ALU controls

The ALU has a fairly large number of control lines that control the functions of the ALU as well as the movement of data in and out of the ALU.

The overall architecture of the ALU is it is 4 bits wide, which is kind of surprising, since most 8-bit processors have an 8 bit ALU. As a result, the ALU is used twice for operations, first to process the low-order bits, and then to process the high-order bits.

Even though the ALU is 4 bits wide, the ALU's bus is 8 bits wide. The ALU has two sets of 8-bit latches for the two input arguments - an 8-bit argument is loaded into the ALU and latched into one of the latches. To perform an operation, first the lower 4 bits are fed into the ALU's operation unit to generate a 4-bit result. This result is stored in the 4-bit output latch. This computation takes two half-cycles.

Then half a cycle is used to update the H carry flag. Not only is this value used as a flag, but it is also used to provide a carry from the lower 4 bit operation to the higher 4 bit operation.

Next, two half-cycles are used to perform a 4-bit operation on the upper 4 bits of the latched arguments.

The lower 4 bits of the result are fed onto the ALU bus from the 4-bit output latch, while the upper 4 bits of the result are fed onto the ALU bus “live”, without being latched.

One interesting design feature of the Z80's ALU is that a shift left or shift right happens when the data byte is loaded into the ALU. This is in contrast to other contemporary CPUs, which typically perform a shift-right as an ALU operation (which passes each bit “backwards”, and perform a shift-left by adding the value to itself.

To accomplish the shift, there are three control lines that control data entry to the ALU. Control line “A” passes the unshifted data from the register bus to

the ALU bus. Control line “B” passes the data from the register bus shifted right to the ALU bus. Control line “C” passes the data from the register bus shifted left to the ALU bus. The circuits to do these shifted loads are clearly visible in the die photo as “fingers” from the ALU reaching to the appropriate register bus lines.

The shifts have complex boundary conditions - what happens to bits that are shifted off one edge or enter from the other edge. These circuits will be discussed later.

To provide a framework for this discussion, let's look at how a simple ALU operation such as ADD A, B works.
M1T4l: AF goes on the register bus. A is latched into ALU op1 and op2 latches. The F register is stored in the flag latches.
M1T1l: B goes on the register bus. B is latched into ALU op2. The sum of the lower 4 bits is computed and is latched into the ALU's output latch.
M1T1h: The sum computation continues and is latched.
M1T2l: The ALU's carry is latched into the half carry flag.
M1T2h: The sum of the higher 4 bits is computed. The 8-bit result goes onto the ALU bus and the register bus. (Note that the lower 4 output bits come from the output latch, which holds the earlier result. The high 4 output bits come from the operation unit.)
M1T3l: The sum continues to be computed and the result continues going to the register bus. The result is stored in the accumulator. The carry flag latch is updated.
M1T3h: The flag latches are written to the register bus.
M1T4l: The flag latches continue being written to the register bus. They are stored in the F register.

Note that the ADD operation doesn't start until M1T4l and doesn't finish until M1T4l of the next instruction. This shows the fetch-execute overlap in the Z80.

Some ALU operations have been omitted from the above. The ALU constantly has control lines changing, even if the ALU is not actually needed. This is because there are many states during which the control line values

don't matter. For these states, the control line ends up doing whatever makes the control logic simpler.

One thing to note is the movement of the flags between the register file and the flag latches is optimized depending on the sequence of instructions. The movement of flags depends on if the previous operation used the flags or not. This can be seen above where the first M1T4l copies the accumulator and flags out of the register file, while the second M1T4l copies flags back into the register file.

Looking at the circuit for ALU “A”, the control of this line is extremely complex since almost every instruction uses it.

Looking at gate 947, both inputs must be 0 to activate A.
gate 626 deactivates A in several cases:
M1T3 for a BITS instruction
M4T1 for a BITS instruction using indexed addressing
M1T5 for an im0 interrupt
M1T4 and M4T3 for a rotate/shift instruction

gate 661 activates A in several cases (subject to the above deactivations):
M1T4 and M4T3 for an arithmetic instruction, LD, IN, rotate/shift, DAA
g257:
M1T5 for a im0 interrupt or RST
M1T3 or M2T3 for every instruction
M5T1 for 16-bit add or sub
M4T1 for 16-bit add/sub or an indexed addressing op
gate 588:
M2T2 for 273 or indexed address operation. (This is when the indexed address operation loads the low byte of the index register into the ALU before adding the displacement.)
658: M1T3 for anything, or M4T2 for shift/rotate or indexed address
M3T3 and M4T3 for the 273 cluster of instructions (which is hard to characterize)

M1T5 for im2 interrupt
M4T4 for indexed addressing op. An indexed addressing op is an instruction with an IX/IY prefix that normally accesses (HL) but instead accesses (IX+*) or (IY+*).
M1T4 for indexed HL op or the 273 cluster

Let's take a look at the shift/rotate timing for RLC (HL) (a BITS-prefixed instruction, so it has two M1 cycles).
In the first M1, the BITS prefix is loaded.
In the second M1:
M1T4l: flags latch loaded from F register.
M4T1l: HL loaded into incrementer latch.
M4T1h: incrementer latch loaded into address latch, to load (HL)
M4T3h: byte read loaded onto dbus
M4T4l: dbus loaded into op2, shifted left. 0 loaded into op1. Sum to output latch.
M4T4h: sum to output latch
M5T1l: half carry updated
M5T1h: upper-4 bit sum to regbus, dbus. incrementer latch to address latch (i.e. HL for write back)
M5T2l: upper-4 bit sum to regbus, dbus, latched to dpins for write
M5T3l: write
M5T3h: write

Note that in M4T4l, the shifted value is loaded into the ALU, so the regular ALU load needs to be blocked. (This gets set up in M4T3 and then delayed half a cycle.)

Finally, let's look at an indexed address op, such as LD (IX+*),B.
This will be M1 (IX prefix), M1 (opcode), M2 (offset), M3 (no memory access, just calculation), M4 (store to memory).

This is quite a busy instruction. The bulk of the work is doing the 16-bit addition of IX and the displacement. This requires 4 passes through the ALU.

M2T3l: IX (l) goes over the regbus and is stored in ALU's op1 latch

M2T3h: The fetched offset goes to the dbus
M2T4l: The fetched offset is loaded into op2
M2T4h: The lower sum goes into the output latch
M3T1l: The lower sum goes into the output latch
M3T2l: The half carry is updated
M3T2h: The sum goes to the regbus
M3T3l: The sum goes to the regbus and is stored in Z. The carry flag is updated.
M3T4l: IX (h) is stored in op1. 0 is stored in op2. The lower sum is computed and latched in the output latch.
M3T4h: The lower sum is computed and latched in the output latch.
M3T5l: The half carry is updated.
M3T5h: The upper sum is computed
M4T1l: The upper sum is computed and written to W. WZ is copied to the incrementer latch.
M4T1h: The incrementer latch is copied to the address latch for the write
M4T2l: B is copied to the data pin latch.
M4T3l: write
M4T3h: write

The A line is active in
M1T3 (prefix)
M1T3 (opcode)
M1T4
M2T2: IX (l) is loaded into ALU op1
M2T3: Fetched offset is loaded into ALU op2
M3T3: IX (h) is loaded into ALU op1
M4T1
M4T2
M4T3
M4T4
(The line as used is active half a clock after this.)
Note that most of the A line activations aren't actually necessary.