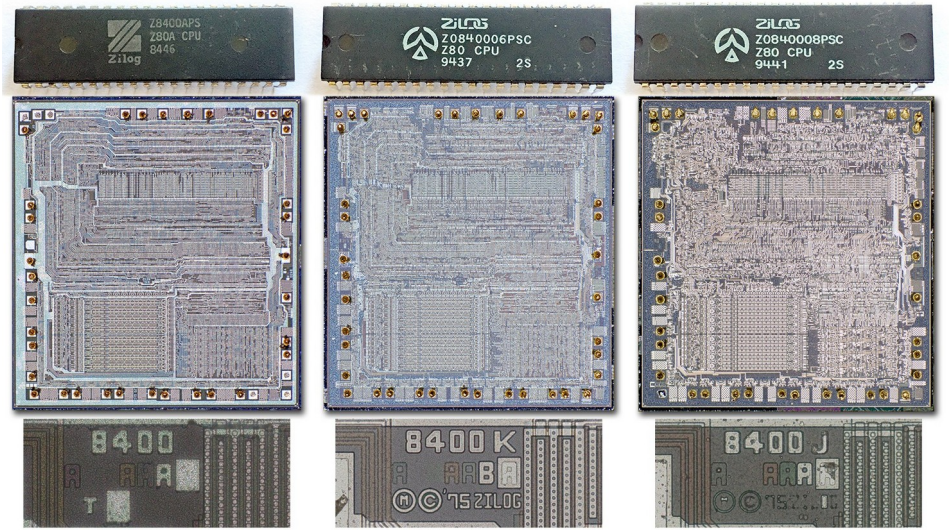
Z80: the last secrets



USER

Anisse Astier

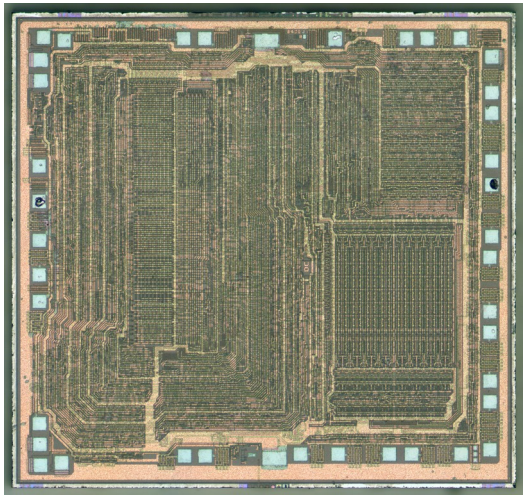
Linux Engineer

Discovered Z80 while writing an emulator, sharing a few discoveries

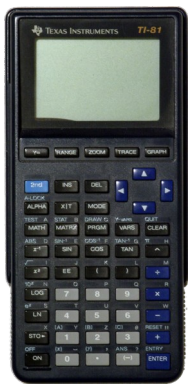
https://anisse.astier.eu

Z80 Chip

- First released in 1976 by Zilog
- Extension of the Intel 8080
- CISC design
- Peaked in 80s
- Zilog still exists and sells it

