Computing the last T state

One of the key factors in the timing and state of the Z80 is determining when to end an M cycle and start a new M cycle. This is done through a circuit that computes LAST_T_STATE/. This wire goes low during the last T state of a M cycle, and indicates that a new M cycle should start.

Because the lengths of M cycles are variable between instructions, the circuit to compute LAST_T_STATE/ is fairly complex. In addition, some conditional instructions have M cycles of variable length, as well as variable numbers of M cycles.

M1
For M1, the cycle is usually 4 states long, which is the minimum length of the cycle.
A few instructions have an M1 cycle that is 5 states long because they need to do additional computation: DJNZ, PUSH rr, RET, RST, LD I, A (and variants), INx/OUTx(r).
A few instructions have a 6 state long M1 cycle: LD SP, rr, INC rr, DEC rr

M2
If an instruction has M2, it is almost always 3 states long.
The exceptions are IN(I/D)(R) which add a wait state before T3.

M3
If an instruction has M3, it is almost always 3 states long.
OUT(I/D)(R) go to T3, but add a wait state before T3.
CALL goes to T4 if the transfer is taken (either because the condition is satisfied, or it is unconditional).
EX (SP), rr; RRD; RLD; goes to T4.
LD(I/D)(R) and CP(I/D)(R) go to T5.
JR and DJNZ to T5 if the jump is taken. (Otherwise, they don't have a M3 cycle.)
Operations using IX or IY with displacement indexing go to T5.

M4

If an instruction has M4, it is almost always 3 states long.
IN A, (*) and OUT (*), A add a wait state before T3.
IN r, (C), OUT (C), r, also add a wait state before T3.
The 16-bit ADD rr, rr, ADC hl, rr, SBC hl, rr, INC (hl), DEC (hl) instructions go to T4.
The BIT instructions accessing memory through (HL), e.g. RRC (HL), BIT 1, (HL), SET 2, (HL), RES 3, (HL) go to T4.
INC (IX+*), DEC (IX+*) go to T4.
Indexed BITS instructions go to T4?
LD(I/D)R, CP(I/D)R, IN(I/D)R, OT(I/D)R go to T5.

M5
If an instruction has M5, it is almost always 3 states long.
The exceptional instructions are EX (SP), rr go to T5.

The circuitry to implement LAST_T_STATE is largely just an implementation of the above special cases through combinatorial logic.

The logic to end a cycle in M1T4 or T5

Gate 296 ends the M1 cycle at T4, unless the instruction is an exception, in which case the cycle will continue. An exception is indicated by PLA outputs 4 (LD X,A/A,X), 5 (LD SP, HL), 9 (INC/DEC rr), 16 (PUSH rr), 20 (OUTX(R)), 21 (INX(R)), 26 (DJNZ), 45 (RET cc), or an interrupt/RST).

Gate 297 ends every M cycle at T5 unless there is an exception (or it has already ended). The exceptions are triggered by PLA 5 (LD SP, HL) and PLA 9 (INC/DEC rr), which go to T6 in M1. (Note that the PLA is unaffected by the IX/IY prefix, so LD SP, IX, and LD SP, IY are also covered by this.)

Interestingly, this gate had a ion implantation trap (discussed in the trap chapter), causing gate 297 to only function for M5, messing up the timing for some instructions.

Next, gate 310 has three inputs that will activate LAST_T_STATE/, for T3, T4, and T6. Each input will cause LAST_T_STATE/ to be activated at the associated T state, so each input controls which state is the ending state.

The first input causes T3 to be the last state if the output from gate 314 is low. The logic to generate this signal is complex and will be described below.

The second input causes T4 to be the last state if the output from gate 427 is high (and the cycle hasn't ended previously). It also has complex logic.

The third input is very simple: every M cycle will end with T6 (if it hasn't already ended).

The logic to end a cycle in T3

The logic to make cycles end in T3 is broken down by M cycle.

The first path is always triggered in M2, so every M2 cycle ends with T3. (Any wait states are inserted before T3, so T3 is the last state even if there were more than 3 states in the cycle.)

The M3 path is complex, since there many instructions don't end M3 with T3. The key inputs to the computation:

g288: This combination of PLA outputs will force the cycle to end with T3. It is active for pla7 (LD rr, **), pla20 (OUTX/OTXR), pla21 (INX(R)), pla29 (JP **), pla30 (LD rr,**), pla31 (LD I,A, etc), pla38 (ld (**),a/a,(**)), pla43 (jp cc,**), and interrupts. In other words, instructions with the “default” end of T3 are explicitly indicated here.

g385: This picks up CALLs, either conditional or unconditional. This is combined with the condition taken line 245, so untaken calls are terminated at M3T3 instead of M3T4.

For some reason, g385 is surprisingly complex, triggering on more than just CALLs, but the other operations aren't conditional, so they end at T3. This may be an optimization to simplify g288 slightly.

The M4 path is also complex. It will trigger, ending the cycle at T3 unless

one of its inputs is 1, letting the instruction continue longer:
g316: this is triggered by pla18 (ldi/ldir/ldd/lddr), pla19 (cpi/cpir/cpd/cpdr), pla20 (outx/otxr), and pla21 (inx/inxr). (These instructions end at T5, except for the non-repeating ones, which end at M3, so this circuit doesn't affect them.)
g386: This is triggered by pla68 (16-bit ADC/SBC) and pla69 (ADD hl,rr). These instructions go to T4.
g273: This is a complex signal, triggering for several different instruction types on different M cycles. Its effect here is to keep INIR/OTIR/INDR/OTDR from terminating at T3 - they will end at T5.

The M5 path is simply triggered by anything except PLA 10 - EX (SP), hl. Thus, every M5 cycle will end with T3 except for these instructions. The exceptional instructions will have M5 end with T5 due to gate 297 described earlier.

The logic to end a cycle in T4

Finally, the logic to end an instruction in T4 is handled by gate 427. Instructions will end in T4 (if not ended in T3) unless either input is 1 for an exception:
g316: This is triggered by LD/CP/IN/OTR. LDI/CPI/LDD/CPD will have M3 keep going to T5. The -R versions will have M3 and M4 both go to T5. Note that M1 and M2 are not affected by this gate because M1 ends in T4 through a different circuit, and M2 ends at T3, so this has no effect.

g1758: Except as above, this forces M4 to end in T4. Also M3 is forced to end in T4 if g385 is 1. This affects RRD, RLD, CALL (taken), and EX (SP), HL. Note that g385 was used to end M3 in T3 for a non-taken conditional.

LAST_T_STATE/ is also activated on a reset, to bring the current instruction to an end.

The number of states in instructions varies based on the instruction and how much internal action needs to take place. The number of states also sometimes varies for conditionals. As a result, the logic to implement the LAST_T_STATE/ control is rather complex. It also lacks an overall guiding structure, being largely a matter of exceptions on top of exceptions to minimize the size of the random logic.

In comparison, the 8085 uses a PLA to determine the number of states in different instructions. This results in a conceptually cleaner design.

The 6502 uses random logic to determine the end of instructions. Because the 6502 doesn't have M cycles, but just T states, the logic is considerably simpler than the Z80. One interesting characteristic of the 6502 is some illegal opcodes fail to trigger the new-instruction logic. Since the T states are handled by a shift register, the last T state can shift out of the shift register, leaving the chip in a state where nothing happens. In addition, interrupts and NMI don't take place until the end of an instruction, so they are ignored forever too. For details, see “How MOS 6502 Illegal Opcodes really work” http://www.pagetable.com/?p=39