MIPS Assembly/Instruction Formats

This page describes the implementation details of the MIPS instruction formats.

Contents

R Instructions

R instructions are used when all the data values used by the instruction are located in registers.

All R-type instructions have the following format:

```
OP rd, rs, rt
```

Where "OP" is the mnemonic for the particular instruction. *rs*, and *rt* are the source registers, and *rd* is the destination register. As an example, the **add** mnemonic can be used as:

```
add $s1, $s2, $s3
```

Where the values in *$s2* and *$s3* are added together, and the result is stored in *$s1*. In the main narrative of this book, the operands will be denoted by these names.

R Format

Converting an R mnemonic into the equivalent binary machine code is performed in the following way:

opcode	rs	rt	rd	shift (shamt)	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

opcode
: The opcode is the machinecode representation of the instruction mnemonic. Several related instructions can have the same opcode. The opcode field is 6 bits long (bit 26 to bit 31).

rs, rt, rd

The numeric representations of the source registers and the destination register. These numbers correspond to the $X representation of a register, such as $0 or $31. Each of these fields is 5 bits long. (25 to 21, 20 to 16, and 15 to 11, respectively). Interestingly, rather than rs and rt being named r1 and r2 (for source register 1 and 2), the registers were named "rs" and "rt" because t comes after s in the alphabet. This was most likely done to reduce numerical confusion.

Shift (shamt)
Used with the shift and rotate instructions, this is the amount by which the source operand *rs* is rotated/shifted. This field is 5 bits long (6 to 10).

Funct
For instructions that share an opcode, the **funct** parameter contains the necessary control codes to differentiate the different instructions. 6 bits long (0 to 5). Example: Opcode 0x00 accesses the ALU, and the funct selects which ALU function to use.

Function Codes

Because several functions can have the same opcode, R-Type instructions need a function (Func) code to identify what exactly is being done - for example, 0x00 refers to an ALU operation and 0x20 refers to ADDing specifically.

Shift Values

I Instructions

I instructions are used when the instruction must operate on an immediate value and a register value. Immediate values may be a maximum of 16 bits long. Larger numbers may not be manipulated by immediate instructions.

I instructions are called in the following way:

```
OP rt, rs, IMM
```

Where *rt* is the target register, *rs* is the source register, and *IMM* is the immediate value. The immediate value can be up to 16 bits long. For instance, the **addi** instruction can be called as:

```
addi $s1, $s2, 100
```

Where the value of $s2 plus 100 is stored in $s1.

I Format

I instructions are converted into machine code words in the following format:

opcode	rs	rt	IMM
6 bits	5 bits	5 bits	16 bits

Opcode
The 6-bit opcode of the instruction. In I instructions, all mneumonics have a one-to-one correspondence with the underlying opcodes. This is because there is no **funct** parameter to differentiate instructions with an identical opcode. 6 bits (26 to 31)

rs, rt
The source and target register operands, respectively. 5 bits each (21 to 25 and 16 to 20, respectively).[1] (http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Mips/format.html)

IMM
: The 16 bit immediate value. 16 bits (0 to 15). This value is usually used as the offset value in various instructions, and depending on the instruction, may be expressed in two's complement.

J Instructions

J instructions are used when a jump needs to be performed. The J instruction has the most space for an immediate value, because addresses are large numbers.

J instructions are called in the following way:

```
OP LABEL
```

Where *OP* is the mnemonic for the particular jump instruction, and *LABEL* is the target address to jump to.

J Format

J instructions have the following machine-code format:

Opcode	Pseudo-Address

Opcode
: The 6 bit opcode corresponding to the particular jump command. (26 to 31).

Address
: A 26-bit shortened address of the destination. (0 to 25). The two most LSBits are removed, and the 4 MSBits are removed, and assumed to be the same as the current instruction's address.

FR Instructions

FR instructions are similar to the R instructions described above, except they are reserved for use with floating-point numbers:

Opcode	fmt	ft	fs	fd	funct

FI Instructions

FI instructions are similar to the I instructions described above, except they are reserved for use with floating-point numbers:

Opcode	fmt	ft	Imm

Opcodes

The following table contains a listing of MIPS instructions and the corresponding opcodes. Opcode and funct numbers are all listed in hexadecimal.

Mnemonic	Meaning	Type	Opcode	Funct
`add`	Add	R	0x00	0x20
`addu`	Add Unsigned	R	0x00	0x21
`and`	Bitwise AND	R	0x00	0x24
`div`	Divide	R	0x00	0x1A
`divu`	Unsigned Divide	R	0x00	0x1B
`jr`	Jump to Address in Register	R	0x00	0x08
`mfhi`	Move from HI Register	R	0x00	0x10
`mflo`	Move from LO Register	R	0x00	0x12
`mult`	Multiply	R	0x00	0x18
`multu`	Unsigned Multiply	R	0x00	0x19
`nor`	Bitwise NOR (NOT-OR)	R	0x00	0x27
`xor`	Bitwise XOR (Exclusive-OR)	R	0x00	0x26
`or`	Bitwise OR	R	0x00	0x25
`slt`	Set to 1 if Less Than	R	0x00	0x2A
`sltu`	Set to 1 if Less Than Unsigned	R	0x00	0x2B
`sll`	Logical Shift Left	R	0x00	0x00
`srl`	Logical Shift Right (0-extended)	R	0x00	0x02
`sra`	Arithmetic Shift Right (sign-extended)	R	0x00	0x03
`sub`	Subtract	R	0x00	0x22
`subu`	Unsigned Subtract	R	0x00	0x23
`j`	Jump to Address	J	0x02	NA
`jal`	Jump and Link	J	0x03	NA
`beq`	Branch if Equal	I	0x04	NA
`bne`	Branch if Not Equal	I	0x05	NA
`addi`	Add Immediate	I	0x08	NA
`addiu`	Add Unsigned Immediate	I	0x09	NA
`slti`	Set to 1 if Less Than Immediate	I	0x0A	NA
`sltiu`	Set to 1 if Less Than Unsigned Immediate	I	0x0B	NA
`andi`	Bitwise AND Immediate	I	0x0C	NA
`ori`	Bitwise OR Immediate	I	0x0D	NA
`lui`	Load Upper Immediate	I	0x0F	NA
`mfc0`	Move from Coprocessor 0	R	0x10	NA
`lw`	Load Word	I	0x23	NA
`lbu`	Load Byte Unsigned	I	0x24	NA
`lhu`	Load Halfword Unsigned	I	0x25	NA
`sb`	Store Byte	I	0x28	NA
`sh`	Store Halfword	I	0x29	NA

Mnemonic	Meaning	Type	Opcode	Funct
`sw`	Store Word	I	0x2B	NA