The Z80's PLA

Once an instruction is loaded into the instruction register, the first step of decoding the instruction is done by the PLA. This chapter discusses how a PLA works, how the PLA in the Z80 is structured, and what the outputs of the PLA represent.

A PLA is a Programmable Logic Array, which is a structured way of implementing combinatorial logic through a grid of transistors. A PLA usually consists of one grid, called the AND plane, which computes AND operations on a selected set of inputs. The output of the AND plan is fed into the OR plane, which is a similar grid which ORs together various inputs. The final output is a “sum of products”. For example, if inputs are A, B, and C, some AND terms could be AB, B, or C. Outputs from the OR plane could combine them into outputs such as AB+BC or AB+C or B+C.

A PLA has several advantages over “random” logic gates. (In the context of circuits, “random” logic means logic built out of gates as needed, rather than following an overall structure. The logic, of course, is not truly random.) Because of the compact structure, of a PLA it can often implement multiple outputs in less space than random logic. In addition, it is more flexible; the logic functions can be changed by simply adding or removing transistors, rather than redesigning the logic circuits, which could require the layout of the chip to be redone.

The PLA in the Z80

The first level of the Z80's PLA is implemented as a collection of multi-input NOR gates, one for each output. Since this is the AND plane, it may be puzzling that NOR gates are used. This can be explained with De Morgan's laws, which can be used to convert between OR and AND logic: “The negation of a conjunction is the disjunction of the negations.” That is, an AND gate is the same as a NOR gate with all the inputs inverted. (An AND gate requires all inputs to be 1 to have a 1 output. A NOR gate requires all inputs to be 0 to have a 1 output. So if you complement the inputs, the two circuits are the same.)

For example, consider an AND expression such as $A.B.\overline{C}$. This is the same logically as $\sim(\overline{A}+\overline{B}+C)$.

The reason to use NOR gates instead of AND gates is that NOR gates are easy to implement with MOS transistors, while AND gates require an extra inverter on the output.

Looking at the hardware implementation, the first level of the Z80's PLA consists of inputs connected to horizontal polysilicon lines. The outputs consist of vertical metal lines. At each crossing point, a transistor can be added: the source is the ground line (one running horizontally for each pair of inputs), the gate is the input line, and the drain is the output line. If the corresponding input is 1, the transistor will pull the output low. If all the inputs connected to transistors are 0, the output will not be pulled low, and a resistor will pull the output high. (The resistor is actually a depletion load transistor as explained earlier.) Thus, each column implements a NOR gate in a very compact form.

In the Z80, the inputs are the 8 instruction bits, along with the 8 complements of these instruction bits. Thus, an output column of the Z80 can match any instruction bit pattern made up of 0's, 1's, or X's (don't care).

For instance, a PLA output line can match an exact instruction such as `01110110`. (This is the HALT instruction.) In this case, the inputs to the PLA are bit7, ~bit 6, ~bit5, ~bit4, bit3, ~bit2, ~bit1, and bit0. If any of these bits are high, the output will be low. It can be seen that the output will be high only for the desired instruction.

A PLA output line can also match a set of instructions, for example all instructions matching the pattern `00xxx011`.(A 16-bit INC rr / DEC rr instruction.) The value `x` means that the bit value doesn't matter, and can be either 0 or 1. For this pattern, the PLA column has transistors for inputs are bit7, bit6, bit2, ~bit1, ~bit0. Since there are no transistors on bits 3, 4, and 5, the values of these bits don't affect the outputs. Thus, by leaving out transistors, the PLA can support “don't care” inputs.

In this way, the PLA can match any bit pattern expressed by 0's, 1's, and X's.

One of the important features of an instruction set design is making sure that instructions can be easily decoded by a PLA. To achieve this, related instructions need to fit into bytes that can be matched by a PLA pattern. As you can see from looking at the Z80 opcode table, most of the instructions are organized in such a way. In particular, the load and match instructions form nice blocks. Only a few instructions, such as the exchange instructions, are wedged into random spots.

Note that the AND plane cannot support more complex patterns, for instance matching either `11101011` or `11011001`. (These are the exchange instructions EX DE, HL and EXX) To support these more complex outputs, the OR plane is used. The OR plane is also used to combine blocks of instructions into more general blocks of instructions.

The OR plane is organized with the inputs running vertically as polysilicon lines (connected to the outputs from the AND plane), and the outputs running horizontally as metal lines. As before, transistors connected to ground are put at the desired crossings, with pull-up resistors on the outputs. Strictly speaking, this plane is a NOR plane, not an OR plane, because it uses NOR gates. Many of the outputs then go through inverters to generate the desired OR logic.

For instance, one output of the OR plane computes pla24 NOR pla42. This output is then inverted to generate pla24 OR pla42. To see what this is doing, the pla24 output matches 11001101, the CALL instruction. pla42 matches 11xxx100, a conditional CALL CC instruction. Thus, pla24 OR pla42 matches a call, either conditional or unconditional. As you can see, a single pattern won't match all of these call instructions, so the OR plane is required.

