Changes to `int` handling

- Invalid octal literals
- Negative bitshifts
- Out of range bitshifts
- Changes to Division By Zero

Invalid octal literals

Previously, octal literals that contained invalid numbers were silently truncated (`0128` was taken as `012`). Now, an invalid octal literal will cause a parse error.

Negative bitshifts

Bitwise shifts by negative numbers will now throw an `ArithmeticError`:

```
<?php
var_dump(1 >> -1);
?>
```

ENTITY

```
int(0)
```

ENTITY

```
Fatal error: Uncaught ArithmeticError: Bit shift by negative number in /tmp/test.php:2
Stack trace:
#0 {main}
  thrown in /tmp/test.php on line 2
```

Out of range bitshifts

Bitwise shifts (in either direction) beyond the bit width of an `int` will always result in 0. Previously, the behaviour of such shifts was architecture dependent.

Changes to Division By Zero

Previously, when 0 was used as the divisor for either the divide (/) or modulus (%) operators, an E_WARNING would be emitted and `false` would be returned. Now, the divide operator returns a float as either +INF, -INF, or NAN, as specified by IEEE 754. The modulus operator E_WARNING has been removed and will throw a `DivisionByZeroError` exception.

```
<?php
var_dump(3/0);
var_dump(0/0);
var_dump(0%0);
?>
```

ENTITY

```
Warning: Division by zero in %s on line %d
bool(false)

Warning: Division by zero in %s on line %d
bool(false)

Warning: Division by zero in %s on line %d
bool(false)
```

ENTITY

```
Warning: Division by zero in %s on line %d
float(INF)

Warning: Division by zero in %s on line %d
float(NAN)

PHP Fatal error:  Uncaught DivisionByZeroError: Modulo by zero in %s line %d
```

□□□ #2
□ □□□ □□□ 2026-10-31 17:33:04 UTC
□ □□□ □□□ 2026-10-31 17:33:05 UTC