

**MIPS ISA**

Written by:

Jonathan Shihata

Chanartip Soonthornwan

TABLE OF CONTENTS

INTRODUCTION

PURPOSE 5

INSTRUCTION SET ARCHITECTURE 5

* R-TYPE INSTRUCTION 5
* I-TYPE INSTRUCTION 5
* J-TYPE INSTRUCTION 5
* ENHANCED INSTRUCTION 5

EXAMPLE 5

R-TYPE INSTRUCTION

shift left logic (SLL) 6

shift right logic (SRL) 7

shift right arithmetic (SRA) 8

jump register (JR) 9

move from hi (MFHI) 10

move from lo (MFLO) 11

multiplication (MULT) 12

division (DIV) 13

addition signed (ADD) 14

addition unsigned (ADDU) 15

subtraction signed (SUB) 16

subtraction unsigned (SUBU) 17

and (AND) 18

or (OR) 19

exclusive or (XOR) 20

not or (NOR) 21

set less than (SLT) 22

set less than unsigned (SLTU) 23

breakpoint (BREAK) 24

set interrupt (SETIE) 25

I-TYPE INSTRUCTION

branch equal (BEQ) 26

branch not equal (BNE) 27

branch less than or equal to zero (BLEZ) #

branch greater than zero (BGTZ) #

addition immediate (ADDI) 28

set less than immediate (SLTI) 29

set less than immediate unsigned (SLTIU) 30

and immediate (ANDI) 31

or immediate (ORI) 32

exclusive or immediate (XORI) 33

load upper immediate (LUI) 34

load word (LW) 35

store word (SW) 36

J-TYPE INSTRUCTION

jump (J) 37

jump and link (JAL) 38

ENHANCED INSTRUCTION

INPUT #

OUTPUT #

RETI #

“E\_KEY” #

CONCLUSION

**this page is intentionally left blank**

introduction

purpose

instruction set architecture

r-type instruction

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  000000 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

i-type instruction

**31 26 25 21 20 16 15 0**

|  |  |  |  |
| --- | --- | --- | --- |
| beq  000100 | rs | rt | offset |

**6 bits 5 bits 5 bits 16 bits**

j-type instruction

**31 26 25 0**

|  |  |
| --- | --- |
| jal  000011 | offset |

**6 bits 24 bits**

example

shift left logic (sll)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  000000 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: sll rd, rt, shamt

purpose: shift left contents of rt by shamt and store it in rd

description: R[rd]= R[rt] << shamt

the destination register(rd) gets the contents of the source(rt) register shifted logically left by the shamt field. zeros are shifted in for the shamt.

restrictions:

the shift amount is limited by what can be specified using only 5 bits or a value of 32.

operation:

PC ← PC +4; IR ← M[PC];

RS ← reg[IR[25-21]]; RT ← reg[IR[20-16]];

ALU\_OUT <- RT << shamt;

reg[IR[15-11]] ← ALU\_OUT(RT << shamt);

exceptions:

n/a

**programming notes:**

n/a

**example:**

shift right logic (srl)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  000010 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: srl rd, rt, shamt

purpose: shift right the contents of rt by shamt and store it in rd

description: R[rd]= R[rt] >> shamt

the destination register(rd) gets the contents of the source(rt) register shifted logically right by the shamt field. zeros are shifted in for the shamt.

restrictions:

the shift amount is limited by what can be specified using only 5 bits or a value of 32.

operation:

PC ← PC +4; IR <- M[PC];

RS ← reg[IR[25-21]]; RT ← reg[IR[20-16]];

ALU\_OUT <- RT >> shamt;

reg[IR[15-11]] ← ALU\_OUT(RT >> shamt);

exceptions:

n/a

**programming notes:**

n/a

**Example:**

shift right arithmetic (sra)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  000011 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: sra rd, rt, shamt

purpose: shift right the contents of rt by shamt and store it in rd

description: R[rd]= R[rt] >>> shamt

the destination register(rd) gets the contents of the source(rt) register shifted arithmetically right by the shamt field. sign bit is shifted in.

restrictions:

the shift amount is limited by what can be specified using only 5 bits or a value of 32.

operation:

PC ← PC +4; IR ← M[PC];

RS ← reg[IR[25-21]]; RT ← reg[IR[20-16]];

