ISA Beta Release

Marc Slaughter, Nicholas Otto
University of San Diego
Comp 300, Digital Hardware and Design
Dr. Saturnino Garcia

March 8, 2013

Chapter 1

Architecture Description

1.1 Registers

The table of registers is as follows:

Name	Description	Notes
$g0-7	General Registers	Used for, suprise, general purposes
$sp	Stack Pointer	Points to top of register stack
$fp	Function Pointer	Stack to keep track of function returns
$ch0-32	Channel Registers	Special registers
$l0-32	Label Registers	Initialized at beginning of program

1.2 Function call ABI

Function calls are implemented using stacks. When making a function call

```
TRAP    3, label
```

the value of the program counter, PC, plus 13 is placed onto the function stack (indicated by $fp). $fp is then incremented to point to the next open stack opsition. PC is then given the value associated with the label. A call to return will set PC to the value popped off the function pointer stack.

If the user would like to pass values, they must first push those values onto the general value stack, whose entry point is indicated by $sp. The called function must then pop the values it needs off of the stack. With this method, the general registers can be overridden without risking overriding parameters.

1.3 Label initialization

Through some mechanism, for now we decided in the hardware, the hardware knows to read in the first 32 addressable locations in memory. The label registers are populated in order by these values. It is the assembler's job to ensure each the first 32 memory locations are the correct values.

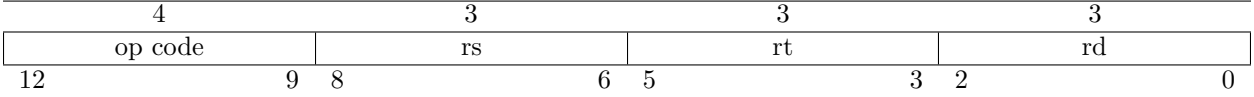
1.4 Instructions

The instructions have been separated into two types, "R" or "regular" type instructions and "S" or "special" type instructions. The distinction is analogous to, but not an exact replication of, the MIPS R and I type instructions.

1.4.1 R Type Instructions

Instruction Name

GENERAL DESCRIPTION

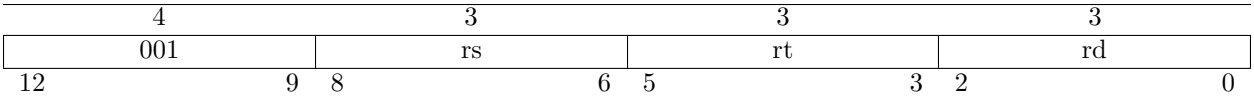


Format
The instruction calling format

Description
The instruction description

mul

MULTIPLICATION



Format

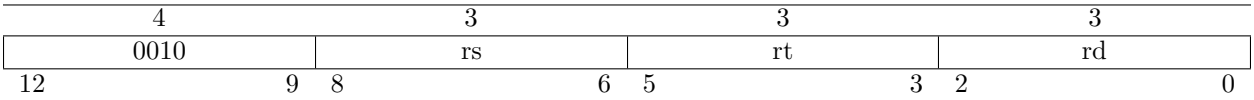
```
mul rd, rs, rt
```

RTL

```
R[rd] = lower34(R[rs] * R[rt])
```

Description
Multiplies the general registers `rs` and `rt` and sets the lower 34 bits of the result to `rd`.

add ADDITION



Format

```
add rd, rs, rt
```

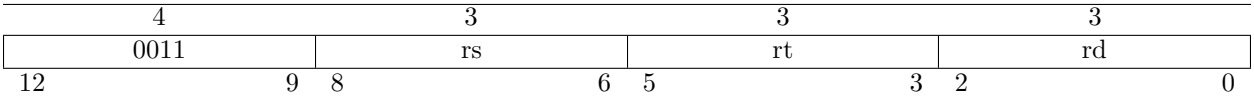
RTL

```
R[rd] = R[rs] + R[rt]
```

Description

Adds the contents of general registers `rs` and `rt` and store the result in general register `rd`.

sub SUBTRACTION



Format

```
sub rd, rs, rt
```

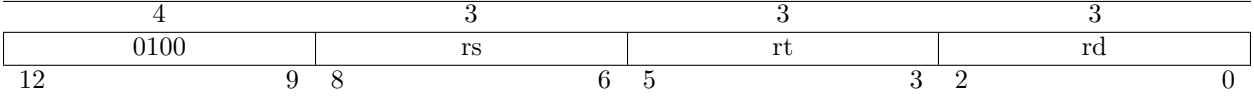
RTL

```
R[rd] = R[rs] - R[rt]
```

Description

Subtracts the contents of general registers `rs` and `rt` and store the result in general register `rd`.