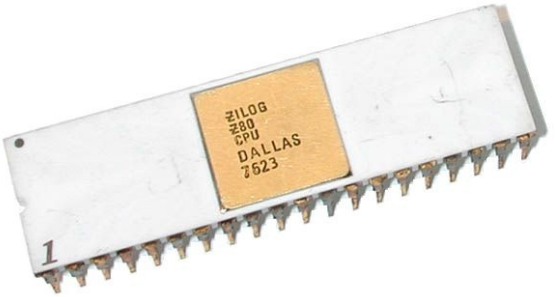


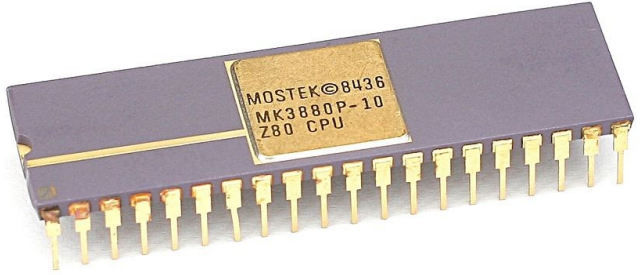
Z80 machines



Z80 Second sources

- Zilog
- Mostek
- Sharp
- NEC
- Toshiba
- Goldstar
- ST
- Soviet T34BM1





Z80 Registers

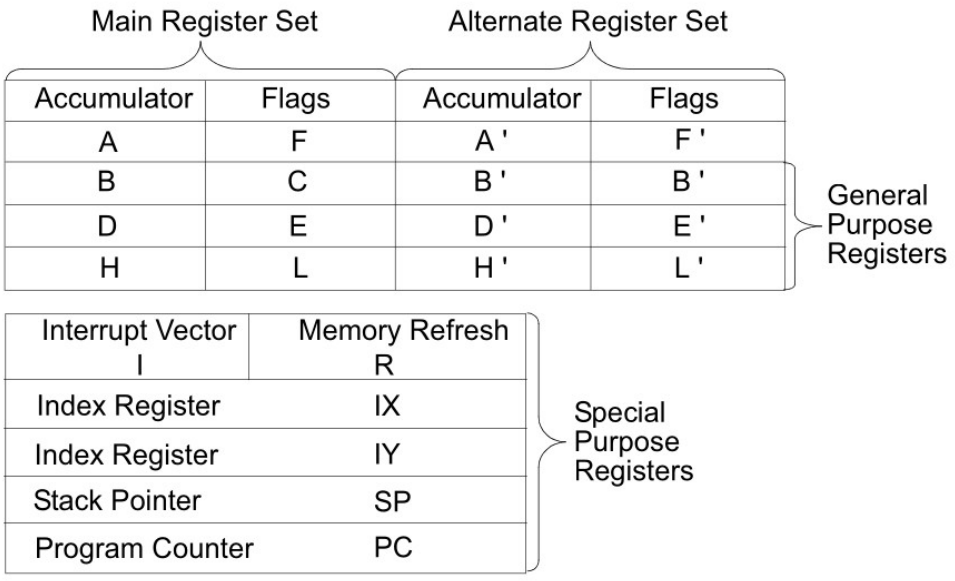


Figure 2. CPU Register Configuration

Z80 instructions

- Instructions often do memory load/store
- Most 8 bits, a few 16 bits operations
- Variable length from 1 to 4 bytes
- From 1 to 3 operands
- Memory-block instructions
- Specialized binary-coded decimal DAA

Flags X: Not Used ?

Bit	7	6	5	4	3	2	1	0
Position	S	Z	X	H	X	P/V	N	C

Table 22. Flag Definitions

Symbol	Field Name
C	Carry Flag
N	Add/Subtract
P/V	Parity/Overflow Flag
H	Half Carry Flag
Z	Zero Flag
S	Sign Flag
X	Not Used

