Z80 Undocumented Instructions

By: Jacco J.T. Bot

At your request for documentation on the extra z80 instructions
I've dug into my subdirectories and out came a file which is listed below.
I hope it helps understanding the Z80, most instructions are a direct result of the way it is microcoded.
That is also visible in an instruction like LD A,HL.
There is the 3-byte fast function and the #ED prefix one, which is considerably slower but implemented anyway when the Great ChipMaker decided to expand the instructionset with LD A,BC; LD A,DE;
The way these instructions are decoded implements automatically the LD A,HL instruction for the second time.
Why Zilog is silent about the next bunch of instructions is a mystery to me.
But they also make the Z800, the Z8000, the Z180, the Z280 and the Z380 which must al be compatible on instruction level so maybe the extra z80 instructions aren't implemented in one of these.
Who cares anyway, lets get to the :)

```
Undocumented Z80 instructions

-----------------------------------------------

If an opcode works with the registers HL, H or L then if that
opcode is preceded by #DD (or #FD) it works on IX, IXH or IXL
(or IY, IYH, IYL), with some exceptions.
  The exeptions are instructions like LD H,IXH and LD L,IYH
because it isn't clear from the opcode on which register the
prefix (#FD or #DD) should operate, so from the opcodes point
of view we can read LD H,IXH but also LD IXH, H and obviously
LD IXH,IXH doesn't do us any good, does it?
  Instructions like LD IXH,IXL should work but I didn't test
them yet, my debugger went bananas and is now retired to the
afterlife for computerprograms...
  I also have doubts about the usefullness and correctness of
the OUT (C),F instruction.
  The IN (C), F instructions is only usefull if you test bits
which have the same number as a flag in the F-register because
some older Z80 lock up otherwise.
  An instruction which uses the #ED and the #DD (or #FD) prefix
is also invalid,
so IN IXH,(C) which should translate to DD ED .. is NOT valid.

Anyway, here's a list of valid, but undocumented by Zilog,
instructions...

These instructions are tested on: Z80A, Z80B and Z80H.


  OPCODE       INSTRUCTION        OPCODE        INSTRUCTION
  -----------  -----------------  ------------  ------------
  #DD #24      INC  IXH           #FD #24       INC  IYH
  #DD #25      DEC  IXH           #FD #25       DEC  IYH
  #DD #26 nn   LD   IXH,nn        #FD #26 nn    LD   IYH,nn
  #DD #2C      INC  IXL           #FD #2C       INC  IYL
  #DD #2D      DEC  IXL           #FD #2D       DEC  IYL
  #DD #2E nn   LD   IXL,nn        #FD #2E nn    LD   IYL,nn
  #DD #44      LD   B,IXH         #FD #44       LD   B,IYH
  #DD #45      LD   B,IXL         #FD #45       LD   B,IYL
  #DD #4C      LD   C,IXH         #FD #4C       LD   C,IYH
  #DD #4D      LD   C,IXL         #FD #4D       LD   C,IYL
  #DD #54      LD   D,IXH         #FD #54       LD   D,IYH
  #DD #55      LD   D,IXL         #FD #55       LD   D,IYL
  #DD #5C      LD   E,IXH         #FD #5C       LD   E,IYH
  #DD #5D      LD   E,IXL         #FD #5D       LD   E,IYL
  #DD #60      LD   IXH,B         #FD #60       LD   IYH,B
  #DD #61      LD   IXH,C         #FD #61       LD   IYH,C
  #DD #62      LD   IXH,D         #FD #62       LD   IYH,D
  #DD #63      LD   IXH,E         #FD #63       LD   IYH,E
  #DD #64      LD   IXH,IXH       #FD #64       LD   IYH,IYH
  #DD #65      LD   IXH,IXL       #FD #65       LD   IYH,IYL
  #DD #67      LD   IXH,A         #FD #67       LD   IYH,A
  #DD #68      LD   IXL,B         #FD #68       LD   IYL,B
  #DD #69      LD   IXL,C         #FD #69       LD   IYL,C
  #DD #6A      LD   IXL,D         #FD #6A       LD   IYL,D
  #DD #6B      LD   IXL,E         #FD #6B       LD   IYL,E
  #DD #6C      LD   IXL,IXH       #FD #6C       LD   IYL,IYH
  #DD #6D      LD   IXL,IXL       #FD #6D       LD   IYL,IYL
  #DD #6F      LD   IXL,A         #FD #6F       LD   IYL,A
  #DD #7C      LD   A,IXH         #FD #7C       LD   A,IYH
  #DD #7D      LD   A,IXL         #FD #7D       LD   A,IYL
  #DD #84      ADD  A,IXH         #FD #84       ADD  A,IYH
  #DD #85      ADD  A,IXL         #FD #85       ADD  A,IYL
  #DD #8C      ADC  A,IXH         #FD #8C       ADC  A,IYH