or LOGICAL OR



Format

```
or rd, rs, rt
```

RTL

```
R[rd] = R[rs] | R[rt]
```

Description

Sets general register `rd` to the bitwise logical or of general registers `rs` and `rt`.

nor LOGICAL NOR

4	3	3	3
0101	rs	rt	rd
12 9	8 6	5 3	2 0

Format

`nor rd, rs, rt`

RTL

`R[rd] = ¬(R[rs] | R[rt])`

Description

Flips the bits of general register `rd` after a normal `or` operation.

sr SHIFT RIGHT BITWISE

4	3	3	3
0110	rs	rt	rd
12 9	8 6	5 3	2 0

Format

`sr rd, rs, rt`

RTL

`R[rd] = R[rs] >> shamt(R[rt]))`

Description

Bitwise shifts the contents of general register `rs` by the amount in `rt` and puts the result in general register `rd`.

lw LOAD WORD (34 BITS)

4	3	3	3
0111	rs	rt	rd
12 9	8 6	5 3	2 0

Format

`lw rd, rs, rt`

RTL

`R[rd] = M[R[rs] + R[rt]]`

Description

Set general register `rd` to the value stored at memory location `rs + rt`.

SW STORE WORD (34 BITS)

4	3	3	3
1000	rs	rt	rd
12 9	8 6	5 3	2 0

Format

`sw rd, rs, rt`

RTL

`M[R[rs] + R[rt]] = R[rd]`

Description

Set value at memory location `rs + rt` to the value of `rd`.

bne BRANCH IF NOT EQUAL

4	3	3	3
1001	rs	rt	label
12 9	8 6	5 3	2 0

Format

`bne rs, rt, label`

RTL

`if(R[rs] != R[rt]):  PC = PC + InstructionLength + R[label]`

Description

Checks to see if general registers `rs` and `rt` are not equal. If check returns true, the program counter is incremented an additional `R[label]` amount (which is taken from the label registers).

beq BRANCH IF EQUAL

4	3	3	3
1010	rs	rt	label
12 9	8 6	5 3	2 0

Format

`beq rs, rt, label`

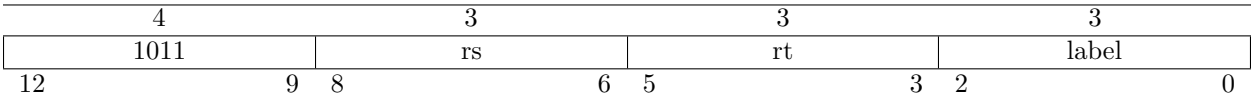
RTL

`if(R[rs] == R[rt]):  PC = PC + InstructionLength + R[label]`

Description

Checks to see if general registers `rs` and `rt` are equal. If check returns true, the program counter is incremented an additional `R[label]` amount (which is taken from the label registers).

slt SET IF LESS THAN



Format

```
slt rd, rs, rt
```

RTL

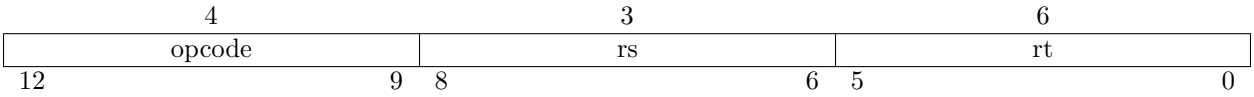
```
if(R[rs] < R[rt]):  R[rd] = 1
```

Description

Checks to see if general register `rs` is less than `rt`. If check returns true, then `R[rd] = 1`.

