Store X Indexed by Y and Vice-Versa With SHX/SHY

The 6510 doesn't have a stx abs,y or sty abs,x. So instead you would normally do something like this:

```
//store X indexed by Y:
txa
sta address,y

//store Y indexed by X:
tya
sta address,x
```

Which each take 7 cycles. But instead you can do this, which only takes 5:

```
//store X indexed by Y:
shx address,y

//store Y indexed by X:
shy address,x
```

However, there's a little catch. The value is and'ed with the high byte of the address before it's stored. And to make it even more confusing, it's actually the high byte + 1. So if you store it in $fe00, the value is and'ed with $ff, and therefore unaffected. But since you don't always use all the 8 bits, other addresses might work as well. E.g. if your values are C64 colors, which are between $00 and $0f, the upper 4 bits don't matter. So in that case all pages ending in $e will work, i.e. $0e00, $1e00, etc.

Instabilities

The opcodes are classified as unstable, but this only affects the behavior. So it doesn't risk crashing the computer like e.g. lax immediate.

The first instability is that the and'ing doesn't always take place. So sometimes the original X/Y value is stored, and sometimes it's and'ed first. But if you use value/address combinations where the and'ing doesn't matter, the result is always the same.

Another instability comes when you cross a page boundary like this:

```
ldy #$14
ldx #$01
shy $0eff,x
```

Normally you would expect the value to end up in $0f00, but for reasons unknown, the page changes to the value stored ((H + 1) & y), which in this case is ($0e + 1) & $14 = $04. So the value ends up in $0400.