ALU\_OUT ← RT >>> shamt;

reg[IR[15-11]] <- ALU\_OUT(RT >>> shamt);

exceptions:

n/a

**programming notes:**

n/a

**Example:**

jump register (jr)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  001000 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: jr rs

purpose: jump to the address stored in rs.

description: pc ← M[rs]

jumps to the effective address sprcified by $rs.

restrictions:

if either of the two least significant bits are not zero, then an error occurs and will violate the branch address.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS;

REG[IR[15-11]] ← ALU\_OUT(RT >>> SHAMT);

exceptions:

n/a

**programming notes:**

n/a

**Example:**

move from hi (mfhi)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  010000 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: mfhi rd

purpose: to copy the special hi register to the general purpose register rd.

description: R[rd] ← hi

the contents of special register hi is loaded into rd.

restrictions:

mfhi will not allow a multiply or divide instruction within two instructions after it has been executed because it violates how the mips pipeline works.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

REG[IR[15-11]] ← ALU\_OUT[31-16];

exceptions:

n/a

**programming notes:**

n/a

**Example:**

move from lo (mflo)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  010010 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: mflo rd

purpose to copy the special lo register to the general purpose register rd.

description: R[rd] ← lo

the contents of special register lo is loaded into rd.

restrictions:

mflo will not allow a multiply or divide instruction within two instructions after it has been executed because it violates how the mips pipeline works.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

REG[IR[15-11]] ← ALU\_OUT[15-16];

exceptions:

n/a

**programming notes:**

n/a

**Example:**

multiply (mult)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  011000 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: mult rs, rt

purpose to multiply 32 bit signed integers

description: {hi, lo} ← R[rs] \* R[rt]

the 32-bit value in $rt are multiplied by the 32-bit value in $rs, and storing the the 64-bit result in {hi,lo}

restrictions:

32-bit values in $rs and $rt must be signed.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT[15-16] ← RS \* RT

exceptions:

n/a

**programming notes:**

n/a

**Example:**

divide (div)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  011010 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: div rs, rt

purpose to divide 32 bit signed integers

description: lo ← R[rs] / R[rt], hi ← R[rs] % R[rt]

the 32-bit value in $rt are divided by the 32-bit value in $rs, and storing the the 64-bit result in {hi,lo}

restrictions:

32-bit values in $rs and $rt must be signed.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT[15-16] ← RS / RT

exceptions:

n/a

**programming notes:**

n/a

Example:

addition (add)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  100000 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: add rd, rs, rt

purpose to add 32 bit signed integers

description: R[rd] ← R[rs] + R[rt]

the 32-bit value in $rt is added to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

32-bit values in $rs and $rt must be signed.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS + RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS + RT);

exceptions:

n/a

**programming notes:**

n/a

**Example:**

addition unsigned (addu)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  100001 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: addu rd, rs, rt

purpose to add 32 bit signed integers

description: R[rd] ← R[rs] + R[rt]

the 32-bit value in $rt is added to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

32-bit values in $rs and $rt must be unsigned.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS + RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS + RT);

exceptions:

n/a

**programming notes:**

n/a

**Example:**

subtract (sub)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  100010 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: sub rd, rs, rt

purpose to subtract 32 bit signed integers

description: R[rd] ← R[rs] - R[rt]

the 32-bit value in $rt is sutracted to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

32-bit values in $rs and $rt must be signed.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS - RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS - RT);

exceptions:

n/a

**programming notes:**

n/a

**Example:**

subtract unsigned (subu)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  100011 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: subu rd, rs, rt

purpose to subtract 32 bit signed integers

description: R[rd] ← R[rs] - R[rt]

the 32-bit value in $rt is subtracted to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

32-bit values in $rs and $rt must be unsigned.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS - RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS - RT);

exceptions:

n/a

**programming notes:**

n/a

**Example:**

and

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  100100 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: and rd, rs, rt

purpose to do a bitwise and

description: R[rd] ← R[rs] & R[rt]

the 32-bit value in $rt is and’d bit by bit to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

n/a

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS & RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS & RT);

exceptions:

n/a

**programming notes:**

n/a

**Example:**

or

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  100101 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: or rd, rs, rt

purpose to do a bitwise or

description: R[rd] ← R[rs] | R[rt]

the 32-bit value in $rt is or’d bit by bit to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

n/a

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS | RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS | RT);

exceptions:

n/a

**programming notes:**

n/a

**Example:**

exclusive or (xor)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  100111 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: xor rd, rs, rt

purpose to do a bitwise exclusive or

description: R[rd] ← R[rs] ^ R[rt]

the 32-bit value in $rt is exclusive or’d bit by bit to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

n/a

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS ^ RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS ^ RT);

exceptions:

n/a

**programming notes:**

n/a

**Example:**

not or (nor)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  000000 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: nor rd, rs, rt

purpose to do a bitwise nor

description: R[rd] ← ~(R[rs] | R[rt])

the 32-bit value in $rt is nor’d bit by bit to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

n/a

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← ~(RS | RT)