1.4.2 S Type Instructions

Instruction DESCRIPTION



Format

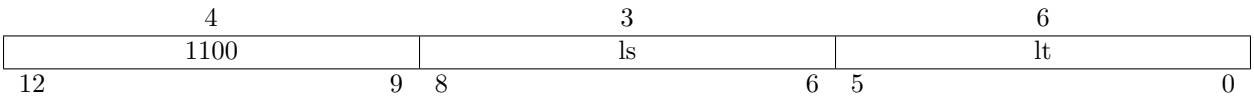
```
op rs, rt
```

RTL

Description

The instruction description

j JUMP



Format

```
j label
```

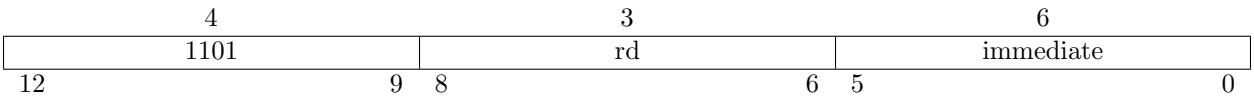
RTL

```
PC = PC + InstructionLength + R[ls] + R[lt]
```

Description

Increments the program counter an additional R[ls] + R[lt] amount (taken from label registers).

set SET TO IMMEDIATE



Format

```
set rd, immediate
```

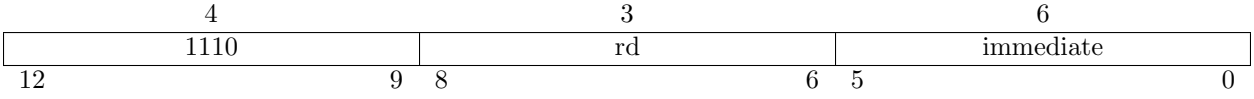
RTL

```
R[rd] = immediate
```

Description

Sets `R[rd]` to the literal (numerical) value of immediate. Assembler will enforce the size of immediate to be 6 bits or less.

sl SHIFT LEFT BITWISE



Format

```
sl rd, immediate
```

RTL

```
R[rd] = R[rd] << immediate
```

Description

Shifts the contents of `rd` left by immediate ammount. Notice that 6 bits is more than enough to be able to shift the least significant bit to the most significant bit in a 34 bit scheme.

1.4.3 TRAP instructions

A `TRAP` instruction signals the hardware that the next 3 bits after the opcode are a function designator. The format is as follows:

TRAP

EXECUTE FUNCTION DESIGNATOR

4	3	6
1111	func	rt
12 9	8 6	5 0

Format

```
TRAP func rt
```

TRAP QUIT

EXIT PROGRAM

4	3	6
1111	000	***
12 9	8 6	5 0

Format

```
TRAP QUIT, rt
```

RTL

```
exit program, returns R[rt]
```

Description

The instruction description

TRAP IN

READ FROM CHANNEL

4	3	6
1111	001	ch
12 9	8 6	5 0

Format

```
TRAP IN, ch
```

RTL

```
R[g0] = R[ch]
```

Description

Stores into general register 0 the value from channel register ch.

TRAP OUT

WRITE TO CHANNEL

4	3	6
1111	010	ch
12 9	8 6	5 0

Format

`TRAP OUT, ch`

RTL

R[ch] = R[g0]

Description

Sets the channel register `ch` to the contents of general register `g0`.

TRAP CALL

CALL FUNCTION AT LABEL

4	3	6
1111	011	label
12 9	8 6	5 0

Format

`TRAP CALL, label`

Description

Execute the function call ABI. Destination is `label`.

TRAP RET

FUNCTION RETURN

4	3	6
1111	100	***
12 9	8 6	5 0

Format

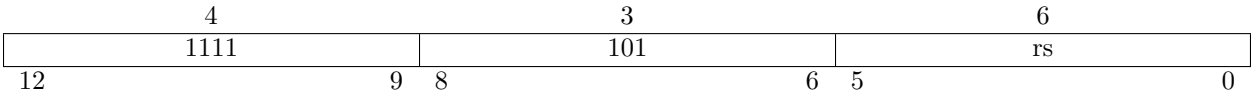
`TRAP RET`

Description

Execute the function return ABI. Destination is the return of function pointer stack.

TRAP PUSH

PUSH REGISTER ONTO STACK



Format

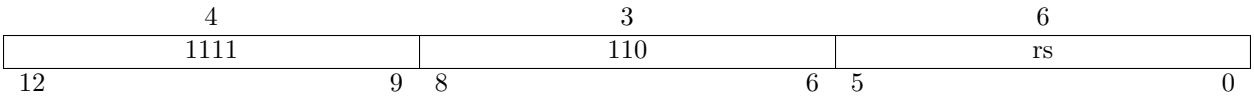
`TRAP PUSH, rs`

Description

Pushes a register value onto the register stack. It is through this mechanism that values are passed to functions.

TRAP POP

POP REGISTER FROM STACK



Format

`TRAP POP`

Description

Pop a value from the register stack and put it into `rs`.