`subi R_d, R_s, immed`	`0001 dddd ssss 0010 iiii iiii iiii iiii`	$R_d \leftarrow R_s - \int(immed)$	Subtract Immediate
`subu R_d, R_s, R_t`	`0000 dddd ssss 0011 0000 0000 0000 tttt`	$R_d \leftarrow R_s - R_t$	Subtract Unsigned
`subui R_d, R_s, immed`	`0001 dddd ssss 0011 iiii iiii iiii iiii`	$R_d \leftarrow R_s - immed$	Subtract Unsigned Immediate
`mult R_d, R_s, R_t`	`0000 dddd ssss 0100 0000 0000 0000 tttt`	$R_d \leftarrow R_s \times R_t$	Multiply
`multi R_d, R_s, immed`	`0001 dddd ssss 0100 iiii iiii iiii iiii`	$R_d \leftarrow R_s \times \int(immed)$	Multiply Immediate
`multu R_d, R_s, R_t`	`0000 dddd ssss 0101 0000 0000 0000 tttt`	$R_d \leftarrow R_s \times R_t$	Multiply Unsigned
`multui R_d, R_s, immed`	`0001 dddd ssss 0101 iiii iiii iiii iiii`	$R_d \leftarrow R_s \times immed$	Multiply Unsigned Immediate
`div R_d, R_s, R_t`	`0000 dddd ssss 0110 0000 0000 0000 tttt`	$R_d \leftarrow R_s \div R_t$	Divide
`divi R_d, R_s, immed`	`0001 dddd ssss 0110 iiii iiii iiii iiii`	$R_d \leftarrow R_s \div \int(immed)$	Divide Immediate
`divu R_d, R_s, R_t`	`0000 dddd ssss 0111 0000 0000 0000 tttt`	$R_d \leftarrow R_s \div R_t$	Divide Unsigned
`divui R_d, R_s, immed`	`0001 dddd ssss 0111 iiii iiii iiii iiii`	$R_d \leftarrow R_s \div immed$	Divide Unsigned Immediate
`rem R_d, R_s, R_t`	`0000 dddd ssss 1000 0000 0000 0000 tttt`	$R_d \leftarrow R_s \ \% \ R_t$	Remainder
`remi R_d, R_s, immed`	`0001 dddd ssss 1000 iiii iiii iiii iiii`	$R_d \leftarrow R_s \ \% \ \int(immed)$	Remainder Immediate
`remu R_d, R_s, R_t`	`0000 dddd ssss 1001 0000 0000 0000 tttt`	$R_d \leftarrow R_s \ \% \ R_t$	Remainder Unsigned
`remui R_d, R_s, immed`	`0001 dddd ssss 1001 iiii iiii iiii iiii`	$R_d \leftarrow R_s \ \% \ immed$	Remainder Unsigned Immediate
`lhi R_d, immed`	`0011 dddd ssss 1110 iiii iiii iiii iiii`	$R_d \leftarrow immed \ll 16$	Load High Immediate
`la R_d, address`	`1100 dddd 0000 aaaa aaaa aaaa aaaa aaaa`	$R_d \leftarrow address$	Load Address