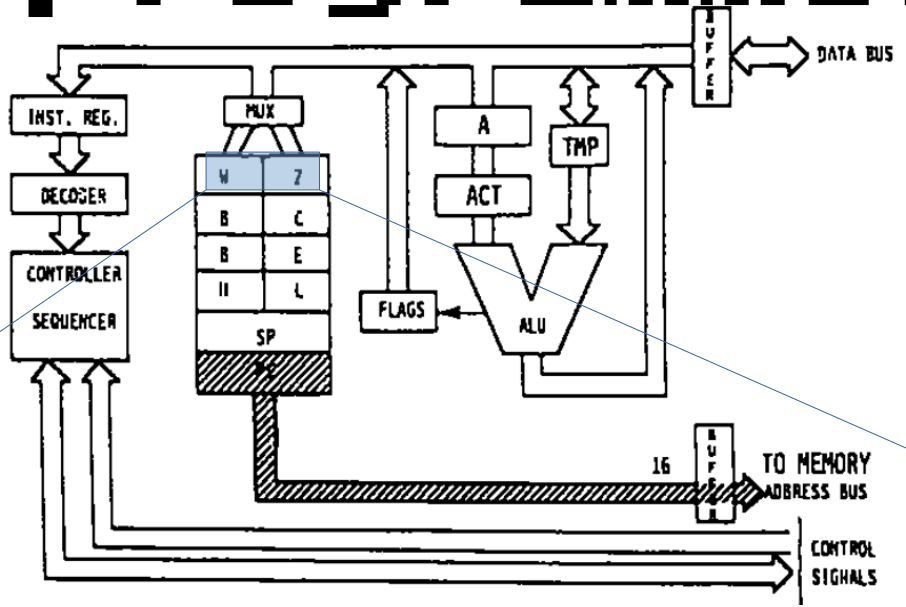
Microarchitectural leaks

- Long-known leak : 53 copy from result
- From Undocumented Z80 documented :

  « Things get more bizarre with the `BIT n,(HL)` *instruction. Again, except for YF and XF the flags are the same. YF and XF are copied from some sort of internal register. This register is related to 16 bit additions. Most instructions do not change this register. »*

  by Sean Young

Unavailable to the programmer



Two special registers are available to the control unit within the Z80 (but not to the programmer). They are "W" and "Z", and are shown in Figure 2.28.

Programming the Z80, 3rd edition, 1981, Rodnay Zaks

no$zx

- Proprietary ZX emulator from the nocash family

Internal MEMPTR Register
This is an internal Z80 register, modified by some instructions, and usually completely hidden to the user, except that Bit 11 and Bit 13 can be read out at a later time by BIT N,(HL) instructions.
The following list specifies the resulting content of the MEMPTR register caused by the respective instructions.

```
Content Instruction
A*100h  LD (xx),A               ;xx=BC,DE,nn
xx+1    LD A,(xx)               ;xx=BC,DE,nn
nn+1    LD (nn),rr; LD rr,(nn)  ;rr=BC,DE,HL,IX,IY
rr      EX (SP),rr              ;rr=HL,IX,IY (MEMPTR=new value of rr)
rr+1    ADD/ADC/SBC rr,xx       ;rr=HL,IX,IY (MEMPTR=old value of rr+1)
HL+1    RLD and RRD
dest    JP nn; CALL nn; JR nn   ;dest=nn
dest    JP f,nn; CALL f,nn      ;regardless of condition true/false
dest    RET; RETI; RETN         ;dest=value read from (sp)
dest    RET f; JR f,nn; DJNZ nn ;only if condition=true
00XX    RST n
adr+1   IN A,(n)                ;adr=A*100h+n, memptr=A*100h+n+1
bc+1    IN r,(BC); OUT (BC),r   ;adr=bc
ii+d    All instructions with operand (ii+d)
```

Also the following might or might not affect MEMPTR, not tested yet:

```
OUT (N),A and block commands LDXX, CPXX, INXX, OUTXX
and probably interrupts in IM 0, 1, 2
```

All other commands do not affect the MEMPTR register - this includes all instructions with operand (HL), all PUSH and POP instructions, not executed conditionals JR f,d, DJNZ d, RET f (ie. with with condition=false), and the JP HL/IX/IY jump instructions.

by Martin Korth

zexall

- Z80 Instruction exerciser initially written by Frank Cringle in 1994
- Works by running instructions, and feeding CPU state to a CRC
- To pass, emulators need to have same CRC as hardware

The Mystery

- WZ or MEMPTR is an internal register
- It leaks through the `BIT n, (HL)` instruction
- Only two bits at a time, in the two « unused » flags of the Flag register
- No emulator passes zexall in 2006