The Z80's PLA is highly optimized rather than using the strict grid approach. It uses the technique of column segmentation and reorganization, as described in “The Architecture of Microprocessors”. Some inputs are only used by some columns. By putting these columns together, those inputs can be provided only for those particular column segments. This reduces the total size of the PLA, but makes it less general.

The Z80's PLA uses several other optimization techniques. It uses lateral inputs and outputs, so the outputs can be closer to where they are used. This doesn't reduce the size of the PLA itself, but makes routing much easier. For instance, some PLA outputs exit the PLA at the top, where they can be directly routed to the timing circuits. Some PLA inputs enter at the top, specifically the ones used for segmented columns.

Another technique used in the AND plane is to put two unrelated circuits in the same column. For instance, a PLA output may be routed out the top, while an unchanged input is routed out the bottom. This reduces the overall PLA width.

The PLA also has some signal lines that are routed vertically through the PLA even though they are not part of the PLA. This avoids requiring signal lines to be routed around the PLA.

The OR plane is very optimized. First, the inputs are segmented so two outputs can be computed in the same row in many cases. This requires the inputs for the first operation to be on one side, and the inputs for the other to be on the other size. This can be accomplished by clever organization of the outputs from the AND stage. The advantage of this is that one row can take the place of two, making the PLA smaller.

Second, the OR plane has many outputs that exit on the left, many outputs that exit on the bottom, and many lines from the AND plane that pass through unchanged. As with the AND plane, this simplifies routing. If a signal is used on the left, it's more convenient to have it exit the PLA on the left, rather than exiting on the bottom and being routed around the PLA.

Finally, the OR plane uses several signals that don't come from the AND plane, but just enter as needed.

Because of its heavily-modified layout, the OR plane isn't as immediately recognizable as a PLA. Looking at its outputs, they are mostly ORs of outputs from the AND plane, so functionally it acts as an OR plane, even if the layout is not a strict grid.

The PLA has two input lines for each instruction bit - one for the bit and one for the complement - but it has several other inputs for various reasons. These are the segmented inputs, which are used by only part of the chip. These support prefix instructions and other special cases.

The first row has three separate inputs, which I call ab, c, and d. Input ab is low if the instruction has an IX/IY prefix. Inputs c and d will be discussed a bit later.

The second row has input e, which is low for a regular instruction (no EXTD or BITS prefix, although IX/IY is allowed). is low for a regular instruction (no prefix). Only a portion of the third row is used: input f is low if the CB prefix (BITS) has been encountered. The fourth input row, g is low for a ed-prefixed (EXTD) instruction.

These inputs are how prefixes get decoded. Most of the PLA patterns use one of these inputs along with the bit inputs. For example, the rotate instructions match the pattern `000xx111:e`. This is implemented by having transistors on inputs bit7, bit6, bit5, ~bit2, ~bit1, ~bit0, and e. The consequence is this PLA line will be triggered if this bit pattern appears without a prefix, but will not be triggered if there has been a prefix.

Similarly, `101xx000:g` matches this bit pattern only when line g is low. So this PLA will only be triggered when the instruction has a EXTD prefix - it corresponds to LDI/LDIR/LDD/LDDR.

Note that the e, f, and g prefixes are unaffected by an IX/IY prefix. This is why instructions are very similar with and without an indexing prefix - the decoding in the PLA is identical, and it is only later that the indexing happens.

Also note that a NOP (no operation) instruction doesn't trigger any PLA lines. This is why it has no effect - since it doesn't trigger any PLA lines, nothing happens, apart from the normal M1 fetch cycle. Many of the EXTD-prefixed bytes don't have instructions documented for them. Many of these instructions also don't trigger any PLA line so they are just NOPs. The role of the PLA in the actions of undocumented instructions will be discussed more later.

Another interesting thing is the pattern `101xx000:e` which matches the IX/IY prefix. Because the e line is there, this pattern will not match after an EXTD or BITS prefix. This is why an IX/IY prefix cannot be placed after an EXTD or BITS prefix - it won't match the PLA.

One interesting PLA output is pla54 matching xxxxxxxx:be. This matches any instruction that has an IX/IY prefix (line b low) and has no other prefix (line e low).

Another interesting output is pla40 matching 00110110:be. As before, this matches on only the IX/IY prefix, but now with an instruction pattern 0x36. This matches LD (IX+*), * or LD (IY+*), *. The reason this instruction gets singled out for special treatment is due to the handling of instructions with two bytes of data. Most of these instructions are unaffected by the IX/IY prefix. For instance, LD HL, (**) and LD IX, (**) both require two address bytes. However, 0x36 is LD (HL), *, which has one immediate byte. The corresponding prefixed instruction is LD (IX+*), * which requires one byte for displacement and one byte for the immediate value. To fetch two bytes in this case, the instruction must be decoded specially, which is the purpose of this PLA line.