Table 1: Arithmetic Instructions

`and R_d, R_s, R_t`	`0000 dddd ssss 1011 0000 0000 0000 tttt`	$R_d \leftarrow R_s \ AND \ R_t$	Bitwise AND
`andi R_d, R_s, immed`	`0001 dddd ssss 1011 iiii iiii iiii iiii`	$R_d \leftarrow R_s \ AND \ immed$	Bitwise AND Immediate
`or R_d, R_s, R_t`	`0000 dddd ssss 1101 0000 0000 0000 tttt`	$R_d \leftarrow R_s \ OR \ R_t$	Bitwise OR
`ori R_d, R_s, immed`	`0001 dddd ssss 1101 iiii iiii iiii iiii`	$R_d \leftarrow R_s \ OR \ immed$	Bitwise OR Immediate
`xor R_d, R_s, R_t`	`0000 dddd ssss 1111 0000 0000 0000 tttt`	$R_d \leftarrow R_s \ XOR \ R_t$	Bitwise XOR
`xori R_d, R_s, immed`	`0001 dddd ssss 1111 iiii iiii iiii iiii`	$R_d \leftarrow R_s \ XOR \ immed$	Bitwise XOR Immediate
`sll R_d, R_s, R_t`	`0000 dddd ssss 1010 0000 0000 0000 tttt`	$R_d \leftarrow R_s \ll R_t$	Shift Left Logical
`slli R_d, R_s, immed`	`0001 dddd ssss 1010 iiii iiii iiii iiii`	$R_d \leftarrow R_s \ll immed$	Shift Left Logical Immediate
`srl R_d, R_s, R_t`	`0000 dddd ssss 1100 0000 0000 0000 tttt`	$R_d \leftarrow R_s \gg R_t$	Shift Right Logical
`srli R_d, R_s, immed`	`0001 dddd ssss 1100 iiii iiii iiii iiii`	$R_d \leftarrow R_s \gg immed$	Shift Right Logical Immediate
`sra R_d, R_s, R_t`	`0000 dddd ssss 1110 0000 0000 0000 tttt`	$R_d \leftarrow \int(R_s \gg R_t)$	Shift Right Arithmetic
`srai R_d, R_s, immed`	`0001 dddd ssss 1110 iiii iiii iiii iiii`	$R_d \leftarrow \int(R_s \gg immed)$	Shift Right Arithmetic Immediate

Table 2: Bitwise Instructions

slt R_d, R_s, R_t	0010 dddd ssss 0000 0000 0000 0000 tttt	$R_d \leftarrow R_s < R_t$	Set on Less than
slti R_d, R_s, immed	0011 dddd ssss 0000 iiii iiii iiii iiii	$R_d \leftarrow R_s < \int(immed)$	Set on Less than Immediate
sltu R_d, R_s, R_t	0010 dddd ssss 0001 0000 0000 0000 tttt	$R_d \leftarrow R_s < R_t$	Set on Less than Unsigned
sltui R_d, R_s, immed	0011 dddd ssss 0001 iiii iiii iiii iiii	$R_d \leftarrow R_s < immed$	Set on Less than Unsigned Immediate
sgt R_d, R_s, R_t	0010 dddd ssss 0010 0000 0000 0000 tttt	$R_d \leftarrow R_s > R_t$	Set on Greater than
sgti R_d, R_s, immed	0011 dddd ssss 0010 iiii iiii iiii iiii	$R_d \leftarrow R_s > \int(immed)$	Set on Greater than Immediate
sgtu R_d, R_s, R_t	0010 dddd ssss 0011 0000 0000 0000 tttt	$R_d \leftarrow R_s > R_t$	Set on Greater than Unsigned
sgtui R_d, R_s, immed	0011 dddd ssss 0011 iiii iiii iiii iiii	$R_d \leftarrow R_s > immed$	Set on Greater than Unsigned Immediate
sle R_d, R_s, R_t	0010 dddd ssss 0100 0000 0000 0000 tttt	$R_d \leftarrow R_s \leq R_t$	Set on Less than or Equal
slei R_d, R_s, immed	0011 dddd ssss 0100 iiii iiii iiii iiii	$R_d \leftarrow R_s \leq \int(immed)$	Set on Less or Equal Immediate
sleu R_d, R_s, R_t	0010 dddd ssss 0101 0000 0000 0000 tttt	$R_d \leftarrow R_s \leq R_t$	Set on Less or Equal Unsigned
sleui R_d, R_s, immed	0011 dddd ssss 0101 iiii iiii iiii iiii	$R_d \leftarrow R_s \leq immed$	Set on Less or Equal Unsigned Imm
sge R_d, R_s, R_t	0010 dddd ssss 0110 0000 0000 0000 tttt	$R_d \leftarrow R_s \geq R_t$	Set on Greater than or Equal
sgei R_d, R_s, immed	0011 dddd ssss 0110 iiii iiii iiii iiii	$R_d \leftarrow R_s \geq \int(immed)$	Set on Greater or Equal Immediate
sgeu R_d, R_s, R_t	0010 dddd ssss 0111 0000 0000 0000 tttt	$R_d \leftarrow R_s \geq R_t$	Set on Greater or Equal Unsigned
sgeui R_d, R_s, immed	0011 dddd ssss 0111 iiii iiii iiii iiii	$R_d \leftarrow R_s \geq immed$	Set on Greater or Equal Unsigned Imm
seq R_d, R_s, R_t	0010 dddd ssss 1000 0000 0000 0000 tttt	$R_d \leftarrow R_s = R_t$	Set on Equal
seqi R_d, R_s, immed	0011 dddd ssss 1000 iiii iiii iiii iiii	$R_d \leftarrow R_s = \int(immed)$	Set on Equal Immediate
sequ R_d, R_s, R_t	0010 dddd ssss 1001 0000 0000 0000 tttt	$R_d \leftarrow R_s = R_t$	Set on Equal Unsigned
sequi R_d, R_s, immed	0011 dddd ssss 1001 iiii iiii iiii iiii	$R_d \leftarrow R_s = immed$	Set on Equal Unsigned Immediate
sne R_d, R_s, R_t	0010 dddd ssss 1010 0000 0000 0000 tttt	$R_d \leftarrow R_s \neq R_t$	Set on Not Equal
snei R_d, R_s, immed	0011 dddd ssss 1010 iiii iiii iiii iiii	$R_d \leftarrow R_s \neq \int(immed)$	Set on Not Equal Immediate
sneu R_d, R_s, R_t	0010 dddd ssss 1011 0000 0000 0000 tttt	$R_d \leftarrow R_s \neq R_t$	Set on Not Equal Unsigned
sneui R_d, R_s, immed	0011 dddd ssss 1011 iiii iiii iiii iiii	$R_d \leftarrow R_s \neq immed$	Set on Not Equal Unsigned Immediate