  #DD #8D      ADC  A,IXL         #FD #8D       ADC  A,IYL
  #DD #94      SUB  IXH           #FD #94       SUB  IYH
  #DD #95      SUB  IXL           #FD #95       SUB  IYL
  #DD #9C      SBC  A,IXH         #FD #9C       SBC  A,IYH
  #DD #9D      SBC  A,IXL         #FD #9D       SBC  A,IYL
  #DD #A4      AND  IXH           #FD #A4       AND  IYH
  #DD #A5      AND  IXL           #FD #A5       AND  IYL
  #DD #AC      XOR  IXH           #FD #AC       XOR  IYH
  #DD #AD      XOR  IXL           #FD #AD       XOR  IYL
  #DD #B4      OR   IXH           #FD #B4       OR   IYH
  #DD #B5      OR   IXL           #FD #B5       OR   IYL
  #DD #BC      CP   IXH           #FD #BC       CP   IYH
  #DD #BD      CP   IXL           #FD #BD       CP   IYL

  #DD #CB nn #00   RLC  (IX+nn)   &  LD   B,(IX+nn)        1)
```

```
   #DD #CB nn #01   RLC  (IX+nn)   &  LD   C,(IX+nn)
   #DD #CB nn #02   RLC  (IX+nn)   &  LD   D,(IX+nn)
   #DD #CB nn #03   RLC  (IX+nn)   &  LD   E,(IX+nn)
   #DD #CB nn #04   RLC  (IX+nn)   &  LD   H,(IX+nn)
   #DD #CB nn #05   RLC  (IX+nn)   &  LD   L,(IX+nn)
   #DD #CB nn #06   RLC  (IX+nn)   &  LD   F,(IX+nn)
   #DD #CB nn #07   RLC  (IX+nn)   &  LD   A,(IX+nn)

   #DD #CB nn #08   RRC  (IX+nn)   &  LD   B,(IX+nn) etc.
   #DD #CB nn #10   RL   (IX+nn)   &  LD   B,(IX+nn) etc.
   #DD #CB nn #18   RR   (IX+nn)   &  LD   B,(IX+nn) etc.
   #DD #CB nn #20   SLA  (IX+nn)   &  LD   B,(IX+nn) etc.
   #DD #CB nn #28   SRA  (IX+nn)   &  LD   B,(IX+nn) etc.
   #DD #CB nn #30   SLL  (IX+nn)   &  LD   B,(IX+nn) etc.
   #DD #CB nn #38   SRL  (IX+nn)   &  LD   B,(IX+nn) etc.
   #DD #CB nn #80   RES  0,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #88   RES  1,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #90   RES  2,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #98   RES  3,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #A0   RES  4,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #A8   RES  5,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #B0   RES  6,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #B8   RES  7,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #C0   SET  0,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #C8   SET  1,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #D0   SET  2,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #D8   SET  3,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #E0   SET  4,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #E8   SET  5,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #F0   SET  6,(IX+nn) &  LD   B,(IX+nn) etc.
   #DD #CB nn #F8   SET  7,(IX+nn) &  LD   B,(IX+nn) etc.

   #CB #30          SLL  B                                 2)
   #CB #31          SLL  C
   #CB #32          SLL  D
   #CB #33          SLL  E
   #CB #34          SLL  H
   #CB #35          SLL  L
   #CB #36          SLL  (HL)
   #CB #37          SLL  A

   #ED #70          IN   F,(C)
   #ED #71          OUT  F,(C)                             (?)

   All the above instructions also work with IY, IYH and IYL,
just replace #DD for #FD.

NOTES:
1) These instructions are strange because in fact they are
   two instructions in one.
   Lets examine the folowing line

   #DD #CB nn #00   RLC  (IX+nn)   &  LD   B,(IX+nn)

   The value at address (IX+nn) is left rotated through the
   carry AND is loaded in the B register. I bet that some sophisticated
   programmers may make excellent use of these commands but I reccomend
   against using them because its task it not obvious from the opcode,
   and my assembler doesn't (YET) support them, I've got to find a nice
   short symbol for them, maybe something like RLC(IX+nn)->B.

2) These instructions are equivalent to the SLA command. That is
   perfectly logical:
   An arithmethic rightshift (SRA) has to preserve the 7-th (sign)bit, A
   logical rightshift (SRL) doesn't. The arithmetic leftshift doesn't
   preserve anything, in two's complement the 6-th bit becomes the sign.
   A logical leftshift hasn't got anything to do with signbits so it also
   doesn't preserve any bits. The difference is the way we look at the
   resulting number, a 8-bit byte or a 7-bit integer with 1 sign-bit.

Original list by Richard Spijkers, 1992 ricosoftware.
Translated and Updated by Jacco Bot, 1996 JBSoft
```

