Yes, the 8086 wanted to be mechanically translatable from the 8080, but why not add the ability to indirect through AX, CX and DX?

devblogs.microsoft.com/oldnewthing/20220725-00

July 25, 2022

Raymond Chen

Some time ago, I noted that the 8086 was designed so that existing 8080 code could be machine-translated instruction by instruction into 8086. The 8086 `BX` register stood in for the `HL` register pair on the 8080, and it is also the only register that you could indirect through, mirroring the corresponding limitation on the 8080.

But that explains only part of the story. Yes, the 8086 had to let you indirect through `BX` so that 8080 instructions which operate on `M` (which was the pseudo-register that represented `[HL]`) could be translated into operations on `[BX]`. But that doesn't mean that the 8086 had to forbid indirection through the other registers. After all, the 8086 had plenty of other instructions that didn't exist on the 8080.

So you can't take away `BX`, but more is better, right? Why didn't the 8086 let you indirect through `AX`, `CX` or `DX`, as well as `BX`?

Basically, because there was no room.

The encoding of two-operand instructions on the 8086 went like this:

7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0
op						d	w		mod		reg			r/m		

The `op` determines the operation to be performed.

The `d` is the direction (reg to r/m or r/m to reg).[1]

The `w` indicates whether it is a byte operation or a word operation.

The `mod` is the *mode* and describes how the `r/m` is to be interpreted.

The `reg` is the first operand, always a register (although the `d` bit can reverse the first and second operands).

The interesting thing here is the `mod` + `r/m` combination, since those capture the possible memory operands.

r/m	mode+w				
	00+*	01+*	10+*	11+0	11+1
000	* PTR [BX+SI]	* PTR [BX+SI+imm8]	* PTR [BX+SI+imm16]	AL	AX
001	* PTR [BX+DI]	* PTR [BX+DI+imm8]	* PTR [BX+DI+imm16]	CL	CX
010	* PTR [BP+SI]	* PTR [BP+SI+imm8]	* PTR [BP+SI+imm16]	DL	DX
011	* PTR [BP+DI]	* PTR [BP+DI+imm8]	* PTR [BP+DI+imm16]	BL	BX
100	* PTR [SI]	* PTR [SI+imm8]	* PTR [SI+imm16]	AH	SP
101	* PTR [DI]	* PTR [DI+imm8]	* PTR [DI+imm16]	CH	BP
110	imm	* PTR [BP+imm8]	* PTR [BP+imm16]	DH	SI
111	* PTR [BX]	* PTR [BX+imm8]	* PTR [BX+imm16]	BL	DI

The encoding leaves room for 8 memory addressing modes. We are forced to have `[BX]` for compatibility, but we can choose the other seven. You need to be able to indirect through the base pointer so that you can access your local variables and parameters. And it's expected that you can indirect through `SI` and `DI` since those are the registers used for block memory operations.

That leaves four more addressing modes, and the architects decided to use the four ways of combining `BX` / `BP` with `SI` / `DI`. The `BP+x` addressing modes let you access arrays on the stack, and the `BX+x` addressing modes let you access arrays on the heap, where `SI` and `DI` serve as the index registers.

Now, the architects could have chosen to allow indirection through the other three 16-bit registers, but that would have left room for only one array indexing mode. Giving the instructions to the array indexing modes means that you lose `[AX]`, `[CX]`, and `[DX]`, but that's less of a loss because you can still indirect through `[SI]` and `[DI]` (and `[BP]`, but that's intended to be the frame pointer, not a general-purpose pointer register).

The other choice would be to increase the number of addressing modes by going to a three-byte instruction encoding, thereby picking up eight more bits. But that seems like quite an excessive step, seeing as the original 8080 consisted only of one-byte instructions. (I'm not counting immediate bytes toward encoding counts for the purpose of this comparison.)

It was a game of trade-offs, and the trade-off was to pick up indexed addressing, and give up on supporting indirection through all of the 16-bit registers.

[1] Note that this means that register-to-register operations can be encoded two ways:

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
op						0	w	1	1	reg1			reg2		
op						1	w	1	1	reg2			reg1		

These redundant encodings are used by some assemblers to “fingerprint” their output.

Raymond Chen

Follow