REG[IR[15-11]]; RT ← ALU\_OUT(~(RS | RT));

exceptions:

n/a

**programming notes:**

n/a

**Example:**

set on less than (slt)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  101010 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: slt rd, rs, rt

purpose to compare for less than and return the boolean result as a 0 or 1

description: R[rd] ← (R[rs] < R[rt]) ? 1:0

the 32-bit value in $rt is compared to be less than the 32-bit value in $rs, and storing a 1 in $rd if its true or 0 if its false.

restrictions:

n/a

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← (RS < RT) ? 1:0;

REG[IR[15-11]]; RT ← ALU\_OUT((RS < RT) ? 1:0);

exceptions:

n/a

**programming notes:**

n/a

**Example:**

set on less than unsigned (sltu)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  101011 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: sltu rd, rs, rt

purpose to compare for an unsigned less than and return the boolean result as a 0 or 1.

description: R[rd] ← (R[rs] < R[rt]) ? 1:0

the 32-bit value in $rt is compared to be less than the 32-bit value in $rs, and storing a 1 in $rd if its true or 0 if its false.

restrictions:

The 32-bit values of register $rs and $rt must be unsinged.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← (RS < RT) ? 1:0;

REG[IR[15-11]]; RT ← ALU\_OUT((RS < RT) ? 1:0);

exceptions:

n/a

**programming notes:**

n/a

**Example:**

breakpoint (break)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  001101 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: break

purpose to stop in a program

description:

will display that the break instruction has been fetched with the time, then inform of a safe break and comlete a register dump task and a memory dump task.

restrictions:

n/a

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

NS ← BREAK;

exceptions:

n/a

**programming notes:**

n/a

**Example:**

set interrupt (setie)

**31 26 25 21 20 16 15 11 10 6 5 0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode  000000 | rs  00000 | rt  00000 | rd  00000 | shamt  00000 | funct  011111 |

**6 bits 5 bits 5 bits 5 bits 5 bits 6 bits**

format: setie rd, rs, rt

purpose to set the interrupt flag high

description:

the current interrupt flag will be updated to a value of 1’b1 and pass it to the next interrupt enable flag.

restrictions:

n/A

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

NS

exceptions:

n/a

**programming notes:**

n/a

**Example:**

branch on equal (beq)

**31 26 25 21 20 16 15 0**

|  |  |  |  |
| --- | --- | --- | --- |
| beq  000100 | rs | rt | offset |

**6 bits 5 bits 5 bits 16 bits**

format: beq rt, rs, offset

purpose: equal register-register compare and branch with 16-bit offest.

description: if(r[rs] == r[rt]) pc ← pc + 4 + branch\_addr

if the condition is true, the branch is taken and the follwed instruction will not be executed. if the branch is taken, 16-bit offset will be used to reform a pc-relative effective address by concatenated with 14-bit of offset[15], the offset, and two zeroes.

restrictions:

the branch may be followed by another branch or jump that would delay the execution

operation:

branch\_addr ← {14{offset[15]}, offset, {00}}

beq: cond ← r[rs] = r[rt]

if cond then

pc ← (pc+4+ branch\_addr)

end if

exceptions:

n/a

**programming notes:**

**example:**

branch on not equal (bne)

**31 26 25 21 20 16 15 0**

|  |  |  |  |
| --- | --- | --- | --- |
| bne  000101 | rs | rt | offset |

**6 bits 5 bits 5 bits 16 bits**

format: bne rt, rs, offset

purpose: not equal register-register compare and branch with 16-bit offest.

description: if(r[rs] != r[rt]) pc ← pc + 4 + branch\_addr

if the condition is true, the branch is taken and the follwed instruction will not be executed. if the branch is taken, 16-bit offset will be used to reform a pc-relative effective address by concatenated with 14-bit of offset[15], the offset, and two zeroes.

