The basics

This is a brief description whose purpose is to understand the basics.

The assembler is a very basic language whose purpose is to simplify the creation of a program in machine language. The statements are therefore mainly transcribing CPU statements plus a few others to simplify the development of a program.

Machine statements are called opcodes. The others are usually functions that vary from one assembler to another.

The registers

It is important to understand what a register is before going further.

Register is a specialized internal memory of 8bit. Internally always two of these registers form a pair that is called register pair. During programming the registers can be used individually (8bit) or as a register pair (16bit) depending of what instruction is used. Regardless of the method they point to same physical memory. Changing ie. register D will affect register pair DE and other way around. Please note that when the register pairs are loaded or stored to memory the later member of pair will be always stored to the address pointed and the first member will be stored to address +1. This means that storing ie. DE will become E,D in RAM.

The registers can not be thought as a general memory since not all of the Z80 commands can accept all registers as parameters and this may feel a bit hard at start. Each register pair have anyway their own characteristics of what they are good for. Here we do not try to describe all that you can do with them as there are manuals for that, but just these kind of typical characteristics.

The most important register pairs to know are:

A and F that form register pair AF

A is also called accumulator on some documentations. This register is the most important register since all 8bit calculations are done against this register. This is also why it practically can not be used to store information as you will end up loading and saving this register very often. F is also very important register, but it can not be used directly. It is divided to individual status bits called flags that change each time the CPU executes a command that needs to return a status. The individual bits of this register have different names such as carry flag (C), Zero flag (Z) and so on. Please see flags for more information. AF as a register pair can not be used together to other than storing the accumulator together with the status flags to memory or restoring that.

B and C that form register pair BC

These registers can be used as temporary CPU internal storage. B is usually good choice as a 8bit loop counter. C is usually used when you need to point I/O port. Together register pair BC is usually used for storing 16bit "length" or other 16bit value needed for calculation.

D and E that form register pair DE

As individual registers they don't have any "special gifts" other than being good for CPU internal temporary storage, but when they are used together as a pair they are usually used to point a "destination" in memory. DE is also good choice when 16bit value is needed for calculation.

H and L that form register pair HL

As individual registers these also don't have any "special gifts" but when they are used together they form the most important 16bit pair as typically all 16bit calculations are done against HL. It is often used also as a "source" when pointing memory.

Register pair SP

This is the 16bit pointer that defines where commands like CALL or PUSH store their information. Beginner programmer very rarely needs to manipulate this register directly, but it is good to know. It is also important to know that this points to END of storage, so any new values are stored before the address this is pointing to and the content of SP is decreased by 2. 8bit values that form the register pair are not ever used individually, so they don't even have 8bit names.

If you have read this far about registers, you can skip rest of this chapter and come back later to read more. With these already mentioned registers you can already do practically everything that the MSX can do. Naturally knowing about rest of the registers is also important when you get to speed, but I suggest that you stick to these in your first programs.

Rest of the registers:

Register pair PC

Also known as Program Counter. This always points to where the Z80 is reading program from and although it is very important, programmer very rarely need to think it as a register pair. 8bit values that form the register pair are not ever used individually, so they don't even have 8bit names. PC is also not ever pointed directly, but it is manipulated with commands like JP xxxx (jump) that can be thought as LD PC,xxxx or RET that can be thought as POP PC.

IXh and IXl that form register pair IX

As individual registers these are quite good as temporary storage. They are slower than previously mentioned alternatives, but still faster than using CPU external RAM. Using them as individual registers is anyway officially undocumented. Practically these are safe to use, but you may find an assembler that does not compile the code if you use the names like IXl or IXh. As a pair they behave very similar than HL only slower. The special feature is that many times you may use extra offset value when you are using IX as a pointer.

IYh and IYl that form register pair IY

These registers individually as well as a pair work in all cases exactly like IX does.

I and R that form register pair IR. These are very special registers that rarely have any practical use. I is related to interrupt handling, but has very few use cases in MSX environment. Part of R is used as Z80 internal counter that is sometimes used as a source of random seed. These are newer used as register pair although IR may pop up in some documents.

