

Since $immr \leq imms$, the right-rotation by $immr$ is the same as a right-shift by $immr$.

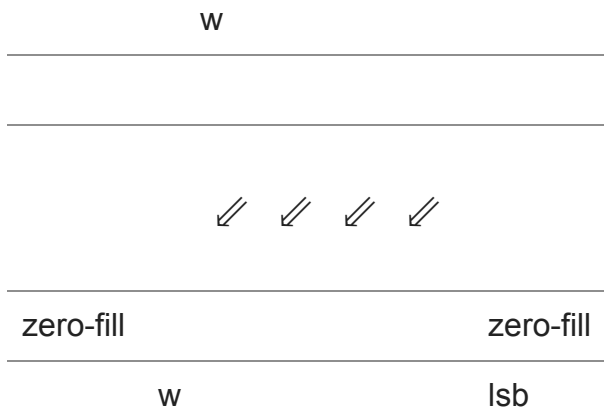
And then we have the other case, where $immr > imms$:

```

; unsigned bitfield insert into zeroes
; (used when immr > imms)
; extract low-order w bits and shift left by lsb
ubfiz  Rd/zr, Rn/zr, #lsb, #w

```

The **UBFIZ** instruction reinterprets the **UBFM** as a bitfield insertion, and reinterprets the right-rotation as a left-shift. This reinterpretation is valid because $immr > imms$, so we are always rotating more bits than we extracted.



There is also a signed version of this instruction:

```

; signed bitfield move
;
; if immr ≤ imms:
;   take bits immr through imms and rotate right by immr
;   sign-fill upper bits
;
; if immr > imms:
;   take imms+1 low bits and rotate right by immr
;   sign-fill upper bits

sbfm  Rd/zr, Rn/zr, #immr #imms

```

This behaves the same as the unsigned version, except that the upper bits are filled with the sign bit of the bitfield. Like **UBFM**, the **SBFM** instruction is also never seen in the wild; it is always replaced by a pseudo-instruction.

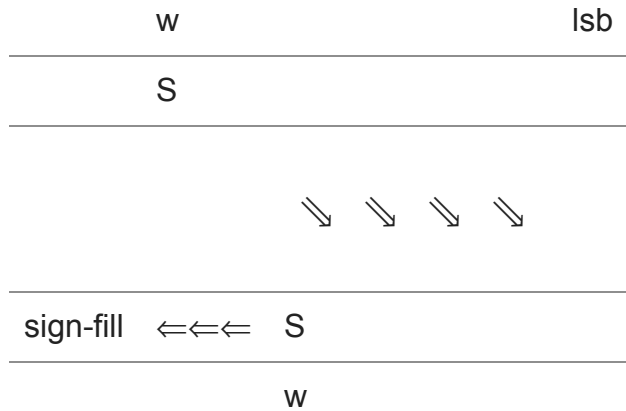
```

; signed bitfield extract
; (used when immr ≤ imms)
; extract w bits starting at position lsb
; sign-fill upper bits
sbfx    Rd/zr, Rn/zr, #lsb, #w

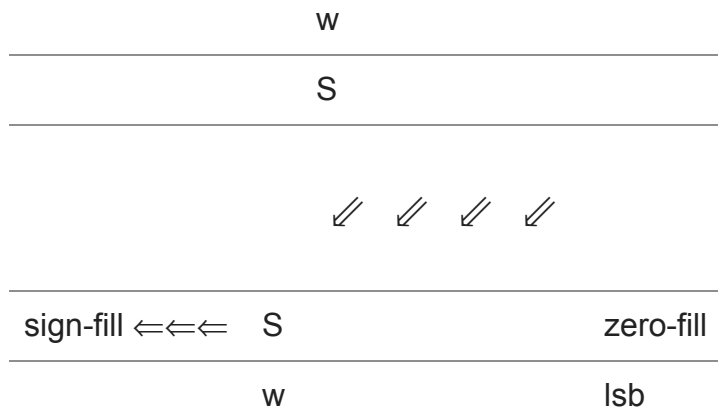
; signed bitfield insert into zeroes
; (used when immr > imms)
; extract low-order w bits and shift left by lsb
; sign-fill upper bits
sbfiz   Rd/zr, Rn/zr, #lsb, #w

```

Here is the operation of **SBFX** in pictures:



And here is **SBFIZ** :



Note that in the case of **SBFIZ**, the lower bits are still zero-filled.

