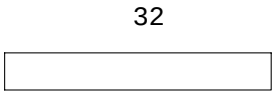
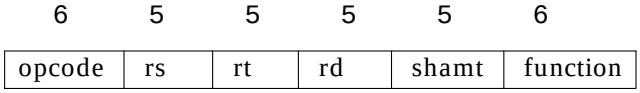
Lecture 7

On the structure of instructions -- how the 32 bits per each instruction are portioned into sections, and what each of those sections means. On different types of instructions -- R-type, I-type, and J-type.

- Each instruction in MIPS is 32 bits long



Below is one type of instruction, called an R-type instruction



An R-type instruction is so called because the data values used are located in registers

rd - register destination, rs - register source, rt - register target

- In memory, the bits are still stored in 8 bit blocks, but this is how we organize the bits by their semantic meanings
- 6 leftmost bits are the **opcode**
  - The opcode tells us what operation will be performed
  - e.g. opcode for addition is *0*
- We also have a section at the front specifying more information called the **function** field -- there are many instructions that share the same opcode, but they are differentiated by the bits in the *function* field.
  - e.g. ADD, SUB, MULT, all have an opcode of 0, but different function fields
- Example R-type instruction:

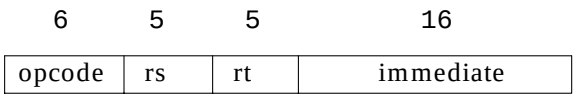
```
ADD $s0, $s0, $t0

    rd    rs   rt
```

- When assembling, register names are converted to their corresponding "number" (each has one, as specified by the architecture -- e.g. $s0 - $s8 are 16 - 23), and these numbers are stored as bits in the *rd/rs/rt* fields.

- You can also store each byte as hexadecimal, for convenience.
  - Hexadecimal is commonplace because it's very simple to convert between it and binary -- simply group each 4 bits together and convert them to a hexadecimal digit (0 - F), and voila.

Below is another type of instruction, called an I-type instruction



An I-type instruction is so called because the data values used are immediate values

- Immediate - a piece of data that is stored as part of the instruction itself instead of being in a memory location or a register. Immediate values are typically used in instructions that load a value or performs an arithmetic or a logical operation on a constant

- An example I-type instruction is as follows:

```
ADDI $s3, $zero, 4
```

- Note that the immediate value can be written in decimal, hexadecimal, binary, or any other format the assembler accepts (they will all be converted to binary upon assembling)
- Also note that the immediate value is stored in two's complement notation

Another slightly different-looking I-type intruction is:

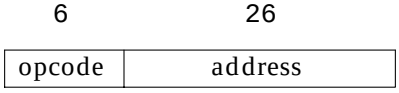
```
LW $s3, 4($s7)
```

'4' here is a 16-bit signed integer offset

A MIPS instruction is 32 bits (always). A MIPS memory address is 32 bits (always). How can a load or store instruction specify an address that is the same size as itself?

An instruction that refers to memory uses a base register and an offset. The base register is a general purpose register that contains a 32-bit address. The offset is a 16-bit signed integer contained in the instruction. The sum of the address in the base register with the (sign-extended) offset forms the memory address.

The last type of instruction is called an J-type (jump) instruction



- Examples of J-type instructions include *J* (which is used in loops) and *JAL* (which is used to implement functions)