BC', DE', HL' and AF'. These are called "shadow registers" and are just alternative internal storage places for the BC, DE, HL and AF register pairs. You can swap the storage place with commands EXX and EX AF,AF' but this is all you can do.

Flags

Flags are the things that live in F-register. They allow you to act based on comparisons and do ie. conditional decisions. If you are a beginner programmer, pay special attention to working of CF and ZF since those two are the most important flags out there. Almost everything is usually done by using only those two.

As a general rule everything that involves calculation of something affect also flags. 16bit INC and DEC commands are exception to this rule. Generally flags are NOT affected by jump instructions, CPU or interrupt control instructions or load instructions. Load from I or R register is exception to this second rule.

Here is list of flags in F-register:

bit 7, SF, Sign flag. This is copy of the results most significant bit. If the bit is set (= 1 = "M") "Minus" the 2-complement value is negative other ways the result is positive (= 0 = "P") "Plus". Note that this flag can be used only with conditional JP-instruction.

bit 6, ZF, Zero flag. If the result of mathematical operation is zero the bit is set (= 1 = "Z") other ways the bit is reset (= 0 = "NZ") "Not zero"

Bit 5, YF, copy of the results 5th bit.

Bit 4, HF, "Half-carry" from bit 3 to 4. Z80 uses this internally for BCD correction.

Bit 3, XF, copy of the results 3rd bit.

Bit 2, PF/VF, Parity flag. The flag is set to a specific state depending on the operation being performed. With logical operations and rotate instructions it indicates whether the resulting parity is even (PF). The number of 1 bits in a byte are counted. If the total is Odd, ODD parity is flagged (i.e., P = 0). If the total is even, even parity is flagged (i.e., P = 1). For arithmetic operations, this flag (VF) indicates a 2-compliment signed overflow condition when the result is greater than maximum value or is less than minimum value. To use this flag you use the jump/call/ret commands with conditionals, for example: "JP PE, nnnn" and "JP PO, nnnn". Note that the different arithmetic commands may affect the VF differently with 8-bit operands vs 16-bit operands. The flag is used for a few other things as well, please refer to the Z80 manual for further details.

Bit 1, NF, this bit is used by Z80 internally to indicate if last operation was addition or subtraction (needed for BCD correction)

Bit 0, CF, Carry flag = Overflow bit. This bit is the most high bit that did not fit to the result. In addition to large calculations and bit transfers it is mostly used in comparisons to see if the subtraction result was smaller or greater than zero. If the carry flag was set (= 1 = "C") the result did overflow. Other ways the flag is reset (= 0 = "NC") "No carry". Please note that 8bit INC/DEC commands do not update this flag.

In addition to these flags there are two more Z80 internal flags that user rarely needs to know about, but they are mentioned for completeness sake:

IFF1 This is used by Z80 internally to indicate if there is pending interrupt

IFF2 this is backup copy of IFF1 so that the state can be restored.

The Stack

The stack is a memory location specified by the SP register used to store the back addresses of subroutines called by CALL or RST. CALL or RST statement stores to the stack the address behind it, and then the RET statement retrieves that address to know where to resume the execution of the program after the routine execution.

You can also store or retrieve an address into the stack with PUSH and POP but be careful not to disturb the execution of program.

Be careful too not to overwrite data or a routine by over-storing addresses.

Statements for opcodes

There are several types of statement that we can classify into nine groups.

Note: The use of registers or value are not freely usable with all statement. Refer to a documentation to know the constraints.

Generally in the Z80 assembler the lines are formatted like:

[<label>[:]]<space/tab><statement>[[<space/tab><argument1>],<argument2>]

first parameter of statement is the destination where value will be stored after execution. In case of calculation it is also the first value of the calculation. The second parameter is the source or 2nd parameter in calculation formula. In some cases the first parameter should be left missing (AND, OR, XOR, NEG, CPL, SUB, CP). In these cases the first parameter is always "A" logically although it is not written out. In fast versions of the bit shift operations register A may be written as part of the command it self. (ie. RLC A = slow, RLCA = fast)

Parentheses "()" are used to indicate that the 16bit argument or register pair inside is not used directly, but it is used as a memory pointer to retrieve / store the 8 or 16bit value. Exception to this rule is JP-command where the parentheses should be ignored in logical sense. ie. "JP (HL)" logically works as "JP HL"

Data transfers

- LD <argument1>,(<argument2>)
- LD (<argument1>),<argument2>
- LD <argument1>,<argument2>

o LD "load" is the statement to transfer a value from one place to another.

o <argument1> is the destination. (It can be a double/single register or a 8/16 bit value)

o <argument2> is the source. (It can be a double/single register or a 8/16 bit value).

