The programmer's view of the Z80

This chapter describes how the Z80 appears to the programmer: its architecture and instruction set. Whole books can be written on this (I recommend “Programming the Z80” by Rodnay Zaks), so this chapter will necessarily leave out a lot of details.

The Z80 is an 8-bit processor with a 16-bit address bus. With its 8-bit data bus, it accesses memory one byte at a time, and most of the instructions are one byte long. Registers are a mix of 8-bit registers and 16-bit registers, with some registers being usable as either a pair of 8 bit registers or a single 16-bit register.

The diagram below (from Programming the Z80) is commonly used to show the structure of the Z80. As will be seen in the remainder of this book, there are a number of differences between the diagram and the actual implementation of the chip.

The Z80 has a fairly large register set for microprocessors of its time. It has an 8-bit accumulator (A), 8-bit registers B and C that can be used as the 16-bit BC register, 8-bit registers D and E that can be used as the 16-bit DE register, and the 8-bit registers H and L that can be used as the 16-bit HL register. The flags are stored in an 8-bit flag register F.

One innovation of the Z80 is these 8 registers (A, F, B, C, D, E, H, L) are duplicated into a second set of registers. A single instruction can swap the primary and secondary register sets. This feature provides a fast method of saving registers when switching tasks or handling interrupts.

The Z80 has 16-bit program counter, which is used to step through memory when executing a program. It also has a 16-bit stack pointer, which points to the top of the stack. Because of its 16-bit address space, the Z80 could support 64K (65536 bytes) of memory.

The Z80 also included two 16-bit index registers IX and IY which could be used to efficiently access tables of memory.

Finally, the Z80 has two somewhat unusual 8-bit registers. The interrupt register I holds the top 8 bits of a 16 bit interrupt vector address. This address is used when the processor receives an interrupt. This register will be explained in more detail later.

The 8-bit R register is used to refresh memory. Dynamic RAM needs to be periodically accessed to refresh its contents. The Z80 performs these refreshes automatically, unlike other microprocessors of the time which required extra circuitry. The R register is incremented on each instruction and refreshes memory at that address. Curiously, only 7 bits of the R register are incremented.

The Z80's ALU is 4 bits wide internally, but this is invisible to the programmer. To the programmer, the ALU appears to be a standard 8 bit ALU. (The operation of the ALU will be described later.)

The ALU supports standard arithmetic operations: 8- and 16-bit add, add with carry, subtract, and subtract with borrow. Also the Z80 has 8 and 16- bit increments and decrements. Multiplication and division are not supported by the Z80; implementation of these operations in hardware didn't become common until a few years later. The Z80 supports a few subtraction-based operations: compare (which basically subtracts two values without storing the result), and negation.

It also supports bitwise logic operations: AND, OR, and XOR.

The Z80's ALU supports a variety of shift operations: arithmetic and logical shift left, arithmetic and logical shift right. The Z80 also supports an unusual BCD shift that shifts three 4-bit blocks around.

The Z80 has a variety of 8- and 16-bit load and store operations, which can access memory or registers.

For control flow, the Z80 has absolute and relative jump instructions with a variety of conditions. The Z80 also has conditional and unconditional subroutine call and return instructions.

The Z80 has a variety of input and output instructions that use the Z80's 256 I/O ports.

One innovation of the Z80 is its block operations. A single operation can loop multiple times, using the BC register pair as a counter. For example, the LDIR and LDDR instructions do memory-to-memory copy. The CPIR and CPDR instructions search a block of memory for a value matching the accumulator. The INIR and INDR instructions read a block of data from an I/O device to memory, while the OTIR and OTDR instructions write a block of data from memory to an I/O device. For these instructions, the -I- instruction increments through memory while the -D- instruction decrements through memory. These block instructions provide a powerful mechanism for performing a loop with a single instruction.

Another innovation of the Z80 is the bit instruction. The Z80 has a large number of instructions that allow a single bit in a register or memory to be tested, set, or cleared. Without these instructions, these operations would take multiple instructions.

Addressing modes

The Z80 has multiple addressing modes. They have a big impact on the hardware implementation of the chip as well as the timing of the operations. The following list of addressing modes is based on the documentation.

Register addressing

Many operations specify a register in the opcode and do not require an extra byte to specify the address.One example is INC BC.

Implied addressing

Similar to register addressing is implied addressing, where the opcode implies a particular register. For instance, arithmetic operations use the accumulator. For instance, RLA or CPL.

Register indirect addressing

Some instructions use a 16-bit register pair to specify the memory location.

The HL register is commonly used for this, but other registers can be used as well. e.g. LD (HL), D.

The immediate addressing mode uses the byte following the opcode, which contains a one-byte operand. An example instruction is LD A, n.

The immediate extended addressing mode uses two bytes of data following the opcode, first the low-order byte and then the high-order byte. An example is LD BC, nn.

Modified Page Zero addressing
The single-byte restart instructions transfer execution to one of 8 locations in page 0 of memory. (Page zero is the first 256 bytes of memory and is called page zero, since the top byte of an address in page 0 is 0.) Bits 5, 4, and 3 of the opcode are bits 5, 4, and 3 of the destination address - the other bits are all 0.