The last bitfield opcode is **BFM**, which follows the same pattern, but just combines the results differently:

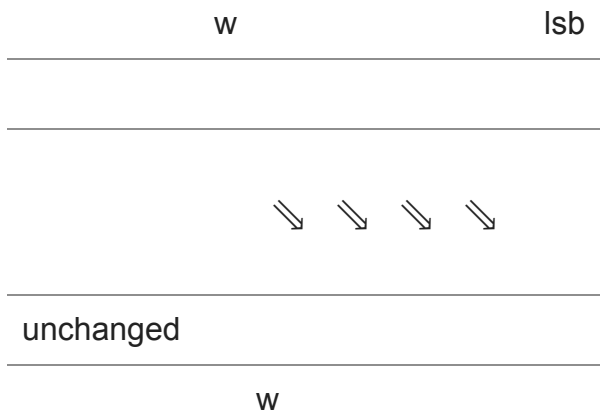
```
; bitfield move
;
; if immr ≤ imms:
;   take bits immr through imms and rotate right by immr
;   merge with existing bits in destination
;
; if immr > imms:
;   take imms+1 low bits and rotate right by immr
;   merge with existing bits in destination

bfm    Rd/zr, Rn/zr, #immr #imms
```

Again, you will never see this instruction in the wild because it always disassembles as a pseudo-instruction:

```
; bitfield extract and insert low
; (used when immr ≤ imms)
; replace bottom w bits in destination
; with w bits of source starting at lsb
;
; Rd[w-1:0] = Rn[lsb+w-1:lsb]
;
bfxil  Rd/zr, Rn/zr, #lsb, #w
```

The **BFXIL** instruction is like the **UBFX** and **SBFX** instructions, but instead of filling the unused bits with zero or sign bits, the original bits of the destination are preserved.

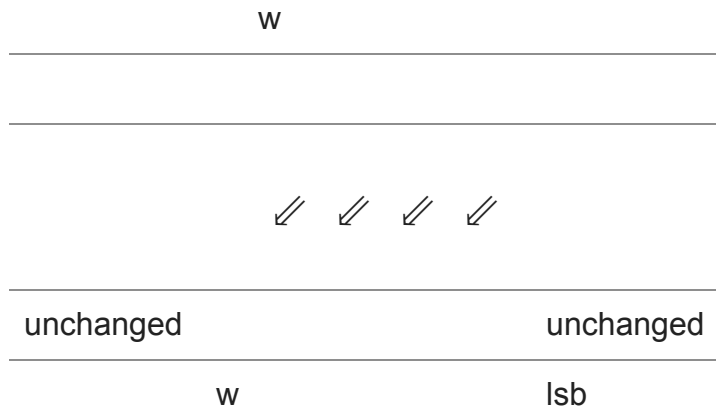


```

; bitfield insert
; (used when immr > imms)
; replace w bits in destination starting at lsb
; with low w bits of source
;
; Rd[lsb+w-1:lsb] = Rn[w-1:0]
;
bfi    Rd/zr, Rn/zr, #lsb, #w

```

The **BFI** instruction is like the **UBFIZ** and **SBFIZ** instructions, but instead of filling the unused bits with zero or sign bits, the original bits of the destination are preserved.

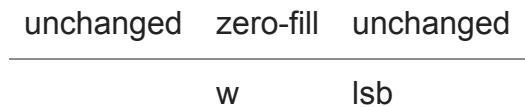


```

; bitfield clear
; replace w bits in destination starting at lsb
; with zero
;
; Rd[lsb+w-1:lsb] = 0
;
bfc    Rd/zr, #lsb, #w    ; bfi Rd/zr, zr, #lsb, #w

```

The **BFC** instruction just inserts zeroes.



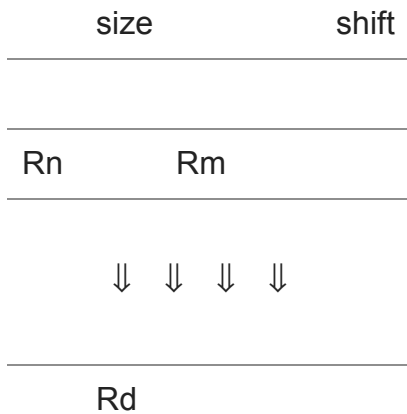
The last instruction in the bitfield manipulation category is word/doubleword extraction.

```

; extract a register from a pair of registers
;
; Wd = ((Wn << 32) | Wm)[lsb+31:lsb]
; Xd = ((Xn << 64) | Xm)[lsb+63:lsb]
;
extr   Rd/zr, Rn/zr, Rm/zr, #lsb

```

The *extract register* instruction treats its inputs as a register pair and extracts a register-sized stretch of bits from them. This can be used to synthesize multiword shifts.



Note that the two input registers are concatenated in big-endian order.

It turns out that a lot of other operations can be reinterpreted as bitfield extractions. We'll look at some of them next time.

Bonus chatter: AArch32 also had instructions `bfi`, `bfc`, `ubfx`, and `sbfx`, but each was treated as a unique instruction. AArch64 generalizes them to cover additional scenarios, leaving the classic instructions as special cases of the generalized instructions.

[Raymond Chen](#)

Follow