MORE Undocumented Z80 Opcodes

Operations with the High and Low Bytes of *IX* and *IY*

- *HX* and *LX* are the high and low bytes of *IX*.
- *HY* and *LY* are the high and low bytes of *IY*.
- To use *HX* and *HY*, prefix opcodes that use *H* with $DD for *HX* and with $FD for *HY*.
- To use *LX* and *LY*, prefix opcodes that use *L* with $DD for *LX* and by $FD for *LY*.

Examples

- To get *LD HX,A* use *.DB $DD* followed by *LD H,A*.

- To get *CP LY* use *.DB $FD* followed by *CP L*.

The following instructions support this feature:

- ADC
- ADD
- AND
- CP
- DEC
- INC
- OR
- SBC
- SUB
- XOR
- LD *rd,nn*
- LD *rd,rs* (except *LD HX,(IX+nn)* and the like)

Shift Left with One Insertion (SL1)

- This instruction operates like *SLA*, except that a 1 is inserted into the low bit of the operand.
- This instruction is inappropriately called *SLL* in other documents. (Probably for symmetry in the naming scheme.)

Instruction	Opcode	r	
SL1 r	CB 30+r	0 B	4 H
SL1 (IX+nn)	DD CB nn 36	1 C	5 L
SL1 (IY+nn)	FD CB nn 36	2 D	6 (HL)
		3 E	7 A

Bit Operations with Autocopy

- These instructions perform their operations and also load the result into a register.
- Instructions using *IY* can be formed by substituting $FD for $DD in the opcode.
- The missing chunk in the opcode space from *40+r* to *78+r* is for BIT commands, which do not support autocopy.

Instruction	Opcode	r
RLC r,(IX+nn)	DD CB nn 00+r	0 B
RRC r,(IX+nn)	DD CB nn 08+r	1 C
RL r,(IX+nn)	DD CB nn 10+r	2 D
RR r,(IX+nn)	DD CB nn 18+r	3 E
SLA r,(IX+nn)	DD CB nn 20+r	4 H
SRA r,(IX+nn)	DD CB nn 28+r	5 L
SL1 r,(IX+nn)	DD CB nn 30+r	6 no autocopy (documented)
SRL r,(IX+nn)	DD CB nn 38+r	7 A

```
RES r,0,(IX+nn)    DD CB nn 80+r
RES r,1,(IX+nn)    DD CB nn 88+r
RES r,2,(IX+nn)    DD CB nn 90+r
RES r,3,(IX+nn)    DD CB nn 98+r
RES r,4,(IX+nn)    DD CB nn A0+r
RES r,5,(IX+nn)    DD CB nn A8+r
RES r,6,(IX+nn)    DD CB nn B0+r
RES r,7,(IX+nn)    DD CB nn B8+r



SET r,0,(IX+nn)    DD CB nn C0+r
SET r,1,(IX+nn)    DD CB nn C8+r
SET r,2,(IX+nn)    DD CB nn D0+r
SET r,3,(IX+nn)    DD CB nn D8+r
SET r,4,(IX+nn)    DD CB nn E0+r
SET r,5,(IX+nn)    DD CB nn E8+r
SET r,6,(IX+nn)    DD CB nn F0+r
SET r,7,(IX+nn)    DD CB nn F8+r
```