restrictions:

the branch may be followed by another branch or jump that would delay the execution

operation:

branch\_addr ← {14{offset[15]}, offset, {00} }

bne: cond ← r[rs] ≠ r[rt]

if cond then

pc ← (pc+4+ branch\_addr)

end if

exceptions:

n/a

**programming notes:**

**example:**

add immediate (addi)

**31 26 25 21 20 16 15 0**

|  |  |  |  |
| --- | --- | --- | --- |
| addi  001000 | rs | rt | immediate |

**6 bits 5 bits 5 bits 16 bits**

format: addi rt, rs, offset

purpose: add a constant with 16-bit offest.

description: r[rt] ← r[rs] + sign\_extend(immediate)

adding 32-bit value in r[rs] with 16-bit signed *offet* and the 32-bit logical result is placed into r[rt].

restrictions:

n/a

operation:

sign\_extend ← {16{immediate[15]}, immediate}

temp ← r[rs] + sign\_extend(immediate)

r[rt] ← temp

exceptions:

n/a

**programming notes:**

**example:**

set on less than immediate (slti)

**31 26 25 21 20 16 15 0**

|  |  |  |  |
| --- | --- | --- | --- |
| slti  001010 | rs | rt | immediate |

**6 bits 5 bits 5 bits 16 bits**

format: slti rt, rs, offset

purpose: record the result of a less-than comparison with a constant.

description: r[rt] ← r[rs] < sign\_extend(immediate)

compare the content of r[rs] and the 16-bit signed *immediate*, and then record a boolean result of the comparison in r[rt]. if r[rs] less than the immediate is true, return 1 (true). Or 0 (flase) otherwise.

restrictions:

n/a

operation:

sign\_extend ← {16{immediate[15]}, immediate}

if r[rs] < sign\_extend(immediate) then

r[rt] **←** zero || 1

else

r[rt] **←** zero

exceptions:

n/a

**programming notes:**

**example:**

set on less than immediate unsigned (sltiu)

**31 26 25 21 20 16 15 0**

|  |  |  |  |
| --- | --- | --- | --- |
| sltiu  001011 | rs | rt | immediate |

**6 bits 5 bits 5 bits 16 bits**

format: sltiu rt, rs, immediate

purpose: record the result of a less-than comparison with a constant.

description: r[rt] ← r[rs] < sign\_extend(immediate)

compare the content of r[rs] and the 16-bit signed *immediate* as unsigned integer, and then record a boolean result of the comparison in r[rt]. if r[rs] less than the immediate is true, return 1 (true). Or 0 (flase) otherwise.

restrictions:

n/a

operation:

if r[rs] < sign\_extend(immediate) then

r[rt] **←** zero || 1

else

r[rt] **←** zero

exceptions:

n/a

**programming notes:**

since sign-extend of immediate is 16 bit, 16th bit could extend the range of unsigned number from [0,32767] to [0, 65535]

**example:**

and immmediate (andi)

**31 26 25 21 20 16 15 0**

|  |  |  |  |
| --- | --- | --- | --- |
| andi  001100 | rs | rt | immediate |

**6 bits 5 bits 5 bits 16 bits**

format: andi rt, rs, immediate

**purpose:** perform a bitwise logic and with a constant.

description: r[rt] ← r[rs] & zero\_extend(immediate)

extend immediate with 16-bit zero, then bitwise logical and the content of r[rs], and then place the result into r[rt].

restrictions:

n/a

operation:

zero\_extend **←** {16{zero}, immediate}

r[rt] **←** r[rs] & zero\_extend

exceptions:

n/a

**programming notes:**

**example:**

or immediate (ori)

**31 26 25 21 20 16 15 0**

|  |  |  |  |
| --- | --- | --- | --- |
| ori  001101 | rs | rt | immediate |

**6 bits 5 bits 5 bits 16 bits**

format: ori rt, rs, immediate

**purpose:** perform a bitwise logic or with a constant.

description: r[rt] ← r[rs] | zero\_extend(immediate)

extend immediate with 16-bit zero, then bitwise logical or the content of r[rs], and then place the result into r[rt].

restrictions:

n/a

operation:

zero\_extend **←** {16{zero}, immediate}

r[rt] **←** r[rs] | zero\_extend

exceptions:

n/a

**programming notes:**

**example:**

exclusive or immediate (xori)

