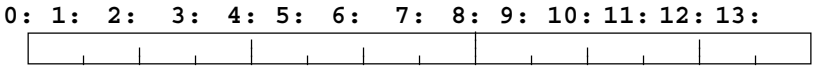
Loads, Stores and Branches

Loads and stores reference memory for data, while the branches reference memory for instructions.

Memory Organization



- Memory is a linear array of bytes (8-bits)
- References can be to individual bytes, words (32), half words (16) or double words (64)
- References should be aligned by data type

Alignment

Type	Address LSB
byte	any
halfword	0
word	00
double word	000

Byte Order

Big-endian: 0 is MSB

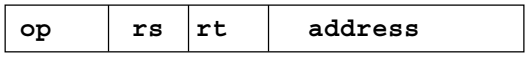
0	1	2	3

Little-endian: 0 is LSB

3	2	1	0

Load & Store Instructions

- MIPS is a **load store architecture** ... all operations are performed on values in registers, so data in memory must be loaded from or stored to memory
- Load, Store are I-format instructions, where the “immediate” field is called “address” and the instruction format looks like subscripting



I-type instructions

```
lw    rt, address(rs)     Load Word
sw    rt, address(rs)     Store Word
```

Others
lb sb
lh sh
ld sd

Base Addressing

- Load, Store use “base addressing” ... the **effective address**, the address used to reference memory, is the sum of the **address** field and the contents of register **rs**

Example: `lw $7, 4100($3)`

```
                0000000000000000 0001000000000100   4100
                                 +                    +
Reg 3:          00000000000000000000000000011000     24
                00000000000000000001000000011100    4124
```

The effective address of the memory word referenced

Base Addressing For Indexing

- "Base/offset addressing" gets its name from the idea that a series of addresses could be created from a fixed base and a series of offsets:

```
4100 +  0 = 4100
4100 +  4 = 4104
4100 +  8 = 4108
4100 + 12 = 4112
     ...
```

Base

Offset

Effective Address

4100:	c
4104:	2933c400
4108:	ffffffff
4112:	0

Conclusion: Keep offset in the **rs** register

Base Addressing For Relative Ref

- Base/offset addressing can be used for referencing relative to a specific point in memory, e.g. the top of a stack:

Reg 4: 00000000000000000001000000010000

```
3 From Top = Top + -12 = -12($4)
2 From Top = Top +  -8 =  -8($4)
1 From Top = Top +  -4 =  -4($4)
     Top = Top +   0 =   0($4)
    ...
```

4100:	c
4104:	2933c400
4108:	ffffffff
4112:	0

Offset

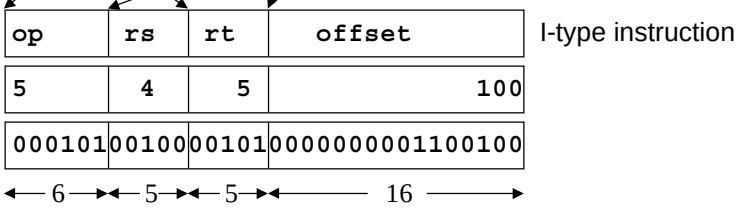
Base

Conclusion: Keep base in the **rs** register

Branches

- Branch instructions perform a test, and if true change the control flow to begin executing at the Label

```
beq  $8, $9, Label  # if $8=$9 goto Label
bne  $5, $4, Label  # if $5≠$4 goto Label
```



beq has op = 4

PC Relative Addressing

- The **offset** is the number of *instructions* forward or backward that must be skipped to get to the instruction labeled Label
- Thus, 4*offset + PC+4 is the address of the instruction labeled with Label
  - PC+4 is used because the PC already points to the next instruction
  - Figuring offsets manually is difficult because of pseudoinstructions ... leave it to the assembler
  - Forward is a + **offset**, backward is a - **offset**

An R-type Instruction For Testing

- The "set less than" instruction is an R-type instruction that tests the less than relation (<)

  `slt result, left_operand,right_operand`

  If the left operand is (strictly) less than the right operand as signed integers, set result register to 1; otherwise set it to 0

Example: **`slt $7, $3, $4  # Is Reg3 < Reg4?`**

0	3	4	7	0	42
000000	00011	00100	00111	00000	101010

Other Conditions

- Branches are available for other conditions --
  - bgt, bge, blt, ble have form: bxx src1, scr2, Label
- These branches are pseudoinstructions constructed by the assembler from **slt** and **bne**

Example: **`blt  $4, $5, Label`**

becomes

```
slt $1, $4, $5    # Set R1 to 1 if $4 < $5
bne $1, $0, Label # $1≠$0 implies slt true
```