Null Input and Output

Instruction	Opcode	Description
IN (C)	ED 70	Input from port *C*, but alter flags only.
OUT (C),0	ED 71	Output 0 to port *C*.

EVEN MORE Undocumented Z80 Opcodes

--------- From NORTHERN BYTES Volume 3 Number 10 (Oct. '82) ----------

UNDOCUMENTED Z-80 OPCODES by Bill Smythe is reprinted from the CHICATRUG NEWS (Chicago TRS-80 Users' Group). NORTHERN BYTES editor's note: This is the most comprehensive article that I have seen to date covering this subject!

If you are willing to set aside your Editor/Assembler for a while and take a direct look at the Z-80 opcodes, certain patterns will become apparent. For example, of the 256 possible opcodes (00 through FF) all but four are in use as one-byte instructions. Some of these stand alone; others are followed by one- or two-byte operands. For example:

```
        13          INC DE          (no operand)
        3E nn       LD A,nn         (1-byte operand)
        10 dd       DJNZ dd         (1-byte operand)
        01 nn mm    LD BC,mmnn      (2-byte operand)
```

The four missing opcodes are CB, DD, ED, and FD. These serve as "escape codes" to alert the Z-80 that a two-byte instruction is coming up. Examples:

```
     CB C0          SET 0,B
     DD 21 nn mm    LD IX,mmnn
     ED 80          LDIR
     FD 2B          DEC IY
```

Like the one-byte instructions, some two-byters are used with operands, while others stand alone.

One of the first observations made by any Z-80 hacker is the parallel among HL, IX, and IY instructions:

```
        21 nn mm    LD HL,mmnn
     DD 21 nn mm    LD IX,mmnn
     FD 21 nn mm    LD IY,mmnn

        77          LD (HL),A
     DD 77 jj       LD (IX+jj),A
     FD 77 jj       LD (IY+jj),A
```

Almost any instruction referring to HL can be changed to the corresponding instruction for IX or IY by prefixing

DD or FD, respectively. When HL appears in parentheses, it becomes (IX+jj) or (IY+jj), where the index jj appears in the opcode as the second byte following the DD or FD. Meanwhile, the two-byte opcodes beginning with CB form an orderly set of shift, rotate, set, reset, and bit-test instructions, while the relatively disorganized ED set performs a variety of useful tasks.

And then there are the combined prefixes DDCB and FDCB. Their relationship to the CB codes are pretty much what you would expect:

```
   CB 46        BIT 0,(HL)
DD CB jj 46     BIT 0,(IX+jj)
FD CB jj 46     BIT 0,(IY+jj)
```

--------- T h e F u n B e g i n s ----------

Everything said so far is pretty much common knowledge, well-documented by Zilog, the Z-80 manufacturer. But what happens when DD or FD is prefixed to an instruction not containing HL?

```
   37           SCF
   C3 nn mm     JP mmnn

DD 37           ?????
FD C3 nn mm     ?????
```

As far as I can determine, the prefixes in these cases have no effect. The two instructions shown here continue to function as SCF and JP mmnn, respectively.