Cracking it

- In 2006 : a few uses already known, most aren’t
- Boo-boo and Vladimir Kladov sat down on zx.pk.ru to reverse it, with testing help from other members

Breakthrouh

*But **miraculously** in result of many experiments (based on the hypothesis that index addressing instructions initialize the MEMPTR always the same way) and also after the **deep meditations** under the results of these samples we have found that* `CPI` *instruction increments the MEMPTR by 1 whereas* `CPD` *instruction decrements it.*

From a 2-bit peek to a 14-bit leak

Algorithm :

```
bit53 = bit_leak()
count = 0
loop
    prev53 = bit53
    decrement_cpd()
    bit53 = bit_leak()
    if prev53 == 00 && bit53 == 11
        break
    end if
    count++
end loop
result = count
```

memptr_eng.txt

```
LD A,(addr)
MEMPTR = addr + 1

LD (addr),A
MEMPTR_low = (addr + 1) & #FF,  MEMPTR_hi = A
Note for *BM1: MEMPTR_low = (addr + 1) & #FF,  MEMPTR_hi = 0

LD A,(rp)  where rp -- BC or DE
MEMPTR = rp + 1

LD (rp),A  where rp -- BC or DE
MEMPTR_low = (rp + 1) & #FF,  MEMPTR_hi = A
Note for *BM1: MEMPTR_low = (rp + 1) & #FF,  MEMPTR_hi = 0

LD (addr), rp
LD rp,(addr)
MEMPTR = addr + 1

EX (SP),rp
MEMPTR = rp value after the operation

ADD/ADC/SBC rp1,rp2
MEMPTR = rp1_before_operation + 1

RLD/RRD
MEMPTR = HL + 1

JR/DJNZ/RET/RETI/RST (jumping to addr)
MEMPTR = addr

JP(except JP rp)/CALL addr (even in case of conditional call/jp,
independantly on condition satisfied or not)
MEMPTR = addr

IN A,(port)
MEMPTR = (A_before_operation << 8) + port + 1

IN A,(C)
MEMPTR = BC + 1

OUT (port),A
MEMPTR_low = (port + 1) & #FF,  MEMPTR_hi = A
Note for *BM1: MEMPTR_low = (port + 1) & #FF,  MEMPTR_hi = 0

OUT (C),A
MEMPTR = BC + 1

LDIR/LDDR
when BC == 1: MEMPTR is not changed
when BC <> 1: MEMPTR = PC + 1, where PC = instruction address

CPI
MEMPTR = MEMPTR + 1

CPD
MEMPTR = MEMPTR - 1

CPIR
when BC=1 or A=(HL): exactly as CPI
In other cases MEMPTR = PC + 1 on each step, where PC = instruction address.
Note* since at the last execution BC=1 or A=(HL), resulting MEMPTR = PC + 1 + 1
  (if there were not interrupts during the execution)

CPDR
when BC=1 or A=(HL): exactly as CPD
In other cases MEMPTR = PC + 1 on each step, where PC = instruction address.
Note* since at the last execution BC=1 or A=(HL), resulting MEMPTR = PC + 1 - 1
  (if there were not interrupts during the execution)

INI
MEMPTR = BC_before_decrementing_B + 1

IND
MEMPTR = BC_before_decrementing_B - 1

INIR
exactly as INI on each execution.
I.e. resulting MEMPTR = ((1 << 8) + C) + 1

INDR
exactly as IND on each execution.
I.e. resulting MEMPTR = ((1 << 8) + C) - 1

OUTI
MEMPTR = BC_after_decrementing_B + 1

OUTD
MEMPTR = BC_after_decrementing_B - 1

OTIR
exactly as OUTI on each execution. I.e. resulting MEMPTR = C + 1

OTDR
exactly as OUTD on each execution. I.e. resulting MEMPTR = C - 1

Any instruction with (INDEX+d):
MEMPTR = INDEX+d

Interrupt call to addr:
As usual CALL. I.e. MEMPTR = addr
```

