# Z80 timing: M cycles and T states

The timing of every instruction in the Z80 is organized around M cycles and T states. While M cycles and T states may seem like pointless numerology, they are actually very important to understanding how the chip works.

An M cycle is a memory cycle and corresponds to one memory access (either a read or a write). An instruction requires at least one M cycle to read the instruction opcode from memory. This M cycle is called M1. The Z80 indicates the M1 cycle through an output pin, allowing external parts of the system to know when opcodes are fetched.

If an instruction requires additional memory accesses, another M cycle is used for each access: M2, M3, etc. Memory accesses may be required to read additional instruction bytes, for instance an address or an immediate data value - each byte adds another M cycle. The instruction may also read data from memory, write data to memory, or both. Each of these memory accesses also uses another M cycle.

Each M cycle is made up in turn of multiple T states, each one corresponding to one system clock cycle. An M cycle can have 3 to 6 T states, depending on the instruction and how much computation needs to be done. The processor goes through T states in order: T1, T2, T3, etc. Putting the M cycles and T states together, an instruction will go through a sequence such as M1T1, M1T2, M1T3, M1T4, M2T1, M2T2, M2T3. As will be seen later, these states are critical to the processing of an instruction. Each step of the instruction is tied to a particular state such as M2T1.

Inside the Z80, each T state corresponds to one clock cycle, starting with the clock low and ending with the clock high. As a result, M cycles start with the lock low and end (several cycles later) with the click high.

Externally, the timings are slightly different. Signals are delayed by half a clock cycle, so each T state and M cycle starts with the clock high and ends with the clock low. Since this document is describing the chip internally, I will use the internal timings. But if you read documentation, in particular

timing diagrams, be aware that the external timings are different.

There are a few special cases with M cycles. First, prefixes are treated (for the most part) as separate instructions with their own M1 cycle. That is, a prefixed instruction has multiple M1 cycles. As a consequence, the R register is incremented multiple times. As will be explained in the interrupt chapter, interrupts are not permitted after a prefix, so prefixes are not treated as separate instructions in this context.

Second, some instructions don't step sequentially through the M1, M2, M3, … sequence, but have M cycles out of order. This is entirely invisible from outside the chip, since the Z80 still goes through the expected number of cycles, but internally the sequencing is different. This is probably done to simplify the logic, but more investigation is needed.

By counting the total number of T states in an instruction, you can determine exactly how long a particular Z80 instruction will take. Modern chips don't have this sort of deterministic timing for a variety of reasons, from caches and prefetching to internal instruction reordering and branch prediction.

A Z80 instruction can take longer than expected in a few cases. First, an external device can request the bus with the BUSREQ pin. Additional T states (Tx) will be inserted while the external device is controlling the bus, which can be arbitrarily long.

Second, if the memory in the system is too slow to respond during the standard access time, it can insert wait states through the WAIT pin.

Finally, when the Z80 receives an interrupt, it has a special M1 cycle but inserts two wait states so the peripherals have time to deal with the interrupt.

Implementing the timing logic

A large part of the Z80 circuitry generates the M cycles and T states. Each M cycle and each T state has a control line associated with it. These control lines are heavily used by the instruction decode logic to perform the right

action at the right time.

The M cycle and T state control signals are generated in the upper part of the chip.

A key control signal is LAST_T/, which is 0 during the last T state of an M cycle. This control signal has two important effects. It resets all the T latches and loads the T1 latch to return the T state to T1. It also advances the M cycle. LAST_T/ is blocked by a bus request, allowing the external device to control the bus before the T1 state starts. The signal W174 goes to 0 just before the start of T1. It sets T1 and resets all the other T latches, causing the state to be T1.

The T states are controlled by a sequence of latches forming a shift register. The first latch holds T1. As described above, this latch is loaded with the value of W174, causing the state to move to T1.

The output of the first latch is T1, and is fed into the second latch, whose output is T2.