But what if the instruction deals with either H or L, but not HL?

```
   24           INC H
   2E nn        LD L,nn
```

Just as the register pair HL is made up of two registers, H and L, it appears that the 16-bit IX register consists of two 8-bit registers, which I shall call HX and LX. Similarly for IY. The above instructions become:

```
DD 24           INC HX
DD 2E nn        LD LX,nn
FD 24           INC HY
FD 2E nn        LD LY,nn
```

The last instruction, for example, loads the low-order half of the IY register with the value nn while leaving the high-order half untouched.

If, however, an instruction contains both H and (HL), or both L and (HL), then only the (HL) part is affected by the addition of DD or FD:

```
   66           LD H,(HL)
DD 66 jj        LD H,(IX+jj)
FD 66 jj        LD H,(IY+jj)
```

Adding a second DD or FD in front has no additional effect. Apparently, there are no such instructions as LD HX,(IX+jj) or LD HY,(IY+jj).

--------- W h a t ' s Y o u r C B H a n d l e ? ----------

The CB instructions are divided into four groups:

```
CB00 through CB3F:     rotate and shift group
CB40 through CB7F:     BIT testing
CB08 through CBBF:     RESet group
CBC0 through CBFF:     SET group
```

Of these four groups, all seem complete except for the first. The first group is divided into eight subgroups of eight instructions each:

```
CB00-CB07:      RLC (rotate left circular)
CB08-CB0F:      RRC (rotate right circular)
CB10-CB17:      RL  (rotate left through carry)
CB18-CB1F:      RR  (rotate right through carry)
CB20-CB27:      SLA (shift left arithmetic)
CB28-CB2F:      SRA (shift right arithmetic)
CB30-CB37:      ???????
CB38-CB3F:      SRL (shift right logical)
```

The missing CB30 group looks as though it ought to be a shift left logical, whatever that is. Trouble is, SLA (shift left arithmetic) is already pretty logical. So what's left for SLL to do? As might be expected, SLL shifts bits 0 through 6 leftward into bits 1 through 7, while bit 7 goes into the carry flag. But then comes the surprise -- bit 0 is set. Yes, Virginia, regardless of the previous status of any bit, or of any flag, bit 0 is turned on. Thus SLL, in effect, multiplies by 2 and adds 1. As with the other CB instructions, the affected register is B, C, D, E, H, L, (HL), or A depending on

which instruction, CB30 through CB37, is used.

-------- C o m b i n a t i o n s A n d M o r e C o m b i n a t i o n s ----------

Of the combined opcodes, DDCBjjxx and FDCBjjxx, only every eighth one is documented by Zilog:

```
DD CB jj 06     RLC (IX+jj)
DD CB jj 0E     RRC (IX+jj)
     (etc.)
```

One might not expect much from those not on the list, since the corresponding DD-less CB instructions have nothing to do with HL. Not so, however -- a lot of weird stuff is going on here:

```
DD CB jj 00     RLC B,(IX+jj)
DD CB jj 01     RLC C,(IX+jj)
DD CB jj 02     RLC D,(IX+jj)
DD CB jj 03     RLC E,(IX+jj)
DD CB jj 04     RLC H,(IX+jj)
DD CB jj 05     RLC L,(IX+jj)
DD CB jj 06
DD CB jj 07     RLC A,(IX+jj)
```

-- and similarly for RRC, RL, RR, SLA, SRA, SLL, and SRL. But what does that mean, "RLC B,(IX+jj)"? THat's just a name I chose for a peculiar phenomenon in which (IX+jj) is rotated left cirular, then copied into B. In other words, RLC B,(IX+jj) is like RLC (IX+jj) followed by LD B,(IX+jj).

The SET and RESet instructions are even more curious:

```
DD CB jj 80     RES B,0,(IX+jj)
DD CB jj 81     RES C,0,(IX+jj)
DD CB jj 82     RES D,0,(IX+jj)
DD CB jj 83     RES E,0,(IX+jj)
DD CB jj 84     RES H,0,(IX+jj)
DD CB jj 85     RES L,0,(IX+jj)
DD CB jj 86
DD CB jj 87     RES A,0,(IX+jj)
```

I don't know how many Editor/Assemblers could handle 3 operands, even if they recognized undocumented opcodes. I couldn't think of any other way to express what happens -- (IX+jj) first has bit 0 reset, then is copied into the indicated register (e.g. B). The action is equivalent to RES 0,(IX+jj) followed by LD B,(IX+jj). Of course, the same can be done with bits 1 through 7, with SET as well as RESet, and with IY as well as IX.

