Undocumented Z80 Flags

Revision 1.0 – 21st August 2018

David Banks

Introduction

In developing and testing the Z80 Decoder, it's become apparent there are still a few wrinkles in the way the Z80 sets the flags (where Zilog states they are undefined).

My starting point for the undocumented flags was Sean Riddle's excellent document titled: The Undocumented Z80 Documented, specifically section 4.

In this page I'll summarise the new information I have discovered (or possibly rediscovered)

I will use the following notation for the flags:

```
SF - Flag bit 7 - Sign Flag
ZF - Flag bit 6 - Zero Flag
YF - Flag bit 5 - Undocumented (also known as F5)
HF - Flag bit 4 - Half Carry Flag
XF - Flag bit 3 - Undocumented (also known as F3)
PF - Flag bit 2 - Parity Flag (also sometimes used for Overflow)
NF - Flag bit 1 - Negation Flag (last ALU op was subtract or compare)
CF - Flag bit 0 - Carry Flag
```

Also the function Parity() is defined to match the Z80 parity (i.e. returns 0 for odd parity, 1 for even parity)

Interrupted block instructions

When an LDxR / CPxR / INxR / OTxR instruction completes normally, the flags are exactly as described in the reference above. However, when they are interrupted, the interrupt handler sees some flags in a different state.

It seems the extra 5 T-states and the end of a block instruction that decrements the PC by two are also unexpectedly changing certain flags. Normally this would not be visible, because when the instruction repeats these flags are updated again. But in this case of an interrupt these values do become visible.

We now describe the **additional** flag changes that happen during these extra 5 T-states.

LDxR / CPxR interrupted

When LDxR / CPxR is interrupted, the following flags are modified compared to the non-interrupted LDx / CPx:

```
YF = PC.13
XF = PC.11
```

where PC is the address of the start of the instruction (i.e. the 0xED prefix)

INxR / OTxR interrupted

When INxR / OTxR is interrupted, the following flags are modified compared to the non-interrupted INx / OUTx:

```
YF = PC.13
XF = PC.11
if (CF) {
   if (data & 0x80) {
      PF = PF ^ Parity((B - 1) & 0x7) ^ 1;
      HF = (B & 0x0F) == 0x00;
   } else {
      PF = PF ^ Parity((B + 1) & 0x7) ^ 1;
      HF = (B & 0x0F) == 0x0F;
   }
} else {
   PF = PF ^ Parity(B & 0x7) ^ 1;
}
```

where PC is the address of the start of the instruction (i.e. the 0xED prefix)

SCF/CCF

In 2012 Patrik Rak discovered that the values of XF and YF following an SCF or CCF instruction depends on whether the preceding instruction modified the flags or not. Specifically:

```
if <previous instruction modified the flags> then
   YF = A.5
   XF = A.3
else
   YF = YF | A.5
   XF = XF | A.3
endif
```

I verified this to be the case with the Zilog NMOS Z80 (and second sources).

As previously noted, POP AF is not treated as a flag modifying instruction.

But with the NEC NMOS Z80 it is always:

```
   YF = A.5
   XF = A.3
```

And with the ST CMOS Z80 it is:

```
XF = A.3
if <previous instruction modified the flags> then
   YF = A.5
else
   YF = YF | A.5
endif
```