# The MIPS R4000, part 6: Memory access (unaligned)

devblogs.microsoft.com/oldnewthing/20180409-00

April 9, 2018

Raymond Chen

Unaligned memory access on the MIPS R4000 is performed with pairs of instructions.

```
LWL     rd, n+3(rs)     ; load word left
LWR     rd, n(rs)       ; load word right
```

This is easier to explain with a diagram rather than with a formula.

| | n+3(rs) | | | n(rs) | | | |
|---|---|---|---|---|---|---|---|
| | ↓ | | | ↓ | | | |
| AA | BB | CC | DD | EE | FF | GG | HH |
| | | | | | | | |
| | 11 | 22 | 33 | 44 | | rd | |
| | `LWL rd, n+3(rs)` | | | | | | |
| | BB | CC | DD | 44 | | rd | |
| | `LWR rd, n(rs)` | | | | | | |
| | BB | CC | DD | EE | | rd | |

You give the “load word left” instruction the effective address of the most significant byte of the unaligned word you want to load, and it picks out the correct bytes from the enclosing word and merges them into the upper bytes of the destination register.

The “load word right” works analogously: You give it the effective address of the least significant byte of the unaligned word you want to load, and it picks out the correct bytes from the enclosing word and merges them into the lower bytes of the destination register.

Since the results are combined via merging, you can issue the `LWL` and `LWR` instructions in either order, and together they will load the complete four-byte value.[1] (If the address happened to be aligned, then both instructions will load the complete word.)

There are corresponding left/right instructions for storing an unaligned word:

```
SWL     rd, n+3(rs)     ; store word left
SWR     rd, n(rs)       ; store word right
```

These are the counterparts to the load versions. They store the upper and lower part of the word to the corresponding parts of memory.

For unaligned halfword access, you might be tempted to do this:

```
; Try to load unaligned word unsigned from rs to rd
; Does this work?
LWL     rd, n+3(rs)     ; load word left
LWR     rd, n(rs)       ; load word right
ANDI    rd, rd, 0xFFFF  ; keep the lower 16 bits
```

Unfortunately, this doesn't work because the `n+3(rs)` might cross into an invalid page. Consider the case where the halfword is the very last halfword on its page: If you tried to load it as a word, you would need to load the first halfword on the next page (to fill the top 16 bits), and that could crash if the next page were invalid.

Instead, you need to perform unaligned halfword access by loading two bytes and combining them:

```
; Load unaligned word signed from rs to rd
LB      at, n+1(rs)     ; load high byte
LBU     rd, n(rs)       ; load low byte
SLL     at, at, 8       ; shift high byte into position
OR      rd, rd, at      ; combine the bytes
```

If you want to load an unaligned word unsigned, you would change the first instruction from `LB` to `LBU`.

For the same reason as loading, storing an unaligned word is done by storing the bytes separately.

```
; Store unaligned word to rd from rs
SRL     at, rs, 8       ; shift high byte into position
SB      at, n+1(rd)     ; store high byte
SB      rs, n(rd)       ; store low byte
```

The assembler provides pseudo-instructions for these unaligned memory operations:

```
ULW     rs, disp16(rd)  ; unaligned load word
USW     rs, disp16(rd)  ; unaligned store word
ULH     rs, disp16(rd)  ; unaligned load halfword signed
ULHU    rs, disp16(rd)  ; unaligned load halfword unsigned
USH     rs, disp16(rd)  ; unaligned store halfword

; and again for absolute addressing
ULW     rs, global_var  ; unaligned load word
USW     rs, global_var  ; unaligned store word
ULH     rs, global_var  ; unaligned load halfword signed
ULHU    rs, global_var  ; unaligned load halfword unsigned
USH     rs, global_var  ; unaligned store halfword
```

Mind you, these pseudo-instructions don't help you when debugging. The debugger shows the underlying real instructions.

If you've been paying attention, you may have noticed that the `ULW rd, disp16(rs)` pseudo-instruction fails if *rs* and *rd* happen to be the same register, because the `LWL` will damage the base register before it can be used to load the right half. In that case, the assembler uses this alternate version:

```
LWL     at, n+3(rs)     ; load word left into temporary
LWR     at, n(rs)       ; load word right into temporary
OR      rs, at, at      ; move to final destination
```

Okay, next time we'll look at atomic memory operations.

[1] In versions of the MIPS architecture with load delay slots, there was a special exception for `LWL` and `LWR`: You were allowed to issue them directly after the other, and they would merge correctly, provided they target different bytes of the same destination register or update the entire destination.

Raymond Chen

**Follow**