The parentheses indicate that the argument is 16bit memory pointer where the value is fetched/put (Instead of direct value or registerpair)

- LDD

o Transfers an 8 bit value from source (HL) to Destination (DE) with decrementing of the addresses and counter (BC)

- LDDR

o "LDD with Repeat" Transfers a memory block by executing LDD until counter reaches 0

- LDI

o Transfers an 8 bit value from source (HL) to Destination (DE) with incrementing of the addresses and decrementing of counter (BC)

- LDIR

- "LDI with Repeat" Transfers a memory block by executing LDI until counter reaches 0

Register Exchanges

- EX <operand1>,<operand2>
- EX (<operand1>),<operand2>
- Exchanges the values of the two operands.
- EXX
- Exchanges the values of DE, HL and BC with the auxiliary registers.

Arithmetic operations

- ADC <operand>
- Add the Accumulator with operand and the bit Carry. (A = A + <operand> + Carry)
- ADD <operand>
- Add the Accumulator with operand. (A = A + <operand>)
- DAA
- "Decimal Adjust Accumulator" Adjusts each quartet of the Accumulator to decimal.
- DEC <operand>
- Decrement the contents of register. (<operand> = <operand> - 1)

If the operand is a double register, the operation does not modify the flags (register F).

- DEC (<operand>)
- Decrement the contents of memory specified by the register.
- INC <operand>
- Increment the contents of register. (<operand> = <operand> + 1)

If the operand is a double register, the operation does not modify the flags (register F).

- INC (<operand>)
- Increment the contents of memory specified by the register.
- NEG
- "Negative". Subtract zero with the contents of the Accumulator then place the result in the Accumulator. (A = -A)
- SBC
- Subtract the Accumulator with operand and the bit Carry. (A = A - <operand> - Carry
- SUB <operand>
- Subtract the Accumulator with operand. (A = A - <operand>)

- SUB (<operand>)
  - Subtract the contents of memory specified by the register.

Logical operations

- AND <operand>
  - Logical operation: A = A and <operand>.
- AND (<operand>)
  - Logical operation: A = A and (<operand>).
- CPL
  - "Complement" Reverses all bits of A (= NOT)
- OR <operand>
  - Logical operation: A = A or <operand>.
- OR (<operand>)
  - Logical operation: A = A or (<operand>).
- XOR <operand>
  - Logical operation: A = A xor <operand>.
- XOR (<operand>)
  - Logical operation: A = A xor (<operand>).

Handling bit(s)

- CCF
  - Reverses the status of the Carry bit (in Register F).
- RES <bit number>,<operand>
- RES <bit number>,(<operand>)
  - Reset (=make 0) a bit of the operand. The parentheses indicate that it is the content.
- RL <operand>
  - "Rotate Left" Carry and operand bits are shifted to left with rotation.
- RLA
  - Fast version of "RL A". Carry and Accumulator bits are shifted to left with rotation.
- RR <operand>
  - "Rotate Right" Carry and operand bits are shifted to right with rotation.
- RRA
  - Fast version of "RR A". Carry and Accumulator bits are shifted to right with rotation.

- RLC <operand>

o "Rotate Left with Carry" Operand bits are shifted to left with rotation on itself. Carry takes the status of most significant bit from the operand.

- RLCA

o Fast version of "RLC A". Accumulator bits are shifted to left with rotation on itself. Carry takes the status of most significant bit from the Accumulator.

- RRC <operand>

o "Rotate Right with Carry" Operand bits are shifted to right with rotation on itself. Carry takes the status of least significant bit from the operand.

- RRCA

o Fast version of "RRC A". Accumulator bits are shifted to right with rotation on itself. Carry takes the status of least significant bit from the Accumulator.