Table 3: Test Instructions

Branch Instructions			
j address	0100 0000 0000 aaaa aaaa aaaa aaaa aaaa	$PC \leftarrow Address$	Jump
jr R_s	0101 0000 ssss 0000 0000 0000 0000 0000	$PC \leftarrow R_s$	Jump to Register
jal address	0110 0000 0000 aaaa aaaa aaaa aaaa aaaa	$\$ra \leftarrow PC$, $PC \leftarrow Address$	Jump and Link
jalr R_s	0111 0000 ssss 0000 0000 0000 0000 0000	$\$ra \leftarrow PC$, $PC \leftarrow R_s$	Jump and Link Register
beqz R_s, offset	1010 0000 ssss oooo oooo oooo oooo oooo	$if(R_s = 0)\ PC \leftarrow PC + offset$	Branch on equal to 0
bnez R_s, offset	1011 0000 ssss oooo oooo oooo oooo oooo	$if(R_s \neq 0)\ PC \leftarrow PC + offset$	Branch on not equal to 0
Memory Instructions			
lw R_d, offset(R_s)	1000 dddd ssss oooo oooo oooo oooo oooo	$R_d \leftarrow MEM[R_s + offset]$	Load word
sw R_d, offset(R_s)	1001 dddd ssss oooo oooo oooo oooo oooo	$MEM[R_s + offset] \leftarrow R_d$	Store word
Special Instructions			
movgs R_d, R_s	0011 0000 0000 1100 0000 0000 0000 0000	$R_d \leftarrow R_s$	Move General to Special Register
movsg R_d, R_s	0011 0000 0000 1101 0000 0000 0000 0000	$R_d \leftarrow R_s$	Move Special to General Register
break	0010 0000 0000 1100 0000 0000 0000 0000		Generate Break Point Exception
syscall	0010 0000 0000 1101 0000 0000 0000 0000		Generate Syscall Exception
rfe	0010 0000 0000 1110 0000 0000 0000 0000	$PC \leftarrow \$ear$	Return from Exception

Table 4: Other Instructions