The BIT instructions are less interesting. Undocumented codes DDCBjj40 through DDCBjj47 turn out to be equivalent to the documented version. No copying into registers B, C, D, etc. is done:

```
DD CB jj 40     BIT 0,(IX+jj)
DD CB jj 41     BIT 0,(IX+jj)
DD CB jj 42     BIT 0,(IX+jj)
DD CB jj 43     BIT 0,(IX+jj)
DD CB jj 44     BIT 0,(IX+jj)
DD CB jj 45     BIT 0,(IX+jj)
DD CB jj 46     BIT 0,(IX+jj)
DD CB jj 47     BIT 0,(IX+jj)
```

-------- L o t s o f R o o m f o r M o r e ----------

My investigations into the ED group yielded little of interest. There were some duplications; for example, the eight instructions ED44, ED4C, ED54, ED5C, ED64, ED6C, ED74, ED7C all turned out to be NEG, even though only the first is documented as such. I also found duplicates for RETN, RETI, IM 0, IM 1, and IM 2. There were also a couple of "expected" duplicates:

```
ED 63 nn mm     LD (mmnn),HL
ED 6B nn mm     LD HL,(mmnn)
```

-- but these instructions already exist, and execute faster, in the non-ED set.

The ED group did provide a couple of lone curiosities:

```
ED 70           IN --,(C)
ED 71           OUT (C),--
```

These appear where you would expect the "missing" IN (HL),(C) and OUT (C),(HL). Nothing happens with (HL), though. IN --,(C) appears to function like IN A,(C), IN B,(C), etc., except that the result does not go anywhere. The flags, however, are set as expected. OUT (C),-- seems to output a zero to the port.

I could not detect any action for the first and fourth quarters of the ED set, ED00-ED3F and EDC0=EDFF. Three-fourths of the third quarter is also "missing", as are two instructions in the second quarter, ED77 and

ED7F. This leaves room for 178 more instructions -- anyone for an upgrade? I'd like to see instructions like LD B,(DE) and CP (DE) and SUB (DE).

--------- E X C E P T I O N S ----------

I said that any HL instruction could be changed to IX or IY by simply prefixing DD or FD. That was a little white lie. The following instructions are not convertible because they begin with ED:

```
     ED 42          SBC HL,BC
     ED 4A          ADC HL,BC
     ED 52          SBC HL,DE
     ED 5A          ADC HL,DE
     ED 62          SBC HL,HL
     ED 6A          ADC HL,HL
     ED 72          SBC HL,SP
     ED 7A          ADC HL,SP
     ED 67          RRD
     ED 6F          RLD
```

In addition, the following instructions cannot be converted even though they are one-byters:

```
        D9          EXX
        EB          EX DE,HL
```

-- and the JP (HL) instruction becomes JP (IX) or JP (IY), not JP (IX+jj) or JP (IY+jj):

```
        E9          JP (HL)
     DD E9          JP (IX)
     FD E9          JP (IY)
```

--------- O n e a t a ' t i m e , P l e a s e ----------

Except for DDCB and FDCB, there is no benefit in combining two or more of the four escape codes. There are three cases:

(1) In the event of multiple DDs and/or FDs, all but the last will be ignored.

(2) EDCB, EDDD, EDED, and EDFD will be ignored entirely since they lie in the inoperative fourth quarter of the ED set.

(3) If either DD or FD precedes ED, the former will be ignored.

--------- A W o r d o f C a u t i o n ----------

To any machine-language programmers whose appetites may have been whetted: Most Z-80 Editor/Assemblers do not recognize undocumented opcodes, so you'll have to enter these codes as DEFBs. More important, it is conceivable that not all Z-80s will respond to these codes in the same way. If you plan to sell your programs, the best advice is not to use undocumented instructions.