Relative addressing
In relative addressing, the byte following the opcode is used to jump to an address relative to the current PC address. The byte is a signed 2's complement value between -128 and +127. This allows execution to jump either forwards or backwards to a nearby address. The advantage of relative addressing is only one data byte is required, rather than the two bytes required to specify an absolute address.

A second advantage of relative addressing is that it allows relocatable code. That is, the code can be moved anywhere in memory and will work unmodified. If code implements loops with absolute (fixed) addresses, these jumps will break if the code moves to a new location.

Extended addressing
Extended addressing (also called absolute addressing on some processors) specifies a full 16-bit address in the two bytes following the opcode (low-order address first, followed by high-order address). Extended addressing is used to jump to a specific address in memory. Extended addressing is also

used to access a specific memory location.

Indexed addressing

In indexed addressing, the opcode is followed by one byte, which is the displacement. The displacement is added to one of the index registers (IX or IY) to obtain the address in memory. As with relative addressing, the displacement is a signed two's complement value between -128 and +127.

One use of indexed addressing is to easily access a location in a table; the index register can point to the start of the table, and the displacement selects which entry in the table to use. Indexed addressing also permits relocatable code.

Bit addressing

The bit set, reset, and test instructions act on one particular bit of the byte, which is specified in the opcode. The byte is specified through register, register indirect, or indexed addressing.

Addressing mode combinations

Many instructions combine multiple addressing modes. For instance, LD DE, nn uses register addressing to select the register pair and immediate extended to provide the data.

The format of the indexed bit instructions is somewhat unusual. These instructions consist of two prefix bytes, the displacement byte, and finally the opcode byte. Other instructions have the displacement or other addressing bytes following the opcode, not before the opcode.

In addition, the timing of indexed bit instructions is unusual. There are two M1 cycles, one for each prefix. The displacement is loaded in a M2 cycle and the opcode is loaded in a M3 cycle. This is very different from all other instructions, which load the opcode in a M1 cycle. This will be discussed in more detail in the timing and instruction decode chapters.

The Z80 has a one byte flag register. There is a separate flag register in the two register sets, so the flags are swapped when the register sets are swapped.

The programmer's view of flags

Six of the 8 bits in the flag are used:
Bit 7 is the sign flag: S
Bit 6 is the zero flag: Z
Bit 4 is the half carry flag: H
Bit 2 is the parity or overflow flag: P or V
Bit 1 is the add/subtract flag: N
Bit 0 is the carry flag: C

Four of the bits (C, P/V, Z, and S) can be used for conditional operations. The H and N flags are used for BCD arithmetic.

The carry flag is set or cleared in arithmetic operations. For addition, the carry flag indicates the addition generated a carry out of the top bit. For subtraction, the carry flag indicates the subtraction required a borrow into the top bit. The carry flag is generally used with unsigned arithmetic, either to indicate overflows or to implement multi-byte arithmetic.

The effect of a shift or rotate instruction on the carry flag depends on the specific operation. This will be discussed in the shift/rotate section.

Logical instructions (AND, OR, and XOR) clear the carry flag.

The carry flag can be set or complemented directly by the Set Carry Flag (SCF) and Complement Carry Flag (CCF) instructions.

The add/subtract flag N is cleared by an addition instruction, and set by a subtraction instruction. The purpose of this flag is to allow the Decimal Adjust Accumulator instruction (DAA) to perform the right correction when addition or subtraction is performed on binary coded decimal values. See the DAA chapter for more information.

The parity/overflow flag has two totally different usages depending on the instruction that affects it. For signed arithmetic operations, the flag indicates overflow of a two's complement value that won't fit in a byte, that is a value outside -128 to +127. See the chapter on overflow for more detail.

For logic and rotate operations, the flag indicates parity. If the result has an odd number of 1 bits (or equivalently an odd number of 0 bits), the parity flag is set to 0. If the result has an even number of 1 bits (or equivalently an even number of 0 bits), the flag is set to 1.

For block operations, the flag has yet another use. P/V flag indicates if is set if the byte counter is nonzero, and cleared if the byte counter is 0.

Finally, for LD A, I and LD A, R instructions, the flag has a different meaning, holding the value of the interrupt control filp-flop IFF2. This is explained in more detail in the interrupt chapter.

The half-carry flag H indicates a carry or borrow between bits 3 and bits 4 during an addition or subtraction. This flag is used by the DAA instruction.

The zero flag Z is set if an arithmetic instruction has a result of 0, and is cleared otherwise. For compare instructions, the Z flag is set if the values are equal. (Compare is similar to subtracting the two values.)

The Z flag also holds the result of a bit test instruction. If the bit is 1, Z is set to 0. If the bit is 0, Z is set to 1.

For INI, IND, OUTI, and OUTD instructions, the Z flag indicates if the B register is decremented to 0.

Finally the sign flag S is set to the top bit of the result of an arithmetic operation. For a two's-complement number, the value is negative if the sign bit is set, and zero or positive if the sign bit is cleared.