- SCF

o "Set Carry Flag" Sets the Carry bit (in Register F).

- SET <bit number>,<operand>

- SET <bit number>,(<operand>)

o Sets (= Make 1) a bit of the operand. The parentheses indicate that it is the content.

- SLA <operand>

o Operand bits are shifted to left. Least significant bit is reseted. Carry takes the status of most significant bit from the operand.

- RLD

o 4 least significant bits of Accumulator and all of (HL) will be shifted 4 bits to the left with rotation. 4 most significant bits of the Accumulator will remain as is.

- SRA <operand>

o Operand bits are shifted to right. Most significant bit keep its status. Carry takes the status of least significant bit from the operand.

- SRL

o Carry and Accumulator bits are shifted to right. Most significant bit is reseted. Carry takes the status of least significant bit from the operand.

- RRD

o 4 least significant bits of Accumulator and all of (HL) will be shifted 4 bits to the right with rotation. 4 most significant bits of the Accumulator will remain as is.

Data comparisons and bit(s) test

- BIT <bit number>,<operand>

- BIT <bit number>,(<operand>)

o Tests a bit of the operand is reset or not. The parentheses indicate that it is the content of memory pointed by the address of specified register.

(Z flag is set if the bit is reset.)

- CP <argument>

o "Compare" Subtracts the argument with A, but does not store result. (Z-flag is set if same, C-flag is set if the argument is greater than A)

- CPD

o Comparison of a byte from memory with A. Then registers B (counter) and HL (pointer) are decremented.

- CPDR

o Repetitive comparison of a byte from memory with A. Register B (counter) and HL (pointer) are decremented until A=(HL) or B is zero.

- CPI

o Comparison of a byte from memory with A. Then register B (counter) is decremented and HL (pointer) is incremented.

- CPIR

o Repetitive comparison of a byte from memory with A. Register B (counter) is decremented and HL (pointer) is incremented until A=(HL) or B is zero.

Jumps, stack, subroutines

- CALL <address>

o Pushes return address to stack and jumps to the specified address.

- CALL <condition>,<address>

o Pushes return address to stack and jumps to the specified address in case of named condition.

- DJNZ <distance>

o "Dec b and Jump if Not Zero". Usually used for "loop" (backward) or "case" (forward) type of execution.

- JP <address>

o Jump to the specified address.

- JP <condition>,<address>

o Jump to the address in case of named condition.

- JP (<registerpair>)

o Jump to the address in register. (Please note the logical error in syntax!)

- JR <distance>

o Relative jump. Can jump to address -126 to +129 bytes further than current address.

- JR <condition>,<distance>

o Relative jump in case of named condition.

- PUSH <registerpair>

o Stores the registerpair to top of stack and decreases SP by 2..

- POP <registerpair>

o Loads the registerpair from the top of the stack increase SP by 2.

- RET

o Resume execution after the last CALL executed. (="POP PC")

- RST <address>

o "Reset" or "Software interrupt". Works like CALL, but is smaller and faster and works only to 1 of 8 fixed addresses that are 0h, 8h, 10h, 18h, 20h, 28h, 30h or 38h.

Data Input/Output

Avoid using his statements on MSX. Only some ports are standard and you have to pay attention to timings. Use the BIOS corresponding to the hardware to use as much as possible through the jump table.

- IN A,(<port>)

o Read the specified port.

- IND

o Reading of the specified port by C and storage of the value into memory. Register B (counter) and HL (pointer) are decremented.

- INDR

o Repetitive reading of the specified port by C with storage of the the data into memory. Register B (counter) and HL (pointer) are decremented until B is reseted.

- INI

o Reading of the specified port by C and storage of the value into memory. Register B (counter) decremented and HL (pointer) is incremented.

- INIR

o Repetitive reading of the specified port by C with storage of the the data into memory. Register B (counter) decremented and HL (pointer) is incremented until B is reseted.

- OUT (<port>),A

o Send a byte to the specified port.

- OUTD

- Send a byte from memory to the specified port by C. Then register B (counter) and HL (pointer) are decremented.

- OTDR
  - Send a series of bytes to the specified port by C. Register B (counter) and HL (pointer) are decremented until B is reseted.

