10 | J Format Instructions

(I is from last week)

J Format Instructions

J (jump format) Example: j lp

- opcode 6 bits
- jump (word) address 26 bits
- We're talking about unconditional jump instructions, like j and jal

opcode	**jump (word) address**
31 … 26	25 … 0

But memory addresses are 32 bits. How are we squeezing a 32-bit memory address into 26 bits? Unlike conditional branches, these are **absolute addresses**, not relative.

> Low order 2 bits are not in the instruction (always 00) because an instruction address always ends in 0, 4, 8, or c. This is because all instructions are 4 bytes. The first instruction is at location 0, then 4, then 8, then c, then 10, then 14, counting by 4's in hexadecimal. The low-order digit is always going to be 0, 4, 8, or c, meaning the last two bits are always 00.
>
> **Example:** AKA: 0x0 = 0000, 0x4 = 0100, 0x8 = 1000, 0xc = 1100
>
> We're also going to assume that the first 4 high-order bits are also all 0's (0000_2, or first hexadecimal digit), so therefore instructions begin in the lower-order set of memory, not high. The program will never be so huge that it will reach the highest-order bits.
>
> ***Thus, use bits 2 to 27 for the full 32 bit address.***

Example:

```
test:                          # at location 0x004003c (jump here)
        j       test
```

opcode	jump (word) address
00 0010	0000 0100 0000 0000 0000 0011 11

- Opcode is 02 (hexadecimal) → 0000 0010 (binary) → 00 0010 (fit to 6 bits)
- Target address is 0x004003c → 0000 0000 0100 0000 0000 0000 0011 1100 (binary) → **0000 0100 0000 0000 0000 0011 11** (removing first 4 and last 2 bits, 26bit address)

Now, we group these bits into those of 4: **0000 1000 0001 0000 0000 0000 0000 1111**.
In hexadecimal, it's **$0810000f_{16}$**. This can also be checked with MARS!