**31 26 25 21 20 16 15 0**

|  |  |  |  |
| --- | --- | --- | --- |
| xori  001110 | rs | rt | immediate |

**6 bits 5 bits 5 bits 16 bits**

format: xori rt, rs, immediate

**purpose:** perform a bitwise logic xor with a constant.

description: r[rt] ← r[rs] ^ zero\_extend(immediate)

extend immediate with 16-bit zero, then bitwise logical xor the content of r[rs], and then place the result into r[rt].

restrictions:

n/a

operation:

zero\_extend **←** {16{zero}, immediate}

r[rt] **←** r[rs] ^ zero\_extend

exceptions:

n/a

**programming notes:**

**example:**

load upper immediate (lui)

**31 26 25 21 20 16 15 0**

|  |  |  |  |
| --- | --- | --- | --- |
| lui  001111 | rt | 00000 | immediate |

**6 bits 5 bits 5 bits 16 bits**

format: lui rt, offset

**purpose:** load an upper half word to a register

description: r[rt] ← {immediate, 16{zero}}

extend immediate with 16-bit zero, then bitwise logical xor the content of r[rs], and then place the result into r[rt].

restrictions:

n/a

operation:

r[rt] **←** {immediate, 16{zero}}

exceptions:

n/a

**programming notes:**

assign rs as 5-bit zero at rs bits in the instruction.

**example:**

load word (lw)

**31 26 25 21 20 16 15 0**

|  |  |  |  |
| --- | --- | --- | --- |
| lw  100011 | base | rt | offset |

**6 bits 5 bits 5 bits 16 bits**

format: lw rt, offset(base)

**purpose:** load a word from memory.

description: r[rt] ← m[gpr[base] + offset]

load 32-bit word from a memory located at the fetched effective address (base+offset) and then place to r[rt].

restrictions:

effective address 2 least-significant bits have to be two zeroes, otherwise, address error exception will occurs.

operation:

sign\_extend ← {16{offset}, offset}

maddr **←** sign\_extend + gpr[base]

r[rt] **←** memory[maddr]

exceptions:

n/a

**programming notes:**

**example:**

store word (sw)

**31 26 25 21 20 16 15 0**

|  |  |  |  |
| --- | --- | --- | --- |
| sw  101011 | base | rt | offset |

**6 bits 5 bits 5 bits 16 bits**

format: lw rt, offset(base)

**purpose:** store a word from memory.

description: M[gpr[base]+offset] ← r[rt]

store 32-bit word to a memory located at the fetched effective address (base+offset) and then place to r[rt].

restrictions:

effective address 2 least-significant bits have to be two zeroes, otherwise, address error exception will occurs.

operation:

sign\_extend ← {16{offset}, offset}

maddr **←** sign\_extend + gpr[base]

memory[maddr] **←** r[rt]

exceptions:

n/a

**programming notes:**

**example:**

jump uncondition (j)

**31 26 25 0**

|  |  |
| --- | --- |
| j  000010 | offset |

**6 bits 24 bits**

format: j offset

**purpose:** unconditional branch (or jump)

description: pc ← jmp\_address

form a pc-related effective target address from concatenated 4 most-significant bit of pc+4, offset, and two zeroes, and then update the program counter (pc) with the new pc.

restrictions:

the branch may be followed by another branch or jump that would delay the execution

operation:

jmp\_address ← {pc+4 [31:28], offset, 2{zero}}

pc ← jmp\_address

exceptions:

n/a

**programming notes:**

**example:**

jump and link (jal)

**31 26 25 0**

|  |  |
| --- | --- |
| jal  000011 | offset |

**6 bits 24 bits**

format: jal offset

**purpose:** jump and link

description: gpr[31] ← pc+8; pc ← jmp\_address

store the return address in returning address register (gpr[31]). form a pc-related effective target address from concatenated 4 most-significant bit of pc+4, offset, and two zeroes, and then update the program counter (pc) with the new pc.

restrictions:

the branch may be followed by another branch or jump that would delay the execution

operation:

jmp\_address ← {pc+4 [31:28], offset, 2{zero}}

gpr[31] ← pc+8

pc ← jmp\_address

exceptions:

n/a

**programming notes:**

**example:**

**this page is intentionally left blank**