- OUTI
  - Send a byte from memory to the specified port by C. Then register B (counter) is decremented and HL (pointer) is incremented.

- OTIR
  - Send a series of bytes to the specified port by C. Register B (counter) is decremented and HL (pointer) is incremented until B is reseted.

Interruptions and waiting times

The CPU receives interrupt signals from other devices. The statements below allow you to manage them. For example the VDP sends a signal at the end of each screen display (50 or 60 times per second). At each interruption the CPU executes a RST 38.

- DI
  - "Disable Interrupts" (Interrupts are disabled immediately after DI execution.)
- EI
  - "Enable Interrupts" (Interrupts are re-established after the next statement is executed.)
- HALT
  - Waits up to next interrupt.
- IM0
  - Interrupt mode 0.
- IM1
  - Interrupt mode 1. Mode used by MSXs.
- IM2
  - Interrupt mode 2.
- RETI
  - Resume execution after an interrupt routine in mode 2.
- RETN
  - Resume execution after a non maskable interrupt routine. (not used on MSX)
- NOP
  - Wait 4 T states.

Labels

Labels is a symbolic name (symbol) for a routine or value. A label defines constant address or value. Set a label consists to place a name of your choice at the beginning of a line, followed by a colon. The colon is optional in many assemblers.

The symbol's value will be the address of the label's position when assembled, unless followed by an EQU statement, in which case the value will be the one specified as the EQU's argument.

The ORG statement specifies the start address of the code follows it, and the address of any position in the code is derived from this.

Notes:

- Many assemblers are case sensitive for labels.
- Assemblers for old machine support short labels only to preserve the memory. This is the case for GEN80 by default but you can add the statement *S 14 at the beginning of your program to support labels up to 14 characters. Of course you can replace 14 with another value when necessary.
- A error occurs with a few assemblers when a colon is used with the EQU statement.
- A error occurs with a few assemblers when a label following by colon is used with a statement on the same line.

Program format

An assembler program is a plain text file that must be assembled to obtain an executable file.

We can only put one label followed by colon and one statement per line.

A label followed by colon and a statement must be separated by at least one space or tab.

Statement and its parameters must be separated by a space or tab.

The remarks should be placed at the end of the line behind a semicolon separated by at least one space or tab from the preceding statement or label.

Some assemblers do not support empty lines nor or statements without spaces or tabs in front.

A few assemblers do not support empty lines. Just put a semicolon in this case.

Most assemblers need space or tabs in front when there is no label.

Values

Usually values can be specified in binary, octal, decimal, hexadecimal or with a character as follows.

1010000b = %1010000 = 120o = o120 = 80 = 50h = $50 = 'P' = "P"

If you use the notation with the H for the hexadecimal, it is necessary to put a digit at front. So if the value starts with a letter (from A to F), put a zero in front otherwise it will be taken for a label.

11111111b = %11111111 = 377o = o377 = 255 = $FF = 0FFh

Some assemblers support value format like in language C.

0b11111111 = 0o377 = 255 = 0xFF

Main expressions

Expressions are math operations that are calculated during compile time and are not compiled in to machine language. Because of this registers or memory content can't be part of expressions, but labels, values and constants can.

Expressions that we can use are +, -, *, / for addition, subtraction, multiplication and division. Logical expressions are &, |, ^ for And, Or and Xor operations.

Data

Following statements can be used to insert data.

defb or db for entering 8 bit values or characters.

defw or dw for entering 16 bit values.

defs or ds to reserve an area. The first value is the length, the second is the value to fill the area. By default the area will be filled with zeros or 255.

Note: Each value must be separated by a comma. The characters must be a string in quotes or apostrophe.

That's all?

To begin, yes. Once you have assimilated the bases, you have to find a library of routines to complete the gaps of BIOS.

Assemblers also have statements for including external routines or making macros but you will be able to discover them for yourself once you understand the basics of this language.

Now, the main thing for you is to find a suitable assembler for your needs. I advise you a cross assembler. That is an assembler able to assemble machine code for Z80 with a modern PC. This will simplify the task.

Z80 Microprocessors User Manual by Zilog and the wiki will be a very valuable help. Good luke!