One complication is that T2 can be followed by wait states. The T3 latch is fed from T2, but the output from the T3 latch is blocked if the WAIT pin is activated. Thus, the T3 state is held in the latch but is not visible to the rest of the chip. Meanwhile, the T3 latch is re-enabled during a wait state. Thus, when the wait state ends, the T3 signal is (finally) activated.

TBD: where do the wait states come from for an interrupt, or I/O?

T3 is the input to T4's latch. T4 is the input to T5's latch. T5 is the input into T6's latch. Thus, the T state normally advances by one each clock as the active state is passed from latch to latch.

The LAST_T/ signal is generated from a bunch of gates that combine a bunch of conditions: T6 or (T3 and an instruction and M cycle where T3 is

the last cycle) or (T4 and an instruction and M cycle where T4 is the last cycle) or (M1T4 except for an instruction that has more T states in M1) or (T5 except for an instruction that goes to T6). In other words, the default is to keep going after T3 or T4 and end at M1T4, or T5, or T6, but some instructions are exceptions to these. (Except T6 is always followed by T1.)

The M states (with a few exceptions) have the following actions:
M1: read op code
M2: read instruction byte
M3: read instruction byte
M4: read/write memory
M5: read/write memory

An instruction can end after any M cycle. It can also skip M2 and M3 or skip M3 if the instruction is 2 or 3 bytes long. The following state diagram shows how this works:

The M states are controlled by W207, which goes high at the beginning of each new M state.

Another important control signal is W171 - if it is high at the beginning of an M state, the M state resets to M1. This signal is generated from several inputs. The first input indicates an instruction that is one M cycle long, based on the PLA decoding. The second input indicates that an instruction with multiple M cycles is in the last cycle: M5 (for any instruction), M4 for an instruction ending in M4, etc. The third input indicates a conditional instruction (branch, return, etc) that is not being taken, and terminates the instruction early.

The M states are controlled by a sequence of latches which are clocked by the W207 signal. M1's latch is loaded with W171. M2's latch is loaded with M1, so M2 follows M1. However, M2's latch is cleared by W171, which indicates the next cycle is M1. M2's latch is also cleared by W148, which indicates the

next cycle is M4 out of order.

The instructions that go from M1 to M4 include LD (rr), A; ADD rr, rr, LD A, (rr), inc (rr), push rr, pop rr, ret, rst. in. While many of these operations perform memory accesses indexed by a register, not all do. It's unclear what they have in common or why they use M4 instead of M2. It looks like M2 and M3 are mostly used for reading additional instruction bytes, while M4 and M5 are used for memory accesses. Although RRD, LDI, CPI are exceptions that access memory in M2.

M2 is fed into M3's latch, similarly cleared by W171 and W148, allowing a jump from M2 to M4 for instructions such as LD D,*, which read the immediate instruction value in M2, and then jump to M4 to store the value in memory.

M3 is fed into M4's latch, but in this case W148 sets the latch rather than clearing it. As with the other latches W171 clears the latch. Thus, the M4 state follows M3, or follows any state when W148 is set, unless the next state is M1.

M4 is fed into M5's latch. This latch, like the others is cleared by W171. Thus, M5 always follows M4, unless the next state is M1.

M5 triggers the new M1 signal W171, so M5 is always followed by M1. Some sources (such as Wikipedia) show an M6 state for some prefixed instructions, but that doesn't exist. Instead, prefixed instructions have multiple M1 states.

The result of all this logic is that the chip moves through the appropriate M and T states.

The M and T control lines flow down the left side of the chip and underneath the PLA to the instruction decode logic. As will be explained in more detail in the instruction decode chapter, the M and T control lines are combined with the PLA outputs to generate control signals indicating that something should happen for a particular instruction in a particular cycle. This logic is

what controls the specific activities of the processor and makes things happen in sequence. Note that the Z80 isn't microcoded - this logic is all implemented by gates.