Example: CPU ID

- Clones do have MEMPTR, and did reverse engineer it, even if by accident
- MEMPTR implementation varies in clones
- Can be used to identify CPU variant
- BM1 (Soviet clones) MEMPTR effect is documented

ZX Vega

- MEMPTR knowlege used to analyze ZX Vega, (2015 redesigned ZX Spectrum) as a blackbox
- Was found by Tautology to likely use an old version of FUSE emulator before it implemented MEMPTR : https://tautology.org.uk/blog/2018/03/20/reversing-the-zx-vega-the-emulator/
- Probably GPL violations

The end ?

- Most emulators catchup and implement it
- Everything is reliable and accurate
- Success ?

Another one

Patrik Rak

November 2012 edited November 2012

Well, having finally nailed the MEMPTR few years ago, just because of BIT n,(HL), it now seems that there is yet another internal register which actually affects the outcome of (at least) SCF/CCF. That's quite disheartening, as it means that no Z80 emulation out there is actually accurate, again.

SCF / CCF

- Set Carry Flag / Change Carry Flag
- Remember 53 copy ?
- 53 copy is not systematic
- It might be an OR instead, depending on some internal state

Q: 1-bit state

- Supposed to be a temporary zone where F is assembled
- CCF and SCF do copy only if F changed previously, otherwise OR with Q (previous F)
- Only need 1-bit to emulate: F-changed

z80test

- Patrik’s testsuite is the gold standard
- Supports both MEMPTR and Q
- Found that Q is only on Zilog’s version, not NEC’s

My emulator

- gears, written in Rust
- Passes z80test, zexall and fusetests
- Available at https://github.com/anisse/gears

Last secrets?

- Probably not, unfortunately
- Clones behavior is not 100% known
- Lots of undocumented tribal knowledge lost to the past

Questions

References

- zexall: https://mdfs.net/Software/Z80/Exerciser/
- z80test: https://github.com/raxoft/z80test
- FUSE emulator: http://fuse-emulator.sourceforge.net/
- gears emulator: https://github.com/anisse/gears
- iz80 emulator: https://github.com/ivanizag/iz80
- Zilog Z80 CPU manual: http://www.zilog.com/docs/z80/um0080.pdf
- Undocumented Z80 documented: http://www.myquest.nl/z80undocumented/
- no$zx docs on web archive: https://web.archive.org/web/20020614003602/http://www.work.de/nocash/zxdocs.htm
- memptr_eng.txt: https://zx-pk.ru/attachment.php?attachmentid=3002&d=1143926516
- MAME's Q pre-discovery: https://mametesters.org/view.php?id=2701
- Patrik Rak's world of spectrum z80 Q discovery: https://worldofspectrum.org/forums/discussion/41704/scf-ccf-flags-new-discovery

Citations

- Title image by Antoine Bercovici https://twitter.com/Siliconinsid/status/1446472128757370884
- Wikipedia images :
  - By ZeptoBars - http://zeptobars.ru/en/read/Zilog-Z80-Z0840004PSC, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=33458723
  - By I, Coyau, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1074761
  - By Piotr433 - This file was derived from: Sharp PC-E220.jpg:, CC0, https://commons.wikimedia.org/w/index.php?curid=38764661
  - By Bill Bertram - Own work, CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=170050
  - By Bilby - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=10955492
  - By Evan-Amos - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=14249084
  - By Evan-Amos - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=13317134
  - By The original uploader was Damicatz at English Wikipedia. - Transferred from en.wikipedia to Commons., CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=1830432
  - By Konstantin Lanzet - CPU collectionCamera: Canon EOS 400D, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=7446937
  - By Konstantin Lanzet - CPU collection, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=8041331
  - By MarcoTangerino - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=66413906
- Font: Press Start 2P by Cody "CodeMan38" Boisclair in SIL Open Font License
- Sonic logo from Master System version by Sega