Earlier, the PLA input line “c” was mentioned. This input line is PLA output pla95 fed back into the PLA. pla95 is triggered by the HALT instruction. The purpose of this line is to decode the LD (HL), r and LD r, (HL) instructions. As you can see from the instruction table, according to the pattern for these instructions, you'd expect 0x76 to decode to the pointless LD (HL), (HL), but instead it decodes as HALT. The reason is that pla59 matches 01110xxx:ce (LD (HL), r) and pla58 matches 011xxx110:ce. Since line c goes high for a HALT instruction, these PLA outputs are inhibited, and the instruction is not decoded as a LD instruction. To summarize, this PLA line “c” is the hardware reason that HALT appears where you'd expect to see a LD instruction.

The PLA line “d” was also mentioned earlier. This line is also feedback from the PLA output to the input: in this case it is set by pla64+pla65. These two PLA patterns are 11xxx110:e (add/sub/etc a, *) and 10xxxxxx:e (add/sub/etc a, r). That is, line d is triggered by one of the 8 types of arithmetic instructions on the accumulator. In turn, line d triggers decoding of the associated instruction type:
pla84: xx000xxx:d is ADD
pla80: xx001xxx:d is ADC
pla78: xx010xxx:d is SUB
pla79: xx011xxx:d is SBC
pla85: xx100xxx:d is AND
pla88: xx101xxx:d is XOR
pla86: xx110xxx:d is OR
pla76: xx111xxx:d is CP (compare)
Thus, these 8 PLA outputs indicate the 8 types of arithmetic instructions in the family.

Note that because immediate and register operations have different bit patterns, a single PLA pattern couldn't match against all the ADD instructions, for instance. Without using the special d control line, two PLA patterns would be required for each instruction type, doubling the number of patterns required.

Another special case in the PLA is output pla83. This matches 0101x111:g (LD A, I or LD A, R). However, there's an extra transistor at the top connected to the IFF2 interrupt flip flop. These instructions have the unusual property of loading the contents of IFF2 into the P/V flag. This is accomplished by having pla83 trigger for these instructions, only if the flip flop is clear (or set?). This PLA output goes to the P/V flag control circuit, overriding the normal overflow value stored in the flag. Unlike all the other PLA outputs, pla83 doesn't just indicate the result of instruction decoding, but depends on internal state. This seems like a bizarre hack, both in the way the PLA is used, and in the way the value is stored in the flag. It would be interesting to know the history behind these instructions.

The PLA can be compared to other processors. In the 6502, the PLA is different in many ways. First, there's only an AND plane; random logic takes the place of any OR plane operations.

The 6502 uses the PLA for instruction decoding and timing together: the 6 timing state lines go into the PLA, so outputs typically indicate a particular instruction at a particular time state. Since the timing in the 6502 is simpler (there aren't M states and instructions use fewer cycles), this is practical in the 6502. In comparison, the Z80 only does instruction decoding in the PLA and uses a large amount of random logic to add in the timing.

The 6502 feeds the top 6 bits into the PLA (uncomplemented and complemented), but the low 2 bits are decoded into three separate control lines. G1 is 0 if the bottom two bits are X1, G2 is 0 if the bottom two bits are 1X, and G3 is 0 if the bottom two bits are 00. The benefit of doing this is that only three PLA inputs are needed to process the bottom two bits, rather than four if each bit and its complement had a separate line as is typical. This makes the PLA one row smaller, shrinking the chip. The consequence, however, is that the 6502 can't use instructions ending in 11, as they trigger G1 and G2 and decode as a combination of both. For more details, see http://www.pagetable.com/?p=39 and http://www.llx.com/~nparker/a2/opcodes.html and http://visual6502.org/wiki/index.php?title=6507_Decode_ROM

The 8085, on the other hand, uses a much more complex PLA system with 7 components. The AND plane decodes the instruction using the 8 bits and their complements, along with two control lines denoting memory accesses. As in the Z80, 3 bits select the register, except 110 indicates a memory access indexed through the HL register. The 8085 checks for 110 (HL) as the source and destination, feeding these two signals into the initial decode PLA.

The outputs from the first 8085 plane indicate the instruction category, similar to the Z80's AND plane. These outputs go into a small instruction timing PLA, which defines the M cycles associated with each instruction. In comparison, the Z80 uses a large amount of random logic to control the movement from one M cycle to the next.

The 8085's instruction decode outputs also go to a decoding OR plane that computes instruction groups. This is similar to the OR plane in the Z80.

The output from the instruction groups goes to two separate PLAs. The first is the register timing PLA, which combines the instruction group information with timing information to determine if an action should take place, producing register timing outputs. Finally, these outputs go into a register control PLA that generates controls for individual register operations.

The second path for the instruction groups is on the other side of the instruction group PLA, and is the ALU timing PLA, which is similar to the register timing PLA but for ALU controls. The outputs from this go into the ALU control PLA that generates controls for individual ALU operations.

As can be seen, the 8085's PLA structure is very complex, reducing the amount of random logic in the 8085.

See http://www.pastraiser.com/