The Z and S flags are also controlled by a byte input from an I/O device by an IN r, (C) instruction.

A look at rotate instructions

The Z80 has a variety of shift and rotate instructions. These instructions result in a lot of special-case circuitry to implement the different operations.

The rotate left circular (RLC) instruction shifts the bits of a byte to the left. The top bit (bit 7) is put into the lowest bit (bit 0) as well as the carry.

The rotate right circular (RRC) instruction is similar to RLC, except the bits are shifted to the right. The lowest bit (bit 0) is put into the top bit (bit 7), as well as the carry.

After 8 rotate circular instructions, the byte will be back to its starting position.

The rotate left instruction (RL) shifts the bits left. The contents of the carry flag are put into the lowest bit (bit 0). The top bit (bit 7) is put into the carry flag. In effect, the byte and carry flag form a 9-bit value that is rotated.

The rotate right instruction (RR) is similar, except bits are shifted to the right. The contents of the carry flag are put into bit 7. The contents of bit 0 are put into the carry flag.

The shift left arithmetic (SLA) operation shifts the byte to the left. Bit 7 goes into the carry flag, while 0 goes into bit 0. In effect, SLA is equivalent to adding the byte to itself or multiplying it by 2. This works for both signed and unsigned values.

The shift right arithmetic (SRA) operation shifts the byte to the right. Bit 7 remains in bit 7 as well as being shifted into bit 6. Bit 0 is put into the carry flag. In effect SRA divides the byte by 2, for a signed twos complement value. (Preserving the top bit keeps the sign the same.)

The shift right logical (SRL) operation shifts the byte to the right. 0 is put into bit 7. Bit 0 is put into the carry flag. SRL is equivalent to dividing an unsigned byte by 2.

There is no shift left logical operation defined for the Z80 since shift left arithmetic makes sense for both signed and unsigned values, unlike the right shift.

These shifts can be defined by looking at what gets put into the "empty" bit, and what gets put into the carry flag. The following table may make the structure clearer:

For left shifts/rotates:

instruction	into bit 0	into carry
RLC	bit 7	bit 7
RL	carry	bit 7
SLA	0	bit 7

For right shifts/rotates:

instruction	into bit 7	into carry
RRC	bit 0	bit 0
RR	carry	bit 0
SRA	bit 7	bit 0
SRL	0	bit 0

Finally, the Z80 has complicated instructions to rotate digits left or right. These instructions act on 4-bit binary coded decimal (BCD) digits, which will be explained in a later chapter. These instructions are very different from the other shifts and rotates as they shift 4 bits at a time, not one bit.

For rotate digit left (RLD), the 4 lower bits of the accumulator go into the 4 lower bits of a memory location. The 4 lower bits of memory are shifted left into the 4 higher bits of the memory location. The 4 higher bits of the memory location go into the 4 lower bits of the accumulator.

Rotate digit right (RRD) is similar, but shifts the memory location to the right. The 4 lower bits of the accumulator go into the 4 upper bits of the memory location. The 4 upper bits of the memory location go into the 4 lower bits of the memory location. And the 4 lower bits of the memory location go into the 4 lower bits of the accumulator.

RLD and RRD are intended for use with BCD numbers, for example unpacking them or performing BCD shifts for multiplication/division.

Opcode prefix structure

The Z80's main instructions use a single opcode byte, potentially followed one or two more bytes depending on the addressing mode. These instructions are for the most part based on the 8080 instructions.

To extend the instruction set, the Z80 provides several instruction prefixes. These are additional bytes that can be put before the opcode, providing a larger set of instructions than would fit in one byte.

The ED prefix (which I will also call EXTD) extends the instruction set with some additional instructions including I/O instructions and arithmetic instructions. These instructions consist of the ED byte, an opcode byte, and then any additional data bytes.

The CB prefix (which I will also call BITS) extends the instruction set with the bit set, clear, and test instructions, as well as additional rotates and shifts. These instructions consist of the byte CB followed by an opcode byte.

The DD prefix (which I will also call IX), provides instructions using the IX index register. These instructions consist of the byte DD followed by an opcode byte, followed by any additional data bytes. Many of these instructions use a displacement byte which is added to the IX register to get the memory address.

The FD prefix (which I will also call IY), provides instructions using the IY index register. This is analogous to DD (IX), except IY is used instead of IX.

The DD prefix followed by the CB prefix provides bit or rotate operations (similar to the CB prefix), that use the IX register for indexing (similar to the DD prefix). Thus, the instructions are similar to a combination of the two prefixes. These instructions consist of the DD prefix byte, followed by the

CB prefix byte, followed by the displacement byte, followed by the opcode. Note that the opcode occurs after the displacement, rather than before as with the DD prefix alone.

The FD prefix followed by the CB prefix provides bit or rotate operations using the IY register. These operations are analogous to the DDCB instructions, except using IY instead of IX.

The use of one or two prefix bytes provides a mechanism for the Z80 to greatly increase the size of its instruction set beyond the limits of one-byte opcodes.