Machine Language

We need to map MIPS assembly instructions into an executable binary machine language program.

We can covert each of these assembly language elements into a binary code:

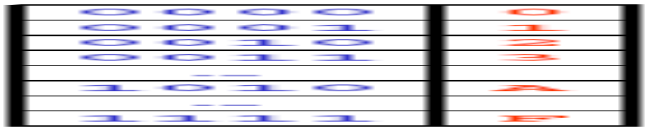
- *registers*: 5 bits unsigned is enough to store registers 0-31.
- *instructions*: 6-8 bits are enough for all instructions. See the instruction sheet for corresponding number to each instruction (in hexadecimal).
- *labels & variables*: 32 bits are enough for memory addresses
- *constants*: stored as described in "Data Representation" section
- *shift amounts*: 5 bits

Now we need to pack all the above into 32 bits per instruction.

We need more than 32 bits to generally store them all, so we store depending on type of instructions.

Each instruction's type (R,I or J) can be found in the instructions sheet.

1) R-Type (registers only)



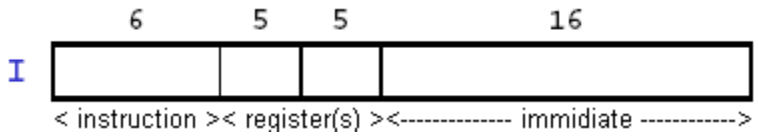
- The first 6 bits are always zero for this type.
- The next 3 set of 5-bits are for registers.
- The 4th 5-bit set is zero (except for shift – see below)
- The last 6 bits are for the instruction.

e.g. `add $t0, $s0, $a0`

000 000	10000	00100	01000	00000	100 000

Note: For **Shift** instructions, the shift amount is small enough (5 bits - unsigned) that it can be stored in the last 5-bit part of R-type instructions. This is why MIPS has `sll` and `sllv` instead of slli and sll.

2) I-Type (immediate**)**



- The first 6 bits are for the instruction.
- The next 2 set of 5-bits are for the registers.
- The final 16 bits are for the immediate.

e.g. `addi $t0, $s0, 7`

001 000	10000	01000	0000 0000 0000 0111

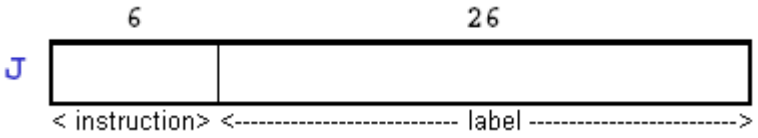
Another example that uses variables:

```
      .data
x:    .byte 4            # address of x = 0
y:    .ascii “York”      # address of y = 1
z:    .word 7, 2         # address of z = 8
      .text
      lw $t1, z($t2)     # t2 = 0
```

100 011	01010	01001	0000 0000 0000 1000

In this example ‘z’ is replace with its address (8) and then stored in the last 16 bits.

3) J-Type (jump: j & jal**)**



- The first 6 bits are for the instruction which is either 1 (`ja`) or 3 (`jal`).
- The next 16 bytes are the address of label.

e.g.

```
      addi  $t0. $0, 7
      bgtz  $t0, pos
```

```
        addi   $s0, $0, 5
        j      done
pos:    addi   $s0, $0, 6
```

But the address of pos might be somewhere in memory where the memory address is too large to fit in 26 bits. See the next section for the solution to this.