--------- S u m m a r y o f U s e f u l U n d o c u m e n t e d Z - 8 0 O p c o d e s ----------

```
DD24     INC HX     DD62  LD HX,D       DD8C  ADC A,HX
DD25     DEC HX     DD63  LD HX,E       DD8D  ADC A,LX
DD26 nn  LD HX,nn   DD64  LD HX,HX      DD94  SUB HX
DD2C     INC LX     DD65  LD HX,LX      DD95  SUB LX
DD2D     DEC LX     DD67  LD HX,A       DD9C  SBC A,HX
DD2E nn  LD LX,nn   DD68  LD LX,B       DD9D  SBC A,LX
DD44     LD B,HX    DD69  LD LX,C       DDA4  AND HX
DD45     LD B,LX    DD6A  LD LX,D       DDA5  AND LX
DD4C     LD C,HX    DD6B  LD LX,E       DDAC  XOR HX
DD4D     LD C,LX    DD6C  LD LX,HX      DDAD  XOR LX
DD54     LD D,HX    DD6D  LD LX,LX      DDB4  OR HX
DD55     LD D,LX    DD6F  LD LX,A       DDB5  OR LX
DD5C     LD E,LX    DD7C  LD A,HX       DDBC  CP HX
DD5D     LD E,LX    DD7D  LD A,LX       DDBD  CP LX
DD60     LD HX,B    DD84  ADD A,HX
DD61     LD HX,C    DD85  ADD A,LX
```

The corresponding instructions for HY and LY may be obtained by using FD in place of DD.

```
        CB30   SLL B            CB34   SLL H
```

```
CB31   SLL C           CB35   SLL L
CB32   SLL D           CB36   SLL (HL)
CB33   SLL E           CB37   SLL A


  DDCBjj00-DDCBjj07   RLC r,(IX+jj)
  DDCBjj00-DDCBjj0F   RRC r,(IX+jj)
  DDCBjj10-DDCBjj17   RL r,(IX+jj)
  DDCBjj18-DDCBJJ1F   RR r,(IX+jj)
  DDCBjj20-DDCBjj27   SLA r,(IX+jj)
  DDCBjj28-DDCBjj2F   SRL r,(IX+jj)
  DDCBjj30-DDCBjj37   SLL r,(IX+jj)
  DDCBjj38-DDCBjj3F   SRL r,(IX+jj)

  DDCBjj80-DDCBjj87   RES r,0,(IX+jj)
  DDCBjj88-DDCBjj8F   RES r,1,(IX+jj)
  DDCBjj90-DDCBjj97   RES r,2,(IX+jj)
  DDCBjj98-DDCBjj9F   RES r,3,(IX+jj)
  DDCBjjA0-DDCBjjA7   RES r,4,(IX+jj)
  DDCBjjA8-DDCBjjAF   RES r,5,(IX+jj)
  DDCBjjB0-DDCBjjB7   RES r,6,(IX+jj)
  DDCBjjB8-DDCBjjBF   RES r,7,(IX+jj)

  DDCBjjC0-DDCBjjC7   SET r,0,(IX+jj)
  DDCBjjC8-DDCBjjCF   SET r,1,(IX+jj)
  DDCBjjD0-DDCBjjD7   SET r,2,(IX+jj)
  DDCBjjD8-DDCBjjDF   SET r,3,(IX+jj)
  DDCBjjE0-DDCBjjE7   SET r,4,(IX+jj)
  DDCBjjE8-DDCBjjEF   SET r,5,(IX+jj)
  DDCBjjF0-DDCBjjF7   SET r,6,(IX+jj)
  DDCBjjF8-DDCBjjFF   SET r,7,(IX+jj)

In the last 3 tables, the corresponding instructions for
(IY+jj) may be obtained by using FD in place of DD.  The
value of r is determined as follows:

      Last digit of opcode:    register r:
             0 or 8                B
             1 or 9                C
             2 or A                D
             3 or B                E
             4 or C                H
             5 or D                L
             6 or E             (blank)
             7 or F                A
```
