The AArch64 processor (aka arm64), part 3: Addressing modes

devblogs.microsoft.com/oldnewthing/20220728-00

July 28, 2022

Raymond Chen

Every addressing mode on AArch64 begins with a base register, which can be any numbered register or *sp*. On top of that, you can add various sprinkles.

In the discussion, the term *size* refers to the size of the data being transferred, and *sizeshift* is the base-2 logarithm of that size:

Operand	size	sizeshift
byte	1	0
halfword	2	1
word	4	2
doubleword	8	3

For illustration purposes, I'll use the `LDR` instruction, which loads a register.

Register indirect with offset

```
    ldr     x0, [Xn/sp, #imm]
    ldr     x0, [Xn/sp]         ; #0 is implied if omitted
```

This loads a value from the address calculated by adding the immediate to the value in the *Xn* register or *sp*.

The immediate can be a signed integer offset in the range −256 to +255, or an unsigned multiple of the operand size up to 4095 × *size*.

Size	Signed reach	Unsigned reach
byte	−256 to +255	0 to 4095

halfword		0 to 8190
word		0 to 16380
doubleword		0 to 32760

Register indirect with pre-increment

Putting an exclamation point after the close-bracket means that the calculated effective address is written back to the base register.

```
    ; load from (Xn/sp + imm)
    ; then set Xn/sp = Xn/sp + imm
    ldr     x0, [Xn/sp, #imm]!
```

Register indirect with post-increment

Putting the immediate offset outside the close-bracket means that the base register is adjusted *after* the memory is read.

```
    ; load from Xn/sp
    ; then set Xn/sp = Xn/sp + imm
    ldr     x0, [Xn/sp], #imm
```

PC-relative with offset

```
    ldr     x0, [pc, #imm]
```

The PC-relative addressing mode reads memory from a position given as a signed offset from the current instruction. The offset must be a multiple of 4, and the reach is ±1MB.

This instruction is typically used to load large constants from memory, and the disassembler does the math for you and decodes it as

```
    ldr     x0, =imm
```

by calculating the effective address and fetching the value from that location.

The assembler typically generates literals into the code segment between subroutines, and the large reach of this instruction means that the need to dump literals prematurely is largely a thing of the past. (By comparison, AArch32's PC-relative addressing mode had a reach of only ±4KB, so it was not uncommon to dump literals in the middle of a function.)

Register indirect with index

```
    ldr     x0, [Xn/sp, Rn/zr, extend]
```

This addressing mode takes the *Rn/zr*, transforms it according to the extended register operation *extend*, and adds the result to the *Xn/sp* register to form the final address.

For memory access, the following extended register operations are available:

- `UXTW`
- `UXTX` (aka `LSL`)
- `SXTW`
- `SXTX`

The only acceptable shifts are zero and *sizeshift*. This means that the index register can be treated either as a byte offset or as an element index, where the element is the size of the operand. For example, if you are loading a halfword, then the index register is either a byte offset of a halfword index.

Writing out all the possibilities produces these possible extended registers:

Extended	Effective address	Index format

`[a, b, UXTW #0]`
`[a, b, UXTW]`
	`a + (uint32_t)b`	32-bit unsigned byte offset.

`[a, b, UXTW #sizeshift]`	`a + (uint32_t)b * size`	32-bit unsigned element offset.

`[a, b, SXTW #0]`
`[a, b, SXTW]`
	`a +  (int32_t)b`	32-bit signed byte offset.

`[a, b, SXTW #sizeshift]`	`a +  (int32_t)b * size`	32-bit signed element offset.

`[a, b, UXTX #0]`
`[a, b, UXTX]`
`[a, b, LSL #0]`
`[a, b]`
	`a + (uint64_t)b`	64-bit unsigned byte offset.

`[a, b, UXTX #sizeshift]`
`[a, b, LSL #sizeshift]`
	`a + (uint64_t)b * size`	64-bit unsigned element offset.

`[a, b, SXTX #0]`	`a +  (int64_t)b`	64-bit signed byte offset.
`[a, b, SXTX #sizeshift]`	`a +  (int64_t)b * size`	64-bit signed element offset.

If no extended operation is provided, it defaults to `UXTX #0`, which means “use the whole register, no shift.”

There is no pre-increment or post-increment option for the indexed addressing modes.

Okay, so those are the addressing modes. Quite a lot to choose from. Next time, we’ll start doing arithmetic.

Raymond Chen

Follow