MIPS Instruction Formats

- All MIPS instructions are encoded in binary.
- All MIPS instructions are 32 bits long.

  (Note: some assembly langs do not have uniform length for all instructions)
- There are three instruction categories: R-format (most common), I-format, and J-format.
- All instructions have:
  - op (or opcode): operation code (specifies the operation) (first 6 bits)

R-format Instructions

- Have op 0. (all of them!)
- Also have:
  - rs: 1st register operand (register source) (5 bits)
  - rt: 2nd register operand (5 bits)
  - rd: register destination (5 bits)
  - shamt: shift amount (0 when N/A) (5 bits)
  - funct: function code (identifies the specific R-format instruction) (6 bits)

Name	Format	Layout						Example
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
		op	rs	rt	rd	shamt	funct	
add	R	0	2	3	1	0	32	add $1, $2, $3
addu	R	0	2	3	1	0	33	addu $1, $2, $3
sub	R	0	2	3	1	0	34	sub $1, $2, $3
subu	R	0	2	3	1	0	35	subu $1, $2, $3
and	R	0	2	3	1	0	36	and $1, $2, $3

or	R	0	2	3	1	0	37	or $1, $2, $3
nor	R	0	2	3	1	0	39	nor $1, $2, $3
slt	R	0	2	3	1	0	42	slt $1, $2, $3
sltu	R	0	2	3	1	0	43	sltu $1, $2, $3
sll	R	0	0	2	1	10	0	sll $1, $2, 10
srl	R	0	0	2	1	10	2	srl $1, $2, 10
jr	R	0	31	0	0	0	8	jr $31

NOTE: op is 0, so funct disambiguates

Example

add $s0, $s1, $s2 (registers 16, 17, 18)

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32
000000	10001	10010	10000	00000	100000

NOTE: Order of components in machine code is different from assembly code. Assembly code order is similar to C, destination first. Machine code has destination last.

C: a = b + c
assembly code: add $s0, $s1, $s2 # add **rd**, rs, rt
machine code: 000000 10001 10010 10000 0000 100000
(op rs rt **rd** shamt funct)

I-format Instructions

- Have a constant value immediately present in the instruction.
- Also have:
  - rs: register containing base address (5 bits)
  - rt: register destination/source (5 bits)
  - immediate: value or offset (16 bits)

Name	Format	Layout						Example
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
		op	rs	rt	immediate			
beq	I	4	1	2	25 (offset)			beq $1, $2, 100

bne	I	5	1	2	25 (offset)	bne $1, $2, 100
addi	I	8	2	1	100	addi $1, $2, 100
addiu	I	9	2	1	100	addiu $1, $2, 100
andi	I	12	2	1	100	andi $1, $2, 100
ori	I	13	2	1	100	ori $1, $2, 100
slti	I	10	2	1	100	slti $1, $2, 100
sltiu	I	11	2	1	100	sltiu $1, $2, 100
lui	I	15	0	1	100	lui $1, 100
lw	I	35	2	1	100 (offset)	lw $1, 100($2)
sw	I	43	2	1	100 (offset)	sw $1, 100($2)

Example

lw $t0, 32($s3) (registers 8 and 19)

op	rs	rt	immediate
35	19	8	32
100011	10011	01000	0000000000100000

Example: beq

beq $t0, $zero, ENDIF

The offset stored in a beq (or bne) instruction is the number of instructions from the PC (the instruction after the beq instruction) to the label (ENDIF in this example). Or, in terms of addresses, it is the difference between the address associated with the label and the PC, divided by four.

offset = (addrFromLabelTable - PC) / 4

In the example above, if the beq instruction is at address 1004, and thus the PC is 1008, and if ENDIF is at address 1028, then the value stored in the machine instruction would be

offset = (1028 - 1008) / 4 = 5

op	rs	rt	immediate
4	8	0	5
000100	01000	00000	0000000000000101

J-format Instructions

- Have an address (part of one, actually) in the instruction.

Name	Format	Layout						Example

		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
		op	address					
j	J	2	2500					j 10000
jal	J	3	2500					jal 10000

Example

```
j LOOP
```

The address stored in a j instruction is 26 bits of the address associated with the specified label. The 26 bits are achieved by dropping the high-order 4 bits of the address and the low-order 2 bits (which would always be 00, since addresses are always divisible by 4).

```
address = low-order 26 bits of (addrFromLabelTable/4)
```

In the example above, if LOOP is at address 1028, then the value stored in the machine instruction would be 257.

op	address
2	257
000010	00000000000000000100000001